GlobalKnowledgeNetwork

CONNECTING PEOPLE TO KNOWLEDGE

UNIX Utilities and Commands Course Guide

Order Number: EY-K721E-SG-0002

BUSINESS PARTNER
digital
Authorized Training Provider

April 1994

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.

Reproduction or duplication of this courseware in any form, in whole or in part, is prohibited without the prior written permission of Digital Equipment Corporation.

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 1993. All Rights Reserved.

The following are trademarks of Digital Equipment Corporation:

Alpha AXP, Bookreader, CDA, DEC, DECmcc, DECnet, DECserver, DECstation, DECsystem, DECterm, DECwindows, DNA, Digital, DMS, LA, LAT, LinkWorks, LN03, MicroVAX, OpenVMS, PrintServer, Q–bus, RSTS, RSX–11, RZ, TK, TMSCP, TOPS–10, ULTRIX, VAX, VAX C, VAXcluster, VAXELN, VAXstation, VMS, VT, VT100, VT300, VT320, XUI and the DIGITAL logo.

Apollo is a registered trademark of Apollo Computer, Inc. AT&T is a registered trademark of the American Telephone & Telegraph Company. Hewlett-Packard is a registered trademark of Hewlett-Packard Company. IBM is a registered trademark of the International Business Machines Corporation. MS–DOS is a registered trademark of Microsoft Corporation. NFS is a registered trademark of Sun Microsystems, Inc. OSF, OSF/1, and OSF/Motif and Motif are registered trademarks of Open Software Foundation, Inc. PostScript is a registered trademark of Adobe Systems, Inc. SPARCstation is a trademark of Sun Microsystems, Inc. UNIX is a registered trademark licensed exclusively by X/Open Co., Ltd.

This document was prepared using VAX DOCUMENT Version 2.1.

Contents

About This Course xix

1 Introduction to UNIX Utilities and Commands

UNIX Utilities and Commands 1–2
Introduction 1–2
Objectives 1–2
Resources 1–2
History of UNIX 1–3
Creators of UNIX 1–3
UNIX Evolution 1–3
Berkeley (BSD) UNIX 1–3
AT&T UNIX and System V 1–3
Open Software Foundation 1–4
Digital OSF/1 1–4
Digital OSF/1 Features 1–4
DEC OSF/1 Operating System 1–6
Overview 1–6
Description 1–6
The Windowing Interface 1–7
Overview 1–7
OSF/Motif 1–7
Logging In and Out of the System 1–8
Overview 1–8
Using a VT Terminal 1–8
Using a Workstation 1–9
Starting a DECterm Window 1–10
Entering a Command 1–11
Logging Out of a VT Terminal 1–12
Logging Out of a Workstation 1–12
Learning Activity: Logging In and Out 1–13
Learning Activity Solution: Logging In and Out 1–13
Passwords 1–14
Overview 1–14
Choosing Passwords 1–14
Password Guidelines 1–14
Changing Your Password 1–15
Confirming Your Password 1–15
Learning Activity: Changing Passwords 1–16
Learning Activity Solution: Changing Passwords 1–16
DEC OSF/1 Documentation 1–17
Overview 1–17
Using Hardcopy Documentation 1–17
Using Online Documentation 1–18

Organization of man Reference Pages 1–18
Using the man Command 1–18
Using the man passwd Command 1–19
Using apropos to Learn About Related Commands 1–19
Using the help Command 1–20
Learning Activity: Using Documentation 1–20
Learning Activity Solution: Using Documentation 1–20
Summary 1–22
History of UNIX 1–22
DEC OSF/1 Operating System 1–22
The Windowing Interface 1–22
Logging In and Out of the System 1–22
Passwords 1–22
DEC OSF/1 Documentation 1–23

2 Setting Up Your Terminal

Getting Started 2–2
Introduction 2–2
Objectives 2–2
Resources 2–2
Terminal Setup 2–3
Overview 2–3
Displaying Keyboard Settings 2–3
Using Special Keyboard Settings 2–3
Using Control Keys 2–4
Changing Keyboard Settings 2–4
Undefining a Key Definition 2–5
Saving Keyboard Settings Between Sessions 2–5
Learning Activity: Terminal Setup 2–6
Learning Activity Solution: Terminal Setup 2–6
Workstation Setup 2–8
Overview 2–8
Creating a DECterm Window 2–8
DECterm Window 2–9
DECterm Window Menus 2–10
Using the DECterm Options Menu 2–11
DECterm General Options Menu 2–12
DECterm Keyboard Options Menu 2–13
Automatically Starting Applications 2–14
Customizing Your Session Manager 2–15
Learning Activity: Workstation Setup 2–16
Summary 2–18
Terminal Setup 2–18
Workstation Setup 2–18

3 Invoking Motif Applications

Invoking Motif Applications 3–2
Introduction 3–2
Objectives 3–2
Resources 3–2
Desktop Applications 3–3
Overview 3–3

Using man to Learn About Desktop Applications 3–4
Using Bookreader 3–4
Starting Bookreader 3–5
Using CDA Viewer 3–6
Starting CDA Viewer 3–6
Learning Activity: Desktop Applications 3–6
The Calendar Application 3–7
Using the Calendar Application 3–7
Starting the Calendar Application 3–7
About the Default Calendar 3–7
Selecting the Day 3–8
Creating a Calendar Entry 3–9
Using the calendar Entry Editor 3–10
Customizing the Calendar Display 3–11
Customizing the Day View Defaults 3–12
Calendar Day View 3–13
Learning Activity: Using the Calendar Utility 3–13
Summary 3–14
Desktop Applications 3–14
The Motif Calendar Application 3–14

4 File System Concepts

File System Concepts 4–2
Introduction 4–2
Objectives 4–2
Resources 4–2
File Types 4–3
Overview 4–3
Ordinary Disk Files 4–3
Hidden Files 4–3
Directories 4–3
Special Files 4–3
Learning Activity: File Types 4–3
Learning Activity Solution: File Types 4–4
Determining File Type 4–4
Learning Activity: Using the file Command 4–5
Learning Activity Solution: Using the file Command 4–5
Naming Files and Directories 4–6
Overview 4–6
Naming Conventions 4–6
Acceptable File Names 4–6
Naming Guidelines 4–6
UNIX Directory Structure 4–8
Overview 4–8
Hierarchical File System 4–8
Summary 4–10
File Types 4–10
Naming Files and Directories 4–10
UNIX Directory Structure 4–10

5 Directory and File Access

Introduction 5–1
Objectives 5–1
Resources 5–1
Changing Directories 5–2
Overview 5–2
Home Directory 5–2
Working Directory 5–2
Home Versus Working Directory 5–2
Pathnames 5–3
Full Pathnames 5–3
Relative Pathnames 5–4
Full Versus Relative Pathnames 5–4
The cd Command 5–4
Using Special Pointer Files . and 5–5
Learning Activity: Changing Directories 5–5
Learning Activity Solution: Changing Directories 5–6
Listing Files 5–7
Overview 5–7
The ls Command 5–7
Using the ls Command 5–7
Using the ls Command with Options 5–8
Using the -a Option 5–9
Using the -F and -a Options 5–9
Using the -l Option 5–9
Field Descriptions of Long List 5–10
Learning Activity: Listing Files 5–10
Learning Activity Solution: Listing Files 5–11
Finding Files 5–13
Overview 5–13
Locating Files on the System 5–13
whereis Command Options 5–13
Using the whereis Command 5–14
The find Command 5–14
Using the find Command 5–14
The find Command Expressions 5–15
Learning Activity: Finding Files 5–16
Learning Activity Solution: Finding Files 5–17
Viewing Files 5–18
Overview 5–18
Viewing Entire Files 5–18
Viewing Portions of Files 5–18
The head Command 5–19
The tail Command 5–19
Viewing a Screenful of a File 5–19
Using the more Command 5–20
File Display Examples 5–21
Learning Activity: Viewing Files 5–21
Learning Activity Solution: Viewing Files 5–21
Summary 5–22
Changing Directories 5–22
Listing Files 5–22
Finding Files 5–23
Viewing Files 5–23

6 File Management

Managing Files 6–2
Introduction 6–2
Objectives 6–2
Resources 6–2
Creating Files 6–3
Overview 6–3
Using the touch Command 6–3
touch Command Options 6–3
Using the cat Command 6–4
Using cat to Create Files 6–4
Using cat to Add Text to Files 6–5
Using cat to Append File Contents to Files 6–6
Learning Activity: Creating Files 6–6
Learning Activity Solution: Creating Files 6–7
Creating Directories 6–8
Overview 6–8
The mkdir Command 6–8
Using mkdir to Create Directories 6–9
Interpreting Results of mkdir 6–10
Learning Activity: Creating Directories 6–10
Learning Activity Solution: Creating Directories 6–11
Changing File and Group Permissions 6–12
Overview 6–12
About File Permissions 6–12
Reading the User, Group, Other (ugo) Permissions 6–12
Interpreting File Permission Fields 6–13
Using chmod to Change File Permissions 6–14
Using chmod with Letters and Symbols 6–15
Using chmod with Octal Numbers 6–15
Interpreting Octal Permission Combinations 6–16
Learning Activity: Changing File Permissions 6–17
Learning Activity Solution: Changing File Permissions 6–18
Changing Group Permissions 6–19
Using the chgrp Command 6–19
Understanding the chgrp Command 6–20
Creating a Private Project Directory 6–20
Renaming Files 6–22
Overview 6–22
Using the mv Command 6–22
Moving Files Safely 6–22
Learning Activity: Moving and Renaming Files 6–23
Learning Activity Solution: Moving and Renaming Files 6–23
Copying Files 6–24
Overview 6–24
Using the cp Command 6–24
Copying Files Safely 6–24
Copying Directories and Subdirectories 6–24
Learning Activity: Copying Files 6–25
Learning Activity Solution: Copying Files 6–25
Removing Files and Directories 6–27
Overview 6–27
Using the rm Command 6–27
Using rm with Multiple Links 6–27

Removing Files Safely 6–28
Removing Files and Directories Together 6–28
Using the rmdir Command 6–28
Learning Activity: Removing Files and Directories 6–29
Learning Activity Solution: Removing Files and Directories 6–29
Linking Files 6–31
Overview 6–31
Using Inodes 6–31
Types of Links 6–31
Using Hard Links 6–32
Generating Hard Links 6–32
Relationship Between Hard Link and File 6–33
Symbolic (Soft) Links 6–33
Relationship Between Soft Links and Files 6–34
Learning Activity: Linking Files 6–35
Learning Activity Solution: Linking Files 6–36
Summary 6–40
Creating Files 6–40
Creating Directories 6–40
Changing File and Group Permissions 6–40
Renaming Files 6–41
Copying Files 6–41
Removing Files and Directories 6–42
Linking Files 6–42

7 Shell Overview and Korn Shell Introduction

Shell Overview 7–2
Introduction 7–2
Objectives 7–2
Resources 7–2
Shell Introduction 7–3
Overview 7–3
Purpose of Shells 7–3
DEC OSF/1 Available Shells 7–3
Bourne Shell Features 7–3
C Shell Features 7–3
Korn Shell Features 7–4
DEC OSF/1 Shell Features Summary 7–5
Determining the Login Shell 7–5
Temporarily Changing Your Shell 7–6
Changing Your Login Shell 7–6
Learning Activity: Changing Your Shell 7–7
Learning Activity Solution: Changing Your Shell 7–7
Wildcard Operators 7–9
Overview 7–9
Matching File Names 7–9
The * Operator 7–9
The ? Operator 7–10
Using Wildcard Operators in Commands 7–10
Escaping Wildcard Operators 7–10
Learning Activity: Using Wildcard Operators 7–10
Learning Activity Solution: Using Wildcard Operators 7–10
Korn Shell Environment 7–12

Overview 7–12
Parent and Child Processes 7–12
The Environment 7–12
Environment Characteristics Passed from the Parent Process 7–12
Environment Characteristics NOT Passed from the Parent Process 7–12
Korn Shell Variables 7–13
Reserved Variables 7–13
Setting Korn Shell Variables 7–15
Using set to Display Korn Shell Variables 7–16
Using print to Display Korn Shell Variables 7–17
Unsetting Korn Shell Variables 7–17
Declaring Variable Attributes 7–17
Using typeset with Korn Shell Variables 7–17
Using Attributes with typeset 7–18
Using typeset to Display Variables 7–18
Using typeset to Define Variables 7–19
Korn Shell Options 7–21
Setting Korn Shell Options 7–21
Displaying Korn Shell Options 7–21
Learning Activity: Displaying and Setting Korn Shell Variables and Options 7–21
Learning Activity Solution: Displaying and Setting Korn Shell Variables and Options 7–23
Korn Shell Startup Files 7–26
Overview 7–26
Korn Shell Startup Files 7–26
Korn Shell .profile Files 7–26
Korn Shell .profile File 7–27
Korn Shell Environment Script File 7–27
Learning Activity: Startup Files 7–28
Summary 7–29
Shell Introduction 7–29
Wildcard Operators 7–29
The Korn Shell Environment 7–29
Korn Shell Startup Files 7–30

8 Introduction to UNIX Editors and the emacs Editor

UNIX Editors and emacs 8–2
Introduction to the UNIX Editors 8–3
The emacs Editor 8–5
Using emacs 8–7
emacs Help 8–12
Basic Editing 8–16
Search and Replace Operations 8–30
Using Buffers and Windows 8–35
Correcting Mistakes 8–44
Summary 8–50

9 The vi Editor

Using the vi Editor 9–2
Editing with vi 9–3
Moving the Cursor 9–8
Finding, Inserting, and Changing Text 9–14
Copying, Moving, and Saving Text 9–19
Summary 9–28

10 Redirecting, Filtering, and Piping

Redirecting, Filtering, and Piping 10–2
Redirecting Input and Output 10–3
Filters 10–7
Sorting 10–10
The grep Command 10–21
Pipelines 10–31
The awk Utility 10–34
Summary 10–41

11 Communicating with Other Users

Communicating with Other Users 11–2
Obtaining User Information 11–3
Introduction to Mail 11–7
Customizing Mail 11–22
Using Mail with DEC OSF/1 Motif 11–28
The write Utility 11–31
Summary 11–36

12 Recalling and Editing Korn Shell Commands

Recalling and Editing Korn Shell Commands 12–2
Korn Shell Command History 12–3
Korn Shell Command-Line Editing 12–12
Editing Korn Shell Commands in a Temporary Buffer 12–26
Summary 12–30

13 Controlling Processes

Korn Shell Process Control 13–2
The Process Concept 13–3
Obtaining Process Status 13–11
Job Control Commands 13–17
Scheduling Jobs to Run at Appropriate Times 13–26
Summary 13–33

14 Introduction to Korn Shell Scripts

Korn Shell Scripts 14–2
Korn Shell Scripts 14–3
Korn Shell Variables 14–7
Flow Control and Loops 14–17
Summary 14–29

15 Using TCP/IP Networks

TCP/IP Networks 15–2
Accessing Remote Systems 15–3
Invoking Remote Motif Applications 15–11
Summary 15–15

16 Printing

Printing 16–2
Printing Overview 16–3
Submitting Print Requests 16–6
Controlling Print Queues 16–9
Summary 16–16

17 Using Tapes and Disks

Tapes and Disks 17–2
Identifying Devices 17–3
The Tape Archive Utility (tar) 17–5
Other Useful Commands 17–11
Using Disks 17–20
Summary 17–24

18 Test

Introduction 18–2
Questions 18–3
Answers 18–16

A stty Command Arguments

Setting Terminal Characteristics A–2

B Korn Shell Command-Line Editing Commands for emacs and vi

emacs and vi Command-Line Editing Commands B–2

C Mail Options Using the set Command

Mail Options Using the set Command C–2

Glossary

Examples

1 .emacs Initialization File xxxi
1–1 Using the date Command with Options 1–12
1–2 Using the man passwd Command 1–19
1–3 Output of the help Command 1–20
2–1 Using the stty everything Command 2–3
4–1 Using the file Command 4–5
5–1 Using .. with the cd Command 5–5

5–2 Using the ls Command 5–8
5–3 Using the whereis Command 5–14
5–4 Viewing a File Using the cat Command 5–18
5–5 Viewing Part of File Using head 5–19
5–6 Viewing Part of File Using tail 5–19
5–7 Viewing a File Using more 5–20
6–1 touch Command Examples 6–3
6–2 Creating Files Using cat 6–4
6–3 Adding Lines to a File Using cat 6–5
6–4 Appending File Contents to Files Using cat 6–6
6–5 Using Letters and Operation Symbols 6–15
6–6 Using Octal Numbers 6–17
6–7 Creating a Private Project Directory 6–20
6–8 Using the rm Command 6–27
6–9 Generating a Hard Link 6–32
6–10 Creating a Symbolic Link 6–34
7–1 /etc/passwd Entry 7–6
7–2 Setting Some Korn Shell Variables 7–16
7–3 Displaying Korn Shell Variables Using set 7–16
7–4 Displaying Variables Using typeset 7–19
7–5 Defining Variables Using typeset 7–19
7–6 More typeset Commands 7–20
7–7 Changing Korn Shell Options 7–21
7–8 Displaying Korn Shell Options 7–22
7–9 Korn Shell .profile File 7–27
8–1 Mode Line 8–8
8–2 Buffer List 8–36
9–1 Invoking vi for an Existing File 9–6
9–2 Invoking vi for a New File 9–6
9–3 vi Set Options 9–26
9–4 .exrc File 9–27
10–1 Using the wc Command with Standard Input 10–7
10–2 Using a File Name with wc 10–8
10–3 Contents of the Files hot and cool 10–10
10–4 Result of the sort Command on the Files hot and cool 10–10
10–5 Contents of the family File 10–13
10–6 Contents of the baseball File 10–14
10–7 Results of Sorting Files by Fields—The family File 10–15
10–8 Results of Sorting Files by Fields—The baseball File 10–17
10–9 A File of Fruit Prices 10–21
10–10 Using grep to List Berries 10–22
10–11 Specifying a Choice of Characters Using grep 10–23
10–12 Selecting a Range of Characters Using grep 10–24
10–13 Using Negation with grep 10–24
10–14 Using * to Specify Repeating Characters 10–28
10–15 Not Using a Pipeline 10–31
10–16 Using a Pipeline 10–31

10–17 Invoking awk Using a File for Patterns and Actions 10–36
10–18 Using awk with Expressions and Operators 10–38
10–19 Using Logical Operators in awk Programs 10–39
11–1 who Command Output 11–3
11–2 finger Command Output 11–4
11–3 Sending Mail Session 11–9
11–4 Sending and Receiving Mail 11–10
11–5 Using Tilde Escapes to Send Mail 11–12
11–6 Mail Menu 11–12
11–7 Adding set Commands to Your .mailrc File 11–24
12–1 Modifying the Korn Shell History Variables 12–4
12–2 Viewing Korn Shell Commands 12–5
12–3 history Command Output 12–8
12–4 Recalling Commands from the History File 12–9
12–5 Editing Commands in a Temporary Buffer 12–27
13–1 Grouping Commands with the Semicolon 13–6
13–2 Grouping Commands with Parentheses 13–6
13–3 Running Jobs in the Background 13–7
13–4 Using the jobs Command 13–11
13–5 Using the ps Command with No Options 13–13
13–6 Using the ps Command with the -a Option 13–14
13–7 Using the ps Command with the -ef Options 13–15
13–8 Using the ps Command with the -fu Options 13–15
13–9 Suspending the Present Process 13–17
13–10 Using the fg Command 13–18
13–11 Using the bg Command 13–18
13–12 Using the kill Command 13–19
13–13 Using the nice Command 13–19
13–14 Waiting for Completion of a Background Job 13–21
13–15 Checking Exit Status 13–22
13–16 Some at and batch Commands 13–27
13–17 crontab File Entry 13–30
14–1 Simple Korn Shell Script 14–3
14–2 Modifying the PATH Variable 14–4
14–3 Assigning and Displaying Variables 14–8
14–4 Using Korn Shell Variables 14–9
14–5 Using the Korn Shell read Command 14–10
14–6 Korn Shell Built-In Variables 14–11
14–7 Sample Korn Shell .profile File 14–12
14–8 Using Relational Operators 14–19
14–9 Using Logical Operators 14–20
14–10 while Loop 14–21
14–11 until Loop 14–21
14–12 for Loop 14–22
14–13 Execution Based On Success 14–22
14–14 Execution Based On Failure 14–23
14–15 Using the trap Command 14–24

14–16 Debugging Korn Shell Scripts with ksh -x 14–25
14–17 Using nl to Display Line Numbers 14–25
15–1 ftp Command 15–7
15–2 telnet Command 15–8
16–1 Examining /etc/printcap to Determine Available Printers 16–4
16–2 lpq Output 16–10
17–1 Using tar for Disk-to-Tape Operations 17–8
17–2 Using tar for Tape-to-Disk Operations 17–9
17–3 Using the cpio Command 17–13
17–4 Using the ltf Command 17–14
17–5 Using the mt Command 17–17
17–6 Tape Status 17–18
17–7 Using the df Command 17–21
17–8 Using the du Command 17–22

Figures

1–1 Session Manager Pull-Down Menu 1–10
1–2 A DECterm Window 1–11
1–3 Logging Out of the Session Manager 1–13
2–1 Creating a DECterm Window 2–9
2–2 A DECterm Window 2–9
2–3 DECterm Options Menu 2–11
2–4 DECterm General Options Menu 2–12
2–5 DECterm Keyboard Options Menu 2–13
2–6 Session Manager Customize Menu 2–14
2–7 Session Manager Customize Autostart 2–15
3–1 A Bookreader Session 3–5
3–2 Choosing the calendar Application 3–7
3–3 Default Calendar 3–8
3–4 Calendar by Day Screen 3–9
3–5 Entry Editor Window 3–11
3–6 Default (General) Screen 3–12
3–7 Day View Default Screen 3–13
4–1 Hierarchical File System Structure 4–8
5–1 Home and Working Directories 5–3
5–2 Full and Relative Pathnames 5–4
6–1 Creating Directories 6–9
6–2 Results of mkdir Commands 6–10
6–3 Hard Links 6–33
6–4 Symbolic Links 6–34
9–1 vi Modes 9–4
10–1 The awk Field Numbering Scheme 10–35
11–1 DEC OSF/1 Mail Window 11–28
11–2 DEC OSF/1 Mail Create Window 11–29
11–3 Using the write Command 11–31
11–4 A Two-Way Conversation Using the write Command 11–32

11–5 Two-Way Conversation Using the write Command with Protocol 11–34

Tables

1 Course Conventions xxvi
1–1 DEC OSF/1 Features 1–4
1–2 Logging In Using a VT Terminal 1–8
1–3 Logging In Using a Workstation 1–9
1–4 DEC OSF/1 Documentation Set 1–17
1–5 The man Reference Pages 1–18
2–1 Selected Keyboard Settings 2–3
2–2 Creating a DECterm Window 2–8
2–3 DECterm Window Menu Functions 2–10
2–4 Autostarting a DECterm Window 2–15
3–1 DEC OSF/1 Desktop Applications 3–3
5–1 Using the cd Command 5–5
5–2 ls Command Examples 5–7
5–3 whereis Command Options 5–13
5–4 Viewing Files Using the head, tail, and more Commands 5–21
6–1 Commonly Used touch Command Options 6–3
6–2 Creating Directories 6–9
6–3 chmod Command Format 6–14
6–4 chmod Command Parameters 6–16
6–5 Permission Combinations 6–16
6–6 Using the mv Command 6–22
6–7 The cp Command 6–24
6–8 The rm Command Functions 6–27
6–9 The rmdir Command Functions 6–29
6–10 Commonly Used ln Options 6–32
7–1 DEC OSF/1 Shell Features Summary 7–5
7–2 Shell Pathnames 7–6
7–3 Some Variables Set and Updated by the Korn Shell 7–14
7–4 Korn Shell Variables Not Set by the Korn Shell 7–15
7–5 Attributes Used with typeset 7–18
7–6 Some .kshrc Commands 7–28
8–1 emacs Help on Commands 8–13
8–2 emacs Help on General Information 8–14
8–3 Reading, Inserting and Saving Files with emacs 8–17
8–4 Cursor Movement Commands 8–19
8–5 Marking Text 8–21
8–6 Delete, Kill, and Yank Commands 8–22
8–7 New Key Bindings When Using the *.emacs* Initialization File 8–25
8–8 Stopping and Undoing Commands 8–28
8–9 Search Commands 8–31
8–10 emacs Search and Replace Commands 8–32
8–11 Responses to Query Replace 8–34
8–12 Buffer Commands 8–36

8–13 Buffer Listing MR Column 8–37
8–14 Buffer List Commands 8–38
8–15 emacs Window Commands 8–41
8–16 Transposing Characters and Words 8–44
8–17 Cursor Position for Transposition Commands 8–44
8–18 Capitalization Commands 8–45
8–19 Using the Mouse with emacs 8–46
9–1 Editing Modes 9–5
9–2 Interrupting, Cancelling, and Exiting vi 9–7
9–3 Cursor Movement by Characters and Lines 9–8
9–4 Cursor Movement Within a Line 9–8
9–5 Cursor Movement by Words 9–9
9–6 Cursor Movement: Moving to a Line Position 9–10
9–7 Cursor Movement: Moving by Sentences, Paragraphs, or Sections ... 9–11
9–8 Cursor Movement: Paging and Scrolling 9–11
9–9 Searching for Patterns 9–14
9–10 Inserting Text 9–15
9–11 Changing Text 9–16
9–12 Yank and Delete Buffers 9–19
9–13 Copying and Moving Text 9–21
9–14 Restoring and Repeating Changes 9–22
9–15 Writing the Buffer 9–22
9–16 Editing Another File 9–23
9–17 Screen Control Commands 9–23
10–1 Functions of the > Symbol 10–4
10–2 wc Command Options 10–8
10–3 Common Options to the sort Command 10–11
10–4 sort Command Options Relating to Field Position 10–14
10–5 Other Options to the grep Command 10–25
10–6 Matching Arbitrary Characters 10–25
10–7 Escape Character Sets for Metacharacters 10–26
10–8 Expressions 10–37
10–9 Comparison Operators 10–37
11–1 Common finger Command Options 11–4
11–2 Sending Mail 11–8
11–3 Selecting Your Editor in Mail 11–9
11–4 Tilde Escape Functions 11–11
11–5 Mail Symbols 11–13
11–6 Common Mail Commands 11–14
11–7 Working with Folders 11–22
11–8 User Information Commands 11–36
12–1 Korn Shell History Buffer and Command Recall Variables 12–3
12–2 vi Insert Mode Commands 12–15
13–1 Process Status Information Categories 13–12
13–2 Selected ps Command Options 13–12
13–3 DEC OSF/1 kill Commands 13–18
13–4 Useful Priorities for the nice Command 13–20

13–5 Korn Shell Options to the wait Command 13–21
13–6 at Command Options 13–27
13–7 crontab Command Options 13–29
14–1 Variable Assignment 14–7
14–2 Numeric Relational Operators 14–17
14–3 String Relational Operators 14–18
14–4 Positional Parameters 14–18
14–5 Logical Operators 14–19
15–1 rlogin Options 15–3
15–2 rsh Command Options 15–4
15–3 rcp Options 15–5
15–4 Some ftp Commands 15–6
15–5 Some telnet Commands 15–7
15–6 Remote Access Commands 15–15
16–1 Printer Commands 16–3
16–2 Some lp Options 16–6
16–3 Some lpr Options 16–7
16–4 Some pr Options 16–8
16–5 Some lpq Options and Arguments 16–9
16–6 lpstat Options 16–11
16–7 lprm Options 16–12
16–8 lpc User Commands 16–13
17–1 Device Special Files 17–3
17–2 Device Types 17–3
17–3 DEC OSF/1 Disk and Tape Device Mnemonics 17–4
17–4 Commonly Used tar Options 17–6
17–5 Writing Files to Tape Archives 17–7
17–6 Extracting Files from Tape Archives 17–7
17–7 Commands Used with Tapes 17–11
17–8 cpio Command Formats 17–11
17–9 Commonly Used cpio Options 17–12
17–10 Commonly Used ltf Options 17–14
17–11 Commonly Used dd Options 17–16
17–12 Commonly Used mt Commands 17–17
17–13 Some df Options 17–20
17–14 Some du Options 17–22
A–1 stty Command Arguments: Control Modes A–2
A–2 stty Command Arguments: Input Modes A–3
A–3 stty Command Arguments: Output Modes A–4
A–4 stty Command Arguments: Local Modes A–5
A–5 stty Command Arguments: Combination Modes A–7
A–6 stty Command Arguments: Compatibility Mode A–7
B–1 Fetching Commands from the History File B–3
B–2 Command Execution B–3
B–3 Moving the Cursor B–4
B–4 Deleting or Replacing Characters, Words, and Lines B–5
B–5 Entering vi Insert Mode B–6

B–6 Using Uppercase and Lowercase Letters B–6
B–7 Pathname Expansion or Completion B–6
B–8 Miscellaneous Commands B–7

About This Course

General Course Information

Introduction

The *UNIX Utilities and Commands* course is an entry-level, text-based instruction (TBI) course. It is designed to provide you with the fundamental skills needed to become productive using a UNIX type operating system, including DEC OSF/1 UNIX System V or BSD.

The concepts presented in this course are suitable for use in any UNIX environment, and will prepare you to use any UNIX based operating system. The examples, exercises, and their solutions are all designed for DEC OSF/1. This course includes functionality of the DEC OSF/1 Version 2.0 operating system, which may be running on a variety of processors.

The length of the course is five days.

Your instruction is provided by a student workbook that is divided into a number of chapters, each designed to cover a group of topics. Most topics are interspersed with learning activities, provided to reinforce the material presented in the chapter. Solutions to the learning activities are located after each activity. If no solution is required, the statement, "No solution required" is indicated.

If further study is needed, read the `man` reference pages, which we will discuss shortly, for the desired topic.

"About This Course" describes the contents of the course, and suggests ways in which you can most effectively use its materials.

The following topics are discussed:

- Course overview: A brief overview of the course contents
- Intended audience: Who should take this course
- Prerequisites: Skills and knowledge needed to ensure your success in this course
- Course goals: Skills or knowledge the course provides
- Nongoals: Skills or knowledge the course does not provide
- Chapter organization: The structure of a chapter
- Course map: A diagram showing the sequence in which the chapters should be studied
- Suggested schedule: A list of average times needed to complete the course
- How to take this course: A suggested method for taking this course
- Course conventions: The typographical and educational standards used
- Chapter descriptions: A short description of each chapter

- Course materials and resources: Textual or online information you may refer to for further study of the material presented in the chapter
- Preparation for learning activities: Any special preparation that must be carried out prior to starting learning activities

Course Overview

The *UNIX Utilities and Commands* course is designed to help you develop the skills necessary to perform the following functions:

- Set up the terminal or workstation prior to using the DEC OSF/1 operating system
- Manipulate and maintain files within the DEC OSF/1 file system
- Create and edit text files using the `emacs` and `vi` editors
- Execute, recall, and edit shell commands
- Use redirection, filters, and pipes
- Communicate with other users
- Submit and execute processes
- Write and debug Korn shell scripts
- Use TCP/IP networks for file transfer and remote execution
- Print selected files
- Use tape and disk utilities

Intended Audience

The course is designed for customers who have limited or no experience using an ULTRIX UNIX, or OSF environment, who must become familiar with the available utilities and commands. The course is suitable for new users of UNIX systems.

Prerequisites

There are no course prerequisites.

Course Goals

This course is designed to prepare the students to use a UNIX style operating system, including the following tasks:

- Use commands to interface with the operating system
- Identify the relationship between the shell, kernel and software utilities
- Use file manipulation techniques to create directories, move, rename, copy and remove files, and change file and group permissions
- Use the `emacs` and `vi` editors
- Use redirection, filters, piping, and `awk` to process files
- Manage multiple processes
- Communicate with other users
- Write and debug introductory-level Korn shell scripts

- Use TCP/IP networks for file transfer and remote execution
- Use printers and tape drives

Nongoals

This course does not address the following topics:

- Utilities and commands specific to a UNIX System V environment
- Utilities and commands specific to an ULTRIX environment
- troff and ditroff text-formatting utilities
- Operating system installation
- System management techniques
- Detailed shell programming
- Commands necessary to perform hardware or software diagnostic tasks
- C language Programming
- DECnet ULTRIX software
- ULTRIX mail Connection
- VMS ULTRIX Connection (UCX)
- UWS terminal emulator (dxterm, DECterm, or xterm) in detail
- UIL (User Interface Language) programming
- XUI toolkit
- DRM (DECwindows Resource Manager)
- Motif window manager
- International language facilities

Chapter Organization

This course is divided into chapters. Each chapter covers a skill or related group of skills required to fulfill the course goals.

In this course, each chapter consists of:

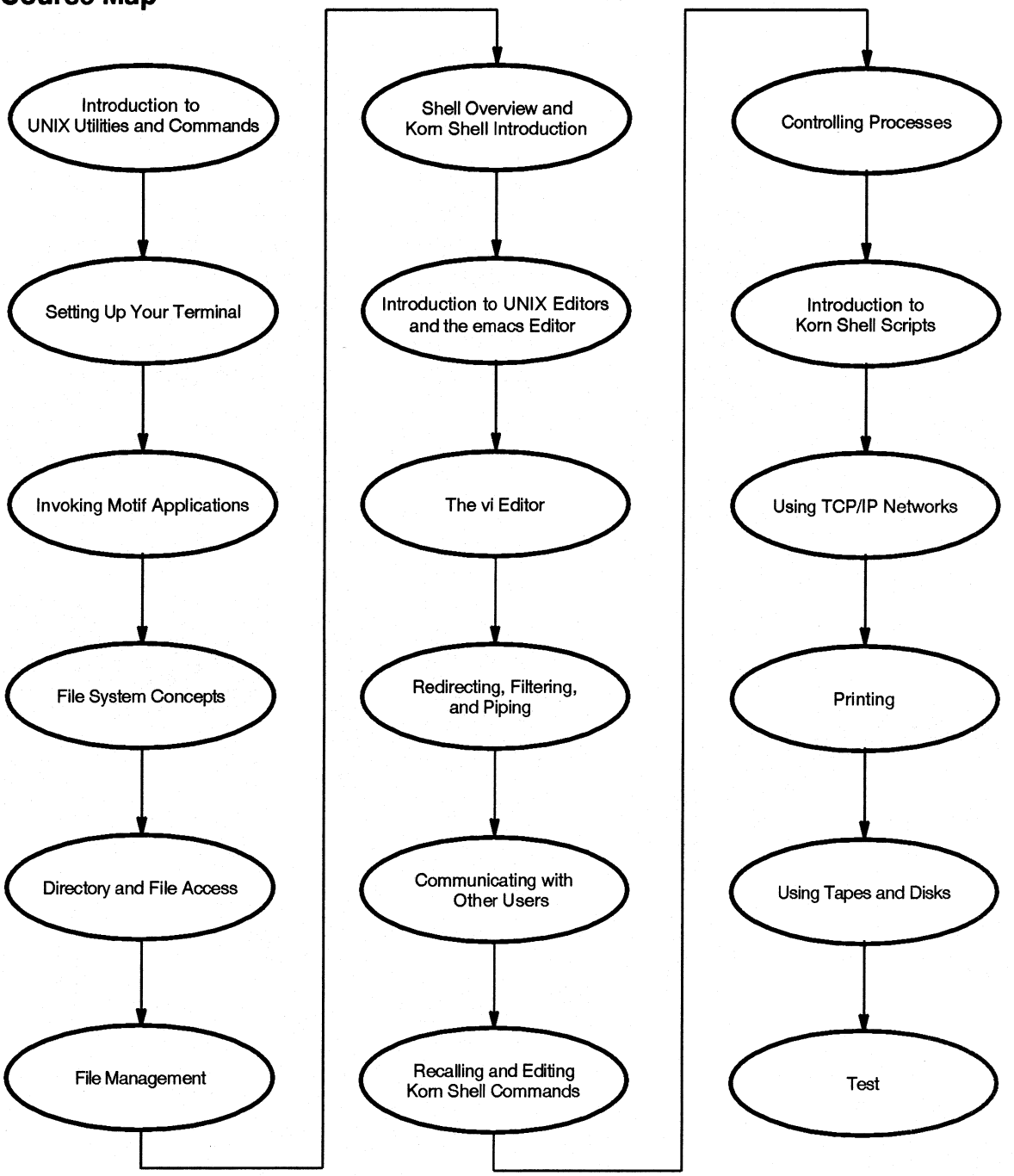
- An **introduction** to the subject matter of the chapter.
- One or more **objectives**, which describe the goals of the chapter.
- A list of **resources**, (manuals you need while studying the chapter). These may be available for reference in your classroom or lab.
- The **text** of each chapter, which includes outlines, tables, figures, examples, and learning activities.
- The **summary** includes a brief review of the topics presented in the chapter.

The learning activities, placed throughout the topics, are powerful instructional elements that provide you with actual operations and concepts. They give you ample opportunity for hands-on practice, and enable you to practice your skills and measure your mastery of the information learned during the course.

Regardless of your level of experience, you are more likely to master the objectives of a unit by reading the instructional text and performing the learning activities in addition to reading the relevant `man` reference pages. Try to spend as much time as possible reading the reference documentation and working at the computer terminal.

The course administrator should provide help in using any of these course materials as well as the necessary reference documentation.

Course Map



ZKOX-055000058-77-RAGS

Suggested Schedule

This course is self-paced, which means that you can learn at whatever rate is comfortable for you. The following schedule suggests the average time in hours needed to complete each chapter.

Course Chapter	Completion Time
Day 1	
1—Introduction to UNIX Utilities and Commands	1.5
2—Setting Up Your Terminal	1.5
3—Invoking Motif Applications	1
4—File System Concepts	1
5—Directory and File Access	1.5
Day 2	
6—File Management	2
7—Shell Overview and Korn Shell Introduction	2
8—Introduction to UNIX Editors and the emacs Editor	3
Day 3	
9—The vi Editor	2.5
10—Redirecting, Filtering, and Piping	3
11—Communicating with Other Users	2
Day 4	
12—Recalling and Editing Korn Shell Commands	2.5
13—Controlling Processes	2
14—Introduction to Korn Shell Scripts	3.5
Day 5	
15—Using TCP/IP Networks	1.5
16—Printing	1.5
17—Using Tapes and Disks	2
18—Test	1
Total	35

The time for learning activities is included in the completion time for each chapter. However, time spent reading `man` pages and other documentation is not included. These are only suggested completion times.

How to Take This Course

This course is intended for independent self-study. You complete chapters according to your own schedule and style of learning. A course administrator will install and maintain the necessary laboratory environment, and provide further assistance if needed. Someone with UNIX, ULTRIX, or DEC OSF/1 system management experience should be available to assist with the technical aspects of the course.

Each course chapter consists of instructional units and learning activities, which provides hands-on practice sessions, allowing you to actually use the DEC OSF/1 software. The learning activities allow you to evaluate your mastery of the material covered in each topic.

Each chapter is divided into main topics. A test is provided after the chapters. Answers to the test follow the questions.

It is suggested that you take this course in the following order:

1. Read the *About This Course* chapter first.
2. Complete *Chapter 1.*

   Begin each chapter by reading the introduction and objectives. Then study the instructional material and do the learning activities. References to documentation resources are provided in each chapter in case you need additional reading for further clarification on a topic. Before leaving the chapter, review the summary, then answer the test questions for the chapter.

   If you have difficulty answering the questions or find that many of your answers are incorrect, review the appropriate chapter and topic.
3. Complete *Chapters 2 through 17.*
4. If you did not answer the test questions for a chapter immediately after studying the chapter, take the test.

Course Conventions

The following table provides a list of the conventions used in this course.

Table 1 Course Conventions

Convention	Description
$	The dollar sign represents the user prompt for the Bourne and Korn shells.
>	The greater than sign represents the Korn shell secondary prompt.
%	The percent sign represents the user prompt for the C shell.

(continued on next page)

Table 1 (Cont.) Course Conventions

Convention	Description
#	The number sign represents the superuser prompt.
bold	Within interactive examples, **boldface** type indicates typed user input.
{ }	In syntax descriptions, braces indicate mandatory items.
[ ]	In syntax descriptions, brackets indicate optional items.
key	All keyboard keys are referred to as boxed keys.
Ctrl/X	Depress and hold the Ctrl key while depressing the X key.
Esc	Depress the Esc key. If F11 does not function as Esc on your terminal, use Ctrl/ [as Esc.

Chapter Descriptions

1. Introduction to UNIX Utilities and Commands: Contains a brief history of UNIX, a basic description of the DEC OSF/1 operating system, and an introduction to Motif. Logging in and out, changing passwords and using the DEC OSF/1 documentation is also included.
2. Setting Up Your Terminal: Describes terminal and workstation setup.
3. Invoking Motif Applications: Describes the use of the mouse with the Motif graphical user interface and some simple desktop applications.
4. File System Concepts: Describes the directory tree structure, types of files and file naming conventions.
5. Directory and File Access: Discusses the concept of pathnames, and basic commands used to access and display the contents of files and directories.
6. File Management: Covers file management commands to create, copy, rename, remove, and protect files and directories.
7. Shell Overview and Korn Shell Introduction: Describes the purpose of a shell, the shells available with DEC OSF/1, and the general features of each of these shells. The Korn shell variables and startup files are also discussed at an introductory level.
8. Introduction to UNIX Editors and the `emacs` Editor: Introduces the editors available with DEC OSF/1, and provides a detailed description of the `emacs` text editor.
9. The `vi` Editor: Covers the `vi` editor at a detailed level.

10. Redirecting, Filtering, and Piping: Covers input and output redirection, common filters such as `wc`, `sort` and `grep`, as well as the use of pipes. Simple text manipulation using `awk` is also presented.
11. Communicating with Other Users: Teaches commands that show who is logged in to the system, and contains an introductory overview of the mail features of the DEC OSF/1 operating system with additional discussion of the talk and write features of the operating system.
12. Recalling and Editing Korn Shell Commands: Defines the use of variables associated with Korn shell command recall, command editing, and the history buffer, and describes how to recall commands from the history buffer, and edit and reexecute Korn shell commands.
13. Controlling Processes: Covers the concept of processes and how to manage them. Understanding the information in this module is fundamental to how successful you will be in using the features of the Korn shell. Topics include the definition of a process, how to create processes, commands that control processes, and how to run processes in the background and at scheduled times.
14. Introduction to Korn Shell Scripts: Provides an introduction to the Korn shell script as well as its creation and execution. This module also covers features of the Korn shell programming environment.
15. Using TCP/IP Networks: Describes how to exchange files with, execute commands on, and how to log in to remote systems.
16. Printing: Discusses how to determine what name to use to access printers available on your system, how to print files, determine what files are waiting to be printed, and how to delete print requests.
17. Using Tapes and Disks: Defines how to identify devices types and names, the `tar` and `ltf` commands for working with tapes, and the `df` and `du` disk commands.

Course Materials and Resources

Additional Resources for Further Study

If you want to study the `emacs` or `vi` editors or the Korn shell beyond the depth covered in this course, the following texts are useful resources. They are not part of the course and are not required to complete the course, but are recommended for further study.

- *GNU emacs Manual*
  Richard Stallman
  published by Free Software Foundation, Cambridge, MA (c) 1987

Note

A copy of the *GNU emacs Manual* is available if the software subset FSFEMACSSRC200 is installed. The pathname for the manual is `/usr/lib/emacs/doc/emacs.ps`.

- *Learning GNU emacs*, ISBN 0-937175-84-6
  Debra Cameron and Bill Rosenblatt
  published by O'Reilly & Assoc. Inc, Sebastopol, CA (c) 1991
- *Learning the vi Editor*
  Linda Lamb
  published by O'Reilly & Assoc. Inc, Sebastopol, CA (c) 1991
- *Kornshell Programming Tutorial*, ISBN 0-201-56324-X
  Barry Rosenberg
  published by Addison-Wesley Publishing Company, Inc. Reading, MA 01867 (c) 1991
- *The Kornshell Command and Programming Language*
  Morris I. Bolsky and David G. Korn
  published by Prentice Hall, Englewood Cliffs, New Jersey 07632 (c) 1989

Hardware/Software Requirements for this Course

This course may be delivered with any UNIX system, subject to the following:

- The examples and exercises are designed for use with the DEC OSF/1 operating system, Version 2.0. Other UNIX systems may be used to deliver the course, but some command formats may be different and system files may be in different locations. The assigned subject matter expert must be able to explain any differences to the students.
- A Motif workstation must be available for each student to utilize the Motif desktop applications. If it is not possible for each student to have a workstation, the course may be taken, but with reduced functionality.

- A TCP/IP network is necessary if students are to use the network commands.
- A print queue must be set up on the UNIX machine in order to use the printer commands.
- A tape drive must be set up on the UNIX machine in order to use the tape commands.
- See the Administrator's Guide for a list of required software subsets for a new installation.

Preparing for Learning Activities

General Learning Activities

Although you must create a file for some learning activities, these files are short and require a minimum of typing. Check with your course administrator to determine if any of them already exist on line.

Preparing for the emacs Learning Activities

If you are using the emacs editor from a terminal or in a DECterm window, you can use an .emacs initialization file to define other keys in place of Ctrl/S and Ctrl/Q in emacs. The contents of one possible .emacs file is shown in Example 1.

Example 1 .emacs Initialization File

```
(define-key global-map "\C-\\" 'isearch-forward)      ; sets isearch-forward

(defconst search-repeat-char ?\C-\\                   ; sets the search repeat
"Character to repeat incremental search forwards.")   ; character to Ctrl/ \

(define-key esc-map "s" 'save-buffer)                 ; redefines a key for
                                                      ; saving a buffer

(define-key ctl-x-map "q" 'toggle-read-only)          ; redefines a key for
                                                      ; toggle-read only
```

Preparing for the Printer Learning Activities

Prior to starting the printer lab exercise, have the course administrator or system manager, as superuser, stop the spooling daemon on the printer or printers that you have access to. Printing is not to be disabled, just the spooling daemon. This will enable you to queue up print requests, but prevent printing the requests, allowing you to view and delete your requests, and save a few trees.

The command to turn off all, or just the applicable printer, (if there are printers other than the default printer) is:

/usr/sbin/lpc stop [all | printer]

After you have completed the printer learning activity, have the course administrator or system manager clear the queues with lprm -Pprinter - , then restart the spooling daemon with:

/usr/sbin/lpc start [all | printer]

1

Introduction to UNIX Utilities and Commands

UNIX Utilities and Commands

Introduction

This chapter is an overview, a collection of different topics that you should be familiar with before starting later topics. To better understand and appreciate the UNIX operating system, you should know its history.

It is also helpful to look at the UNIX operating system as a whole, and its features and characteristics in general before studying them in detail in the following chapters. Organization of the UNIX documentation is also covered, as well as how to log in and out of the system and change passwords.

Objectives

To use a UNIX operating system, you should be able to:

- Describe the history of the UNIX operating system and its relation to DEC OSF/1
- Briefly describe the UNIX operating system
- Describe the OSF/Motif windowing interface
- Log in and log out of the system
- Change your password
- Locate specific commands in the DEC OSF/1 documentation set
- Locate specific commands by using the `man` command

Resources

- *OSF/1 User's Guide*
- *ULTRIX to DEC OSF/1 Migration Guide*
- *OSF/Motif User's Guide*

History of UNIX

Creators of UNIX

UNIX was created in 1969 by Ken Thompson at Bell Laboratories. Thompson originally wrote the UNIX operating system in PDP-7 assembly language. Thompson was responsible for the development of the UNIX operating system and its associated philosophy and concepts. He was joined by Dennis Ritchie who had earlier developed the "C" programming language. Together, they rewrote almost the entire operating system in this versatile, high-level programming language. UNIX was one of the first operating systems to be written in a high-level language.

UNIX Evolution

UNIX, Version 6, released in 1976, was the first version of UNIX to be made widely available outside of Bell Laboratories. In 1978, Bell Laboratories began distributing UNIX, Version 7. At this time Bell Laboratories was not in the computer industry. Bell Laboratories made UNIX available to computer science departments at colleges and universities free of charge (for the price of the distribution media) and free of licensing restrictions. UNIX became the object of widespread research and development at both undergraduate and graduate levels. Many advances in UNIX development occurred in academic institutions outside of Bell Laboratories.

Berkeley (BSD) UNIX

In 1978, the University of California at Berkeley acquired a research version of UNIX designed to run on DEC PDP hardware. The University of California at Berkeley improved the operating system substantially, especially in the area of networking, and it began its own distribution of UNIX called Berkeley Software Distribution (BSD). The first release was 3BSD in 1979 and several subsequent versions have since been issued. The current release is 4.4BSD.

UNIX has always run on Digital Equipment Corporation hardware, and in 1982 Digital released its own version of UNIX called the ULTRIX operating system which is based on 4.2BSD with 4.3BSD functionality because of its networking capabilities.

AT&T UNIX and System V

DEC OSF/1 offers users the programming environment features of AT&T System V and BSD, which are not available in most UNIX systems.

Open Software Foundation

On May 17, 1988, six leading computer companies announced an international foundation to develop and provide a completely open software environment to make it easier for customers to use computers and software from many vendors. These companies were Digital, Apollo, Groupe Bull, Hewlett-Packard, IBM and Siemens. The group is called the Open Software Foundation, or OSF.

OSF is developing a software environment, including application interfaces, advanced systems extensions and a new operating system, using X/Open and POSIX specifications as the starting point.

OSF is incorporated as a nonprofit, industry-supported research and development organization. It will define specifications, develop a leadership operating system, and promote an open, portable application environment.

Digital OSF/1

Digital provides the DEC OSF/1 operating system environment, an operating system based on the Open Software Foundation's operating system, OSF/1. This operating system has the "look and feel" of UNIX.

Digital OSF/1 Features

The following table lists the features provided with the DEC OSF/1 operating system.

Table 1–1 DEC OSF/1 Features

Feature	Description
Programming Environment	Provides alternative programming environments. Offers the programming environment features of AT&T System V and BSD UNIX systems.
Shared Libraries	A key improvement in load time and space saving. Shared libraries are loaded into memory at boot time. Developers have the option of loading them dynamically.
Real-Time Features	Includes a preemptive kernel, fixed-priority scheduling, clocks and timers, process memory locking, asynchronous I/O and threads.
GUI	Motif V1.2 is based on the X Window system developed at Massachusetts Institute of Technology.
MIT X11 Release 5 Servers	Provides improvements in performance and memory utilization.

(continued on next page)

Table 1–1 (Cont.) DEC OSF/1 Features

Feature	Description
Logical Volumes	Provide the ability to have a single file span multiple physical volumes. Logical volumes increase system reliability and performance.
File System	Supports all major UNIX file systems: The BSD UNIX UFS file system The NFS file system The ISO 9660 (CDROM file system standard)

DEC OSF/1 Operating System

Overview

DEC OSF/1 is Digital Equipment Corporation's version of the Open Software Foundation's UNIX operating system called OSF/1. OSF/1 was developed by OSF and was first available for licensing to other vendors in 1990. OSF licenses its OSF/1 operating system to vendors like Digital who then port the software to their own hardware. At the same time, these vendors can add or remove certain features to create their own operating systems.

Description

The DEC OSF/1 operating system is a multiuser, interactive time-sharing system. It allows you to run more than one program at a time while performing other online tasks simultaneously.

DEC OSF/1 includes utilities and commands for:

- Electronic mail and phone communication capabilities
- File manipulation, organization, and protection
- Text processing
- Searching files
- Sorting and merging files
- Programming tools
- Record searching and reporting utilities

DEC OSF/1 supports a variety of hardware configurations which can be obtained from the Software Product Description.

The DEC OSF/1 operating system uses **hierarchical file systems** which offer increased organization and security at several levels. Public directories are accessible to everyone. These directories store commands, online documentation, and other important information and programs.

The **shell** is the command interpreter for the operating system – the translator that communicates your requests to the system. The shell also contains a full programming language used to write command procedures and prototype applications in the form of **shell scripts**.

The Windowing Interface

Overview

The DEC OSF/1 operating system provides a windowing graphical user interface (GUI) for workstations, and a terminal interface for character-cell terminals such as the VT320 or a workstation window emulating a terminal. The workstation interface is based on the industry standard OSF/Motif GUI.

For additional information refer to:

- *OSF/Motif User's Guide*
- *ULTRIX to DEC OSF/1 Migration Guide*

  Appendix D of this book provides a summary of XUI and OSF/Motif differences.

OSF/Motif

The OSF/Motif interface was developed by the Open Software Foundation and released in 1989. It functions similar to the ULTRIX DECwindows interface based on X user interface, XUI. The main difference is in the look and feel of the three-dimensional visuals such as window frames and push buttons.

Motif includes:

- The OSF/Motif Window Manager which customizes the interface appearance and behavior
- A user interface toolkit composed of GUI objects
- A user interface language (UIL), which describes the visual aspects of objects

Motif lets you create and manipulate windows on your workstation screen using a mouse to point, click, and drag your selections. You can personalize your Motif environment to suit your own needs and tastes using the Session Manager menu options.

Logging In and Out of the System

Overview

Your system administrator must provide you with **username** and **password** information before you can log in to the UNIX operating system. Your user name is typically some form of your name and your password is a word or group of characters that confirms your identity.

Once you know your user name and password information, you are ready to log in. Enter information in response to system requests, or prompts. The prompts for logging in and system response differ depending upon the type of terminal you are using.

Note

UNIX is sensitive to uppercase and lowercase letters and treats them as different characters. When you receive your user name and password from your system administrator, be sure to note uppercase and lowercase letters.

Throughout this course, whenever you are asked to **enter** a command, you should type the command and press Return. In contrast, the word **type** means to type the characters and **not** press Return.

Note

Enter commands exactly as they appear in the course or in the documentation. Be careful, because some commands may have upper and lowercase letters.

Using a VT Terminal

The table lists the system prompts you will see when you log in using a VT terminal, and the appropriate response to make.

Table 1–2 Logging In Using a VT Terminal

System Prompt:	User Action:
`login:`[1]	Type your user name and press Return.
`Password:`	Type your password and press Return. For security reasons, when you type your password, you will not see it on your terminal.

[1]On some systems, you must press Return a few times to display the login: prompt.

(continued on next page)

Table 1–2 (Cont.) Logging In Using a VT Terminal

System Prompt:	User Action:
`Last login: Mon Jun 15 10:07:00 ...` `$`	You will see the date and time of your last login. Depending upon your system, a welcome message may also appear. You are now in the system and can start entering commands at the **shell prompt** ($), also called the command line.

Note

In this course the dollar sign ($) is usually used as the shell prompt. The symbol for the shell prompt is customizable and may not be a dollar sign on your system.

Common Problems:

- If you make a mistake in the login procedure, the system responds with `login incorrect` and gives you the opportunity to log in again.
- Do not backspace to correct typing errors when you are entering your user name or password. Use the ⌫ key.
- If your terminal responds with unusual characters immediately after you log in, you probably pressed the Lock key when you logged in. The best thing to do is log out by typing `exit` or by pressing Ctrl/D, and log in again.

Using a Workstation

The table lists the system prompts for logging in using a workstation and the appropriate user response.

Table 1–3 Logging In Using a Workstation

System Prompt:	User Action:
Name:	Type your user name and press Return.[1]
Password:	Type your password and press Return. You will see an **xcons** box icon and a Session Manager window appear on your screen.

[1]Make sure that the mouse pointer is located in the login window on the middle of the screen before entering your user name.

With the Session Manager window up and running on your worksystem, you have officially begun a login session. However, you must open a terminal window before entering commands. This terminal window is called a DECterm window.

Common Problems:

- If you make a mistake in the login procedure, the system responds with a window that includes the message "Invalid attempt to login." Press Return and try again.
- Do not backspace to correct typing errors when you are entering your user name or password. Use the ⌫, or delete key.

Starting a DECterm Window

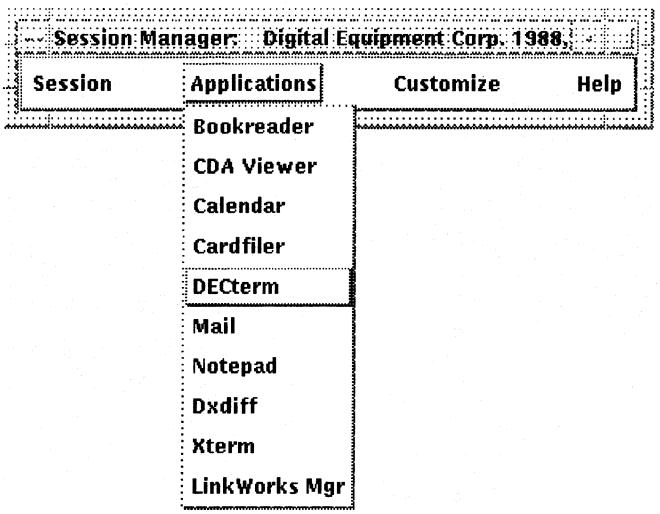
Depending upon your system startup procedure, you can open a DECterm window automatically after the Session Manager window. If so, the DECterm window appears.

If a DECterm window has **not** started, start one to enter shell commands. To do this:

1. Choose DECterm... from the Applications menu in the Session Manager window.
2. Once you have a DECterm window and the shell prompt appears, position your mouse anywhere in the DECterm window and press MB1. At this point, you can begin entering commands.

The following figure shows a pull-down menu of choices that appears.

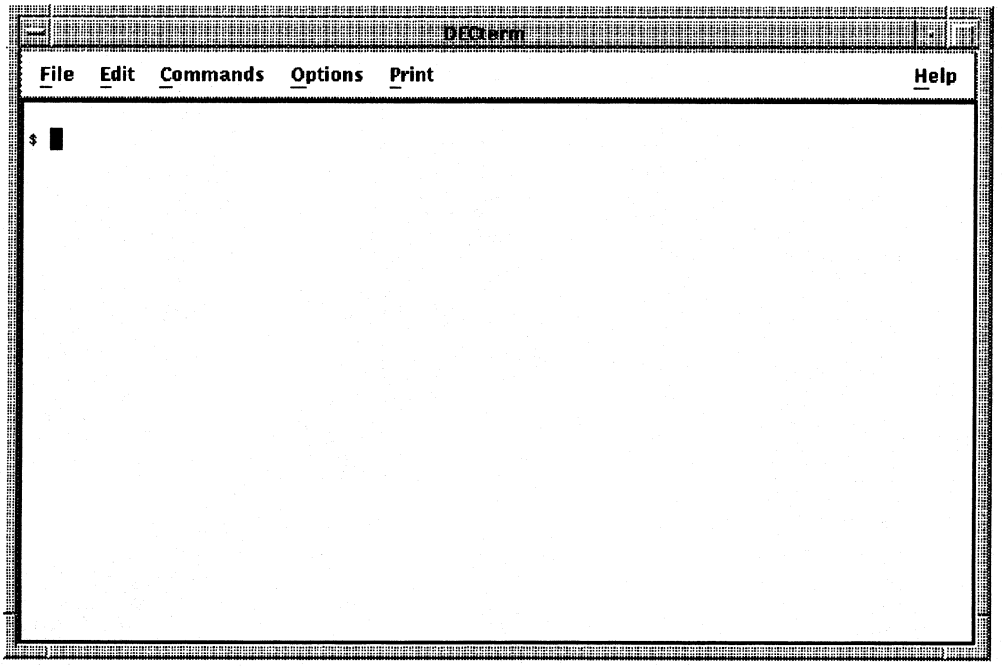
Figure 1–1 Session Manager Pull-Down Menu



ZKOX-055000058-11-RAGS

The figure shows the DECterm window that appears after choosing the DECterm option.

Figure 1–2 A DECterm Window



ZKOX–055000058–12–RAGS

Entering a Command

Enter the name of the command to issue a command on a UNIX system. For instance, entering the `date` command displays the date and time on your system.

The format of the `date` command is:

```
date  [options]
```

where `[options]` allows you to specify command options to the `date` command. See the reference material for more options to the `date` command.

With no options, the `date` command displays the system date and time in a system standard format:

```
$ date
Tue Feb 15 15:54:31 EST 1994
$
```

The options allow you to format the date output in a variety of ways. Guidelines for using options:

- Options must be preceded by a plus sign (+).
- Each individual option must be preceded by a percent sign (%).

- If more than one option is used, enclose all of them in double quotes (").
- Any other characters placed within the quotes are sent to standard output unmodified. This allows the inclusion of format characters such as commas, colons, or periods.

The first `date` command in the example shows the `date` command options `%A,%B, %d, %Y, %I, %M,` and `%p`. A comma separates the date from the year, and a semicolon separates the year from the time of day.

The second command contains lowercase letters for the `a, b,` and `y` options, changes the hour from 12 to 24 hour format, and adds an `n` to force a new line.

Example 1–1 Using the date Command with Options

```
$ date +"%A %B %d, %Y; %I%M %p"
Thursday February 17, 1994; 0214 PM
$
$ date +"%a %b %d, %y; %n%H%M %p"
Thu Feb 17, 94;
1415 PM
$
```

Logging Out of a VT Terminal

There are two ways to log out of the system from a terminal:

- Enter `exit` on the command line and press Return.
- Press the exit key sequence, often mapped to Ctrl/D. To use Ctrl/D, hold down the Ctrl key and press D at the same time. However, Ctrl/D can be redefined to perform another function and may not work as a logout command.

Logging Out of a Workstation

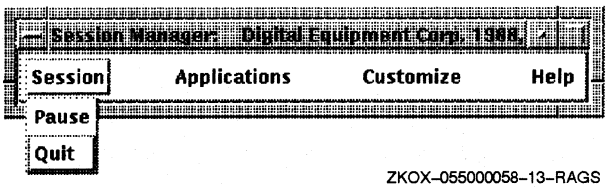
The Session Manager window controls your login session on a workstation. You can log out of a DECterm window the same way you log out of a VT terminal. This merely closes the DECterm window.

The Session Manager remains open so that a new DECterm window or other application can be started if you choose. You are not completely logged out until you quit the Session Manager.

To do this, choose Quit... from the Session menu in the Session Manager window.

Depending upon how your system is customized, you may have to click on Yes in a verification box to log out.

Figure 1–3 Logging Out of the Session Manager



Learning Activity: Logging In and Out

1. Log in to the system.
2. Use the `date` command to see the current date and time. Enter `date` in lowercase letters at the system prompt `$`.
3. Try both methods of logging out.
   a. Log out using Ctrl/D. If the key has not been reassigned, you are logged out. If this happens, log in again.
   b. Log out using the `exit` command.

Learning Activity Solution: Logging In and Out

1. Log in to the system.

   `login:` (Enter your user name)
   `password:` (Enter your password)
2. Use the `date` command to see the current date and time. Enter `date` in lowercase letters at the system prompt `$`.

   ```
   $ date
   Thu Feb 03 11:32:47 EST 1994
   ```
3. Try both methods of logging out.
   a. Log out using Ctrl/D. If the key has not been reassigned, you are logged out. If this happens, log in again.

      `$` Ctrl/D
   b. Log out using the `exit` command.

      ```
      $ exit
      ```

Passwords

Overview

When you are given an account on the system, you are also given a password. When you log in to your account for the first time, you should change your password. Your password is confidential. Once you set it, you are the only person who knows it and the only person to have access to your account.

Choosing Passwords

When choosing passwords, do **not**:

- Choose words found in the dictionary
- Use personal information
- Use your default (original) passwords
- Choose a password that is difficult to remember; it should be hard to guess, but not hard to remember
- Write down your password

Password Guidelines

Passwords can contain nonalphanumeric characters as well as letters and numbers. Select a password that is easy to remember, but not one that is so obvious that other system users can guess it. For security reasons, it is best to select a password containing more than six characters.

On a DEC OSF/1 system, passwords:

- Must contain at least five characters

  If you enter fewer than five characters as a password, the system requests that you enter a longer password.
- Can contain more than eight characters, but only the first eight are recognized.

  If more than eight characters are entered, the system warns you that only the first eight characters of the password will be used.
- Must consist of mixed upper and lowercase letters. They must include at least one uppercase letter, one digit, or one control character.

  If you enter all lowercase letters as the password, the system advises you to include some uppercase letters, digits, or control characters in your password. Your password can be:
  - All uppercase letters
  - All numbers
  - All control characters
  - Any combination of letters, numbers, and or control characters

- Should be changed frequently

Changing Your Password

The `passwd` command changes your password. When you enter `passwd`, the DEC OSF/1 system prompts you to:

- Enter your old password
- Enter your new password
- Confirm your new password

When you enter a password, it is not echoed (displayed) on the screen.

The following table lists the system prompts and what you enter when changing your password from **Magic!** to **Mundane.**

System Prompt	Enter	Comment
`$`	**`passwd`**	Enter `passwd` command
`Changing password for` *`[username]`* `Old password:`	**`Magic!`**	Enter current password, **`Magic!`**. It does not echo on the screen
`New password:`	**`Mundane`**	Enter new password, **`Mundane`**. It does not echo on the screen
`Retype new password:`	**`Mundane`**	Confirm new password, **`Mundane`**. It does not echo on the screen
`$`		System prompt displayed

Confirming Your Password

If you make a mistake typing your old password, the system displays the message `Sorry.`

The system requires you to confirm your password by reentering it. Password confirmation ensures that you do not set your password to an unknown word derived from a typographical error — unless you make the same mistake twice.

If what you type when confirming your password does not match what you typed as your password, the system informs you by displaying the message `Mismatch - password unchanged.` and does not change your original password.

If either of these problems occur, reenter the `passwd` command to start the process again.

Note

If you forget your password, your system administrator can reset it for you.

Learning Activity: Changing Passwords

Select a password that is easy to remember.

Change your password to include both letters and numbers. The password must contain more than five characters. Although it can contain over eight characters, OSF/1 recognizes only the first eight characters. There must be at least one uppercase letter, number, or control character in the password.

1. Change your password to Mundane. When asked to retype your new password, purposely enter Mundame to observe the system response. Notice how your password remained unchanged.
2. Change your password to something unique. **Remember this password! You will be using it for your exercises throughout the course.**

Learning Activity Solution: Changing Passwords

Select a password that is easy to remember.

Change your password to include both letters and numbers. The password must contain more than five characters. Although it can contain over eight characters, OSF/1 recognizes only the first eight characters. There must be at least one uppercase letter, number, or control character in the password.

```
$ passwd
Changing password for you.

Old password: (Enter old password)
New password: (Enter new password with letters and numbers)
Retype new password: (Enter new password again for verification)
$
```

1. Change your password to Mundane. When asked to retype your new password, purposely enter Mundame to observe the system response. Notice how your password remained unchanged.

   ```
   $ passwd
   Changing password for you.

   Old password: (Enter old password)
   New password: Mundane
   Retype new password: Mundame
   Mismatch - password unchanged.
   $
   ```

2. Change your password to something unique. **Remember this password! You will be using it for your exercises throughout the course.**

   ```
   $ passwd
   Changing password for you.

   Old password: (Enter old password)
   New password: (Enter new password)
   Retype new password: (Enter new password again for verification)
   $
   ```

DEC OSF/1 Documentation

Overview

Digital supplies a complete set of manuals with each DEC OSF/1 operating system. DEC OSF/1 documentation is available in the traditional hardcopy documentation set consisting of several manuals. OSF/1 commands are also documented online in the `man` pages.

Using Hardcopy Documentation

The DEC OSF/1 documentation set is divided into four separate categories by usefulness for the target audience. It consists of the following kits:

Table 1–4 DEC OSF/1 Documentation Set

Documentation Kit	Description
Startup	Designed for those who install the DEC OSF/1 operating system – *Release Notes* – *Guide to Installing DEC OSF/1* – *Realtime Installation Guide* – *System and Network Setup and Configuration*
General User	Designed for general users or anyone who uses the DEC OSF/1 operating system – *Reader's Guide* – *OSF/Motif User's Guide* – *OSF/1 User's Guide* – *Network and Communications Overview*
System and Network Management	Designed for anyone who manages a DEC OSF/1 system or network – *Guide to System Administration* – *Network & Communications Management and Problem Solving* – *Guide to Prestoserve*

(continued on next page)

Table 1–4 (Cont.) DEC OSF/1 Documentation Set

Documentation Kit	Description
Programming	Designed for those who develop programs for the DEC OSF/1 operating system. It is divided into two categories: – General programming: Geared toward any programmer developing applications on or for DEC OSF/1 operating system. – Window Programming: Contains programming information specific to developing X window applications. The General Programming and Window Programming documentation sets contain additional manuals.

Using Online Documentation

Online documentation is furnished with the operating system when it is purchased. Its installation on the system is optional but highly recommended. If the online pages have been installed on your system, access them by using the `man` command.

Organization of man Reference Pages

The `man` reference pages are separated into sections according to function as follows:

Table 1–5 The man Reference Pages

Section	Topic
`man(1)`	Commands
`man(2)`	System Calls
`man(3)`	Subroutines
`man(4)`	Special Files
`man(5)`	File Formats
—	There is no `man(6)`, which is "games" on some UNIX systems
`man(7)`	Macro Packages and Conventions
`man(8)`	Maintenance

Using the man Command

The `man` command provides information about particular commands. If you want information about the `passwd` command, enter **`man passwd`**. The `man` command displays information about a command, one screenful at a time. Press Return or Spacebar to see additional text, as shown in the table.

Key	Function
Return	Displays additional information, one line at a time
Spacebar	Displays additional information, one screenful at a time
Q OR q	Stops displaying additional information and returns you to the command-line prompt

Using the man passwd Command

Entering **man passwd** accesses the online page for the passwd command and displays it on the screen, as shown (abbreviated) in the following example.

Example 1–2 Using the man passwd Command

```
$ man passwd
passwd(1)                          OSF                          passwd(1)

NAME

  passwd, chfn, chsh - Changes password file information

SYNOPSIS

  passwd [-f | -s] [username]

  chfn [username]

  chsh [username]

  The passwd command changes (or installs) the password associated with your
  username (by default) or the specified username.  The chfn command changes
  GECOS information associated with your username or the specified username.
  The chsh command changes the login shell of your username or of the speci-
  fied username.

  .
  .
  .

FILES

  /etc/passwd
            Contains user information.

  /etc/shells
            The list of approved shells.

RELATED INFORMATION

  Commands: finger(1), login(1).

  Files: passwd(4).
$
```

Using apropos to Learn About Related Commands

The apropos command lists commands related to a particular topic. To list commands related to passwords, enter **apropos password.**

- The man command displays reference information about a command.

- The `apropos` command lists information related to a topic. The `apropos` command is the same as the `man` command with the *-k* option.

Using the help Command

The DEC OSF/1 system does not have a Help facility. However, if you enter the `help` command, the system gives you a brief list of helpful commands and their functions, as shown in the example.

Example 1–3 Output of the help Command

```
$ help
The commands:
    man -k keyword      Lists commands relevant to a keyword
    man command         Prints out the manual pages for a command

                        Other basic commands are:

    cat                 - concatenates files (and just prints them out)
    vi                  - text editor
    finger              - user information lookup program
    ls                  - lists contents of directory
    mail                - sends and receives mail
    passwd              - changes login password
    sccshelp            - views information on the Source Code Control System
    tset                - sets terminal modes
    who                 - who is on the system
    write               - writes to another user
You could find programs about mail by the command:      man -k mail
and print out the man command documentation via:        man mail
You can log out by typing "exit".
$
```

Learning Activity: Using Documentation

1. Use the `man` command to display the online description of the `passwd` command. Press the Spacebar to continue reading. Type Q when you have finished reading.
2. Use the `apropos` command to display commands related to "password".

Learning Activity Solution: Using Documentation

1. Use the `man` command to display the online description of the `passwd` command. Press the Spacebar to continue reading. Type Q when you have finished reading.

```
$ man passwd
passwd(1)                                   passwd(1)

NAME

    .
    .
    .
```

2. Use the apropos command to display commands related to "password".

```
$ apropos password
acceptable_password (3) - Determines if a password meets deduction requirements
conflict (8)            - Searches for alias/password conflicts
getpass (3)             - Reads a password
    .
    .
    .
```

Summary

History of UNIX

DEC OSF/1 is an open system based on the OSF operating system called OSF/1. It has the "look and feel" of UNIX.

DEC OSF/1 Operating System

DEC OSF/1 Version 2.0 is the Digital implementation of OSF/1 developed by the Open Software Foundation.

The Windowing Interface

The DEC OSF/1 windowing interface for workstations is the OSF/Motif graphical user interface.

- OSF/Motif includes:
  - The window manager
  - A user interface toolkit
  - A user interface language
- Motif is mouse driven and provides advanced features such as three-dimensional visuals, scroll bar sliders, and cascade menus.
- Use the Session Manager menu options to personalize Motif

Logging In and Out of the System

- You must know your user name and password to log in to the system.
- The DEC OSF/1 command line is indicated by a customizable symbol. This course uses the `$` to indicate the command line.
- Use Ctrl/D or the `exit` command to log out of the system.

Passwords

Use the `passwd` command to change your password. Passwords:

- Can contain nonalphanumeric characters as well as letters and numbers.
- Should contain at least five characters.
- Can contain more than eight characters, but only the first eight are recognized.
- Should not consist entirely of lowercase letters. Passwords should include at least one uppercase letter, one digit, or one control character.
- Should be changed frequently.

DEC OSF/1 Documentation

The DEC OSF/1 operating system includes both the traditional printed documentation set and an online documentation set. Use this documentation to obtain information on the format of commands and their functions.

- The `help` command displays a brief listing of some DEC OSF/1 commands and their descriptions.
- Use the `man` command to display reference information about commands.

2

Setting Up Your Terminal

Getting Started

Introduction

The first chapter discussed how to log in to the UNIX system, enter and change your password, and obtain online help.

This chapter discusses how to set up your terminal or workstation for proper UNIX operation.

Learning activities are presented throughout the chapter so you can become more familiar with the terminal and workstation setup procedures.

Objectives

To efficiently use the terminal or a workstation, you should be able to:

- Perform terminal and keyboard setup
- Perform workstation setup

Resources

- *OSF/1 User's Guide*
- DEC OSF/1 `man` pages

Terminal Setup

Overview

When you log in to a UNIX system, you can determine which keys are automatically set to default values. You can use these keys at either the command level or at the system prompt.

This section pertains only to users working on terminals or in DECterm windows on workstations. Workstation setup is covered in another section.

Displaying Keyboard Settings

You can determine your terminal settings by using the Set TTY (stty) command. The `stty everything` command displays the keyboard settings, the speed of the terminal, and settings that affect your screen, as shown in the following example.

Note

The caret character (^) indicates the Ctrl key, so ^C means Ctrl/C in the following example.

Example 2–1 Using the stty everything Command

```
$ stty everything
#2 disc;speed 9600 baud; 24 rows; 80 columns
erase = ^?; werase = ^W; kill = ^U; intr = ^C; quit = ^\; susp = ^Z
dsusp = ^Y; eof = ^D; eol <undef>; eol2 <undef>; stop = ^S; start = ^Q
lnext = ^V; discard = ^O; reprint = ^R; status <undef>; time = 0
min = 1
-parenb -parodd cs8 -cstopb hupcl cread -clocal
-ignbrk brkint -ignpar -parmrk -inpck -istrip -inlcr -igncr icrnl -iuclc
ixon -ixany -ixoff imaxbel
isig icanon -xcase echo echoe echok -echonl -noflsh -mdmbuf -nohang
-tostop echoctl -echoprt echoke -altwerase iexten -nokerninfo
opost -olcuc onlcr -ocrnl -onocr -onlret -ofill -ofdel tabs -onoeot
$
```

Using Special Keyboard Settings

Some helpful keyboard settings are shown in the following table.

Table 2–1 Selected Keyboard Settings

Command	Key[1]	Function
erase	Ctrl/ ?[2]	Backs up one space and removes a single character each time it is pressed

[1]Default: a different key can be mapped to each function.

[2] <X sends Ctrl/ ?

(continued on next page)

Table 2–1 (Cont.) Selected Keyboard Settings

Command	Key[1]	Function
`kill`	Ctrl/U	Erases the entire command line, discarding the command
`intr`	Ctrl/C	Stops a command that is currently executing If you are typing a line and press `intr`, the system ignores the line you were typing and returns to the prompt
`stop`	Ctrl/S	Suspends output (stops scrolling)
`start`	Ctrl/Q	Resumes output (starts scrolling)
`eof`	Ctrl/D	Designates an end of file
`susp`	Ctrl/Z	Suspends the current process and returns control to the parent process

[1]Default: a different key can be mapped to each function.

Using Control Keys

Type a control sequence by holding down the Ctrl key and pressing the specified key at the same time.

Note

When using the key functions to delete characters from commands, characters may or may not physically disappear from the screen. Although they still appear on the screen, characters are actually deleted and can be retyped or reentered.

For a complete description of all information displayed by the `stty everything` command, refer to Appendix A.

For more information on the `stty` command, enter the command: `man stty`. Press Spacebar to read the next screen, or press Q to return to the command-line prompt.

Changing Keyboard Settings

Although the UNIX system automatically sets up default key functions, you may want to change these settings to meet your individual needs.

When using an operating system with a different keyboard setup, you might want to set the `erase`, `kill`, and `intr` functions to more familiar settings.

The format of the `stty` command is:

```
stty function char
```

where `function` is a keyboard function such as `erase`, and `char` is the new key or control key sequence you want to associate with the function.

For example:

```
$ stty intr ^C
$ stty erase ^?
```

Functions that do not take character arguments can be set as follows:

`$ stty even` (allows even parity input)

`$ stty -even` (disallows even parity input)

Undefining a Key Definition

You can also reset a function by undefining it. To set `kill` undefined, enter:

```
$ stty kill undef
```

Saving Keyboard Settings Between Sessions

When you log out and then log in again, you must set up your keyboard again by issuing the same sequence of `stty` commands.

An example command line in your startup file can be:

```
stty -tabs dec new cr0.
```

Learning Activity: Terminal Setup

This exercise uses the stty command to display and change your terminal and keyboard settings. Use the man stty command as necessary when you need additional information regarding the stty command.

Using Keyboard Settings

1. Enter the stty everything command to display your terminal settings. Your settings may be different from those illustrated here.
2. Type a shell command incorrectly, but do not press Return. Press several of the Ctrl keys to see how they function.
3. Use the man command to learn more about stty. Press Ctrl/C or Q to return to the command line.

Changing Key Functions

1. Use the stty command to change the erase character to Ctrl/H.
2. Display this new keyboard setting on your screen using the stty command.
3. Type a command incorrectly on the command line and press Ctrl/H to see its new function.
4. Use the stty command to change the erase character back to ⌫.

Learning Activity Solution: Terminal Setup

This exercise uses the stty command to display and change your terminal and keyboard settings. Use the man stty command as necessary when you need additional information regarding the stty command.

Using Keyboard Settings

1. Enter the stty everything command to display your terminal settings. Your settings may be different from those illustrated here.

```
$ stty everything
#2 disc;speed 9600 baud; 24 rows; 80 columns
erase = ^?; werase = ^W; kill = ^U; intr = ^C; quit = ^\; susp = ^Z
dsusp = ^Y; eof = ^D; eol <undef>; eol2 <undef>; stop = ^S; start = ^Q
lnext = ^V; discard = ^O; reprint = ^R; status <undef>; time = 0
    .
    .
    .
```

2. Type a shell command incorrectly, but do not press Return. Press several of the Ctrl keys to see how they function.

   No solution required.

3. Use the man command to learn more about stty. Press Ctrl/C or Q to return to the command line.

```
$ man stty          stty(1)
stty(1)

NAME

     .
     .
     .
```

Changing Key Functions

1. Use the `stty` command to change the `erase` character to Ctrl/H.

   ```
   $ stty erase Ctrl/H
   ```

2. Display this new keyboard setting on your screen using the `stty` command.

   ```
   $ stty -a
   #2 disc;speed 9600 baud; 24 rows; 80 columns
   erase = ^H; werase = ^W; kill = ^U; intr = ^C; quit = ^\; susp = ^Z
       .
       .
       .
   ```

3. Type a command incorrectly on the command line and press Ctrl/H to see its new function.

   No solution required.

4. Use the `stty` command to change the `erase` character back to <X.

   ```
   $ stty erase <X
   ```

Workstation Setup

Overview

The DEC OSF/1 Window Manager provides a terminal emulation application called DECterm. DECterm windows are used to enter DEC OSF/1 commands, or to run applications that are only supported on terminals.

Note

If you are left handed, you may want to set up a left-handed mouse by choosing Pointer from the Session Manager window Customize... menu.

Creating a DECterm Window

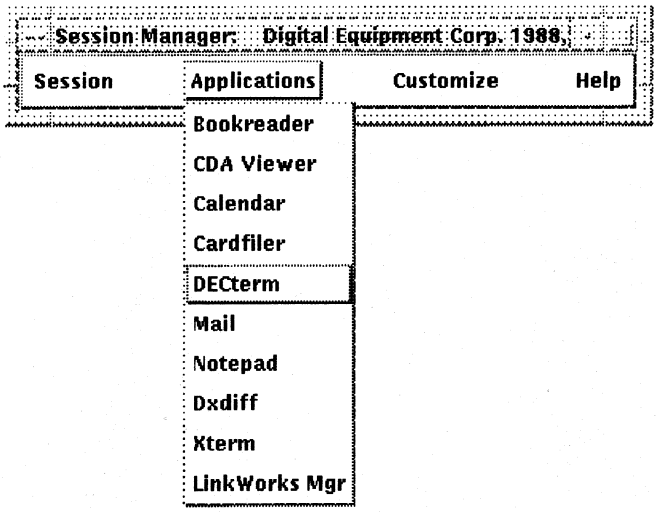
Follow the steps shown in the table to create a DECterm window.

Table 2–2 Creating a DECterm Window

Step	Action
1	Move the mouse pointer to any visible portion of the Session Manager window.
2	Click mouse button MB1. The Session Manager window pops on top of any other windows.
3	Click and hold MB1 on the Applications menu (Figure 2–1). While holding MB1, drag the highlighted box down the menu and select the DECterm application by releasing MB1 on DECterm. The Application menu disappears.
4	Wait for a DECterm window to appear (Figure 2–2).
5	To activate the DECterm window, click MB1 anywhere in the window. The window borders change and the cursor blinks where your input appears.

Once the DECterm window starts, you can enter commands as you would on a terminal.

Figure 2–1 Creating a DECterm Window

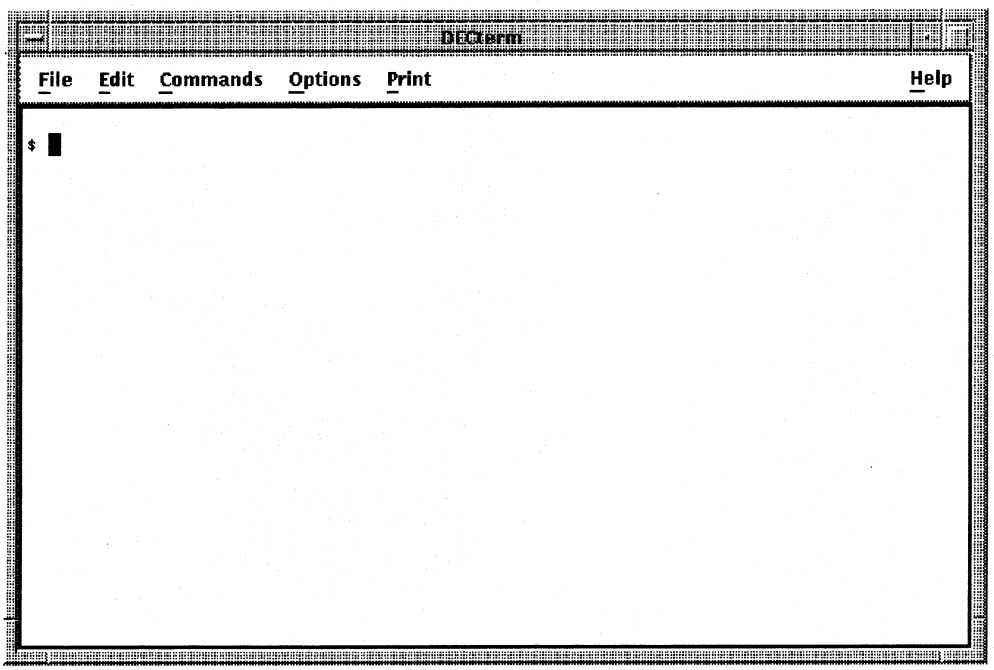


ZKOX-055000058-11-RAGS

DECterm Window

The figure shows the resulting DECterm window.

Figure 2–2 A DECterm Window



ZKOX-055000058-12-RAGS

DECterm Window Menus

You can open a VT100 emulation terminal window by opening a DECterm window from the Applications menu.

In addition to DEC OSF/1 commands, you can communicate with other systems and share information and resources if your system is part of a network.

You can start multiple DECterm windows depending upon your system resources. Each window is a separate login process running on your system.

The title bar of your DECterm window displays menus that let you set up your personal DECterm environment, as shown in the following table.

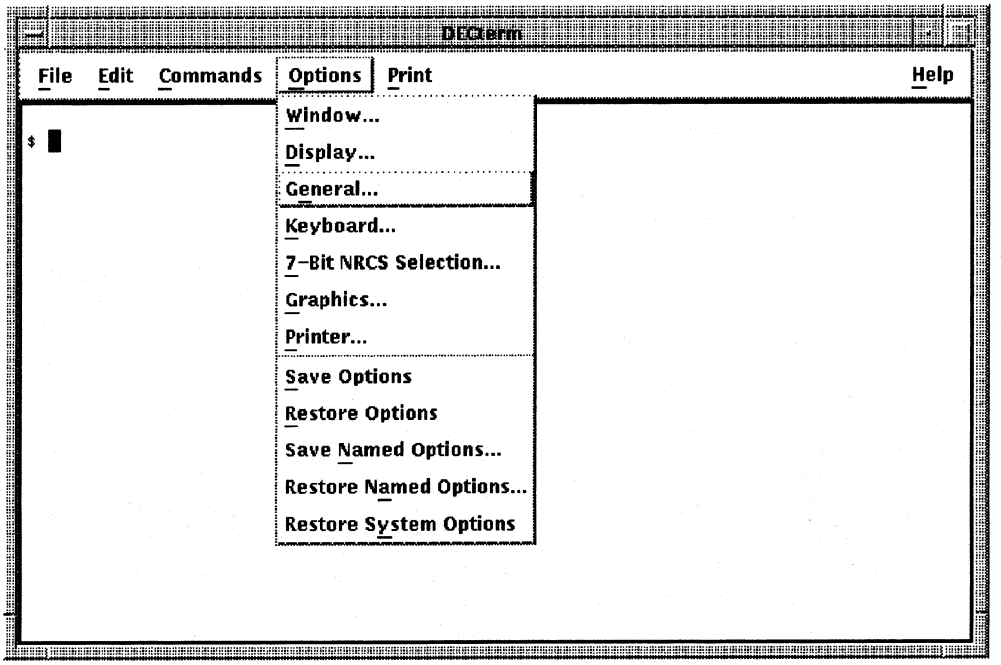
Table 2–3 DECterm Window Menu Functions

Menu	Function
File	Exit the DECterm window
Edit	Select, copy, and paste text
Commands	Choose a secure keyboard Clears terminal screen Resizes window Clears the communication lines to terminal Resets terminal
Options	Allows you to customize terminal window size, font, title, rows, columns, and icon name Customizes display lines, scrolling, color or contrast, and cursor Customizes general window characteristics to a VT100, VT200, or VT300 series terminal style and keypad, cursor keys, and bit features for each Customizes keyboard keys for various applications Customizes graphics options for various applications Customizes printing applications Saves and restores options
Print	Prints text page, selected text, all text, graphics, and finishes or cancels printing
Help	Provides help on: Context, Window, Help, and Version

Using the DECterm Options Menu

Choose Keyboard... from the Options menu to set keyboard functions. Choose Save Options after you have made your choices. Choose General... from the Options menu (Figure 2–4) to select terminal type and keypad functions.

Figure 2–3 DECterm Options Menu

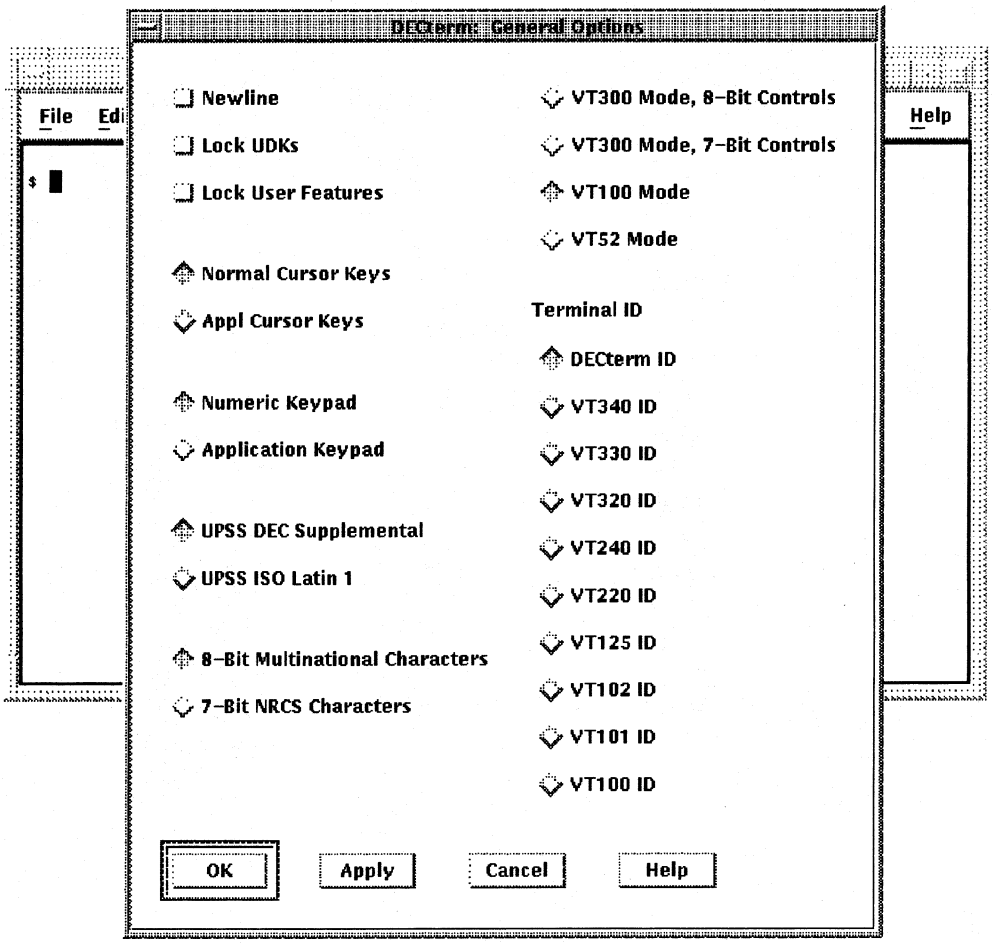


ZKOX-055000058-14-RAGS

DECterm General Options Menu

After you choose General... from the Options menu, you can set options as shown in the figure.

Figure 2–4 DECterm General Options Menu



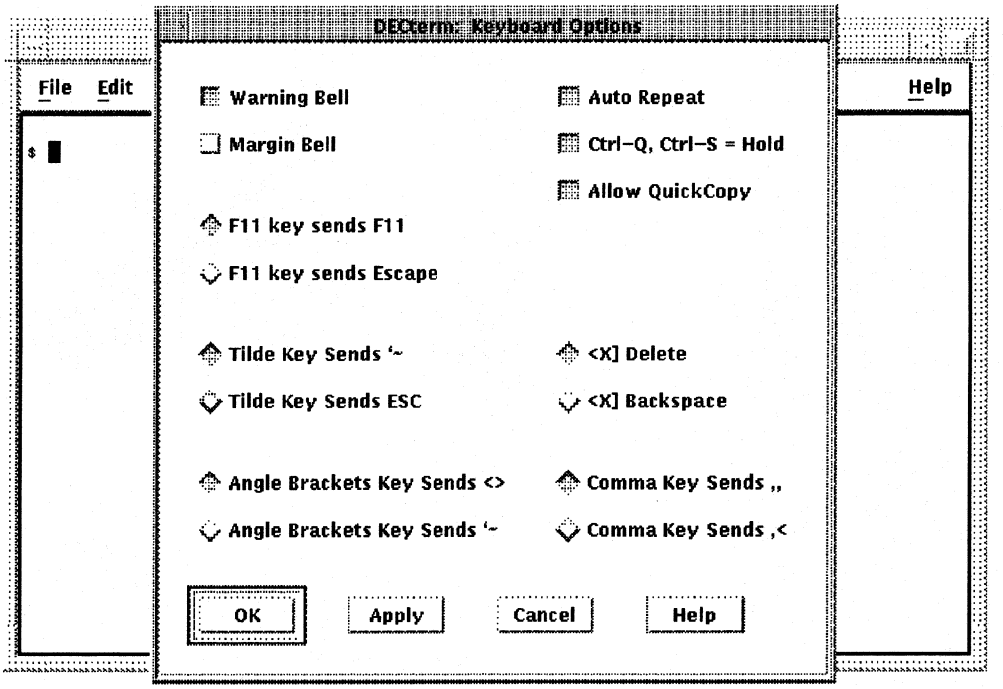
ZKOX-055000058-15-RAGS

DECterm Keyboard Options Menu

There are two methods of specifying keyboard settings for a DECterm window:

- Use the `stty` command arguments, as previously introduced in the Terminal Setup section, to display and set keyboard key and terminal functions.
- Choose Keyboard... from the DECterm window Options menu. A Keyboard Options window appears, as shown in the figure. This lets you choose your keyboard settings quickly and easily. Be sure to save those settings by choosing Save Options from the Options menu.

Figure 2–5 DECterm Keyboard Options Menu

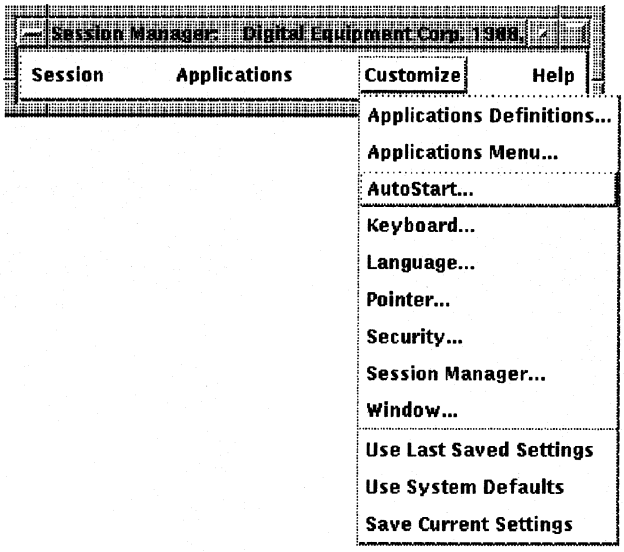


ZKOX-055000058-16-RAGS

Automatically Starting Applications

The figure shows how to use AutoStart... to set up your workstation to automatically start applications each time you log in.

Figure 2–6 Session Manager Customize Menu



ZKOX–055000058–17–RAGS

Customizing Your Session Manager

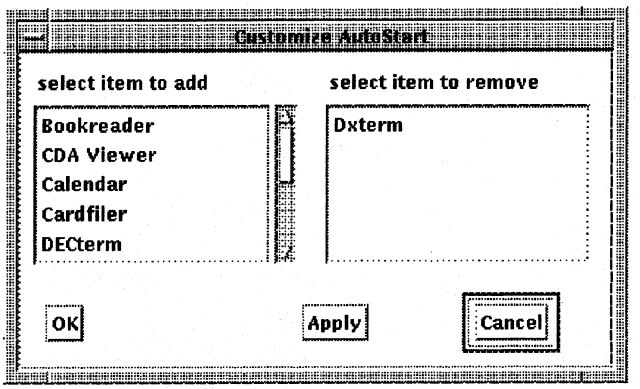
Follow the steps shown in the table to AutoStart your DECterm window.

Table 2–4 Autostarting a DECterm Window

Step	Action
1	Move the mouse to any visible portion of the Session Manager window.
2	Click mouse button MB1. The Session Manager window pops on top of any other windows.
3	Choose AutoStart... from the Customize menu.
4	In the select item to add list, scroll down using the vertical slider. Click MB1 on DECterm.
5	Click MB1 on OK to leave the Customize AutoStart menu.

A DECterm window starts automatically each time you log in, as shown in the figure.

Figure 2–7 Session Manager Customize Autostart



ZKOX–055000058–18–RAGS

Learning Activity: Workstation Setup

This exercise provides practice in setting up the DEC OSF/1 workstation windows environment.

Getting Started

1. Log in to DEC OSF/1 using your login name.
2. Click mouse button MB1 outside of the Session Manager window. Choose Workspace from the Options menu.
3. Click MB1 on Select The Window. Choose to make a window active when you move the mouse pointer into it. Otherwise, you must click MB1 in the window you want to make active.
4. Choose to show feedback when sizing windows.
5. Click MB1 on [OK].

Setting Up an Autostart DECterm Window

1. Click MB1 on the Session Manager Customize... menu, and drag the pointer to choose AutoStart... .
2. To cause a DECterm window to start up automatically each time you log in, add DECterm to the box on the right side of the Customize AutoStart window. Later you can try adding as many DECterm and other applications as you want.
   - Use MB1 to move the vertical slider in the left box. Notice the various items you can add to AutoStart (the right box).
   - Click MB1 on DECterm, then click MB1 on Apply.
   - For now, add only one DECterm window, so click MB1 on [OK].
3. Quit the session.
4. Log in to DEC OSF/1 again and notice the DECterm window displays automatically.
5. It is common practice to start a session with several applications displayed on your screen as icons. For example, you could have quick access to Mail, Calendar, or Bookreader by simply double-clicking MB1 on the icon.

Customizing a DECterm Window

1. Click MB1 on the icon symbol (or dot) at the upper right of your DECterm window to iconify it. Click MB1 on the icon, and drag the icon around on your screen. Double-click MB1 on the icon to restore it.
2. Rename your DECterm window and icon. Hint: Use the Window... choice of your DECterm window Options menu.
3. Try other custom settings, and change the window size and position on the screen.
4. Choose Save Options from the DECterm window Options menu.

5. Choose Save Current Settings from the Session Manager window Customize menu.
6. Quit the session, and select [Yes] to save settings.
7. Log in again to verify your custom settings were saved.

No solution required.

Summary

Terminal Setup

- Use the `stty -a` command to display your terminal and keyboard settings.
- Press the `interrupt` key (usually Ctrl/C) to stop a process, or to regain control in an interactive program.
- Use `stty` command arguments in your `.profile` file to automatically set up your default terminal and keyboard settings each time you log in.
- Use the `man stty` command to learn more about `stty` command arguments.

Workstation Setup

- Choose Save Options from the Options menu to reuse your current terminal settings each time you log in to your workstation.
- Choose the AutoStart application from the Customize menu to automatically display a DECterm window each time you log in to your workstation.

3

Invoking Motif Applications

Invoking Motif Applications

Introduction

The Motif graphical user interface used in the DEC OSF/1 operating system is an enhanced version of the X Window System graphical user interface used with many UNIX operating systems.

With Motif, you can execute applications by using the mouse to choose from menus and list boxes, rather than by typing commands.

Microsoft Windows and Apple Macintosh are other graphical user interfaces that are menu-driven.

Objectives

To use Motif applications, you should be able to:

- Describe, at the basic level, the use of:
  - Bookreader
  - CDA Viewer
- Use the Motif calendar application to keep track of daily meetings and appointments

Resources

- *OSF/1 User's Guide*
- *ULTRIX to DEC OSF/1 Migration Guide*
- *OSF/Motif User's Guide*

Desktop Applications

Overview

A number of desktop applications are provided by DEC OSF/1. Choose applications from the Applications... menu in the Session Manager window. You can move any of the application windows to a new position, or shrink them to icons.

For additional user applications and commands information, refer to the *ULTRIX to DEC OSF/1 Migration Guide*. The following table lists the DEC OSF/1 desktop applications and their functions.

Table 3–1 DEC OSF/1 Desktop Applications

Application	Function
Bookreader	Provides fast, easy access to online documentation and courses in CDROM online documentation libraries.
Calendar	Displays an electronic calendar allowing you to view daily, weekly, monthly, and yearly displays. You can also set alarms to remind you of appointments.
Cardfiler	Lets you create an electronic card file. Two windows allow you to edit and organize cards alphabetically or by groups.
CDA Viewer[1]	The Compound Document Architecture (CDA) Viewer lets you display DDIF and PostScript (PS) input file formats on your workstation.
Dxdiff	Displays the differences between two text files.
DECterm	Opens a Motif based terminal window. Use the window menu items to edit, customize, or shrink the window to an icon.
Mail	Provides a full-function, menu-driven electronic mail service. Read, answer, forward, delete, and print mail. Create, edit, deliver, and file new mail into electronic folders.
Notepad	A mouse-driven text editor that lets you record messages, write letters, or build data files. Notepad saves your notes in files, which you can store for later revision or incorporation into other files.
Xterm	Opens a Motif based X window system without editing and customizing features.

[1]The default file format is DDIF

Using man to Learn About Desktop Applications

The table below lists the man command to enter for help on specific applications.

For help on	Enter
Bookreader	$ man dxbook
Calendar	$ man dxcalendar
Cardfiler	$ man dxcardfiler
CDA Viewer	$ man dxvdoc
Dxdiff	$ man dxdiff
DECterm	$ man dxterm
LinkWorks Mgr	$ man lwkmanager

Mail	$ man mail
	$ man mailx
	$ man mhmail

Notepad	$ man dxnotepad
Xterm	$ man xterm

Using Bookreader

Bookreader provides fast, easy access to online documentation. Use the mouse to select books from an online bookshelf. Read pages of text and view graphics in any order.

Bookreader features include random access to topics via in-text cross-reference, a table of contents and an index. View large graphics or tables in their own pop-up windows for quick reference beside the related text.

Bookreader requires system managers to set up electronic "bookshelves."

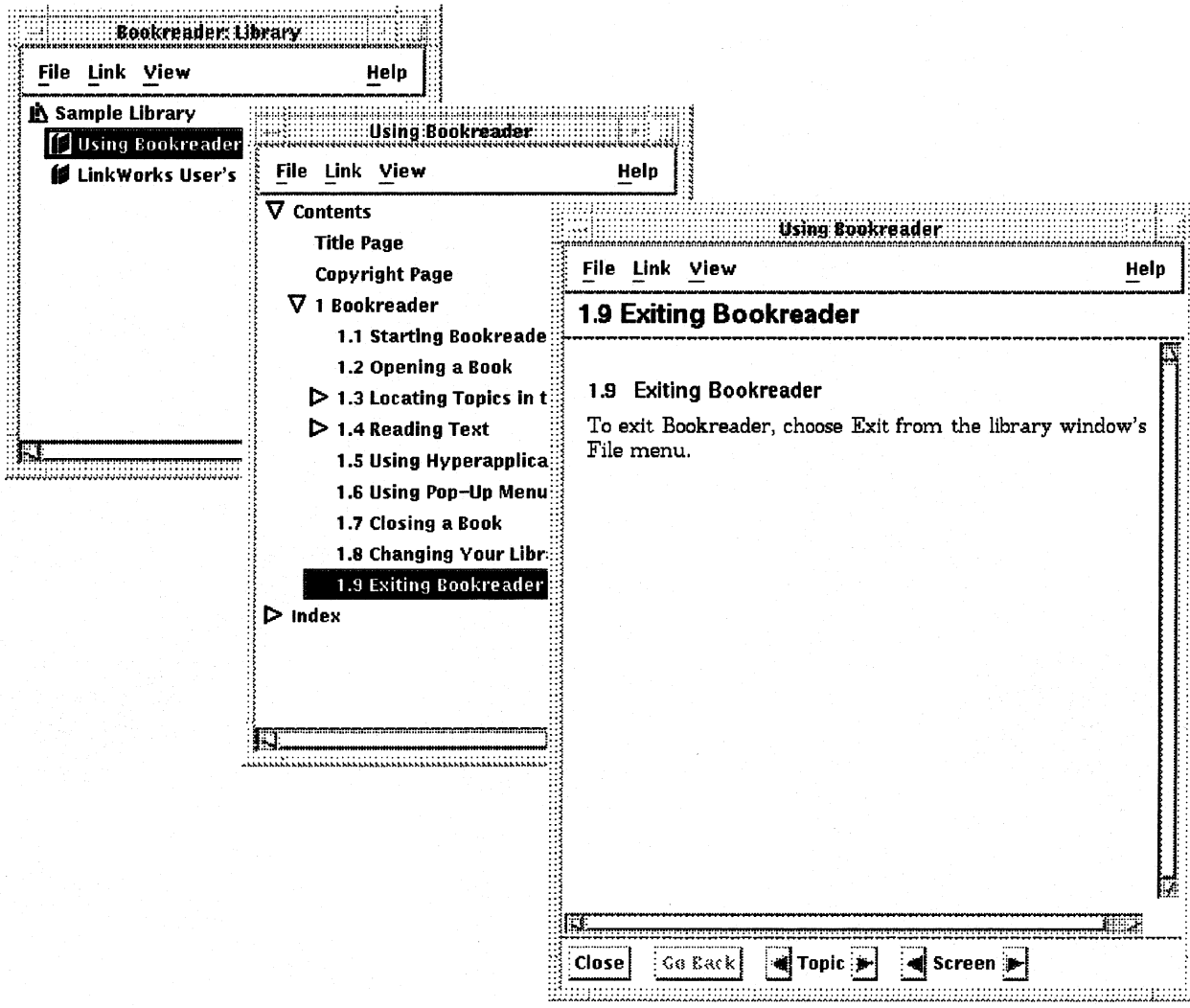
A bookshelf is a listing of books. When you start Bookreader, a selection window contains a list of books (a bookshelf), a list of more bookshelves, or both. To read a book, open it from a bookshelf.

Bookshelves usually group related books to make the process of finding a particular book easier.

Use Bookreader with CDROM online documentation libraries to save storage space and the cost of hardcopy books.

The figure shows a sample Bookreader session.

Figure 3–1 A Bookreader Session



Starting Bookreader

To start Bookreader:

1. Choose Bookreader from the Applications menu in the Session Manager window.
2. Click MB1 on a book title to select it.
3. Click MB1 on TOC or Index to choose the access format you prefer.
4. In the Bookreader window, use the screen commands to navigate backward and forward through the document.

Using CDA Viewer

Use the CDA Viewer to view:

- Digital Document Interchange Format (DDIF) files
- PostScript files

You can also use the CDA Viewer as a previewer to save unnecessary printing.

Starting CDA Viewer

To start using CDA Viewer:

1. Click MB1 on the Session Manager window Applications menu, drag the pointer to CDA Viewer and release MB1.
2. Select a DDIF file from the Open window Files in the pathname box, and click MB1 on OK. The DDIF file image displays after a short wait.
3. Use the click-and-drag method on the vertical and horizontal sliders to view additional portions of the image.
4. To exit CDA Viewer, select Quit from the File menu.

The above steps can be used for the PS Viewer option by substituting PS for DDIF.

Learning Activity: Desktop Applications

Determine which desktop applications are installed on your system. Experiment with those that are of interest to you. The Calendar utility is described more fully in the next section.

There is no specific solution to this Learning Activity.

The Calendar Application

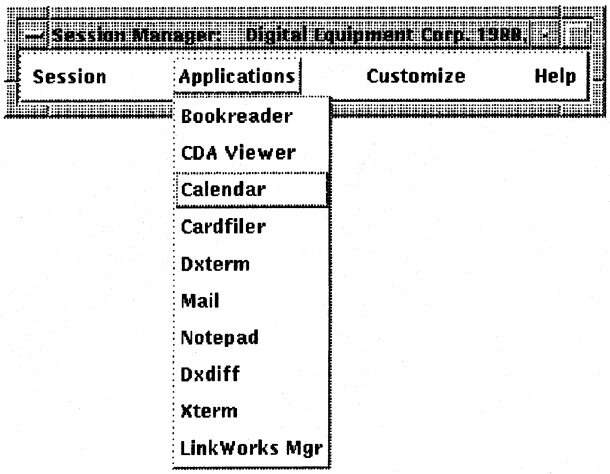
Using the Calendar Application

The calendar application provides an automated calendar and appointment book for the users. Keep track of meetings and plan your time, look at yearly or monthly displays or review your scheduled appointments one day at a time. You can have more than one calendar and use each calendar for a different purpose.

Starting the Calendar Application

Choose the Calendar option from the Applications menu of the Session Manager, as shown in the figure.

Figure 3–2 Choosing the calendar Application



ZKOX–055000058–39–RAGS

The first time you run the calendar program (and any time the calendar program cannot find a data file), calendar displays a warning window stating that it could not open the calendar data file, and asks if it should create a new one. (If this is the first time you have run the program, the answer is yes.)

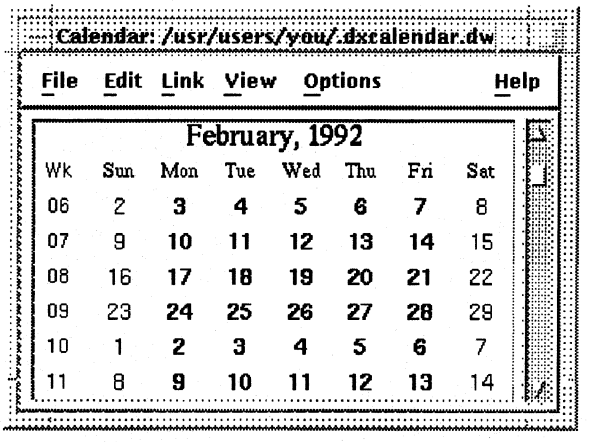
About the Default Calendar

The first time you run calendar, it displays the default screen shown in Figure 3–3 with the current day highlighted.

Features of this screen include:

- Help menu to direct new users
- Scroll bar on right to scroll through the months and years
- Five pull-down menus for calendar features
- Other variations of default screens available under the Options menu

Figure 3–3 Default Calendar



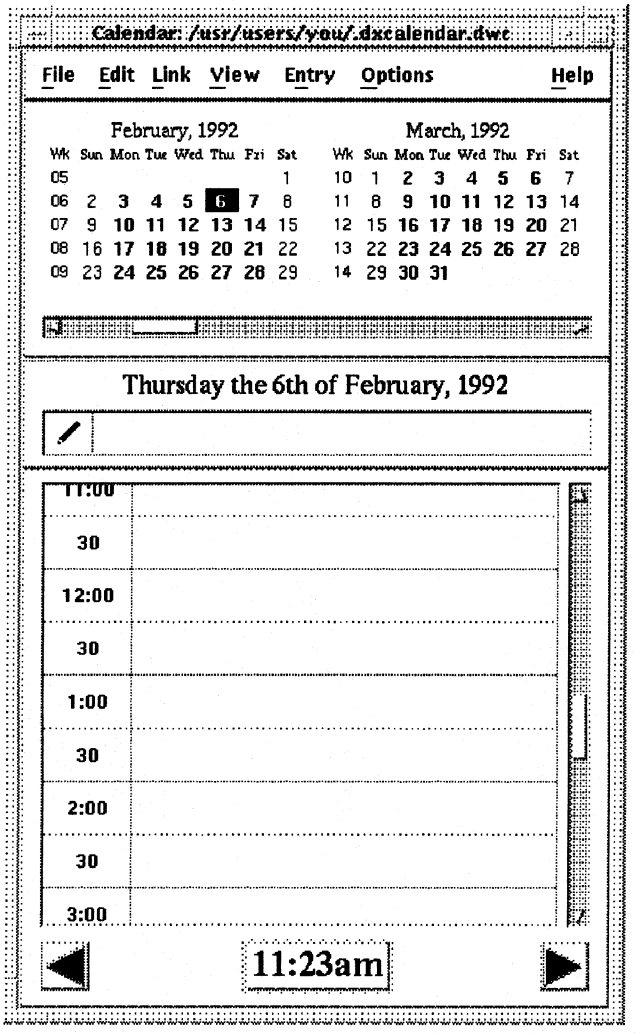
ZKOX-055000058-40-RAGS

Selecting the Day

There are two ways to select the day from the default calendar.

- Double-clicking on the highlighted day using MB1
- Selecting Day from the View menu

Figure 3–4 Calendar by Day Screen



ZKOX-055000058-43-RAGS

Creating a Calendar Entry

Once you select the day and you see the calendar by day screen, you can create or add an entry in the calendar.

To do this:

1. Double-click on the time of the meeting or appointment you are entering.

   The Entry Editor window will appear as shown in Figure 3–5.

2. Position your mouse in the box provided for entering information about the appointment and press MB1.
3. Enter your appointment information.
4. Optional steps:
   - When you have entered all information, you can select a time to be notified of your appointment. You have the option of being notified at the time your appointment

begins, five minutes before, or one half hour before. Choose the time by clicking MB1 on any of those options.

- You can also select any of the icons provided to remind you what the appointment is for. Clicking on any of these icons places them to the left of the meeting description. Click again to remove an icon from meeting a description.

5. When you finish adding an entry, click on OK.

When you finish using the calendar, exit calendar by choosing Close from the File menu.

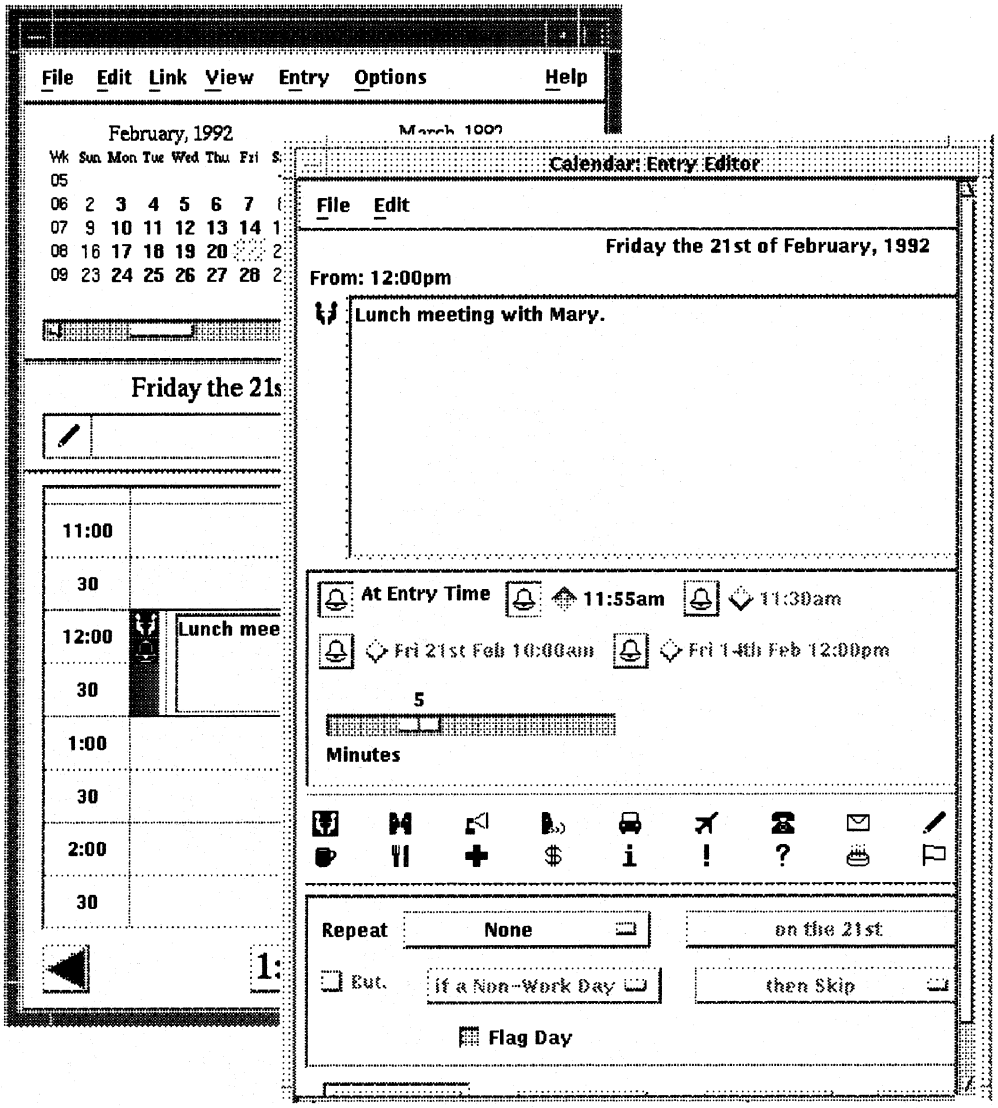
Help is provided with calendar. For additional information on the calendar utility, enter:

```
$ man dxcalendar
```

Using the calendar Entry Editor

The figure shows a calendar entry editor window.

Figure 3–5 Entry Editor Window



ZKOX-055000058-38-RAGS

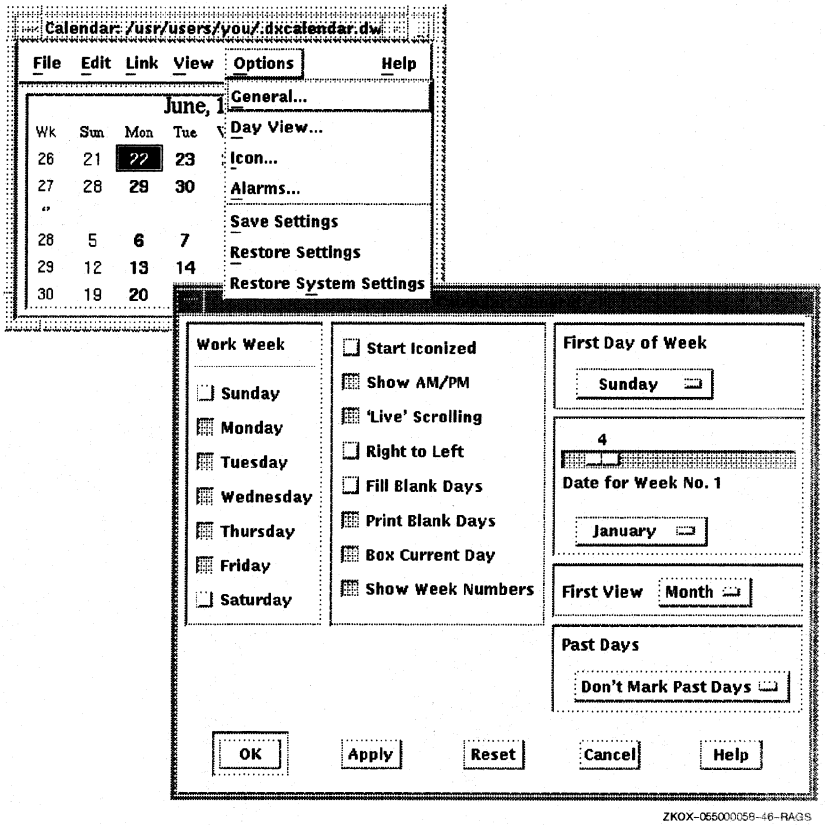
Customizing the Calendar Display

Choose General from the Options menu of the default calendar to customize the general display of the calendar, as shown in Figure 3–6.

The screen allows you to set the following features:

- Work days of the week
- First day of the week
- Consecutive numbering of weeks
- Past days
- Fill or print blank days
- Box current day
- Show week numbers

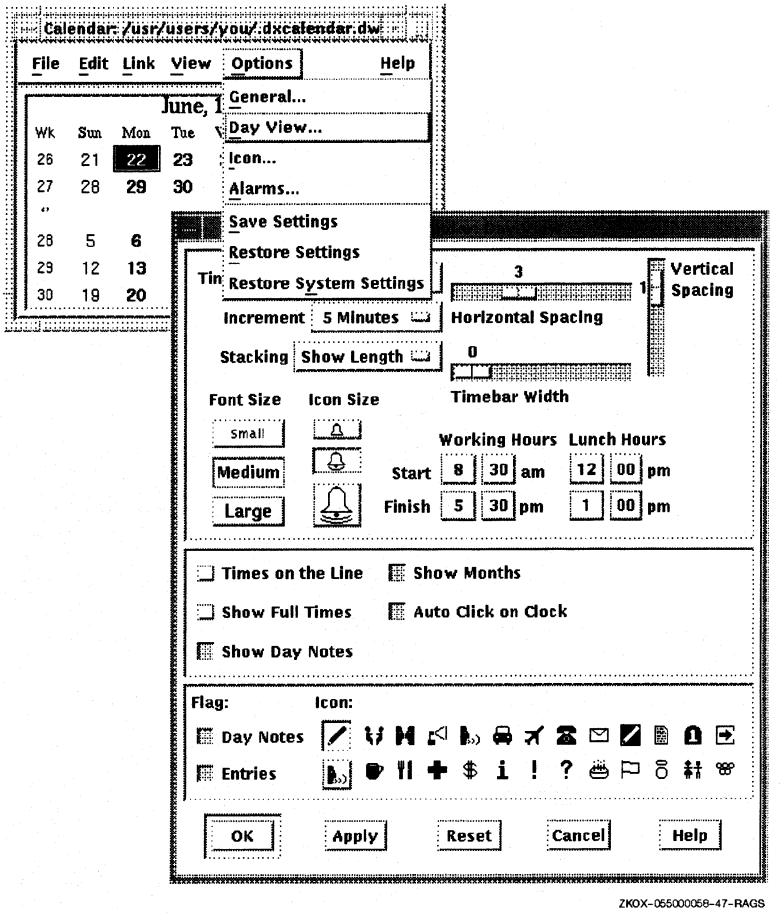
Figure 3–6 Default (General) Screen



Customizing the Day View Defaults

You can customize the display of the calendar by day by choosing Day View from the Options menu. When you do this, a Day View screen appears, as shown in the figure.

Figure 3–7 Day View Default Screen



ZKOX-055000056-47-RAGS

Calendar Day View

The Calendar Day View screen allows you to customize how the calendar looks:

- Vertical and horizontal spacing
- Time slot size
- Default icons for day notes and entries
- Working hours
- Lunch hours
- Font and icon size

Learning Activity: Using the Calendar Utility

Run the calendar utility and explore its many features. Try setting general defaults and adding appointments to a particular day.

No solution required.

Summary

Desktop Applications

The DEC OSF/1 Desktop Applications use the OSF/Motif windowing interface. These mouse-driven applications can be selected from the Session Manager window Applications menu.

Desktop applications include:

- Bookreader gives you fast and easy access to online documentation.
- Calendar provides an electronic calendar with daily, weekly, monthly, and yearly displays for scheduling dates. You can also set alarms to remind you in advance of appointments.
- Cardfiler lets you create an electronic box of index cards.
- CDA Viewer lets you display DDIF and PS (PostScript) input file formats on your workstation.
- Dxdiff displays the differences between two text files.
- DECterm opens a Motif-based terminal window.
- Mail gives you a full-function, menu-driven electronic mail service.
- Notepad is an easy-to-use, mouse-driven text editor that lets you do everyday writing tasks.

The Motif Calendar Application

The Motif Calendar application provides a visual display of your appointments calendar.

4

File System Concepts

File System Concepts

Introduction

In the UNIX operating system, files are the key to organizing data and accessing it. UNIX systems use a hierarchical file system. Since a file is a named collection of information, the information can be:

- A program
- Data for a program
- A directory
- A device special file

In the UNIX file system, files and directories are often grouped by function.

To efficiently work with a UNIX system, you must understand how the UNIX file system is organized and the different types of files that can be created. You should know limitations regarding naming of files and directories and determine the type of information a file contains.

Objectives

To understand UNIX file system concepts, you should be able to:

- Define file types
- Determine the type of a file with the `file` command
- Describe file and directory naming conventions
- Describe the UNIX directory tree structure

Resources

- *OSF/1 User's Guide*

File Types

Overview

The UNIX file system uses three types of files:

- Ordinary disk files
- Directories
- Special files

Ordinary Disk Files

An **ordinary** (or regular) file contains whatever information you place in it. The system does not require an ordinary file to have a particular structure. It can be a file of text consisting of a string of characters. Some files must have a particular structure or format to be used by a program. But it is the program, not the system, that controls the structure.

Hidden Files

There is a subtype of ordinary files called **hidden** files. The names of these files always begin with a dot (.) and each may have a special function. For example, `.profile` is a hidden file that defines the user's shell environment at login time. Users can customize this file to their preference. Hidden files are discussed later in the course with the `ls` command.

Directories

Directories provide the connection between the names of files and the files themselves. Each directory stores the names and inode numbers of the files contained in it. **inode** numbers are pointers used to locate information on a device. Directories impose a structure on the file system as a whole. Information is stored in files, files are grouped into directories, and directories are grouped into higher-level directories.

Special Files

Special files are files used by the system. One type of special file is a file that represents a device. For example, each device (line printer, disk drive, terminal) has a file associated with it that represents that device. These files are contained in the directory `/dev`.

Other special files are sockets and named pipes.

Learning Activity: File Types

1. What are the three types of files used by UNIX system?
2. What is the purpose of directories?
3. What type of file is a **hidden** file?

Learning Activity Solution: File Types

1. What are the three types of files that a UNIX system uses?
   - Ordinary files
   - Directories
   - Special files
2. What is the purpose of directories?

   Directories provide structure to the file system. Directories enable you to group similar files together.
3. What type of file is a **hidden** file?

   A hidden file is a type of ordinary file that always begins with a dot. For example, `.profile` is a hidden file.

Determining File Type

Use the `file` command to determine the type of contents contained in a particular file. It provides information on whether a file contains ASCII text, or programming text, or is a directory or special file.

The format for the `file` command is:

file *filename*.

This command displays the file name and the type of information it contains. Example 4–1 shows the output of some `file` commands.

Note

In the following example, an extra Return was entered between commands to separate the commands.

Example 4–1 Using the file Command

```
$ file bin
bin:            directory
$
$ file myfile
myfile:         ascii text
$
$ file nosuch
file: Cannot get file status on nosuch.
nosuch: cannot open for reading
$
$ file compute_average.c
compute_average.c:      c program text
$
$ file /dev/tty
/dev/tty:       character special (1/0)
$
```

The `file` command actually examines the file contents to determine the type.

Some file descriptions shown above might be too technical. For now, you only need to know the purpose of using the `file` command.

Learning Activity: Using the file Command

Using the `file` command, determine the type of contents each file listed below contains.

- `.profile`
- `/usr`
- `/`

Learning Activity Solution: Using the file Command

Using the `file` command, determine the type of contents each file listed below contains.

- `.profile`

  ```
  $ file .profile
  .profile:       commands text
  ```

- `/usr`

  ```
  $ file /usr
  /usr:   directory
  ```

- `/`

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

Naming Files and Directories

Overview

Think of a file as a document stored in a file cabinet. You can find a document in a file cabinet if you know what folder the document is in and what it is called. When using the UNIX operating system, you create directories as electronic folders in which to store your files. You give names to your files when you create them. Therefore, you can find a file by its name and directory.

Naming Conventions

The naming conventions for files and directories are the same. The name of a file or directory is its **file name** and is used whenever you want to refer to the file in a command.

File and directory names can include letters, numbers, periods (.), and underscores (_).

A period in a file or directory name has no special meaning to the system, except when it is the first character. If a file has a period as the first character, the file is considered **hidden**.

The maximum length is system dependent, and is set up when the operating system kernel is built. File and directory names can contain up to 255 characters for BSD systems, but as few as 14 characters for some SVID systems.

Acceptable File Names

The following file names are acceptable:

- `.login`
- `.profile`
- `junk_file`
- `file2`
- `FILE2`
- `compute_average.c`
- `compute_average.out`
- `letter.txt`

Naming Guidelines

Keep these guidelines in mind when naming files:

- Avoid beginning your file or directory names with a period (.) unless you want it to be hidden.
- Upper and lowercase letters are distinct (HELLO, Hello, and hello are three different file or directory names). That is, file names are case sensitive.
- Avoid characters with special programming or system meanings, such as:

Special Character	Comment
/	Reserved for root directory
*	Used for wildcard substitution
?	Used for single character substitution
[ ]	Used for selective file name substitution
<	Indicates less than or redirects input
>	Indicates greater than or redirects output
-	Indicates options
$	Indicates argument substitution
'	Turns off special meaning
"	Groups characters into a single argument
&	Performs command in background
!	Used for history substitution

You can access files with special characters in the name by escaping the special character with a backslash, or quoting the entire name to the shell.

- Do not use spaces or periods (.) and (..) alone for file or directory names; these are reserved for system files.
- The UNIX operating system does not retain old versions of a file if it is updated, although some applications do.
- You can use part of the file name as a file extension preceded by a period (.) to indicate the contents of the file.

File Name	File Contents
`prog.c`	Source code for a C language program
`prog.out`	Executable file
`constitution.txt`	Text file
`prog.dat`	Data file

File extensions are meaningful to some applications. For example, the C compiler will try to compile a file with the `.c` extension, but will skip the compile step and try to link a file with a `.o` extension. The UNIX operating system allows you to give a file of your own either of these extensions without regard to its contents.

UNIX Directory Structure

Overview

In the UNIX file system, files and directories are often grouped by function. Users can control their own files and directories. System managers control and manage system files and directories.

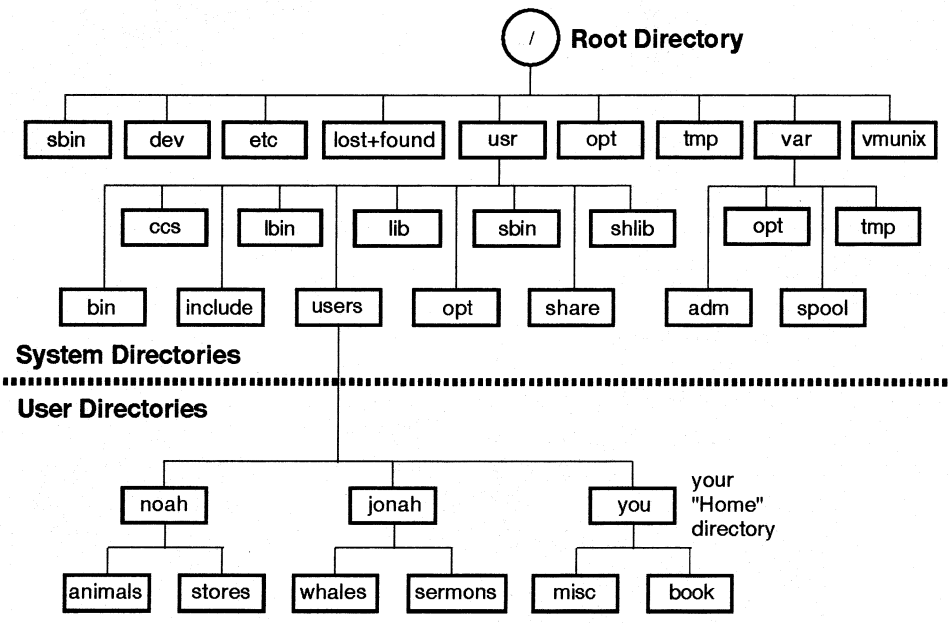
Hierarchical File System

All information stored in the file system hierarchy stems from the **root directory** (/). Directly under the root are the first-level directories. Under the first-level directories are further levels of subdirectories.

Think of the file system as the root structure of a tree. The top level is the root. All other directories and files branch from it in a widening structure. This is called a **hierarchical** file system. All information is divided into logical groups. Therefore, the hierarchical file system provides a way to organize the vast amount of information stored in the computer.

The figure shows a portion of a representative UNIX file system. You do not need to learn what all of these directories are for.

Figure 4–1 Hierarchical File System Structure



ZKO-055-000058-51-RGS

Note the root directory, which is always represented by a slash (/), and how all other directories stem from it, making the structure hierarchical. Note also that the home directory is typically a user's name, but throughout this course it is represented by you.

Remember that there are more directories and subdirectories than represented here.

Summary

File Types

There are three types of files in the UNIX file system:

- Ordinary files
- Directories
- Special files

The `file` command allows you to distinguish the type of information of a file.

Naming Files and Directories

Follow these guidelines when naming files:

- Use letters, numbers, underscores, and periods within file names.
- Avoid beginning your file or directory names with underscores (_) or periods (.).
- Upper and lowercase letters are distinct.
- Avoid characters with special programming or system meanings.
- The UNIX operating system does not retain multiple versions of files by default, but some applications do.
- You can use part of the file name as a file extension preceded by a period (.) to indicate the contents of the file.

UNIX Directory Structure

The UNIX file system is hierarchical. All files stem from the root directory, represented by the slash symbol, (/). Grouping files and directories according to function allows system files and directories to be protected.

Directories provide the connection between the names of files and the files themselves.

5

Directory and File Access

Introduction

As you create and add files to your directory, it becomes increasingly useful to organize the files into other directories and subdirectories. To work in such an environment, you must be able to access other subdirectories and the files in them. Also, as more and more files are created, it becomes harder to remember exactly where files are, and what their exact file names are.

You need to be able to change your directory and determine the names of the files that are in the new directory. You must be able to find files anywhere on the system, and display the contents of files.

This module discusses methods to:

- Change directories
- List the files in a directory
- Locate files based upon file characteristics, such as file name, owner, group, or file size
- Display the contents of a file

Objectives

To access the contents of a file or directory, you should be able to:

- Use full and relative pathnames
- Determine the current working directory using the `pwd` command
- Change directories using the `cd` command
- List files and directories using the `ls` command
- Find files and directories using the `find` command
- View files, or portions of files, using the `cat`, `more`, `head`, and `tail` commands

Resources

- *OSF/1 User's Guide*

Changing Directories

Overview

Because the UNIX file structure is hierarchical, all files or directories are accessed by tracing a path through a chain of directories until the desired one is reached.

To understand how to change directories, you must first understand the following concepts:

- Home and working directories
- Pathnames

Home Directory

Each user has a directory of files called a **home** directory. This directory is usually identified by a **pathname** that includes your username as its last component. Your home directory is assigned when your account is created, and can be found in the `/etc/passwd` file. When you log in to your account on a UNIX operating system, you are automatically placed in your home directory.

Working Directory

Your **working** directory is defined as your current location in the directory tree. If you are in your home directory, that is also your working directory. However, if you move to another directory, that becomes your working directory. Even though your working directory may change, your home directory always remains the same.

You may, at times, forget which directory you are working in. Use the `pwd` command (**p**rint **w**orking **d**irectory) to display your current location.

The format of the `pwd` command is:

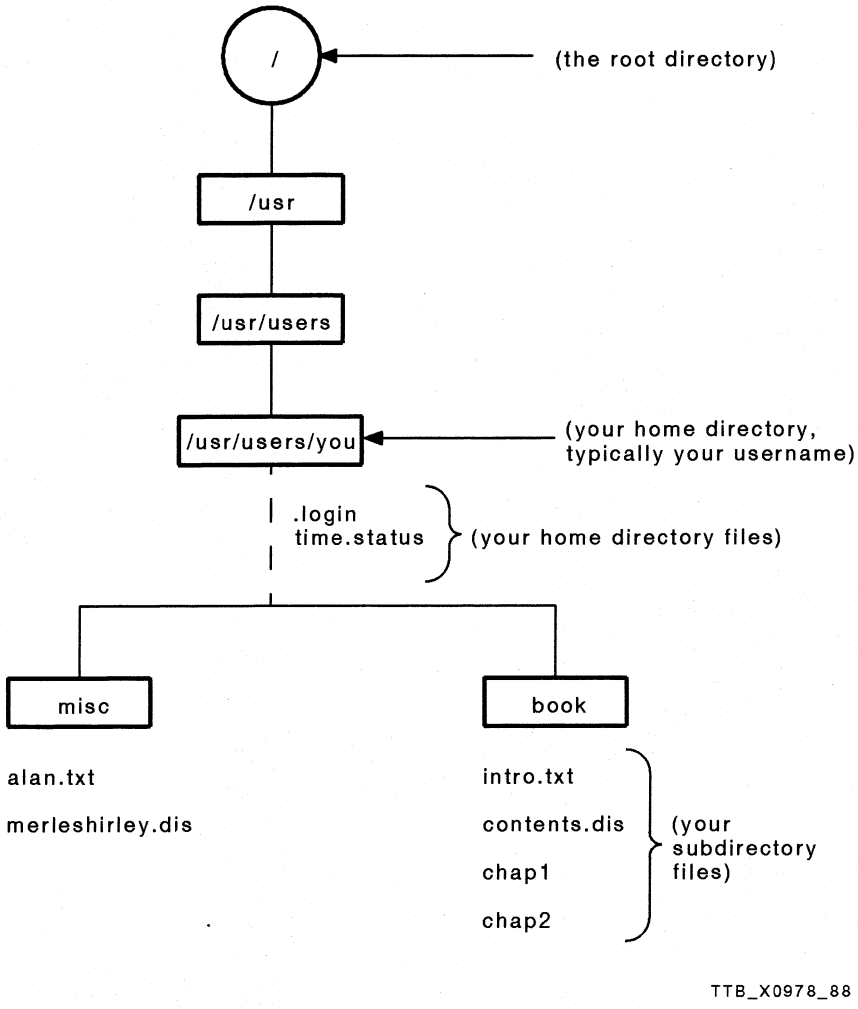
```
$ pwd
/usr/users/you
```

Home Versus Working Directory

Figure 5–1 represents a sample structure of a user's directory and files. Suppose the user is **you**. There are two files in your home directory: `.login` and `time.status`. To better organize your files, you created two subdirectories: `misc` and `book`. These subdirectories contain additional files.

When you log in, you are automatically placed in your home directory: `/usr/users/you`. At this point, your home and working directory are the same. If you change directories to `/usr/users/you/book`, your working directory becomes `/usr/users/you/book`. However, your home directory stays the same.

Figure 5–1 Home and Working Directories



Pathnames

A **pathname** represents the directories and subdirectories that are passed through to reach a specified file or directory. A pathname refers to a file or directory in the hierarchy. It represents the path of a directory from a specific point in the directory structure. There are two types of pathnames: **full** pathnames and **relative** pathnames.

Full Pathnames

A full pathname indicates the path of a directory starting from the **root** (/). For example, if you have a subdirectory in your home directory called `book`, containing a file called `chap1`, the full pathname for the file `chap1` is `/usr/users/you/book/chap1`.

Full pathnames always begin at the root (/) directory. Using full pathnames allows you to change or refer to directories and files regardless of your current location.

Full pathnames are also known as absolute pathnames.

Relative Pathnames

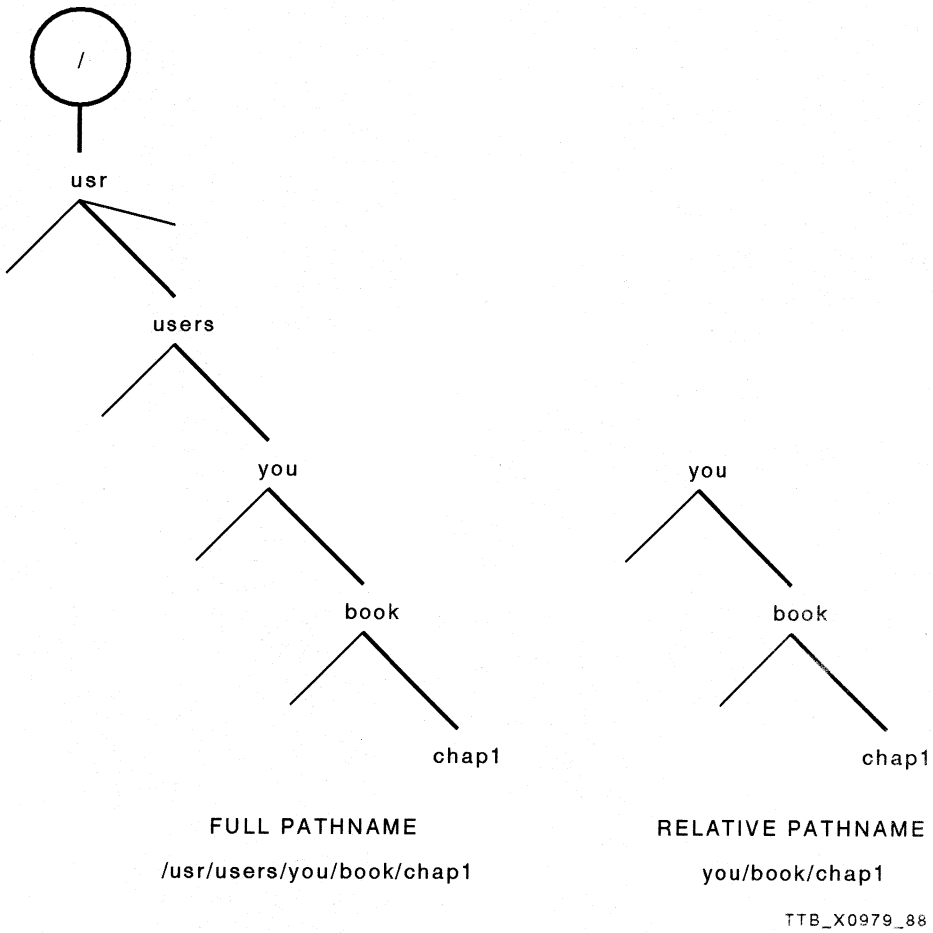
Relative pathnames are based on your current working directory. For example, if your current working directory is `/usr/users/you`, and you have a subdirectory called `misc` containing a file named `alan.txt`, the relative pathname of the file is `misc/alan.txt`. In contrast, the full pathname for that same file is `/usr/users/you /misc/alan.txt`.

Relative pathnames always begin from your current location. They are "relative" to where you are in the directory structure.

Full Versus Relative Pathnames

Figure 5–2 illustrates full and relative pathnames for accessing a file named `chap1` if the working directory is `/usr/users`.

Figure 5–2 Full and Relative Pathnames



The cd Command

To change directories, use the `cd` (**c**hange **d**irectory) command followed by the pathname of the directory you want to change to. The format for the `cd` command is:

`cd` *`pathname`*

Using Special Pointer Files . and ..

There are two special pointer files that refer to specific directories.

- Dot (.) refers to the current directory.
- Dot dot (..) refers to the **parent** directory. That is, the directory one level above the current directory.

Since . (dot) refers to your current directory, entering `cd .` will not change your working directory. Also, entering `cd ..` will not change your working directory, if your working directory is the root directory.

Table 5–1 lists some examples and functions of the `cd` command.

Table 5–1 Using the cd Command

Example	Changes working directory to:
`$ cd`	Home directory
`$ cd book`	Subdirectory `book`
`$ cd /usr/users/misc`	Directory `/usr/users/misc`
`$ cd ..`	Parent directory
`$ cd ../..`	Two directories above current directory

Example 5–1 shows the use of the .. (dot dot) file to change directories from `/usr/users/stu1` to `/usr/users/stu2`

Example 5–1 Using .. with the cd Command

```
$ pwd
/usr/users/stu1
$ cd ../stu2
$ pwd
/usr/users/stu2
```

Learning Activity: Changing Directories

If at any point in this exercise you are not sure what your current directory is, use the `pwd` command to find out.

1. The `/etc/passwd` file contains the encrypted passwords of all users on your system, and the real-life name that corresponds to each user's login name. The `cat` command will display the contents of a file on your terminal. The format is `cat` *`filename`*
   .

   Try to display the file `/etc/passwd` on your terminal screen in each of the following ways:

   a. Using the full pathname from your home directory.

   b. Changing your working directory to /, and using a relative pathname.

2. Try entering `$ cat passwd` while your working directory is /. It will not work because you did not give the system a valid pathname.

3. Change your working directory to /etc. Now the command $ **cat passwd** will display the file.
4. Now, change back to your home directory and verify it.

Learning Activity Solution: Changing Directories

If at any point in this exercise you are not sure what your current directory is, use the pwd command to find out.

1. The /etc/passwd file contains the encrypted passwords of all users on your system, and the real-life name that corresponds to each user's login name. The cat command will display the contents of a file on your terminal. The format is cat *filename*.

   Try to display the file /etc/passwd on your terminal screen in each of the following ways:

   a. Using the full pathname from your home directory.

   ```
   $ cat /etc/passwd
   root:PrFX.BclhWLHc:0:1:system PRIVILEGED account:/:
   nobody:*Nologin:65534:65534:anonymous NFS user:/:
       .
       .
       .
   ```

   b. Changing your working directory to /, and using a relative pathname.

   ```
   $ cd /
   $ cat etc/passwd
   root:PrFX.BclhWLHc:0:1:system PRIVILEGED account:/:
   nobody:*Nologin:65534:65534:anonymous NFS user:/:
       .
       .
       .
   ```

2. Try entering $ **cat passwd** while your working directory is /. It will not work because you did not give the system a valid pathname.

   ```
   $ cat passwd
   cat: cannot open passwd
   ```

3. Change your working directory to /etc. Now the command $ **cat passwd** will display the file.

   $ **cd etc** OR **cd /etc**

   ```
   $ cat passwd
   root:PrFX.BclhWLHc:0:1:system PRIVILEGED account:/:
   nobody:*Nologin:65534:65534:anonymous NFS user:/:
       .
       .
       .
   ```

4. Now, change back to your home directory and verify it.

   ```
   $ cd
   $ pwd
   /usr/users/you
   ```

Listing Files

Overview

Suppose you forget the name of a file. Fortunately, the system keeps track of all of your files. You can list the names of all the files in a directory by using the `ls` command. The `ls` command is derived from the word **list**. The format of the `ls` command is:

```
ls [options] [pathname].
```

The ls Command

Table 5–2 gives some examples of `ls` commands and their functions. The examples in this table coincide with the directories shown in Figure 5–1. This table assumes that the working directory is `/usr/users/you`.

Table 5–2 ls Command Examples

Example	Function
`$ ls`	If no pathname is specified, the `ls` command displays a list of files in the working directory, `/usr/users/you`.
`$ ls /usr/users/you/book`	Displays a list of files in the directory `/usr/users/you/book` using a full pathname.
`$ ls book`	Displays a list of files in the directory `/usr/users/you/book` using a relative pathname.
`$ ls .`	Using the special symbol . (dot) with the `ls` command also displays a list of files in the working directory.
`$ ls ..`	Using the special symbol .. (dot dot) with the `ls` command displays a list of files in the parent directory.

If the pathname specifies an ordinary file and not a directory, that filename is listed.

Using the ls Command

The callouts are keyed to Example 5–2, which shows the use of the `ls` command.

❶ Displays the current working directory

❷ Changes the working directory to `/usr/users/you/book` using a relative pathname

❸ Lists the files found in the directory `/usr/users/you/book`

❹ Lists the files in the directory one level above `/usr/users/you/book`

❺ Changes the working directory to `/usr/users/you`

❻ Lists the files found in the current working directory and in the /usr/users/you/book directory. The names of the two directories are separated by spaces

Example 5–2 Using the ls Command

```
$ pwd          ❶
/usr/users/you
$ cd book      ❷
$ ls           ❸
chap1          chap2          contents.dis   intro.txt
$ ls ..        ❹
book           misc           time.status
$ cd           ❺
$ ls . book    ❻
.:
book           misc           time.status

book:
chap1          chap2          contents.dis   intro.txt
$
```

Using the ls Command with Options

A UNIX operating system keeps track of a lot of information, some of which is actually stored in your directories. There are several options that can be used with the ls command to obtain this information. Depending on the option(s) that is used, different information may be supplied with the directory listing. Some of the common options are:

- -a

  Lists all files in directory including those that begin with a period. Files with names that begin with a period are called hidden files.

- -F

  In addition to marking directories with a trailing slash (/), this option also marks symbolic links with a trailing at sign (@), and executable files with a trailing asterisk (*). Symbolic links are covered later in the course.

- -l

  This option provides a long listing of the files in a directory. This includes information such as file type, access rights, owner, group, size, and date of creation or modification.

For more information on other options, use the man ls command.

Using the -a Option

Every directory contains the two files . and .. that help the system keep track of the current and parent directories. However, when you list your files with the `ls` command, the . and .. do not show up on the list, nor does your `.profile` file. This is because file names beginning with a period are invisible to, or **hidden** from, the `ls` command. To list **all** of the files in a directory, including the hidden ones, enter: **`ls -a`**

```
$ pwd
/usr/users/you
$ ls -a
.                       book                    short_cut
..                      compute_average.c       time.status
.cshrc                  compute_average.out
.profile                misc
$
```

Using the -F and -a Options

You can use more than one option at a time. The example below uses both the `-a` and `-F` options to list all the files in a directory as well as marking directories with a slash and executable files with an asterisk (*).

```
$ ls -aF /usr/users/you
./                      book/                   short_cut
../                     compute_average.c       time.status
.cshrc*                 compute_average.out*
.profile*               misc/

$
```

Using the -l Option

The `-l` option provides a long listing of the regular (not hidden) files in a directory. This long listing includes information on file type, access rights, owner, group, size, and date of creation or modification.

Note

You must have execute permission on the directory to get a long listing.

The example below shows the use of the `ls` command with the `-l` option.

```
$ ls -l
total 6
drwxr-xr-x   2 you   users        512 Nov 05 09:38 book
-rw-r--r--   1 you   users         88 Sep 03 11:41 compute_average.c
-rwxr-xr-x   1 you   users        120 Sep 04 14:06 compute_average.out
drwxr-xr-x   2 you   users        512 Nov  5 11:16 misc
-rw-r--r--   1 you   users        512 Apr 07 14:07 short_cut
-rw-r--r--   1 you   users        178 Nov  5 09:30 time.status
```

To include hidden files with this long listing, use both the `-l` and `-a` options.

```
$ ls -al
total 11
drwxr-x--x   5 you   users        512 Nov  5 11:04 .
drwxr-xr-x  26 root  system       512 Oct 18 15:41 ..
-rwxr-x--x   1 you   users        373 Jun 29 15:22 .cshrc
-rwxr-x--x   1 you   users        145 Jun 29 15:22 .profile
drwxr-xr-x   2 you   users        512 Nov 05 09:38 book
-rw-r--r--   1 you   users         88 Sep 03 11:41 compute_average.c
-rwxr-xr-x   1 you   users        120 Sep 04 14:06 compute_average.out
drwxr-xr-x   2 you   users        512 Nov  5 11:16 misc
-rw-r--r--   1 you   users        512 Apr 07 14:07 short_cut
-rw-r--r--   1 you   users        178 Nov  5 09:30 time.status
```

Field Descriptions of Long List

The table below describes each field in a long listing. As a sample, the table uses the entry:

```
-rwxrw-r--   1 you   users         88 Sep  3 10:30 letters
```

Field	Description
-	Indicates that this file is an regular file. A `d` in this field indicates that the file is a directory.
`rwxrw-r--`	Indicates file access permissions for the file. More details about file protection are presented later in this course.
`1`	Indicates the number of links to the file. There is only `1` link to this file.
`you`	Indicates the owner of the file. The owner of this file is `you`.
`users`	Indicates the group that the file is assigned to for access purposes, in this case, the group `users`.
`88`	Indicates the file size in bytes. This file is 88 bytes.
`Sep`	Indicates the month the file was last accessed.
`3`	Indicates the day of the month the file was last accessed.
`10:30`	Indicates the time the file was last accessed. This file was last accessed September 3rd, at 10:30.
`letters`	Indicates the file name.

Learning Activity: Listing Files

1. List the files in your home directory using the `ls` command.
2. List all files in both your home directory and its parent directory using the special pointer files `.` (dot) and `..` (dot dot).
3. Change your working directory to `/` and, with one command, list the files in the `/bin` and `/etc` directories.

4. Change your working directory to /usr/bin using the cd command and list the files in the /usr/bin directory.
5. Without changing your working directory, list the files in the directory /usr and your home directory.
6. Now change back to your home directory.
7. Display a listing of **all** files in your home directory, the / directory, and the /usr directory.

   Notice that the files . and .. appear in every directory.
8. Display a long listing of all the files in your home directory.
9. Obtain a listing of all files in your home directory that indicates which files, if any, are subdirectories.

Learning Activity Solution: Listing Files

1. List the files in your home directory using the ls command.

   ```
   $ ls
   ```
2. List all files in both your home directory and its parent directory using the special pointer files . (dot) and .. (dot dot).

   ```
   $ ls . ..
   ```
3. Change your working directory to / and, with one command, list the files in the /bin and /etc directories.

   ```
   $ cd /
   $ ls bin etc
   ```
4. Change your working directory to /usr/bin using the cd command and list the files in the /usr/bin directory.

   ```
   $ cd /usr/bin
   $ ls
   ```
5. Without changing your working directory, list the files in the directory /usr and your home directory.

   If your home directory is, for example, /usr/users/you, you could use:

   ```
   $ ls /usr /usr/users/you
   ```

   OR

   ```
   $ ls .. ../users/you
   ```
6. Now change back to your home directory.

   ```
   $ cd
   ```
7. Display a listing of **all** files in your home directory, the / directory, and the /usr directory.

   Notice that the files . and .. appear in every directory.

   ```
   $ ls -a . / /usr
   ```
8. Display a long listing of all the files in your home directory.

   ```
   $ ls -al
   ```

9. Obtain a listing of all files in your home directory that indicates which files, if any, are subdirectories.

```
$ ls -aF
```

Finding Files

Overview

There are times when you need to locate files on the system, but you do not know all of the information necessary to locate them. For example, you may know the name of a file, but not the directory where it is located. You could go to every directory listing files until you find it. The `whereis` and `find` commands are useful in these situations.

Locating Files on the System

The `whereis` command is used to locate program files stored in standard libraries. It locates the directory that contains the source, binary and `man` page files for a program. With no options, `whereis` looks in the default directories for all three types of files.

The format for the `whereis` command is:

```
whereis [-bmsu] [-BMS directory ...  -f] program
```

whereis Command Options

Options to the `whereis` command are shown in Table 5–3.

Table 5–3 whereis Command Options

Option	Function
`-b`	Searches for an executable program
`-m`	Searches for `man` pages for a program
`-s`	Searches for the sources of a program
`-u`	Used with other flags to indicate that there is no file for the program in the default or specified directory or directories (source, binary, or reference page)
`-B` directory	Specifies binary directory to search - use full pathname
`-M` directory	Specifies reference directory to search - use full pathname
`-S` directory	Specifies source directory to search - use full pathname
`-f`	Delimits a `-B`, `-M`, or `-S` list

Using the whereis Command

Example 5–3 Using the whereis Command

```
$ whereis emacs          ❶
emacs: /usr/bin/emacs /usr/lib/emacs /usr/share/man/man1/emacs.1
$ whereis -b emacs       ❷
emacs: /usr/bin/emacs /usr/lib/emacs
$ whereis -m emacs       ❸
emacs: /usr/share/man/man1/emacs.1
$ whereis -s emacs       ❹
emacs:
$
```

❶ Searches for files named `emacs` in the default source, binary, and `man` page directories.

❷ Searches for `emacs` executable programs.

❸ Searches for `emacs man` pages.

❹ Searches for `emacs` source files. None are found on this system.

The find Command

The `find` command searches specified directory trees for files which meet particular criteria.

The format for the `find` command is:

`find` *`pathname expression`*

where:

`pathname` is the directory tree to search from
`expression` is the specified criteria to search for

Using the find Command

You can use the `find` command to search the entire directory structure for all files named `chap` using the following command:

```
$ find / -name chap -print
```

In the example shown above, `/` is the pathname to search from (which is the root directory) and `-name chap` is the expression that indicates to search for files named `chap`. `-print` tells the system to display any matching file names on your screen.

In the next example, the directory `/etc` is the pathname to search from and `-name passwd` is the expression to search for. This example searches from the directory `/etc` for any files named `passwd` and displays them on the screen.

```
$ find /etc -name passwd -print
```

NOTE

Only those directories which you have read access to will be searched.

By default, the search begins at the `pathname` you specify, and searches all directories and subdirectories beneath

it. If you use the -prune expression, only the pathname specified is searched.

The find Command Expressions

You do not have to search for files based on their file names. There are several expressions that you can use with the find command. For example, you can search for files that have a particular owner, block size, or modification date.

Some common find command expressions are:

- -print

  The -print expression displays the file names matching specified criteria on your terminal screen. Unless you use this expression with the find command, the file names are **not** displayed.

  ```
  $ find . -name chap -print
  ```

  In this example, the search begins at the current directory, finds all the files named chap, and displays those file names on the screen.

- -name

  This expression tells the system to find files based on their file names.

  ```
  $ find /usr/users -name chap -print
  ```

  This example searches from the directory /usr/users for any files named chap and displays the names on the screen.

  The -name expression searches for files that match the arguments you supply. Suppose you want to find all files whose file names begin with the word chap. You can use pattern-matching characters to search for partial file names. If you use these characters, you must precede them with the backslash character (\) or enclose the string in quotes. A couple of these characters are:

  - Asterisk (*) (also called, splat) — substitution for anything. Using chap*, or "chap*" in the example will find all files that begin with chap.
  - Question mark (?) — substitution for a single character. Using chap\? or "chap?" in the example will find all files that begin with chap and have only one more character. For example, it would find chap1, but not chap12.

- -ls

  The -ls expression is similar to -print in that it displays the matching file names on the screen. However, this long listing gives additional information about the files. It is similar to using the ls -l command.

  ```
  $ find . -name chap -ls
  ```

In this example, the search begins at the current directory, finds all the files named chap, and displays those file names on the screen in the form of a long listing.

- -user

  This expression finds files based on their owner.

  ```
  $ find / -user jones -print
  ```

  This example searches from the root directory for all files that are owned by user jones and displays those file names on the screen.

- -group

  This expression finds files based on their group.

  ```
  $ find /usr -group student -ls
  ```

  This example searches from the /usr directory for all files that are owned by any user and the group student, and displays those file names on the screen in the form of a long list.

- -size

  The -size expression finds files based on size.

  ```
  $ find . -size 1 -user jones -print
  ```

  This example searches from the current directory for all files that are owned by user jones **and** are 1 block (512 bytes) in size, then displays those file names on the screen.

  The -size expression is in blocks. A block is 512 bytes. You can find files based on size in bytes or kilobytes by adding a c (for bytes) or a k (for kilobytes) after the number. You can also specify plus (+) or minus (-) before a size greater or less than the specified number. For example, `find / -size +5` will find all files with sizes greater than 5 blocks. The `find / -size +5k` command will find all files greater than 5 kilobytes.

- -atime

  The -atime expression finds files based on when they were accessed.

  ```
  $ find / -atime 7 -ls
  ```

  This example searches from the root directory for all files that were accessed within 7 days and lists them. Use +7 to find files that were accessed more than 7 days ago.

Learning Activity: Finding Files

In your home directory, use the find command to:

1. Search from the /etc directory for files named motd and display those file names on the terminal.
2. Search the /usr/users directory for all files owned by you and display their file names. (Note: You receive error messages for subdirectories under /usr/users that you do not have access to.)

3. Search from the /dev directory for file names that have the characters tt and display those file names in the form of a long listing.

Learning Activity Solution: Finding Files

In your home directory, use the find command to:

1. Search from the /etc directory for files named motd and display those file names on the terminal.

   ```
   $ find /etc -name motd -print
   ```

2. Search the /usr/users directory for all files owned by you and display their file names. (Note: You receive error messages for subdirectories under /usr/users that you do not have access to.)

   ```
   $ find /usr/users -user you -print
   ```

3. Search from the /dev directory for file names that have the characters tt and display those file names in the form of a long listing.

   ```
   $ find /dev -name \*tt\* -ls
   ```

   OR

   ```
   $ find /dev -name "*tt*" -ls
   ```

Viewing Files

Overview

To view a file is to display its contents on your terminal screen. With a UNIX operating system, there are many ways to view files and portions of files.

Several commands that provide this functionality are:

- cat
- more
- head
- tail

Viewing Entire Files

The cat command displays the entire contents of a file on your screen, with the first lines scrolling off the screen if the entire file won't fit. The format for the cat command is:

cat *filename*

Example 5–4 uses the cat command to view a file named phones.

Example 5–4 Viewing a File Using the cat Command

```
$ cat phones
Maynard MA (508)493-4123
Nashua NH (603)884-2413
Cambridge MA (617)654-9531
Springfield MA (413)747-3452
Stamford CT (203)258-3232
Burlington MA (617)221-3895
Merrimack NH (602)884-8230
Novi MI (313)344-1839
Kalamazoo MI (616)384-8310
Littleton MA (508)952-4839
Augusta ME (207)271-2211
Lancaster, MA (508)874-6021
Burlington VT (802)657-2895
Charleston SC (803)744-9046
Greenville SC (803)288-3627
Philadelphia PA (215)246-7028
Rochester NY (716)385-8240
Syracuse NY (315)453-1057
$
```

The cat command is covered in more depth later in this course.

Viewing Portions of Files

When you use the cat command to view the contents of a file, the entire file is displayed on your screen immediately, leaving the only the last screenful of the file on your screen. However, there may be times when you only want to see certain portions of files. The head and tail commands provide this functionality.

The head Command

By default, the head command displays the first 10 lines of file. You can specify how many lines of the file you want to display. Only a negative number can be specified with this command. This indicates that starting from the top of the file, the specified number of lines will be displayed.

Example 5–5 displays the first six lines of the file named phones.

Example 5–5 Viewing Part of File Using head

```
$ head -6 phones
Maynard MA (508)493-4123
Nashua NH (603)884-2413
Cambridge MA (617)654-9531
Springfield MA (413)747-3452
Stamford CT (203)258-3232
Burlington MA (617)221-3895
$
```

The tail Command

By default, the tail command displays the last 10 lines of file. As with the head command, you may specify how many lines of the file you want to display. However, with the tail command, you can specify either a negative or positive number. A negative (-) number indicates that starting from the bottom of a file, the specified number of lines will be displayed. A positive (+) number indicates that starting from the top of the file, all lines will be displayed **except** the specified number.

For example, tail +50 /usr/lab/longtext displays the file longtext, omitting the first 50 lines.

Example 5–6 displays the last four lines of the phones file.

Example 5–6 Viewing Part of File Using tail

```
$ tail -4 phones
Greenville SC (803)288-3627
Philadelphia PA (215)246-7028
Rochester NY (716)385-8240
Syracuse NY (315)453-1057
$
```

Viewing a Screenful of a File

The more command is used to view the contents of a file one screenful at a time starting at the beginning of a file. After each screen is displayed, the name of the file and the percentage of the file displayed appears at the bottom of your screen, and the system waits for your response.

Valid responses to the `filename (nn%)` prompt are:

- Press Return to display another line.
- Type a number and press Return to display that many more lines. When you type the number, it does not display on your terminal.
- Press Spacebar to display the next screen.
- Press B to display the previous screen.
- Press Q to finish viewing the file and return to the shell prompt.
- Press H to display additional options.

Using the more Command

Example 5–7 shows the use of the `more` command. This example displays the file named `phones`. Note that this file will fit on one screen, but for longer files, the filename will be displayed on the bottom of the screen along with the percentage of the file that has been viewed.

At the end of file prompt, Spacebar will return you the shell prompt, and the other responses behave normally.

Example 5–7 Viewing a File Using more

```
$ more phones
Maynard MA (508)493-4123
Nashua NH (603)884-2413
Cambridge MA (617)654-9531
Springfield MA (413)747-3452
Stamford CT (203)258-3232
Burlington MA (617)221-3895
Merrimack NH (602)884-8230
Novi MI (313)344-1839
Kalamazoo MI (616)384-8310
Littleton MA (508)952-4839
Augusta ME (207)271-2211
Lancaster, MA (508)874-6021
Burlington VT (802)657-2895
Charleston SC (803)744-9046
Greenville SC (803)288-3627
Philadelphia PA (215)246-7028
Rochester NY (716)385-8240
Syracuse NY (315)453-1057
phones: END
```

File Display Examples

Table 5–4 lists some examples of the `head`, `tail` and `more` commands and their functions.

Table 5–4 Viewing Files Using the head, tail, and more Commands

Command Example	Displays the:
`head /usr/lab/longtext`	First 10 lines of the file `longtext`
`head -6 /usr/lab/longtext`	First 6 lines of the file `longtext`
`tail /usr/lab/longtext`	Last 10 lines of the file `longtext`
`tail -4 /usr/lab/longtext`	Last 4 lines of the file `longtext`
`more /usr/lab/longtext`	First screen of the file `longtext`, and waits for your response

Learning Activity: Viewing Files

1. From your home directory, list the files in the `/etc` directory. Look for a file named `passwd`.
2. Display the entire contents of the file named `passwd` on your terminal.
3. Display the entire contents of the file named `passwd` again. This time, display the contents one screenful at a time.
4. Using the `head` and `tail` commands, display the first six and last nine lines of the `passwd` file.

Learning Activity Solution: Viewing Files

1. From your home directory, list the files in the `/etc` directory. Look for a file named `passwd`.

   ```
   $ ls /etc
   ```

2. Display the entire contents of the file named `passwd` on your terminal.

   ```
   $ cat /etc/passwd
   ```

3. Display the entire contents of the file named `passwd` again. This time, display the contents one screenful at a time.

   ```
   $ more /etc/passwd
   ```

4. Using the `head` and `tail` commands, display the first six and last nine lines of the `passwd` file.

   ```
   $ head -6 /etc/passwd
   $ tail -9 /etc/passwd
   ```

Summary

Changing Directories

A **full** pathname represents the path of a directory from the root directory. A **relative** pathname represents the path of a directory from the current working directory.

To display your current working directory, use the `$ pwd` command.

The table below summarizes the commands used to change directories.

You Enter:	Result
`$ cd`	Changes working directory to home directory using no pathname
`$ cd /usr/book`	Changes working directory to `/usr/book` using a full pathname
`$ cd book`	Changes working directory to subdirectory `book` using a relative pathname
`$ cd ..`	Changes working directory to parent directory using special symbol ..

Listing Files

The following table summarizes the commands and options used to list files.

You Enter:	Result
`$ ls`	Displays a list of file names not beginning with a . in the current working directory
`$ ls -a`	Displays a list of **all** files in the working directory
`$ ls -l`	Displays a list of files in the working directory, with information about each file
`$ ls -F`	Displays a list of files that identifies subdirectories, symbolic links, and executable images symbolically
`$ ls -al`	Displays a list of **all** files in the working directory, with file information

Finding Files

The following table summarizes some of the expressions used to find files.

You Enter:	Result
`$ whereis emacs`	Displays the names of all the files in certain system directories that are named `emacs`.
`$ find / -name test -print`	Displays the file names of all the files in the system that are named `test`.
`$ find /usr -user you -print`	Displays the file names of all files in `/usr` and its subdirectories that are owned by user `you`.
`$ find /etc -size +5 -print`	Displays the file names of all files in `/etc` and its subdirectories that are greater than 5 blocks in size.
`$ find . -atime +7 -ls`	Searches from the current directory for all files that have been accessed more than 7 days ago and displays those file names on the terminal in the form of a long list.

Viewing Files

The following table summarizes the commands used to view files.

You Enter:	Result
`$ cat` *filename*	Entire file is displayed on your terminal screen
`$ more` *filename*	File is displayed one screenful at a time
`$ head` *filename*	First 10 lines of file are displayed
`$ head -n` *filename*	First n lines of file are displayed
`$ tail` *filename*	Last 10 lines of file are displayed
`$ tail -n` *filename*	Last n lines of file are displayed

6

File Management

Managing Files

Introduction

File management involves organizing files into directories and subdirectories, moving files between directories, protecting files, and removing files you no longer need.

This chapter discusses the commands necessary to organize and maintain a collection of files.

Objectives

To manage files, you should be able to:

- Create an empty file or update the file access or modification time using the `touch` command.
- Create and concatenate files using the `cat` command.
- Create directories using the `mkdir` command.
- Change file permissions and group permissions using the `chmod` command.
- Move and rename files using the `mv` command.
- Copy files using the `cp` command.
- Remove files and directories using the `rm` and `rmdir` commands.
- Create links to a file using the `ln` command.

Resources

- *OSF/1 User's Guide*

Creating Files

Overview

There are several ways to create files in a UNIX operating system. One method uses editors such as `vi` and `emacs`. Another method uses the `cat` command to create short files. Or you can create empty files using the `touch` command.

Using the touch Command

Use the `touch` command to create an empty file. It can also be used to update both the access time and the modification time of a file.

The format for the `touch` command is:

```
touch [-acfm] [-r reference_file | -t time] file
```

touch Command Options

When used with no options, `touch` updates both the access and modification times of each file or directory named to the time specified on the command line, or the present time. The options are shown in the following table.

Table 6–1 Commonly Used touch Command Options

Option	Function
`-a`	Changes only the access time
`-c`	Prevents the creation of the file
`-f`	Attempts to force file access in spite of file permissions
`-m`	Changes only the modification time
`-r` *reference-file*	Changes the modification time to that of the reference file
`-t` *time*	Changes the modification time to the specified time in the form: [[CC]YY]MMDDhhmm[.SS]

You cannot use the `-r` and `-t` options together.

The following example demonstrates how to use the `touch` command.

Example 6–1 touch Command Examples

```
$ touch poker.c ❶
$ touch -m -c poker.c ❷
$ touch -a -t 07070707 poker.c ❸
$ touch empty ❹
```

❶ Update the access and modification time of the file `poker.c` to the current date and time.

❷ Update the modification time of `poker.c` to the current date and time. If the file does not exist, do not create it.

❸ Set the access date of `poker.c` to July 7, 7:07 a.m. of the current year.

❹ Create an empty file named `empty`.

Using the cat Command

The `cat` command is most often used to view the contents of file. However, you can also use it to:

- Create files
- Add lines to files
- Combine files by appending the contents of one to another

The `cat` command is derived from the word conCATenate.

Using cat to Create Files

The format for creating a file using the `cat` command is:

`cat > filename`

The > (redirection) symbol indicates that the output of the command be placed in the file following the > symbol. The > symbol can be used with many other commands, and is discussed in more detail later in the course.

To help prevent accidental destruction of files, use the `set noclobber` command at the system prompt or include it in your `.login` or `.profile` file.

The following example shows how to use the `cat` command to create a file called `homephones`.

Example 6–2 Creating Files Using cat

```
$ cat > homephones    ❶
Susan     432-7215 ❷
John      226-1945
Mary      321-1776
Ctrl/D    ❸
```

❶ Enter the command `cat > homephones`.

❷ Enter the text of the file. Press Return at the end of each line. Simple typing mistakes can be corrected using the ⌫ key. However, you cannot move the cursor to a previous line to correct a typing error.

❸ With the cursor positioned on a new line, press Ctrl/D to save the text into a file.

You can now view the contents of that file at any time by entering **cat homephones.**

Using cat to Add Text to Files

The cat command provides a simple way to add text to the bottom of a file. This is done using the >> symbol. The >> symbol indicates that the output of the command should be appended to the file following the >> symbol.

This format of the cat command is:

cat >> *filename*

Both > and >> put the output of a command into the file following the symbol. The table below distinguishes between the two symbols.

Symbol	Example	Function
>	**$ cat file1 > file2**	Creates file2 containing the contents of file1. If a file called file2 already exists, it is overwritten with the contents of file1.
>>	**$ cat file1 >> file2**	Appends the contents of file1 to the existing file called file2.

When you add lines to a file, no file name is specified before the >> symbol. Simply type the information that you want to add to the bottom of the file, then press Ctrl/D.

This example shows how to add a line to the bottom of an existing file named homephones using the cat command.

Example 6–3 Adding Lines to a File Using cat

```
$ cat >> homephones  ❶
Bill     464-8251  ❷
Ctrl/D   ❸
```

❶ Enter the command to append text to the file homephones.

If you use only one > symbol, a new file called homephones is created containing only Bill's name and number. The contents of the original homephones file is overwritten and lost unless noclobber is set.

❷ Enter the line(s) you want to add to the file. Press Return at the end of each line. You can correct simple typing mistakes using the ⌫ key. However, you cannot move the cursor to a previous line to correct a typing error.

❸ Position the cursor on a blank line and press Ctrl/D to save the text into a file.

Now, if you view the contents of the homephones file, it contains the new information.

```
$ cat homephones
Susan     432-7215
John      226-1945
Mary      321-1776
Bill      464-8251
```

Using cat to Append File Contents to Files

You can use the cat command to append the contents of files to other files.

The format of this usage of the cat command is:

cat *filename1* >> *filename2*

When you append the contents of one file to another, a file name is specified before the >> symbol. That file's contents is appended to the file name following the >> symbol. There is no need to enter any text.

This example appends the contents of homephones to an existing file called phones. The contents of the homephones file is added to the end of the file called phones.

Example 6–4 Appending File Contents to Files Using cat

```
$ cat homephones >> phones
$ cat phones
   .
   .
   .
Susan     432-7215
John      226-1945
Mary      321-1776
Bill      464-8251
```

Learning Activity: Creating Files

1. In your home directory, create a file called homephones that includes names and phone numbers.
2. Create a file called fruits with names of 10 different kinds of fruit. Also create a file called vegetables with the names of 10 different kinds of vegetables.
3. Append the fruits file to the vegetables file.
4. Now view the files you have created: homephones, fruits, and vegetables.

Learning Activity Solution: Creating Files

1. In your home directory, create a file called `homephones` that includes names and phone numbers.

```
$ cat > homephones
name  phone-number
name  phone-number
name  phone-number
name  phone-number
name  phone-number
name  phone-number
Ctrl/D
```

2. Create a file called `fruits` with names of 10 different kinds of fruit. Also create a file called `vegetables` with the names of 10 different kinds of vegetables.

```
$ cat > fruits
apples
bananas
cantaloupes
dates
grapes
grapefruit
honeydews
kiwi
peaches
plums
Ctrl/D
$ cat > vegetables
asparagus
brussel sprouts
broccoli
cauliflower
carrots
corn
green beans
spinach
yams
zucchini
Ctrl/D
```

3. Append the `fruits` file to the `vegetables` file.

```
$ cat fruits >> vegetables
```

4. Now view the files you have created: `homephones`, `fruits`, and `vegetables`.

```
$ cat homephones
    .
    .
    .
$ cat fruits
    .
    .
    .
$ cat vegetables
    .
    .
    .
```

OR

```
$ cat homephones fruits vegetables
```

Creating Directories

Overview

When your home directory becomes cluttered, for instance, when you list your files and they fill the screen, you should begin to organize your files into directories.

The mkdir Command

Use the `mkdir` command to create directories.

The format for the command is:

`mkdir` *directory-name*

When creating subdirectories, you must know what your current working directory is. Unless an alternate pathname is given for the new directory, it is automatically a subdirectory of your current working directory.

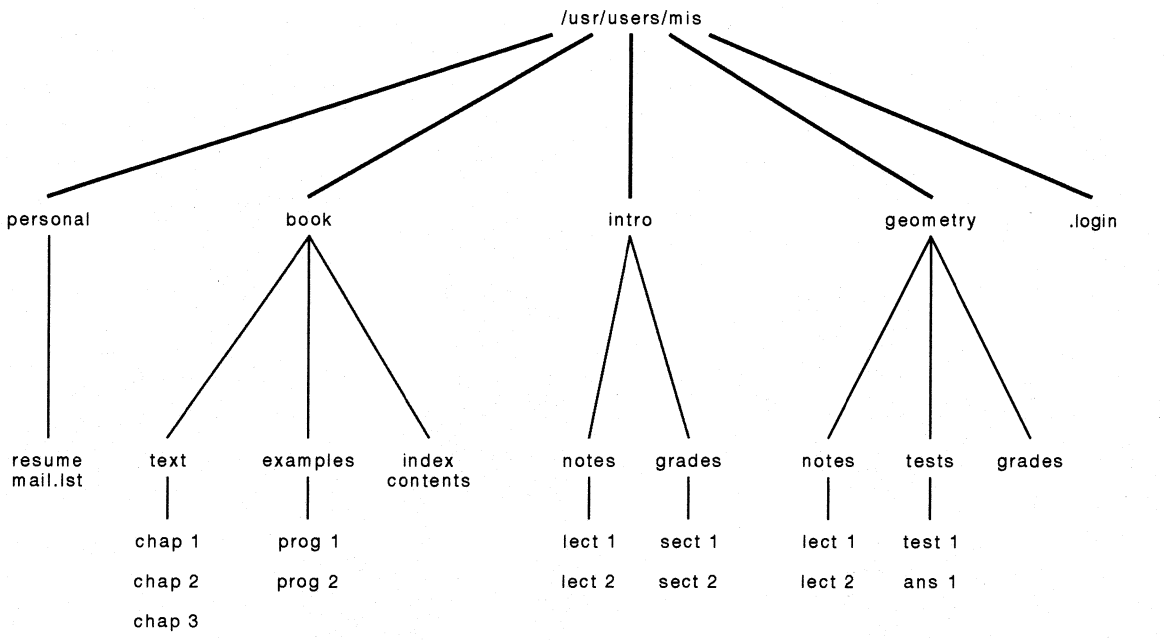
The figure represents the organization of files into four directories: `personal`, `book`, `intro`, and `geometry`. Assuming the working directory is `/usr/users/mis`, the commands used to create the four directories are:

```
$ mkdir personal
$ mkdir book
$ mkdir intro
$ mkdir geometry
```

These four directories can also be created using just one command:

```
$ mkdir personal book intro geometry
```

Figure 6–1 Creating Directories



Using mkdir to Create Directories

The following table shows uses of the `mkdir` command and their results.

Table 6–2 Creating Directories

	Command	Function
1.	`$ cd /usr/users/mis`	Sets working directory to `/usr/users/mis`
	`$ mkdir book/examples`	Creates subdirectory `examples` in the `book` directory
	`$ mkdir book/text`	Creates subdirectory `text` in the `book` directory
2.	`$ cd /usr/users/mis/book`	Changes working directory to `/usr/users/mis/book`
	`$ mkdir examples`	Creates subdirectory `examples` in the `book` directory
	`$ mkdir text`	Creates subdirectory `text` in the `book` directory
3.	`$ cd /usr/users/mis`	Changes working directory to `/usr/users/mis`

(continued on next page)

Table 6–2 (Cont.) Creating Directories

Command	Function
`$ mkdir examples`	Creates subdirectory `examples` in the `/usr/users/mis` directory
`$ mkdir text`	Creates subdirectory `text` in the `/usr/users/mis` directory

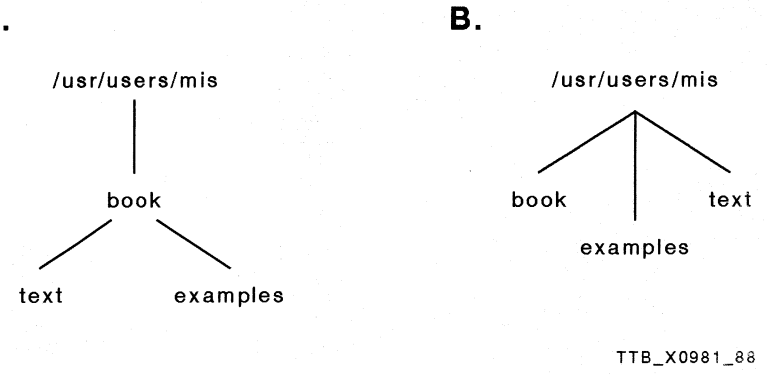
The same two directories (`text` and `examples`) are created in each sequence.

- In the first and second sequences, `text` and `examples` are created in the same place, in `/usr/users/mis/book`.
- In the third sequence, `text` and `examples` are created in a different place, in `/usr/users/mis`.

Interpreting Results of mkdir

The following figure contains two parts: A and B. Part A illustrates the directory structure obtained from command sequences 1 or 2 shown in Table 6–2. Part B illustrates the directory structure obtained from command sequence 3.

Figure 6–2 Results of mkdir Commands



Learning Activity: Creating Directories

1. In your home directory, create a directory called `exercises` using the `mkdir` command.
2. Without changing your working directory, list all files in the `exercises` directory.
3. Again, without changing your working directory, create a subdirectory called `sub1` in the `exercises` directory.
4. Change your working directory to `exercises`, and create a subdirectory called `sub2`.
5. Now, list all the files in the `exercises` directory. Both `sub1` and `sub2` should appear with the . and .. files.
6. Change back to your home directory and verify you are there.

Learning Activity Solution: Creating Directories

1. In your home directory, create a directory called `exercises` using the `mkdir` command.

   ```
   $ mkdir exercises
   ```

2. Without changing your working directory, list all files in the `exercises` directory.

   ```
   $ ls -a exercises
   ```

   Only the files . and .. will appear.

3. Again, without changing your working directory, create a subdirectory called `sub1` in the `exercises` directory.

   ```
   $ mkdir exercises/sub1
   ```

4. Change your working directory to `exercises`, and create a subdirectory called `sub2`.

   ```
   $ cd exercises
   $ mkdir sub2
   ```

5. Now, list all the files in the `exercises` directory. Both `sub1` and `sub2` should appear with the . and .. files.

   ```
   $ ls -a
   .     ..    sub1  sub2
   ```

6. Change back to your home directory and verify you are there.

   ```
   $ cd
   $ pwd
   /usr/users/you
   ```

Changing File and Group Permissions

Overview

The need to protect personal or confidential files from other users is obvious. Without protection, few users would store valuable information on a computer. Passwords keep nonusers out of the system, but they do not always provide protection from other users. Changing file and group permissions lets users restrict or allow other users to access files.

About File Permissions

File permissions are divided into three categories:

- User (owner) permissions

  When you try to access a file, the operating system looks to see what type of protection the file has. If you are the owner of the file (your user ID matches the user ID of the file), it checks the owner's rights.

- Group permissions

  If you are in the owner's group (your group ID matches the group ID of the file), the system checks permissions that the owner grants to the group.

- Other (world) permissions

  If neither your user ID nor your group ID matches those of the file, the operating system then checks the other permissions, to see if other system users are allowed to access the file.

  The superuser can access other users' files without regard to permissions.

Reading the User, Group, Other (ugo) Permissions

Use the `-l` option with the `ls` command to determine file permissions.

```
$ ls -l
-rwxr-x--x   1 you    users        145 Jun 29 15:22 reports
-rwxrw-r--   1 you    users        321 Nov 02 17:21 a.out
```

Starting in the second character position of the first field, the first three characters represent the **user's** permissions; the next three, the **group's** permissions; and the last three, the **others'** permissions.

Within each group of three characters:

Character & Position	Permission Granted to that Set of Users
`r` in the first position	Users can read the file.

Character & Position	Permission Granted to that Set of Users
w in the second position	Users can write to (change) the file. Write access implies delete access as well.
x in the third position	Users can execute the file. An executable file is a special file that can be run as a program. The file can be a binary file, or an ASCII file, such as a shell script. A directory is another example of an executable file. If a file is a directory, you must have both read and execute permission to access all the information about the files in it. Read permission lets you read the file names. Execute permission is required to get detailed information, such as file size and permissions.

Interpreting File Permission Fields

The table below describes the file permissions for the sample entry:

```
-rwxrw-r--   1 you    users        321 Nov 02 17:21 a.out
```

Field	Name	Description
-	File type	This first character indicates the file type. It is not part of the file permissions. File types include: - ordinary file d directory c character special file p named pipe s socket l symbolic link
rwx	User access	These three characters indicate the file access for the user (owner). The owner of this file has read, write and execute access.
rw-	Group access	The next three characters indicate the file access for the group. Group members have only read and write access to this file.
r--	Other access	The last three characters indicate file access for others (world). The world has only read access to this file.

Using chmod to Change File Permissions

The owner of the file can set and/or change permissions using the chmod (change file mode) command. There are two different ways to change file permissions:

- Using letters and operation symbols
- Using octal numbers

The format for the chmod command when specifying permissions using letters and operation symbols is:

chmod *class operation permission filename*

The following table describes the chmod format.

Table 6–3 chmod Command Format

Parameter	Function
class	Specifies which group's permission you are changing. Possible values are:
operation	Specifies whether you are adding or removing a permission. Possible values are:
permission	Specifies the type of permission you are adding or removing. Possible values are:
filename	Specifies the name of the file whose permissions you want to change.

class values:

Value	Description
u	User's or owner's permission
g	Group's permission
o	Others' or world permission
a	All (user, group and world)

operation values:

Value	Description
+	Add permission
-	Remove permission
=	Assign permission regardless of previous settings

permission values:

Value	Description
r	Read
w	Write
x	Execute

Using chmod with Letters and Symbols

This example shows how to use the chmod command with letters and operation symbols. The functions of each command are explained in the callouts below.

Example 6–5 Using Letters and Operation Symbols

```
$ ls -l a.out ❶
-rw-r-----   1 you   users          23951 Jan 08 13:02 a.out
$
$ chmod g+w,o+w a.out ❷
$ ls -l a.out ❸
-rw-rw--w-   1 you   users          23951 Jan 08 13:02 a.out
$
$ chmod g-r,o+r a.out ❹
$ ls -l a.out ❺
-rw--w-rw-   1 you   users          23951 Jan 08 13:02 a.out
$
$ chmod g+rx a.out ❻
$ ls -l a.out ❼
-rw-rwxrw-   1 you   users          23951 Jan 08 13:02 a.out
$
$ chmod a=rw a.out ❽
$ ls -l a.out ❾
-rw-rw-rw-   1 you   users          23951 Jan 08 13:02 a.out
```

❶ Display the current permissions on the file a.out.

❷ Give group and others (world) write access to the file a.out.

❸ List the permissions on the file a.out.

❹ Remove read access from the group and add read access to the world on the file a.out.

❺ List the permissions on the file a.out.

❻ Add read and execute access for the group on the file a.out.

❼ List the permissions on the file a.out.

❽ Give user, group, and others read and write access only to the file a.out.

❾ List the permissions on the file a.out.

Using chmod with Octal Numbers

You can also use octal numbers to change file permissions.

The format for the chmod command when specifying permission using octal numbers is:

chmod *octalnumber filename*

The chmod parameters are described in the following table.

Table 6–4 chmod Command Parameters

Parameter	Description
octalnumber	A three-digit octal number specifying the entire permission code, one digit each for the user, group, and others 1st digit — permission for the user 2nd digit — permission for the group 3rd digit — permission for others (world)
filename	The name of the target file or directory whose permissions you want to change

Interpreting Octal Permission Combinations

Each permission type (read, write, and execute), has an octal number that corresponds to it:

- Execute = 1
- Write = 2
- Read = 4

To specify a group of permissions (user, group, and others), add together the appropriate octal numbers that r, w, and x represent. For example:

- If you want to give the user read, write, and execute access, make the first digit of the three-digit octal number 7 (4 + 2 + 1 = 7).
- If you want the group to have write and execute access, make the second digit of the three-digit octal number 3 (2 + 1 = 3).
- If you want others to have read and write access, make the third digit of the three-digit octal number 6 (4 + 2 = 6).

The following table lists the eight possible permission combinations.

Table 6–5 Permission Combinations

Octal Number	Permission	Description
0	`---`	No permissions granted
1	`--x`	Execute
2	`-w-`	Write
3	`-wx`	Write and execute
4	`r--`	Read
5	`r-x`	Read and execute
6	`rw-`	Read and write
7	`rwx`	Read, write, and execute

The following example shows how to use the chmod command with octal numbers. The functions of each command are explained in the callouts below.

Example 6–6 Using Octal Numbers

```
$ ls -l a.out ❶
-rw-r-----   1 you   users          23951 Jan 08 13:02 a.out
$
$ chmod 662 a.out ❷
$ ls -l a.out ❸
-rw-rw--w-   1 you   users          23951 Jan 08 13:02 a.out
$
$ chmod 626 a.out ❹
$ ls -l a.out ❺
-rw--w-rw-   1 you   users          23951 Jan 08 13:02 a.out
$
$ chmod 777 a.out ❻
$ ls -l a.out ❼
-rwxrwxrwx   1 you   users          23951 Jan 08 13:02 a.out
$
$ chmod 600 a.out ❽
$ ls -l a.out ❾
-rw-------   1 you   users          23951 Jan 08 13:02 a.out
```

❶ Display the current permissions on the file a.out.

❷ Give the user read and write access, the group read and write access, and others write access to the file a.out.

❸ List the permissions on the file a.out.

❹ Give the user read and write access, the group write access, and others read and write access to the file a.out.

❺ List the permissions on the file a.out.

❻ Give the user, group, and others read, write and execute access to the file a.out.

❼ List the permissions on the file a.out.

❽ Give the user read and write access to the file a.out.

❾ List the permissions on the file a.out.

Learning Activity: Changing File Permissions

1. In your home directory, enter the command:

   ```
   cp  /etc/group  groupfile
   ```

   This command copies the file /etc/group to groupfile.
2. Display the current permissions on the file groupfile.
3. Using the chmod command with letters and operation symbols:
   a. Change the permissions on the file groupfile so that the permission code reads -rwxrw---x. Verify that the

permissions are set as you requested using the `ls -l` command.

b. Change the permissions on the file `groupfile` so that the user has read and write access only and group and others have no access. Verify that the permissions are set as you requested.

4. Using the `chmod` command with octal numbers:

a. Change the permissions on the file `groupfile` so that the permission code reads `-rwxr-xr-x`. Verify that the permissions are set as you requested using the `ls -l` command.

b. Change the permissions on the file `groupfile` so that the user, group, and others have only execute access. Verify that the permissions are set as you requested.

c. Change the permissions on the file `groupfile` so that the user has read and write access and group and others have only read access. Verify that the permissions are set as you requested.

Learning Activity Solution: Changing File Permissions

1. In your home directory, enter the command:

```
cp  /etc/group  groupfile
```

This command copies the file `/etc/group` to `groupfile`.

```
$ cp /etc/group groupfile
```

2. Display the current permissions on the file `groupfile`.

```
$ ls -l groupfile
-rwxr-x---   1 you   users      268 Jan 09 08:43 groupfile
```

3. Using the `chmod` command with letters and operation symbols:

a. Change the permissions on the file `groupfile` so that the permission code reads `-rwxrw---x`. Verify that the permissions are set as you requested using the `ls -l` command.

```
$ chmod g+w-x,o=x groupfile
$ ls -l groupfile
-rwxrw---x   1 you   users   268 Jan 09 08:43 groupfile
```

b. Change the permissions on the file `groupfile` so that the user has read and write access only and group and others have no access. Verify that the permissions are set as you requested.

```
$ chmod uo-x,g-rw groupfile
$ ls -l groupfile
-rw-------   1 you   users   268 Jan 09 08:43 groupfile
```

4. Using the chmod command with octal numbers:

   a. Change the permissions on the file groupfile so that the permission code reads -rwxr-xr-x. Verify that the permissions are set as you requested using the ls -l command.

```
$ chmod 755 groupfile
$ ls -l groupfile
-rwxr-xr-x   1 you   users   268 Jan 09 08:43 groupfile
```

   b. Change the permissions on the file groupfile so that the user, group, and others have only execute access. Verify that the permissions are set as you requested.

```
$ chmod 111 groupfile
$ ls -l groupfile
---x--x--x   1 you   users   268 Jan 09 08:43 groupfile
```

   c. Change the permissions on the file groupfile so that the user has read and write access and group and others have only read access. Verify that the permissions are set as you requested.

```
$ chmod 644 groupfile
$ ls -l groupfile
-rw-r--r--   1 you   users   268 Jan 09 08:43 groupfile
```

Changing Group Permissions

There are usually several users on the system that belong to the same group called users. Users in the same group can only protect a file or files from another member of that group by removing the group and others' access to those files. There may be times when you want only some members of the group to have access to a set of files. You can accomplish this by having the system manager or superuser create another group containing just the users with whom you want to share those files.

If you are a member of more than one group, you can change the group membership of your files using the chgrp command.

Using the chgrp Command

The chgrp command changes the group ID of specified files.

The format of the chgrp command is:

chgrp *group filename*

For example, the following command changes the group of the file reports to projX.

```
$ chgrp projX reports
```

The group projX in the previous example refers to the membership affiliation listed in the /etc/group file. To list the groups available on your system, enter:

```
$ cat /etc/group
```

To list the groups to which you belong, enter:

```
$ groups
```

The user invoking `chgrp` must belong to the specified group and be the owner of the file, or be the superuser. If you are not a member of a group with which you want to share files, ask the superuser or system manager to add you as a member of that group. Only superusers can change group affiliation for members.

Understanding the chgrp Command

Assume there are three users: Smith, Thomas, and Jones. Smith and Jones are working on a joint project that requires both of them to access the same files. Access to those files must be restricted from Thomas and anyone else not working on that project.

Currently, the three users, Smith, Thomas, and Jones, belong to the group `users`. This is defined in the `/etc/group` file. There is an entry in this file that looks similar to:

```
users:*:15:smith,thomas,jones
```

Any file that is created by a member of the group `users` is accessible to Smith, Thomas, and Jones as long as the file permissions on those files have group access.

For Smith and Jones to have access to files, and Thomas not to have access, Smith and Jones must belong to an additional group. A superuser can add a new group `/etc/group` file. The `/etc/group` file now contains an additional entry:

```
users:*:15:smith,thomas,jones
projX:*:16:smith,jones
```

Creating a Private Project Directory

Once Smith and Jones belong to a new group, Smith or Jones can create a directory called `newproject` for the files that they want to share. The following example shows how to do this.

Example 6–7 Creating a Private Project Directory

```
$ mkdir newproject ❶
$ ls -ld newproject ❷
drwxr-x---   2 smith    users       512 Jan 13 09:38 newproject
$
$
$ chgrp projX newproject ❸
$ ls -ld newproject
drwxr-x---   2 smith    projX       512 Jan 13 09:38 newproject
$
$
$ chmod 770 projX newproject ❹
$ ls -ld newproject ❺
drwxrwx---   2 smith    projX       512 Jan 13 09:38 newproject
```

❶ Create a new directory using the `mkdir` command.

❷ The `-d` option on the `ls` command tells the system to list only the specified directory, and not the contents of that directory.

❸ When the `newproject` was created, its group was `users`. `chgrp` changes the group to `projX`.

❹ Set the file permission so that only group members of `projX` have read, write, and execute access to the directory.

❺ When Smith and Jones log out and back in to the system, they will have access to that directory, but Thomas will not. And any files created within that new subdirectory will now be accessible to the group `projX`, which only includes Smith and Jones.

Renaming Files

Overview

The mv command renames a file from one file name to another, or moves a file from one directory to another.

Using the mv Command

When a file is moved and/or renamed using the mv command, the file itself on the disk is not moved or copied if the new file is on the same file system. Rather, the pointer to that file is moved from one place to another.

When an mv command moves a file from one file system to another, it copies the file to the new file system and deletes it from the original file system.

The following table shows examples of the mv command and their functions.

Table 6–6 Using the mv Command

Command/Example	Function
mv *filename directory* $ **mv mail.lst personal**	Moves the file mail.lst to the subdirectory personal under the current working directory.
mv *filename filename directory* $ **mv res skills personal**	Moves the files res and skills to the subdirectory personal under the working directory. (This command moves two files at the same time.)
mv *filename newfilename* $ **mv mail.lst mail**	Renames the file mail.lst in the current working directory to mail.
mv *filename dir/filename* $ **mv res intro/resume**	Moves the file res to the subdirectory intro under the working directory **and** renames it to resume.
mv *dir/allfiles newdir/allfiles* $ **mv personal/* intro/***	Moves all the files in the directory personal to the directory intro, keeping the same filenames. This command empties the personal directory.

Moving Files Safely

When you move a file into a file name that already exists, the original file is overwritten. For example, if you already have a file called chores and you enter the command $ *mv todo chores*, the original contents of the chores file is lost.

Use the -i option with the mv command if you want the system to prompt you for confirmation anytime a file might be overwritten. Type y to continue the copy or n to cancel the copy.

```
$ mv -i todo chores
overwrite chores? y
```

Learning Activity: Moving and Renaming Files

1. In your home directory, enter the command: `$ ls > listing`. This command puts the list of files in your home directory into a file called `listing`.
2. Use the `mv` command to rename the file `listing` to `dir.list`.
3. In your home directory, enter the command: `$ mkdir temp`. This command creates a subdirectory called `temp`. Using only one command, rename `dir.list` to `listing` and move it to `temp`.
4. Change your working directory to `temp` and list the files in that directory. You should see a file called `listing`. Then change back to your home directory.

Learning Activity Solution: Moving and Renaming Files

1. In your home directory, enter the command: `$ ls > listing`. This command puts the list of files in your home directory into a file called `listing`.

   ```
   $ ls > listing
   ```

2. Use the `mv` command to rename the file `listing` to `dir.list`.

   ```
   $ mv listing dir.list
   ```

3. In your home directory, enter the command: `$ mkdir temp`. This command creates a subdirectory called `temp`. Using only one command, rename `dir.list` to `listing` and move it to `temp`.

   ```
   $ mkdir temp
   $ mv dir.list temp/listing
   ```

   Or, if your home directory is `/usr/users/you`, you could use the following command:

   ```
   $ mkdir temp
   $ mv dir.list /usr/users/you/temp/listing
   ```

4. Change your working directory to `temp` and list the files in that directory. You should see a file called `listing`. Then change back to your home directory.

   ```
   $ cd temp
   $ ls
   listing
   $ cd
   ```

   Or, if your home directory is `/usr/users/you`, you could use the following command:

   ```
   $ cd /usr/users/you/temp
   $ ls
   listing
   $ cd
   ```

Copying Files

Overview

You may sometimes want to copy a file rather than rename it. You should make a backup copy of a file before making any major revisions to it. The cp (copy) command performs this function. When you copy a file using the cp command, the system creates a new copy of the file with a new name and leaves the original file intact.

The format of the cp command is:

cp *filename new-filename*

Using the cp Command

The following table gives some examples of the cp command and their functions.

Table 6–7 The cp Command

Example	Function
$ **cp mail memo**	Creates a copy of the mail file and puts it in a file called memo in the working directory.
$ **cp resume pers/skills**	Creates a copy of the resume file and puts it in a file called skills in the pers directory. The original file resume remains unchanged in the working directory.

Copying Files Safely

If you copy a file into a file that already exists, the old file is overwritten and lost. Use the -i option with the cp command if you want the system to prompt you for confirmation anytime a file might be overwritten. Type y to continue the copy or n to cancel the copy.

```
$ cp -i mail memo
overwrite memo? y
```

Copying Directories and Subdirectories

Use the -R option with the cp command to copy an entire directory, including subdirectories, to the specified destination.

```
$ cp -R /usr/users/jonah/whales /usr/users/noah/animals
```

This command copies the directory /usr/users/jonah/whales to /usr/users/noah/animals. The entire directory whales, including all its files, will be a subdirectory of /usr/users/noah/animals.

Learning Activity: Copying Files

1. Change your working directory to your home directory.
2. Copy the file `/etc/passwd` to your working directory and name it `pass1`.
3. Use the `mkdir` command to create a directory called `practice` in your home directory.
4. Without changing your working directory, copy the file `pass1` to the `practice` directory.
5. Change your working directory to `/usr/users/you/practice` and list the files in that directory. You should see the `pass1` file.
6. Again, copy the file `/etc/passwd` to your working directory and name it `pass1`. This time, use the `-i` option to confirm whether you want to continue to copy the file. Enter **y** or **n** when prompted to overwrite the existing file.

Learning Activity Solution: Copying Files

1. Change your working directory to your home directory.

   ```
   $ cd
   ```

   Or, if your home directory is `/usr/users/you`, you could use:

   ```
   $ cd /usr/users/you
   ```

2. Copy the file `/etc/passwd` to your working directory and name it `pass1`.

   ```
   $ cp /etc/passwd pass1
   ```

   Or, if your home directory is `/usr/users/you`, you could use:

   ```
   $ cp /etc/passwd /usr/users/you/pass1
   ```

3. Use the `mkdir` command to create a directory called `practice` in your home directory.

   ```
   $ mkdir practice
   ```

   Or, if your home directory is `/usr/users/you`, you could use:

   ```
   $ mkdir /usr/users/you/practice
   ```

4. Without changing your working directory, copy the file `pass1` to the `practice` directory.

   ```
   $ cp pass1 practice
   ```

   Or, if your home directory is `/usr/users/you`, you could use:

   ```
   $ cp pass1 /usr/users/you/practice
   ```

5. Change your working directory to `/usr/users/you/practice` and list the files in that directory. You should see the `pass1` file.

   ```
   $ cd  practice
   $ ls
   pass1
   ```

   Or, if your home directory is `/usr/users/you`, you could use:

   ```
   $ cd  /usr/users/you/practice
   $ ls
   pass1
   ```

6. Again, copy the file `/etc/passwd` to your working directory and name it `pass1`. This time, use the `-i` option to confirm whether you want to continue to copy the file. Enter **y** or **n** when prompted to overwrite the existing file.

```
$ cp -i /etc/passwd pass1
overwrite pass1? y
```

Or, if your home directory is */usr/users/you*, you could use:

```
$ cp -i /etc/passwd /usr/users/you/practice/pass1
overwrite /usr/users/you/practice/pass1? y
```

Removing Files and Directories

Overview

When you no longer need certain files or directories, you can delete them using the `rm` and `rmdir` commands.

Using the rm Command

The `rm` (remove) command deletes both single and multiple files.

The format of the `rm` command is:

`rm` *`filename ...`*

The following table gives some examples of the `rm` command and their functions.

Table 6–8 The rm Command Functions

Example	Function
`$ rm book`	Removes the file named `book` from the working directory
`$ rm book intro`	Removes the two files named `book` and `intro` from the working directory

The following example shows the commands to use to list the files in your working directory and remove the files `dead.letter`, `a.out`, `mail.help` and `core`.

Example 6–8 Using the rm Command

```
$ ls
a.out        calendar     dead.letter  intro        mbox
book         core         geometry     mail.help    personal
$
$ rm dead.letter a.out mail.help core
$
$ ls
book      calendar  geometry  intro     mbox      personal
```

Using rm with Multiple Links

The `rm` command does not actually remove a file, but only a link to the file. In some cases, there is more than one link to a file. If there is more than one link to the file, only the link to the file is removed, not the file itself. Once the last link to a file is removed, the file itself is removed.

Hard and soft links are explained later in this chapter.

Removing Files Safely

The -i option to the rm command asks you for confirmation before deleting the specified file(s). The file is deleted only if your response is y or yes.

```
$ rm -i dead.letter a.out mail.help core
rm: remove dead.letter? y
rm: remove a.out? y
rm: remove mail.help? y
rm: remove core? n
```

Note

Be very careful. Once you remove a file, you cannot retrieve it. Make a habit of using the -i option to the rm command.

Removing Files and Directories Together

The -r option to the rm command allows you to remove files and directories at the same time.

The format for the rm -r command is:

rm -r *pathname*.

The pathname specified is the name of the directory you want to remove. If you want to remove all files in the directory /usr/users/you/test and then the test directory itself, enter:

```
$ rm -r /usr/users/you/test
```

You can combine the -r option with the -i option, as shown in the following example.

```
$ rm -ri /usr/users/you/practice
rm: remove files in directory /usr/users/you/practice? y
rm: remove /usr/users/you/practice/pass1? y
rm: remove /usr/users/you/practice/ls.out? y
rm: remove /usr/users/you/practice/report.old? y
rm: remove /usr/users/you/practice? y
```

Using the rmdir Command

Use the rmdir command to remove directories that you no longer need.

The format for the rmdir command is:

rmdir *directory-name*

The rmdir command removes directories that contain no files. Remove files in a directory using the rm command before attempting to remove the directory with the rmdir command. You receive the following error message if you attempt to delete a directory named personal that is not empty.

```
$ rmdir personal
rmdir: personal: Directory not empty
```

The following table gives some examples of the rmdir command and their functions.

Table 6–9 The rmdir Command Functions

Example	Function
`$ rmdir personal`	Deletes the directory named personal from the current working directory if the personal directory is empty
`$ rmdir proj1 proj2 proj3`	Deletes three directories named proj1, proj2, and proj3 from the working directory, if they are empty

Learning Activity: Removing Files and Directories

1. In your home directory, create two directories called practice1 and practice2.
2. Change your working directory to practice1.
3. Use the cat command to create a file named friends that includes the names of some of your friends.
4. Make three copies of friends file named friends1, friends2, and friends3.
5. Delete the file named friends2.
6. Use the rm command with the -i option to delete the file named friends3. Enter y when prompted to remove the file. Then change back to your home directory.
7. Now remove the practice1 directory and the files within it using the rm command with both the -r and -i options.
8. Remove the practice2 directory using the rmdir command.

Learning Activity Solution: Removing Files and Directories

1. In your home directory, create two directories called practice1 and practice2.

   ```
   $ mkdir practice1 practice2
   ```

2. Change your working directory to practice1.

   ```
   $ cd practice1
   ```

3. Use the cat command to create a file named friends that includes the names of some of your friends.

   ```
   $ cat > friends
   Clark
   Marion
   Linda
   Rob
   Ctrl/D
   ```

4. Make three copies of `friends` file named `friends1`, `friends2`, and `friends3`.

```
$ cp friends friends1
$ cp friends friends2
$ cp friends friends3
```

5. Delete the file named `friends2`.

```
$ rm friends2
```

6. Use the `rm` command with the `-i` option to delete the file named `friends3`. Enter `y` when prompted to remove the file. Then change back to your home directory.

```
$ rm -i friends3
rm: remove friends3? y
$ cd
```

7. Now remove the `practice1` directory and the files within it using the `rm` command with both the `-r` and `-i` options.

```
$ rm -ri practice1
rm: remove practice1/friends? y
rm: remove practice1/friends1? y
rm: remove practice1? y
```

8. Remove the `practice2` directory using the `rmdir` command.

```
$ rmdir practice2
```

Linking Files

Overview

A link is a directory entry referring to a file. It provides a mechanism for sharing files. A link allows more than one user to access the same data. A file can have multiple links to it.

Using Inodes

You must understand what an **inode** is to understand links between files. An inode is a file system data structure containing information about a file, such as:

- The type and access mode of a file
- The file's owner
- The group access identifier
- The number of links to a file
- The size of the file in bytes
- The number of physical blocks used by the file
- The time the file was created, accessed, or modified
- Pointers to the data blocks on the disk

Use the `ls -i` command to obtain the inode number. The number is displayed in the first column of the `ls -i` display, before each file name as shown here:

```
$ ls -i
31554 bin        29387 dummy     29385 phone     29386 temp
```

Types of Links

There are two types of links: a hard link and **symbolic** (or soft) link. Links are generated using the `ln` command. By default, the `ln` command creates hard links.

The format of the `ln` command is:

`ln [-fns]` *source_file target_file*

The `source_file` is the existing file name. The `target_file` is the link file name.

The following table describes the `ln` options.

Table 6–10 Commonly Used ln Options

Option	Function
-f	Forces the removal of existing target pathnames before linking
-n	Does not overwrite the contents of the target file if the target file already exists; the -f option overrides this option
-s	Creates a symbolic link

Using Hard Links

When a hard link is established, it is indistinguishable from the original file. If the original file is deleted, the link file still exists, and contains the exact same data.

Hard links:

- Cannot refer to directories
- Cannot span file systems (must be on the same file system as the original file)
- Save disk space because only one copy of the data exists
- Share inodes

A link count is maintained for every file. When a file is originally created, its link count is one. The link count for the original file and each hard link are the same, and each are incremented or decremented as new hard links are established or removed.

Generating Hard Links

This example shows how to generate a hard link to user Smif's mortgage program. Smif's `mortgage` file is linked to a new entry called `mortgage-hlink` in your directory. Also shown are the results of `ls -il` commands for both the source and target files.

Example 6–9 Generating a Hard Link

```
$ pwd ❶
$ /usr/users/you
$ ln /usr/users/smif/mortgage mortgage-hlink ❷
$
$ ls -il mortgage-hlink ❸
17028 -rwxr-xr-x   2 smif users      39772 Apr 06 10:40 mortgage-hlink
$
$ ls -il /usr/users/smif/mortgage ❹
17028 -rwxr-xr-x   2 smif users      39772 Apr 06 10:40 mortgage
```

❶ Display your current working directory.

❷ Create a hard link to user Smif's file.

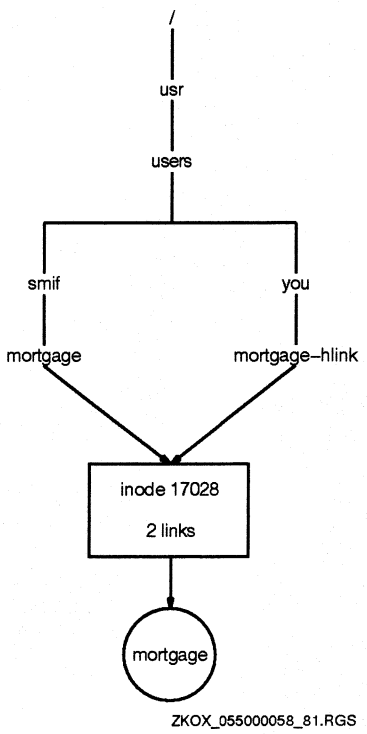
❸ Use the `ls -il` command to verify the hard link was created.

❹ The inode number for the source file and your hard link should be the same; in this case: 17028.

Relationship Between Hard Link and File

The figure shows the relationship between a hard link and the file to which it is linked.

Figure 6–3 Hard Links



Notes on the figure:

- The inode number is the same for both the link and linked file.
- The link count depends upon the number of links.
- If Smif removes his file, `/usr/users/smif/mortgage`, the file is still accessible through the link (`/usr/users/you/mortgage-hlink`)
- When either the link or the original file is removed, the link count is decreased. When the link count is reduced to zero, the file is deleted.

Symbolic (Soft) Links

A **symbolic** or soft link provides a path to a file.

Symbolic links:

- Can be made across file systems
- Can be made to a directory
- Have no effect on link count
- Have different inode numbers than the original file

With symbolic links, if the original file is deleted, the link would still exist, but there is no way to access the original file through the link.

The format of the `ln` command to create a symbolic link is:

`ln -s source_file target_file.`

In this example, a symbolic link is made to `/usr/users/smif/mortgage`. Directory listings for the link and linked files are shown. The `l` in the first bit of the mode field indicates that the file is a symbolic link.

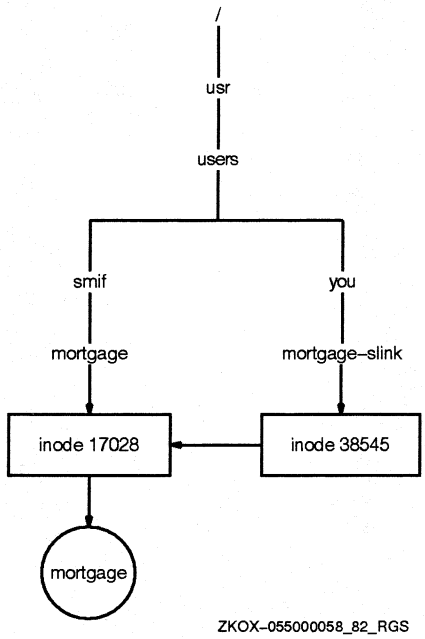
Example 6–10 Creating a Symbolic Link

```
$ ln -s /usr/users/smif/mortgage mortgage-slink
$ ls -il mortgage-slink
38545 lrwxr-xr-x  1 you   users     24 Apr 06 15:13 mortgage-slink -> /usr/users/smif/mortgage
$ ls -il /usr/users/smif/mortgage
17028 -rwxr-xr-x  1 smif  users  39772 Apr 06 10:40 /usr/users/smif/mortgage
$
```

Relationship Between Soft Links and Files

The figure shows the relationship between a symbolic link and the file to which it is linked.

Figure 6–4 Symbolic Links



Notes on the figure:

- Each directory entry has its own inode. The inode for Smif's directory entry points to the file on disk, whereas the inode for the `/usr/user/you` entry points to the inode for the file.
- In the directory listing for the link, `mortgage-slink`, the file size is the number of bytes in the name of the file that is being linked. In this case, the file size is 24 bytes, the number of bytes in the file name `/usr/users/smif/mortgage`.

- If Smif deletes the `mortgage` file, inode 17028 is deleted and the inode for the symbolic link has nothing to point to. The file cannot be accessed.

Learning Activity: Linking Files

Hard Links

1. Log in if necessary and copy the file `/etc/passwd` to a file named `dummy` in your home directory.
2. Get a directory listing showing the inode of the file `dummy` that you just created. Use the command `ls -il dummy`. Record the result here.

3. Create a hard link called `hard_link` to the file `dummy`.
4. Get a directory listing of both the original file and the link file using `ls -il`. Record the results here.

5. Are there any changes in the listing for the original file? If so, what are the changes?

6. Compare the directory listings for the link and the original file. What directory entries are the same? Why?

7. Delete the file `dummy`.
8. Use `cat hard_link` or `more hard_link` to display the file. Can you display the file (after deleting the original file)? Why?

9. Delete the link file `hard_link`.
10. Try to create a hard link `hard_link` to `/etc/passwd`. What happens?

11. If the previous command failed, create a hard link `hard_link` to a file owned by another user on `/usr/users`, or whatever file system your home directory is on.
12. Get a directory listing of your working directory using `ls -l`. What is unusual?

13. Delete the link file `hard_link`. You can delete a file without write permission to the file if you have write permission on the directory it is in.

Symbolic Links

1. Once more, copy `/etc/passwd` to a file called `dummy` in your home directory and get a directory listing for the file that shows the inode. Record the results here.

2. Create a symbolic link called `soft_link` to the file `dummy`.
3. Get a directory listing (`ls -il`) of both the link file and the original file. Record them here.

4. What is there about the directory listings that indicates that `soft_link` is a symbolic link? Has the link count of `dummy` changed?

5. Remove the file `dummy`.
6. Can you still display the file with `cat soft_link` or `more soft_link`. Why or why not?

7. Delete the link file `soft_link`.
8. Try to create a symbolic link `soft_link` to `/etc/passwd`. What happens?

9. Do a directory listing to determine who owns the link file and the `/etc/passwd` file.
10. Delete the link file `soft_link`.

Learning Activity Solution: Linking Files

Hard Links

1. Log in if necessary and copy the file `/etc/passwd` to a file named `dummy` in your home directory.

```
$ cp /etc/passwd dummy
```

2. Get a directory listing showing the inode of the file `dummy` that you just created. Use the command `ls -il dummy`. Record the result here.

```
$ ls -il dummy
80537 -rw-r--r--   1 you    users     2623 Dec 23 13:01 dummy
```

3. Create a hard link called `hard_link` to the file `dummy`.

```
$ ln dummy hard_link
```

4. Get a directory listing of both the original file and the link file using `ls -il`. Record the results here.

```
$ ls -il dummy hard_link
80537 -rw-r--r--   2 you    users     2623 Dec 23 13:01 dummy
80537 -rw-r--r--   2 you    users     2623 Dec 23 13:01 hard_link
```

5. Are there any changes in the listing for the original file? If so, what are the changes?

   The number of links increased from one to two.

6. Compare the directory listings for the link and the original file. What directory entries are the same? Why?

The inode numbers, link count, owner, group, size, and dates are all the same.
The only difference is the file names themselves.

7. Delete the file `dummy`.

```
$ rm dummy
```

8. Use `cat hard_link` or `more hard_link` to display the file. Can you display the file (after deleting the original file)? Why?

```
$ cat hard_link
root:PrFX.BclhWLHc:0:1:system PRIVILEGED account:/:
nobody:*Nologin:65534:65534:anonymous NFS user:/:
    .
    .
    .
```

When the file was deleted, the link count was not reduced to zero.
Therefore, the file is still accessible.

9. Delete the link file `hard_link`.

```
$ rm hard_link
```

10. Try to create a hard link `hard_link` to `/etc/passwd`. What happens?

```
$ ln /etc/passwd hard_link
ln: /etc/passwd and hard_link are located on different file systems.
```

This command may fail because `/etc` and your home directory are often mounted on separate file systems.

11. If the previous command failed, create a hard link `hard_link` to a file owned by another user on `/usr/users`, or whatever file system your home directory is on.

```
$ ln /usr/users/them/sample hard_link
```

12. Get a directory listing of your working directory using `ls -l`. What is unusual?

```
$ ls -l
-rw-r--r--   1 you    users      522 Dec 22 16:59 groupfile
-rwxr-xr-x   2 them   users     1639 Dec 22 11:51 hard_link
-rw-r--r--   1 you    users       29 Dec 22 11:12 now
```

Although it appears in your directory, hard_link is owned by them.

13. Delete the link file `hard_link`. You can delete a file without write permission to the file if you have write permission on the directory it is in.

```
$ rm hard_link
rm: override protection 755 for hard_link? y
```

Symbolic Links

1. Once more, copy /etc/passwd to a file called dummy in your home directory and get a directory listing for the file that shows the inode. Record the results here.

```
$ cp /etc/passwd dummy
$ ls -il dummy
80551 -rw-r--r--   1 you    users     2623 Dec 23 13:19 dummy
```

2. Create a symbolic link called soft_link to the file dummy.

```
$ ln -s dummy soft_link
```

3. Get a directory listing (ls -il) of both the link file and the original file. Record them here.

```
$ ls -il dummy soft_link
80551 -rw-r--r--   1 you   users  2623 Dec 23 13:19 dummy
80553 lrw-r--r--   1 you   users     5 Dec 23 13:26 soft_link -> dummy
```

4. What is there about the directory listings that indicates that soft_link is a symbolic link? Has the link count of dummy changed?

   The first character in the mode field is "l", indicating that it is a symbolic link.
   Also, the file name is shown with the arrow pointing to the file it is linked to.
   The link count for dummy has not changed.

5. Remove the file dummy.

```
$ rm dummy
```

6. Can you still display the file with cat soft_link or more soft_link. Why or why not?

```
$ cat soft_link
cat: cannot open soft_link
```

   When the original file was removed, the inode for the file was removed.
   The link file points to a nonexistent file.

7. Delete the link file soft_link.

```
$ rm soft_link
```

8. Try to create a symbolic link soft_link to /etc/passwd. What happens?

```
$ ln /etc/passwd soft_link
```

   This works because you can do a symbolic link across file systems.

9. Do a directory listing to determine who owns the link file and the /etc/passwd file.

```
$ ls -l soft_link /etc/passwd
-rw-r--r--   1 root  system  2623 Dec 22 13:48 /etc/passwd
lrw-r--r--   1 you   users     11 Dec 23 13:35 soft_link -> /etc/passwd
```

10. Delete the link file `soft_link`.

```
$ rm soft_link
```

Summary

Creating Files

The following table summarizes how to use the `cat` command to create and concatenate files.

Task	Enter	Result
Create a file	`$ cat > filename`	Creates a new file with the specified file name
Add text to a file	`$ cat >> filename`	Appends (adds) text to the bottom of the specified file name
Adds one file to another file	`$ cat file1 >> file2`	Appends the contents of `file1` to the end of `file2`

Creating Directories

The command for creating a directory is: `mkdir filename`.

The following command creates a directory named `report1`. Unless an alternate pathname is given for the new directory, it is automatically a subdirectory of your current working directory.

```
$ mkdir report1
```

Changing File and Group Permissions

There are two methods to change file permissions using the `chmod` command:

- Using letters and operation symbols

  `$ chmod [ugoa] [+-=] [rwx] filenames`

- Using octal numbers

  `$ chmod octalnumber filenames`

The following table lists some examples of how to use the `chmod` command.

Enter	Result
`$ chmod u+rwx,g+rwx,o-rwx chap12`	Changes the file permission on the file `chap12` so that the user and group have read, write, and execute access, and others have no access. This method uses letters and operation symbols.

Enter	Result
`$ chmod 770 chap10`	Changes the file permission on the file `chap10` so that the user and group have read, write, and execute access and others have no access. This method uses octal numbers.

The `chgrp` changes the group ownership of a file. The format is `chgrp group filename`. For example,

```
$ chgrp staff reports
```

changes the group of a file called `reports` to `staff`.

The following rules apply to changing group permissions and changing groups:

- To change the group permission of a file, you must own the file and belong to the new group.
- Only system managers or superusers can add new groups to the `/etc/group` file.
- When the `/etc/group` file is modified to include a new group, members of that new group must log out and log back in again for membership in that group to be active.

Renaming Files

The following table summarizes the commands used to rename files.

Task	Enter	Result
Rename a file	`$ mv file1 file2`	Renames a file called `file1` to `file2`.
Move a file	`$ mv resume personal`	Moves `resume` from the current directory to a subdirectory called `personal`. The name of the file is unchanged.
Move and rename a file	`$ mv resume personal /res1`	Moves `resume` from the current directory to a subdirectory called `personal`. At the same time `resume` is renamed to `res1`.

Copying Files

The following table summarizes the commands used to copy files.

Enter	Result
`$ cp file1 file2`	Copies `file1` into another file called `file2` in the working directory.
`$ cp file1 directory1`	Copies `file1` and puts the copy in the `directory1` directory. The file name remains unchanged.
`$ cp file1 directory1/file2`	Copies `file1` and puts the copy in the `directory1` directory with the new name `file2`.

Removing Files and Directories

The following table summarizes the commands and options used to remove files and directories.

Task	Enter	Result
Remove a file	`$ rm phone.lis`	Removes a file called `phone.lis`
Remove multiple files	`$ rm notes memos`	Removes two files called `notes` and `memos`
Remove an empty directory	`$ rmdir reports`	Removes an empty directory named `reports`
Remove directory and its contents	`$ rm -r tools`	Removes the contents of a directory named `tools`, then the directory itself

Linking Files

There are two types of links:

- Hard links:
  - Indistinguishable from the original entry
  - Cannot span file systems
  - Cannot refer to directories
  - Share inodes
- Symbolic (soft) links:
  - Can be made to a directory
  - Can be made across file systems
  - Creation of a symbolic link has no effect on link count
  - Delete a file which has an existing symbolic link, and there is no way to access the file through the link

7

Shell Overview and Korn Shell Introduction

Shell Overview

Introduction

Although this course primarily discusses the Korn shell and teaches only Korn shell programming, you should know what shells are generally available for use on a UNIX system. You should also know the basic features of each shell, how to determine your default login shell, and how to change shells.

This chapter covers only the Korn shell beyond the introductory level.

Objectives

To select a shell, and initialize the user environment, you should be able to:

- Define the purpose of the shell and list the shells available with DEC OSF/1
- Describe the general features of the DEC OSF/1 shells
- Determine your default login shell and change your shell
- Use wildcard operators to specify lists of file names
- Describe the need for shell variables
- List and define the use of variables with the Korn shell
- Define the need for, and use of, shell user startup files
- Describe typical contents of the Korn shell user startup files

Resources

- *OSF/1 User's Guide*
- DEC OSF/1 `man` pages

Shell Introduction

Overview

This section provides an introduction to the UNIX shells, the purpose of shells, shells available on a DEC OSF/1 system, general features of the shells, and how to change from one shell to another.

Purpose of Shells

UNIX shells can be used as a:

- User interface to the UNIX operating system
- Command-language interpreter:
  - Interprets the command you enter
  - Selects and runs the command or program
  - Provides the results to the screen or file as directed by the command
- Way to execute shell scripts, which consist of a sequence of shell commands in a text file. This provides a way to execute a long sequence of commands that you need to repeat.

DEC OSF/1 Available Shells

The DEC OSF/1 operating system provides the following shells:

- C shell
- Bourne shell (which includes the restricted Bourne shell)
- Korn shell

The default prompts for the DEC OSF/1 shells are:

- C shell: `%`
- Bourne shell: `$`
- Korn shell: `$`

Bourne Shell Features

The Bourne shell, `/bin/sh`, provides string-valued variables and control flow primitives including branching and iteration. It provides a restricted shell which provides specified users a limited set of capabilities and privileges.

C Shell Features

The C shell (`/bin/csh`) is designed for interactive use, providing:

- A command history buffer (must be saved for use at next login)
- Reexecution of commands
- Modification of previously executed commands
- Command aliases
- File name completion

For shell programming, it provides:

- Variable expansion
- Conditional commands (`if`, `switch`
- Iteration commands (`foreach`, `while`, `shift`)
- Arithmetic, bitwise, string comparison, and logical operations
- File inquiry operations
- Reading data from the terminal
- Writing data to the terminal
- Limited signal-handling capabilities

Korn Shell Features

The Korn shell (`/bin/ksh`) combines the ease of use and interactive features of the C shell with the ease of programming of the Bourne shell. For interactive use, it provides:

- A command history buffer which is saved by ksh from login to login
- Reexecution of commands
- Modification of previously executed commands
- Command-line editing
- Command aliases
- File name completion
- The fastest execution time
- User compatibility with the Bourne shell

For shell programming, it provides:

- Variable expansion
- Conditional commands (`if`, `case`)
- Iteration commands (`for`, `select`, `while`, `until`)
- Arithmetic, bitwise, string comparison, and logical operations
- Array operations
- File operators
- Creation and use of menus
- Reading or writing data from/to the terminal
- Opening (and closing) multiple files for reading and writing
- Scripting compatibility with the Bourne shell (most Bourne shell programs will run under the Korn shell)

DEC OSF/1 Shell Features Summary

The following table summarizes features of the three shells available in DEC OSF/1.

Table 7–1 DEC OSF/1 Shell Features Summary

Feature	Bourne Shell	C Shell	Korn Shell	Function
Shell programming	X	X	X	Provides a programming language with features such as looping and condition statements
Signal trapping	X	X	X	Traps interruptions and other signals sent by the operating system
Restricted shells	X			Offers a controlled shell environment to limits user access
Command aliases		X	X	Allows the user to abbreviate shell commands
Command history		X	X	Stores executed shell commands in a command history buffer for display or reexecution
File name completion		X	X	Allows a portion of a file name to be entered, and the shell completes the pathname or provides a list of choices
Array		X	X	Allows data to be grouped and manipulated by name
Integer arithmetic		X	X	Allows the shell to perform integer arithmetic functions
Job control		X	X	Provides the ability to monitor background processes
Command line editing			X	Allows you to edit a command on the command line or a previously entered command on the command line or in a temporary buffer

Determining the Login Shell

The `/etc/passwd` file contains, among other things, the name of the shell to execute when you log in.

To determine your login shell, enter:

```
more /etc/passwd
```

Look for your account name. The entry after the last colon (:) is the pathname of the shell to execute. Shell default pathnames are listed in Table 7–2. An example `/etc/passwd` entry is shown in Example 7–1. In this example, the pathname of the default login shell is `/bin/ksh`, the Korn shell.

Table 7–2 Shell Pathnames

Shell	Pathname
C	/bin/csh
Bourne	/bin/sh
Korn	/bin/ksh

Example 7–1 /etc/passwd Entry

```
you:Gux27ZVJ.QtV6:1745:15:User for U&C:/usr/users/you:/bin/ksh
```

Note

If the last field in your /etc/passwd entry is blank, your login shell is the Bourne shell (/bin/sh).

Temporarily Changing Your Shell

To temporarily change your shell (start a subshell), enter the pathname of the requested shell.

To change shells from the Korn shell to the C shell, enter:

```
$ /bin/csh
%
```

To return to the previous shell, either:

- Enter **exit**

  or

- Press Ctrl/D (the eof character)

Changing Your Login Shell

Use the chsh command (**ch**ange **sh**ell) to change your login shell.

For example, to change your login shell from C shell to the Korn shell, enter:

```
% chsh
Changing login shell for you
Old shell: /bin/csh
New shell: /bin/ksh
%
```

The chsh command changes your /etc/passwd entry to reflect the shell change. You must log out and log back in again for the change to take effect.

You can only change your shell to a shell listed in the /etc/shells file.

Learning Activity: Changing Your Shell

In this section of the lab exercise, you will determine your login shell, change shells temporarily, and change your login shell.

1. Use the `more` command to display the `/etc/passwd` file. Look for your account name and determine your login shell. If it is not the Korn shell, set it to the Korn shell at the end of this lab exercise.
2. Display the file `/etc/shells` to determine which shells are available on your system.
3. Temporarily change shells to a different shell by entering the pathname of the shell you want to execute on the command line.

   Type exit to exit the temporary shell back to your login shell.
4. Change your default shell using the `chsh` command. Again, pick a shell other than your default shell.

   Display `/etc/passwd` and verify that your login shell has changed to the selected shell.

   If your login shell is not now the Korn shell, use `chsh` to change your login shell to the Korn shell.

   If you want to use the new shell you changed to, you must log out then log back in again.

Learning Activity Solution: Changing Your Shell

In this section of the lab exercise, you will determine your login shell, change shells temporarily, and change your login shell.

1. Use the `more` command to display the `/etc/passwd` file. Look for your account name and determine your login shell. If it is not the Korn shell, set it to the Korn shell at the end of this lab exercise.

```
$ more /etc/passwd
root:PrFX.BclhWLHc:0:1:system PRIVILEGED account:/:
nobody:*Nologin:65534:65534:anonymous NFS user:/:
    .
    .
    .
you:kdtLvq1SAJqB2:1745:15:User for U&C:/usr/users/you:/bin/ksh
    .
    .
    .
```

2. Display the file `/etc/shells` to determine which shells are available on your system.

```
$ cat /etc/shells
/bin/sh
/bin/csh
/bin/ksh
```

3. Temporarily change shells to a different shell by entering the pathname of the shell you want to execute on the command line.

For example, if your present shell is the Korn shell, to change to the C shell, you would enter:

```
$ /bin/csh
%
```

Type exit to exit the temporary shell back to your login shell.

```
% exit
% $
```

4. Change your default shell using the `chsh` command. Again, pick a shell other than your default shell.

   This sample assumes that your default shell is the Korn shell and changes the login shell to the C shell.

```
$ chsh
Old shell: /bin/ksh
New shell: /bin/csh
$
```

   Display `/etc/passwd` and verify that your login shell has changed to the selected shell.

```
$ more /etc/passwd
root:PrFX.BclhWLHc:0:1:system PRIVILEGED account:/:
nobody:*Nologin:65534:65534:anonymous NFS user:/:
   .
   .
   .
you:kdtLvq1SAJqB2:1745:15:User for U&C:/usr/users/you:/usr/bin/csh
   .
   .
   .
```

   If your login shell is not now the Korn shell, use `chsh` to change your login shell to the Korn shell.

   This sample assumes that your login shell is now the C shell.

```
% chsh
Old shell: /bin/csh
New shell: /bin/ksh
%
```

   If you want to use the new shell you changed to, you must log out then log back in again.

   No solution required.

Wildcard Operators

Overview

You may sometimes want to process a group of related files without typing out the full names of all of them. The Korn shell provides **wildcard operators** you can use to identify names of files to process.

Matching File Names

The string that you specify as a file name is scanned for the special characters asterisk (*) and question mark (?). If one of these characters is found, the file name is treated as a pattern and compared to the names of the files in that directory. A list of matching file names is generated, and used to replace the wildcard pattern on the command line.

You can also use a wildcard pattern to specify the file name in a command that accepts only a single file name parameter, as long as that pattern matches only one file name.

Wildcard operators cannot be used to match the leading period in hidden file names.

The * Operator

The asterisk (*) operator can be used to match any string of zero or more characters. Some examples of the * operator are shown in the following table.

Pattern	Matches Any Name That...
file*	Begins with the string `file`
*.c	Has the `.c` extension
f	Contains a lowercase `f`
file*.txt	Begins with the string `file` and ends with the `.txt` extension

Note

Since the asterisk (*) operator can be used to match a period, a single * can be used to match a file name with extension, such as `file.txt`

The ? Operator

The question mark (?) operator can be used to match any single character. Some examples of the ? operator are shown in the following table.

Pattern	Matches Any Name That...
file?	Begins with the string `file` and contains exactly one more character
???	Contains exactly 3 characters
file*.?	Begins with the string `file` and ends with any one-character extension

Using Wildcard Operators in Commands

You can use a wildcard pattern to generate a list of file names for a command that accepts multiple file name parameters, as shown in the following table.

Command	Function
cp *.c *.h src	Copies all files with the `.c` or `.h` extension to the `src` directory
ls *proj*	Lists all file names that contain the string `proj`
rm *.log	Deletes all files with the `.log` extension
ls ?	Lists all single-character file names

Escaping Wildcard Operators

The special meaning of wildcard operators can be **escaped** (turned off) in either of two ways:

- Precede the wildcard operator with a backslash (\ )
- Enclose the string in single quotes (')

You can delete a file named `*wow*` with the command `rm \*wow\*` or the command `rm '*wow*'` .

Learning Activity: Using Wildcard Operators

1. Use the `touch` command to create three files, `a`, `b`, and `c`. (If you already have files with these names, it will just update the access time.) Use a wildcard operator with the `ls -l` command to show the access time of these files.
2. Use a wildcard expression with the `ls -ld` command to display the attributes of your hidden files **only**.

Learning Activity Solution: Using Wildcard Operators

1. Use the `touch` command to create three files, `a`, `b`, and `c`. (If you already have files with these names, it will just update the access time.) Use a wildcard operator with the `ls -l` command to show the access time of these files.

```
$ touch a b c
$ ls -l ?
-rw-r--r--   1 you      users          0 Mar  2 16:42 a
-rw-r--r--   1 you      users          0 Mar  2 16:42 b
-rw-r--r--   1 you      users          0 Mar  2 16:42 c
```

2. Use a wildcard expression with the `ls -ld` command to display the attributes of your hidden files **only.**

```
$ ls -ld .*
drwxr-x--x   5 you      users       1536 Mar  2 15:49 .
drwxr-xr-x  32 root     system      1024 Feb 25 16:22 ..
-rwxr-xr-x   1 you      users       1617 Jan  5 10:05 .profile
-rw-------   1 you      users       4262 Mar  2 16:32 .sh_history
$
```

Korn Shell Environment

Overview

This section provides an introduction to the concept of the parent and child process and the environment.

Parent and Child Processes

Assume that you are logged in to a UNIX system, you are using the Korn shell and are running a Korn shell script file called `first.ksh`. The running script, `first.ksh`, is a **child** of the Korn shell. The Korn shell is the **parent** of the script file.

Note

There are other parent-child relationships, as the Korn shell itself has a parent. However, for this discussion, assume that the parent is the Korn shell and the child is a script file or an executed command.

The Environment

Every parent or child has an environment. An environment is a collection of privileges, resources, and traits.

- A parent cannot inherit environmental traits from the child.
- A child cannot damage the environment of the parent.
- By default, a parent process passes rights and privileges to the children.
- By default, a parent does not pass variables, functions, or aliases to the child.

Environment Characteristics Passed from the Parent Process

The following environmental characteristics are passed from a parent to a child:

- The ability to read, write, or execute objects (the parent's access rights to files, directories, and so forth)
- Files that the parent has opened
- Resource limits (for example, the amount of main memory that a process is allowed to use)
- Response to signals
- The value of the **IFS** variable

Environment Characteristics NOT Passed from the Parent Process

A child does not inherit the following environmental characteristics from the parent:

- Aliases defined by the parent
- Functions defined by the parent
- Variables defined by the parent
- The values of the parent's Korn shell reserved variables

- The values of the parent's option settings

Note

For a child to inherit a variable or function from the parent, the parent must **export** the variable or function to the child.

Korn Shell Variables

Variables perform an important role in defining the Korn shell environment. There are two types of variables:

- Reserved variables
- User-defined variables

  User-defined variables are any variables that you define for your own use.

Reserved Variables

The Korn shell has many reserved variables, and they fall into two classes:

- Variables whose values the Korn shell sets and updates. Some of them are shown in Table 7–3.
- Variables whose values the Korn shell does not automatically set. Some of them are shown in Table 7–4.

Korn shell reserved variables have uppercase names. You should give user-defined variables lowercase names. This makes the reserved variables easier to differentiate.

Table 7–3 Some Variables Set and Updated by the Korn Shell

Variable	Description
ERRNO	The error number of the most recently failed system call
LINENO	The current line number being executed within a script or function
OLDPWD	The previous working directory
PPID	The process identification number (PID) of the parent
RANDOM	A random integer between 0 and 32,767
SECONDS	The number of seconds since the Korn shell was invoked
$	The PID of the current process
!	The PID of the last process placed in the background
?	The exit status of the most recent Korn shell statement, command, or user program. An error status of zero (0) indicates success, while a nonzero error status indicates failure

Table 7–4 Korn Shell Variables Not Set by the Korn Shell

Variable	Function
CDPATH	A colon (:) separated list of directories used by the `cd` command. If the directory specified for `cd` does not start with / (full pathname), the shell searches each directory in **CDPATH** in order, looking for the specified directory, changing to that directory
EDITOR	Defines the pathname of the command-line editor
ENV	Defines the pathname of the environmental startup script file which the Korn shell executes every time you start up a new Korn shell or run a Korn shell script
FCEDIT	Defines the default editor for the built-in `fc` command
HISTFILE	The pathname of the `history` file
HISTSIZE	The number of accessible commands in the `history` file
HOME	The login directory
MAIL	The pathname of the master mail file
MAILCHECK	The frequency at which the Korn shell checks for the receipt of new mail. The default is 600 seconds
PATH	Defines a colon (:) separated list of directories where the Korn shell should search for the command you want to execute. The default is `$HOME/bin:/usr/bin:`. A null is assumed after the trailing colon (:), designating the current directory.
PS1	The command-line prompt
PS2	The prompt for a command that extends more than one line
TERM	The type of terminal your are using
VISUAL	Another variable that defines the command-line editor

Setting Korn Shell Variables

An assignment statement is used to set variables.

The format is of an assignment statement is:

name=value

Note

No unquoted spaces or tabs are allowed before or after the equal sign.

As shown in the example, you can create the variables **myname** and **address**, and assign a value to the **EDITOR** and **FCEDIT** variables.

Example 7–2 Setting Some Korn Shell Variables

```
$ myname="Charlie Brown"
$ address="2 Tired Way"
$ EDITOR=emacs
$ FCEDIT=emacs
```

If quotes are required as part of the variable assignment, use single quotes in the assignment as:

```
$ myname='"Charlie Brown"'
```

Using set to Display Korn Shell Variables

All Korn shell variables can be displayed using the `set` command with no options on the command line, as shown in the example.

Example 7–3 Displaying Korn Shell Variables Using set

```
$ set
CDPATH=.:..:/usr/users/you:/usr
EDITOR=emacs
ENV=/usr/users/you/.kshrc
FCEDIT=emacs
    .
    .
    .
TERM=vt100
USER=you
myname='Charlie Brown'
address='2 Tired Way'
```

Note

The variables **EDITOR** and **FCEDIT** are placed in alphabetical order with the other variables. The two variables we defined are placed at the end of the display, in the order in which we defined them.

Using print to Display Korn Shell Variables

You can also display an individual variable with the `print` command, which has the format:

```
print $name
```

For example:

```
$ print $myname $address
Charlie Brown 2 Tired Way
$
```

To control the spacing of the `print` command output, enclose the parameters in double quotes as follows.

```
$ print "$myname          $address"
Charlie Brown     2 Tired Way
$
```

Unsetting Korn Shell Variables

Korn shell variables are unset (cleared) with the `unset` command, which has the format:

```
unset name
```

For example, to clear the variables **myname** and **address**, type:

```
$ unset myname
$ unset address
$
```

Declaring Variable Attributes

The Korn shell supports some attributes, and you may want to declare the attributes of the variables you use for Korn shell programming.

- You do not have to explicitly declare the attributes of a variable before using it. With the Korn shell, all variables are by default, strings, but mathematical operations can be carried out on Korn shell strings. Advanced Korn shell programmers will probably want to use the capabilities to declare a variables' attribute.
- The Korn shell supports the integer data type, and by declaring integer variables, mathematical operations are speeded up.
- By careful declaration, strings can be converted to uppercase or lowercase and other attributes can be used to make neat columns of output.

Using typeset with Korn Shell Variables

A Korn shell built-in variable, `typeset`, is used to:

- Set variable attributes
- Unset variable attributes
- Set values
- Display variables and/or their attributes. Do not specify a name, and specify:

  -: To display names and values of all variables that have the attributes you specify

+: To display only the names

Nothing (neither - nor +): To display names and attributes of all variables

Using Attributes with typeset

Some of the attributes shown in the table can be specified for `typeset`.

Table 7–5 Attributes Used with typeset

Attribute	Function
`-u`	Uppercase
`-l`	Lowercase
`-i`*n*	Integer, where `n` specifies the arithmetic base
`-r`	Read only
`-x`	Export
`-L`*n*	Left-justifies; `n` specifies field width
`-LZ`*n*	Left-justifies and strips leading zeros; `n` specifies field width
`-R`*n*	Right-justifies; `n` specifies field width
`-RZ`*n*	Right-justifies and fills with leading zeros; `n` specifies field width

Using typeset to Display Variables

The following example shows how to use the `typeset` command as another way to display variables. The names of the variables, and their attributes, are provided by using `typeset` with no options.

Example 7–4 Displaying Variables Using typeset

```
$ typeset
export VISUAL
export PATH
integer ERRNO
export CDPATH
integer OPTIND
function LINENO
export EXINIT
export LOGNAME
export MAIL
function SECONDS
integer PPID
export PS1
PS3
PS2
OPTARG
export USER
function RANDOM
export SHELL
integer TMOUT
export SAVEPID
export HOME
export FCEDIT
export TERM
export PWD
export TZ
export ENV
integer MAILCHECK
$
```

Using typeset to Define Variables

The example shows how to use the `typeset` command to define variables.

Example 7–5 Defining Variables Using typeset

```
$ typeset -u uper="This is an uppercase Test" ❶
$ typeset -l lower="LOWERCASE This" ❷
$ print "$uper     $lower" ❸
THIS IS AN UPPERCASE TEST     lowercase this
$ typeset -r konstant=8214 ❹
$ print $konstant  ❺
8214
$ konstant=96420  ❻
ksh: konstant: is read only
$ typeset +r konstant  ❼
$ typeset -r konstant=96420 ❽
$ print $konstant  ❾
96420
$
```

❶ Defines the variable **uper** and sets the attributes to uppercase. Assigns a value to it.

❷ Defines the variable **lower** and sets the attributes to lowercase. Assigns a value to it.

❸ Displays the variables **uper** and **lower** with some spaces between them. Note that **uper** is forced to uppercase and **lower** is forced to lowercase.

When using `print` to display variables, use double quotes to allow the use of spaces. Use `\t` to provide a tab, or a `\n` to provide a new line. Either would have to be within the double quotes.

❹ Define the variable **konstant** and set the attributes to read only. Assign a value to it.

❺ Display the constant (read only) variable **konstant**.

❻ Try to assign another value to **konstant**. Because it is read only, you cannot change the value. You must unset the attributes with `typeset +r konstant`, then reset the attributes.

❼ Unset the attributes of **konstant**.

❽ Redefine the variable **konstant**, set the attributes, and assign it the new value of 96420.

❾ Display the value of **konstant**.

Example 7–6 More typeset Commands

```
$ typeset -L9 first=Ambrosia ❿
$ typeset -L3 mi="B." ⓫
$ typeset -L10 last="Salad" ⓬
$ print "$first$mi$last" ⓭
Ambrosia B. Salad
$ first="Leaf-lettuce" ⓮
$ print "$first$mi$last" ⓯
Leaf-lettB. Salad
$ typeset -lur ⓰
konstant=96420
lower='lowercase this'
uper='THIS IS AN UPPERCASE TEST'
$
```

❿ Define the variable **first**, and set the attributes to left-justified with a field width of 9. Assign a value to it.

⓫ Define the variable **mi** and set the attributes to left-justified with a field width of 3. Assign a value to it.

⓬ Define the variable **last** and set the attributes to left-justified with a field width of 10. Assign a value to it.

⓭ Display the values of **first**, **mi**, and **last**. The variables are not separated by spaces to ensure that the variable attributes are used to provide the spacing.

⓮ Assign a value to **first** which contains more characters than the attributes will allow.

⓯ Display the values of **first**, **mi**, and **last**. Notice that "Leaf-lettuce" is truncated to nine characters.

⓰ Display all variables that have the attributes of lowercase, uppercase, and read-only.

Korn Shell Options

The Korn shell, like any shell, is itself a program. Several options affect how the Korn shell processes commands. The Korn shell turns on some options, and you can specify some options. The Korn shell assigns defaults for other options, but you can change most of them.

When you first log in to a UNIX system with the Korn shell as your login shell, you are, by default, in the interactive mode of operation. The Korn shell turns on the interactive option, which allows the Korn shell to issue a prompt. When you execute a shell script, the interactive option is off during the execution of the script.

Setting Korn Shell Options

Use `set` to turn on, turn off, or display Korn shell options. The formats for the `set` command are:

```
set -o          # Display all options
set -o [option] # Turn on an option
set +o [option] # Turn off an option
```

The example shows how to turn off the `ignoreeof` option and turn on `gmacs`.

Example 7–7 Changing Korn Shell Options

```
$ set +o ignoreeof
$ set -o gmacs
```

Displaying Korn Shell Options

The following example shows two displays from the `set -o` command placed side by side. The one on the left shows the output of the `set -o` command prior to the modifications shown in Example 7–7. The one on the right is the output after the modifications have taken place, after `ignoreeof` has been turned off and `gmacs` has been turned on.

Some of the options shown in Example 7–8 are described when the Korn shell startup files are covered later in this chapter.

Learning Activity: Displaying and Setting Korn Shell Variables and Options

In this lab exercise, you will display default Korn shell variables, then set and display some user variables.

1. Display the Korn shell variables with the `set`, then the `typeset` commands (no options on the command line). Compare the displays. Note that the `set` command provides the names and values assigned to each variable. The `typeset` command displays the name and attributes of each variable.

Example 7–8 Displaying Korn Shell Options

```
Current option settings                Current option settings
(Before changes)                       (After changes)
-----------------------                -----------------------
allexport       off                    allexport       off
bgnice          on                     bgnice          on
emacs           on                     emacs           off
errexit         off                    errexit         off
gmacs           off                    gmacs           on
ignoreeof       on                     ignoreeof       off
interactive     on                     interactive     on
     .                                      .
     .                                      .
     .                                      .
trackall        on                     trackall        on
verbose         off                    verbose         off
vi              off                    vi              off
viraw           off                    viraw           off
xtrace          off                    xtrace          off
```

2. Display the variable **RANDOM** with `print $RANDOM`. Repeat the command numerous times. The **RANDOM** variable provides a random number between 0 and 32,767.
3. Assign variables as follows:

```
$ myname="Mickey Mouse"
$ address="Disneyland"
$ age="52"
$
```

4. Display the variables using the following commands (output is not shown):

```
$ print "My name is " $myname
$ print "My address is " $address
$ print "I am " $age "years old"
```

5. Clear the three variables **myname, address**, and **age** with the `unset` command. Display the variables and verify that the three user variables no longer exist.

   If you assign a variable a null value (`name=""`), it will show up as a valid variable. You have to use `unset` to delete the variable.
6. Display all Korn shell options with `set -o`
7. Locate an option that is turned off and turn it on with `set -o option_name`. Verify that the option is turned on. Turn the option off with `set +o option_name`. Verify that the option is turned off.
8. Check the status of the `emacs`, `gmacs`, and `vi` options. If none of them are turned on, turn one of them on. Verify that one of those three options is on. Now, turn on one of the other two options. Verify that the option is on, and that the option that was on, is now off.

Learning Activity Solution: Displaying and Setting Korn Shell Variables and Options

In this lab exercise, you will display default Korn shell variables, then set and display some user variables.

1. Display the Korn shell variables with the `set`, then the `typeset` commands (no options on the command line). Compare the displays. Note that the `set` command provides the names and values assigned to each variable. The `typeset` command displays the name and attributes of each variable.

```
$ set
ERRNO=0
FCEDIT=/usr/bin/ed
HOME=/usr/users/you
   .
   .
   .
$ typeset
export PATH
function ERRNO
   .
   .
   .
```

2. Display the variable **RANDOM** with `print $RANDOM`. Repeat the command numerous times. The **RANDOM** variable provides a random number between 0 and 32,767.

```
$ print $RANDOM
14768
$ print $RANDOM
22034
$ print $RANDOM
13180
$ print $RANDOM
1729
$
```

3. Assign variables as follows:

```
$ myname="Mickey Mouse"
$ address="Disneyland"
$ age="52"
$
```

No solution to this step is provided.

4. Display the variables using the following commands :

```
$ print "My name is " $myname
My name is  Mickey Mouse
$ print "My address is " $address
My address is  Disneyland
$ print "I am " $age "years old"
I am  52 years old
```

5. Clear the three variables **myname, address**, and **age** with the `unset` command. Display the variables and verify that the three user variables no longer exist.

   If you assign a variable a null value (`name=""`), it will show up as a valid variable. You have to use `unset` to delete the variable.

```
$ unset myname address age
$ print $myname $address $age

$
```

6. Display all Korn shell options with `set -o`

```
$ set -o
Current option settings
allexport       off
bgnice          on
emacs           off
   .
   .
   .
```

7. Locate an option that is turned off and turn it on with `set -o option_name`. Verify that the option is turned on. Turn the option off with `set +o option_name`. Verify that the option is turned off.

```
$ set -o allexport
$ set -o
Current option settings
allexport       on
bgnice          on
emacs           off
   .
   .
   .
$ set +o allexport
$ set -o
Current option settings
allexport       off
bgnice          on
emacs           off
   .
   .
   .
```

8. Check the status of the `emacs`, `gmacs`, and `vi` options. If none of them are turned on, turn one of them on. Verify that one of those three options is on. Now, turn on one of the other two options. Verify that the option is on, and that the option that was on, is now off.

```
$ set -o
Current option settings
allexport        off
bgnice           on
emacs            off
errexit          off
gmacs            off
   .
   .
   .
vi               off
   .
   .
   .
$ set -o emacs
$ set -o
Current option settings
allexport        off
bgnice           on
emacs            on
errexit          off
gmacs            off
   .
   .
   .
vi               off
   .
   .
   .
$ set -o vi
$ set -o
Current option settings
allexport        off
bgnice           on
emacs            off
errexit          off
gmacs            off
   .
   .
   .
vi               on
   .
   .
   .
```

Korn Shell Startup Files

Overview

When a user logs in, the default shell executes system and user login files to customize the user environment. The user environment is defined by variables and options. This section discusses examples of the Korn shell startup files, and some of the variables and options that can be set.

Korn Shell Startup Files

The Korn shell startup files are:

- `/etc/profile`

  When you log in, the `/etc/profile` script runs first. Every Korn shell user on the system shares this script; therefore, it is a good place for the system administrator to provide information that every user should use.

- `$HOME/.profile`

  The `$HOME/.profile` script runs next. Because the script is located in each user's `$HOME` directory, each user can customize the script to his or her own needs.

- The file indicated by the expansion of the **ENV** variable (we will use `$HOME/.kshrc`)

  Each time you start up a new Korn shell or run a Korn shell script, the system automatically invokes the environment script, the script file designated by the expansion of the **ENV** variable.

  Use the environment file to define any alias that you want available each time you start up the Korn shell or run a Korn shell script file.

  The assignment of the file name to the **ENV** variable is done in the `$HOME/.profile` file.

Korn Shell .profile Files

When you log in with the Korn shell as the default shell, the Korn shell executes the `/etc/profile` script if it exists, then executes your `$HOME/.profile` script file.

Use your `$HOME/.profile` script to:

- Set and export variables that you want to have set for all programs that you run

  Any variable to be passed to the child process must be designated for export. In Example 7–9, the variables, **PATH**, **ENV**, and **PS1** are designated to be exported to child processes.

- Set options that you want to apply to your login shell only
- Specify a script file to execute when you log out

The `trap` statement specifies some action to take when a condition arises. In the following example, the action is to execute a logout script file, `.kshlogout`. The condition that `trap` is looking for is a logout (EXIT).

The file name provided for the `trap` command is `.kshlogout` to ensure that there is no chance of confusion with `.logout`, which is the default script file invoked by the C shell at logout.

Korn Shell .profile File

A sample Korn shell `.profile` file is shown in this example. Each of the entries, except `trap`, is defined in Table 7–4.

Example 7–9 Korn Shell .profile File

```
CDPATH=.:..:$HOME
ENV=$HOME/.kshrc
EDITOR=emacs
FCEDIT=emacs
PS1='! ksh> '
export PATH ENV PS1
trap $HOME/.kshlogout EXIT
```

Note

If you use variables or the exclamation point when assigning values to the prompts, you must use single quotes. Also, the exclamation point is the Korn shell variable that designates the PID of the last process placed in the background. However, when used within a prompt, the exclamation point refers to the command number. Therefore, the prompt will be the command number followed by `ksh>` .

Korn Shell Environment Script File

Each time the Korn shell is invoked, it expands the **ENV** variable, and if non-null, executes that script file. We will use `$HOME/.kshrc` as defined in the `.profile` of Example 7–9.

Use the `.kshrc` environment script file to:

- Define aliases and functions that apply to interactive use and for scripts invoked by the Korn shell
- Set default options that you want to apply to all Korn shell invocations
- Set variables that you want to apply to the current Korn shell invocation

Some lines that might appear in a Korn shell environment file are shown in the table.

Table 7–6 Some .kshrc Commands

Use [1]	To
`set -o monitor`	Enable the Korn shell to display the job number when a background job is started, and display a status message when the job completes.
`set -o trackall`	Reduce the amount of time it takes the Korn shell to find some commands. The Korn shell creates an alias for the command the first time it is executed.
`set -o noclobber`	Prevent the Korn shell from writing over an existing file when using the > (redirection) operator.
`alias lal='ls -al'` [2]	Define an alias for getting a long list of all files.
`alias h=history`	Define an alias that provides a history of commands.
`alias c=clear`	Define an alias that clears the screen.
`HISTSIZE=64`	Define the maximum number of previously entered commands that you can access by using the Korn shell to 64. The default is 128.
`umask 022`	Create a file creation mask. `umask` is a 3-digit octal number that specifies the permissions the system should remove whenever the Korn shell or any of its child processes creates a file.

[1]Use set -o to turn on an option, set +o to turn off an option, set with no options to display all options, and set -o with no options to show only options that are turned on or off.

[2]Use single quotes if there are spaces or tabs in the alias definition.

Learning Activity: Startup Files

There is no learning activity for this section. After completing the section on one of the editors, create your own `.profile` and `.kshrc` startup files.

Summary

Shell Introduction

- The DEC OSF/1 available shells, their default prompts, and file names are:
  - C shell; `%`; `/bin/csh`
  - Bourne shell; `$`; `/bin/sh`
  - Korn shell; `$`; `/bin/ksh`
- Shells:
  - Provide a user interface to the DEC OSF/1 operating system
  - Provide a command language interpreter
  - Can be used as a programming language

Wildcard Operators

The asterisk (*) operator can be used to match any string of zero or more characters.

The question mark (?) operator can be used to match any single character.

The special meaning of wildcard operators can be **escaped** (turned off) by preceding the wildcard operator with a backslash (\) or by enclosing the string in single quotes.

The Korn Shell Environment

Variables perform an important role in defining the Korn shell environment. There are two types of variables:

- Reserved variables
- User-defined variables

The Korn shell has many reserved variables, and they fall into two classes:

- Variables whose values the Korn shell sets and updates
- Variables whose values the Korn shell does not automatically set

Korn shell variables are defined with an assignment statement with the format `name=value`.

Variables are displayed with the `set`, `typeset`, or `print $name` commands.

The `typeset` command is used to set the attributes of a variable.

Korn Shell Startup Files

- The system and user login scripts are used to customize the following environment features:
  - Terminal characteristics
  - Search path
  - Shell variables
  - Default file creation permissions for new files with `umask`
  - Checking for mail
  - Command aliases
  - History variables
  - The `trap` command
- The following files are used to set up shell and environment variables and options:
  - `/etc/profile`
  - `$HOME/.profile`
  - The environment file designated by the expansion of the **ENV** variable

8

Introduction to UNIX Editors and the emacs Editor

UNIX Editors and emacs

Introduction

The UNIX operating system offers a choice of editors. Each available editor is introduced, but only the `GNU emacs` and `vi` editors are covered in detail in this course. The `GNU emacs` editor is referred to as `emacs` throughout the course.

The `emacs` and `vi` editors are covered sufficiently to allow you to edit files, recover a file you may have been editing when the system had an interruption, and other basic editing commands. Lab exercises allow you to become more familiar with the `emacs` and `vi` editors, including the `emacs` tutorial.

Objectives

To effectively edit files on a UNIX system, you should be able to:

- Describe the advantages or disadvantages of the various UNIX editors
- Select an appropriate editor for a particular task
- Use the `emacs` editor to create and modify files

Resources

- *OSF/1 User's Guide*
- *GNU Emacs Reference Card*
- DEC OSF/1 `man` pages

Introduction to the UNIX Editors

Overview

The topics that follow cover the emacs editor in more detail. The vi editor is covered later in the course.

UNIX Editors

The following list introduces each of the UNIX editors.

- ed: A line-editing program that copies a file into a temporary buffer and operates on one line at a time. Any changes are made to the temporary buffer and not to the original file. The edited buffer is written to the original file, or any named file with the write command.

  There is a restricted version of ed, red, which is available with the restricted Bourne shell, rsh. With red, the user is able to edit files in the current directory or /tmp.

  ed is the only editor available in single-user mode, and therefore should be taught in a DEC OSF/1 System Management course.

- ex: A line-oriented text editor similar to ed, but more powerful. The ex command provides a subset of the vi screen editor commands, and provides multiline displays with access to a screen-editing mode.

  Use ex to access windows of text and scroll through the text. Use an undo command to reverse the effects of a command.

  The vi editor can be invoked from ex, providing its screen editing and cursor movement features.

- edit: A line editor designed for beginner users. It is a simplified version of ex.

- sed: A streamlined editor that modifies lines from the specified file. The modifications are determined by sed script commands on the command line or from a script file.

- emacs: One of the most powerful editors available to UNIX users. More than an editor, emacs is a complete working environment. From emacs you can get a listing, edit, rename, delete, or copy files, send and receive mail, check your spelling, execute a single shell command, or run a shell in an emacs window.

  The GNU emacs editor was originally written for the PDP-10 by Richard Stallman, the founder of the Free Software Foundation.

  emacs has an extensive Help facility which provides apropos help, and help on commands or functions. It also has a built-in tutorial which you will use in the learning activities.

- `vi`: `vi`, short for visual editor, is a full screen editor, and is probably the most useful standard text editor on your system. Unlike `emacs`, it is available in almost identical format on almost every UNIX system.

  If you supply more than one file name on the command line, `vi` edits each file in the order specified.

  We will discuss `vi` in greater detail, and you will have an opportunity to learn more about `vi` in the learning activity.

- `view`: A read-only version of `vi` that allows file protection while browsing. Changes can be made to the buffer, but the buffer cannot be written back to the file.

The emacs Editor

Overview

The emacs editor is the most powerful standard editor in the UNIX world today, and is a complete working environment. With proper training and experience, you can use emacs to:

- Get a listing
- Edit, rename, delete, and copy files
- Compile programs
- Send and receive mail
- Check for proper spelling
- Execute a single shell command
- Run a shell in an emacs window

This section covers only the basics, but you can still do useful work with emacs. You can use emacs to edit files, including searching and replacing text, and use multiple windows. You will even be shown how to define keys, should the need ever arise.

The emacs editor is not difficult to learn, however it does take practice, and with the excellent Help facility, you become more proficient as you use it.

The original emacs was written by Richard Stallman for the PDP-10 as a set of macros for the TECO line editor. TECO is an older interactive line text editor that runs on TOPS–10 (for the PDP-10), OS/8, RSTS, RSX–11, and OpenVMS systems. emacs stands for "Editing Macros." GNU emacs comes from the Free Software Foundation and the GNU project (GNU's Not UNIX), a complete operating system that Stallman and his associates are generating.

We will refer to GNU emacs as emacs throughout this course.

As you review the section on the emacs editor, notice that most commands are provided. Do not let the number of commands discourage you; you will not need all of them. In some cases, there is even more than one way to do something. Do not try to learn more than one way to do the same task.

While reading the material, look at the commands and note those that you think will be most useful, then try them out as directed in the learning activities.

emacs Files and Buffers

The emacs editor does not edit the actual file, but places the contents of the file into a temporary emacs buffer and edits that buffer. All edits are made to the buffer, not the original file. emacs does not modify the original file on disk until the file is written by emacs. The buffer name is usually the same as the file name.

When you use emacs, there can be many buffers open at one time. You can edit multiple files, each of which has its own buffer, or you can open buffers at your discretion. emacs itself opens the **scratch** buffer each time you edit a file. If you get help, emacs generates another buffer, the **Help** buffer.

emacs Modes

The emacs editor has different operating modes. The modes allow emacs to operate differently based on the contents of the file being edited. The meanings of a few keys are changed, and comments are delimited differently. There are major modes and minor modes.

Major Modes

Some examples of major modes are:

- Fundamental mode: The default mode, no special operation
- Text mode: For writing text
- Indented text mode: Indents all the text you write
- Picture mode: For creating simple line drawings
- C mode: For writing C programs
- FORTRAN mode: For writing FORTRAN programs
- emacs LISP mode: For writing emacs LISP functions

emacs selects the major mode based on the file extension or the contents of the file. If it cannot determine a major mode, the default major mode is the fundamental mode.

There are other major modes, and as you gain experience editing with emacs, you will learn more about them.

Minor Modes

There are also minor modes that can be turned on or off within a major mode. They define a particular aspect of emacs behavior; for example, fill mode means that emacs should wrap long lines where appropriate.

The minor modes are:

- Abbrev mode: Allows the use of word abbreviation
- Fill mode: Enables word wrap
- Overwrite mode: Allows typing over characters instead of inserting them
- Autosave mode: Saves your file automatically at predetermined intervals

Using emacs

Starting emacs

You can edit a file on a character cell terminal or from a DEC OSF/1 workstation.

From a DEC OSF/1 workstation, you can start emacs within the current DECterm or Xterm window, or in a separate X window as a background process. If you are using a character-cell terminal, you cannot create a separate screen window for emacs.

Starting emacs on a Terminal or in a Workstation's Current Window

The format of the emacs command is:

emacs [-*option*] *filename*

Some of the options you may want to use are:

- +*number*: Start the edit on the line specified by *number*
- -q: Do not load an initialization file
- -u *user*: Load *user*'s initialization file
- -nw: Do not use the special interface to X. When used to invoke emacs from an Xterm window, display is in the present window

To start emacs from a terminal without using an initialization file, use:

```
emacs -q junkfile
```

To start emacs on a DEC OSF/1 workstation, with the editing session in the current window, use:

```
emacs -nw junkfile
```

Starting emacs in a DECterm or Xterm Window

To run emacs in its own X window, the format of the emacs command is:

emacs [-*option*] *filename*

Some of the options you may want to use with a workstation are:

- -wn *name*: Specifies the name of the emacs window.
- -i: Displays the "kitchen sink" bitmap icon (looks like a deep sink) when iconifying the emacs window.
- -w *geometry*: Sets the emacs window's width, height, and position. The default width and height are specified in characters, and is 80 by 24.
- -fg *color*: Sets the color of the text on color displays. The colors can be found by displaying the /usr/lib/X11/rgb.txt file.
- -bg *color*: Sets the color of the window's background on color displays
- -bd *color*: Sets the color of the window's border on color displays

- `-cr` *color*: Sets the color of the text cursor on color displays
- `-ms` *color*: Sets the color of the mouse cursor on color displays
- `title` *name*: Sets the title of the `emacs` window to `name`
- `icon` *name*: Sets the icon name for the `emacs` window icon

If you try to set color values while using a black and white display, the window's characteristics will default as follows:

- Black foreground
- White background
- Gray border
- Black text and mouse cursors

For more information on `emacs` windows, see the `emacs man` pages (At the shell prompt, enter **man emacs.**)

The most efficient way to run `emacs` is probably in an X window. You can use the mouse to change from window to window.

The emacs Screen

Before learning any `emacs` commands, you must learn a few things about the `emacs` screen.

When you edit a file, there is the workspace, which is the area of the screen for the text, a mode line, located two lines up from the bottom of the screen, and the minibuffer area.

The cursor marks the place where the next character will be entered, and the position of the cursor is referred to as **point** or **dot** throughout `emacs` help. With `emacs`, you can start typing right away. `emacs` does not have separate modes for inserting text and giving commands.

A typical mode line is shown in the example.

Example 8–1 Mode Line

```
--**-Emacs: junkfile   (Fundamental)----------------------Top--------
```

The mode line can contain the following information, starting at the left of the screen:

- Two asterisks (**): If present, they indicate that the buffer you are editing has been modified since the last time it was saved.
- The word `Emacs`: You are using `emacs`.
- The name of the buffer you are editing: `junkfile`. The buffer name is usually the same as the file you are editing.
- One or more words in parentheses: The first word within the parentheses is the major mode, in this case `Fundamental`.

- Any further information within the parentheses indicates minor modes that are turned on. They will show up as:
  - Fill
  - Abbrev
  - Ovwrt

Autosaving is turned on by default, but does not show up in the mode line. An autosave is done every time 300 characters are typed. Autosaving can be toggled with Esc X `auto-save-mode`.

- The information on the right of the mode line indicates where you are in the file, which can take on the following attributes:
  - All: The entire buffer is shown on the screen
  - Top: At the top of the buffer
  - Bot: At the bottom of the file
  - A percentage (34%): How far you are into the file

Below the mode line is the **minibuffer** area. This is the area where `emacs` echoes the commands you enter, and where you specify file names for `emacs` to find and values for `emacs` to search for.

Every buffer has a mode line, therefore, if you have two windows, you have two mode lines.

Introducing emacs Commands

Before talking about specific `emacs` commands, you should know something about `emacs` commands in general.

There are hundreds of `emacs` commands, but you need to learn only those that are necessary. `emacs` commands have long command names. However, for the commands you use most often, you do not have to remember the command name. Just type a key sequence. Because a command is executed by one particular key sequence, we say that a key is bound to a command.

If you know the name of the command, and cannot remember the key sequence, you can always use the command name to execute the command. It is much slower though, and as you use `emacs`, you will learn the most frequently used key sequences.

When `emacs` was designed, the commands used most frequently were bound to the keys easiest to reach. Also, you may change the key bindings if you choose.

The most commonly used commands (cursor movement commands) are bound to Ctrl/char, where char is the character you press while holding down the Ctrl key.

Slightly less used commands are bound to Esc char.

Note

The Free Software Foundation emacs documentation (GNU emacs Manual, GNU emacs Reference Card, and emacs help) refers to the Meta key, abbreviated by M. The Meta key does not exist on many terminals, or is hidden. For all practical purposes, the Esc key is equivalent to the Meta key. If F11 does not work as Esc on your terminal, use Ctrl/ [.

Other commands are bound to a sequence of Ctrl/X followed by some character, for example, Ctrl/X A.

You can execute any command by typing Esc X followed by the command name. There are some commands that do not have key sequences bound to them, and this is the normal way of executing these commands. An example is the goto-line command which would be executed as:

Esc X **goto-line** *line-number*

There are also specialized commands bound to Ctrl/C char, but they are related to the special modes, and are not discussed in this course.

Note

The emacs editor is not, in general, case sensitive. Ctrl/a and Ctrl/A will do the same thing.

Leaving and Temporarily Suspending emacs

If, while you are editing a file with emacs, you discover that you need to print the file, copy it to another file, or some other operation that makes it necessary to suspend emacs. You can write the file, then suspend emacs with Ctrl/Z as was previously mentioned.

This places you back at the shell prompt. (You can get back into emacs by entering fg.) Suspending emacs is very useful, but you must be careful and not start another session of emacs on the same file. You could corrupt the file.

Note

Users of systems such as OpenVMS, which use Ctrl/Z for **exit**, should be careful when using UNIX systems: the Ctrl/Z key will make emacs disappear from the screen, but it is only suspended, and the edited file has not been written out.

An emacs session is terminated by Ctrl/X Ctrl/C. If you have made changes to the file, emacs asks you if you want to write the file. If you type y, emacs writes the file, then exits. If you type n, emacs responds with:

```
Modified buffers exist; exit anyway? (yes or no)
```

If you type y or n instead of the yes or no as requested, emacs responds with:

```
Please answer yes or no.
```

If you answer yes, you exit emacs and any changes you made during the editing session do not become permanent.

If you answer no, the normal emacs session continues just as if you had never typed the Ctrl/X Ctrl/C.

emacs Help

emacs Help Facility

The emacs editor has one of the most useful, and complete, Help facilities of any text editor. emacs help consists of:

- Completion, which helps you complete typing the names of things
- The Help key, which allows you to get help on many emacs topics

File Name Completion

You can type in the shortest, most unambiguous sequence of characters and tell emacs to figure out the rest of the name. The name must consist of enough characters, starting from the beginning, to distinguish it from any other possibilities. emacs allows completion for:

- File names (in a given directory)
- Buffers
- Command names
- Variable names

In most cases, when you are prompted for a name in the minibuffer, emacs provides completion. Type some of the characters of a name, then use one of three characters to tell emacs to help complete the name.

- Tab: Complete the name as far as possible
- Spacebar: Complete the name to the next punctuation character
- ?: List any choices of possible names in the **Completion** window

For example, assume you have a C program called poker.c in your default directory. When you type Ctrl/X Ctrl/F to visit the file, emacs prompts for the file name. See what the different responses would be if you type "pok":

- Tab: emacs completes the file name poker.c if there are no other files that begin with "pok"
- Spacebar: emacs completes the file name to poker
- ?: Generates a **Completion** window and provides the names of all files that start with pok

When asked for a file name, you must type enough characters to make the file name unique. If you have not supplied enough characters to make the file name unique, emacs will supply you with all of the file names that you may choose from in the present directory that match the provided letters.

If, on the other hand, you have a series of files named something like `file`, `file1`, `file2`, `file3`, and so forth, and you type in "fil Tab, `emacs` will provide the "e." If you again press Tab, `emacs` will tell you that the file name is complete, but not unique. Type ? and `emacs` will show you all the file names that match the characters entered so far.

Getting emacs Help

Ctrl/H is the `emacs` Help key. Type Ctrl/H followed by a single character which represents the type of help you need. The following table lists the `emacs` help commands.

Table 8–1 emacs Help on Commands

Key Sequence	Command Name	Function
Ctrl/H	help-command	Provides a list of help options
Ctrl/H A *expr*	command-apropos	Searches for commands that match *expr*, and if any are found, provides the name of the command, key binding, if any, and the first line of documentation in a help window
Ctrl/H B	describe-bindings	Provides a list of key bindings
Ctrl/H C *key-sequence*	describe-key-briefly	Provides the name of the command executed by *key-sequence*
Ctrl/H F *cmd-name*	describe-function	Provides a description of the command provided by *cmd-name*
Ctrl/H Ctrl/H or Ctrl/H ?	help-for-help	Provides a list of letters that are valid help commands
Ctrl/H K *key-sequence*	describe-key	Provides the name of the command executed by *key-sequence*, and provides a description of the command
Ctrl/H L	view-lossage	Displays the last 100 keystrokes
Ctrl/H M	describe-mode	Displays information about current major mode
Ctrl/H S	describe-syntax	Describes the syntax table for this buffer
Ctrl/H V *variable-name*	describe-variable	Provides a description and the value assigned to the variable *variable-name*
Ctrl/H W	where-is	Provides the key binding for the command

If you cannot remember the name of a command, use command apropos, Ctrl/H A. Provide `emacs` with a term that you think might do what you want to do. For example, suppose you want to copy some text and cannot remember the name of the command or what to type to do it. Type: Ctrl/H A *copy*

If you think you remember the command name, or part of the command name, type Ctrl/H F and provide the first part of the command when `emacs` asks for it. Then, use ? to see if there are any commands that fit.

Unfortunately, help on function (Ctrl/H F) does not provide you with the key sequence needed to execute the command. Use Ctrl/H W, and provide emacs with the name of the function. You will be provided a key sequence if one is bound to the function.

If you cannot remember what to type for help on a particular topic, type Ctrl/H Ctrl/H. emacs prints the message:

```
A B C F I K L M N S T V W C-c C-d C-n C-w.  Type C-h again for more help:
```

You would type the character that represents the type of help you want, equivalent to typing Ctrl/H *character* in the first place.

If you need more hints, type another Ctrl/H and emacs will display a **Help** window that provides a short description of all the help options.

The emacs editor provides help on the Free Software Foundation's policies on the copying and distribution of emacs. There is also an emacs tutorial, which you have already used in the learning activity, and an information documentation reader, which is not covered. Upon starting the information documentation reader, help is provided.

All help commands are listed in Table 8–1, emacs Help on Commands, or in Table 8–2, emacs Help on General Information.

Table 8–2 emacs Help on General Information

Key Sequence	Command Name	Function
Ctrl/H I	info	Starts the information documentation reader
Ctrl/H T	help-with-tutorial	Provides an emacs tutorial
Ctrl/H N or Ctrl/H Ctrl/N	view-emacs-news	Provides a history of changes in emacs since the last major revision
Ctrl/H Ctrl/C	describe-copying [1]	Provides information about how you can redistribute copies of emacs
Ctrl/H Ctrl/D	describe-distribution [1]	Provides information about ordering emacs from the Free Software Foundation
Ctrl/H Ctrl/W	describe-no-warranty [1]	Describes the (non)-warranty for emacs

[1]Help on these topics is not available unless the *emacs* source files are installed on the system.

Using emacs Help

This section directs you through a series of commands that allow you to use the emacs Help facility. Assume that you want to write or save a file you have been editing, but cannot remember which command to use to save files or even exit emacs.

1. Edit the file file1.

2. Enter Ctrl/H A. When emacs places the expression "Command apropos (regexp):" in the minibuffer, reply with a word that aptly describes the information you are looking for. Then press Return. For this case, type Ctrl/H A write. After looking at that, repeat with Ctrl/H A save. With each of them you will see a list of emacs functions that have something to do with write and save.

   Notice that you can use Ctrl/X W to write a file, Ctrl/X Ctrl/S (and possibly Esc S) to save a buffer, Ctrl/X Ctrl/C to save buffers and exit emacs, and Ctrl/X S to save some of the buffers.

3. If you think you know the name of a command, but do not know which key strokes invoke the command, use Ctrl/H W followed by the name of the command. Assume that you do not even know the complete command name, just the first part. For example, you know that the command to exit starts with save. Enter Ctrl/H w save. At this point, press Tab. emacs will complete the command as far as it can. It does not do much, only providing the hyphen (-). Type a question mark, or another Tab, and emacs provides you with a list of all the commands that complete what you have typed. This provides a list of possibilities, one of which is "save-buffers-kill-emacs." You only have to type enough to make the selection unambiguous, type another Tab, and emacs fills in the rest. Follow with a Return, and emacs provides the key sequence to execute the command in the minibuffer area, in this case, Ctrl/X Ctrl/C.

4. If you want to know which command is executed when you type a key sequence, use Ctrl/H C or Ctrl/H K followed by typing in the key sequence. Ctrl/H K provides more information about the function. For example, type Ctrl/H C Ctrl/X Ctrl/C and emacs responds in the minibuffer with: " Ctrl/X Ctrl/C runs the command save-buffers-kill-emacs."

5. If you want to know all of the key bindings, use Ctrl/H B and emacs provides a complete list. Try it. If you want to print out this list, you can copy the buffer into a file and write the file. Or, easier yet, write the buffer to a file with Ctrl/X Ctrl/W filename.

6. Enter Ctrl/H Ctrl/H and look at what you get, then enter another Ctrl/H.

7. Exit without saving any changes you may have made.

Basic Editing

Reading, Inserting, and Saving Files

Once you have started to edit a file with emacs (emacs *filename*), the commands in Table 8–3 can be used to read in a file other than the one you are editing, to insert a file into the current buffer, or save a modified file.

Note

There are the special circumstances with Ctrl/S and Ctrl/Q.

If you are on DEC OSF/1 workstations, and use emacs with the -nw option (no windows), the Ctrl/S and Ctrl/Q keys do not work as defined in emacs. You may either create the .emacs initialization file shown below in your login directory, or you may select the **Keyboard** option in the window **Options** menu and turn off "Ctrl-Q, Ctrl-S = Hold." .emacs initialization file.

The .emacs initialization file defines new keys, as shown in the following table.

Command Name	Default Key Binding	New Key Binding
isearch-forward	Ctrl/S	Ctrl/ \
save-buffer	Ctrl/X Ctrl/S	Esc S
toggle-read-only	Ctrl/X Ctrl/Q	Ctrl/X Q

Table 8–3 Reading, Inserting and Saving Files with emacs

Key Sequence	Command Name	Function
Ctrl/X Ctrl/C	save-buffers-kill-emacs	Exits emacs. If buffers have been modified, emacs will ask if you want to save them
Ctrl/X Ctrl/F *file*	find-file	Finds *file* and reads it
Ctrl/X I *file*	insert-file	Inserts *file* at cursor
Ctrl/X Ctrl/S [1]	save-buffer	Saves buffer in the file with the same name as the buffer
Ctrl/X Ctrl/W *file*	write-file	Writes buffer to *file*
Ctrl/X Ctrl/R	find-file-read-only	Edits file but does not save without confirmation
Ctrl/X Ctrl/V *file*	find-alternate-file	Reads an alternate file, *file*, replacing the present file. If the present buffer has been modified, emacs asks if you want to kill it anyway

[1]If Ctrl/S is used for flow control, this key sequence will not work. The *.emacs* initialization file redefines this command as Esc S.

When emacs is instructed to execute a command that involves getting or saving a file, it places a message in the minibuffer that corresponds to the command executed. Also, the default directory is shown in anticipation of the addition of the remainder of the file name.

For example, the following would be shown in the minibuffer for these commands:

- Ctrl/X Ctrl/F *file*: Find file: ~/
- Ctrl/X Ctrl/V *file*: Find alternate file: ~/
- Ctrl/X I *file*: Insert file: ~/
- Ctrl/X Ctrl/W *file*: Write file: ~/

The first time you save a file, emacs creates a backup file, which is a copy of the original file. The name of the backup file is the same as the file you are editing, with a tilde (~) appended to the name. If you were editing `junkfile`, the backup file is `junkfile~`.

Fill Mode

If you want emacs to provide the carriage return at the end of the line, type the following sequence:

Esc X **auto-fill-mode**

Fill mode will toggle each time you execute the command. You can turn it on or off at will by typing the above key sequence. When fill mode is turned on, it will show up in the mode line.

Caution

Do not turn on fill mode until instructed to do so in the emacs tutorial. If you do, part of the tutorial will not work properly.

Repeating Commands

Very often you want to repeat a command a number of times, especially cursor motion. emacs provides two ways to repeat commands.

- Precede the command with Esc *number*, where *number* represents the number of times you want to repeat the command. For example, if you gave the command to move down the file by one line, and preceded the command with Esc 100, the cursor would move 100 lines down. A negative number reverses the direction.
- Precede the command with Ctrl/U *number*, or use Ctrl/U by itself. When you use Ctrl/U with a number, it works the same as Esc number. If you use Ctrl/U without a trailing number, emacs automatically inserts the constant four.

  When the key sequence Ctrl/U Ctrl/U is executed, emacs multiplies 4 x 4, repeating the command 16 times. You can use this to generate multiples of 4, such as 4, 16, 64, and 256.

Moving the Cursor

Some of the most frequently used cursor positioning commands are shown in Table 8–4.

Table 8–4 Cursor Movement Commands

Key Sequence	Command Name	Function
Ctrl/F or Right arrow	forward-char	Moves forward one character (right)
Ctrl/B or Left arrow	backward-char	Moves backward one character (left)
Ctrl/P or Up arrow	previous-line	Moves to previous line (up)
Ctrl/N or Down arrow	next-line	Moves to next line (down)
Esc F	forward-word	Moves one word forward
Esc B	backward-word	Moves one word backward
Ctrl/A	beginning-of-line	Moves to beginning of line
Ctrl/E	end-of-line	Moves to end of line
Ctrl/V	scroll-up	Moves forward one screen
Esc V	scroll-down	Moves backward one screen
Esc >	end-of-buffer	Moves to end of file
Esc <	beginning-of-buffer	Moves to beginning of file
Esc 0 .. Esc 9	digit-argument	Provides a numeric argument to a command
Ctrl/U *n*	universal-argument	Repeats the next command *n* times, or if *n* is not present, four times
Ctrl/L	recenter	Redraws the screen with the current line in the center
—	goto-line *n*[1]	Goes to line *n* in file
—	goto-char *n*[1]	Goes to character *n* in file

[1]Executed with Esc X *command-name*

If the cursor is at the end of one line, the next-line command moves the cursor to the beginning of the next line. If the cursor is at the beginning of a line, the previous-line command will move the cursor to the end of the previous line.

The exceptions to the above are:

- If the cursor is at the beginning of the file, the previous-line command does nothing
- If the cursor is at the end of the file, the next-line command opens a line just as Return does.

Using the Cursor Motion Commands

Create a long file from the `emacs man` page to practice `emacs` commands. Type the following command:

```
man emacs | sed 's/. Ctrl/V Ctrl/H//g' > man-emacs
```

Edit the file man-emacs. Remember that if you are on a workstation and plan to use the -nw option, or are logged in from a remote host, you must copy the .emacs initialization file to your home directory.

1. Edit the file man-emacs using the command **emacs man-emacs** (with whatever options you want to use).
2. Using 17 Ctrl/N s, move the cursor down to the 17th line.
3. Move the cursor forward to the "is" with Ctrl/F s.
4. Move it back to the "E" in "Emacs" with Ctrl/B s.
5. Move the cursor forward to the space between "the" and "author" with Esc F s.
6. Move it back to the beginning of the line with Esc B s.
7. Move the cursor to the end of the line with Ctrl/E.
8. Move the cursor to the beginning of the line with Ctrl/A.
9. Move the cursor two screens beyond the present position with Ctrl/V s.
10. Display the previous screen of text with Esc V.
11. Enter Ctrl/L to position the present line at the center of the display.
12. Take note of what is on the bottom line of the display. Move the cursor off the bottom of the screen (slowly) with Ctrl/N s. The line that was at the bottom of the screen should end up in the middle of the display.
13. If the cursor is not at the beginning of the line, move it there with Ctrl/A. Then, move the cursor forward with Ctrl/U Ctrl/F. Try Ctrl/U Ctrl/U Ctrl/F. What is the difference?
14. Try Esc 10 Esc F to move the cursor forward 10 words.
15. Use Esc < to move the cursor to the top of the file.

Marking Text

If you want to delete a phrase, several sentences, or parts of two paragraphs, using the line-deletion command may be unwieldy. It is often easier to mark the area you want to delete. To define a region, use a secondary pointer called **mark**. In GNU emacs, the mark is not seen on the screen.

You can set the mark at one end of the region with Ctrl/@ or Ctrl/Spacebar, then move the cursor (point) to the other end of the region. Ctrl/Spacebar is much easier to use. The other way requires the use of Shift, Ctrl, and @ all at the same time (rather awkward).

Because you cannot see the location of the mark, and because the text in the marked region is not highlighted, you should verify that the region is really marked properly. Use Ctrl/X Ctrl/X to swap the positions of *point* and *mark*. By swapping their positions,

you can verify that *mark* is really where you wanted it to be. Repeating the Ctrl/X Ctrl/X swaps *point* and *mark* back where they were, but you do not have to, the kill-region command (Ctrl/W) deletes text between *mark* and *point*, and it does not matter which is at the beginning of the region. You can repeat the Ctrl/X Ctrl/X as many times as you want without changing the region you have marked.

If the *mark* has not been set, the cursor will not move with the execution of Ctrl/X Ctrl/X.

Also, once *mark* is set, you can make edits to the file without disturbing the *mark*. It will stay set until it is reset.

The *mark* is reset by an explicit setting (Ctrl/Spacebar), and there are commands, mostly cursor movement commands, which set *mark*. Any time a command resets *mark*, a message will show up in the minibuffer indicating that the mark has been set (Mark set).

The following table shows the commands used to mark text prior to killing or copying the region.

Table 8–5 Marking Text

Key Sequence	Command Name	Function
Ctrl/Spacebar or Ctrl/@	set-mark-command	Sets *mark* at cursor position
Ctrl/X Ctrl/X	exchange-point-and-mark	Swaps the positions of *point* and *mark*
Esc @	mark-word	Sets mark after end of next word
Esc H	mark-paragraph	Sets mark at end of a paragraph and moves cursor to beginning of paragraph
Ctrl/X Ctrl/P	mark-page	Marks a region between form feeds, or beginning or end of buffer and a form feed
Ctrl/X H	mark-whole-buffer	Marks entire buffer

Deleting Text

The emacs editor provides many ways to delete text, with the easiest being the use of the delete key, <⌫. The emacs delete commands are shown in the following table.

Table 8–6 Delete, Kill, and Yank Commands

Key Sequence	Command Name	Function
Ctrl/D	delete-char	Deletes character under cursor
<X	delete-backward-char	Deletes previous character
Esc Z *char*	zap-to-char	Kills up to next occurrence of *char*
Esc D	kill-word	Deletes the word the cursor is on
Esc <X	backward-kill-word	Deletes previous word
Ctrl/K	kill-line	Deletes from cursor to end of line
Esc K	kill-sentence	Deletes sentence the cursor is on
Ctrl/X <X	backward-kill-sentence	Deletes the previous sentence
Ctrl/Y	yank	Restores the last deletion
Esc Y	yank-pop	Recovers killed text that is no longer the most recent kill. Used with the kill ring, and can only be used after a Ctrl/Y
Ctrl/W	kill-region	Deletes a marked region
Esc W	copy-region-as-kill	Saves the region as if deleted, but does not delete it. Copies text from one place to another
Ctrl/X Ctrl/O	delete-blank-lines	Deletes all but one consecutive blank line
Esc Spacebar	just-one-space	Deletes all but one space before and after point

For the kill-word and backward-kill-word, only the characters between the cursor and the end or beginning of a word are deleted. For the kill-sentence and backward-kill-sentence, only characters between the cursor and end or beginning of the sentence are deleted.

The kill-line command does not kill the carriage return. It takes two Ctrl/Ks to kill the line of text and the carriage return.

Any deleted text may be immediately placed back in the buffer with yank, Ctrl/Y.

Moving Text

Very often you discover that you have placed text in the wrong location, and want to move it. This is easy to do using the deletion commands to delete the text you want to move, moving the cursor to a new location, then using *yank* to place the deleted text in the new location.

For example, to move a region of text, use the following sequence of commands:

1. Place the cursor at the beginning of the region.
2. Set *mark* using Ctrl/Spacebar or Ctrl/@. `emacs` responds with "Mark set" in the minibuffer region.
3. Move the cursor to the end of the region you want to move, or use a mark paragraph, mark page, or mark buffer command.

4. Use Ctrl/X Ctrl/X to verify that *mark* is in the right place.
5. Delete the region of text with Ctrl/W.
6. Move the cursor to where you want the text to be placed.
7. Yank the text into the buffer with Ctrl/Y.

Note

If you place the text in the wrong place, immediately delete it again with Ctrl/W. When you yank the text in, the mark is still set around the region, so it may be immediately deleted again.

Copying Text

There are times when you have typed some text that you expect to use over and over again. You do not want to mark the text, delete it, yank it back to its original location, then move the cursor and yank it back in again. It is easier to copy the text from one location to another.

Note

If you are in a read-only buffer, you cannot use delete, but you can copy the text.

The command to copy text is Esc W, and works the same as Ctrl/W, except that the marked text is not deleted.

The command for text copying is shown with the deletion commands in Table 8–6.

To copy a region of text from one location to another, use the following sequence of commands:

1. Place the cursor at the beginning of the region.
2. Set *mark* using Ctrl/Spacebar or Ctrl/@. `emacs` responds with "Mark set" in the minibuffer region.
3. Move the cursor to the end of the region you want to copy. You can also use the mark paragraph, mark page, or mark buffer commands.
4. Use Ctrl/X Ctrl/X to verify that *mark* is in the right place.
5. Copy the region of text with Esc W.
6. Move the cursor to where you want the text to be copied.
7. Yank the text into the buffer with Ctrl/Y.

Using the Deleting, Moving, and Copying Text Commands

If you exited the emacs editor, edit the file, *man-emacs* again. Then execute the following commands.

1. If the cursor is not at the top of the buffer, move it there with Esc <.
2. Move the cursor down 17 lines with successive Ctrl/Ns. The line starts with "GNU Emacs."
3. Delete the line with Ctrl/K. Notice that the line feed is not deleted.
4. Yank the line back with Ctrl/Y.
5. Move the cursor to the beginning of the line with Ctrl/A.
6. Delete the line, and line feed with two Ctrl/Ks.
7. Yank the line back into the text with Ctrl/Y.
8. Move the cursor back up to the line with Ctrl/P.
9. Delete the line one more time with Ctrl/K, but do not delete the line feed.
10. Move the cursor to the beginning of the next paragraph with Ctrl/Ns.
11. Delete the first word in the sentence with Esc D.
12. Replace the word with Ctrl/Y.
13. Delete the entire next word with successive Ctrl/Ds.
14. Enter the command Ctrl/Y. What was yanked back into the text? Note that only deletions of one word or more are placed into the kill buffer. The characters that were deleted with Ctrl/D are lost.
15. Move to the beginning of the next paragraph with a Ctrl/A and Ctrl/Ns.
16. Set the mark with Ctrl/Spacebar.
17. Use Ctrl/Ns to move the cursor to the blank line after the paragraph.
18. Enter Ctrl/X Ctrl/X to swap the point and mark to ensure that the mark is where you thought it was. Swap point and mark back with another Ctrl/X Ctrl/X if you want.
19. Delete the paragraph with Ctrl/W. Use Ctrl/W, not Esc W.
20. Yank the paragraph back with Ctrl/Y.
21. Move the cursor to the next screen with Ctrl/V and find the first blank line. Insert a blank line with Ctrl/M or Return. Yank the paragraph back into the text with Ctrl/Y.
22. Move the cursor to the first character of the next paragraph and set the mark with Ctrl/Spacebar.

23. Move the cursor to the blank line after the paragraph. Enter Ctrl/X Ctrl/X to make sure the mark is where you think it should be.
24. Copy the paragraph with Esc W. Use Esc W, not Ctrl/W.
25. Move the cursor down a paragraph, open a line, then yank the paragraph back with Ctrl/Y.
26. Exit the emacs editor with Ctrl/X Ctrl/C. Do not save the file. You do not want to save any modifications made to the file as you will use it again later.

Learning Activity: Editing with the emacs Editor Part 1

Part of this learning activity is accomplished using the emacs tutorial subset of the emacs Help facility.

There is a problem using key sequences containing Ctrl/S and Ctrl/Q if you are editing with emacs on a workstation using the `-nw` options, or if you are logged in remotely. In either of these cases, you must use the `.emacs` initialization file.

If you **do** use the `.emacs` initialization file, the new key bindings for those commands are shown in the table.

Table 8–7 New Key Bindings When Using the *.emacs* Initialization File

Command Name	Default Key Binding	New Key Binding
isearch-forward	Ctrl/S	Ctrl/ \
save-buffer	Ctrl/X Ctrl/S	Esc S
toggle-read-only	Ctrl/X Ctrl/Q	Ctrl/X Q

The emacs Tutorial

Use the emacs editor to edit a file (any file will do), as `emacs junkfile`. When the editor has completed initialization, and the mode line is on the bottom of the screen, start the emacs tutorial by typing Ctrl/H T.

Follow the steps in the tutorial. Remember you may be using Ctrl/ \ for incremental searches instead of Ctrl/S. Therefore, when the tutorial says to use Ctrl/S, use Ctrl/ \.

Note

If F11 does not work as escape on your terminal, use Ctrl/ [.

If you want to take a break, or quit, with intentions of resuming where you left off, place some type of marker at the place you want to come back to. An odd combination of characters, such as qq, is easy to remember. Use Ctrl/X Ctrl/C to exit the emacs editor.

When emacs asks if you want to save the file ~/*TUTORIAL*, answer "yes." emacs saves the file in your directory.

When you want to resume the tutorial at the point you left off, edit the file *TUTORIAL* that you saved (**emacs TUTORIAL**). When the mode line is on the screen, use incremental search (Ctrl/\ or Ctrl/S) to find the "qq", or whatever you used as a marker. For example, you might type: Ctrl/\ qq. Use Esc to terminate the search when emacs finds the "qq." You are now ready to resume the tutorial.

Note

Work with the tutorial until you get to the header, "**FILES.**" At that point, put the tutorial aside and resume the course.

No solution required.

The emacs Kill Ring

When you delete and copy text with the Ctrl/W or Esc W commands, the deleted or copied region of marked text is placed in the kill ring. The Ctrl/Y yank command takes the deleted or copied text from the kill ring and places it back into the editing buffer.

The kill ring stores the last 30 deletions. Any deletions of one word or more are placed in the kill ring. A yank command removes the most recently deleted text from the kill ring. If you have made several deletions, you have the capability to retrieve all the deletions from the kill ring.

Note

The number of locations in the kill ring may be determined by executing the help variables command Ctrl/H V **kill-ring-max**. The contents of the kill ring may be viewed with the help variable command Ctrl/H V **kill-ring**.

If successive kill-word (Esc D), kill-line (Ctrl/K), or other kill commands are executed, with no intervening cursor movement commands, each deletion is appended to the previous deletion so that one yank (Ctrl/Y) will retrieve all the successive deletions just as if it were one deletion.

The Esc Y (yank-pop) command is used to retrieve successive deletions from the kill ring. It can only be used after a yank (Ctrl/Y), and there can be no intervening commands. If you try to use the Esc Y without a previous yank, emacs will tell you that the "Previous command was not a yank."

When you complete a yank-pop (Esc Y) operation, emacs leaves the kill ring pointer at the text last yanked. If you were to move the cursor around, then do another yank (Ctrl/Y), you would get the same text as the last yank-pop.

The pointer is moved back to the beginning of the kill ring the next time you delete text that is greater than a character.

As was mentioned earlier, a Ctrl/Y yanks the most recently deleted text. An Esc Y (yank-pop) removes that text from the buffer and gets the next-most-recent text from the kill ring. Another Esc Y removes that text and gets the next-next-most-recent text from the kill ring. When you have yank-popped sufficient times to get the text you wanted from the kill ring, just stop executing Esc Y commands.

Using the emacs Kill Ring

We will create a smaller file for this practice session. Type the following:

```
$ cat > lines
line 1
line 2
line 3
line 4
line 5
line 6
line 7
line 8
line 9
Ctrl/D
```

Use emacs to edit the file lines.

1. Use Ctrl/K to kill every line in the file, including the line-feeds. Do not type any other commands between Ctrl/Ks.
2. Enter the Ctrl/Y command. Notice that you get back all 9 of the lines that were killed. As you execute, kill, or delete commands in succession, each kill is appended to the previous kill, as long as there are no intervening commands.
3. Move the cursor to the top of the buffer with Esc <.
4. Kill line 1 and the line feed.
5. Move the cursor to the beginning of line 3. Kill line 3 and its line feed.
6. Move the cursor to the beginning of line 6. Kill line 6 and its line feed.
7. Without moving the cursor, yank line 6 back into the buffer with Ctrl/Y.
8. Move the cursor up to the beginning of line 4 and enter Ctrl/Y. Notice that you get line 6 again. Enter Esc Y. Line 6 should disappear and you get line 3 back. By entering the Esc Y after a Ctrl/Y, you pop the kill ring.

9. Move the cursor to the beginning of line 2 and enter Ctrl/Y. This time you get line 3 because that is where the kill ring pointer is. Enter Esc Y and get line 1 back.
10. Now try the same thing again, but this time introduce an intervening kill.
11. Move the cursor to the top of the buffer; then kill lines 1, 3, and 6 with the line feeds.
12. Yank line 6 back right after killing it.
13. Move the cursor to the beginning of line 4 and yank (Ctrl/Y); then yank-pop (Esc Y).
14. Now, move the cursor to the beginning of line 9 and kill it. Do not kill the line feed.
15. Move the cursor to the beginning of line 2 and yank. You get line 9 back, and there is no line feed between line 9 and line 2. Do a yank-pop and you get line 6 back. Two more yank-pops and you have line 1 back. The difference is caused by the kill of line 9. It set the kill ring pointer back at the beginning, whereas the previous time the kill ring pointer was not reset to the beginning.
16. Exit the editor without saving changes.

Stopping Commands and Undoing Changes

All too often you will make a mistake on a command you are typing, or execute a command that you did not want to execute. A command can be stopped with the keyboard-quit command, Ctrl/G. `emacs` indicates that a command has been stopped with "Quit" in the minibuffer.

Commands that stop a command, or undo the results of a command, are shown in the table.

Table 8–8 Stopping and Undoing Commands

Key Sequence	Command Name	Function
Ctrl/G	keyboard-quit	Aborts the current command
Ctrl/X U	advertised-undo	Undoes the last edit
Ctrl/_	undo	Undoes the last edit
—	revert-buffer [1]	Restores buffer to the state it was in when the file was last saved (or autosaved)

[1]Executed with Esc X *revert-buffer*

You can abort a partially typed command. For example, if you typed Ctrl/X and decided that is not part of the command you want to use. The Ctrl/X is displayed in the minibuffer, and you

cannot type ⌫ to delete the Ctrl/X. If you do, emacs would execute the backward-kill-sentence command (Ctrl/X ⌫). Use Ctrl/G to clear a partially typed command from the minibuffer. There is no response from emacs other than to clear the minibuffer.

If you have just made a big mistake, or you suddenly realize that you made one a few commands previously, you can restore the buffer to the original state before the mistake, using the undo (Ctrl/X U or Ctrl/_) commands.

By typing Ctrl/X U repeatedly, you can gradually work your way back to a point before your mistake. This is convenient if your mistake was four or five commands previous, but the further back the error is, the less helpful Ctrl/X U may be.

emacs allows all changes made in the text of a buffer to be undone up to an amount of change of about 8000 characters.

If the undo command is not useful, you can always restore the buffer to an earlier state if the buffer is not a temporary buffer. If you want to get the buffer back to the same state as the file stored on disk, use Esc X revert-buffer.

The revert-buffer command has a drawback. emacs creates an autosave file every 300 keystrokes. This file may be identified in your directory by the pound sign (#) preceding and after the file name. For example, if you were editing the file junk, an autosave would create the file *#junk#* in the default directory. These files show up at the beginning of a directory listing.

If an autosave file has been created since the last time you have saved your edit, emacs asks you if you want to revert from the autosave file.

```
Buffer has been autosaved recently. Revert from autosave file? (y or n)
```

If you type y, emacs prepares to restore your file from the autosave file. If you type n, emacs prepares to restore your file from the file on disk. Before it does, it verifies your intentions by asking:

```
Revert buffer from file filename?  (yes or no)
```

The name of the file is the name of the autosave file or the name of your original file, depending upon the answer to the previous question. Type yes to restore the file, no if you have changed your mind.

Search and Replace Operations

emacs Search Operations

Like any good editor, emacs provides search commands, in fact there are several different search commands:

- Incremental search: As soon as you type a character, emacs starts the search for the next occurrence of that character. As you type in more characters, it searches for that combination of characters. If the search fails to find the character string you have typed in, emacs places "Failing I-Search" in the minibuffer. At that time, the cursor is at the place where it found as much of your search string as it could.

  If you make a mistake and type a character incorrectly, wait until emacs finds the search string, or indicates that it cannot find it. Then use the ⌫ to erase the incorrect characters, and type in the correct ones.

  For example, say you wanted to search for "found" but instead entered the search string "foul" and the search failed at the "foul" in "foul-mouthed." You can use ⌫ to delete the "l" from the search string, or you can use Ctrl/G, which removes all characters from the search string that were not found.

  When the cursor is at the desired position, the search operation is normally terminated with Esc, but any non-typing character not used in any of the searches will also work. Any cursor movement commands that do not use the Esc key will terminate the search, for example Ctrl/B.

  Typing Ctrl/G cancels the search if typed at the conclusion of a successful search. If this is the case, the cursor is moved back to the position it was in when the search started.

- Nonincremental or simple search: Provide emacs with a search string and it will find the next occurrence, if it exists. The search terminates at the found string or fails if the string is not found.

- Word search: Like a simple search, except that it searches only for full words and phrases. For example, if you are searching for the word "mental," you will not have to worry about finding the word "incremental" or "fundamental." If you type a string of words, separated by spaces, the string can be found even if there are multiple spaces, new lines, or other punctuation separating the words.

  By default, emacs searches are not case sensitive. If you search for "home," emacs will find "Home," "HOME," and even "hoMe."

- Regular expression search: This can be very complicated, and is not covered in this course.

Use the regular expression search to search for patterns that may vary slightly. For example, if you want to find all instances of R0 and R1 in a file, you could search for the regular expression R[01]. As you have not encountered regular expressions yet, do not worry about them.

The following table provides the commands needed to implement the incremental, simple, or word searches.

Table 8–9 Search Commands

Key Sequence	Command Name	Function
Ctrl/S *string* [1]	isearch-forward	Starts incremental search forward. Follows command with search string. The command also finds next occurrence of *string* in the forward direction
Ctrl/R *string*	isearch-backward	Starts incremental search backward. Follows command with search string. The command also finds next occurrence of *string* in the reverse direction
Esc	—	Exits a successful incremental search
Ctrl/G	keyboard-quit	Cancels incremental search and returns to starting point or deletes any characters typed after a search failed
<X	—	Deletes incorrect character in search string
Ctrl/S Esc *[string]* [1]	—	Starts nonincremental search forward
Ctrl/R Esc *[string]*	—	Starts nonincremental search backward
Ctrl/S Esc Ctrl/W *[string]* [1]	word-search-forward	Starts word search forward
Ctrl/R Esc Ctrl/W *[string]*	word-search-backward	Starts word search backward

[1]If the *.emacs* initialization file is being used, use Ctrl/\ instead of Ctrl/S.

When doing a nonincremental or word search, when the Ctrl/R or Ctrl/S is typed to start the command, you will see "*I-Search:*" in the minibuffer. When Esc is typed, the minibuffer will then show "*Search:*". Additionally, when doing a word search, when Ctrl/W is typed, the minibuffer will show "*Word Search:*".

A failing search will show up in the minibuffer as follows:

- I Search: *Failing I-search: string*
- Search: *Search failed: "string"*
- Word Search: *"\\bstring\\b"*

After an incremental search has found the search string, another incremental search for the same string may be started by typing Ctrl/S or Ctrl/R.

A nonincremental or word search may be repeated by typing Ctrl/S Ctrl/S or Ctrl/R Ctrl/R. Additional searches may be done by just typing Ctrl/R or Ctrl/S. You can repeat the search in either direction, or change directions.

Search and Replace

You may want to combine a search with a replace operation. For example, you may have problems with your fingers when typing certain words, like "teh" instead of "the," or often misspell a word like "recieve" instead of "receive." The hard way is to search for the word and change it manually when found. emacs provides search and replace operations that let you make changes like this very quickly.

emacs has two types of search and replace operations, a simple search and replace, and a query replace.

Simple Search and Replace

If you want to replace every occurrence of one string with another, use the simple search and replace. The command format is shown in Table 8–10.

Because search and replace works only in the forward direction, if you want to make sure you replace every occurrence in the file, move the cursor to the top of the file before executing the command. If you do not know that there are changes to be made in the reverse direction, and there are none in the forward direction, emacs indicates that it has completed the operation (Done) very quickly.

Table 8–10 emacs Search and Replace Commands

Key Sequence	Command Name	Function
Esc X **replace-string** Return *{old-string}* Return *{new-string}* Return [1]		Searches for all occurrences of *old-string* and replaces them with *new-string*
Esc %	query-replace	Searches for a text string. Replacement is dependent upon the response at each found string

[1]There is no command that can be bound to a key for this key sequence.

When you give the command Esc X *replace-string*, the emacs responses and expected inputs in the minibuffer area are:

Replace string: *{old-string}* Return
Replace string *{old-string}* with: *{new-string}* Return

Using the Search Commands

Edit the file `man-emacs` once more.

1. Move the cursor to line number 7 in the buffer, the line that begins with "NAME" You can use Ctrl/N s or Esc 7 Ctrl/N to get there.
2. Use incremental search to find the string "Emacs" with Ctrl/S or Ctrl/\, whichever is appropriate. Type in the letters slowly and observe the cursor.
3. When the search is complete, repeat the Ctrl/S or Ctrl/\ a few more times to find the next two occurrences of the search string.
4. Now enter Ctrl/R s to get the cursor back to the first occurrence of the search string.
5. Terminate the search with Esc.
6. Move the cursor to line 7 once more.
7. Repeat the search for "Emacs," but this time make a mistake while typing in the search string. For example, you might type in "Exact." Look at the minibuffer area and notice the error condition. Delete the erroneous characters with individual <X s or with Ctrl/G. Then complete the search and terminate.
8. Use a nonincremental search forward to find the string "Free Software" Notice the difference in the search. The `emacs` editor waits until the entire search string has been entered, and Return pressed before executing the search.

   Enter the command as follows:
   Ctrl/S Esc `Free Software` Return
   or
   Ctrl/\ Esc `Free Software` Return
9. Exit the `emacs` editor without saving changes.

Query Replace

If you are not sure that you want to replace each and every occurrence, use query replace. It lets you decide on a case-by-case basis. The responses used during a query replace are shown in Table 8–11.

Let's say that you want to replace every occurrence of rec**ie**ve with rec**ei**ve. There are three sequential `emacs` responses in the minibuffer area when you type Esc %. They are listed below, with your response.

Query replace: *{old-string}*
Query replace **recieve** with: *{new-string}*
Query replacing **recieve** with **receive**:

While you are doing a query replace, you often see something else that needs fixing. You might try to remember the error, or you might even write it down, so you can go back and fix the error later. When you have finished, if you did not write it down, many times you will not remember what it was you were going to fix.

With the emacs query replace recursive edit option, you do not have to wait. You can fix the problem right away and not have to worry about remembering it.

After typing Ctrl/R to start a recursive edit, use the normal emacs commands to move the cursor around and make edits. When the edits are complete, use Esc Ctrl/C or Ctrl/] to get out of recursive edit and either return to the query replace, or exit both. If you resume the query replace, emacs puts the cursor right back where it was when you started the recursive edit.

While the recursive edit option is active, brackets will be placed around the modes on the mode line, for example, *[(Fundamental)]*.

Table 8–11 Responses to Query Replace

Response	Function
Spacebar or Y	Replaces *old-string* with *new-string* and searches for the next instance of the string
N or <X	Does not replace, goes to next instance
.	Replaces the current instance and quits
,	Replaces the current instance, but does not move on
!	Replaces all remaining occurrences without asking
^	Backs up to the previous instance
Esc	Exits query replace
Ctrl/R	Enters recursive edit
Esc Ctrl/C	Exits recursive edit
Ctrl/]	Exits recursive edit and exits query replace

Using Buffers and Windows

Using Multiple Buffers and Windows

One of the most useful features of emacs is the ability to edit multiple buffers at one time, and to display more than one buffer on the screen at one time.

In addition to the buffers that are a working copy of the files you are editing, you can generate temporary buffers that have no relationship to the files. emacs itself creates buffers, and generally these buffers have names of the format ***buffer name***, such as ***Help***, ***scratch***, and ***Buffer List***.

There is no limit to the number of buffers you can have, but most of the time only one or two buffers will be displayed. But, even though you cannot see them, the other buffers are still active.

A window is an area on the screen in which a buffer is displayed. Each window has its own mode line which separates the windows from each other. You can have multiple windows, as many as you want, but there comes a point where the windows become so small that they are just not useful.

When you have multiple windows:

- They can be the same buffer or different buffers
- You can move from one window to another, and therefore from one buffer to another
- You can move from one place in one buffer to another place in the same buffer
- You can quickly and easily compare buffers or copy from one buffer to another

Buffer Commands

When you initially start editing a file with emacs, you provide the file name of the file you want to edit. The file name becomes the buffer name. If you want to edit a second file, use the Ctrl/X Ctrl/F *{filename}* command. emacs creates a new buffer for its working copy of the file.

If you have been editing for an extended period of time, you may have many different buffers. You must be able to move between buffers, and often, you will forget the names of buffers, so you must be able to determine which buffers are available, and manipulate the buffers. The buffer commands are listed in the following table.

Table 8–12 Buffer Commands

Key Sequence	Command Name	Function
Ctrl/X B *buffer-name*	switch-to-buffer	Moves to the buffer specified by *buffer-name*
Ctrl/X Ctrl/B	list-buffers	Displays the buffer list
Ctrl/X K *buffer-name*	kill-buffer	Deletes the buffer specified by *buffer-name*
Ctrl/X S	save-some-buffers	Asks if you want to save each of the modified buffers
—	kill-some-buffers [1]	Asks before deleting a buffer
—	rename-buffer [1]	Changes the name of the buffer to the name specified

[1]Executed with Esc X command-name

If you know the name of the buffer you want to edit, use Ctrl/X B *buffer-name*.

Getting a List of Buffers

If an emacs editing session has been active for any length of time, it is possible that the list of buffers has grown so long you cannot keep track of all of them. You can get a list of buffers with Ctrl/X Ctrl/B. emacs creates another window on the screen and creates the buffer **Buffer List**. Example 8–2 shows a sample buffer list and Table 8–13 provides the meaning of the *MR* (Modified or Read Only) column.

Example 8–2 Buffer List

```
 MR Buffer          Size  Mode           File
 -- ------          ----  ----           ----
. % docs            6994  Fundamental    /usr/users/you/docs
 *  contacts        737   Fundamental    /usr/users/you/contacts
.*  proj-dlist      31    Fundamental    /usr/users/you/project/distribution-list
 *  distribution-list  28 Fundamental    /usr/users/you/distribution-list
    ordering-emacs 12253 Fundamental    /usr/users/you/ordering-emacs

    *Help*         517   Fundamental
 *  *Buffer List*  252   Buffer Menu
    *scratch*      0     Lisp Interaction
    .
    .
    .
--%%-Emacs: *Buffer List*          (Buffer Menu)----All-----------------------
```

Table 8–13 Buffer Listing MR Column

If MR contains a	The buffer is
. (Period)	Displayed
*	Modified
%	Read only
D	Marked for deletion
>	Marked for display
S	Marked for saving

Using the Buffer List

The buffer list is more than a display of buffer names. You can use the buffer list to:

- Display a buffer
- Delete a buffer
- Save a buffer

Use Ctrl/X o to move the cursor to the window containing the buffer list (Example 8–2); then use any command shown in Table 8–14 to mark buffers for saving or deletion, or to select buffers for display. When you mark a buffer for saving or deletion, they are not deleted until you enter X.

You can move the cursor around in the buffer list using normal emacs cursor commands, and the ⌫ is equivalent to a Ctrl/P followed by a U.

Table 8–14 Buffer List Commands

Type	To	It will happen
Ctrl/N	Move the cursor to the next buffer in the list	Immediately
Ctrl/P	Move the cursor to the previous buffer in the list	Immediately
D or K	Mark a buffer for deletion	When you press X
S	Mark a buffer to be saved	When you press X
U	Unmark a buffer	Immediately
X	Execute other one-letter commands (D or K, S, and U)	Immediately
<X	Unmark the previous buffer in the list	Immediately
~	Mark the buffer as unmodified	Immediately
1	Display this buffer in a full screen	Immediately
2	Display this buffer and a marked buffer in two windows	Immediately
F	Replace the buffer list with this buffer	Immediately
O	Replace the other window with this buffer	Immediately
M	Mark buffers to be displayed in windows	When you press Q
Q	Display all buffers marked with *m*	Immediately

The Q command divides the screen into the number of windows designated by the marked buffers plus the buffer where the cursor is located. If you want two buffers, mark one buffer with **M**, and place the cursor on the line containing the name of the other buffer you want. Or, you can mark one buffer with **M**, place the cursor on the line containing the name of the other buffer you want, and type **2**.

Also, when using the **2**, only one other buffer can be marked. If more than one buffer is marked, you will get two buffers in two windows, but you are not guaranteed to get the buffers you want.

Saving Buffers

We have already discussed how to save buffers with:

- Ctrl/X Ctrl/S (save-buffer) or Esc S if you are using the *.emacs* initialization file
- Ctrl/X Ctrl/C (save-buffers-kill-emacs)

You can also save buffers with Ctrl/X S. `emacs` cycles through all modified buffers, and asks you if you want to save the buffer. The buffer will be saved if you answer "y."

Killing Buffers

You can kill the buffer you are currently editing, with Ctrl/X K. If you kill a buffer, it has no affect on the original file, it remains unchanged.

If you do not supply a buffer-name, emacs prepares to kill the default buffer and responds with:

Kill buffer: (*default buffer-name*)

- Press Ctrl/G to cancel the command.
- Press Return to kill the buffer immediately if it has not been modified. If the buffer has been modified, emacs informs you that it has been modified and asks if you want to kill it anyway.
- If you provide a buffer-name to the Ctrl/X K key sequence, emacs prepares to kill the named buffer as it would the default buffer.

Use Esc X kill-some-buffers to selectively kill buffers. emacs sequences through the buffer list, and informs you if a buffer has been modified to provide you with the opportunity to keep the buffer, and asks if you want to kill the buffer. If you respond with "yes," emacs gives you another chance to change your mind by informing you that the file has been modified, and asks you if you want to kill the buffer anyway.

If you decide that want to get out of the kill-some-buffers command, you can use Ctrl/G at any time.

emacs even lists its internal buffers, allowing you to kill them if you want. If you **do** kill an emacs internal buffer by mistake, emacs will re-create the buffer when it needs it.

Renaming Buffers

Let's create a hypothetical situation where you have multiple directories that contain a file by the same name, *distribution-list*. If you edit this file from one directory, then use Ctrl/X Ctrl/F to bring in the file with the same name from another directory, emacs names the buffer for the second file *distribution-list<2>* to indicate that it is the second buffer with this name. You may want to change the name of the buffer to make it easier to keep track of them while you are editing the buffers. Use Esc X *rename-buffer {buffer-name}* to rename the buffer. Even with the different buffer names, emacs keeps the corresponding file names straight. After you have completed buffer modifications, when you save the files, emacs saves them to the proper files.

Read-Only Buffers

There will be times when you want to copy text from one file to another without changing the file you are copying from. It is very easy to hit keys mistakenly and unknowingly add text. You may want to make a buffer read only, preventing the possibility of this from happening. You can then copy into the kill ring and yank the text to the other file (remember, you cannot delete text in a read-only file). To set the default buffer to read only, use:

Ctrl/X Ctrl/Q

If you are using the `.emacs` initialization file, you would use Ctrl/X Q, or you can use Esc X *toggle-read-only*.

Working with Windows

Windows are areas on the screen where `emacs` displays the buffers you are editing. You can have many windows on the screen at one time, each displaying a different buffer, or all of the windows displaying the same buffer, but at different locations in the buffer.

`emacs` windows cannot overlap, so as you increase the number of windows, the size of the windows becomes smaller and smaller.

Splitting Windows

You can split windows horizontally, or vertically. We will not discuss vertical windows in this course.

To make a window taller, use Ctrl/X ^. The other windows will shrink. A window is made shorter with Esc X *shrink-window*. `emacs` shrinks the current window by one line and makes another window larger. Obviously, reducing the size of a window one line at a time is tedious. To speed up the process, prefix the commands with Ctrl/U, which will increase operation by a factor of four. Therefore, Ctrl/U Ctrl/X ^ increases the height of the current window by four rows.

Moving the Cursor Between Windows

The cursor can be in only one window at any time. It therefore must be moved from window to window with the Ctrl/X O `other-window` command. There is no way to designate which window to move the cursor to. It moves from the top to the bottom of the screen and back to the top window again. You can add an Esc number, or Ctrl/U, prefix to Ctrl/X O to designate the number of windows to move the cursor through. Esc 3 Ctrl/X O moves the cursor to the third window after the present window.

When you are editing a file, and have generated the second window, use Ctrl/X Ctrl/F *filename* in the desired window to get a second file into that window. By using Ctrl/X O you can now move from one window to the other at will. You can "kill" or copy text in one file and "yank" it into the other.

Using multiple buffers is particularly useful for reading one file while editing a second file, copying text from one file to another, or editing a source program while looking at errors in the log file.

Deleting Windows

To delete the window the cursor is in, use Ctrl/X 0 (zero). The buffer is not deleted, it is still there and you can switch back to it at any time. To delete all windows except the window the cursor is in, use Ctrl/X 1 (one).

Window Commands

The following table provides a summary of the emacs window commands.

Table 8–15 emacs Window Commands

Key Sequence	Command Name	Function
Ctrl/X 2	split-window-horizontally	Divides the current window horizontally into two windows
Ctrl/X O	other-window	Moves cursor to the other window; if there are more than two windows, moves down the screen
Ctrl/X 0	delete-window	Deletes the current window
Ctrl/X 1	delete-other-windows	Deletes all windows but the one the cursor is in
Ctrl/X ^	enlarge-window	Makes window taller
—	shrink-window [1]	Makes window shorter
Esc Ctrl/V	scroll-other-window	Scrolls the other window; if there are more than two windows, scrolls the window below the window containing the cursor

[1]Executed with Esc X *shrink-window*

Using the Buffer and Window Commands

For this practice session, we will create several small files. Type the following:

```
$ cat > file1
file1
Ctrl/D
$ cat > file2
file2
Ctrl/D
$ cat > file3
file3
Ctrl/D
$ cat > file3
file3
Ctrl/D
$ cat > file4
file4
Ctrl/D
$ cat > file5
file5
Ctrl/D
```

This section directs you through a series of commands to edit multiple files, move around the various buffers, determine which buffers are available, and work with the buffer list and buffer list commands.

1. Edit *file1*.

2. When the editor has initialized, edit *file2* with Ctrl/X Ctrl/F *file2*. Repeat for *file3* and *file4*.
3. Split the screen into two windows with Ctrl/X 2.
4. Edit *file5* in the window the cursor is in with Ctrl/X F *file5*.
5. You should have two windows on the screen, with *file4* in the bottom window and *file5* in the top window.
6. Add some text to the file, just so you know that it has been modified.
7. Move the cursor to the other window (*file4*) with Ctrl/X O and type in some text.
8. Move the cursor back to the window that contains *file5* with Ctrl/X O.
9. Go to one window mode with only *file5* on the screen with Ctrl/X 1.
10. Use Ctrl/X Ctrl/B to display the buffer list. `emacs` opens another window in which to display the buffer list. Also, notice that files 5, 4, and the buffer list are marked with an asterisk (*) to signify that they have been changed. Also, *file5* is marked with a period (.) which indicates that the buffer is displayed.
11. Use Ctrl/X O to move the cursor to the window containing the buffer list. Move the cursor down the list with Ctrl/N to *file3* and mark it for deletion by typing a **d**. Note that the cursor moves to the next buffer, and `emacs` displays a **D** prior to the buffer marked for deletion.
12. Move the cursor to *file1* and also mark it for deletion with **d**.
13. You change your mind and decide that you do not want to delete *file3*. Use Ctrl/P to move the cursor back up to *file3* and enter *U* to unmark the file.
14. Type an **x** to delete the marked files.
15. Mark files 4 and 3 for display by entering an **M** while the cursor is on the line for the file. `emacs` provides a > to signify that the file is marked for display. Place the cursor on the line for *file2*. Enter **q** to display three windows with files 2, 3, and 4 in the windows.
16. Repeat Ctrl/X O a few times to move the cursor around the buffers just to see how it moves.
17. Stop the cursor in the middle buffer and split that window into two windows with Ctrl/X 2. You could edit this file in two different places if you desired.
18. Place the cursor in the top window and enter Ctrl/X 1. Notice that all the other windows disappear.
19. Split the window into two windows with Ctrl/X 2. Now increase the size of the top window with Ctrl/U Ctrl/X ^.

20. Decrease the size of the window with Ctrl/U Esc X `shrink-window`.
21. Use Ctrl/X B to switch to another buffer.
22. Copy the contents of one file into the other file at the end of the file.
23. Exit `emacs` without writing any of the files.

Correcting Mistakes

Transposing Characters and Words

One of the most common typing mistakes is the transposing of characters. emacs provides commands to transpose characters, words, sentences, or paragraphs, as shown in the table.

Table 8–16 Transposing Characters and Words

Key Sequence	Command Name	Function
Ctrl/T	transpose-chars	Transposes two characters
Esc T	transpose-words	Transposes two words
Ctrl/X Ctrl/T	transpose-lines	Transposes two lines
—	transpose-sentences [1]	Transposes two sentences
—	transpose-paragraphs [1]	Transposes two paragraphs

[1]Executed with Esc X *command-name*

Which characters, words, lines, and so forth are transposed depends upon cursor position, and there are differences. It will take practice to become familiar using the cursor position. You will probably find that you seldom have to transpose lines, sentences or paragraphs. See the following table.

Table 8–17 Cursor Position for Transposition Commands

To transpose two...	Place the cursor...
Characters	On the space following the characters, if at the end of the line, otherwise on the last of the two characters
Words	Between the two words
Lines	On the second line
Sentences	On the first sentence
Paragraph	On the first paragraph

Capitalization Commands

Another mistake we often make is in capitalization. emacs provides special commands for fixing capitalization problems, as shown in the following table.

Table 8–18 Capitalization Commands

Key Sequence	Command Name	Function
Esc C	capitalize-word	Capitalizes the first letter of word
Esc U	upcase-word	Uppercases word
Esc L	downcase-word	Lowercases word
Esc - Esc C	capitalize-word	Capitalizes previous word, negative argument
Esc - Esc U	upcase-word	Uppercases previous word, negative argument
Esc - Esc L	downcase-word	Lowercases previous word, negative argument
—	capitalize-region [1]	Capitalizes region
Ctrl/X Ctrl/U	upcase-region	Uppercases region
Ctrl/X Ctrl/L	downcase-region	Lowercases region

[1]Executed with Esc X *capitalize-region*

When capitalizing a word, if the cursor is not on the first letter, the letter under the cursor will be capitalized. If the cursor is then moved to the first character, Esc C will capitalize the first letter and change any other capitalized characters in the word to lowercase.

Using the Mouse with emacs

If you are using a DEC OSF/1 workstation, when you edit a file with emacs, an emacs session is started. The session window behaves the same as the window on a character cell terminal, except that you have the advantage of using the mouse.

If you edit different files with emacs from two Xterm sessions, you can use the mouse to cut and paste between emacs sessions.

If you are using a workstation with a windowing system and mouse, you can move the cursor, cut and paste, and manipulate emacs windows with the mouse.

For the select buffer pop-up menu, hold down the keys and mouse button until the menu is present on the screen. You have to keep the Ctrl and Shift keys held down until the menu is present. Then, move the mouse to the desired file and click any mouse button.

Mouse help provides a stack (or card deck) of pop-up menus that duplicate the emacs Ctrl/H help commands. Move the mouse to the title bar of the desired card and you will see the entire list of options. Move the mouse down the options to the desired option and click any mouse button. When prompted in the minibuffer, type in the necessary information. Upon pressing Return, emacs provides a help window.

emacs Mouse Commands

The following table provides direction for using the mouse with emacs.

Table 8–19 Using the Mouse with emacs

Mouse Click	Command Name	Equivalent emacs Command
MB1	x-mouse-set-point	Moves cursor to location mouse points to
MB2 or Shift/MB3	x-paste-text	yank, Ctrl/Y
MB3 or Shift/MB2	x-cut-text	copy-region-as-kill, Esc W
Ctrl/MB2	x-cut-and-wipe-text	kill-region, Ctrl/W
Ctrl/MB3	x-mouse-select-and-split	split-window-vertically, Ctrl/X 2
Ctrl/Shift/MB3	x-mouse-keep-one-window	delete-other-windows, Ctrl/X 1
Ctrl/Shift/MB1	x-buffer-menu	Creates a pop-up menu with buffer names, similar to Ctrl/X Ctrl/B, list-buffers. Click any mouse button on the buffer name to select that buffer.
Ctrl/Shift/MB2	x-help	Creates a stack of pop-up menus for emacs help

Learning Activity: Editing with the emacs Editor Part 2

Part of this learning activity is accomplished by completing the tutorial that you started earlier in the previous learning activity. The rest of the learning activity provides practice with other useful commands that are not covered in the tutorial.

The emacs Tutorial

To resume the tutorial, at the point you left, edit the file *TUTORIAL* that you saved. When the mode line is on the screen, use incremental search (Ctrl/\ or Ctrl/S) to find the "qq," or whatever you used as a marker. For example, you might type: Ctrl/\ qq. Use Esc to terminate the search when emacs finds the "qq." You are now ready to resume the tutorial.

When you have completed the tutorial portion of the learning activity, exit emacs with Ctrl/X Ctrl/C. This time, say "no" when emacs asks if you want to save the file *TUTORIAL*. You can always get back into the tutorial with Ctrl/H T.

Other Useful Commands not Covered in the Tutorial

Again, use the emacs tutorial, but this time as a text file while we look at some other commands that were not covered in the tutorial. Use emacs to edit the empty file, junkfile, then start the emacs tutorial. If you already have a file with the name junkfile, use mv to rename the file or remove it with rm. Also, if you have saved the file, *TUTORIAL*, do not use it in this part of the learning activity. You may have made changes to the file.

After you have started emacs on junkfile, start the emacs tutorial with Ctrl/H T. You will only have the *TUTORIAL* buffer visible.

Cutting and Pasting from One File to Another

1. Create a second window with Ctrl/X 2.
2. Use Ctrl/X B to switch to the buffer junkfile, which is the default. You now should have junkfile in one window, and the tutorial in the other window.
3. Switch the cursor to the window containing the tutorial with Ctrl/X O.
4. Use incremental search, Ctrl/S or Ctrl/ \ , to find the text string "*several windows*."
5. Use Ctrl/A to terminate the search and move the cursor to the beginning of the line. Note that you do not have to terminate the search with Esc then use Ctrl/A to move the cursor. Most cursor motion commands will terminate a search.
6. Set the mark with Ctrl/Spacebar.
7. Move the cursor to the end of the paragraph with Esc].
8. Verify that the mark is set where you thought it was with Ctrl/X Ctrl/X. If mark and point are not where you thought, set the mark again.
9. Delete the marked text with Ctrl/W.
10. Move the cursor to the other window with Ctrl/X O.
11. Yank the text into the buffer from the kill ring with Ctrl/Y.

Query Replace and Recursive Edit

1. Move the cursor to the other window (*TUTORIAL*) with Ctrl/X O.
2. Move the cursor to the top of the tutorial with Esc <.
3. Use Esc % to query replace "M-" with "Esc/".
4. Enter Y or Spacebar to replace the first four occurrences of "M-". When you reach the fifth "M-", you should be on the line that begins with "If moving by characters is too slow...". Assume that we notice a mistake and want to replace "characters" with "one character."
5. To enter recursive edit, type Ctrl/R.
6. Move the cursor to the beginning of the word "character" with **isearch-backward**, Ctrl/R, or by normal cursor movement. Add the word "one" and a space before "character", then delete the "s" from the end of "characters". It should now read "If moving by one character is too slow..."
7. Exit recursive edit with Esc Ctrl/C. Notice that the cursor moves to the next "M-", right where you left off.

8. Enter [Y] or [Spacebar] to replace a few more occurrences of "M-".
9. Enter [.] to exit the query replace.
10. Restart the query replace and try out some of the other responses.

Working with the Buffer List

Exit the editing session with [Ctrl/X] [Ctrl/C]. Save `junkfile`, but not *TUTORIAL*. Use `cp` to copy `junkfile` to `junk1`, `junk2`, `junk3`, `junk4`, and `junk5`. Again use `emacs` to edit `junkfile`.

1. Open a second window with [Ctrl/X] [2].
2. Find and read file `junk1` with [Ctrl/X] [Ctrl/F] `junk1`.
3. Repeat the preceding step, finding and reading `junk2`, `junk3`, `junk4`, and `junk5`.
4. Move the cursor to the bottom of `junk5` and add some text.
5. View the buffer list with [Ctrl/X] [Ctrl/B].
6. Move the cursor to the window containing the buffer list with [Ctrl/X] [O].
7. Notice that there is a . (displayed file) and * (modified) before `junk5`. Also notice how the two `emacs` buffers are named. Both have asterisks as the first and last character of the buffer name. If you access these buffers, you have to provide the asterisks as part of the file name.
8. Move the cursor to the line containing `junk5` with normal cursor motion commands. Mark `junk5` to be saved by typing "s". `emacs` places an **S** next to the files marked for saving.
9. Type "d" for `junk4` and `junk3` to delete those buffers. `emacs` places a **D** next to the buffers marked for deletion.
10. Move the cursor to `junk1` and type "m" to "mark" the buffer to be displayed. Enter an "m" for `junkfile`. Notice that these files are marked with an **>**.
11. Type "x" to execute the save and deletions previously marked. The buffers you marked for deletion are gone. The files still exist, but the buffers are gone. Also, the modified file, `junk5`, which was marked with an * , has been saved, so is no longer marked as modified.
12. Type "q" to exit the buffer list and display the two marked buffers. We might have expected `emacs` to provide us with just the two buffers we marked. Instead, `emacs` provides three buffers, the two we marked, plus the buffer after the last one we marked.

If you want two buffers on the screen, use one of the following:

- Use [M] and [Q].
  - Mark one of the buffers you want to display with [M].
  - Place the cursor on the line containing the buffer name of the other buffer you want to display.
  - Type [Q] to cause the buffers to be displayed.
- Use [M] and [2].
  - Mark one of the buffers you want to display with [M].
  - Place the cursor on the line containing the buffer name of the other buffer you want to display.
  - Type [2] to cause the two buffers to be displayed.

13. Display just the two buffers, *junk5* and *junkfile*:
    - Enter [Ctrl/X] [1] to remove all buffers but one. It does not matter which window you are in.
    - Enter [Ctrl/X] [B] to get the buffer list back.
    - Enter [Ctrl/X] [O] to move the cursor to the window containing the buffer list.
    - Move the cursor to the line containing the file name `junk5` and mark the file for display with an **m**.
    - Move the cursor to the line containing the file name `junkfile` and enter a **2** to get the two files displayed.
14. Use the tables in the chapter to try out some of the other buffer list commands.

No solution required.

Summary

Introduction to UNIX Editors

The standard editors available on most UNIX systems are:

Editor	Comment
ed	A line editor that operates on one line at a time. ed is the only editor available in single-user mode.
ex	A line-oriented editor similar to ed, but more powerful. It provides a subset of the vi screen editor commands. The vi editor can be invoked from ex.
edit	A line editor designed for beginning users. A simplified version of ex.
sed	A stream line editor that modifies lines based on script commands from the command line or from a script file.
vi	A full screen display editor based on the ex line editor. Contains different operational modes, and options that can be customized.
view	A read-only version of vi.
emacs	A powerful, extensible, screen editor which allows editing in multiple windows and has an extensive Help facility.

The emacs Editor

The emacs editor is the most powerful standard UNIX editor.

- emacs is more than an editor; it is a work environment because you can do so much while you are in emacs, such as:
  - Get a directory
  - Edit, rename, delete, and copy files
  - Compile programs
  - Send and read mail
  - Check for proper spelling
  - Execute a single shell command
  - Run a shell in an emacs window

emacs has major and minor modes. The major modes allow emacs to operate differently for different file types. For example: text mode, C mode, FORTRAN mode, and Lisp mode.

- The minor modes modify the major modes; they allow word abbreviation, fill, autosave, and overwrite.
- emacs does not edit the file; it edits a copy of the file in a buffer.
- emacs can have many active buffers at one time.

- You can have buffers that contain copies of the files you are working on, and temporary buffers.
- emacs creates some buffers for its own use, and you can also edit them.

Using emacs

You can run emacs in the current window on a terminal or workstation. You can also use emacs in a separate X window on a workstation.

You start emacs with emacs [options] *filename*.

The emacs screen is divided into three areas:

- The editing area (which may contain one or more windows)
- A mode line for each window
- The minibuffer area

emacs commands can be executed by:

- Typing the key sequence
- Typing Esc/X followed by the command name

You can exit or suspend your editing session:

- Temporarily suspend emacs: Ctrl/Z or Ctrl/X Ctrl/Z
- Exit emacs: Ctrl/X Ctrl/C

emacs Help

emacs has a very extensive and useful Help facility. You can initiate help with Ctrl/H followed by the character that represents the type of help you want. Another Ctrl/H provides a list of applicable characters. Yet another Ctrl/H provides a description of each of the available help categories.

Basic Editing

emacs commands can be repeated by using Esc *number* or Ctrl/U *number* ahead of the command key sequence.

You can stop emacs commands and undo changes:

- If you start a command by mistake, it can be stopped with Ctrl/G.
- If you made a mistake and changed some text you should not have changed, you can undo the mistake with Ctrl/X u.

emacs creates files in your directory:

- When emacs does an autosave, it creates a file with the same name as the original file, but preceded with, and appended with the pound sign, as **#filename#**.
- The first time you save a file, emacs creates a backup file. It has the same name as the file you are editing, but the file name has a tilde appended to the file name, as **filename~**.

Search and Replace Operations

emacs provides commands to search for a text string and optionally replace it. Start a search with Ctrl/S.

emacs provides several options for replacing.

Using Buffers and Windows

emacs allows you to edit multiple buffers at one time. A window on the screen displays a buffer.

The Ctrl/X B command switches buffers.

Correcting Mistakes

emacs provides commands to transpose characters, words, sentences, or paragraphs.

emacs provides commands to change the capitalization of text.

9
The vi Editor

Using the vi Editor

Introduction

This chapter introduces the vi editor. One major advantage of vi over emacs is that most UNIX systems offer the vi editor as one of the screen editors. It is also similar to the ed editor used for system management purposes.

If you stay with vi, and have to move from one UNIX based operating system to another, you might not have to learn any other editor.

In this course, the vi editor is covered sufficiently to allow you to create and edit files, recover a file you may have been editing when the system had an interruption, and other basic editing commands. Learning activities allow you time to become more familiar with the editing commands.

Objectives

To efficiently edit using the vi editor, you should be able to:

- Use the vi editor to create, modify, and recover files

Resources

- *OSF/1 User's Guide*
- *ULTRIX vi Beginner's Reference*
- DEC OSF/1 man pages

Editing with vi

Overview

This chapter discusses the vi modes of operation as well as how to start and terminate an editing session. Also discussed are the vi commands for:

- Cursor movement
- Text insertion
- Text deletion and substitution

This chapter also covers simple vi customization and how to recover from an aborted editing session.

Setting Up Your Terminal

If you are using a DEC OSF/1 system from a remote host (for example, an ULTRIX system), set up your workstation window or terminal for VT100 mode. Otherwise, when Esc is used to return to command mode, the case of the last letter typed may be changed (uppercase to lowercase or vice versa).

To rectify this situation:

System Type	Task
Terminal on a remote ULTRIX system	Use the General Setup directory and select VT100 mode.
Workstation	Choose General... from the Options Menu in the terminal window, and turn on VT100 mode.

Many keyboards do not contain a labeled Esc (Escape) key. There are alternate keystrokes that may work on your system.

- Press the F11 key to see if it functions as Esc .
- With VT100 mode enabled, press Ctrl/ [to perform the Escape function.

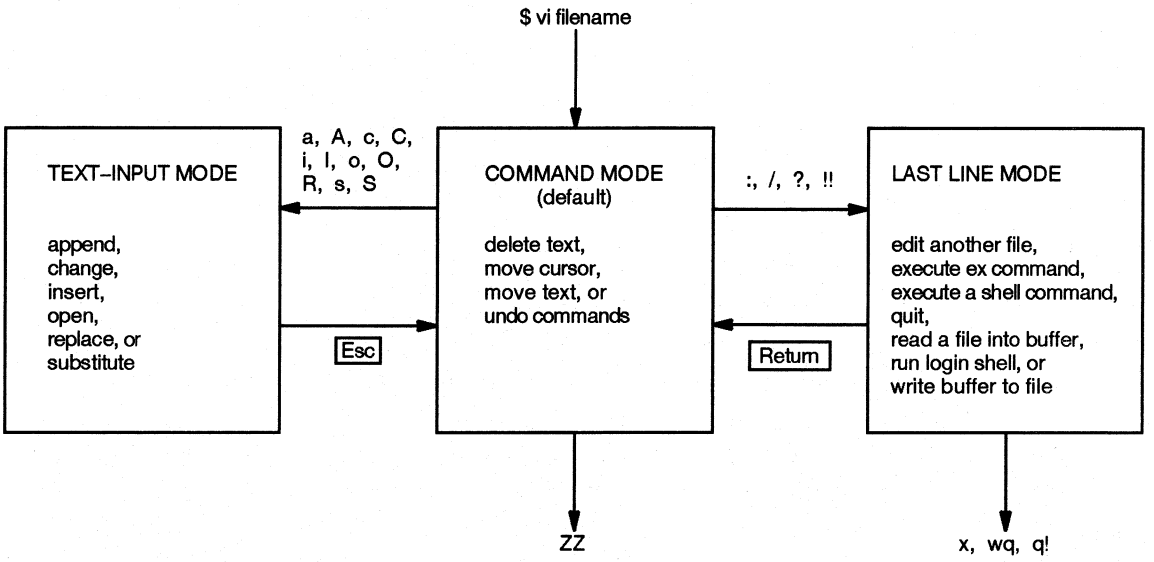
Editing Modes

The vi editor is called a **modal** editor, because the effect of a series of keystrokes depends on which of three modes the editor is in:

- Command (editing) mode
- Text-input mode
- Last-line mode

The relationship between the three operational modes is shown in greater detail in the following figure.

Figure 9–1 vi Modes



Moving Between Editing Modes

You must understand how and when to navigate between editing modes.

Table 9–1 Editing Modes

Mode	Description
Command	The default mode. Entered upon editing a file with `vi`. Used mainly to move the cursor, and delete or change text. Any subcommand can be entered in this mode except text-input mode commands. Entered from text-input mode by pressing Esc. Entered from last-line mode by pressing Return. Typing Esc in command mode cancels a partial subcommand. Pressing the interrupt key (usually Ctrl/C) will also cancel a partial subcommand.
Text Input	Entered from command mode by typing the letters `a`, `A`, `cx`, `C`, `i`, `I`, `o`, `O`, `R`, `s` or `S`. The `x` in `cx` represents the scope of the command. After entering one of these commands, you can enter text into the buffer at the current cursor position. To return to command mode, press Esc or Ctrl/[.
Last Line	Entered from command mode by typing a colon (:), slash (/), question mark (?), or exclamation marks (!!). A prompt appears on a line at the bottom of the screen. Type a command. Press Return to perform the command, or press the interrupt key (usually Ctrl/C) to cancel the subcommand.

If you are not sure what mode you are in, press Esc twice, and when you hear the beep, you have returned to command mode.

Invoking vi

You can invoke the `vi` editor several ways:

- Specify the name of an existing file you want to edit.
- Specify the name of a new file you want to create.
- Invoke `vi` without specifying a file name.

  If you do not specify a file name when you invoke `vi`, you can specify the name of the file when you save it later.
- Use the `-R` option to open a file in read-only mode.
  - You can use normal editing commands in the buffer, but you cannot save the file when exiting.
  - You can also use the `view` command instead of `vi -R` to view a file in read-only mode.

The format of the `vi` command is:

```
$ vi [options] [filename].
```

Editing an Existing File

The example shows how to edit the `vegetables` file in the current directory.

Example 9–1 Invoking vi for an Existing File

```
$ vi vegetables
carrots ❶
peas
cauliflower
broccoli
beans
~ ❷
~
~
    .
    .
    .
~
~
"vegetables" 5 lines, 40 characters ❸
```

❶ If the file exists, the first lines of the file are displayed on the screen.

❷ If the number of lines is too few to fill the screen, all lines that do not contain characters have a tilde (~) in the first column.

If you are editing a new file, the screen contains the tilde (~) character in the first column of most lines on the screen except the last line.

❸ The last line indicates the name of the file and number of lines and characters in the file.

Invoking vi to Create a New File

When you edit a new file with `vi`, the screen contains the tilde (~) character in the first column of most lines on the screen except the last line, as shown in the example.

Example 9–2 Invoking vi for a New File

```
$ vi junk
~
~
~
    .
    .
    .
~
~
"junk" [New file]
```

Interrupting, Canceling, and Exiting vi

The following table lists the commands used to quit vi, execute a command, or run the login shell from within vi.

Table 9–2 Interrupting, Cancelling, and Exiting vi

To	Use
Exit vi, saving changes, if any.	ZZ
Quit vi. If there were changes, vi displays a warning message, and does not quit.	:q
Quit vi, without saving changes, with no warning message.	:q!
Run the specified shell command. The command will not change the edit buffer.	:w ! shell_command
Run the login shell. Exiting the shell will return you to vi.	:sh
Suspend vi.	Ctrl/Z

Several commands start with a colon, which invokes last-line mode followed by a last-line mode command.

The Ctrl/ Z command is used for end-of-text on many editors including several standard editors on OpenVMS systems.

If you issue a Ctrl/ Z command in the vi editor, the shell prompt reappears. Beginners often assume that they have exited the editor and saved the new version of the file. In fact, the editor has just been suspended and any subsequent commands will utilize the old version of the file.

There are other commands in this category, but they are beyond the scope of this course. See the man pages for vi.

Moving the Cursor

Overview

When you start to edit a file with `vi`, you are in command mode. You must be in command mode to execute the cursor positioning commands.

The arrow keys are the easiest way to move the cursor when you first start using `vi`.

Fast typists may find the h, j, k, and l keys easier to use than the arrow keys.

Moving the Cursor by Characters and Lines

The following table shows several sets of keystrokes used to move the cursor near the line you are currently editing.

Table 9–3 Cursor Movement by Characters and Lines

Move the cursor	Use
One character to the left	h, Ctrl/H, or Left Arrow
Down one line	j, Ctrl/N, or Down Arrow
Up one line	k, Ctrl/P, or Up Arrow
One character to the right	l, Spacebar, or Right Arrow

Note

The `vi` editor is case sensitive and the use of uppercase and lowercase for the same letter often provides different results.

Moving the Cursor Within a Line

The following table lists some keystrokes that allow you to quickly navigate within a line.

Table 9–4 Cursor Movement Within a Line

Move the cursor	Use
To the beginning of the line	0
To the first nonspace character	^
To the end of the line	$
To the next character x	fx
To the previous character x	Fx
One column before the next character x	tx
One column after the previous character x	Tx
To column [count]	[count] |

To	Use
Repeat the last f, F, t, or T command	;
Repeat the last f, F, t, or T command in the opposite direction	,

Repeating Cursor Movement Commands Multiple Times

You can prefix vi commands with the number of times you want the command to execute.

- Press k to move the cursor up one line.
- Prefix the k command with 6, (6 k), to move the cursor up 6 lines.

  This only works if there are 6 existing lines above the present cursor position.

Cursor Movement by vi Word

Consider the definition of a vi "big word" and "small word" before examining Table 9–5. You can move the cursor forward and backward by word.

vi Word Type	Description
Big word	A sequence of letters, digits, and/or punctuation marks delimited by new line, space, tab, or the beginning of a line. Punctuation marks are big words when they are delimited by a space or tab on each side.
Small word	A sequence of letters and/or digits delimited by new line, space, tab, the beginning of a line, or a punctuation mark. A series of punctuation marks in sequence is a small word.

Table 9–5 Cursor Movement by Words

Move the cursor to	Use
Next small word	w
Previous small word	b
End of the next small word	e
Next big word	W
Previous big word	B
End of the next big word	E

Understanding Cursor Movement by Word

Small Word

Consider the following sequence of characters:

```
cursor motion.stop letters b,B,e,E,w,W! more text and yet some more text
```

Start with the cursor on the first letter in the line. A series of `w` (small word) commands moves the cursor as follows:

To the `m` in motion
To the period
To the `s` in stop
To the `l` in letters
To the `b`
To the comma
To the `B`
To the next comma

The output is shown below.

```
cursor motion.stop letters b,B,e,E,w,W! more text and yet some more text
       w     ww    w       wwww
```

Big Word

Consider the following sequence of characters again:

```
cursor motion.stop letters b,B,e,E,w,W! more text and yet some more text
```

Start with the cursor at the beginning of the line. If you type a series of `W` (big word) commands, the cursor moves as follows:

To the `m` in motion
To the `l` in letters
To the `b`
To the `m` in more
To the `t` in text

The output is shown below.

```
cursor motion.stop letters b,B,e,E,w,W! more text and yet some more text
       W          W       W           W    W
```

Cursor Movement by Line, Sentences, and Paragraphs

This table shows how to move the cursor by lines and sentences.

Table 9–6 Cursor Movement: Moving to a Line Position

Move cursor to:	Use
Top line on the screen	H
Middle line on the screen	M
Bottom line on the screen	L
First nonspace character on the next line	+ or Return
First nonspace character on the previous line	-

In addition to moving the cursor by words and lines, you can move the cursor by sentences, paragraphs, and pages of text.

Table 9–7 Cursor Movement: Moving by Sentences, Paragraphs, or Sections

Move the cursor to the beginning of:	Use
Previous sentence	(
Next sentence	)
Previous paragraph	{
Next paragraph	}
Previous section	[[
Next section	]]

Cursor Movement by Page

The amount that the screen scrolls depends upon:

- The `stty` **rows** value
- The `vi` **windows** value

  Ranges:

  - VT100: 12 through 24
  - Workstation: 24 through 48

Provide a `[count]` value for Ctrl/U or Ctrl/D to control how many lines scroll at a time.

If you enter the command 8 Ctrl/D, the cursor moves down or into the file by 8 lines.

Additional Ctrl/D commands will scroll the file by 8 lines.

The next time you use a count with Ctrl/D, the screen scrolls the number of lines you indicate in the count.

Table 9–8 Cursor Movement: Paging and Scrolling

To	Use
Scroll up	Ctrl/U
Scroll down	Ctrl/D
Page forward one screen	Ctrl/F
Page backward one screen	Ctrl/B
Scroll window up one line	Ctrl/Y
Scroll window down one line	Ctrl/E
Page up	z+
Page down	z^
Place the cursor at line number `n` or at the last line	n G

Setting Up Learning Activity

To prepare for this learning activity, convert the `vi man` pages to a plain text file by entering the following command exactly as given:

```
$ man vi | sed 's/.Ctrl/V Ctrl/H//g' > man-vi
```

Learning Activity: Moving the Cursor

This learning activity provides practice using the `vi` cursor movement commands. Choose a set of commands and use it consistently.

Choose one of the following sets of keys:

- The arrow keys
- h, j, k, and l
- Ctrl/H, Ctrl/N, Ctrl/P, and Spacebar

1. Edit the file named `man-vi`.

   ```
   $ vi man-vi
   ```

2. Use cursor movement commands to move the cursor:
   a. Down 10 lines with Down Arrow, j, or Ctrl/N to the line that starts `vi, vedit, view -`.
   b. Right to the word `files` with Right Arrow, l, or Spacebar.
   c. Back to the left, to the first letter in `view` with Left Arrow, h, or Ctrl/H.
   d. Down 10 more lines, then back up to the present position with Up Arrow, k, or Ctrl/P.
   e. To the beginning of the line with 0.
   f. To the end of the line with $.
   g. To the first nonspace character with ^. Repeat a few 0 and ^s. Notice the difference in the two commands.
   h. To the next character `n` with fn.

      Try it again. Note that the command is valid only on a single line of text. The cursor does not move to another line.

   i. To the previous `w` with Fw.
   j. To the beginning of the line with 0.
   k. To the first `e` with fe.

      Each time you enter ;, an f, F, t, or T command is repeated. Repeat the command twice.

      Each time you enter ,, an f, F, t, or T command is repeated, but in the opposite direction. Repeat the command in the opposite direction a few times.

3. Use some commands from Table 9–5 to move the cursor to the line two lines after `SYNOPSIS`.

If you do not see the line on the screen, enter 15[G]. The line in question starts `vi/vedit/view [-lRv]`.

4. Use [0] to position the cursor at the beginning of the line. Then move the cursor:
   a. To the next small word with [w]. Repeat the command several times.
   b. Back a small word with [b]. Repeat the command until the cursor is at the first character in the line.
   c. To the next big word with [W]. Repeat the command a few times.
   d. To the previous big word with [B]. Repeat the command.
5. Press [Ctrl/F] to get to the next screen of text.
   Move the cursor:
   a. To the top of the screen with [H]
   b. To the bottom of the screen with [L]
   c. To the middle of the screen with [M]
6. Practice the following:
   a. Scroll the screen up with [Ctrl/U]
   b. Scroll the screen down with [Ctrl/D]
   c. Page forward one screen with [Ctrl/F]
   d. Page backward one screen with [Ctrl/B]
   e. Move to the bottom of the file with [G]
   f. Move to the top of the file with 1[G]
   g. Move to line 57 with 57[G]

No solution required.

Finding, Inserting, and Changing Text

Overview

This section discusses how to search for, insert and change text in a file. The following tables provide the commands necessary to complete the operations. Learning activities allow you to practice your skills in each area.

Searching for Patterns

The [f] and [F] keys are useful only when searching for a single character in the line where the cursor is currently located.

The commands shown in the following table allow you to search for patterns anywhere in a file.

Table 9–9 Searching for Patterns

To	Use
Place the cursor at the next line containing `pattern`	[/] `pattern`
Place the cursor at the previous line containing `pattern`	[?] `pattern`
Repeat the last search for `pattern` in the same direction	[n]
Repeat the last search for `pattern` but in the opposite direction	[N]
Place the cursor `n` lines after the line matching `pattern`	[/] `pattern` [/+] `n`
Place the cursor n lines before the line matching `pattern`	[?] `pattern` [?-] `n`
Find the parenthesis or brace that matches the one at the current cursor position	[%]

Learning Activity: Search Commands

This learning activity provides practice using some of the `vi` search commands.

1. Make sure the cursor is at line 57.

   If you exited the `man-vi` file from the previous learning activity, edit the file again and place the cursor at line 57 using 57[G].

   a. Place the cursor at the next line containing `modes` with [/] modes [Return].

   b. Find the next occurrence of `modes` in the same direction with [n]. Repeat the command.

   c. Find the next occurrence of `modes` in the opposite direction with [N]. Repeat the command.

   d. Move the cursor to the bottom of the file with [G].

e. Find the previous line containing the pattern `subcommand` with [?] `subcommand` [Return] /.

f. Repeat the command in the same direction with [n]. Notice that as the search was towards the beginning of the buffer, [n] continues in that direction.

g. Repeat the search, but in the opposite direction with [N].

2. Exit the file with [:q!].

No solution required.

Inserting Text

Starting from command mode, practice placing the `vi` editor in text-input mode and adding text to the file.

- Press [Esc] to enter text-input mode.
- Press one of the text insertion keys listed in Table 9–10 to initiate text editing.

  All keys you type now appear as normal text until you press the [Esc] key to return to command mode.
- Press [Esc] to exit text-input mode and reenter command mode.

Table 9–10 Inserting Text

To	Use
Append text to the right of the cursor	[a]
Append text at the end of the line	[A]
Insert text before the cursor	[i]
Insert text before the first nonspace character in the line	[I]
Open an empty line for text insertion below the current line	[o]
Open an empty line for text insertion above the current line	[O]

Changing Text

Use the commands in the following table to change text.

Table 9–11 Changing Text

To	Use
Change this word	cw
Change previous word	cb
Change to the end of the line	c$
Change to the beginning of the line	c0
Change the entire line	cc or C
Delete text from the present cursor position to the end of the word (and following space)	dw
Deletes text to the end of the current word (but not a following space)	de
Deletes to the end of the line	d$ or D
Deletes to the beginning of the line	d0
Deletes a complete line	dd
Replace the present character with the character represented by `char`	r`char`
Overstrike existing characters with new characters	R
Delete the present character and enter new text	s
Delete the present line and enter new text	S
Delete the present character	x
Delete the previous character	dh or X
Join the next line to the end of the present line	J
Undo the previous change	u
Change the case of the letter under the cursor	~

Learning Activity: Inserting and Changing Text

This learning activity provides practice using the commands to insert and change text.

1. Edit the `man-vi` file.

   ```
   $ vi man-vi
   ```

2. Move the cursor to the bottom of the file with [G].

   If you have trouble with the file, exit the editor with [:q!] and reedit the file.

   a. Use [A] to append a few carriage returns and some text after the last line of the file.

      Press [Esc] to return to command mode.

   b. Move the cursor to any line in the middle of the file.

      Position the cursor on a space in the middle of a line.

      Use [a] to append text to the right of the cursor.

      Add any word, then press [Esc] to return to command mode.

   c. Move the cursor to a different line.

      Use [i] to insert a few words.

      Press [Esc] to exit text-input mode and return to command mode.

   d. Use [I] to enter some text at the beginning of the line.

      Add a few words and exit text-input mode with [Esc].

   e. Exit the `vi` editor without saving changes with [:q].

3. Edit the same file again.

   ```
   $ vi man-vi
   ```

4. Practice some of the commands to change and delete text.

   a. The text that can be changed is bounded by the present cursor position and the motion part of the command.

   b. After you enter the command, the amount of text to be changed is indicated by a dollar sign ($) at the end of the change area.

   c. You can add more text than the motion indicates. Any existing text beyond the dollar sign is pushed to the right.

5. Use the [c]`motion` command to change text.

   a. Place the cursor on the first character of the line.

      Enter 2[cw] to change the first two words to `If someone`.

      Note the position of the dollar sign.

      Press [Esc] when the change is complete.

   b. Change the two words back to the original text.

      Place the cursor on the space after `someone`.

      Enter 2[cb] to change the two previous words back to the original text.

If there is one character left over when you finish typing, it disappears when you press Esc.

Note

If you had not placed the cursor at the space, you would probably have had characters left over. You can illustrate this by changing the two words to `If someone` again. Then, place the cursor on the last letter of `someone` and enter 2cb and change the words back. This time there is an extra character left over.

c. Move the cursor to any character. Change the case of `char` with r`char`.

   The r`char` changes the character to any other character.

d. Place the cursor at the beginning of a line.

   Enter R to place `vi` in overstrike mode.

   Overstrike all characters in the present line.

   Press Esc to return to command mode.

e. Move the cursor to delete text as follows.

   You can use a numerical prefix to delete multiples according to the `motion`. For example, 6x deletes the present character plus the next five characters.

   Use x to delete the present character. Repeat a few times.
   Use X to delete the previous character. Repeat a few times.
   Use dw to delete the next word.
   Use db to delete the previous word.
   Use dd to delete the entire line the cursor is on.

f. Move the cursor to the middle of a line and delete the remainder of the line with d$.

g. Move the cursor to the middle of the next line and delete all text from the cursor to the beginning of the line with d0 (zero).

h. Move the cursor to the end of the previous line and join the two half-lines with J.

i. Exit the `vi` editor with :q!.

No solution required.

Copying, Moving, and Saving Text

Overview

This section discusses how to use buffers with the `yank` and `put` commands and text deletions to copy or move text. It also covers commands necessary to save text by writing the buffer to a file.

Copying and Moving Text

A useful feature of any editor is the ability to delete or copy and move chunks of text to a different location.

The `vi` editor uses `yank` and `put`, along with the delete commands, to allow you to move text to other parts of the file.

- The `d`motion commands remove the text from the editing buffer.
- The `y`motion commands place a copy of the text in the appropriate buffer.

Using vi Buffers

The `vi` editor uses three types of buffers to hold deleted or yanked text. The contents of these buffers can be restored to the edit buffer. The three types of buffers are:

- undo buffer
- Numbered (delete) buffer
- Named buffer

Table 9–12 Yank and Delete Buffers

Buffer	Description	Comment
undo Buffer	Used for normal deletions and yanks. Contents are available until the next deletion or yank, at which time it is overwritten.	Any text deleted or yanked is placed in the undo buffer, and can be immediately placed back in the editing buffer using a `put` command. • The contents of the undo buffer is changed with each deletion or yank. • If you used `dw` three times in succession to delete three words, only the last deleted word would be available in the undo buffer. • Similarly, if you yanked three words with three successive `yw` commands, only the last yanked word would be available in the undo buffer.

(continued on next page)

Table 9–12 (Cont.) Yank and Delete Buffers

Buffer	Description	Comment
Numbered (delete) Buffer	Stores deletions in sequential locations when a full line or more is deleted (dd, d), d }, d]]).	There are nine locations in this buffer. When a number of successive deletions are made, the numbered (delete) buffer acts as a push-down stack. • When you retrieve the contents with p or P, you retrieve the most recent deletion. • To retrieve more deletions, provide the number as "np or "nP. • To retrieve the next-to-last item, location 2, use "2p or "2P. • If you made nine deletions, place the first deleted item back in the edit buffer with "9p or "9P. These commands are shown in Table 9–13.
Named Buffer	Provides 26 locations, named a through z, to hold deletions or yanks.	Each location must be named individually for the deletion, yank, or put. For example: • "bdd deletes a line and places the contents in location b of the named buffer. • "fd) deletes text up to the next sentence, placing the text in location f of the named buffer. • Retrieve these two deletions by using "bp and "fp, or "bP and "fP.

Note

The locations of the named buffer are alphabetical, lowercase. If you use an uppercase letter, the deletion or yank is appended to the contents of the named buffer of the same lowercase letter.

For example, delete a line of text into named buffer g. Then, delete another line of text into named buffer G.

If you now execute a put for named buffer g or G, you will get both lines.

Using Commands with the Buffers

Use these commands with undo, numbered, or named buffers.

Table 9–13 Copying and Moving Text

To	Use
Yank text indicated by `motion` into the undo buffer. [yw] yanks a word, [y)] yanks text to the beginning of the next sentence	[y] `motion`
Yank the entire line into the undo buffer	[Y]
Yank text indicated by `motion` into the named buffer designated by alphabetical character `x`	["xy] `motion`
Delete text indicated by `motion` into named buffer `x`. [dd] deletes the entire line; [d)] deletes text up to the next sentence	["xd] `motion`
Put back text from the undo buffer after the cursor	[p]
Put back text from the undo buffer before the cursor	[P]
Put back text from numbered buffer `n` after the cursor [1]	["np]
Put back text from numbered buffer `n` before the cursor [1]	["nP]
Put back text from named buffer `x` after the cursor	["xp]
Put back text from named buffer `x` before the cursor	["xP]
Repeat the last ["np] command and increment the count	[.]

[1]Deletions of at least one line or more are placed in successive locations of the numbered buffer.

The following examples show how to use the commands in Table 9–13 with the undo, numbered, or named buffers.

- [y)] yanks text from the cursor to the beginning of the next sentence into the undo buffer. After cursor motion, copy text into the edit buffer with [p] or [P].
- [Y] yanks a line into the undo buffer.
- Use a series of sequential [dd] commands to delete a few lines.
  - Each line, up to 9, is placed in sequential locations in the numbered buffer.
  - A [p] or [P] retrieves the last deleted line from location 1.
  - ["2p] or ["2P] retrieves the contents of location 2.
  - If the remaining locations are required, [.] (period) repeats the last ["np] command and increments the count.

- [fy}] copies text from the cursor (until the beginning of the next paragraph) into named buffer `f`. It can be retrieved using ["fp] or ["fP].

Restoring and Repeating Changes

To restore the buffer to the condition it was in before you executed a command (all edits on a single line, or repeat the last command), use the following commands in command mode.

Table 9–14 Restoring and Repeating Changes

To	Use
Undo the last command if you make an error. The cursor need not be on the line where the original edit was made.	[u]
Undo all edits on a single line, as long as the cursor remains on that line.	[U]
Repeat the last command. If the last command was ["np], the [.] command increments `n`.	[.]

The period [.] increments only the location for the numbered buffer, not the named buffer, and only then if you have previously repeated a series of deletions.

If you delete text into numbered buffers with (["nd]), you cannot retrieve it. You must use successive deletions.

Saving Changes

To write the buffer to a file, use the commands shown in the table in command mode. The colon brings you to last-line mode where the commands are executed.

Table 9–15 Writing the Buffer

To	Use
Write the editing buffer to the original file	[:w]
Write the editing buffer to `file`	[:w] `file`
Overwrite `file` with the contents of the editing buffer	[:w!] `file`

Editing Another File

When editing several files, you can save time by not invoking and exiting `vi` for each file.

If you make changes to the current file, and decide not to keep these changes, you can start again and reedit the file from the beginning of the editing session.

Use the following commands in command mode to edit a different file.

Table 9–16 Editing Another File

To	Use
Edit `file`	:e `file`
Reedit the current file and discard all changes	:e!
Read `file` into the editing buffer below the cursor	:r `file`
Run shell command `command`, placing the command output in the file below the cursor position	:r!`command`

Screen Control Commands

While editing a file, you may receive mail or system messages that interfere with the display. Use the following commands executed in command mode to redraw the screen.

Table 9–17 Screen Control Commands

To	Use
Clear and redraw the screen	Ctrl/L
Redraw the screen with the current line at the top of the screen	z Return
Redraw the screen with the current line at the bottom of the screen	z-
Redraw the screen with the current line at the center of the screen	z.
Redraw the screen with the line containing `pattern` at the bottom of the screen	/pattern/z-

Learning Activity: Copying and Moving Text, and Writing Files

This learning activity provides practice with some of the `vi` commands to copy and move text, and write files.

1. Edit the `man-vi` file again.
2. The `vi` editor uses y `motion` to yank text into the undo buffer and p or P to put text back.

   Note that y `motion` is used to copy text, not to delete it.

   a. Yank any four words to the undo buffer with 4y w.

   b. Move the cursor to the bottom of the buffer with G.

      Open a few lines with o followed by a few carriage returns.

      Press Esc to return to command mode.

      Put (copy) the text into the buffer using p.

   c. Move the cursor back to a full line of text.

      Place the cursor at the beginning of the line and yank the line to the undo buffer using y $.

d. Move the cursor to an open line and put the text into the buffer using [p].

3. Delete three lines to the named buffers and put them back into the buffer in reverse order.

a. Place the cursor at the beginning of line 1 and delete the line to named buffer `a` with ["add].

b. Delete the next line to named buffer `b` with ["bdd].

c. Delete the next line to named buffer `c` with ["cdd].

d. Put line 1 back with ["aP].

e. Put the next line back with ["bP].

f. Put the last line back with ["cP].

4. Use commands to write and save the buffer to a file other than the default file and exit `vi` in different ways.

a. Write the present buffer to a file named `xx` with [:w] `xx`.

b. Write the file to the original file with [:w].

c. Insert some text in the file and try to quit `vi` with [:q].

Note that `vi` indicates that the buffer has not been written to since the last change.

d. Exit `vi` with [:q!].

e. Edit the file `man-vi` again.

f. Execute the `ls` command with [:w !]`ls`. Make sure that there is a space between the `w` and `!`.

g. Run your login shell with [:sh].

Once it starts, execute a few commands (`date`, `who`).

Exit the shell back to the editor with [Ctrl/D].

h. Edit another file, `vegetables` with the command [:e] `vegetables`.

i. Move the cursor to the bottom of the file and open a line. Press [Esc] to return to command mode, and read the file `phones` into the present file with [:r] `phones`.

j. Exit `vi` with [:q!].

No solution required.

Recovering an Editing Session

If a system failure occurs while you are editing a file, any edits made after you last saved the file are lost. However, a copy of the edit buffer, containing your edit changes, should be saved.

The next time you log in, you may receive a mail message indicating that your buffer was saved.

To determine if there are any recover buffers available, execute the `vi` command with the `-r` option:

```
$ vi -r
```

You receive a list of any of your editing buffers that the system has saved.

To recover the buffer, use the `-r` option with the file name as follows:

```
$ vi -r filename
```

Make sure that you recover all saved buffers before you edit them. If you do not, you could lose any previous edits you made prior to the system fault.

If you want to test the recovery option, you can simulate a system crash in a DECterm window by selecting Exit from the File menu.

Customizing vi

The `vi` editor does not automatically wrap long lines. You must press Return, which some people find annoying.

When searching for text, the text might contain uppercase letters in some, but not all cases. Although `vi` does not ignore case during searches, you may want to.

The `vi` editor has many customization options available that can be seen from command mode with the `ex` command:

```
:set all
```

The output may be similar to that shown in the following example.

Example 9–3 vi Set Options

```
:set all
noautoindent            nonumber                        noslowopen
autoprint               nonovice                        nosourceany
noautowrite             nooptimize                      tabstop=8
nobeautify              paragraphs=IPLPPPQPP LIpplpipbp taglength=0
directory=/var/tmp      prompt                          tags=tags /usr/lib/tags
noedcompatible          noreadonly                      term=vt100
noerrorbells            redraw                          noterse
flash                   remap                           timeout
hardtabs=8              report=5                        ttytype=vt100
noignorecase            scroll=23                       warn
nolisp                  sections=NHSHH HUuhsh+c         window=47
nolist                  shell=/bin/ksh                  wrapscan
magic                   shiftwidth=8                    wrapmargin=0
mesg                    noshowmatch                     nowriteany
nomodeline              noshowmode
```

Using set Options

There are two types of options that can be changed with the `:set` command:

- Toggle options (either on or off)

  Toggle options may be on or off by default. To turn a toggle option on or off, the commands are:

  `:set option` (on) and `:setnooption` (off)

  To indicate that you want a search pattern to ignore the case of the target words, type:

  `:set ignorecase` or `:set ic`.

  To indicate that you want a search pattern to search the case of the target words, type:

  `:set noignorecase` or `:set noic`.

- Options that take a numeric or string value

  Some options have a value assigned to them. For example, to specify that `vi` should insert a carriage return 10 columns from the right margin, type:

  `:set wrapmargin=10` or `:set wm=10`

  It may become tedious having to customize `vi` each time a file is edited. The options can be set from the `.exrc` initialization file. You can have different `.exrc` files in different directories.

  When using an `.exrc` file to customize `vi`, do not provide the colon (:). The commands are executed by `ex` before it enters `vi`. An `.exrc` file shown in the following example sets the wrap margin and specifies that pattern searches will ignore case.

Example 9–4 .exrc File

```
set ignorecase
set wrapmargin=10
```

The remainder of the ex set options are beyond the scope of this course.

Summary

Editing with vi

- A file can be edited in read-only mode with `vi -R filename`. The `view` command is a read-only version of `vi`.
- The `vi` editor has three modes:
  - Command mode (the default)
  - Text-input mode

    Enter from command mode using any of the `a, A, cx, C, i, I, o, O, r, s` or `S` commands. Exit by typing Esc.
  - Last-line mode

    Enter from command mode by executing subcommands. Subcommands begin with `:, /, ?` or `!!`. After execution, reenter command mode.
- The `vi` commands can be preceded with a `count` to modify the number of times the command is executed.
- The `vi` editor uses three types of buffers to hold deleted or yanked text:
  - undo buffer
  - Numbered (delete) buffer
  - Named buffer
- The last command can be undone with the u command.
- The last command is repeated with the . command.
- Changes to the file are written to disk with :w.
- The `vi` editor is exited with ZZ, :q or :q!.

10

Redirecting, Filtering, and Piping

Redirecting, Filtering, and Piping

Introduction

The UNIX operating system has three very powerful tools: redirection of input and output, filters that read from standard input and print to standard output, and piping the output of one process to the input of another. This chapter explains how to use these tools and demonstrates some of their power.

Objectives

To use some of the more powerful tools of the UNIX operating system, you should be able to:

- Redirect shell command output to a file
- Redirect input to a shell command from a file
- Use filters such as `wc`
- Sort text with the `sort` filter
- Perform basic searches using the `grep` command
- Pipe basic shell commands
- Manipulate text using the `awk` utility:
  - Field-oriented processing
  - Arithmetic expression
  - Using predefined variables
  - Logical operations

Resources

- *OSF/1 User's Guide*
- *Guide to Programming Support Tools*

Redirecting Input and Output

Overview

Most programs including commands expect input and/or generate output. Consider the commands `ls` and `passwd`. The `ls` command generates output that tells you what files are in your current directory. The `passwd` command requires you to input your old and new passwords.

The UNIX operating system has three predefined file pointers:

- stdin: standard input, the keyboard by default
- stdout: standard output, the terminal screen by default
- stderr: standard error output, the terminal screen by default

Many UNIX programs use these pointers for input and output. If these pointers are redirected to point somewhere other than the default, the program will utilize the new location.

Redirecting standard input or output means that the input comes from somewhere other than the terminal keyboard or the output goes somewhere other than the terminal screen.

Redirection Symbols

Standard input and output can be **redirected** using the following symbols:

- > — Redirects standard output
- >> — Appends standard output
- 2> — Redirects standard error output
- < — Redirects standard input

Redirection is performed by the shell. The operators shown in this chapter are for the Korn shell, the C shell uses slightly different ones.

Using the > Symbol

The > symbol redirects standard output. You can perform several useful tasks using the redirection symbol >. It can be used with many different commands to put what is normally displayed on the terminal screen into a file that you can further manipulate. The > symbol:

- Replaces the contents of a file with the output of a command, even if the file already exists (except if `noclobber` is set)
- Creates a file if the file does not exist
- Generates an error if the file exists, but you do not have write access
- Opens the output file for writing before performing the command, so `cat r > r` will produce an empty file

Using the >> Symbol

The >> symbol also redirects standard output. This symbol appends the output of a command to a named file.

Functions of the > Symbol

The following table shows some examples of using the > redirection symbol and their functions.

Table 10–1 Functions of the > Symbol

Example	Function
`cat chap1 > newchap1`	Directs the output of `cat chap1` to a file named `newchap1`
`ls > listfile`	Directs the output of the `ls` command to a file named `listfile` which contains a listing of files in the current directory
`man cd > cdref`	Directs the output of the command `man cd` to a file named `cdref`
`cat workphones >> homephones`	Directs the output of `cat workphones` to be appended to an existing file named `homephones`
`cat report1 report2 >> report3`	Directs the output of `cat report1 report2` to be appended to a file named `report3`
`date > date_file 2> date_err`	Directs the output of the `date` command to the file `date_file` and directs any error messages to the file `date_err`

Using the 2> Symbol

Use the 2> symbol to redirect standard error output. By default, error output is displayed on the terminal screen. The following example shows how to use the 2> symbol to force an error message to a file named `err_file`.

```
$ chmod 000 phones
$ cat phones
cat: cannot open phones
$ cat phones 2> err_file
$ cat err_file
cat: cannot open phones
```

Using the < Symbol

The < symbol redirects standard input. It can only be used with commands that accept input from standard input. For example, you cannot use the < symbol with the `passwd` command because `passwd` reads its input from the keyboard device itself and not from standard input. It is legal but meaningless to specify < with commands that read no input.

The < symbol tells the system to get expected input from the file specified after the < symbol.

command-name < *input-filename*

- *command-name* is any command that accepts standard input.
- *input-filename* is the file that contains the input for the specified command.

The following example shows the use of the < symbol.

```
$ cat < chap3
```

The `cat` command displays the contents of a file on your terminal screen. If you do not supply a file name, the `cat` command reads its input from the keyboard. However, in the example shown above, the input comes from the file named `chap3`.

Note that the `cat chap3` command would have resulted in the same output because, with most commands, an input file name can be specified without the < symbol. However, the < symbol is required with some commands to perform special actions. For example:

```
mail student2 < chap10
```

This sends the file `chap10` to the user named `student2`. If no input file was specified, the system would wait for input from the keyboard.

Learning Activity: Redirecting Input and Output

1. Use the `ls` command to find out what files and directories are in your current directory. Instead of having the `ls` print its output to the screen, redirect it to a file called `ls.out`.
2. Use the > symbol with the `cat` command to copy the contents of `ls.out` to another file named `listing`. Then, view the file named `listing`.
3. Use the `ls -l` command to generate a long listing of the files and directories in your current directory. Redirect the output of this command so that it is appended to the bottom of the file named `listing`. Then view the file named `listing`. It should contain the listing that you did in step 1, plus the listing you did in step 3.
4. View the file named `listing` using the < symbol.

Learning Activity Solution: Redirecting Input and Output

1. Use the `ls` command to find out what files and directories are in your current directory. Instead of having the `ls` print its output to the screen, redirect it to a file called `ls.out`.

   ```
   $ ls > ls.out
   ```

2. Use the > symbol with the `cat` command to copy the contents of `ls.out` to another file named `listing`. Then, view the file named `listing`.

   ```
   $ cat ls.out > listing
   $ cat listing
       .
       .
       .
   ```

3. Use the `ls -l` command to generate a long listing of the files and directories in your current directory. Redirect the output of this command so that it is appended to the bottom of the file named `listing`. Then view the file named `listing`. It should contain the listing that you did in step 1, plus the listing you did in step 3.

```
$ ls -l >> listing
$ cat listing
    .
    .
    .
```

4. View the file named `listing` using the < symbol.

```
$ cat < listing
```

Filters

Overview

Filter is a term frequently used to refer to redirection of input and output. A filter is a command that reads standard input, processes the input, and prints the result to standard output.

Examples of filters that you have already used are:

- `cat`
- `head`
- `tail`

These commands read standard input and print the results to standard output. Like many filters, they will read their input from a file if a file name is specified as a parameter, or otherwise read from standard input.

For example, the `head file1` and `head < file1` commands both read a file (`file1`) and print the first lines to the terminal screen (standard output).

The `wc` filter is discussed in this section. The more complex `sort`, `grep`, and `awk` filters are discussed in later sections.

Filters are generally used as intermediate commands in pipelines. Pipelines are discussed in a later section.

Using the wc Filter

The `wc` (**w**ord **c**ount) filter is often used by programmers who want to see how much code they have written or by writers who want to see how much text they have written.

The `wc` filter performs a word count on its input file(s), or on standard input if no file name is given. It also provides a count of the number of lines and the number of characters in the file(s).

In the output, reading from left to right, the first number refers to the number of **lines** in the file, the second refers to the number of **words** in the file, and the third refers to the number of **characters** in the file, as shown in the example.

Example 10–1 Using the wc Command with Standard Input

```
$ wc
three
lines,
four words
Ctrl/D
       3       4       24
$
```

When an existing file name, for example, `newfile`, is used on the command line, `wc` prints the name of the file in the output, as shown in the following example.

Example 10–2 Using a File Name with wc

```
$ wc newfile
      10      27      139     newfile
$
```

If more than one file is named, the total number of lines, words, and characters for all of the input files is provided, as shown below:

```
$ wc newfile oldfile junkfile
      10      27      139     newfile
       9      23      121     oldfile
      22     143      861     junkfile
      41     193     1121     total
$
```

wc Command Options

The wc command has three options, each of which allows you to list one of the counts normally provided in the output. The options are shown in the table.

Table 10–2 wc Command Options

Option	Result
-l	Prints only the number of **lines** a file contains
-w	Prints only the number of **words** a file contains
-c	Prints only the number of **characters** a file contains

The options to the wc command can be combined. For example,

```
$ wc -lc file1
```

lists the number of **lines** and **characters** that file1 contains.

The /etc/passwd file contains one line for each account that is on the system.

A quick way to determine how many accounts there are on the system is to use the wc filter with the -l option on the /etc/passwd file.

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

Learning Activity: Using wc

1. Use the wc command to count the number of lines, words, and characters in the /etc/passwd file.
2. Use options to the wc command to display:
   a. Only the number of characters in the /etc/passwd file.
   b. Only the number of lines in the /etc/passwd file.
   c. Only the number of words **and** lines in the /etc/passwd file.

Learning Activity Solution: Using wc

1. Use the wc command to count the number of lines, words, and characters in the /etc/passwd file.

```
$ wc /etc/passwd
       39       115      2623 /etc/passwd
```

2. Use options to the wc command to display:

   a. Only the number of characters in the /etc/passwd file.

```
$ wc -c /etc/passwd
     2623 /etc/passwd
```

   b. Only the number of lines in the /etc/passwd file.

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

   c. Only the number of words **and** lines in the /etc/passwd file.

```
$ wc -lw /etc/passwd
       39       115 /etc/passwd
```

Sorting

Overview

Another useful filter is the sort command, which alphabetizes its input according to the first letters in each line and prints the sorted list to standard output. The example displays the contents of two files, hot and cool.

Example 10–3 Contents of the Files hot and cool

```
$ cat hot
orange
yellow
scarlet
magenta
fuchsia
$ cat cool
turquoise
aqua
violet
teal
chartreuse
```

The following example displays the result of using the sort filter on the two files, hot and cool.

Example 10–4 Result of the sort Command on the Files hot and cool

```
$ sort hot cool
aqua
chartreuse
fuchsia
magenta
orange
scarlet
teal
turquoise
violet
yellow
```

The sort command reorders text based on the **ASCII** values of the characters. Numbers and special symbols precede letters, and all uppercase letters precede all lowercase letters. ASCII stands for **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange. Every character is associated with a standard numerical code; ordering is based on these codes.

If only the hot file had contained uppercase letters, the output to the sort hot cool command would have been different, as shown below:

```
$ sort hot cool
FUCHSIA
MAGENTA
ORANGE
SCARLET
YELLOW
aqua
chartreuse
teal
turquoise
violet
```

You can tell sort to "fold" the uppercase and lowercase letters together by using the -f option. This option causes the command to ignore the cases of the letters, as shown below:

```
$ sort -f hot cool
aqua
chartreuse
FUCHSIA
MAGENTA
ORANGE
SCARLET
teal
turquoise
violet
YELLOW
```

sort Command Options

The table lists common options to the sort command and examples of each.

Table 10–3 Common Options to the sort Command

Option	Example	Result
-f	$ sort -f hot cool	Ignores the cases of letters. Sorts the files hot and cool, disregarding whether the letters in the file are uppercase or lowercase.
-r	$ sort -r cool	Sorts the file cool in **r**everse alphabetical order.
-u	$ sort -u hot cool	Sorts the files hot and cool eliminating any duplicates. Ensures that each item in the output is **u**nique.
-m	$ sort -m file1 file2	Reads the files file1 and file2 and **m**erges the results of each on the screen. This works only on files that are already sorted.
-o	$ sort -o warm hot	Sorts the file hot and places the **o**utput in the file name immediately following the -o option (warm). Enables sorting a file into itself, which cannot be done with >.

Sorting in Reverse Order

The following output shows sorting in reverse order.

```
$ sort -r cool
violet
turquoise
teal
chartreuse
aqua
```

Eliminating Duplicate Items

Sometimes when you sort several files, they will contain duplicate items. You may only want one copy of each of item. The `-u` option ensures that each item in the output is unique. Consider the `hot` and `cool` files: some people might consider chartreuse and fuchsia hot colors, while others might consider them cool colors. It is conceivable that they may be in both files. If you issue the `sort` command without the `-u` option, the output lists both fuchsia and chartreuse twice.

Merging

Use the `sort` command with the `-m` option to merge sorted files. To combine large files that are already in order (for example, two mailing lists of 500 names each), using the `-m` option is much faster than issuing an ordinary `sort` command. This option only merges files, it does not sort them. Consequently, it only works on files that are already sorted. Using it to merge unsorted files produces unpredictable output.

Sorting a File into a File

The `-o` option indicates to put the output of the `sort` command into a file whose name immediately follows the `-o` flag. The command `sort -o warm hot` has almost the same effect as `sort hot > warm`. The advantage of the `-o` option is that you can sort a file into itself. You cannot use the redirection symbol > to output to the same file name as you are sorting. When the command line is first scanned, because there is output redirection, the file is first opened for output and zeroed out, so there will be nothing left to sort.

Sorting by Different Fields

People often need to sort files that are more complex than the `hot` and `cool` files. Usually, files of data contain more than one type of information, like the `family` and `baseball` files shown in Example 10–5 and Example 10–6.

Each line in the examples, `family` file and `baseball` file, can be called a **record**. Each record can contain any number of **fields**. A field is a group of nonblank characters. The end of field is usually indicated by a blank character (Space, Tab, or Return). The fields in a record are numbered from left to right beginning at zero.

The term **sort key** refers to the portion of a record that is compared to other records when the file is sorted. The standard `sort` command (without any options) treats the entire record, or line, as a sort key.

It is often necessary to sort a file according to a specific field. Suppose, for example, you want an alphabetically sorted copy of the `family` file. The standard `sort` command sorts the file according to first names, using the entire record as a sort key. Usually when people's names are alphabetized, they are sorted according to last names. This is accomplished using options that relate to sorting by field.

Example 10–5 Contents of the family File

```
$ cat family
Mary Miller         555-1011
Tom Miller          555-3110
Rick Miller         555-0107
Emily Miller        555-7200
Sherry Miller       555-0912
Peter Parker        201-4019
Barbara Parker      101-2040
Scott Brown         555-3131
Christi Brown       702-0625
```

Example 10–6 Contents of the baseball File

```
$ cat baseball
Babe Ruth          .342     714
Reggie Jackson     .268     490
Hank Aaron         .305     755
Mickey Mantle      .298     536
Lou Gehrig         .340     493
Ty Cobb            .367     118
Rod Carew          .331      87
Willie Mays        .302     660
Joe DiMaggio       .325     361
Roberto Clemente   .317     240
Jackie Robinson    .311     137
Carl Yastremski    .285     452
Willie Stargell    .282     475
Ted Williams       .344     521
```

sort Options Relating to Field Position

The table displays the options to the sort command that relate to field position.

Table 10–4 sort Command Options Relating to Field Position

Option	Function
+pos	Starts sort key in specified field **pos**ition (Numbering of fields begins at zero)
-pos	Stops sort key in specified field **pos**ition
-n	Sorts fields according to **n**umeric values instead of ASCII values

Beginning Sort Key Position

This sort command option indicates the first field for the beginning of the sort. For example,

```
sort +1 family
```

is interpreted as "Start sorting at the beginning of the record, plus one field." This sorts the file family by last name. The result is shown in the first part of Example 10–7.

In Example 10–7, after sorting the field specified by the beginning sort key position, sorting continues to the end of the line, if necessary, to resolve any conflicts. Here, because spaces were used to separate the last names from the phone numbers, where the last names are the same, the spaces are compared, then the numbers. Because the entry for Tom Miller has more spaces than any other Miller, he is listed first. With Rick and Mary Miller, because their entries both contain eight spaces, the phone number is used to resolve the conflict. If an equal number of tabs were used instead of spaces, the order would be different because the phone numbers would be used to resolve conflicts.

Ending Sort Key Position

This option indicates the last field for the end of the sort.

While the `sort +1 family` command sorts the file alphabetically by last names, the first names are not in the correct order. Ideally, when there are identical names, the first names should be compared. Because the end of sort key (`-pos`) was never given, the sort key started at the last name (field 1) and continued to the end of the line. The output is not really what you want because the first names are not ordered.

If you want the `sort` command to compare only the last names and resolve conflicts (identical items) by looking at the first name also, you must specify the end of the sort key.

```
sort +1 -2 family
```

is interpreted as "Start sorting at the second field (field 1), and stop before the third field (field 2)." Any conflicts are resolved by comparing the lines again from the beginning. Since the first names are at the beginning of the line, the result is that the names are sorted by first and last names, as shown in the second part of the following example.

Example 10–7 Results of Sorting Files by Fields—The family File

```
$ sort +1 family
Scott Brown          555-3131
Christi Brown        702-0625
Tom Miller           555-3110
Rick Miller          555-0107
Mary Miller          555-1011
Emily Miller         555-7200
Sherry Miller        555-0912
Peter Parker         201-4019
Barbara Parker       101-2040
$
$ sort +1 -2 family
Christi Brown        702-0625
Scott Brown          555-3131
Emily Miller         555-7200
Mary Miller          555-1011
Rick Miller          555-0107
Sherry Miller        555-0912
Tom Miller           555-3110
Barbara Parker       101-2040
Peter Parker         201-4019
```

Sorting Numerically

Suppose you want to sort the `baseball` file (Example 10–6) in order of the number of career home runs each player has hit. You could issue the command:

```
sort +3 baseball
```

The output of that command is:

```
Ty Cobb             .367     118
Jackie Robinson     .311     137
Roberto Clemente    .317     240
Joe DiMaggio        .325     361
Carl Yastremski     .285     452
Willie Stargell     .282     475
Reggie Jackson      .268     490
Lou Gehrig          .340     493
Ted Williams        .344     521
Mickey Mantle       .298     536
Willie Mays         .302     660
Babe Ruth           .342     714
Hank Aaron          .305     755
Rod Carew           .331      87
```

The output is sorted according to ASCII codes, not according to numerical values. The number 87 begins in the same field as the 3-digit numbers, and the first digit is an eight, which is larger than the first digit of any of the other numbers. Therefore 87 is considered larger than 755.

Since these numbers have numerical significance, they should be sorted by value. You can tell the `sort` command that the field you specify is numeric by adding an `n` to the field name you are sorting. For example,

```
sort +3n baseball
```

results in the `baseball` file being sorted in order of the numerical values of the field specified, 3. The resulting output is shown in Example 10–8.

Example 10–8 Results of Sorting Files by Fields—The baseball File

```
$ sort +3n baseball
Rod Carew          .331     87
Ty Cobb            .367    118
Jackie Robinson    .311    137
Roberto Clemente   .317    240
Joe DiMaggio       .325    361
Carl Yastremski    .285    452
Willie Stargell    .282    475
Reggie Jackson     .268    490
Lou Gehrig         .340    493
Ted Williams       .344    521
Mickey Mantle      .298    536
Willie Mays        .302    660
Babe Ruth          .342    714
Hank Aaron         .305    755
```

Learning Activity: Using the sort Filter

1. Create a file called `familydates` that contains the first names, last names, and birthdays of a few members of your family. Press Tab to separate the name from the birthday. The entries should use the numerical form of the date in the format month/day/year, for example:

   ```
   John Doe Tab3/15/50
   ```

   Be sure to include your own name and birthday in the file.
2. Use the `sort` command to print out your birthday list in alphabetical order by first name.
3. Use the `sort` command to print out your birthday list in **reverse** alphabetical order by first name.
4. Create another file (in the same format as `familydates`) called `workdates` that contains the first names, last names, and birthdays of a few of your coworkers. Again, be sure to include your name and birthday in this file.
5. Use the `sort` command to sort both files. Note that your name and birthday appear twice.
6. Use the `-u` option to sort the two files so that your name only appears once in the output.
7. Using the `sort` command, create a file named `familysort` that contains the sorted version of `familydates`. Do the same for the file `workdates` and name it `worksort`.

   Notice that the output is not displayed on the screen. If you list the files in your directory (`$ ls`), you should see the two files `familysort` and `worksort` listed. You can use the `cat` command to display their contents.
8. Try using the `-m` option to sort `familydates` and `workdates`. What happens?

9. Use the `-m` option to merge the sorted versions (`familysort` and `worksort`). Notice that this option does not sort the files; it only merges them. What additional option should you have specified?
10. Sort `familydates` and `workdates` in order of last names. Remember that field position numbering begins at zero.
11. The output to the previous `sort` command (**`$ sort +1 familydates`**) will not necessarily be in "true" alphabetical order. While all last names are in the correct order, the first names may not be. Use the appropriate option to sort `familydates` and `workdates` so that first and last names are in order.
12. Sort `familydates` and `workdates` in order of birth dates. First try using the `+2` option; any October, November and December months (10, 11, and 12) will precede those in February. Sort the files again, using the appropriate option so that the birthdates are sorted in numerical order.

Learning Activity Solution: Using the sort Filter

1. Create a file called `familydates` that contains the first names, last names, and birthdays of a few members of your family. Press Tab to separate the name from the birthday. The entries should use the numerical form of the date in the format month/day/year, for example:

   ```
   John Doe Tab 3/15/50
   ```

   Be sure to include your own name and birthday in the file.

   ```
   $ cat > familydates
   firstname lastname      birthday
   firstname lastname      birthday
   firstname lastname      birthday
   yourfirst yourlast      yourbirthday
   Ctrl/D
   ```

2. Use the `sort` command to print out your birthday list in alphabetical order by first name.

   ```
   $ sort familydates
       .
       .
       .
   ```

3. Use the `sort` command to print out your birthday list in **reverse** alphabetical order by first name.

   ```
   $ sort -r familydates
       .
       .
       .
   ```

4. Create another file (in the same format as `familydates`) called `workdates` that contains the first names, last names, and birthdays of a few of your coworkers. Again, be sure to include your name and birthday in this file.

```
$ cat > workdates
firstname lastname      birthday
firstname lastname      birthday
firstname lastname      birthday
yourfirst yourlast      yourbirthday
Ctrl/D
```

5. Use the `sort` command to sort both files. Note that your name and birthday appear twice.

```
$ sort familydates workdates
.
.
.
```

6. Use the `-u` option to sort the two files so that your name only appears once in the output.

```
$ sort -u familydates workdates
.
.
.
```

7. Using the `sort` command, create a file named `familysort` that contains the sorted version of `familydates`. Do the same for the file `workdates` and name it `worksort`.

   Notice that the output is not displayed on the screen. If you list the files in your directory (`$ ls`), you should see the two files `familysort` and `worksort` listed. You can use the `cat` command to display their contents.

```
$ sort -o familysort familydates
$ sort -o worksort workdates
```

8. Try using the `-m` option to sort `familydates` and `workdates`. What happens?

```
$ sort -m familydates workdates
.
.
.
```

   The output appears in a peculiar order, since the input files were not sorted as is required for the `-m` option.

9. Use the `-m` option to merge the sorted versions (`familysort` and `worksort`). Notice that this option does not sort the files; it only merges them. What additional option should you have specified?

```
$ sort -m familysort worksort
.
.
.
```

   You should have specified the `-mu` option so your name would not appear twice.

10. Sort `familydates` and `workdates` in order of last names. Remember that field position numbering begins at zero.

```
$ sort +1 familydates workdates
.
.
.
```

11. The output to the previous `sort` command (`$ sort +1 familydates`) will not necessarily be in "true" alphabetical order. While all last names are in the correct order, the first names may not be. Use the appropriate option to sort `familydates` and `workdates` so that first and last names are in order.

```
$ sort +1 -2 familydates workdates
.
.
.
```

12. Sort `familydates` and `workdates` in order of birth dates. First try using the `+2` option; any October, November and December months (10, 11, and 12) will precede those in February. Sort the files again, using the appropriate option so that the birthdates are sorted in numerical order.

```
$ sort +2 familydates workdates
.
.
.
$ sort +2n -2 familydates workdates
.
.
.
```

The grep Command

Overview

Another filter, called `grep` (**g**lobal **r**egular **e**xpression **p**rinter), searches its input for a pattern of characters, and outputs any line containing that pattern.

The format of the `grep` command is:

`grep` *`expression filename(s)`*

where:

- *`expression`* — is a pattern of characters
- *`filename(s)`* — refers to the file(s) to be searched

The output from `grep` is a list of lines from the file(s) that contain the expression.

```
$ grep tomato vegetables
tomato
```

If more than one file is given, the name of the file will precede the line in the output, as shown below:

```
$ grep tomato fruits vegetables
fruits:tomato
vegetables:tomato
```

Finding Patterns

The example shows a listing of the contents of a file called `price.fruit`.

Example 10–9 A File of Fruit Prices

```
$ cat price.fruit
apples                    .49
oranges                   .59
grapefruit               0.59
lemons                    .25
limes                     .33
cantaloupe               1.19
watermelon                .25
kiwi                     3.89
bananas                   .87
pineapples               1.49
grapes                   2.19
raspberries              3.39
strawberries             2.38
blueberries              1.89
cranberries              1.59
peaches                   .79
nectarines                .79
plums                     .69
```

The grep command in Example 10–10 looks for the pattern "berries" from Example 10–9 and prints out any line containing the pattern.

This pattern ("berries") is a string of characters containing no spaces or special characters. Sometimes you need to search for a pattern containing spaces or tabs. The operating system interprets spaces as argument separators. To ensure that the blank characters are recognized as part of the expression, you should always surround the expression with either single or double quotes, as shown below:

```
$ grep 'berries' price.fruit
```

OR

```
$ grep "berries" price.fruit
```

Example 10–10 Using grep to List Berries

```
$ grep 'berries' price.fruit
raspberries                    3.39
strawberries                   2.38
blueberries                    1.89
cranberries                    1.59
```

Finding Patterns in Specific Positions

Suppose you want to find the price of a pound of apples. You could use the command:

```
grep 'apples' price.fruit.
```

However, this command prints out the price of any fruit containing the pattern "apples," not just the price of the apples.

Use the special characters:

- ^ (caret) — to indicate to grep to match the expression if it is at the beginning of a line
- $ (dollar sign) — to indicate to grep to match the expression if it is at the end of a line

If you want only the price of apples (not pineapples or crabapples), you can specify that you want grep to print the pattern only if it occurs at the beginning of the line. To **anchor** a pattern to the beginning of a line, use the caret character (^), as shown below:

```
$ grep '^apples' price.fruit
apples                         .49
```

You can also anchor patterns to the end of the line using a dollar sign ($).

```
$ grep '79$' price.fruit
peaches                        .79
nectarines                     .79
```

When you use the special symbols, ^ and &, to anchor a pattern, you must enclose the entire expression in quotation marks. If you do not, it is interpreted incorrectly because ^ and $ have special meaning to the shell.

Any character that has a special meaning to the shell, as ^ and $, are known as **metacharacters.**

Finding Nonfixed Patterns

All patterns specified so far have been **fixed** patterns. In other words, they match only a specific, fixed set of characters. The grep command also has the ability to look for patterns that are not fixed. These patterns are known as **regular expressions.**

Regular expressions are a method of specifying patterns of character strings. They allow you to describe a set of strings instead of having to list every member of the set. Strings that fit the specified pattern are said to **match** the regular expression.

Regular expressions follow a standard notation, consisting of regular characters, white-space characters, and metacharacters. Regular characters match themselves. For example, the expression "de" matches any string that contains the characters "de". Some white-space characters also match themselves. A space is specified by pressing Spacebar. A tab is specified by pressing Tab.

One use of regular expressions is to specify a choice or range of characters. Choices are specified using square brackets ([ ]). To indicate a choice between several characters, place the characters inside brackets.

The example says to print any line in the file price.fruit that begins with the letter k, the letter l, or the letter s.

Example 10–11 Specifying a Choice of Characters Using grep

```
$ grep '^[kls]' price.fruit
lemons                          .25
limes                           .33
kiwi                           3.89
strawberries                   2.38
```

To indicate one character from a range, specify the range using the first character in the range, followed by a hyphen (-), followed by the last character in the range. Example 10–12 says to print every line in the price.fruit file that begins with a letter a through g.

Example 10–12 Selecting a Range of Characters Using grep

```
$ grep '^[a-g]' price.fruit
apples                          .49
grapefruit                     0.59
cantaloupe                     1.19
bananas                         .87
grapes                         2.19
blueberries                    1.89
cranberries                    1.59
```

Note

The pattern [1-50] appears to be 1 through 50. This pattern is actually interpreted as "one through five OR zero." To specify multiple digit numbers, each digit must be explicitly matched as in '[1][0-9]' to match the numbers 0 through 19.

Using Negation with grep

Sometimes it is useful to find lines that do **not** contain a specific pattern. There are two ways to do this:

- Using the `-v` option
- Using the caret symbol (^) within square brackets

To find all lines in the `price.fruit` file that do **not** end with a 9, you can use either of the commands shown in the example.

Example 10–13 Using Negation with grep

```
$ grep -v '9$' price.fruit
lemons                          .25
limes                           .33
watermelon                      .25
bananas                         .87
strawberries                   2.38
$
$ grep '[^9]$' price.fruit
lemons                          .25
limes                           .33
watermelon                      .25
bananas                         .87
strawberries                   2.38
```

To print out all lines that start with nonvowels in the second half of the alphabet, you could use the following command:

```
$ grep '^[^a-m^o^u]' price.fruit
watermelon                    .25
pineapples                   1.49
raspberries                  3.39
strawberries                 2.38
peaches                       .79
nectarines                    .79
plums                         .69
```

Notice that the caret character has a different meaning when it is inside brackets (negation) than when it is outside of the brackets (anchoring a pattern to the beginning of the line).

The `grep -v '^[a-mou]' price.fruit` command produces the same result as the example above.

Additional options to the `grep` command are shown in the following table.

Table 10–5 Other Options to the grep Command

Option	Function
`-c`	Counts the number of lines in which the pattern occurs for each file named
`-l`	Lists all file names that contain one or more occurrences of the pattern
`-n`	Prints the line number with the whole line for each occurrence of the pattern
`-i`	Considers uppercase and lowercase letters identical in making comparisons

Specifying Arbitrary Characters

Occasionally, you may want to match arbitrary patterns. Consider the situations listed in the table.

Table 10–6 Matching Arbitrary Characters

To Find All Lines in a File That . . .	Use the Expression	To Match
Are exactly 10 characters long	'^.....$'	Who knows? telephones Red Blue John Smith
Start with four-letter words beginning with the letter R	'^R... '	Rick says "hello." Ride 'em Cowboy Red Blue

(continued on next page)

Table 10–6 (Cont.) Matching Arbitrary Characters

To Find All Lines in a File That . . .	Use the Expression	To Match
End with six-letter words	'.....$'	Bring extra shoes. Who knows? Ride 'em Cowboy Red Blue

Notice that the line " Red Blue " is printed in all three cases, even though it does not specifically match the verbal description of items two and three. The period character matches **any** character in the ASCII set, including a space. If you want only letters, you must explicitly specify only letters. To find all lines in a file starting with four-letter words that begin with the letter R, this is the expression to use:

```
grep '^R[a-z][a-z][a-z]' file1
```

Finding Special Characters

Certain characters, such as the period and dollar sign, have special meanings when they are inside regular expressions. When you use these characters as part of a regular expression, they are interpreted. What if you actually want to find a period in a file? You must ensure that the period character is not given special meaning.

Use the backslash character (\) to indicate that the character immediately following it should not be given special meaning. The backslash is known as an **escape** character.

When you use the backslash to escape a special character, it is interpreted literally. Metacharacters have special meaning to the shell or certain commands like `grep`. Escaped metacharacters are also known as **literal** characters. When a metacharacter is escaped, it is accepted literally and not given special meaning. Within single quotation marks, you can escape the metacharacters listed in the table.

Table 10–7 Escape Character Sets for Metacharacters

Enter		To Find
\[	matches	[
\]	matches	]
\.	matches	.
*	matches	*
\$	matches	$

(continued on next page)

Table 10–7 (Cont.) Escape Character Sets for Metacharacters

Enter		To Find
\?	matches	?
|	matches	|
\^	matches	^
\\	matches	\

To find all items in the `price.fruit` file whose prices end in .59, use the command:

```
$ grep '\.59$' price.fruit
oranges                         .59
grapefruit                     0.59
cranberries                    1.59
```

There are a few other situations where matching certain characters can be confusing:

- To include a hyphen as part of a character set inside square brackets, make sure the hyphen is at the beginning or the end of the character set. For example:
  - `[a-c]` matches strings containing a, b, or c
  - `[-ac]` matches strings containing -, a, or c
- If a single quote (apostrophe) is part of the expression to match, make sure the entire expression is enclosed in double quotes.
- If double quotes are part of the expression to match, the entire expression should be enclosed in single quotes.

Finding Repeated Characters

You can use a caret before the starting character, and you can use a dollar sign after the ending character, but so far, you have no way of telling the operating system that any number of arbitrary characters may fall between the two. Suppose you want to use an expression to match all lines in a file that start with one character and end with another.

An asterisk (*) is used inside regular expressions to indicate that the character immediately preceding it can be repeated any number of times, including zero.

Using metacharacters to match repeating patterns is called **closure**. Use the following format to specify closure in regular expressions:

`grep` *`character*`* *`filename`*

`character` can be a:

- Regular character
- Character enclosed in square brackets

- Arbitrary character (denoted by a period)
- Literal character

To print out all items in the `price.fruit` file that start with the letters `a` through `g` whose prices are under one dollar, use the command shown in the example.

Example 10–14 Using * to Specify Repeating Characters

```
$ grep '^[a-g].*[0 ]\...$' price.fruit
apples                          .49
grapefruit                     0.59
bananas                         .87
```

The notation `.*` means that any number of characters can be between the first part of the pattern (`^[a-g]`) and the last part of the pattern (`[0 ]\...$`)

Note the space following the zero in the brackets, indicating that there is a zero or a space. If the space is missing, the results are different from those shown in Example 10–14.

If the file `price.fruit` contained tabs instead of spaces, the format in Example 10–14 would be:

```
grep '^[a-g].*[0Tab]\...$' price.fruit
```

If the file contained both tabs and spaces the format would be:

```
grep '^[a-g].*[0Tab ]\...$' price.fruit
```

Additional examples of using metacharacters to match repeating characters are shown in the following table.

Expression	Matches
`'we*k'`	wk wek week weeek
`'w[eao]*k'`	wk wok weak weeeoaok
`'w.*k'`	wk working two tickets walk

Learning Activity: Using grep

This lab exercise provides practice using the `grep` command to find expressions in the file `veg.price`, shown below:

```
lettuce              .89
tomatoes             .69
broccoli             .79
cauliflower          .87
parsley             2.14
avocado             1.19
carrots              .69
celery               .59
string-beans        1.29
onions               .29
asparagus            .89
corn                 .85
```

1. Print out all the vegetables in the file `veg.price` whose names begin with the letter c.
2. Print out all the vegetables with prices ending in a 9.
3. Print out all the vegetables with prices that do **not** end in 9.
4. Print out all the vegetables with names that start with the letters a through m.
5. Print out all of the vegetables whose prices are under 80 cents.
6. Print all of the vegetables whose names start with the letters c, a, or p, with prices under one dollar.

Learning Activity Solution: Using grep

This lab exercise provides practice using the `grep` command to find expressions in the file `veg.price`, shown below:

```
lettuce              .89
tomatoes             .69
broccoli             .79
cauliflower          .87
parsley             2.14
avocado             1.19
carrots              .69
celery               .59
string-beans        1.29
onions               .29
asparagus            .89
corn                 .85
```

1. Print out all the vegetables in the file `veg.price` whose names begin with the letter c.

```
$ grep '^c' veg.price
cauliflower          .87
carrots              .69
celery               .59
corn                 .85
```

2. Print out all the vegetables with prices ending in a 9.

```
$ grep '9$' veg.price
lettuce                 .89
tomatoes                .69
broccoli                .79
avocado                1.19
carrots                 .69
celery                  .59
string-beans           1.29
onions                  .29
asparagus               .89
```

3. Print out all the vegetables with prices that do **not** end in 9.

```
$ grep -v '9$' veg.price
cauliflower             .87
parsley                2.14
corn                    .85
```

OR

```
$ grep '[^9]$' veg.price
cauliflower             .87
parsley                2.14
corn                    .85
```

4. Print out all the vegetables with names that start with the letters a through m.

```
$ grep '^[a-m]' veg.price
lettuce                 .89
broccoli                .79
cauliflower             .87
avocado                1.19
carrots                 .69
celery                  .59
asparagus               .89
corn                    .85
```

5. Print out all of the vegetables whose prices are under 80 cents.

```
$ grep '[0 ]\.[0-7][0-9]$' veg.price
tomatoes                .69
broccoli                .79
carrots                 .69
celery                  .59
onions                  .29
```

6. Print all of the vegetables whose names start with the letters c, a, or p, with prices under one dollar.

```
$ grep '^[acp].*[0 ]\...$' veg.price
cauliflower             .87
carrots                 .69
celery                  .59
asparagus               .89
corn                    .85
```

Pipelines

Overview

There are situations when you want to use the output from one command or program as input to another, therefore constructing a **pipeline** of commands. A **pipe** is a connection that allows one command to get its input from another. For example, one command writes its output to the pipe and another command reads that output and uses it as input from the pipe. Filters are often used as the intermediate commands in pipelines.

A pipeline has the same effect as:

- Storing the output from one command or program in a temporary file
- Using the temporary file as input for a second command or program
- Removing the temporary file

Using Pipelines

Piping output from one program or command into another is done using the vertical bar (|) symbol.

Example 10–15 and Example 10–16 both perform the following tasks:

- Generate a file that is a long listing of files in your current directory
- Sort that listing numerically by the size of the files
- Search the sorted list for all files that end in `.txt`

Example 10–15 Not Using a Pipeline

```
$ ls -l > all.lst
$ sort -o all.sort +4n all.lst
$ grep '\.txt$' all.sort
-rw-r--r--  1 you   users         178 Jan 18 11:21 vegetables.txt
-rw-r--r--  1 you   users         365 Jan 18 11:43 report2.txt
-rw-r--r--  1 you   users        2132 Jan 18 11:42 chap15.txt
```

Example 10–16 Using a Pipeline

```
$ ls -l | sort +4n | grep '\.txt$'
-rw-r--r--  1 you   users         178 Jan 18 11:21 vegetables.txt
-rw-r--r--  1 you   users         365 Jan 18 11:43 report2.txt
-rw-r--r--  1 you   users        2132 Jan 18 11:42 chap15.txt
```

In Example 10–15, Not Using a Pipeline:

- First you have to redirect the output of the `ls -l` command to a file (`all.lst`).
- After you sort the file, it must also be redirected to a file (`all.sort`).
- Finally, you can search for the files that end in `.txt`.
- If you no longer want the intermediary files `all.lst` and `all.sort`, you must delete them using the `rm` command.

In Example 10–16, Using a Pipeline:

- You can issue all three commands with one command line.
- You do not have to redirect output to any file, in either the `ls` or `sort` commands. And, you do not need to specify input files for either the `sort` or `grep` commands.
- You do not need to delete any intermediary files.

The output of the `grep` command in Example 10–16 could have been redirected to a file, ending the pipe as shown:

```
$ ls -l | sort +4n | grep '\.txt$' > grep.all
```

You then could have used `more` to display the file `grep.all`.

However, you could not have redirected the output of `grep` and continued the pipe. This would not produce any screen display:

```
$ ls -l | sort +4n | grep '\.txt$' > grep.all | more
```

The tee Command

The `tee` command reads from standard input, and writes the input both to standard output and to a specified file name. This is useful if you want to save the output of a command to a file, and watch it on your screen at the same time, or if you want to save the intermediate output of a pipeline while allowing processing to continue.

The format of the `tee` command is:

```
tee [-a] filename
```

If the `-a` option is used, the output will be appended to the file instead of replacing the old contents.

With the `tee` command, the pipeline in the previous section could be rewritten as:

```
$ ls -l | sort +4n | grep '\.txt$' | tee grep.all | more
```

Learning Activity: Pipelines

1. From your home directory, list all the files in `/usr` and its subdirectories. This listing is very long and difficult to read because it scrolls off the screen so fast.
2. Again, list all the files in `/usr` and its subdirectories. This time **pipe** the output to the command `more`. This displays the files one screenful at a time and allows you to read the listing slowly. When the prompt is displayed, press Spacebar to see another screenful, or q to **q**uit displaying the listing.
3. Without creating any new files, generate a listing in order of size, of all files in your directory.

Learning Activity Solution: Pipelines

1. From your home directory, list all the files in `/usr` and its subdirectories. This listing is very long and difficult to read because it scrolls off the screen so fast.

```
$ ls  -R  /usr
   .
   .
   .
```

2. Again, list all the files in `/usr` and its subdirectories. This time **pipe** the output to the command `more`. This displays the files one screenful at a time and allows you to read the listing slowly. When the prompt is displayed, press Spacebar to see another screenful, or q to **q**uit displaying the listing.

```
$ ls  -R  /usr | more
   .
   .
   .
```

3. Without creating any new files, generate a listing in order of size, of all files in your directory.

```
$ ls -al | sort +4n
   .
   .
   .
```

The awk Utility

Overview

The awk utility is a report generator that processes ASCII text files and generates reports involving selective retrieval and string manipulation. It was developed at Bell Laboratories and named after its inventors: Alfred V. **A**ho, Peter J. **W**einberger, and Brian W. **K**ernighan. It is one of the most flexible methods of manipulating text and retrieving specific information from files. Once the information is retrieved, awk can print it, alter it, reformat it, or ignore it. The awk utility is primarily used to generate tables and reports from files of raw data. The awk program is the most powerful of the text processing utilities.

The awk utility works by searching through files for patterns and performing specified actions on the lines found by the search. It has many applications including:

- Generating reports
- Creating mailing lists
- Database queries
- Writing tables of text

The awk utility is more than a report generator; it is a very sophisticated programmable pattern scanning and processing language. awk programs can include sophisticated flow control code such as "if-else-if" statements used in for loops and while loops when generating conditional reports.

This section discusses only a few of the most basic awk features from an interactive user's point of view.

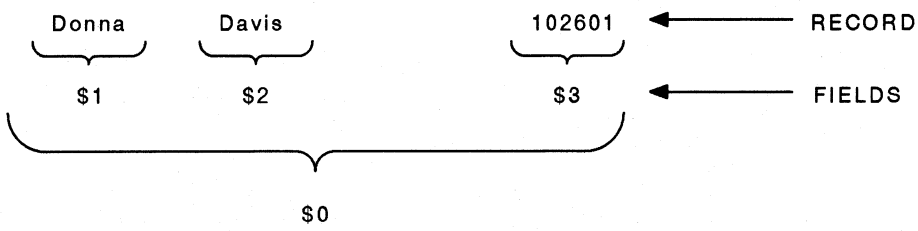
Note

For more information on any topic covered in this section, refer to the documentation listed under resources in this workbook or use the man awk command to gain information in the manual pages.

awk Terminology

In awk, as in most filters, the term **record** refers to a line of input. The different parts of a record are known as **fields**. The fields are separated by a field separator, which, by default, is a space. The way fields are numbered by awk is different from the way they are numbered by sort. Numbering of awk fields begins with 1 for the first field in the record, 2 for the second field, up to n for the nth field. Field 0 refers to the entire record. To reference the fifth field in a record, the notation is $5. Figure 10–1 demonstrates the way fields are numbered.

Figure 10–1 The awk Field Numbering Scheme



TTB_X0993_88

awk Program Structure

awk programs are defined in terms of **patterns** and **actions.**

The format of an awk program is:

```
<pattern_1> { <action_1> }
<pattern_2> { <action_2> }
            .
            .
            .
<pattern_n> { <action_n> }
```

The awk utility uses <patterns> to select the records. The input file is processed one line at a time, and as each record is read, it is compared to <pattern_1>. If <pattern_1> matches the record, <action_1> is executed. This select and execute process is repeated for every pattern/action pair in the awk program. Then the next record is read and processed, beginning again with the first pattern/action pair. This procedure continues until all records in the input file(s) have been processed.

If any <pattern> is missing, the associated <action> is performed for all records in the input file(s). If any <action> is missing, awk prints every record that matches the associated <pattern>. awk does not automatically print every line in the original file. If no action is given, the default action is to print the record. If an action is supplied, however, the record is not printed unless a print statement is included in the action. Actions may consist of a single command or several. The left brace { of an <action> must appear on the same line as the associated <pattern>. However, <actions> can be spread over several lines.

Using awk

There are two ways to use awk.

- Invoke awk on the command line:

```
$ cat names
Donna Davies
Peter Parker
Paul Baker
$ awk '{print $2, $1}' names
Davies Donna
Parker Peter
Baker Paul
```

- Create an awk program and run awk with the -f option. This way is preferred for more complex commands and reports. Example 10–17 uses the /etc/passwd file. The result may vary from system to system.

Example 10–17 Invoking awk Using a File for Patterns and Actions

```
$ cat list.awk
{
FS = ":"
OFS = "Tab"
}
{print $1, $5}
$ awk -f list.awk /etc/passwd | tail -5
dan     David Nolan
seg     Susan Greenburg
mw      Marcy Williams
eap     Eric Patterson
dbs     Diana Sanchez
```

Expressions and Variables

The following table provides some of the expressions used to write awk programs. These expressions are really predefined variables that can be changed within the awk program to a new value.

Table 10–8 Expressions

Expression	Significance
BEGIN <action>	The <action> is performed once, before any other <action> in the program, and before any input is processed
END <action>	Causes <action> to be performed after all other <action> in the program, and after all input has been processed
NR	The number of records that have been read at the time of evaluation
NF	The number of fields in the current record
FS	The input field separator
OFS	The output field separator

Comparison Operators

These arithmetic expressions are the same as in the C programming language and are used by awk to evaluate and compare expressions.

Table 10–9 Comparison Operators

Symbol	Definitions
==	Is equal to
!=	Is not equal to
>	Is greater than
<	Is less than
>=	Is greater than or equal to
<=	Is less than or equal to

Example 10–18 uses the NF and FS expressions and the == comparison operator to search the /etc/passwd file for users with no shell or with the same User and Group ID. It prints the results to the screen.

Example 10–18 Using awk with Expressions and Operators

```
$ awk ' BEGIN { FS=":" }
> NF==6    {print $1, "has no shell" }
> $3==$4   {print $1, "has same uid and gid" }
> END      {print NR, "records processed" }
> ' /etc/passwd
nobody has same uid and gid
daemon has same uid and gid
uucp has same uid and gid
who has same uid and gid
9 records processed
$
```

The **BEGIN** expression sets the field separator before any input is processed. The **END** expression causes the number of records processed to be printed after all input records have been processed.

If the **END** is not present, the statement to print the number of records is executed after each record is processed.

Using Logical Operators with awk

Logical operators are Boolean statements that compare two or more expressions.

- **AND** (&&): Condition 1 **AND** condition 2 are both true
- **OR** (| |): Either condition 1 **OR** condition 2 is true

The following file, phones, is used to illustrate the use of awk in Example 10–19.

```
$ cat phones
Maynard MA (508)493-4123
Nashua NH (603)884-2413
Cambridge MA (617)654-9531
Springfield MA (413)747-3452
Stamford CT (203)258-3232
Burlington MA (617)221-3895
Merrimack NH (602)884-8230
Novi MI (313)344-1839
Kalamazoo MI (616)384-8310
Littleton MA (508)952-4839
Augusta ME (207)271-2211
Lancaster, MA (508)874-6021
Burlington VT (802)657-2895
Charleston SC (803)744-9046
Greenville SC (803)288-3627
Philadelphia PA (215)246-7028
Rochester NY (716)385-8240
Syracuse NY (315)453-1057
Boston MA (617)106-5472
Providence RI (401)107-8932
```

Example 10–19 shows **&&** and **| |** usage within awk programming. The phones file contains several telephone numbers. The first awk program, eastmass.awk, prints any records where the state is Massachusetts **AND** the area code is 617. If the state is Massachusetts and the area code is not 617, the record is not printed. The second awk program, massri.awk, uses the **OR** operator to print any record where the state is either Massachusetts **OR** Rhode Island.

The following example includes the use of the substr command which extracts a specific part of an argument string. In this case, it is used to extract parentheses around the phone numbers and replace the output with hyphens.

Example 10–19 Using Logical Operators in awk Programs

```
$ cat > eastmass.awk
($2 == "MA") && (substr($3, 1, 5) == "(617)") {
print $1 " --- " substr($3, 6)
}
Ctrl/D
$
$ awk -f eastmass.awk phones
Cambridge --- 654-9531
Burlington --- 221-3895
Boston --- 106-5472
$
$ cat > massri.awk
($2 == "MA") || ($2 == "RI") {
print $1, $2 " --- " substr($3, 2, 3) "-" substr($3, 6)
}
Ctrl/D
$
$ awk -f massri.awk phones
Maynard MA---508-493-4123
Cambridge MA---617-654-9531
Springfield MA---413-747-3452
Burlington MA---617-221-3895
Littleton MA---508-952-4839
Lancaster, MA---508-874-6021
Boston MA---617-106-5472
Providence RI---401-107-8932
$
```

Learning Activity: Using awk

Using awk

What is the awk command to print the list of phone numbers from the phones file, first, followed by the city and state?

Searching for Patterns in awk

Determine the awk command to search the /etc/passwd file for any user with a password field of just *, and display their username and real name.

Writing awk programs

Write an awk program that will search the phones file for any phone number with a state of NY or CT and print out the numbers followed by city and state.

Use this program to produce a directory of phones located in NY or CT, sorted by phone number, with city and state.

Learning Activity Solution: Using awk

Using awk

What is the awk command to print the list of phone numbers from the phones file, first, followed by the city and state?

```
$ awk ' {print $3, $1, $2 } ' phones
    .
    .
    .
```

Searching for Patterns in awk

Determine the awk command to search the /etc/passwd file for any user with a password field of just *, and display their username and real name.

```
$ awk '{FS=":"} ($2=="*") { print $1, "->", $5 } ' /etc/passwd
    .
    .
    .
```

Writing awk programs

Write an awk program that will search the phones file for any phone number with a state of NY or CT and print out the numbers followed by city and state.

Use this program to produce a directory of phones located in NY or CT, sorted by phone number, with city and state.

```
$ cat > phones.awk
($2=="NY") || ($2=="CT") { print $3, $1, $2 }
Ctrl/D
$ awk -f phones.awk phones | sort
(203)258-3232 Stamford CT
(315)453-1057 Syracuse NY
(716)385-8240 Rochester NY
```

Summary

Redirecting Input and Output

The table below summarizes the symbols used to redirect standard input and output with the Korn shell.

Symbol	Example	Function
<	`$ cat < todo.lis`	Reads standard input from a file named `todo.lis` instead of from the keyboard (standard input)
>	`$ ls -l > listing`	Writes the output of the command `ls -l` to a file named `listing` instead of to the terminal screen (standard output)
>>	`$ ls -al >> listing`	Appends the output of the command `ls -al` to an existing file named `listing`

Filters

A filter is a command that reads standard input, processes it, and prints the results to standard output.

The `wc` filter counts the number of lines, words, and characters a file contains and prints the results on the terminal screen.

- `-l` counts only the number of lines
- `-w` counts only the number of words
- `-c` counts only the number of characters

Sorting

The `sort` filter sorts each record in a file alphabetically, or numerically, and displays the results on the terminal screen.

The grep Command

The `grep` command searches for one or more files for regular expressions specified on the command line.

Regular expressions are a method of specifying patterns of character strings. Strings that fit the specified pattern are said to match the regular expression. Regular expressions follow a standard notation consisting of regular characters, white-space characters, and metacharacters.

Pipelines

Pipes allow the output from one process to be the input to another eliminating the need for temporary files. Two processes may be piped together if the process on the left side of the pipe symbol (|) writes output to standard output and the process on the right side of the pipe accepts input from standard input. For example: `$ ls -l | sort +4n`

The pipe symbol (|) can only be used with a redirection symbol > on the same command line if the command with the redirection symbol is the last part of the pipe.

For example: `$ ls -l | sort +4n > files_by_size`

The awk Utility

The awk utility is used for pattern matching in text files.

- awk has a field and record expression scheme that takes the first field as $1 to $n for the last field in a record.
- awk uses arithmetic expressions similar to the C programming language and variables definable by the program as in scripts.
- awk can be used on the command line, or awk commands can be read from a program file.

11

Communicating with Other Users

Communicating with Other Users

Introduction

This chapter provides information on how to communicate with other users on the UNIX operating system. The ability to communicate with others is a key feature of the operating system. The information age of networked computers requires fast and efficient means of communication and UNIX provides different ways to electronically communicate with other users. This chapter discusses how to:

- Obtain information about other system users.
- Send and receive messages using the `mail` utility.
- Communicate with users currently logged in to your system or other systems that you are networked to using the `write` utility.

Objectives

To electronically communicate with other users, you should be able to:

- Use user information commands like `who`, `users`, and `finger`
- Use the `mailx` utility to electronically communicate with other users on a UNIX operating system
- Use the `mailx` utility to send, receive, select, and maintain mail messages
- Customize your own mail environment
- Create and utilize a mail folder
- Describe mh (Message Handler) mail
- Use the `write` utility

Resources

- *OSF/1 User's Guide*

Obtaining User Information

Overview

This section discusses a few elementary and useful commands that allow you to quickly get information about users and the system. For example:

- Finding out who is logged in
- Obtaining the system date and time

Finding Out Who is Logged In

The `who` command shows you who and where users are logged in. `who` examines the system file `/etc/utmp`, then it lists the login name, terminal name, and login date and time for each user, as shown in the example.

Example 11–1 who Command Output

```
$ who
anderson tty00   Jan 14 08:01
johnston tty02   Jan 14 10:30
smith    ttyp0   Jan 14 14:41
thomas   ttyp3   Jan 14 15:22
```

The `who` command has two other forms:

- `who am i`

  Displays information only about the user logged in at the terminal where the command is entered. This command is useful when you must determine your initial login name.

  ```
  $ who am i
  smith  ttyp0   Mar 14 14:41
  $
  ```

- `whoami`

  Displays the login name of the current user. It examines the system file `/etc/passwd` and then displays current login name based on that file.

  ```
  $ whoami
  smith
  $
  ```

  After using the substitute user (`su`) command to change your current user name, this is useful to remind yourself who you currently are. Remember to use `who am i` to determine who you **really** are.

Obtaining a Quick User List

The `users` command displays a one-line list of the login names of all current users on your system.

```
$ users
mcfelty johnston smith jones
$
```

This is useful when you quickly want to see who is currently logged in so you can communicate with them. The `users` command is often used in shell scripts as a front end to filters such as `wc` to determine how many users are currently on the system.

Obtaining More Information About Users

The `finger` command displays a detailed list of users, including users' real names, amount of time elapsed since they last entered a command, and their phone extensions, as shown in Example 11–2.

The format for the `finger` command is:

`finger [option] [user ...]`

finger Command Options

Some common options to the `finger` command are shown in the following table.

Table 11–1 Common finger Command Options

Option	Function
`-i`	Produces a quick listing with idle times
`-q`	Produces a quick listing, containing only login name, terminal name, and login time
`-w`	Forces narrow, short format listing

finger Command Output

The output of the `finger` command differs depending upon the options used with the command.

The following example shows sample output of the `finger` command used several ways; without specifying options, specifying a user (pmj), and specifying the options, `-i`, `-q`, and `-w`.

Example 11–2 finger Command Output

```
$ finger

Login         Name          TTY    Idle   When       Office
John miller   Manager       d0     10     Tue 08:37  ext 344
pmj           Pat Jones     d2     1      Tue 11:06  ext 347

$ finger pmj
Login name: pmj                        In real life: Pat Jones
Directory: /usr/users/pmj              Shell: /bin/csh
Last login Fri Jan 4  13:47 on ttyd2
```

(continued on next page)

Example 11–2 (Cont.) finger Command Output

```
$finger -i
Login       TTY             When              Idle
mcd         ttyp0     Mon Jan 27 09:22    13 minutes
hgh         ttyp1     Mon Jan 27 09:55    1 hour 37 minutes
evant       ttyp3     Mon Jan 27 11:29    22 seconds
tlb         ttyp2     Mon Jan 27 10:08    35 minutes
tlb         ttyp5     Mon Jan 27 10:56    14 minutes

$ finger -q
Login       TTY             When
mcd         ttyp0     Mon Jan 27 09:22
hgh         ttyp1     Mon Jan 27 09:55
evant       ttyp3     Mon Jan 27 11:29
tlb         ttyp2     Mon Jan 27 10:08
tlb         ttyp5     Mon Jan 27 10:56

$ finger -w
Login     TTY Idle    When            Office
mcd        p0   17 Mon 09:22
hgh        p1 1:41 Mon 09:55
evant      p3    1 Mon 11:29
tlb        p2   39 Mon 10:08
tlb        p5   18 Mon 10:56
$
```

You can also use the `finger` command with remote systems on your network. To obtain information about users on remote systems, include an at sign (@) followed by the node name of the system. For example, if you want a listing of the users on node `sys002`, you would enter:

```
$ finger @sys002
```

Learning Activity: Finding System Users

1. Issue the `who` command to see who is currently logged in to the system.
2. Issue the `who am i` command to determine your initial login name and terminal ID.
3. Issue the `whoami` command to see your current login name.
4. Issue the `users` command to quickly see how many users are on the system.
5. Issue the `finger` command to find out the full names of people on the system.
6. Issue the `finger` command with a user name to find out more information about a specific user.

Learning Activity Solution: Finding System Users

1. Issue the `who` command to see who is currently logged in to the system.

```
$ who
mcflower    ttyp0        Feb 14 10:11
evant       ttyp1        Feb 14 11:25
$
```

2. Issue the `who am i` command to determine your initial login name and terminal ID.

```
$ who am i
evant       ttyp1        Feb 14 11:25
$
```

3. Issue the `whoami` command to see your current login name.

```
$ whoami
evant
$
```

4. Issue the `users` command to quickly see how many users are on the system.

```
$ users
evant mcflower
$
```

5. Issue the `finger` command to find out the full names of people on the system.

```
$ finger
Login       Name              TTY Idle    When           Office
mcflower Bob McFlower          p0 1:36 Fri 10:11          D66/E4
evant    Evan Tyler            p1      Fri 11:25          D67/E4
```

6. Issue the `finger` command with a user name to find out more information about a specific user.

```
$ finger evant
Login name: evant                       In real life: Evan Tyler
Directory: /usr/users/evant             Shell: /bin/ksh
On since Feb 14 11:25:44                2 minutes 22 seconds Idle Time
on ttyp1 from sys002.dec.com
No Plan.
$
```

Introduction to Mail

Overview

Information exists everywhere. A key to using this information effectively is to distribute it efficiently. Today, the most efficient way to distribute information is electronically.

One of the most important functions of today's computer is its communication capability. Benefits of electronic communications include: faster distribution of information, easier access to that information, up-to-date and reliable information, the elimination of information bottlenecks, and a consequent improvement in information management.

The UNIX operating system allows you to electronically communicate with other users. You can receive, select, view, send and maintain mail messages. Messages can consist of individual memos or entire documents. You can send messages to system users who are either logged in (active) or who are logged out (nonactive).

Electronic mail is available on your UNIX system through command-line execution or through a window application menu. Sending and receiving mail in either environment is dependent upon the computer equipment you are using and personal preference. This section explains the use of electronic mail through command-line execution.

Commands for Invoking mail

The DEC OSF/1 `Mail` and `mailx` utilities are comparable to the `mail` utility in a UNIX or ULTRIX environment. If you are not using a DEC OSF/1 system, read the `man` pages for the `mail` utility on your system to check for differences.

The DEC OSF/1 commands for invoking mail, `Mail` and `mailx` are equivalent; therefore, you can use either command. In fact, both commands use the same `man` pages.

There is also a `mail` utility on DEC OSF/1, but it is a stripped-down mail handler, and does not have the features of `mailx`. This course does not cover the DEC OSF/1 `mail` utility.

Why `Mail` and `mailx`? The SVID3 standard requires `mail` and `mailx`, and the POSIX 1003.2 and XCU4 standards require `mailx`. Also, BSD uses `Mail`.

Note

Be careful with capitalization using `Mail`. If you prefer to stay with lowercase letters, use the `mailx` command.

Sending Mail

The following table shows how to send mail to other users, whether or not they are currently logged in.

Table 11–2 Sending Mail

System Prompt	Enter
$	`Mail` username, and press Return (where username is the desired recipient).
Subject:	The subject of your message, and press Return; or press Return if you choose to leave the subject blank.
[no prompt]	The body of your message, then press Return to move the cursor to the beginning of the next line. Press Ctrl/D to end the message.
Cc:	Usernames to whom carbon copies/additional copies should be sent. For several users, separate names with spaces or commas. Cc: may not appear on your system. Press Return when the list is complete to send the mail.
$	Indicates you have successfully sent your mail message.

The following example shows a mail session that sent a message to three system users; jar, kls, and sjg. A carbon copy of this mail message was also sent to ske.

Example 11–3 Sending Mail Session

```
$ Mail jar kls sjg
Subject: Book Modification [Return]
We need to get together sometime very soon to discuss the
modifications necessary if the book is ever going to be
published.
I sent a copy to the publishers last week to see if they are at
all interested, but obviously we have to make a lot of changes to make
it conform somewhat to industry standards.  How's Friday at 2 pm for a
meeting?

[Return]
[Ctrl/D]
Cc: ske  [Return]
$
```

Methods for Editing Mail

When you are entering the text of your mail message, editing is limited only to the current line. This can be frustrating, but avoidable by using one of the following three methods.

Editing a File Before Mailing It

Method 1

Create and edit a file (for example, `letter.mail`) and send this file.

```
Mail dan kls ar rln < letter.mail
```

This command sends the document `letter.mail` to users dan, kls, ar and rln. This process is useful when sending entire documents to other users. Notice the redirection symbol <. Notice that the Subject field is blank.

Editing a File While in Mail

Method 2

If you have been prompted, you can use an editor, as shown in the table. Enter `~e` or `~v` at the beginning of a blank line.

Table 11–3 Selecting Your Editor in Mail

Enter	Function
`~e`	Uses the editor defined by the **EDITOR** variable; the default editor is the `ed` line editor if **EDITOR** is not defined
`~v`	Uses the editor defined by the **VISUAL** variable; the default is `vi` if **VISUAL** is not defined

Editing and Receiving Mail

Example 11–4 emphasizes how to use the `vi` editor, however `emacs`, `ed` or `ex` can be used to edit as well. Define your editor of choice in the `.mailrc` file to be discussed in the next section.

```
set VISUAL=/usr/bin/emacs
```

The ~v command then activates the emacs editor in mail. To alternate editors, the **EDITOR** variable can also be defined. For example, set EDITOR=/usr/bin/vi to use the vi editor with the ~e command in Mail.

A spell checking program is also available in mail that can be used to check for spelling messages. To use its features, enter the following:

```
~w spellchk
~!spell spellchk
```

A list of misspelled words highlighted by the ! prompt appears in the mail message. An editor must be used (~v or ~e) to correct the spelling errors.

Sending and Receiving Mail

The following example shows how to send and receive a mail message.

Example 11–4 Sending and Receiving Mail

```
$ Mail smith
Subject: Return
~s Sample Send Mail Editing Session
~v
(Note: Typing an i causes vi to enter the insert mode)
This is a sample editing session using the vi editor.
Notice how the text wraps automatically when you reach the end of the
line.  Don't forget to press Esc when you want to stop editing.
Then enter :wq to return to the send mail session.
Esc
:wq

"/tmp/Re00120" 4 lines, 242 characters
(continue)
Ctrl/D
Cc: Return
You have new mail.
$ Mail
Mail $Revision: 4.2.4.2 $  Type ? for help.
"/usr/spool/mail/smith": 1 message 1 new
>N  1 smith  Thu Feb  3 10:59  11/399 "Sample Send Mail Editing Sess"
? Return

Message  1:
From smith Thu Feb  3 10:59:15 1994
Date: Thu, 3 Feb 1994 10:59:12 -0500
From: smith <Jim Smith>
To: smith
Subject: Sample Send Mail Editing Session
Status: R

This is a sample editing session using the vi editor.  Notice how the
test wraps automatically when you reach the end of the line.  Don't
forget to press Esc when you want to stop editing.  Then enter :wq to
return to the send mail session.

? q Return
$
```

Using ~r to Send a Named File

Method 3

If the file already exists, it can be read into the message with `~r`, as shown in the following sequence:

```
$ mail jar
Subject: Book Modification [Return]
~r file1  [Return]
"file1" lines/characters"
[Ctrl/D]
Cc: kls sjg  [Return]
$
```

Using Tilde in Mail

The tilde followed by another character is referred to as a **tilde escape**. `~v` is one example of how the tilde (`~`) can be used within the `mail` utility. Like `~v`, they are only recognized at the beginning of a blank line.

Some commonly used tilde escapes are shown in the table. For a full list of the available tilde escapes, consult the *OSF/1 User's Guide*, Appendix A under `Mail`, or enter the option `~?`.

Table 11–4 Tilde Escape Functions

Option	Function
`~e`	Invokes the editor designated by the **EDITOR** variable or the `ed` editor (default)
`~v`	Invokes the editor designated by the **VISUAL** variable or the `vi` editor (default)
`~b username(s)`	Adds `username(s)` to the list of people to receive blind copies of the message
`~c username(s)`	Adds `username(s)` to the list of carbon copy recipients
`~f message_list`	Includes one or more additional messages in the current message to forward to the recipients
`~p`	Prints out header fields, followed by the entire contents of the message so far
`~r filename`	Reads the named file into the message
`~s string`	Uses the string as the subject field in the message header
`~t username(s)`	Adds `username(s)` to the list of direct recipients
`~w filename`	Writes a message to the file, `filename`
`~!` command	Starts a shell, runs `command`, returns to editor
`~?`	Displays a help list of tilde escapes

Tilde escapes make sending mail messages easier and faster. The following example shows how to use tilde escapes when sending mail.

Notice how ~s uses the string as the subject field, ~t adds pnh to the recipient list, ~c adds ard to the Cc list, and ~p prints the entire message including header information.

Example 11–5 Using Tilde Escapes to Send Mail

```
$ Mail jar kls sjg

~s current thoughts on the book project
We need to get together sometime very soon to discuss the
modifications necessary if the book is ever going to be
published.  I sent a copy to the publishers last week to see
if they are at all interested, but obviously we have to make a
lot of changes to make it conform somewhat to industry
standards.  How's Friday at 2 pm for a meeting?
~t pnh
~c ard
~p

Message contains:
To: jar kls sjg pnh
Subject: current thoughts on the book project
Cc: ard

We need to get together  sometime  very  soon to discuss the
modifications  necessary  if the  book is  ever  going to be
published.  I sent a copy to the publishers last week to see
if they are at all interested, but obviously we have to make
a lot of  changes  to  make  it conform somewhat to industry
standards.  How's Friday at 2 pm for a meeting?
(continue)
Ctrl/D
$
```

Reading Mail

Now that we have discussed how to send mail, let's look at how to read it. If you have mail when you log in, you are informed with the message: "You have mail."

To read your mail, type the Mail command and press Return, as shown in the example.

Example 11–6 Mail Menu

```
$ Mail
Mail $Revision: 4.2.4.2 $  Type ? for help.
"usr/spool/mail/sjg": 3 messages 3 new
 U  1 sjg  Wed  Feb  9 13:25 11/295 "Don't you dare forget"
>N  2 dan  Wed  Feb  9 22:04 11/302 "current thoughts about the book p"
 N  3 ard  Thu  Feb 10 06:35 11/273 "pls rspnd asap"
?
```

When reading your mail, the numbered menu of your current mail messages will appear on the screen followed by the mail prompt, which is a question mark ?. When you are ready to return to the $ prompt, enter ex or x or q.

As you study the menu, notice that symbols U, N, and > precede the message entries. The table describes the meaning of those symbols.

Table 11–5 Mail Symbols

Symbol	Meaning
N	New message. Mail has no record of your having seen it before.
U	Unread message. You have seen the message header in the mail menu, but did not read it at that time.
>	Points to the current message. Immediately after you invoke mail, the > points to the first new message on the list.

Examine the first menu item of Example 11–6 from left to right:

U means you have not yet read this mail message

1 means this is message number 1

sjg indicates **who** sent the message

The next four fields, **Wed Feb 9 13:25**, show the day of the week, month, date, and time the message was sent.

11/295 shows the number of lines and characters in the message, including the header.

The last field, "**Don't you dare forget**," is the **subject** message.

The message headers in the menu are listed in the order in which they were received.

Additional Mail Commands

The following table contains most of the commands you need to know to read and manipulate mail. As you study the table, remember these two definitions:

- An `msg_list` is a list of one or more message numbers, the numbers occurring before the user who sent the message in the mail menu, separated by spaces. In most instances the list can be omitted, and the last message typed is used.
- A `user_list` is a list of login names separated by spaces.

Table 11–6 Common Mail Commands

Function	Command	Result
Online help	? ?, help	Displays a list of commands with a summary of their meanings.
Stop mail	? q Ctrl/D	Stops mail; all changes made to your mailbox (for example, deletions) are final.
	? x, ex	Stops mail and returns you to the $ prompt. Your mailbox is left as it was at the beginning of the mail session.
Read a message	? Return	Displays the **current** message.
	? p, t	Displays or redisplays the **current** message.
	? n, +	Displays the **next** message.
	? -	Displays the **previous** message.
	? *number*	Displays the selected **numbered** message.
Delete messages	? d *msg_list*	Deletes all messages whose numbers are in msg_list. If no list is given, only the most recently read message is deleted.
	? dp, dt	Deletes the most recently read message and displays the next message.
	? u *msg_list*	Undeletes all messages whose numbers are in msg_list and places them back in the user's mailbox. Only messages from the current mail session can be undeleted.
Reply to messages	? R	Allows you to send a response to the user who sent you the message you just read. The subject field of the message contains the subject of the message you are responding to, preceded by "Re:". Use the tilde escapes just as you would when sending mail.
	? r	Performs same as R, but sends a response to all recipients of the original message as well as the sender.
	? m *user_list*	Sends mail to all users named in the user_list. Can be used with the ~f escape to forward messages. The ~f escape says to read the given messages into the one you are sending.
	? file#	Returns you to your mailbox file.
	? set	Lists mail variables.
Save messages	? RET, s, w	Saves the message in the file mbox in your home directory.
	? pre, ho	Holds (preserves) the message in your system mailbox.
	? co	Holds the message in your mailbox and copies it into mbox.
	? s, w, co *msg_list filename*	Saves the messages in msg_list in the filename instead of in mbox. The filename is required.
List all messages received	? h, f	Displays menu of messages. Deleted messages are not included in the list. Preserved messages are preceded by a P, new messages by an N, and the current message by a >.

(continued on next page)

Table 11–6 (Cont.) Common Mail Commands

Function	Command	Result
	`? top`	Prints the first five lines of each message.
	`? z, z-`	Moves headers one screen forward or one screen backward if the list is too long to display in a single screen.

Learning Activity: Common Mail Commands

Sending Mail

Use the `Mail` command to send yourself the three messages below, in order. Use the words "one", "two", and "three" as the subject lines for the messages. Use the `~s` escape to give the last message a subject instead of using the Subject: prompt.

- This is the first of a series of messages.
- This message is the second of three.
- And this is message number three.

You will read the messages that you sent in the next lab.

Using Common Mail Commands

1. To invoke the Mail utility, issue the `Mail` command. You will see a menu of your mail messages similar to the menu in the lecture material.
2. Get a help list of available commands by entering a question mark (`?`).
3. Enter `1` to read the first message (one).
4. Delete message 1 by entering `d`.
5. Press Return to display the next message. Forward message 2 to yourself by entering `m` followed by your login name and pressing Return. Then enter `~m` Return to copy to yourself the message you just read. Press Ctrl/D to end the message.
6. Press Return to see the third message. Save it to the file `mail.file`.
7. Tell the shell to show the date and time by entering `!date`.
8. Exit Mail using the command `x`. Enter the command `cat mail.file` to see the message you saved.
9. Now issue the `Mail` command again to see what is in your mailbox. You should see four messages, including the one you forwarded from the previous mail session.
10. Enter the number `2` to see the second message. Enter `pre` to preserve it in your mailbox. Now enter - (the minus sign) to see the previous message (1). Delete it by entering `d` .
11. Enter the number `3` to see the third message. Reply to the message by:

    Entering the `r` command

Typing **This is a reply to message 3.** and pressing Return
Pressing Ctrl/D

12. Press Return to read message 4. Delete it using the `d` command.
13. Enter `h` to see the headers of all the remaining messages.
14. Undelete messages 1 and 4 using the command: `u 1 4`. Now enter the `h` command again.
15. Quit Mail with the `q` command.

Learning Activity Solution: Common Mail Commands

Sending Mail

Use the `Mail` command to send yourself the three messages below, in order. Use the words "one", "two", and "three" as the subject lines for the messages. Use the `~s` escape to give the last message a subject instead of using the Subject: prompt.

- This is the first of a series of messages.
- This message is the second of three.
- And this is message number three.

You will read the messages that you sent in the next lab.

```
$ Mail you
Subject:   one
This is the first of a series of messages.
Ctrl/D
$ Mail you
Subject:   two
This message is the second of three.
Ctrl/D
$ Mail you
Subject:    Return
~s three    Return
And this is message number three.
Ctrl/D
$
```

Using Common Mail Commands

1. To invoke the Mail utility, issue the `Mail` command. You will see a menu of your mail messages similar to the menu in the lecture material.

```
$ Mail
Mail $Revision: 4.2.4.2 $  Type ? for help.
"/usr/spool/mail/you": 3 messages 3 new
>N  1 you              Mon Jan  3 11:55  10/292  "one"
 N  2 you              Mon Jan  3 11:56  10/286  "two"
 N  3 you              Mon Jan  3 11:56  10/285  "three"
?
```

2. Get a help list of available commands by entering a question mark (`?`).

```
? ?
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:
   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 ==============================
$
```

3. Enter `1` to read the first message (one).

```
? 1
Message  1:
From you Mon Jan  3 11:55:38 1994
Date: Mon, 3 Jan 1994 11:55:38 -0500
From: User for U and C <you>
To: you
Subject: one

This is the first of a series of messages.

?
```

4. Delete message 1 by entering `d`.

```
? d
?
```

5. Press Return to display the next message. Forward message 2 to yourself by entering `m` followed by your login name and pressing Return. Then enter `~m` Return to copy to yourself the message you just read. Press Ctrl/D to end the message.

```
? Return
Message  2:
From you Mon Jan  3 11:56:21 1994
Date: Mon, 3 Jan 1994 11:56:21 -0500
From: User for U and C <you>
To: you
Subject: two

This message is the second of three.

? m you
Subject: forwarded message
~m
Interpolating: 2
(continue)
Ctrl/D
?
```

6. Press Return to see the third message. Save it to the file mail.file.

```
? Return
Message  3:
From you Mon Jan  3 11:56:56 1994
Date: Mon, 3 Jan 1994 11:56:56 -0500
From: User for U and C <you>
To: you
Subject: three

And this is message number three.

? s mail.file
"mail.file" [New file] 10/285
?
```

7. Tell the shell to show the date and time by entering !date.

```
? !date
Mon Jan  3 12:09:39 EST 1994
!
?
```

8. Exit Mail using the command x. Enter the command cat mail.file to see the message you saved.

```
? x
you have mail in /usr/spool/mail/you
$ cat mail.file
From you Mon Jan  3 11:56:56 1994
Received: by sys1.dec.com; id AA05272; Mon, 3 Jan 1994 11:56:56 -0500
Date: Mon, 3 Jan 1994 11:56:56 -0500
From: User for U and C <you>
Message-Id: <9401031656.AA05272@sys1.dec.com>
To: you
Subject: three
Status: R

And this is message number three.

$
```

9. Now issue the `Mail` command again to see what is in your mailbox. You should see four messages, including the one you forwarded from the previous mail session.

```
$ Mail
Mail $Revision: 4.2.4.2 $  Type ? for help.
"/usr/spool/mail/you": 4 messages 4 new
>N  1 you              Mon Jan  3 11:55  10/292  "one"
 N  2 you              Mon Jan  3 11:56  10/286  "two"
 N  3 you              Mon Jan  3 11:56  10/285  "three"
 N  4 you              Mon Jan  3 12:05  19/559  "forwarded message"
?
```

10. Enter the number `2` to see the second message. Enter `pre` to preserve it in your mailbox. Now enter - (the minus sign) to see the previous message (1). Delete it by entering `d` .

```
? 2
Message  2:
From you Mon Jan  3 11:56:21 1994
Date: Mon, 3 Jan 1994 11:56:21 -0500
From: User for U and C <you>
To: you
Subject: two

This message is the second of three.

? pre
? -
Message  1:
From you Mon Jan  3 11:55:38 1994
Date: Mon, 3 Jan 1994 11:55:38 -0500
From: User for U and C <you>
To: you
Subject: one

This is the first of a series of messages.

? d
?
```

11. Enter the number `3` to see the third message. Reply to the message by:

    Entering the `r` command
    Typing **This is a reply to message 3.** and pressing Return
    Pressing Ctrl/D

```
? 3
Message  3:
From you Mon Jan  3 11:56:56 1994
Date: Mon, 3 Jan 1994 11:56:56 -0500
From: User for U and C <you>
To: you
Subject: three

And this is message number three.

? r
To: you
Subject: Re:  three

This is a reply to message 3.
Ctrl/D
?
```

12. Press Return to read message 4. Delete it using the `d` command.

```
? Return
Message  4:
From you Mon Jan  3 12:05:43 1994
Date: Mon, 3 Jan 1994 12:05:43 -0500
From: User for U and C <you>
To: you
Subject: forwarded message

        From you Mon Jan  3 11:56:21 1994
        Received: by sys1.dec.com; id AA05267; Mon, 3 Jan 1994 11:56:21 -0500
        Date: Mon, 3 Jan 1994 11:56:21 -0500
        From: User for U and C <you>
        Message-Id: <9401031656.AA05267@sys1.dec.com>
        To: you
        Subject: two

        This message is the second of three.

? d
?
```

13. Enter `h` to see the headers of all the remaining messages.

```
? h
>P  2 you              Mon Jan  3 11:56  10/286  "two"
    3 you              Mon Jan  3 11:56  10/285  "three"
?
```

14. Undelete messages 1 and 4 using the command: `u 1 4`. Now enter the `h` command again.

```
? u 1 4
? h
    1 you              Mon Jan  3 11:55  10/292  "one"
 P  2 you              Mon Jan  3 11:56  10/286  "two"
    3 you              Mon Jan  3 11:56  10/285  "three"
>   4 you              Mon Jan  3 12:05  19/559  "forwarded message"
?
```

15. Quit Mail with the `q` command.

```
? q
New mail has arrived.
Saved 3 messages in /usr/users/you/mbox
Held 1 message in /usr/spool/mail/you
?
```

Customizing Mail

Overview

If you read your messages and do not delete or save them in a specified file, they are saved in a file called `/usr/spool/mail/you` when you use `q` to quit the `mail` utility. If your mailbox already exists, the `Mail` utility appends the messages that you have read (but did not delete) to the end of your mailbox file.

You can read your mailbox file by entering:

```
cat /usr/spool/mail/you
```

Once you start using mail you will accumulate many mail messages. You may want to save some of these messages and organize them for easier access. Mail includes a simple facility for organizing groups of messages together in **folders**.

Using Folders

Mail allows you to create as many folders as you need. Each folder of messages is a single file. For your convenience, all your folders are kept in a single directory of your choosing.

Your folder directory must be specified in your `.mailrc` options file with a command like: `set folder=letters`, where `letters` is the directory name. If your folder directory does not begin with a slash (/), mail assumes that your folder directory is a subdirectory of your home directory.

If your home directory is `/usr/users/you`, `set folder=letters` tells mail to find your folder directory in `/usr/users/you/letters`.

Anywhere a file name is expected, you can use a folder name, preceded with a plus sign (+). The table shows various commands for working with folders.

Table 11–7 Working with Folders

To	Enter at the ? Prompt
Select folder `folder_name`	`fo +folder_name`
Select the previous folder	`fo #`
Select the system mailbox	`fo %`
Select your personal mailbox	`fo &`
Determine the current folder and number of messages it contains	`folder`
Determine the names of all folders	`folders`
Copy the current message to folder `folder_name`	`co +folder_name`
Save the current message in folder `folder_name`	`s +folder_name`

Setting Up Your Mail Option File .mailrc

If you use the tilde escapes ~s and ~c often, you can set up your mail account to automatically prompt you for subjects and carbon copies every time you send mail by creating a .mailrc options file.

The .mailrc file must be located in your home directory. The .mailrc file is not present when you log in for the first time; you must create it using an editor or with cat somewhat like the following:

```
$ cd
$ cat > .mailrc
set ask
set askcc
Return
Ctrl/D
$
```

This .mailrc file contains two options:

- ask, which tells mail to prompt you for a subject line whenever you send a message.
- askcc, which prompts you for carbon copies when you finish typing a message.

When sending mail, if you choose not to have a subject or send any carbon copies of a given message, press Return in response to the prompt.

The System Mail File /usr/lib/Mail.rc

When the mail utility is called, before it turns control over to you, it executes the contents of the /usr/lib/Mail.rc file. /usr/lib/Mail.rc is a systemwide file maintained by the system administrator and contains commands applicable to all system users.

You can display the /usr/lib/Mail.rc file by typing the following command:

```
$ cat /usr/lib/Mail.rc
```

If your system responds with a subject prompt after you type mail and a user's name, and you do not have your own personal .mailrc file, the system administrator has included the ask command in the /usr/lib/Mail.rc file.

To create mbox, the hold option must be deleted from the system .Mailrc file.

Customizing Your Mail Option File

In addition to automatically displaying the subject and Cc: in your mail account, you can use the set command in the system Mail.rc file, and in your .mailrc file to further customize mail. The set command was used previously to set the ask and askcc options. There are two types of options that can be set:

- Options that are either on or off (binary). For example, the set command turns the option on: set option

  And, the unset command turns the option off: unset option

- Options that have a value. For example: `set option=value`

To set an option, edit or create your `.mailrc` file to include a line or lines, as shown in the following example. You can specify several options in a single `set` command. The example uses additional options.

Example 11–7 Adding set Commands to Your .mailrc File

```
set crt=20 #invokes a screen at a time read of mail messages
set metoo    #creates copies of sender messages for the sender

set append   #adds mail messages to the end of the mail box file
set ask askcc
```

Aliases

You can also use your `.mailrc` file to define aliases. An alias is a name that stands for one or more real user names. Mail sent to an alias is really sent to the list of real users or a distribution list associated with it.

The `alias` command can be defined for the members of a project so that you can send mail to the whole project by sending mail to just a single name.

For example, suppose that the participants in a project have user names sam, sally, steve, and susan. To define an alias called "project" for them, enter the following command in your `.mailrc` file:

```
alias project sam sally steve susan
```

Including this command in your `.mailrc` file defines `project` as a distribution list that includes sam, sally, steve and susan. To send mail to all users on this distribution list, enter:

```
mail project
```

The `alias` command can also be used to provide a convenient name for someone whose user name is inconvenient. For example, if a user named Bob Anderson had the login name anderson, you might want to use:

```
alias bob anderson
```

This command allows you to send mail to the shorter name, bob.

When you use personal aliases (the aliases listed in your `.mailrc` file) to send mail, those that receive it can reply to the recipients or everyone listed on the distribution list.

Learning Activity: Customizing Mail

Customizing Mail

1. Use any editor to add the following line to your `.profile` file, or to create it if it does not exist:

   ```
   MAIL=/var/spool/mail/$USER
   ```

2. If you have not already done so, print the /usr/share/lib/Mail.rc file to your screen.
3. Use any editor to add the following lines to your .mailrc file, or to create it if it does not exist:
   a. If ask and askcc are not already included in the /usr/share/lib/Mail.rc file, include them in your .mailrc file. Also add any other options you might like.
   b. Put at least one alias into your .mailrc file.
4. Send mail to all the users in one of the aliases in your .mailrc file using the alias.

Using Folders

1. Make a directory in your home directory for your folders using the mkdir command. Call it whatever you like.
2. Add a line to your .mailrc file that will tell mail where to find your folder directory. It should have the form: set folder=myfolders where myfolders is the name you called your folder directory.
3. Enter Mail, and save a mail message you have received to a folder called practice using the save command. Mail will automatically create the folder.
4. Copy a mail message you have received to the folder called practice using the copy command.
5. Leave mail using the quit command and reenter mail using the command:

   ```
   $ Mail -f +practice
   ```

   Read the messages in your practice folder using the regular mail commands.

Learning Activity Solution: Customizing Mail

Customizing Mail

1. Use any editor to add the following line to your .profile file, or to create it if it does not exist:

   ```
   MAIL=/var/spool/mail/$USER
   ```

   No solution required.
2. If you have not already done so, print the /usr/share/lib/Mail.rc file to your screen.

```
$ cat /usr/share/lib/Mail.rc
    .
    .
    .
#
#       Mail.rc 1.3  com/cmd/mailx,3.1,9021 12/21/89 10:11:47
#
set ask noaskcc dot nokeep save
    .
    .
    .
```

3. Use any editor to add the following lines to your `.mailrc` file, or to create it if it does not exist:

   a. If `ask` and `askcc` are not already included in the `/usr/share/lib/Mail.rc` file, include them in your `.mailrc` file. Also add any other options you might like.

   b. Put at least one alias into your `.mailrc` file.

```
$ cat .mailrc
set ask askcc
alias project sam sally steve susan
alias jim zuchchowsen
```

4. Send mail to all the users in one of the aliases in your `.mailrc` file using the alias.

```
$ Mail project
Subj: Enter subject here
Enter message here
Ctrl/D
Cc: Return
```

Using Folders

1. Make a directory in your home directory for your folders using the `mkdir` command. Call it whatever you like.

```
$ mkdir myfolders
```

2. Add a line to your `.mailrc` file that will tell mail where to find your folder directory. It should have the form: `set folder=myfolders` where `myfolders` is the name you called your folder directory.

   No solution required.

3. Enter Mail, and save a mail message you have received to a folder called `practice` using the `save` command. Mail will automatically create the folder.

```
$ Mail
Mail $Revision: 4.2.4.2 $  Type ? for help.
"/usr/spool/mail/you": 2 messages 1 new
    1 you                Mon Jan  3 11:56  11/297  "two"
>N  2 you                Mon Jan  3 12:25  10/286  "Re:  three"
? Return
Message  2:
From you Mon Jan  3 12:25:39 1994
Date: Mon, 3 Jan 1994 12:25:39 -0500
From: User for U and C <you>
To: you
Subject: Re:  three

This is a reply to message 3.

? s +practice
"/usr/users/you/myfolders/practice" [New file] 10/286
?
```

4. Copy a mail message you have received to the folder called `practice` using the `copy` command.

```
$ 1
Message  1:
From you Mon Jan  3 11:56:21 1994
Date: Mon, 3 Jan 1994 11:56:21 -0500
From: User for U and C <you>
To: you
Subject: two

This message is the second of three.

? copy +practice
"/usr/users/you/myfolders/practice" [Appended] 11/297
?
```

5. Leave mail using the `quit` command and reenter mail using the command:

```
$ Mail -f +practice
```

Read the messages in your `practice` folder using the regular mail commands.

No solution required.

Using Mail with DEC OSF/1 Motif

Overview

If you are using DEC OSF/1, you have the choice of standard mail, or the DEC OSF/1 Motif mail facility. DEC OSF/1 Motif mail is an interface to `mh`. `mh` is not covered in this course.

DEC OSF/1 Mail

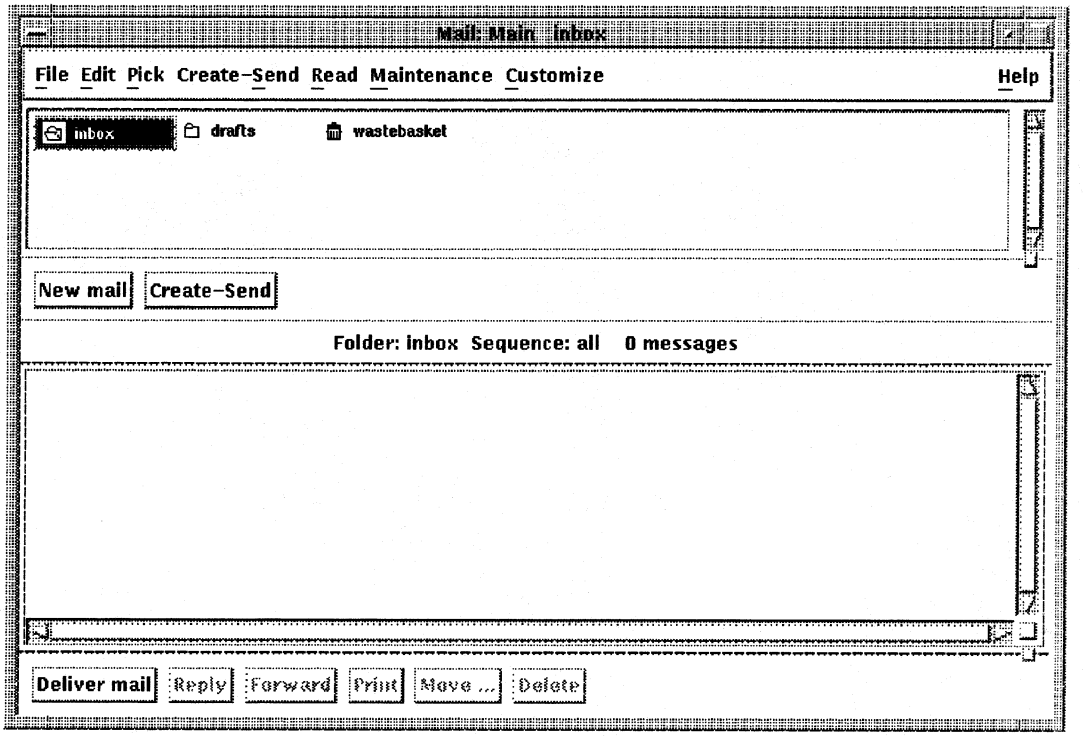
To use DEC OSF/1 mail, either select the Mail option from the Applications menu of your Motif Window Manager window on your screen by clicking on it with your mouse, or enter the following command in a terminal emulation window:

```
/usr/bin/X11/dxmail
```

DEC OSF/1 Mail Window

The figure shows the new window that appears on your screen after starting DEC OSF/1 mail.

Figure 11–1 DEC OSF/1 Mail Window

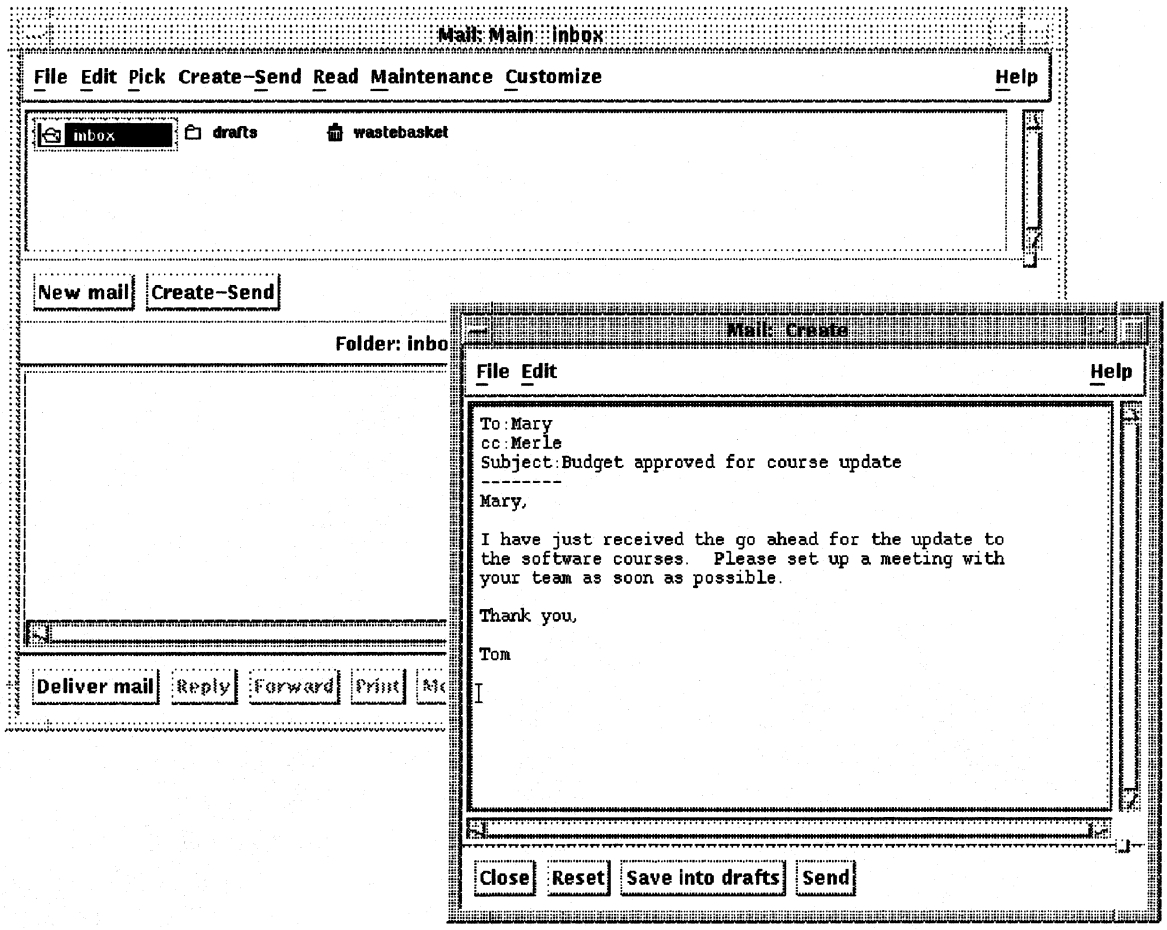


ZKOX-055000058-01-RAGS

Sending and Receiving DEC OSF/1 Mail

After entering DEC OSF/1 mail, you can choose various options by clicking on the option word displayed on your screen. The following figure shows the screen display after Create-Send has been chosen.

Figure 11–2 DEC OSF/1 Mail Create Window



ZKOX-055000058-02-RAGS

The available options are displayed on your screen at all times. Simply click on the word (or box) and you will either be presented with another menu to choose from, or the desired action will be performed.

Learning Activity: Using DEC OSF/1 Motif Mail

This learning activity provides practice using the DEC OSF/1 Motif mail windowing environment. You must have a workstation with DEC OSF/1 Motif windowing.

1. Examine the menu choices for the DEC OSF/1 Motif Mail window. How many types of Help menu choices are there?
2. Customize your Mail Profile so that your window is in outline style.
3. Customize your Mail window so that carbon copies are sent to yourself on every reply to mail.
4. Customize your Mail window to use `mh` profile system defaults.
5. Create a folder through the Maintenance menu.

6. Sort your mail by date after you have several messages in your Mail window.
7. Use Create-Send to send a message in your new folder.
8. Use help to get information on the Pick menu and select some of your mail using the pick search feature.
9. If a printer is available, print out a mail message under the File menu.

Learning Activity Solution: Using DEC OSF/1 Motif Mail

1. Examine the menu choices for the DEC OSF/1 Motif Mail window. How many types of Help menu choices are there?

   No solution required.

2. Customize your Mail Profile so that your window is in outline style.

   Choose "Outline Style" from the Customize and Modify Mail profile menu.

3. Customize your Mail window so that carbon copies are sent to yourself on every reply to mail.

   Choose "Do send carbon copies..." from the Customize and Modify Mail profile menus.

4. Customize your Mail window to use mh profile system defaults.

   Choose "Act according to..." from the Customize and Modify Mail profile menus.

5. Create a folder through the Maintenance menu.

   Select Create folder from the Maintenance menu and type your folder name in the window, then select OK to add the folder.

6. Sort your mail by date after you have several messages in your Mail window.

   Choose Sort by date from the Maintenance menu.

7. Use Create-Send to send a message in your new folder.

   No solution required.

8. Use help to get information on the Pick menu and select some of your mail using the pick search feature.

   Choose On Context from the Help menu, then click the ? pointer on Pick to get context-sensitive help.

   Select Pick from the menu and the menu choice for the search routine that you need.

9. If a printer is available, print out a mail message under the File menu.

   Select Print...from the File menu and see what printer(s) are available on your system, then select one with MB1 and select OK.

The write Utility

Overview

The `write` command allows you to have an online conversation with another user on your system by displaying your comments on his or her terminal screen.

Online Conversations

The following table shows how to have an online conversation.

Enter	Function
`write` *username*	Starts your conversation by beeping the recipient's terminal and displaying a message to inform that a message will be displayed.
The text of your message. It may be any length you choose.	
Ctrl/D	Ends your side of the conversation.

Using the write Command

Suppose, for example, that a group of friends decide to meet for lunch at 11:45. At 11:55, one of them still has not arrived. Since he has a reputation for getting absorbed in his work, the others might decide to send him a message.

Figure 11–3 Using the write Command

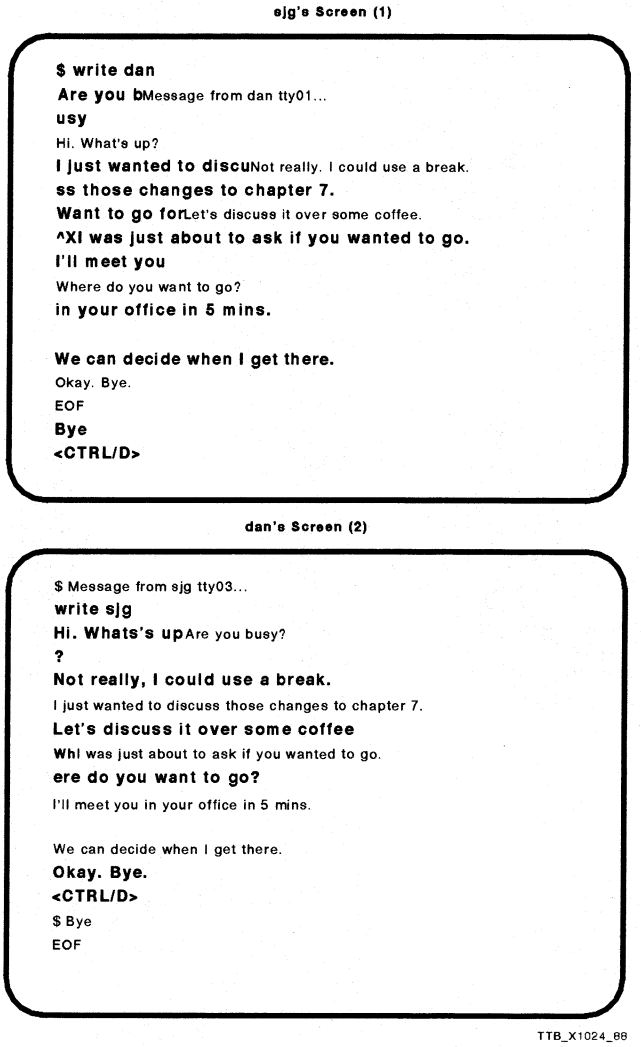


The terminal screen of sjg (one of the impatient friends) on the left and the screen of dan (who is unaware of the time) on the right.

Using the write Utility

Try to avoid overwriting each other's conversation as much as possible. Rely upon your intuition and knowledge of spelling and sentence structure. There is no distinction on the terminal between what you type and what the other person types.

Figure 11–4 A Two-Way Conversation Using the write Command



TTB_X1024_88

Using the write Utility for Two-Way Conversation

The `write` command is convenient for one-way communication that requires no response. When you use `write`, you usually want to have a two-way conversation.

To use `write` for two-way communication, the recipient of the message must write back. In Figure 11–4, Screen 1 belongs to sjg and Screen 2 to dan. As you study the conversation, there are three points you should notice:

- Each user's input to his or her own terminal is displayed in boldfaced text. When you press Return, all text on the line you just typed is immediately transmitted to the other's terminal

in the middle of whatever text the other person is typing or reading.

- Because both users are typing at the same time, some redundancy is likely. On the eighth line of sjg's terminal, you will see the character ^X, which is the kill character on sjg's terminal. By pressing Ctrl/X, sjg "killed" the portion of the conversation that said "Want to go for."
- When you press Ctrl/D to end your part of the conversation, the characters EOF appear on the recipient's screen. However, when one person ends his or her side of the conversation, the other person is still "writing" until they, too, press Ctrl/D.

To avoid the confusion of the conversation in Figure 11–4, users should establish a protocol (standard procedure) for their conversations, in which each user waits to respond until the other person signals that his or her message is complete.

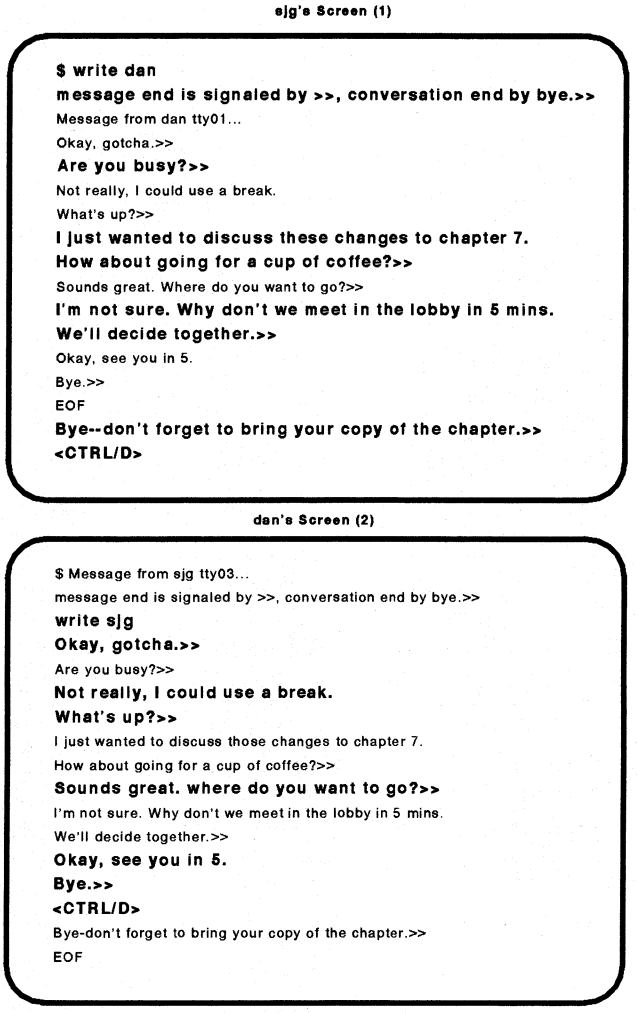
One such protocol is to follow each message with the character **o** (for over) and to end the entire conversation with the characters **oo** (for over and out). You may also establish your own protocols at the beginning of the conversation, as seen in Figure 11–5. Compare the conversation in Figure 11–5 with the one in Figure 11–4. You will see that Figure 11–5 is much easier to follow.

When you issue the `write` command, the system searches through its list of current users to find the person you are writing to. If the name is not found in the list, you receive the error message telling you that the user is not logged in.

Note

The system does not check to make sure that the specified user is actually a valid user name. Therefore, if you type a user name incorrectly and actually write to a name with no account, you are told that the person is not logged in, not that there is no such account.

Figure 11–5 Two-Way Conversation Using the write Command with Protocol



Locking Out Messages

You enhance your ability to communicate by sending and receiving messages. However, receiving messages can affect your concentration because they disturb the image on your screen.

Some commands and processes lock out other users so that the person running them cannot receive messages. You can also prevent people from sending you messages. If you try to write to a person who has locked you out, you will receive the message `Permission denied`.

To lock out messages, that is, to keep people from disturbing you with interrupting messages, enter `mesg n` into your `.profile` file. To receive messages, enter `mesg y`.

If you are not sure whether or not other users can send you messages, enter `mesg` without an argument. The response is `yes` if you are receiving messages, and `no` if you are not receiving

messages. Next time you log in, `mesg` will automatically be reset to `yes` if you do not set it with the `.profile` file.

Learning Activity: The write Utility

This activity explores the features of the `write` utility of your DEC OSF/1 system. You must have another user on the system that you can communicate with to complete the questions.

1. Use the `write` command to hold a conversation with another system user who is currently logged in to your system.
2. At the beginning of your conversation, establish some conventions that were suggested in the lecture:

   End your statements with **o**
   End your conversation with **oo**

Learning Activity Solution: The write Utility

1. Use the `write` command to hold a conversation with another system user who is currently logged in to your system.

   Type `$ write username` **to initiate the conversation from your terminal.**

2. At the beginning of your conversation, establish some conventions that were suggested in the lecture:

   End your statements with **o**
   End your conversation with **oo**

   To respond, type `$ write username` **to the username sending the message.**

   Then type into the terminal whatever conversation protocol you can agree on with your other party.

Summary

Obtaining User Information

Table 11–8 User Information Commands

Command	Function
`who`	Lists the users who are logged in by login name, terminal name, and login date and time
`who am i`	Lists information only about the user who is logged in at the terminal where the command is entered
`whoami`	Displays your current login name from the `/etc/passwd` file
`users`	Provides a quick one-line list of users
`finger`	Provides detailed information about users

Introduction to Mail

The DEC OSF/1 operating system allows a user to send mail to other users through two commands:

- `Mail` username
- `mailx` username

To send mail to another user:

System Prompt	Enter
$	`Mail` username, and press Return (where username is the desired recipient)
Subject:	The subject of your message, and press Return; or press Return if you choose to leave the subject blank.
[no prompt]	The body of your message, then press Return to move the cursor to the beginning of the next line. Press Ctrl/D to end the message.
Cc:	Usernames to whom carbon copies/additional copies should be sent. For several users, separate names with spaces or commas. Cc: may not appear on your system. Press Return when the list is complete to send the mail.
$	Indicates you have successfully sent your mail message.

Customizing Mail

You can customize your mail utility by editing the `.mailrc` file in your home directory. The `.mailrc` file can be edited to include `set options`, such as `ask` and `askcc` to automatically display Subject and Cc: as well as `aliases` to set up distribution lists.

Mail messages can be organized in folders. Any mail command used with a file can be used with a folder.

Using Mail with DEC OSF/1 Motif

The `mh` mail handler provides a Motif user interface to perform mail functions.

The write Utility

The `write` utility holds an on-line conversation with another user.

12

Recalling and Editing Korn Shell Commands

Recalling and Editing Korn Shell Commands

Introduction

The Korn shell allows you to store, view, and reexecute previously executed shell commands. Each shell command executed can be stored in a command history buffer. You can edit commands either on the command line or in a temporary buffer.

The Korn shell has three built-in command-line editors: `vi`, `emacs`, and `gmacs`. These command-line editors are subsets of the `vi` and `GNU emacs` editors. Both the `emacs` and `gmacs` built-in command-line editors are a subset of the `GNU emacs` editor. The only difference between the two is the way they handle Ctrl/T, which is covered later.

Additionally, the `vi` or `GNU emacs` editors can be used to edit a command in a temporary buffer. When editing a command in a temporary buffer, the normal `vi` or `emacs` editor is being used, not the Korn shell built-in command-line editor.

Learning activities are provided in this chapter to allow you to practice viewing, recalling for reexecution, and editing previously executed commands.

Objectives

To efficiently execute Korn shell commands, you should be able to:

- Define the use of variables associated with Korn shell command recall
- Recall commands from the command history file using the Korn shell
- Define the use of Korn shell variables for editing commands on the command line
- Edit Korn shell commands on the command line using the `emacs`, `gmacs`, or `vi` built-in command-line editors
- Edit Korn shell commands in a temporary buffer using the `emacs` or `vi` editor

Resources

- *OSF/1 User's Guide*
- DEC OSF/1 `man` pages

Korn Shell Command History

Overview

The command history file stores the commands you enter and allows you to display them. You can select a previous command for reexecution, or you can modify a previously executed command. This Korn shell feature saves you time because it allows you to reuse commands instead of retyping them.

Korn Shell Command History

The Korn shell, by default, retains the last 128 executed commands in a history file. The user does not have to take any action but may modify the number of recallable commands and the name of the history file. The history file, (default is `.sh_history`), is maintained when the user logs out. This file is not automatically purged or deleted, and could become very large. Therefore, the user should occasionally delete the history file.

If you are running multiple sessions on the same system, consider creating multiple history files, one for each session running the Korn shell. If you do not create separate history files, the commands you execute in each session will be archived in the one history file, `$HOME/.sh_history`. Some users like this feature, but others find it confusing.

Korn Shell Command History Variables

Table 12–1 shows the variables you can modify to alter the number of commands you can recall or the name of the history file.

Table 12–1 Korn Shell History Buffer and Command Recall Variables

Variable	Description
HISTSIZE=`n`	`n` determines the number of Korn shell commands that can be recalled. The default is 128. Set this variable in the `.profile` file.
HISTFILE=`filename`	`filename` is the pathname of the file that stores the command history. The default is `$HOME/.sh_history`. Set this variable in the `.profile` file if you want to change it.

Example 12–1 provides commands to place in the `.profile`. This sets the number of recallable commands to 96, an arbitrary number, and sets the **HISTFILE** variable to provide a separate history file for each session running the Korn shell.

Example 12–1 Modifying the Korn Shell History Variables

```
HISTSIZE=96
HISTFILE=$HOME/.history$$
export HISTFILE
```

Korn Shell Command Viewing and Recall

The Korn shell allows you to view the latest **HISTSIZE** commands, recall and reexecute a command, or recall, modify, and reexecute a command.

To display the command history buffer, use the Korn shell built-in command `fc` (**fix c**ommand). The `fc` command has several formats. The first format is used to view previously executed Korn shell commands.

The first format of the `fc` command is:

`fc [-lnr] first last`

The command options are:

- The `-l` option causes the command lines from **first** to **last** to be displayed with command numbers. If a line number (first) or range of lines (first, last) is not specified, the last 16 commands are displayed. The Korn shell has a preset alias, `history=fc -l`.
- The `-n` option, used with the `l` option, causes commands to be displayed without command numbers, for example, `fc -ln` first last.
- The `-r` option, used with the `l` option, causes commands to be displayed in reverse order, for example, `fc -lr` first last.
- The first and last options can be specified as command numbers or as command names.
  - If specified as numbers, commands are displayed in the same order as the numbers; the `-r` option reverses that order.
  - If specified as names, commands are displayed from the most recent first command to the most recent last command, in first to last order; the `-r` option reverses that order.

Viewing Korn Shell Commands

Example 12–2 uses the `fc` command to view previously executed Korn shell commands from the history file.

Note

The prompt in the example has been changed to display command numbers. This can be done with `PS1="! $ "`.

Example 12–2 Viewing Korn Shell Commands

```
41 $ fc -l 32 36  ❶
32 ls -al
33 more tempfile
34 cd /bin
35 ls -al k
36 echo $myname
42 $ fc -ln 32 36  ❷
   ls -al
   more tempfile
   cd /bin
   ls -al k
   echo $myname
43 $ fc -lr 32 36  ❸
36 echo $myname
35 ls -al k
34 cd /bin
33 more tempfile
32 ls -al
44 $ fc -l ls cd  ❹
35 ls -al k
34 cd /bin
45 $ history ls cd  ❺
35 ls -al k
34 cd /bin
46 $
```

❶ Displays command lines 32 to 36

❷ Displays command lines 32 to 36 without line numbers

❸ Displays command lines 32 to 36 in reverse order

❹ Displays the command lines from the latest `ls` to the previous `cd` command

❺ Displays the command lines from the latest `ls` to the previous `cd` command, but this time using `history`, the Korn shell built-in alias for `fc -l`

Learning Activity: Korn Shell Command History

1. Korn shell command history; command history variables:
   a. By default, the Korn shell retains the last _____ commands in the history file. The default history file is: _____.
   b. Why must this file be occasionally deleted?
      _____
   c. What two variables must you set to control the number of commands you can access in the history buffer and the name of the history file? Do not set either of these variables at this time; some of the following exercise steps would not work properly.
      _____ and _____
2. Korn shell command history setup:
   a. Edit your `.profile` and add `PS1='! ksh> '` if it has not been done. Also, execute that command on the command line. It gives you the command number.
   b. Execute the following commands to set up a known sequence for the following steps. Some commands are chosen at random just to add something to the history buffer.

```
$ more .profile
$ date
$ set
$ stty -a
$ name="your name"                provide your name
$ address="some address"          provide your address here
$ echo $name
$ who
$ finger
```

3. Using the Korn shell built-in function `fc` (**fix** command) with the format:

   `fc [-e editor] [-lnr] first last`

   Execute each of the following commands and answer any questions.
   a. Enter: **`fc -1`** How many commands are displayed?
      _____
   b. Enter: **`history`** How does this display compare with the `fc -l` display? Why?
      _____
      _____

      From the series of commands executed, pick one of the commands as `n` for the following steps.
   c. Enter: **`fc -l`** *`n n+4`* Display five commands from the history file.

d. Enter: `fc -lr n n+4` Display the same five commands, but in reverse order.

e. Enter: `fc -ln n n+4` Again, display the same five commands, but this time without the command numbers.

f. Enter: `fc set echo` Display the commands between the latest `set` and `echo` commands.

Learning Activity Solution: Korn Shell Command History

1. Korn shell command history; command history variables:

   a. By default, the Korn shell retains the last **128** commands in the history file. The default history file is `$HOME/.sh_history`.

   b. Why must this file be occasionally deleted?

   The file could become very large because it is not automatically purged.

   c. What variables would you set to control the number of commands you can access in the history buffer and the name of the history file? Do not set either of these variables at this time; some of the following exercise steps would not work properly.

   HISTSIZE and HISTFILE

2. Korn shell command history setup:

   a. Edit your `.profile` and add `PS1='! ksh> '` if it has not been done. Also, execute that command on the command line. It gives you the command number.

   b. Execute the following commands to set up a known sequence for the following steps. Some commands are chosen at random just to add something to the history buffer.

```
$ more .profile
$ date
$ set
$ stty -a
$ name="your name"                  provide your name
$ address="some address"            provide your address here
$ echo $name
$ who
$ finger
```

No solution is provided for this part of the exercise.

3. Using the Korn shell built-in function `fc` (**fix c**ommand) with the format:

   `fc [-e editor] [-lnr] first last`

   execute each of the following commands and answer any questions.

   a. Enter: `fc -l` How many commands are displayed?

   16, or less if you have entered fewer commands

b. Enter: `history` How does this display compare with the `fc -l` display? Why?

They both display the previous 16 commands. history is the default alias for `fc -l`.

From the series of commands executed, pick one command as `n` for the following steps.

c. Enter: **`fc -l`** *n n+4* Display five commands from the history file.

d. Enter: **`fc -lr`** *n n+4* Display the same five commands, but in reverse order.

e. Enter: **`fc -ln`** *n n+4* Again, display the same five commands, but this time without the command numbers.

f. Enter: **`fc set echo`** Display the commands between the latest `set` and `echo` commands.

No solution is provided for this part of the exercise.

Recalling Previously Executed Korn Shell Commands

Previously executed Korn shell commands can be recalled and reexecuted or can be modified and reexecuted using the `fc` command.

The second format for the `fc` command is:

```
fc -e - [old=new] [command]
```

This command reexecutes a previously executed command. It replaces **old** information with **new** information in the command specified by `command`, where `command` can be a command number or command name. If the command referred to is the last command executed, `command` can be omitted.

Use `fc -e -` to reexecute the previous command without modification. The default alias for `fc -e -` is `r`. To execute the previous command, type `r`.

Example 12–3 provides a short history of commands.

Example 12–3 history Command Output

```
60 $ history 49 53
49 $ date
50 $ first="Charlie"
51 $ mi="M."
52 $ last="Brown"
53 $ print $first $mi $last
61 $
```

Example 12–4 provides some samples of recalling and modifying commands.

Example 12–4 Recalling Commands from the History File

```
61 $ fc -e - Charlie=Susan 50  ❶
first="Susan"
62 $ fc -e - M=L mi    ❷
mi="L"
63 $ r Brown=Smith last  ❸
last="Smith"
64 $ r print     ❹
print $first $mi $last
Susan L. Smith
65 $
```

❶ Recalls command 50 and substitutes "Susan" for "Charlie."

❷ Recalls the last command that started with "mi" and substitutes `L` for `M`.

❸ Recalls the last command, which started with **last** and substitutes Smith for Brown. Uses the default alias, `r`.

❹ Reexecutes the last print command without modifications.

Using the Korn Shell History File

To practice using the Korn shell history file, execute the following commands. The command numbers provided here are relative and are to be used as reference throughout this section. Command response is not shown.

```
  n $ date
n+1 $ whatis emacs
n+2 $ whereis emacs
n+3 $ who
n+4 $ word1="Hello"
n+5 $ word2="World"
n+6 $ print $word1 $word2
```

1. `fc -l`: Display the last 16 commands from the history file. Note the command numbers for commands `n` to `n+6` above.
2. `history`: Same as above, but using the Korn shell, preset alias for `fc -l`.
3. `fc -ln n n+6`: Display commands `n` through `n+6` without command numbers.
4. `fc -lr n n+6`: Display commands `n` through `n+6` in reverse order.
5. `fc -e - emacs=vi n+1`: Recall, modify, and reexecute the latest `whatis` command using the command number. Replace `emacs` with `vi`.
6. `fc -e - emacs=vi whereis`: Recall, modify, and reexecute the latest `whereis` command using the command name.
7. `r n+6`: Use the default alias for `fc -e -` to recall command `n+6`.

8. **r emacs=ksh n+1**: Use r to recall, modify, and reexecute command n+1. Replace emacs with ksh.
9. **r emacs=ksh whereis**: Use r to recall, modify, and reexecute the latest whereis command, using the command name. If the sequence was followed properly, there should be an error. The latest whereis command does not contain emacs; it contains vi. The error is:

   ksh: fc: bad substitution

   Try: **r vi=ksh whereis.**

Learning Activity: Korn Shell Command Editing

Using the Korn shell built-in fc function with the format:

fc -e -[old=new] [string]

execute the following commands and answer any questions.

a. Enter: **history** (Note the command numbers for future reference.)

b. Enter: **fc -e - name=address echo** Repeat the last echo $name command, substituting address for name.

c. Enter: **fc -e - name=USER n** Repeat the last echo $name command, substituting USER for name. Use command number n instead of the command name.

d. Enter: **fc -e - USER=SHELL** Repeat the previous command, substituting SHELL for USER.

   Because it is the previous command, you do not have to provide the command name or number.

e. Enter: **r SHELL=HISTFILE** Repeat the previous command, substituting HISTFILE for SHELL.

   Why does the r command do the same thing as the fc -e - command?

   _____

f. Enter: **fc -e -.profile=.kshrc** Repeat the last more command, substituting .kshrc for .profile. Notice that the . (dot) in the file name cannot be separated from the hyphen (-) as it can be with normal variables.

g. Try some of these commands again using command numbers instead of command names.

Learning Activity Solution: Korn Shell Command History

Using the Korn shell built-in fc function with the format:

fc -e -[old=new] [string]

execute the following commands and answer any questions.

a. Enter: **history** (Note the command numbers for future reference.)

b. Enter: **fc -e - name=address echo** Repeat the last `echo $name` command, substituting `address` for `name`.

c. Enter: **fc -e - name=USER n** Repeat the last `echo $name` command, substituting `USER` for `name`. Use command number `n` instead of the command name.

d. Enter: **fc -e - USER=SHELL** Repeat the previous command, substituting `SHELL` for `USER`.

   Because it is the previous command, you do not have to provide the command name or number.

e. Enter: **r SHELL=HISTFILE** Repeat the previous command, substituting `HISTFILE` for `SHELL`.

   Why does the `r` command do the same thing as the `fc -e -` command?

   Because `r` is a built-in alias for `fc -e -`

f. Enter: **fc -e -.profile=.kshrc** Repeat the last `more` command, substituting `.kshrc` for `.profile`. Notice that the **.** (dot) in the file name cannot be separated from the hyphen (-) as it can be with normal variables.

g. Try some of these commands again using command numbers instead of command names.

Korn Shell Command-Line Editing

Overview

The Korn shell has the ability to edit commands on the command line. The command-line editor also provides additional ways to recall commands and peruse the commands in the history file.

This topic discusses how to select a Korn shell command-line editor and how to recall commands using the selected command-line editors. As you execute other commands throughout the course, you will use the selected editor to edit or correct the commands.

Korn Shell Command-Line Editor Selection

The Korn shell built-in command-line editors are `vi`, `emacs`, and `gmacs`. All three editors are a subset of the `vi` and `GNU emacs` editors. They are a part of the Korn shell and are therefore available even if the corresponding editor is not present on the system.

Use one of the following commands to select a Korn shell command-line editor. Commands are listed in order of priority, lowest to highest.

- `set -o emacs` (`gmacs` or `vi`)
- **EDITOR**=`emacs` (`gmacs` or `vi`)
- **VISUAL**=`emacs` (`gmacs` or `vi`)

If you executed the following commands:

```
$ set -o vi
$ EDITOR=gmacs
$ VISUAL=emacs
```

the **VISUAL** variable would override the other two commands and `emacs` would be the command-line editor used.

You only need to use one method of selecting your command-line editor and can place that command in your `.profile` file.

To determine the state of the `emacs`, `gmacs`, or `vi` command-line editors, use `set -o` with no options, `print $EDITOR`, and `print $VISUAL` commands.

If the `emacs` command-line editor is turned on, turn it off by entering `set +o emacs`, or turn on another command-line editor. For example, if `emacs` is turned on, `set -o vi` turns on the `vi` command-line editor and turns off the `emacs` command-line editor. You can also define either of the variables **EDITOR** or **VISUAL**, because they have higher priority.

Editing Korn Shell Commands on the Command Line

The Korn shell provides the capability to edit commands on the command line with the built-in editors gmacs, emacs, and vi. They are a subset of the GNU emacs and vi editors, and most of the editing commands are the same as their editor counterparts.

The commands for command-line editing are provided in Appendix B. The commands are separated into tables by function as follows:

- Table B–1, Fetching Commands from the History File
- Table B–2, Command Execution
- Table B–3, Moving the Cursor
- Table B–4, Deleting or Replacing Characters, Words, and Lines
- Table B–5, Entering Insert Mode
- Table B–6, Using Uppercase and Lowercase Letters
- Table B–7, Pathname Expansion or Completion
- Table B–8, Miscellaneous Commands

Editing Korn Shell Commands with emacs or gmacs

Both emacs and gmacs are subsets of the GNU emacs editor. The only difference between emacs and gmacs is the way they handle Ctrl/T. The emacs editor swaps the present character with the next character and advances the cursor, whereas gmacs just swaps the previous two characters.

Using the emacs Korn Shell Command-Line Editor

To practice using the emacs Korn shell command-line editor, make sure that neither the **EDITOR** nor the **VISUAL** variable is assigned. To unset the assigned variable(s):

```
$ print $EDITOR          $ unset EDITOR
$ print $VISUAL          $ unset VISUAL
```

Enable emacs as the command-line editor as follows:

```
$ set -o emacs
```

Execute the following commands to demonstrate how to to fetch commands from the history file.

- history: Display the last 16 commands from the history file.
- Ctrl/P: Fetch commands from the history file. After fetching a command, execute it by pressing Return.
- Ctrl/P, Ctrl/N: Use both to move about in the history file to inspect previous commands.
- Choose a command from the history file, and then use:
  - Ctrl/R`string`: Recall the most recent command that contains `string`.
  - Ctrl/R`^string`: Recall the most recent command with `string` as the beginning of the line (usually the command name).

Type the following sequence on the command line to demonstrate cursor motion and similar commands on the command line. It contains a group of words that will not execute. Execute the sequence of commands.

```
this is a very long command that will not execute
```

- Use each of the cursor motion commands:
  - Ctrl/A: Move the cursor to the beginning of the line.
  - Ctrl/F: Move the cursor forward (right) by characters to the a.
  - Esc F: Move the cursor forward by words to the space between command and that.
  - Esc B: Move the cursor backward (left) by words to the v in very.
  - Ctrl/B: Move the cursor backward by characters to the beginning of the line.
  - Ctrl/]v: Find the first v.
  - Ctrl/A: Move the cursor backward to the beginning of the line.
- Cut part of the command and place it at the end of the command line:
  - Ctrl/Spacebar: Set the mark at the beginning of the line.
  - Ctrl/] y: Move the cursor to the y at the end of very.
  - Ctrl/F: Move the cursor to the space between very and long.
  - Ctrl/X Ctrl/X: Verify that the mark is where you want it to be.
  - Ctrl/W: Delete the marked text.
  - Ctrl/E: Move the cursor to the end of the line.
  - Spacebar: Add a space.
  - Ctrl/Y: Place the deleted text at the end of the command line.
- Use deletion commands to remove text:
  - Esc <x: Delete two words at the end of the command line.
  - Ctrl/A: Move the cursor to the beginning of the line.
  - Esc D: Delete the first two words from the command line.
  - Ctrl/E: Move the cursor to the end of the command line.
  - Ctrl/Y: Place the last deleted word back into the command.
- Type characters on the command line until the command line is longer than can be displayed at one time:
  - Note that a "less than" symbol (<) indicates that text on the left end of the command does not fit on the screen.

  - Ctrl/A: Move the cursor to the beginning of the command. Note that a > is now displayed to indicate that text on the right end of the command does not fit on the screen.
- Ctrl/U: Delete the entire command without executing it.

Editing Korn Shell Commands with vi

Like the `vi` editor, the `vi` command-line editor contains both **insert** and **command** modes.

- **Insert mode**: The default mode; type text normally on the command line. To recall a previous command, you must press Esc to enter command mode. Insert mode is used to type in a command or to insert characters into a recalled command. Any character typed is inserted on the command line, except those listed in Table 12–2.

Table 12–2 vi Insert Mode Commands

Command	Function
Esc (escape)	Places the `vi` command-line editor into command mode
<X (erase)	Deletes the character previous to the current cursor position
Ctrl/U (kill)	Clears all input characters from the command line
Ctrl/W	Deletes the previous word
Ctrl/D (eof)	Executes the command if typed with the cursor at the first character on the command line
Ctrl/V	Inserts the next character as a special character (control characters except `erase` and `kill`)
\	Inserts the control characters for erase or kill; press \ followed by the delete or kill character

- **Command mode**: Invoked by pressing Esc (F11 on many keyboards, or use Ctrl/[ ). Use command mode to move through previously executed commands, position the cursor within a command, or delete characters or words in a command.

Appendix B, Korn Shell Command-Line Editing Commands for emacs and vi contains tables of commands to use with `emacs` (`gmacs`) or `vi` command-line editing. The `vi` commands are for command mode only.

Using the vi Korn Shell Command-Line Editor

Practice what you have learned by enabling `vi` as the command-line editor with:

```
$ set -o vi
```

- `history`: Display the last 16 commands from the history file.
- Esc k: Fetch commands from the history file. After fetching a command, execute it with Return. Remember, use Esc to enter command mode.
- Esc k, j: Use both to move about in the history file to inspect previous commands.
- Pick a command from the history file, then use:
  - Esc /`string`: Recall the most recent command that contains `string`. Execute the command with Return.
  - Esc /^`string`: Recall the most recent command with `string` at the beginning of the line (usually the command name). Execute the command with Return.

Again, type the following sequence on the command line to demonstrate cursor motion and similar commands on the command line. It contains a group of words that will not execute.

Execute the sequence of commands. When using the `vi` command-line editor, the default is insert mode (you just start typing a command).

```
this is a very long command that will not execute
```

- Use each of the cursor motion commands:
  - 0: Move the cursor to the beginning of the line.
  - l: Move the cursor forward (right) by characters to the `a`.
  - w: Move the cursor forward by words to the beginning of `that`.
  - b: Move the cursor backward (left) by words to the `v` in `very`.
  - h: Move the cursor backward by characters to the beginning of the line.
  - fv: Find the first `v`.
  - 0: Move the cursor back to the beginning of the line.
- Delete the first three words from the command and place them at the end of the command line:
  - 3dw: Delete three words.
  - $: Move the cursor to the end of the line.
  - A: Enter insert mode. Add a space at the end of the line by pressing Spacebar.
  - Esc Enter command mode.
  - p: Place deleted text back into command line.

- Use deletion commands to remove text:
  - [2db]: Delete two words at the end of the command line.
  - [0]: Move the cursor to the beginning of the line.
  - [2dw]: Delete the first two words from the command line.
  - [S]: Move the cursor to the end of the command line.
  - [p]: Place the last deleted word back into the command.
- [i]: Enter insert mode and type characters on the command line until the command line is too long to be displayed at one time.
- [Esc]: Enter command mode:
  - Note that a "greater than" symbol (<) indicates that text on the left end of the command does not fit on the screen.
  - [0]: Move the cursor to the beginning of the command. Note that a > is now displayed to indicate that there is text on the right end of the command that does not fit on the screen.
- [i]: Enter insert mode.
- [Ctrl/U]: Delete the entire command without executing it. The `vi` command-line editor remains in insert mode allowing you to insert text.

Learning Activity: Editing Korn Shell Commands on the Command Line

This exercise requires you to use a simple interactive Korn shell script to provide a multiple-line command. You do not need to understand the format of the script to complete the exercise.

1. Korn Shell Command-Line Editor Selection

   This section requires you to select the different command-line editors and verify that the desired command-line editor has been selected.

   First, turn off any command-line editor currently turned on. This ensures that the following steps work properly.

   a. `$ set -o`: Execute the command (without options) to display all options that are turned on. Note the state of the `emacs`, `gmacs`, and `vi` options.

      Which command turns off an option?

      _____

      Turn off any `emacs`, `gmacs`, or `vi` options that are turned on.

   b. Execute the following two commands:

```
$ print $EDITOR
$ print $VISUAL
```

Check to see if a value has been assigned to the **EDITOR** or **VISUAL** variables. If either variable has been assigned a value, it must be cleared.

Which command clears a Korn shell variable? Clear either of those two variables if set.

_____

Note

Although the latter part of this lab is structured, the next section allows you to practice the different `emacs` or `gmacs` and `vi` command-line editing commands. It is not necessary to try both `emacs` and `gmacs` as they differ only in reaction to the Ctrl/T key.

If you skip any of the exercise, be sure to complete the section on editing multiple-line commands.

_____

c. List the three ways to turn on the command-line editor in order of priority, lowest to highest.

_____

_____

_____

2. Fetching Commands from the History Buffer

Using the `emacs` option, turn on the `emacs` command-line editor.

a. Using the commands listed in the following table, move through the commands in the history buffer. Try each command. Other commands are found in Appendix B. Do not edit commands yet; that topic is discussed later.

Function	Command
Recalls previous command or [count] commands back	Esc [count][1] Ctrl/P
Recalls next command or [count] commands ahead	Esc [count][1] Ctrl/N
Searches backward through the history buffer to find the first occurrence of a command line containing `string`	Ctrl/R `string`
Searches backward through the history buffer to find the first occurrence of a command line containing `string` at the beginning of the command line	Ctrl/R ^ `string`

[1]Use Esc [count] only to provide a count. If the F11 key does not work as Esc on your terminal, use Ctrl/ [.

b. After practicing the different emacs commands, turn off the emacs command-line editor and turn on the vi command-line editor. Now, using the commands from the following table, move about in the history buffer using vi. Other commands are found in Appendix B. Again, do attempt command-line editing at this time.

Note

To use the commands, the vi command-line editor must be in command mode, and is reached by pressing Esc.

Function	Command
Recalls previous command or [count] commands back	[count] k or [count] -
Recalls next command or [count] commands ahead	[count] j or [count] +
Searches backward through the history buffer to find the first occurrence of a command line containing string	/ string
Searches backward through the history buffer to find the first occurrence of a command line containing string at the beginning of the command line	/ ^ string

3. Editing Long Korn Shell Command Lines

You can type a Korn shell command line of 256 characters, which is wider than your terminal screen. The Korn shell automatically scrolls your command horizontally if the line is wider than the screen window. If the Korn shell scrolls the line, a symbol is displayed in the last column to indicate that there is more text. The following notation is used to indicate that there is more text to the:

Right: >
Left: <
Right and **Left:** *

The Korn shell also sends the bell character when you attempt something that is not allowed, such as a failing search, or pressing Esc with vi in command mode.

a. Editing Long Korn Shell Commands with emacs

Turn off the vi command-line editor, and turn on the emacs command-line editor.

Type a long line of text on the command line, then practice the emacs command-line editing commands using Appendix B. Remember, the emacs command-line editor is a subset of the emacs editor.

b. Editing Long Korn Shell Commands with vi

Turn off the emacs command-line editor and turn on the vi command-line editor.

Type a long line of text on the command line, then practice the vi command-line editing commands in Appendix B. Remember, the vi command-line editor must be in command mode to move the cursor.

4. Editing Multiple-Line Commands

You generally create a script file to execute Korn shell scripts. However, you may occasionally want to execute a short Korn shell script on the command line.

If you then want to repeat a command consisting of more than one command line, it is handled differently by emacs and vi.

First, we will assign some values, then use an if statement as our multiple-line command. It may be repeated, and you can change the comparison number, or anything else, providing you do not modify the essential parts of the command.

When the Korn shell recognizes that the command is not complete, and requires more information, it generates the secondary prompt from variable **PS2**, which by default is the >. The script is executed when Return is pressed after the fi, ending the if statement.

Execute the following commands. Be sure to provide the proper spaces, as shown for the if statement. Notice that there are only three commands. More information on Korn shell scripts is provided later in the course, so do not spend time on the script itself now.

```
$ name="your name"
$ age="your age"
$ if (( $age >= 40 ))
> then print $name "is over the hill!"
> else print $name "isn't that old after all!"
> fi
```

a. Select emacs as your command-line editor.

Note

Before proceeding further, make sure that Ctrl/O does not interfere. By default, discard is defined as Ctrl/O (stty -a). A Ctrl/O turns off the output and does not execute the command on the command line. To prevent this, you must undefine discard. Execute: `$ stty discard 'undef'`

- Use Ctrl/P to recall the if command. When the command is recalled, the shell script occupies the same number of command lines as the original command. You may edit only the portion of the command that is on the command line.

- Make some modifications to the command. You can readily change the age used for comparison or the contents of the print commands.
- When you have completed modifications on a line, use Ctrl/O to execute that line, and go to the next command line.
- When you get to the last command line, press Return. If you use Ctrl/O, and another command follows, the present command line is executed, and the following command is placed on the command line.

b. Select `vi` as your command-line editor.

- For the case of the `vi` command-line editor, when the command is recalled, the shell script is placed on one line, with Ctrl/J separating the different lines. Use the `vi` command-line editor to move through the command and make modifications. Pressing Return from any location on the command-line causes reexecution of the command.
- If you make modifications that cause the entire command to become longer than 256 characters, you will not be able to enter any more characters.
- Use k (after getting to command mode with an Esc), to recall the `if` command.
- Repeat the command, making some modifications to the command. Reexecute the command with Return. Retry a few times, adding text between the quotes to see how many characters you can fit on the line.

Learning Activity Solution: Editing Korn Shell Commands on the Command Line

This exercise requires you to use a simple interactive Korn shell script to provide a multiple-line command. You do not need to understand the format of the script to complete the exercise.

1. Korn Shell Command-Line Editor Selection

   This section requires you to select the different command-line editors and verify that the desired command-line editor has been selected.

   First, turn off any command-line editor currently turned on. This ensures that the following steps work properly.

   a. `$ set -o`: Execute the command (without options) to display all options that are turned on. Note the state of the `emacs`, `gmacs`, and `vi` options.

      Which command turns off an option?
      `set +o [option]`

      Turn off any `emacs`, `gmacs`, or `vi` options that are turned on.

b. Execute the following two commands:

```
$ print $EDITOR
$ print $VISUAL
```

Check to see if a value has been assigned to the **EDITOR** or **VISUAL** variables. If either variable has been assigned a value, it must be cleared.

Which command clears a Korn shell variable? Clear either of those two variables if set.

```
unset variable_name
```

Note

Although the latter part of this lab is structured, the next section allows you to practice the different `emacs` or `gmacs` and `vi` command-line editing commands. It is not necessary to try both `emacs` and `gmacs` as they differ only in reaction to the Ctrl/T key.

If you skip any of the exercise, be sure to complete the section on editing multiple-line commands.

c. List the three ways to turn on the command-line editor in order of priority, lowest to highest.

```
set -o emacs
EDITOR=emacs
VISUAL=emacs
```

2. Fetching Commands from the History Buffer

Using the `emacs` option, turn on the `emacs` command-line editor.

```
$ set -o emacs
```

a. Using the commands listed in the following table, move through the commands in the history buffer. Try each command. Other commands are found in Appendix B. Do not edit commands yet; that topic is discussed later.

Function	Command
Recalls previous command or [count] commands back	Esc [count][1] Ctrl/P
Recalls next command or [count] commands ahead	Esc [count][1] Ctrl/N
Searches backward through the history buffer to find the first occurrence of a command line containing `string`	Ctrl/R `string`

[1]Use Esc [count] only to provide a count. If the F11 key does not work as Esc on your terminal, use Ctrl/ [.

Function	Command
Searches backward through the history buffer to find the first occurrence of a command line containing `string` at the beginning of the command line	Ctrl/R ^ `string`

b. After practicing the different `emacs` commands, turn off the `emacs` command-line editor and turn on the `vi` command-line editor. Now, using the commands from the following table, move about in the history buffer using `vi`. Other commands are found in Appendix B. Again, do attempt command-line editing at this time.

Note

To use the commands, the `vi` command-line editor must be in command mode, and is reached by pressing Esc.

Function	Command
Recalls previous command or [count] commands back	[count] k or [count] -
Recalls next command or [count] commands ahead	[count] j or [count] +
Searches backward through the history buffer to find the first occurrence of a command line containing `string`	/ `string`
Searches backward through the history buffer to find the first occurrence of a command line containing `string` at the beginning of the command line	/ ^ `string`

3. Editing Long Korn Shell Command Lines

You can type a Korn shell command line of 256 characters, which is wider than your terminal screen. The Korn shell automatically scrolls your command horizontally if the line is wider than the screen window. If the Korn shell scrolls the line, a symbol is displayed in the last column to indicate that there is more text. The following notation is used to indicate that there is more text to the:

Right: >
Left: <
Right and **Left**: *

The Korn shell also sends the bell character when you attempt something that is not allowed, such as a failing search, or pressing Esc with `vi` in command mode.

a. Editing Long Korn Shell Commands with `emacs`

Turn off the vi command-line editor, and turn on the emacs command-line editor.

Type a long line of text on the command line, then practice the emacs command-line editing commands using Appendix B. Remember, the emacs command-line editor is a subset of the emacs editor.

No solution required.

b. Editing Long Korn Shell Commands with vi

Turn off the emacs command-line editor and turn on the vi command-line editor.

Type a long line of text on the command line, then practice the vi command-line editing commands in Appendix B. Remember, the vi command-line editor must be in command mode to move the cursor.

No solution required.

4. Editing Multiple-Line Commands

You generally create a script file to execute Korn shell scripts. However, you may occasionally want to execute a short Korn shell script on the command line.

If you then want to repeat a command consisting of more than one command line, it is handled differently by emacs and vi.

First, we will assign some values, then use an if statement as our multiple-line command. It may be repeated, and you can change the comparison number, or anything else, providing you do not modify the essential parts of the command.

When the Korn shell recognizes that the command is not complete, and requires more information, it generates the secondary prompt from variable **PS2**, which by default is the >. The script is executed when Return is pressed after the fi, ending the if statement.

Execute the following commands. Be sure to provide the proper spaces, as shown for the if statement. Notice that there are only three commands. More information on Korn shell scripts is provided later in the course, so do not spend time on the script itself now.

```
$ name="your name"
$ age="your age"
$ if (( $age >= 40 ))
> then print $name "is over the hill!"
> else print $name "isn't that old after all!"
> fi
```

a. Select emacs as your command-line editor.

Note

Before proceeding further, make sure that Ctrl/O does not interfere. By default, discard is defined as Ctrl/O (`stty -a`). A Ctrl/O turns off the output and does not execute the command on the command line. To prevent this, you must undefine `discard`. Execute: `$ stty discard 'undef'`

- Use Ctrl/P to recall the `if` command. When the command is recalled, the shell script occupies the same number of command lines as the original command. You may edit only the portion of the command that is on the command line.
- Make some modifications to the command. You can readily change the age used for comparison or the contents of the print commands.
- When you have completed modifications on a line, use Ctrl/O to execute that line, and go to the next command line.
- When you get to the last command line, press Return. If you use Ctrl/O, and another command follows, the present command line is executed, and the following command is placed on the command line.

No solution required.

b. Select `vi` as your command-line editor.

- For the case of the `vi` command-line editor, when the command is recalled, the shell script is placed on one line, with Ctrl/J separating the different lines. Use the `vi` command-line editor to move through the command and make modifications. Pressing Return from any location on the command-line causes reexecution of the command.
- If you make modifications that cause the entire command to become longer than 256 characters, you will not be able to enter any more characters.
- Use k (after getting to command mode with an Esc), to recall the `if` command.
- Repeat the command, making some modifications to the command. Reexecute the command with Return. Retry a few times, adding text between the quotes to see how many characters you can fit on the line.

No solution required.

Editing Korn Shell Commands in a Temporary Buffer

Overview

With the Korn shell, you can edit one command or multiple commands in a temporary buffer using the `emacs` or `vi` editor, if they are installed on your system.

Reading Commands into a Buffer

The format of this version of the `fc` command is:

```
fc [-e editor] [-r] first last
```

where:

- The editor supplied to the `-e` option must be `emacs` or `vi`. If the `-e` option is not specified, the editor defined by the **FCEDIT** variable is used. The `ed` editor is used if the variable is not defined.
- The `-r` option reverses the order of the commands in the buffer.
- `first` and `last` are the names or numbers of the first and last commands in the sequence to be edited. If `last` is not used, only one command is edited.
  - If specified as numbers, commands are displayed in the same order as the numbers; the `-r` option reverses that order.
  - If specified as names, commands are displayed from the most recent `first` command to the most recent `last` command, in `first` to `last` order; the `-r` option reverses that order.

Exiting the Editor

The selected commands are placed in a temporary buffer, where they can be edited with normal commands for the selected editor.

After making necessary changes, writing the temporary file causes command execution.

If `emacs` is used, when exiting the editor with Ctrl/X Ctrl/C, `emacs` asks the question: `Save file /tmp/sh####.##? (y or n)`

If you answer no, the temporary file is not written and the original (unmodified) command(s) are executed. If you answer yes, the temporary file is written and the new (modified) command(s) are executed.

If `vi` is used to execute the commands, exit the editor with `ZZ` or `:wq`, either of which causes the temporary file to be written and the new commands to be executed. If the editor is exited with `:q!`, the original commands are executed.

In all situations, whether the old or new command(s) are executed, all commands are echoed on the screen, and then they are executed.

Multiple commands that were edited as a group now appear as one command in the history buffer whether or not they were modified.

Example 12–5 shows how the `fc` command edits commands in a temporary buffer.

Example 12–5 Editing Commands in a Temporary Buffer

```
$ fc -e vi 100 104      ❶
$ fc -e emacs 100 104     ❷
$ FCEDIT=emacs       ❸
$ fc 100 104
```

❶ Using `vi` to edit commands 100 through 104 in a temporary buffer

❷ Using `emacs` to edit commands 100 through 104 in a temporary buffer

❸ Assigning `emacs` as the default editor for `fc` with the **FCEDIT** variable, then editing commands 100 through 104 in a temporary buffer

Learning Activity: Korn Shell Command Editing

Use the Korn shell built-in command `fc`, with the format:

```
fc -e editor first last
```

to edit Korn shell commands in a temporary buffer.

The editor to be used is supplied on the command line, or is defined by the **FCEDIT** variable. Do not confuse editing a command in a temporary buffer, which uses the normal `vi` or `emacs` editors, with editing a command on the command line, which uses a subset of the `vi` or `emacs` editors. Also, **FCEDIT** is the only variable used with the `fc` built-in function.

1. Execute this sequence of commands, then the following steps. Note the command numbers denoted by `n`. You will need them for the next few steps.

```
  n $ more .profile
n+1 $ name="your name"
n+2 $ address="your address"
n+3 $ echo $name
n+4 $ fc -l
```

2. Edit command number `n` through `n+3` in a temporary buffer using the `vi` editor.

   ```
   $ fc -e vi n n+3
   ```

   Using the `vi` editor, change:

   `.profile` to `.login`
   The contents of the **name** variable
   The contents of the **address** variable
   $name in the `echo` command to **$address**

   Execute the commands with `:wq` or `ZZ`.

Which commands are executed if you quit `vi` with `:q!` Why? Try it!

3. Assign the **FCEDIT** variable the value `emacs`. The default editor for editing commands in a temporary buffer is now `emacs`. Check the value of **FCEDIT**.

   Now, invoke `fc` without the `-e` option, but with the same command numbers as before. This invokes the `emacs` as the editor. Make the same changes that you did with `vi`, and exit with Ctrl/X Ctrl/C.

4. Repeat the previous step one more time. This time, when `emacs` asks:
   "`Save file /tmp/sh####.##?`",
   reply with *n*.

   Do the modified or unmodified commands get executed? Why?

Learning Activity Solution: Korn Shell Command Editing

Use the Korn shell built-in command `fc`, with the format:

```
fc -e editor first last
```

to edit Korn shell commands in a temporary buffer.

The editor to be used is supplied on the command line, or is defined by the **FCEDIT** variable. Do not confuse editing a command in a temporary buffer, which uses the normal `vi` or `emacs` editors, with editing a command on the command line, which uses a subset of the `vi` or `emacs` editors. Also, **FCEDIT** is the only variable used with the `fc` built-in function.

1. Execute this sequence of commands, then the following steps. Note the command numbers denoted by `n`. You will need them for the next few steps.

```
  n $ more .profile
n+1 $ name="your name"
n+2 $ address="your address"
n+3 $ echo $name
n+4 $ fc -l
```

2. Edit command number `n` through `n+3` in a temporary buffer using the `vi` editor.

   `$ fc -e vi n n+3`

   Using the `vi` editor, change:

   `.profile` to `.login`
   The contents of the **name** variable
   The contents of the **address** variable
   $name in the `echo` command to **$address**

Execute the commands with `:wq` or `ZZ`.

Which commands are executed if you quit `vi` with `:q!` Why? Try it!

The original commands are executed because the temporary file is not written.

3. Assign the **FCEDIT** variable the value `emacs`. The default editor for editing commands in a temporary buffer is now `emacs`. Check the value of **FCEDIT**.

```
$ FCEDIT=emacs
$ print $FCEDIT
emacs
```

Now, invoke `fc` without the `-e` option, but with the same command numbers as before. This invokes the `emacs` as the editor. Make the same changes that you did with `vi`, and exit with Ctrl/X Ctrl/C.

```
$ fc n n+3
```

4. Repeat the previous step one more time. This time, when `emacs` asks:
`"Save file /tmp/sh####.##?"`,
reply with *n*.

Do the modified or unmodified commands get executed? Why?

Unmodified, the modified commands have to be written to the `/tmp` directory to be executed

Summary

Korn Shell Command History

- The Korn shell provides a command history buffer that can be used to recall and reexecute previous commands
- The Korn shell has a default history file, `$HOME/.sh_history`
- Shell variables used with the history files are:
  - **HISTSIZE**
  - **HISTFILE**

Korn Shell Command-Line Editing

- The Korn shell provides the capability to edit commands on the command line with `vi`, `emacs`, or `gmacs`.
- The command-line editors are a part of the Korn shell, and are a subset of the `GNU emacs` and `vi` editors.
- The `vi` command-line editor has two modes: insert and command. Insert mode is the default mode. Command mode is entered by pressing Esc.
- Most command-line editor commands are the same as the editor of which it is a subset.

Editing Korn Shell Commands in a Temporary Buffer

- Use the command `fc -e editor first last` to edit a list of previous commands in a temporary buffer.
- Use the **FCEDIT** variable to select a default editor for editing commands in a temporary buffer.

13

Controlling Processes

Korn Shell Process Control

Introduction

A program (or command) currently executing is called a **process** in the UNIX environment. This chapter discusses the concept of a process and how to manage your user processes. When understood, the concepts presented here will allow you to accomplish complex tasks quickly, efficiently and with fewer errors.

Each shell has its own commands to control processes. This chapter discusses the Korn shell commands. Commands for executing and controlling processes are presented with examples throughout the chapter.

Objectives

To control processes, you should be able to:

- Define the concepts of process and job
- Submit a sequence of commands to the shell
- Group shell commands
- Use the `jobs` and `ps` commands to check job and process status
- Move jobs between the foreground and background
- Set the priority of a process
- Stop an executing process
- Use the `at` and `batch` commands to schedule a job
- Use the `crontab` command to schedule jobs to run periodically

Resources

- *OSF/1 User's Guide*
- DEC OSF/1 `man` pages

The Process Concept

Overview

The operating system has many resources (both hardware and software) that it must manage. One of the software resources is a program currently in execution. A program in execution is called a **process**.

Processes

A process is an object controlled by the operating system and can be created or destroyed. A process has resources allocated to it as well as an environment associated with it.

This environment:

- Is inherited from the parent process
- Consists of all information relative to the process
- Can be changed by issuing shell commands

A process can communicate with and create other processes.

Process Environment

All the information necessary to execute and control a process is called the **process environment**.

The process environment includes such information as:

- Process and process group IDs
- Open files
- Working directory
- File creation mask
- Real and effective user and group IDs
- Resource limits
  - Maximum file size
  - Maximum amount of memory
- Signal action settings
- A set of named variables

Process and Process Group IDs

Each process has a unique number associated with it called the process identification number (PID). This number distinguishes the process from any other active process. It also has a process group ID which places the process into a process group. There is a foreground process group and a background process group. Only one process group can be associated with your terminal at a time; this is the foreground process group. All other processes are in the background process group.

Open Files

The system keeps track of the files opened by each process. Otherwise, things would not work well, especially pipes, and I/O redirection. There are three files open upon process creation by default:

- Standard input: file descriptor 0 is open for reading.
- Standard output: file descriptor 1 is open for writing.
- Standard error: file descriptor 2 is open for reading and writing.

Working Directory

Each process has a working directory where files are located for input and where files are created on output. When you log in, the working directory is set to your home directory. After you log in, change the working directory using the `cd` command and display the working directory with the `pwd` command. The current working directory is inherited by any child processes.

File Creation Mask

The system uses the file creation mask to create a file and to enable and disable permissions for accessing the file. It is a 3-digit octal number that represents the permissions to be disabled when a file is created.

Real and Effective User and Group IDs

Each process has a real **user** ID and one or more real **group** IDs. These are set when you log in and are inherited by all processes you create. The effective user ID and effective group ID are set to the real user ID and real group ID when you log in and determine what permissions the process has to read, write and/or execute files. When you run a program that has `setuid` and or `setig` permissions, the effective process user and group IDs may be changed by the program.

Resource Limits

The system puts limits on the maximum size of files and memory associated with a process.

Signal Action Settings

The system and other processes may send a signal to your process. A **signal** is a message represented by a number with a symbolic name. When you press certain keys on your terminal, they send a message to your process indicating some action is to be taken. For example, pressing Ctrl/Z while a process is executing causes the process to be stopped. A process must either ignore these signals or must specify which specific action to take, otherwise the process terminates. Some such signals are: QUIT, HUP, TERM and KILL.

This signal mechanism also allows for a process to communicate with other processes.

Named Shell Variables

The shell maintains a set of variables that contain information about the environment. Use the `set` command to display these variables and their values. Some variables have an attribute of export which means the variables can be inherited by a child process. You can also create your own variables, which can be inherited by a child process.

Programs have access to, and use, the values stored in the variables. These programs determine the home directory, where to look for data files, executable files, or other information they need during execution. Running programs may temporarily change, but do not permanently change, the value of the variables.

Some common variables are: **LOGNAME**, **USER**, **HOME**, and **PATH**. Variables are assigned values by issuing **name=value**, where **name** is the name of the variable and **value** is the string value.

`INCLUDE=/usr/include` or

`DIRECTORY=/usr/bin`

Creating a Process

When the operating system is started up, a process called `init` is started. This process is responsible for creating login processes that wait for input from terminals.

When a user logs in to the system, the `init` process starts up a user shell process with a standard environment. From this point on, the user creates other processes by issuing commands and running programs and shell scripts. Each process created inherits the environment of the initiating process.

Grouping Shell Commands

So far, we have discussed how to execute one command (or process) at a time in an interactive mode. You can execute multiple commands more quickly when you group them together on one command line.

You can submit multiple commands at the same time by using:

- The semicolon to separate the commands on the command line
- Parentheses to surround commands that are to execute together

The following example shows how to group commands together using the semicolon.

Example 13–1 Grouping Commands with the Semicolon

```
$ pwd; date; cd /etc; ls -al | grep ^d; pwd
/usr/users/you
Tue Jan 10 15:15:32 EST 1992
drwxr-xr-x   6 root    system   2048 May 29 08:46 .
drwxr-xr-x  12 root    system   1024 Mar 06 16:26 ..
drwxr-xr-x   2 root    system   1024 Mar 24 10:25 lib_admin.dir
drwxr-xr-x   4 root    system    512 Mar 24 08:39 namedb
drwxr-xr-x   2 root    system    512 Mar 24 10:31 netman
drwxr-xr-x   3 root    system    512 Mar 24 10:28 nls
/etc
$
```

The following example shows how to group commands together using parentheses.

Example 13–2 Grouping Commands with Parentheses

```
$ pwd; date; (cd /etc; ls -al | grep ^d); pwd
/usr/users/you
Tue Jan 10 15:20:57 EST 1992
drwxr-xr-x   6 root    system   2048 May 29 08:46 .
drwxr-xr-x  12 root    system   1024 Mar 06 16:26 ..
drwxr-xr-x   2 root    system   1024 Mar 24 10:25 lib_admin.dir
drwxr-xr-x   4 root    system   512 Mar 24 08:39 namedb
drwxr-xr-x   2 root    system   512 Mar 24 10:31 netman
drwxr-xr-x   3 root    system   512 Mar 24 10:28 nls
/usr/users/you
$
```

Running Jobs in the Background

When you execute a shell command or shell script, the command or script, by default, runs in the foreground. When that command finishes execution, you can execute another command.

When a job is running in the foreground, the shell cannot accept any further input until the job is finished. In other words, you cannot type other commands to the shell while you wait for the job to finish.

To allow more than one job to run at a time, jobs can be run in the background. A background job does not have direct control of terminal input and output. Processing the job does not tie up your communication with the shell.

This method of running jobs is useful when the job:

- Is time consuming
- Does not require interactive input

Such jobs include the following:

- Sorting
- Compilations
- Spreadsheet calculations

- Complex mathematical calculations

To process a job in the background, append an ampersand (&) to the command. The command format is:

```
command &
```

In Example 13–3, two commands are executed in the background.

Example 13–3 Running Jobs in the Background

```
$ ls -al /usr | grep ^d > user-directories &
$ find / -name emacs -print > find-emacs &
$
```

Learning Activity: Grouping Shell Commands

Grouping Shell Commands

1. Submit the `who` and `date` commands to the shell. First, submit each command separately. Then, try separating the commands with a semicolon and submitting them on a single command line.
2. Use the backslash to continue a command line. Substitute your login name for `you` in the command below to find your entry in the `/etc/passwd` file. This file is used by the shell to store encrypted passwords and other accounting information. Be sure to leave a space before the backslash.

   ```
   $ grep "you" \
     /etc/passwd
   ```

3. Try separating the command sequence below with pipe symbols (|) instead of semicolons. Notice that only the output from the last command is printed. Resubmit the command line using semicolons. Now all of the desired output appears.

   ```
   $ ls; date; who
   ```

4. Substitute your login name for `you` in the command below and try using a semicolon instead of the pipe symbol. Your terminal will hang because `grep` expects to receive `stdin` from the terminal if there is no pipe. Press Ctrl/C (interrupt) or Ctrl/D (end of text) to receive the prompt so that you can issue more commands, and type the command as it appears below. There is a big difference between using pipes and semicolons!

   ```
   $ who | grep "you"
   ```

Command Grouping with Parentheses

1. Change the working directory to `/bin` and list its contents with this command:

   ```
   $ cd /bin ; ls
   ```

2. Repeat the above operation with a subshell:
   a. Issue the pwd command to verify that you are located in the /bin directory.
   b. Return to your home directory with the cd command.
   c. Enter the command: `$ ( cd /bin ; ls ) .`
   d. Use pwd to verify that you are still in your home directory.
3. Redirect shell output with the command:

```
$ grep 'mount' /etc/* > outfile 2> diagfile
```

   Inspect the contents of the files outfile and diagfile.

Learning Activity Solution: Grouping Shell Commands

Grouping Shell Commands

1. Submit the who and date commands to the shell. First, submit each command separately. Then, try separating the commands with a semicolon and submitting them on a single command line.

```
$ who
mcfloyd     ttyp0        Feb 13 07:09
evant       ttyp1        Feb 13 15:42
$ date
Thu Feb 13 15:44:01 EST 1992
$ who; date
mcfloyd     ttyp0        Feb 13 07:09
evant       ttyp1        Feb 13 15:42
Thu Feb 13 15:46:14 EST 1992
$
```

2. Use the backslash to continue a command line. Substitute your login name for you in the command below to find your entry in the /etc/passwd file. This file is used by the shell to store encrypted passwords and other accounting information. Be sure to leave a space before the backslash.

```
$ grep "you" \
  /etc/passwd
```

```
$ grep jones \
> /etc/passwd
jones:nsY/w4wpfU8bs:9:15:malcolm l. jones:/usr/users/jones:/bin/ksh
```

3. Try separating the command sequence below with pipe symbols (|) instead of semicolons. Notice that only the output from the last command is printed. Resubmit the command line using semicolons. Now all of the desired output appears.

```
$ ls; date; who

$ ls  | date | who
mcflower    ttyp0        Feb 14 10:11
evant       ttyp1        Feb 14 11:25
$ ls; date; who
calendar      file_cleanup  func2.o       funcs.so      stdio.h
calendar.txt  find.txt      func3.c       set           sub_mod.c
cleanup.cmd   func1.c       func3.o       sleep.cmd     sub_mod.o
file.txt      func2.c       funcs.c       so_locations  sys.cron
Fri Feb 14 11:47:21 EST 1992
mcflower    ttyp0        Feb 14 10:11
evant       ttyp1        Feb 14 11:25
```

4. Substitute your login name for `you` in the command below and try using a semicolon instead of the pipe symbol. Your terminal will hang because `grep` expects to receive `stdin` from the terminal if there is no pipe. Press Ctrl/C (interrupt) or Ctrl/D (end of text) to receive the prompt so that you can issue more commands, and type the command as it appears below. There is a big difference between using pipes and semicolons!

```
$ who | grep "you"

$ who; grep jones
mcflower    ttyp0        Feb 14 10:11
jones       ttyp1        Feb 14 11:25
Ctrl/D
$ who | grep jones
jones       ttyp1        Feb 14 11:25
```

Command Grouping with Parentheses

1. Change the working directory to `/bin` and list its contents with this command:

```
$ cd /bin ; ls

$ cd /bin; ls
Mail        dxfc         mig             strip
Rsh         dxmail       mkcatdefs       strip2.20
.
.
.
```

2. Repeat the above operation with a subshell:
   a. Issue the `pwd` command to verify that you are located in the `/bin` directory.
   b. Return to your home directory with the `cd` command.
   c. Enter the command: `$ ( cd /bin ; ls )`.
   d. Use `pwd` to verify that you are still in your home directory.

```
$ pwd
/bin
$ cd
$ pwd
/usr/users/jones
$ (cd /bin; ls)
Mail        dxfc         mig          strip
Rsh         dxmail       mkcatdefs    strip2.20
.
.
.
$ pwd
/usr/users/jones
```

3. Redirect shell output with the command:

```
$ grep 'mount' /etc/* > outfile 2> diagfile
```

Inspect the contents of the files `outfile` and `diagfile`.

```
$ grep 'mount' /etc/* >outfile 2> diagfile
$ cat outfile
/etc/disktab:# has the same amount for all disks (for the most part). The
'b' partition is
/etc/rpc:mountd         100005  mount showmount
.
.
.
$ cat diagfile
grep: can't open /etc/nsr.res
grep: can't open /etc/phones
$
```

Obtaining Process Status

Overview

To determine how many processes are active, you can get a quick report with the jobs command, or a longer report using the ps command.

Getting a List of Stopped or Background Jobs

The jobs command displays a list of stopped or background jobs.

The format for the jobs command is:

```
jobs [-lp]
```

When used without any options, the jobs command displays the list of background or stopped jobs with job number and the job status. With the -l option, jobs also shows the PID. With the -p option, jobs shows only the PID.

The example shows how to use the jobs command.

Example 13–4 Using the jobs Command

```
$ jobs -l
[4] + 10652  Running      find / -name vi -print > find-vi &
[3] - 10651  Running      find / -name emacs -print > find-emacs &
[2]   10553  Stopped      (ls -alR /usr | grep ^d) > usr-dirs &
[1]   10147  Stopped      vi notice.txt
$
```

Note that the plus (+) indicates the current job, and the minus [-] indicates the previous job.

Obtaining Process Status

Use the ps (process status) command to obtain more information about your processes, or about other processes on the system. The ps command displays information on your login process as well as any other processes that are active under your login process and on the system.

Note

The ps command probably provides more information than the average user desires or needs, but you should be able to determine which processes are your processes, and determine the PID.

The ps Command

The format for the DEC OSF/1 ps command is:

```
ps [-adejflm] [-oO specifier][=header],...  [-t tlist] [-p plist]
    [-s slist] [-u ulist] [-g glist] [-s slist]
```

With no options on the command line, the ps command provides process status with the information categories shown in the table.

Table 13–1 Process Status Information Categories

Header	Description
PID	Process identification number.
TT	Name of the terminal device that controls the process. In most cases, this is the terminal you were using when you gave the command to be processed.
STAT	State of the process indicated by a sequence of up to four letters.
TIME	Accumulated execution time of the process.
COMMAND	Name of the command and its arguments.

ps Command Options

The ps command has options that can be used to modify the amount and format of the output. Some commonly used options are listed in the table.

Table 13–2 Selected ps Command Options

Option	Function
-a	Prints information to standard output about all processes except the process group leaders and processes not associated with a terminal
-e	Prints information to standard output about all processes except kernel processes
-f	Provides additional information including: user, **PPID**, percentage of CPU usage, and the time the process started
-j	Provides job control information including: user, **PID**, **PPID**, process group ID, and session ID
-l	Provides additional information including: process flags, **UID**, **PPID**, percentage of CPU usage, priority, process scheduling increment, size of the process, and the classification of event on which the process is waiting
-t tlist	Displays information about processes with the terminals listed in tlist

(continued on next page)

Table 13–2 (Cont.) Selected ps Command Options

Option	Function
`-u ulist`	Displays information about processes with the user IDs specified in `ulist`

Note

Do not use the `-f` and `-l` options at the same time because the output usually follows the `-l` format. They control only the format of the output.

The `-f` option is more useful to the average user while the `-l` is more appropriate for the system manager because it contains the **UID**.

Using the ps Command

The following examples show how to use the `ps` command with and without options.

In this example, the only processes showing up are console processes, and they are:

- The window manager
- A special console setup program
- The calendar process

Example 13–5 Using the ps Command with No Options

```
$ ps
  PID TT  STAT      TIME COMMAND
 1085 co  IW     0:03.81 /usr/bin/mwm
 1086 co  IW     0:00.33 sh -c /usr/lib/X11/xconsole -geometry =+64+137
 1088 co  IW     0:07.75 /usr/bin/dxcalendar
$
```

The following example has additional processes showing up, with all processes attached to the terminal. The additional processes are:

- Another console X window startup file (`/usr/bin/xdm`)
- The startup for the other window
- The commands to display the two DECterm windows. The `-ls` option causes the creation of login shells
- An `ls -alR` command is being executed

Example 13–6 Using the ps Command with the -a Option

```
$ ps -a
  PID TT  STAT      TIME COMMAND
  231 co  IW +   0:01.39 /usr/bin/xdm -nodaemon -config /usr/lib/X11/xdm/xdm-co
 1085 co  S      0:03.81 /usr/bin/mwm
 1086 co  IW     0:00.33 sh -c /usr/lib/X11/xconsole -geometry =+64+137
 1087 co  IW     0:01.85 /usr/lib/X11/xconsole -geometry =+64+137
 1088 co  S      0:07.75 /usr/bin/dxcalendar
 1089 co  S      0:15.22 /usr/bin/dxterm -ls
 1090 co  IW     0:02.96 /usr/bin/dxterm -ls
 1132 p2  R N    0:03.17 ls -alR /usr
$
```

Example 13–7 uses the `ps` command with the `-ef` options. This display provides all processes and includes the user name. Note that:

- The Korn shell, PID 1092, running in DECterm window with PID 1089 has the following jobs running:
  - PID 1113: editing the file `locate.ksh` with `vi`
  - PID 1120: a `find` command looking for `vi`
  - PID 1121: another `find` command looking for `emacs`
  - PID 1131: `grep` for files that are directories; this job is the parent of PID 1132, the `ls -alR` command
  - PID 1133: the `ps` command that provided the output

Example 13–7 Using the ps Command with the -ef Options

```
$ ps -ef
USER       PID  PPID %CPU STARTED  TT       TIME COMMAND
root         0     0  5.0   Dec 31 ??  1607:22.55 [kernel idle]
root         1     0  0.0   Dec 31 ??    0:01.57 /sbin/init -a
root         2     0  1.0   Dec 31 ??    0:10.43 [vnode_pager]
root         3     0  0.0   Dec 31 ??    0:00.07 [device pager]
root         4     0  0.0   Dec 31 ??    0:00.00 [device server]
root         5     0  0.0   Dec 31 ??    0:00.01 [exception hdlr]
   .
   .
   .
you       1082   438  0.0 14:19:07 co    0:03.25 /usr/bin/dxsession
you       1085  1082  0.0 14:19:13 co    0:03.81 /usr/bin/mwm
you       1086  1082  0.0 14:19:16 co    0:00.33 sh -c /usr/lib/X11/xconsole -g
root      1087  1086  0.0 14:19:16 co    0:01.85 /usr/lib/X11/xconsole -geometr
you       1088  1082  0.0 14:19:19 co    0:07.75 0:07.75 /usr/bin/dxcalendar
root      1089  1082  3.0 14:19:19 co    0:15.22 /usr/bin/dxterm -ls
root      1090  1082  0.0 14:19:19 co    0:02.96 /usr/bin/dxterm -ls
you       1091  1090  0.0 14:20:02 p1    0:00.50 - (ksh)
you       1092  1089  0.0 14:20:02 p2    0:02.37 - (ksh)
you       1113  1092  0.0 14:24:44 p2    0:00.34 vi locate.ksh
you       1120  1092 16.0 14:27:13 p2    0:29.44 find / -name vi -print
you       1121  1092 19.0 14:27:20 p2    0:28.02 find / -name emacs -print
you       1131  1092  2.0 14:29:09 p2    0:00.64 grep ^d
you       1132  1131 27.0 14:29:09 p2    0:03.17 ls -alR /usr
root      1133  1092 10.0 14:29:18 p2    0:00.73 ps -ef
$
```

The `-u` *user_name* option provides only those processes for the specified user. A sample is shown in Example 13–8.

Example 13–8 Using the ps Command with the -fu Options

```
$ ps -fu you
USER       PID  PPID %CPU STARTED  TT       TIME COMMAND
you       1082   438  0.0 14:19:07 co    0:03.25 /usr/bin/dxsession
you       1085  1082  0.0 14:19:13 co    0:03.81 /usr/bin/mwm
you       1086  1082  0.0 14:19:16 co    0:00.33 sh -c /usr/lib/X11/xconsole -g
you       1088  1082  0.0 14:19:19 co    0:07.75 /usr/bin/dxcalendar
you       1091  1090  0.0 14:20:02 p1    0:00.50 - (ksh)
you       1092  1089  0.0 14:20:02 p2    0:02.37 - (ksh)
you       1113  1092  0.0 14:24:44 p2    0:00.34 vi locate.ksh
you       1120  1092 16.0 14:27:13 p2    0:29.44 find / -name vi -print
you       1121  1092 19.0 14:27:20 p2    0:28.02 find / -name emacs -print
you       1131  1092  2.0 14:29:09 p2    0:00.64 grep ^d
you       1132  1131 27.0 14:29:09 p2    0:03.17 ls -alR /usr
$
```

Learning Activity: Obtaining Process Status

1. Issue the ps command without options.
   What is the process ID of your shell process?
2. Enter the command:

   ```
   $ ps -l
   ```

   What additional process information is available with this form of the process command?

Learning Activity Solution: Obtaining Process Status

1. Issue the ps command without options.
   What is the process ID of your shell process?

   ```
   $ ps
     PID TT  S         TIME COMMAND
   20919 p1  S      0:01.23 -ksh (ksh)
   $
   ```

   In this example, the process ID of the shell process is 20919.
2. Enter the command:

   ```
   $ ps -l
   ```

   What additional process information is available with this form of the process command?

```
$ ps -l
      F S     UID   PID  PPID %CPU PRI  NI  RSS WCHAN    TT         TIME COM
80808005 S    1731  4423  4422  0.0  41   0 408K wait     p1      0:01.06 ksh
$
```

Additional information includes the parent process ID (PPID), the user ID (UID), and priority of the job (PRI) .

Job Control Commands

Overview

To control and manage a job, you need commands that will:

- Stop, suspend or terminate job execution
- Move a job between the background and foreground
- Coordinate the execution of multiple jobs

Suspending a Job

While waiting for a command or shell script to complete, you often realize that there is something you forgot to do. You can suspend the present job while you correct the problem, then continue where you left off with the suspended job. Use the SUSP character (usually Ctrl/Z) to suspend a foreground job.

In the following example, the `vi` editor is started, then suspended with Ctrl/Z. The number in brackets, 3, is the job number. The process identification number is 10147. The job, `vi notice.txt` is stopped.

Example 13–9 Suspending the Present Process

```
$ vi notice.txt
~
~
    .
    .
    .
~
"notice.txt" [New file]
Ctrl/Z
[3] + 10147     Stopped                 vi notice.txt
$
```

Placing a Job in the Foreground

The `fg` command places a stopped or background job in the foreground.

The format for the `fg` command is:

```
fg job_name
```

If `job_name` is not used, the most current job, as indicated by the plus (+) in the `jobs` output, is placed in the foreground. The `job_name` may be referred to by job number when preceded with the per cent sign (`%`), or the PID.

The example shows how to use the `fg` command.

Example 13–10 Using the fg Command

```
$ fg          Places the current job in the foreground
$ fg %3       Places job 3 in the foreground
$ fg 10147    Places job with PID 10147 in the foreground
```

Restarting a Job in the Background

The `bg` command restarts a stopped job in the background.

The format of the `bg` command is:

```
bg job_name
```

When used without the job name, the most recently stopped job is restarted in the background. Otherwise, `bg` needs a job number (preceded by the percent sign), or the PID. The example shows how to use the `bg` command.

Example 13–11 Using the bg Command

```
$ bg %2    Places job 2 in the background
```

The kill Command

You can communicate with a foreground process using the terminal keys, such as Ctrl/C, but you need special commands to communicate with a background job.

The `kill` command sends a signal to a background job. Use `kill` to terminate a job, or, when used with the `trap` command, to direct a script to carry out an appropriate action.

If the process does not catch, or ignores the signal, the command will probably terminate the process.

The format of the `kill` command is:

```
kill [-signal] job_name
```

The `kill` signals are displayed with the `kill -l` command. The table shows the `kill` commands on a DEC OSF/1 system. They may be different on your system.

Table 13–3 DEC OSF/1 kill Commands

1—HUP	7—EMT	13—PIPE	19—CONT	25—XFSZ
2—INT	8—FPE	14—ALRM	20—CHLD	26—VTALRM
3—QUIT	9—KILL	15—TERM	21—TTIN	27—PROF
4—ILL	10—BUS	16—URG	22—TTOU	28—WINCH
5—TRAP	11—SEGV	17—STOP	23—POLL	29—PWR
6—LOST	12—SYS	18—TSTP	24—XCPU	30—USR1
				31—USR2

Using the kill Command

The signal is specified with either a name or a number. The default kill signal for the DEC OSF/1 system is *TERM*, which terminates any process that does not ignore or catch the signal. The example shows how to use the `kill` command.

Example 13–12 Using the kill Command

```
$ kill %4 ❶
$ kill -QUIT %1 ❷
$ kill -9 10652 ❸
```

❶ Sends the default signal, TERM, to job 4. This signal normally terminates processes that do not ignore or catch the signal.

❷ Sends the QUIT signal to job 1.

❸ Sends the KILL signal to job with PID 10652. This is a more certain kill, and may terminate the process even if TERM does not terminate it.

Process Priority

The operating system assigns each process a process priority, which is used to schedule processes and may affect how long it takes for a job to complete. The actual priority values are system dependent.

The UNIX operating system provides the `nice` command, which requests that a process priority should be adjusted. The `nice` value:

- Controls how fast your process executes
- Ranges from -20 (highest priority) to 19 (lowest priority)
- Default is ten (10)
- The superuser may use the highest priorities, which are negative numbers

The operating system maps the `nice` value to an appropriate priority on that system.

Requesting Process Priority

Use the `nice` command to request process priority.

The format of the `nice` command is:

```
nice [-priority] command [argument ...]
```

Nonsuperusers can only **decrease** their priority; they cannot **increase** their priority.

The example shows the priority of the script file `find.ksh` running in the background with a decreased priority of 15.

Example 13–13 Using the nice Command

```
$ nice -15 find.ksh &
```

Note

If the superuser wanted to increase the priority of a process to -15, the entry would be `nice --15`. The hyphen (-) must be used.

Some useful priorities are shown in the following table.

Table 13–4 Useful Priorities for the nice Command

Priority Level	Description
+19	Process runs only when nothing else in the system is running
+10	Default level when no number is specified with `nice`
0	The base scheduling priority
-1 to -20	Makes processes run very fast (superuser only)

You can also use the `renice` command to change the scheduling priorities of one or more running processes. The arguments to `renice` are the process group, user name, or process ID. Only the superuser can make a process run faster.

Making a Process Sleep

The `sleep` command suspends process execution for a specified number of seconds.

Use the `sleep` command to cause execution of a command after a certain amount of time, or to execute a command every so often.

The format of the `sleep` command is:

`sleep` *seconds*

Suppose you need to take medication every hour during a work day. You could create the file `medicine.ksh` to include the following:

```
print "It is time to take your medicine!"
sleep 3600; print "It is time to take your medicine!"
sleep 3600; print "It is time to take your medicine!"
sleep 3600; print "It is time to take your medicine!"
sleep 3600; print "It is time to take your medicine!"
sleep 3600; print "It is time to take your medicine!  Only one more to go!"
sleep 3600; print "It is time to take your medicine!  Last time today!"
```

The shell script could then be executed in the background as follows:

```
$ medicine.ksh &
$
```

Making a Process Wait

Background processing is a convenient way to do large jobs when interaction with programs is unnecessary. However, you might have to wait for a background job to finish executing before you can complete other tasks.

The format of the `wait` command is:

`wait` *[job_name]*

For example, enter the `ls -R` command to list all the files in each subdirectory starting at the root directory.

Assume that you need to use the list of directories in the script you are writing. Use the `wait` command to cause the Korn shell to wait until the background job has completed.

The example provides a sample command you can use to wait until the background job has completed.

Example 13–14 Waiting for Completion of a Background Job

```
$ ls -laR /usr | grep ^d  > dir-list &
[1]     586
$ wait 586; print "The directory list is complete"
The directory list is complete
$
```

The `wait` command is a Korn shell built-in command. Since all background jobs are children of the shell, the login shell must wait for the child processes.

wait Command Options

The `wait` command with no options waits for all background jobs to complete. The options to the `wait` command are shown in the following table.

Table 13–5 Korn Shell Options to the wait Command

Option	Command	Waits for...
-none-	`wait`	All background processes to complete
`%`*job_number*	`wait %2`	Job number 2 to complete
process_number	`wait 346`	Process 346 to complete
`%!`	`wait %!`	Last background process to complete

Exit Status Variable

Any command you submit to the shell either succeeds and displays results on the screen, or fails and prints a diagnostic error message.

The shell sets a special variable **$?** with the exit status of the command. If the command is successful, **$?** is set to zero (0). Any other value usually indicates an error. The meanings of the status codes for specific commands are described in the documentation.

In the example, the first grep command fails and the exit status is 1. The second grep command succeeds and the exit status is 0.

Example 13–15 Checking Exit Status

```
$ grep stu01 /etc/passwd
$ print $?
1
$ grep stu1 /etc/passwd
stu1:xrusfP3Jy5TiE:1713:15:Student One:/usr/class/stu1:/bin/ksh
$ print $?
0
$
```

Learning Activity: Job Control Commands

Running Jobs in the Background

1. Issue the following command to run a job in the background:

   ```
   $ (sleep 20; ls ) &
   ```

   a. Issue the ps command.

   b. What is the job number for this command?

   c. What is the PID?

2. Issue the following command to run in the background:

   ```
   $ (sleep 20; ls ) &
   ```

   Now issue the command to bring that job back into the foreground.

Setting the Priority of a Process

This exercise reduces the priority of a command.

1. Issue the command ps -al to get a long listing of all your processes. The NI column shows the equivalent nice value.
2. Now modify the command to set its nice value to 10.

Stopping a Process

Start up the vi editor by issuing the following command:

```
$ vi file.txt
```

a. Stop the process by typing Ctrl/Z.

b. Determine the job number of the process.

c. Restart the editing job.

d. Exit the editing job by typing ZZ.

Killing a Process

1. Issue the following command and then suspend the process:

   ```
   $ (sleep 30; ls /)
   ```

2. Kill the stopped process.

Using the exit status Variable

1. Enter the following command to the shell, substituting your login name for you:

```
$ grep "you" /etc/passwd
```

   Now list the exit status code of the grep command.

2. Enter the following command to the shell:

```
$ grep "anymouse" /etc/passwd
```

   Print out the exit status code.

Learning Activity Solution: Job Control Commands

Running Jobs in the Background

1. Issue the following command to run a job in the background:

```
$ (sleep 20; ls ) &
```

   a. Issue the ps command.

   b. What is the job number for this command?

   c. What is the PID?

```
$ (sleep 20; ls) &
[1]     4529
$ ps
  PID TT  S           TIME COMMAND
 4423 p1  S        0:00.78 -ksh (ksh)
 4529 p1  S N      0:00.01 -ksh (ksh)
 4530 p1  S N      0:00.01 sleep 20
$
```

   The job number is 1; the PID is 4529. Your results will vary but should be similar.

2. Issue the following command to run in the background:

```
$ (sleep 20; ls ) &
```

   Now issue the command to bring that job back into the foreground.

```
$ (sleep 20; ls) &
[2]     4532
$ fg
(sleep 20; ls)
```

   You must wait for the command to complete.

Setting the Priority of a Process

This exercise reduces the priority of a command.

1. Issue the command ps -al to get a long listing of all your processes. The NI column shows the equivalent nice value.

2. Now modify the command to set its nice value to 10.

```
$ ps -al
      F S      UID   PID  PPID %CPU PRI  NI  RSS WCHAN    TT         TIME COM
80808005 I   +    0   348     1  0.0  44   0  56K ttyin    co      0:00.05 get
80808005 S     1731  4423  4422  0.0  41   0 408K wait     p1      0:01.00 ksh
80808005 R   +    0  4565  4423  0.0  44   0 352K -        p1      0:00.02 ps
$ nice -10 ps -al
      F S      UID   PID  PPID %CPU PRI  NI  RSS WCHAN    TT         TIME COM
80808005 I   +    0   348     1  0.0  44   0  56K ttyin    co      0:00.05 get
80808005 S     1731  4423  4422  0.0  41   0 408K wait     p1      0:01.01 ksh
80808005 R N+    0  4566  4423  0.0  54  10 352K -        p1      0:00.04 ps
```

Stopping a Process

Start up the vi editor by issuing the following command:

```
$ vi file.txt
```

a. Stop the process by typing Ctrl/Z.

b. Determine the job number of the process.

c. Restart the editing job.

d. Exit the editing job by typing ZZ.

```
$ vi file.txt
.
.
.
~
~
"file.txt" [New file]
Ctrl/Z
[2] + Stopped                    vi file.txt
$ fg %2
.
.
.
~
~
ZZ
$
```

Killing a Process

1. Issue the following command and then suspend the process:

```
$ (sleep 30; ls /)
```

2. Kill the stopped process.

```
$ (sleep 30; ls /)
Ctrl/Z
[2] + Stopped                    (sleep 30; ls /)
$ kill -9 %2
```

Using the exit status Variable

1. Enter the following command to the shell, substituting your login name for you:

```
$ grep "you" /etc/passwd
```

Now list the exit status code of the `grep` command.

```
$ grep stu1 /etc/passwd
stu1:xrusfP3Jy5TiE:1713:15:Student One:/usr/class/stu1:/bin/ksh
$ print $?
0
```

2. Enter the following command to the shell:

```
$ grep "anymouse" /etc/passwd
```

Print out the exit status code.

```
$ grep anymouse /etc/passwd
$ print $?
1
```

Note

Notice that the status was equal to zero when the command succeeded (above), but was not equal to zero when the command failed.

Scheduling Jobs to Run at Appropriate Times

Overview

You may have commands that you execute in sequence frequently and you may want them to execute at a specific time as a background job. UNIX has two methods for executing commands (once) at some time in the future. The commands are:

- The `at` command: executes commands once at a specified time on the command line
- The `batch` command: executes commands once when the system load permits; input comes from standard input

The at and batch Commands

The `at` command has the following formats:

```
at [-cskm] time [date] [+increment] [command | file] ...
at -l[user]
at [-fi] -r job_number ...  [-u user] [command | file] ...
at -n[user]
```

The `batch` command has the format:

```
batch
```

at Command Options

The following table lists the functions of some widely used options to the `at` command. The `man` page for `at` describes others.

Table 13–6 at Command Options

Option	Function
-c	Requests that the C shell be used to execute this job.
-k	Requests that the Korn shell be used to execute this job.
-s	Requests that the Bourne shell be used to execute this job.
-m	Mails a message on successful command execution.
-l	Reports your scheduled jobs. If the root user issues the command with this flag, all of the queued at commands are listed with the name of the user who issued each one. The root user can also request a report of scheduled jobs for the specified user only.
-r *job_number*	Removes a job previously scheduled by at or batch, where job_number is the number assigned by at or batch. If you do not have root user authority, you can remove only your own jobs.
-n	Requests the number of files in the queue for the current user. The root user can specify a different user with the user argument.
-f	Suppresses delete verification.
-i	Specifies an interactive delete.
-u *username*	Deletes all jobs for the specified username. This option must be used with the -r option.

Using the at and batch Commands

Some at and batch commands are shown in the following example.

Example 13–16 Some at and batch Commands

```
$ at -km 2300 cleanup.ksh                          ❶
job you.707972400.e will be run at Sun Jun 07 23:00:00 1992
$ at -km now +2 days cleanup.ksh                   ❷
job you.708117700.e will be run at Tue Jun 09 15:21:40 1992
$ at -km 2300 June 10 cleanup.ksh                  ❸
job you.708231600.e will be run at Wed Jun 10 23:00:00 1992
$ batch                                            ❹
cleanup.ksh
Ctrl/D
job you.707944919.b will be run at Sun Jun 07 15:21:59 1992
```

(continued on next page)

Example 13–16 (Cont.) Some at and batch Commands

```
$ at -l                                              ❺
you.707972400.e Sun Jun 07 23:00:00 1992
you.708117700.e Tue Jun 09 15:21:40 1992
you.708231600.e Wed Jun 10 23:00:00 1992
$ at -r you.707972400.e                              ❻
at file: you.707972400.e deleted
$ at -l                                              ❼
you.708117700.e Tue Jun 09 15:21:40 1992
you.708231600.e Wed Jun 10 23:00:00 1992
$ at -u you                                          ❽
unknown word, you
$ at -ru you                                         ❾
at file: you.708117700.e deleted
at file: you.708231600.e deleted
$ at -l                                              ❿
$
```

❶ Use `at` to execute `cleanup.ksh` at 11 PM today

❷ Use `at` to execute `cleanup.ksh` at the present time, but two days from now

❸ Use `at` to execute `cleanup.ksh` at 11 PM June 10

❹ Use `batch` to execute `cleanup.ksh` when the system load allows

❺ Check the scheduled `at` and `batch` jobs

❻ Use the `at -r` option to remove job you.707972400.e

❼ Check the scheduled jobs. Job you.707972400.e has been deleted

❽ Try to delete the remaining jobs for user you without using the `-r` option; it does not work

❾ Delete the remaining jobs with `at -ru you`

❿ No jobs left

The cron Daemon

The system has a daemon called `cron` that runs shell commands at a specified date and time on a periodic basis. The `cron` daemon is started when the operating system is initialized and continues running until system failure or shutdown. It creates a log of activities and periodically checks `crontab` and `at` files in the directory `/usr/spool/cron/crontabs` for jobs.

The `cron` daemon starts jobs contained in the `/usr/spool/cron/crontabs` directory by:

- Invoking a subshell from the invoking process' `$HOME` directory
- Not running `.profile` file
- Supplying a default environment for every shell, defining **HOME, LOGNAME, SHELL** (`/usr/bin/sh`), and **PATH** (`/usr/bin`)

- Initiating jobs with the attributes stored with the job by the invoking process:
  - Login user ID
  - Effective and real user IDs
  - Effective and real group IDs
  - Supplementary groups

Using the crontab Command

Use the crontab command to place jobs in the queue for the cron daemon to process. The crontab command takes as input, a file that contains information about the commands and when you want to execute them. The format of this file is discussed later.

The crontab command has the following formats:

```
crontab [file]
crontab -l | -v
crontab -r
```

Table 13–7 crontab Command Options

Option	Function
-l	Displays the contents of your crontab file
-r	Removes the crontab file from the crontab directory
-v	Displays the name of your crontab file and the date and time at which you submitted it with crontab

The following are some crontab command examples.

```
$ crontab weekly_cleanup

$ crontab -l
30 13 * 1 1-5 echo "I was cronned"

$ crontab -v
crontab file: you submission time: Thu Jan 02 13:00:29 1992
```

Creating a crontab File

The format of a crontab file is shown in the following example. You can create the file with a text editor and then submit the file using the crontab command.

Each user can have one crontab file active at a time. The crontab file can have more than one entry line. Each line in the file consists of six fields which are separated by spaces or tabs.

The above entry instructs cron to run the program called /usr/adm/cleanup at 8:00 p.m. every Monday through Friday of every month. Each of the first five fields may contain:

- A number (within the specified range as shown above)

Example 13–17 crontab File Entry

```
0 20 * * 1-5 /usr/adm/cleanup
^ ^  ^ ^  ^  \--------------/
| |  | |  |          ^
| |  | |  |          |
| |  | |  |          Command to be executed
| |  | |  Day of the week (0 - 6 with 0 = Sunday)
| |  | Month (1-12)
| |  Day of the month (1-31)
| Hour (0-23)
Minute (0-59)
```

- Two numbers separated by a dash to indicate an inclusive range
- A list of numbers, separated by commas, which selects all number in the list
- An asterisk, which indicates all legal values

The following is a `crontab` file example.

```
# @ (#) crontab DEC OSF/1 Version 2.0
#
#
0  5 1 *   * /usr/adm/billing
0 20 * * 1-5 /usr/adm/cleanup
```

Learning Activity: Scheduling Jobs

1. Create a script file that will execute the `ls` command in your home directory. Put a date stamp on the operation by using the `date` command. After you have completed this step, submit the file using the `at` command to run within five minutes. After the job has executed, check your mail to see the results.
2. Modify the script file created in the step above to redirect the output to a file. Then submit the job to run in a few minutes.
3. Use `crontab` to schedule a job for execution.
   a. Create a `crontab` file that will capture the output from the `date` command into a file called `crontab.txt`. Have the command execute three times within the next hour.
   b. Submit the file with the `crontab` command.
   c. Check to see if it executed correctly, then remove your `crontab` file.

Learning Activity Solution: Scheduling Jobs

1. Create a script file that will execute the `ls` command in your home directory. Put a date stamp on the operation by using the `date` command. After you have completed this step, submit the file using the `at` command to run within five minutes. After the job has executed, check your mail to see the results.

```
$ cat listing.ksh
ls
date
$ chmod 755 listing.ksh
$ at -m now + 5 minutes listing.ksh
job you.757608283.a at Mon Jan 03 09:44:43 1994
$ Mail
Mail $Revision: 4.2.4.2 $  Type ? for help.
"/usr/spool/mail/you": 2 messages 2 new
>N  1 you                  Mon Jan  3 09:44  9/279
 N  2 root                 Mon Jan  3 09:44  56/698
? 1
Message  1:
From you Mon Jan  3 09:44:44 1994
Date: Mon, 3 Jan 1994 09:44:44 -0500
From: User for U and C <you>
At: Job you.757608283.a was run.
Apparently-To: you

? 2
Message  2:
From root Mon Jan  3 09:44:45 1994
Received: by sys1.dec.com; id AA05137; Mon, 3 Jan 1994 09:44:44 -0500
Date: Mon, 3 Jan 1994 09:44:44 -0500
From: system PRIVILEGED account <root>
Message-Id: <9401031444.AA05137@sys1.dec.com>
Apparently-To: you

TUTORIAL
for_dir.ksh
    .
    .
    .
Mon Jan  3 09:44:44 EST 1994

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

?
```

2. Modify the script file created in the step above to redirect the output to a file. Then submit the job to run in a few minutes.

```
$ cat listing.ksh
date > daily-log.txt
ls >> daily-log.txt
$ at now + 2 minutes listing.ksh
job you.757610265.a at Mon Jan 03 10:17:45 1994
$ cat daily-log.txt
Mon Jan  3 10:17:46 EST 1994
TUTORIAL
daily-log.txt
for_dir.ksh
    .
    .
    .
```

3. Use `crontab` to schedule a job for execution.
   a. Create a `crontab` file that will capture the output from the `date` command into a file called `crontab.txt`. Have the command execute three times within the next hour.
   b. Submit the file with the `crontab` command.

c. Check to see if it executed correctly, then remove your `crontab` file.

```
$ date
Mon Jan  3 10:36:47 EST 1994
$ cat mycron
40,45,50 10 * * * date >> crontab.txt
$ crontab mycron
$ crontab -l
40,45,50 10 * * * date >> crontab.txt
$ date
Mon Jan  3 10:42:33 EST 1994
$ cat crontab.txt
Mon Jan  3 10:40:00 EST 1994
$ date
Mon Jan  3 10:50:20 EST 1994
$ cat crontab.txt
Mon Jan  3 10:40:00 EST 1994
Mon Jan  3 10:45:01 EST 1994
Mon Jan  3 10:50:00 EST 1994
$ crontab -r
$ crontab -l
crontab: can't open your crontab file.
$
```

Summary

Introduction to the Process Concept

- A process is a program that is executing
- A process has information associated with it called the environment
- You create a process by running programs and commands
- Jobs can be run in the background by appending an & at the end of the command line

Obtaining Process Status

You can check the status of a job using the `jobs` or `ps` command.

Job Control Commands

- Use the `nice` command to change the priority of a process.
- Use the `sleep` command to cause a process to stop executing.
- Use the `wait` command to control the execution of multiple processes.
- Signal a process using the `kill` command.
- Temporarily halt execution of a process using the `stop` command.

Scheduling Jobs to Run at Appropriate Times

To run commands once at a later time, use the:

- `at` command
- `batch` command

To run commands periodically at a later time, use the:

- `crontab` command

14

Introduction to Korn Shell Scripts

Korn Shell Scripts

Introduction

Now that you are familiar with the file system and shell commands of the UNIX operating system, you can begin customizing your environment with **scripts**, which are short programs of shell commands.

This chapter discusses Korn shell script writing at an introductory level. More advanced topics such as arrays and functions are covered in the *KornShell Programming* course.

Objectives

To use the UNIX operating system effectively, you should be able to:

- Identify and create Korn shell scripts
- Make Korn shell scripts executable
- Define variables in Korn shell programming
- Use variables in Korn shell scripts
- Use built-in variables in Korn shell scripts
- Use conditionals and loops to control flow in Korn shell scripts
- Execute further commands based on process exit status
- Debug Korn shell scripts

Resources

- *OSF/1 User's Guide*
- *ULTRIX to DEC OSF/1 Migration Guide*

Korn Shell Scripts

Overview

Shell scripts help you to customize the environment in your `.profile` file and perform repetitive tasks of several commands. Shell scripts are also useful for jobs that must be submitted in background mode.

As the newest shell programming environment, the Korn shell has additional features that the C and Bourne shells lack, and it is backward compatible with the Bourne shell.

Creating Shell Scripts

Shell scripts are easy to create, debug, and maintain. Use any editor to create a file that contains the commands to be executed.

The following example is a simple Korn shell script. The first line of the script indicates the shell type. After the first line, any text following the pound sign (#) on a line is a comment to explain the program and will not be executed.

Example 14–1 Simple Korn Shell Script

```
#!/bin/ksh
#
# intro-script.ksh
#
# Change directory to the home directory,
# print the name and contents of the directory,
# change directory to the book directory and
# print the name and contents of that directory,
# then return to the home directory.
#
cd
pwd
ls
cd book
pwd
ls
cd
```

To execute the script, type the following command:

```
$ ksh intro-script.ksh
/usr/users/you
book    misc    time.status
/usr/users/you/book
chap1   chap2   contents.dis    intro.txt
$
```

Executing Korn Shell Scripts

There are three ways to execute a Korn shell script:

- Invoke a subshell to run it

```
$ ksh filename
$
```

- Run the script in your current shell

```
$ . filename
$
```

- Make the script executable, then invoke it as a command

```
$ chmod u+x filename
$ filename
$
```

Korn Shell Environment

The first line of your Korn shell script should be `#!/bin/ksh`, both to inform the program reader and force execution of the script in the Korn shell environment if invoked directly.

Executing Korn shell scripts can be done in your current directory or remote directories by specifying the search path on the command line. The following example shows how to modify the **PATH** variable to execute a Korn shell script that has been made executable.

Example 14–2 Modifying the PATH Variable

```
$ cd
$ pwd
/usr/users/you
$ intro-script.ksh
ksh: intro-script.ksh: not found
$ print $PATH
/usr/bin:/sbin:/usr/sbin
$ PATH=$HOME:$PATH
$ print $PATH
/usr/users/you:/usr/bin:/sbin:/usr/sbin
$ intro-script.ksh
/usr/users/you
book    misc    time.status
/usr/users/you/book
chap1   chap2   contents.dis    intro.txt
$
```

Learning Activity: Creating and Executing Korn Shell Scripts

These lab exercises explore the creation and execution of simple Korn shell scripts.

1. Create a Korn shell script that:
   - Explains what the script will do in a comment area
   - Shows you your current directory
   - Shows you who is currently logged in to the system

- Prints the current date and time

2. Demonstrate which methods of invoking a shell script create a subshell and which execute the script in your current shell.
   a. Create a Korn shell script that executes the `ps` command, and also prints the value of the shell variable `hi`.
   b. Set the value of `hi` to "HELLO" in your current shell. Do not export this value.
   c. Execute your Korn shell script three ways.

Learning Activity Solution: Creating and Executing Korn Shell Scripts

These lab exercises explore the creation and execution of simple Korn shell scripts.

1. Create a Korn shell script that:
   - Explains what the script will do in a comment area
   - Shows you your current directory
   - Shows you who is currently logged in to the system
   - Prints the current date and time

```
$ cat yourscript
#!/bin/ksh
#
#This script shows your current directory
#then shows who is on the system
#then prints the date and time
#
pwd
who
date
$ . yourscript
/usr/users/you
you          ttyp1       Dec 30 11:13
Thu Dec 30 11:53:29 EST 1993
```

2. Demonstrate which methods of invoking a shell script create a subshell and which execute the script in your current shell.
   a. Create a Korn shell script that executes the `ps` command, and also prints the value of the shell variable `hi`.

```
$ cat ps.ksh
#!/bin/ksh
#
# Korn shell script to display active processes
#
ps
print $hi
$
```

   b. Set the value of `hi` to "HELLO" in your current shell. Do not export this value.

```
$ hi=HELLO
$
```

c. Execute your Korn shell script three ways.

```
$ ksh ps.ksh
  PID TT  S           TIME COMMAND
  573 p5  S        0:03.53 -ksh (ksh)
 4348 p5  S  +     0:00.02 ksh ps.ksh
$ . ps.ksh
  PID TT  S           TIME COMMAND
  573 p5  S  +     0:03.55 -ksh (ksh)
HELLO
$ chmod u+x ps.ksh
$ ps.ksh
  PID TT  S           TIME COMMAND
  573 p5  S        0:03.57 -ksh (ksh)
 4352 p5  S  +     0:00.02 ksh ps.ksh
$
```

Korn Shell Variables

Overview

The Korn shell allows you to use variables interactively, and within shell scripts, to modify information that has changing values. For example, the variable **PATH** returns the pathname search string set up for you in your `.profile` file when you log in to your account, however, you can modify that value to include new directories. Variables such as **PATH** are reserved by the shell. Using both shell and user-created variables in Korn shell scripts allows you to read in new information or change the output of your scripts to meet changing needs.

Variable Assignment

The format for a variable assignment is: `identifier=value`. No unquoted spaces or tabs are allowed before or after the equal sign. Assignment of variable values can be done in a variety of ways, as shown in the table.

Table 14–1 Variable Assignment

Example	Function
`$ nvalue=100`	Assigns a numerical value to the variable, **nvalue**. By default, all variables are strings
`$ typeset -i nvalue=50`	Declares `nvalue` to be an integer and assigns the value of 50 to it
`$ integer y=100`	Forces `y` to be an integer value
`$ dog=cat`	Assigns the value "cat" to the variable **dog**
`$ set`	Displays names and values of all variables
`$ unset dog`	Unassigns a variable value

Setting, Displaying, and Unsetting Variables

Example 14–3 provides examples of commands for setting, displaying, and unsetting variables.

Example 14–3 Assigning and Displaying Variables

```
$ nvalue=100
$ print $nvalue
100
$ integer y=100
$ print $y
100
$ dog=cat
$ print $dog
cat
$ print $PATH
/usr/users/you:/usr/bin:/sbin:/usr/sbin:.
$ unset dog
$ print $dog

$
```

Using Variables

You can substitute your own examples of variable usage, depending upon your experience.

Example 14–4 uses the shell variable, **USER**, which returns the value of the current user's user name. The example also shows how variables provide a way to easily modify information that a command may need to execute.

Example 14–4 Using Korn Shell Variables

```
$ cat greeting.ksh
#!/bin/ksh
#
# greeting.ksh
#
# This Korn shell script uses variables
# to issue a welcome message to the user.
#
message="Good Morning"                                    ❶
application="Shell Programming"                           ❷
print ""                                                  ❸
print "$message, $USER!"                                  ❹
print ""
print "Welcome to the Wonderful World of ..."             ❺
print ""
print "       $application!"
$
$ greeting.ksh                                            ❻
$

Good Morning, you!

Welcome to the Wonderful World of ...

       Shell Programming!
$
```

❶ If the time of day changes, modify the value of `message`.

❷ If the application changes, modify the value of `application`.

❸ The `print` command is used in the Korn shell to print the value of a variable. The `echo` command can also be used.

❹ Placing a dollar sign (`$`) in front of a variable returns the value of the variable.

❺ Quotation marks (`"`) are used in Korn shell programming to enclose a string and return a literal value except for special characters such as backslash (`\`).

❻ Execute the shell script, `greeting.ksh`.

Using the read Command

There are times when the value of a variable must be supplied by the user at run time. The value is usually different each time the script is executed. A shell script that allows its user to enter information at run time is known as an **interactive** script.

Example 14–5 uses the `read` command, followed by the variable name, to allow the script to enter information interactively.

Example 14–5 Using the Korn Shell read Command

```
$ cat printname.ksh
#!/bin/ksh
#
# printname.ksh
#
# This Korn shell script prompts the user for his or her first, middle,
# and last name, then prints the results.
#
print ""
print "Enter your first name:"                    ❶
read first                                        ❷
print "Enter your middle name:"
read middle
print "Enter your last name:"
read last
print ""
print "Your name is $first $middle $last."        ❸
$
$ ksh printname.ksh

Enter your first name:
Garfield
Enter your middle name:
the
Enter your last name:
Cat

Your name is Garfield the Cat.
$
```

❶ Prompts the user every time the script is run

❷ Assigns the variable with the `read` command

❸ The values of the variables are returned by placing the `$` in front of the variable name

Korn Shell Built-in Variables

The Korn shell provides built-in variables that have values with special meaning to the shell. Issuing the `set` command returns the values of the built-in variables, as shown in Example 14–6. The `man` pages under `ksh` list and explain all the built-in variables.

Example 14–6 Korn Shell Built-In Variables

```
$ set
ERRNO=10
FCEDIT=/bin/ed
HOME=/usr/users/you
IFS='
LINENO=1
LOGNAME=you                    ❶
MAILCHECK=600
OPTIND=1
PATH=/usr/bin:.                ❷
PPID=2822                      ❸
PS1='$ '                       ❹
PS2='> '
PS3='#? '
PS4='+ '
PWD=/usr/users/you             ❺
RANDOM=8944
SECONDS=27792
SHELL=/bin/ksh                 ❻
TERM=vt100                     ❼
TMOUT=0
USER=you                       ❽
_=set                          ❾
nvalue=100                     ❿
y=100
```

❶ The name of the user

❷ Colon-separated command search path

❸ The user's parent process identification number

❹ The parent prompt string; PS2 provides a prompt for forked shells. PS3 and PS4 provide prompts for built-in commands

❺ The user's current working directory

❻ The pathname of the user's shell

❼ The type of terminal you are using

❽ The user's login name

❾ A temporary variable containing the last argument of the previous shell command; a portion of the previous shell command executed

❿ A user-created variable

Nonmodifiable Variables

Some variables are set by the Korn shell and should not be modified by the user.

- **ERRNO** — value set by most recent failed system call for debugging purposes
- **LINENO** — line number of the current line being executed within a script
- **PPID** — process number of the shell parent

- **PWD** — present working directory set by the `cd` command
- **RANDOM** — random integer generator ranging from 0 to 32767
- **SECONDS** — returns a value in seconds since the shell was invoked

Modifiable Variables

The Korn shell gives default values to other variables. You can modify them if you choose:

- **PATH** — search path for commands.
- **PS1** — primary prompt string, which by default is a $ in Korn shell.
- **PS2** — secondary prompt string, which by default is a >. Appears when you have pressed Return without having entered a complete command.
- **MAILCHECK** — specifies in seconds how often the shell checks the mail files for new mail.
- **TMOUT** — if set to a value greater than zero, the shell terminates if you do not enter a command within the prescribed number of seconds (plus an additional 60 seconds grace period).
- **IFS** — internal field separator. Normally spaces, tabs, and new lines are used to separate command words or parameters.
- **HISTSIZE** — maximum number of previously executed commands that you can access. The default is 128.

Using Built-in Variables

To have a customized environment, you can define the modifiable built-in variables such as **PATH, PS1, CDPATH**, and **HISTSIZE** in your `.profile` file in your login directory. The example shows a sample of the commands used to set these variables.

Example 14–7 Sample Korn Shell .profile File

```
$ cat .profile
print "executing .profile"
PS1=' ! $PWD> '                          ❶
export PS1                               ❷
export TERM=vt100 \                      ❸
       CDPATH=.:..:"$HOME:/usr" \
       HISTSIZE=150
set -o noclobber                         ❹
```

❶ Customizes the prompt to be the command number followed by the present working directory and an angle bracket (>)

❷ Makes the prompt a global variable by exporting its value to a child process

❸ Makes the **TERM, CDPATH**, and **HISTSIZE** variables global to any child process

❹ Prevents accidental overwriting of files

Global Variables

When a variable is defined, its value is available only in the current shell. To make a variable global or available to child processes of your current shell, use the `export` command after defining the variable as follows:

```
$ dog=cat
$ export dog
$ print $dog
cat
$
```

The `ksh` command can be used to fork a child shell. The global variable can be seen at a child level as follows:

```
$ FOOD=pasta                              ❶
$ print "My favorite meal is $FOOD!!"
My favorite meal is pasta!!
$ ksh                                     ❷
$ print "I love $FOOD"                    ❸
I love
$ Ctrl/D                          ❹
$ export FOOD                             ❺
$ ksh                                     ❻
$ print "I love $FOOD"                    ❼
I love pasta
$
```

❶ Local variable at the parent shell level is created.

❷ Invoke a subshell.

❸ Local variable not available to a child shell.

❹ Terminate the subshell.

❺ Use the `export` command at the parent shell level to make the variable globally available to child shells.

❻ Invoke another subshell.

❼ The global variable is now available to the child shell.

Using Global Variables

Use global variables in conjunction with the `.profile` file since that is where the `export` command defines the variable for global use.

Global variables are specific to how you create them. Although not discussed here, variables within a function are local to that function when created using the `typeset` command. Variables created with the `integer` and `read` command are local to that parent shell.

You can export variables only to your current shell and the children of that shell. Any parent shell of child processes will not know of the exported variables if created at the child level.

By convention, global variables use only uppercase letters.

The typeset command returns the global variables, integer variables, and function variables on your system.

Learning Activity: Using Variables in Scripts

1. Set the value of a variable called first to your first name using typeset first="you". Use set to check the value.
2. Write a short Korn script that uses your variable to welcome you when you log in. Place the script in your .profile file in executable form. Use the greeting script from the lecture as a template.
3. Use the typeset command to define variables and assign attributes.
   a. Define a variable first to be left justified with length 10, with a value of "Charlie".
   b. Define a variable last to be left justified with length 10, with a value of "MacArthy".
   c. Display the values of the two variables next to each other. Redefine the two variables with typeset, but with a length of 8. Use the same print command, and note the difference in output between the two caused by the difference in variable attributes.
   d. Reassign values to the two variables just using = without typeset. Set first to "Pugnacious", and last to "Pugilist". Use the same print command to display them.
   e. Run a second instance of the Korn shell and try to display the variables first and last.
   f. Now, redefine the variables so they will be exported to a second shell. Test this to be sure it works.
4. Write a short script that uses character variables in a loop and then uses integer variables in a loop to see which type of variable executes faster. Use the time command to clock the system time.

Learning Activity Solution: Using Variables in Scripts

1. Set the value of a variable called first to your first name using typeset first="you". Use set to check the value.

```
$ typeset first=you
$ set
   .
   .
   .
first=you
   .
   .
   .
```

2. Write a short Korn script that uses your variable to welcome you when you log in. Place the script in your `.profile` file in executable form. Use the greeting script from the lecture as a template.

```
#!/bin/ksh
#
#
first="Tom"
print "Welcome to the World of $first"
```

3. Use the `typeset` command to define variables and assign attributes.

   a. Define a variable `first` to be left justified with length 10, with a value of "Charlie".

```
$ typeset -L10 first=Charlie
```

   b. Define a variable `last` to be left justified with length 10, with a value of "MacArthy".

```
$ typeset -L10 last=MacArthy
```

   c. Display the values of the two variables next to each other.

```
$ print "$first$last"
Charlie   MacArthy
```

   Redefine the two variables with `typeset`, but with a length of 8. Use the same `print` command, and note the difference in output between the two caused by the difference in variable attributes.

```
$ typeset -L8 first=Charlie
$ typeset -L8 last=MacArthy
$ print "$first$last"
Charlie MacArthy
```

   d. Reassign values to the two variables just using = without `typeset`. Set `first` to "Pugnacious", and `last` to "Pugilist". Use the same `print` command to display them.

```
$ first=Pugnacious
$ last=Pugilist
$ print "$first$last"
PugnacioPugilist
```

   In this case, the value of `first` is truncated.

   e. Run a second instance of the Korn shell and try to display the variables `first` and `last`.

```
$ /bin/ksh
$ print "$first$last"

$ exit
$
```

   The variables are not available to a subsequent shell.

f. Now, redefine the variables so they will be exported to a second shell. Test this to be sure it works.

```
$ typeset -xL11 first=Pugnacious
$ typeset -xL11 last=Pugilist
$ /bin/ksh
$ print "$first$last"
Pugnacious Pugilist
$ exit
$
```

4. Write a short script that uses character variables in a loop and then uses integer variables in a loop to see which type of variable executes faster. Use the `time` command to clock the system time.

```
$ cat expr_times.ksh
#!/bin/ksh
#
# expr_times.ksh
#
# Time loops using character and integer variables
#
print 'Time to do 10000 loops'
print '\nTime using normal variable'
x=1
time while (( x < 10000 )) ; do  let x=x+1; done
#
print '\nTime using integer variable'
integer v            # = typeset -i v;
        # integer vars are faster!
 v=1
 time while (( v < 10000 )) ; do v=v+1 ; done
$ . expr_times.ksh
Time to do 10000 loops

Time using normal variable

real    0m0.95s
user    0m0.93s
sys     0m0.00s

Time using integer variable

real    0m0.55s
user    0m0.55s
sys     0m0.00s
$
```

Flow Control and Loops

Conditional Statements

As you write more complex shell programs, you must be able to verify input and results. The conditional `if-then-else` statement provides for the execution of a set of commands only if a stated condition is **true**. The condition is generally a relational expression in which the value of two or more items is compared. The program reads through the statements until a match, or true condition, is found and then executes the required statement.

```
if (( condition ))
then
      commands
else
      commands
fi
```

Relational Operators

Korn shell programming provides relational operators to make comparison statements with Boolean algebra. Relational operators let you make comparisons in Korn shell scripts and return a numeric, or string value based on a condition being met, much like in conditional looping discussed earlier.

Numeric Relational Operators

Numeric relational operators are shown in the table. The format for a numeric test is:

((`number1 operator number2`)).

Table 14–2 Numeric Relational Operators

Operator	Comparison
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

String Relational Operators

The string operators are shown in the following table. The format for string tests is:

[[`string1 operator string2`]]

Table 14–3 String Relational Operators

Operator	Comparison
=	Equal to
!=	Not equal to
<	Precedes in lexical order
>	Follows in lexical order

Positional Parameters

Positional parameters are used to return the values of command line arguments as shown in Table 14–4.

Table 14–4 Positional Parameters

Parameter	Meaning
`$#`	Number of positional parameters (arguments to shell command)
`$0`	Name of script or function, or pathname of the Korn shell itself if a `set` statement
`$1`	Name of first argument to script, function, or `set`
`$2`	Name of second argument to script, function, or `set`
`$n`[1]	Name of nth argument to script, function, or `set`

[1]If n>=10 it must be isolated with ${n}, for example, ${10}.

Using Relational Operators

In Example 14–8, the `checkname.ksh` script provides an example of the use of relational operators. The script uses command-line variables `$#`, `$0`, `$1 $2 $3` to return values of fields entered on a command line. By default, the field numbering on a command line will start with 0 and can be incremented for each value entered by separating with spaces or tabs. In the example, the `$#` counts the variables entered to check against the relational operator condition of 3 and then executes the appropriate print command.

Example 14–8 Using Relational Operators

```
$ cat checkname.ksh
#!/bin/ksh
#
# checkname.ksh
#
# This script verifies that 3 arguments have been entered on the
# command line and prints them out.
#
if (( $# != 3 ))        # Checks to see if there are 3 variables
                        # on the command line
then
        print "USAGE:  $0 first middle last"
else
        print "Your name is $1 $2 $3."
fi
$
$ checkname.ksh                          ❶
USAGE:  checkname.ksh first middle last
$
$ checkname.ksh Garfield the Cat  ❷
Your name is Garfield the Cat.
$
```

❶ At the first execution of `checkname.ksh` no arguments are provided on the command line, therefore the check for three arguments fails.

❷ In the second execution, the three arguments are provided.

Logical Operators

Korn shell programming provides logical operators to make comparison statements with Boolean algebra.

Examples of logical operators are listed in the table.

Table 14–5 Logical Operators

Operation	Format	Result
OR	| |	Evaluates to **TRUE** if at least one argument is true
AND	**&&**	Evaluates to **TRUE** if both arguments are true

Using Logical Operators

Example 14–9 shows a script that prompts for your age and returns a ticket price based on your input.

Read Example 14–9 slowly and follow the logical operators. Notice their usage in the conditional looping. Also note how the relational operators use greater than or equal to (>=) and less than or equal to (<=). Finally, note the use of the conditional statement `if-then-elif-else` with the double parentheses around the conditional strings. This serves the same function as double brackets when enclosing a compound command.

There are two ways to use the parenthesized conditionals in the example. Either way is correct.

Example 14–9 Using Logical Operators

```
$ cat age.ksh
#!/bin/ksh
#
# age.ksh
#
# Interactive Korn shell script which prompts for age, and
# returns a ticket price based upon age.
#
print "Enter your age -- "
read age
if ((age <= 6 || age >= 64))
then
print "We discount to children and senior citizens:  ticket price is \$2.50"
elif ((age >=40 && age <=43))
then
print "These are difficult years, we won't charge you"
elif ((age >=13 && age <=19))
then
print "We charge double for teenagers:  ticket price is \$10.00"
else
print "Ticket price is \$5.00"
fi
$
$ age.ksh

Enter your age --
100
We discount to children and senior citizens:  ticket price is $2.50
$ age.ksh

Enter your age --
19
We charge double for teenagers:  ticket price is $10.00
$ age.ksh

Enter your age --
40
These are difficult years, we won't charge you
$
```

Flow Control: while Loop

The script `while.ksh` shows the use of the `while` statement which executes as long as a condition is **true**.

Example 14–10 while Loop

```
$ cat while.ksh
#!/bin/ksh
#
# while.ksh
#
# Example of a WHILE loop
# Loop while condition is true
#
integer total=0 n=0
while (( n <= 3 ))
    do
    total=total+n
    n=n+1
done
#
print "Sum while count <= 3 is $total"
$
$ ksh while.ksh
Sum while count <= 3 is 6
$
```

Flow Control: until Loop

The opposite conditional to `while` is the `until` command which operates as long as a condition is **false**. The example provides an `until` loop.

Example 14–11 until Loop

```
$ cat until.ksh
#!/bin/ksh
#
# until.ksh
#
# Example of an UNTIL loop
# Loop until condition is true
#
integer total=0 n=0
until ((n > 3))
    do
    total=total+n
    n=n+1
done
print "Sum until count > 3 is $total"
$
$ until.ksh
Sum until count > 3 is 6
$
```

Compare the `until` loop in Example 14–11 with the `while` loop. The relational operator > is opposite to the one used for the `while` loop.

Flow Control: for Loop

The most common loop command is the `for` command which specifies a list of values for a variable. The example assigns three values to `n` as it loops through the list of elements: 1 2 3.

Example 14–12 for Loop

```
$ cat while.ksh
#!/bin/ksh
#
# while.ksh
#
# Example of a WHILE loop
# Loop while condition is true
#
integer total=0 n=0
while (( n <= 3 ))
    do
    total=total+n
    n=n+1
done
#
print "Sum while count <= 3 is $total"
$
$ ksh while.ksh
Sum while count <= 3 is 6
$
```

Compare the `for` loop with the previous examples. Notice the "in" keyword used as a separator in this example. All three loop examples use the `integer` command to initialize numeric values before the looping begins.

Command Exit Status

Any command you submit to the shell either succeeds and displays results on the screen, or fails and prints a diagnostic error message.

Use the **exit status** to determine whether or not to execute the next command in a shell script.

Execution Based On Success

You can tell the shell to execute a second command if the first succeeds using the Boolean **AND** operator, **&&** (two ampersands with no intervening spaces). The command format is:

```
<first command> && <second command>
```

The example shows a second command providing a success message if the previous command is successful in deleting a file.

Example 14–13 Execution Based On Success

```
$ rm $1 && print "$1 has been deleted."
$
```

Execution Based On Failure

You can also tell the shell to execute a second command if the first fails using the Boolean **OR** operator, | | (vertical bars with no intervening spaces).

The example provides a sample of a command that is executed if the preceding command fails.

Example 14–14 Execution Based On Failure

```
$ grep "$1" datafile || print "$1 is not in the database."
$
```

Signal Trapping

Signals are provided in your UNIX operating system to control the system. To see the signals available on your system, enter `kill -l`.

In programming, signals are used to send a message to a script to stop it altogether or momentarily. Most signals sent to a script will terminate it. To prevent this, programmers try to capture signals rather than let them interfere with their program.

Korn shell programming provides the `trap` command to catch signals and events and capture error information.

The example shows how `trap` can be used to trap incidents of the INT (Ctrl/C) and QUIT (Ctrl/\ ) signals sent to the script. Each occurrence of the INT signal is counted. The QUIT signals are ignored. In this example, after the script is started, numerous Ctrl/C and Ctrl/\, are typed during the 15 seconds it takes to run the script.

Example 14–15 Using the trap Command

```
$ cat traps.ksh
#!/bin/ksh
#
# traps.ksh
#
# Demonstrate the use of interrupt and exit traps
#
integer c=0 j=0
#
trap 'c=c+1; print " ouch $c"' INT #Count INTs (^C)
#
#Print # of ^C's on exit
trap 'print "All done: $c ^Cs"' EXIT
#
trap '' QUIT                #Ignore QUITs (^\)
#
# Wait for 15 seconds
while (( j < 15 ))
do
 sleep 1
 j=j+1
done
$
$ traps.ksh
Ctrl/C ouch 1
Ctrl/ \ Ctrl/ \ Ctrl/C ouch 2
Ctrl/C ouch 3
Ctrl/ \ Ctrl/ \ Ctrl/ \ Ctrl/C ouch 4
All done: 4 ^Cs
$
```

Debugging Shell Scripts

You must be able to correct errors in your Korn shell scripts. The following discussion explains the most common techniques for correcting shell programming errors. If you have an extended interest in Korn shell scripts you will want to learn more extensive debugging techniques than those discussed here.

The `-v` option to the `ksh` command prints each executable line of the script before substitutions are made. The `-x` option prints each executable line after substitutions are made with a + to highlight the execution.

The example uses `age.ksh` to illustrate the use of `ksh -x`.

Example 14–16 Debugging Korn Shell Scripts with ksh -x

```
$ ksh -x age.ksh
+ print Enter your age --
Enter your age --
+ read age
32
+ let age <= 6 || age >= 64
+ let age >=40 && age <=43
+ let age >=13 && age <=19
+ print Ticket price is $5.00
Ticket price is $5.00
$
```

Another debugging technique uses the `nl` command to display shell scripts with line numbers.

The example uses the `while.ksh` file to demonstrate how to use `nl`.

Example 14–17 Using nl to Display Line Numbers

```
$ nl while.ksh
     1  #!/bin/ksh
     2  #
     3  # while.ksh
     4  #
     5  # Example of a WHILE loop
     6  # Loop while condition is true
     7  #
     8  integer total=0 n=0
     9  while (( n <= 3 ))
    10      do
    11      total=total+n
    12      n=n+1
    13  done
    14  #
    15  print "Sum while count <= 3 is $total"
$
```

Scripts running in debug mode are much slower than normal execution, especially if the `ksh -xv` option is combined.

To increase execution performance, many programmers debug pieces of their shell programs and cut down on debugging unnecessary lines of a script by using the following approach:

```
set -x
if  condition
then
    condition
set +x
```

where the `set -x` command turns on debug mode only for the lines needed and then is shut off with `set +x` where normal mode continues.

Learning Activity: Flow Control and Debugging

1. Write a script that will loop through your directory and check for the file type.
2. Write an extension of the short script from lecture that demonstrates the use of `traps` checking for Ctrl/Cs in a program. Add to it the sum of occurrences up to 10, or end after 30 seconds if there were not 10.
3. Create a prompt using a built-in variable that allows you to see line numbers when debugging shell scripts. Export your prompt for use in child shell execution.
4. Write a Korn shell script that searches a file for specified misspelled words and tells you whether or not it has found them.
5. Use the `ksh -xv` command with your misspelling script to trace the execution and debug errors.

Learning Activity Solution: Flow Control and Debugging

1. Write a script that will loop through your directory and check for the file type.

```
#!/bin/ksh
#
# for_dir.ksh
##############
#
# Demonstrate the use of for loops
#
################################################
#
# Loop through each file in the current directory
# saying what type it is
#
for f in * ; do
  print -n - "$f is a "
    if [[ -f $f ]] ; then
         print "regular file"
          elif [[ -d $f ]] ; then
        print "directory"
          else
        print "strange file!"
               fi
         done
```

2. Write an extension of the short script from lecture that demonstrates the use of `traps` checking for Ctrl/Cs in a program. Add to it the sum of occurrences up to 10, or end after 30 seconds if there were not 10.

```
#!/bin/ksh
#
# Demonstrate the use of interrupt and exit traps
#
################################################################
#
integer c=0
integer j=0

trap 'c=c+1; print "ouch $c"' INT    # Count INTs ^Cs
trap 'print "All done: $c ^Cs"' EXIT # Print # of ^Cs on exit
trap '' QUIT            # Ignore QUIT (^\)
#
# Display current traps
#
trap
#
while (( j < 30 )) && (( c < 10 )) ; do
  sleep 1
    j=j+1
    done
```

3. Create a prompt using a built-in variable that allows you to see line numbers when debugging shell scripts. Export your prompt for use in child shell execution.

```
$ PS4='at $LINENO - '
$ export PS4
```

4. Write a Korn shell script that searches a file for specified misspelled words and tells you whether or not it has found them.

```
#!/bin/ksh
#
# misspell.ksh
#
# script for detecting common misspellings in a file
#
#
for file in $@
do
        if [[ -f $1 ]]
 then
     print "Now looking at the file $file:"
     for word in teh tje fiel
     do
           print "The following of $word were detected:"
                 grep -n "$word" $file
           if [[ $? != 0 ]] then
      print "no misspellings found"
    fi
             print ""
     done
        else
     print " the file $file does not exist"
 fi
done
```

5. Use the `ksh -xv` command with your misspelling script to trace the execution and debug errors.

```
$ ksh -xv misspell.ksh chap*
```

Summary

Korn Shell Scripts

Shell scripts are useful for:

- Customizing your environment in your `.profile` file
- Performing repetitive tasks that require several commands
- Running jobs in the background

A shell script can be executed by:

- Invoking it with a subshell
- Running the script in your current shell
- Making it executable and invoking it as a command

Using Variables in Korn Shell Scripts

The Korn shell allows you to use variables to supply information that may be different each time a script is executed.

- Variables for Korn shell scripts can be defined for character or integer values.
- Variables can be defined inside scripts and by the system in the shell programming environment.
- Variable values can be input with the `read` statement.
- Use the `set` command to view variables.

Flow Control and Looping in Korn Shell Scripts

Flow control in Korn shell scripts may involve:

- Use of relational operators for comparisons such as `==` and `!=` and the use of logical operators such as &&, and `| |` in Korn shell scripts.
- Flow control using `if-then-else` statements.
- Use of looping statements such as `while`, `until`, and `for` in scripts.
- Signal trapping using the `trap` command.
- Use of debugging commands such as `ksh -x`, `ksh -v`, and `nl` to see the execution of scripts line by line.

15

Using TCP/IP Networks

TCP/IP Networks

Introduction

This chapter discusses commands that allow access to other systems.

TCP/IP stands for **T**ransmission **C**ontrol **P**rotocol/**I**nternet **P**rotocol. It is often used to refer to the whole suite of internet protocols, which also include the User Datagram Protocol and the Simple Mail Transfer Protocol.

Objectives

To obtain access to other systems, you should be able to:

- Log in to remote systems with the `rlogin` and `telnet` commands
- Execute commands on a remote system with the `rsh` command
- Copy files to and from remote systems with the `rcp` command and the `ftp` utility
- Display Motif applications from remote systems on your own system

Resources

- DEC OSF/1 `man` pages

Accessing Remote Systems

Overview

During the course of your work day, you may need to access another system to locate a manual stored on line, program source files that need to be updated, or to start up programs. This topic covers the commands you can use to gain access to another machine. Once you have access to the other machine, you can issue commands just as you do on your local system.

Network Security

For many TCP/IP applications, the remote host will allow access without a password if one or more of the following apply:

- The local system is included in the remote host's `/etc/hosts.equiv` file, the local user is not the superuser, and the `-l` option is not specified.
- The local system and user name is included in the `$HOME/.rhosts` file in the home directory of the remote user account.
- The remote account does not have a password. This is a security hole and not recommended.

Some commands such as `rlogin` will prompt for a password if none of the above is true, while others such as `rcp` will fail.

Logging In to a Remote System

Use the `rlogin` command to log in to a remote system.

The format of the `rlogin` command is:

`rlogin` *rhost* `[-`*echaracter*`] [-`*8*`] [-`*l username*`]`

Table 15–1 rlogin Options

Option	Function
`-8`	Allows an 8-bit data path at all times. Otherwise, unless the Stop and Continue key sequences on the remote system are not standard, `rlogin` uses a 7-bit data path and the 8th (high) bit of each byte is stripped.
`-`*echaracter*	Changes the escape character to the one you specified.
`-l username`	Changes the remote user name to the one you specified. Otherwise, your local user name is used at the remote host.

rlogin Command Examples

```
$ rlogin sys001
Last login: Mon Mar  7 13:33:14 from sys999

DEC OSF/1 X3.0-8  (Rev. 212); Fri Feb 18 15:28:06 EST 1994
DEC OSF/1 X3.0-8 Worksystem Software (Rev. 212)

There are logfiles that contain a record of your installation.

sys001$
    .
    .
    .
sys001$ exit
$ rlogin sys002 -l smith
Password:
Last login: Mon Mar  7 12:12:20 from sys999

DEC OSF/1 X2.0-8  (Rev. 201.1); Wed Jan 26 13:13:30 EST 1994
DEC OSF/1 X2.0-8 Worksystem Software (Rev. 201)

The installation software has successfully installed your system.

There are logfiles that contain a record of your installation.

sys002$
    .
    .
    .
sys002$ Ctrl/D
$
```

Issuing Commands on a Remote System

One way to issue commands on a remote system is to use the `rsh` command. The `rsh` command connects to the remote host and executes the specified command, or, if no command is specified, performs a login on the remote host.

This command can be used to issue just one command on the remote system, or it can be used to log in to execute multiple commands.

The format of the `rsh` command is:

```
rsh [-dn] [-l user] remote_host [command] [argument ...]
```

rsh Command Options

Options to the `rsh` command are described in the table.

Table 15–2 rsh Command Options

Option	Function
`-d`	Turns on socket debugging on the TCP socket for communication with its copies, ignoring the `umask`
`-l`	Specifies another user ID for login rather than the local user ID
`-n`	Redirects all command input to `/dev/null`; should be used with the C-shell

rsh Command Examples

```
$ rsh sys001 users
evant garcia
$
$ rsh sys002
Last login: Mon Mar  7 14:02:14 from sys999

DEC OSF/1 X2.0-8  (Rev. 201.1); Wed Jan 26 13:13:30 EST 1994
DEC OSF/1 X2.0-8 Worksystem Software (Rev. 201)

There are logfiles that contain a record of your installation.

sys002$
```

Copying Files From a Remote System

One way to copy files between systems is to use the `rcp` command. The name of the local system must be in the `/etc/hosts.equiv` file on the host system, or the system name and user name must be in the `$HOME/.rhosts` file in the account you are copying from.

Two formats of the `rcp` command are:

```
rcp [ -p ] file1 file2
rcp [-r] [-p] file... directory
```

When specifying a file on a remote system, use the remote system name followed by a colon (:) as a prefix to the file name, such as `sys001:/etc/shells`.

To copy files from a particular user account, the user name followed by an at sign (@) should be used as a prefix to the remote system name, such as `you@sys001:.profile`.

The file path will start from that user's home directory.

rcp Options

Options to the `rcp` command are described in the table.

Table 15–3 rcp Options

Option	Function
`-p`	Preserves the modification times and modes of the source files in its copies, ignoring the `umask`
`-r`	Copies files in all subdirectories recursively, if the file to be copied is a directory; destination must be a directory

Note

Be careful when using the `-r` option. Since it copies a directory and all the directories below, it may produce more than you desire.

rcp Command Examples

```
$ rcp  sys001:/usr/users/smith/file.txt file.txt
$ rcp  smith@sys001:file.txt file.txt
$ rcp  -r smith@sys001:/usr/users/smith  \
> smith@sys002:/usr/users/smith
$
```

Using the File Transfer Program

A second way to transfer files is to use the `ftp` program. `ftp` is the user interface to the standard File Transfer Protocol. Once you invoke the `ftp` program, you then issue other commands to accomplish tasks.

The format of the `ftp` command is:

```
ftp [options] [host]
```

For a list of `ftp` options, refer to the `man` page on `ftp`. If you do not specify a remote host, you will be prompted for a command.

ftp Commands

Some `ftp` commands are shown in the following table.

Table 15–4 Some ftp Commands

Command	Function
`open rsys`	Connects to remote system `rsys`
`close`	Closes the current connection
`get`	Receives file
`put`	Sends file
`ls`	Lists contents of remote directory
`quit` OR Ctrl/D	Exits `ftp`
`?` OR `help`	Displays list of `ftp` commands

Refer to the `man` page on `ftp` for a complete list of commands that you may execute.

The following example shows output from the `ftp` command.

Example 15–1 ftp Command

```
$ ftp
ftp> open sys001
Connected to sys001
220 sys001 FTP server (OSF/1 Version 5.60) ready.
Name (sys001:you): Return
331 Password required for you.
Password:
230 User you logged in.
ftp> ls *.ksh
200 PORT command successful.
150 Opening ASCII mode data connection for file list (16.30.0.1).
greeting.ksh
printname.ksh
trap.ksh
226 Transfer complete.
remote: *.ksh
36 bytes received in 0.016 seconds (2.3 Kbytes/s)
ftp> close
221 Goodbye.
ftp> quit
$
```

Using the telnet Program

Another way to connect to a remote system is to use the `telnet` program. This program uses the TELNET protocol for communication between two hosts. One you log in to the remote system, you get a system prompt and can execute any available command.

The format of the `telnet` command is:

```
telnet [host[port]]
```

telnet Commands

If you do not specify a host, you will be prompted for a `telnet` command. Some `telnet` commands are shown in the table.

Table 15–5 Some telnet Commands

Command	Function
`open rsys`	Connects to remote system `rsys`
`close`	Closes the current connection, or you can logout from the remote system
`quit` OR Ctrl/D	Exits `telnet`
`?`	Displays list of `telnet` commands

The following example shows output from the telnet Command.

Example 15–2 telnet Command

```
$ telnet
telnet> open sys002
Trying 16.30.0.2...
Connected to sys002.
Escape character is '^]'.

OSF/1 (sys002) (ttyp3)

login: you
Password:
Last login: Mon Mar  7 14:56:15 from sys999

DEC OSF/1 X2.0-8  (Rev. 201.1); Wed Jan 26 13:13:30 EST 1994
DEC OSF/1 X2.0-8 Worksystem Software (Rev. 201)

There are logfiles that contain a record of your installation.

sys002$
    .
    .
    .
sys002$ exitConnection closed by foreign host.
$
```

Learning Activity: Accessing Remote Systems

To perform these exercises, your system must be connected to a network of UNIX systems using TCP/IP. You must obtain a system name from your system administrator for a remote system that you can log in to and have a user account on that system.

1. Using the `rlogin` command, log in to your remote system.
2. Log out from the system you logged in to in the previous step. Issue the `rcp` command to copy the following file from your remote system to your user directory on your local system: `/usr/include/stdio.h`
3. Issue the `rcp` command as in the previous example, but preserve the modification times and modes of the source file.
4. Issue the `rsh` command to list the `/bin` directory on your remote system.
5. Using the `ftp` command, list the files in the `/usr/include` directory and then copy the file `/usr/include/cpio.h` from the remote system to your user directory on your local system.

Learning Activity Solution: Accessing Remote Systems

To perform these exercises, your system must be connected to a network of UNIX systems using TCP/IP. You must obtain a system name from your system administrator for a remote system that you can log in to and have a user account on that system.

1. Using the `rlogin` command, log in to your remote system.

```
$ rlogin sys002
Password:
Last login: Tue Dec 28 11:04:25 from you@sys001
DEC OSF/1 T2.0-3  (Rev. 176.1); Wed Dec 22 07:59:25 EST 1993
DEC OSF/1 T2.0-3 Worksystem Software (Rev. 176.1)

The installation software has successfully installed your system.

There are log files that contain a record of your installation.
sys002$
```

2. Log out from the system you logged in to in the previous step. Issue the `rcp` command to copy the following file from your remote system to your user directory on your local system: `/usr/include/stdio.h`

```
sys002$ Ctrl/D
$ rcp sys002:/usr/include/stdio.h stdio.h
$ ls -l stdio.h
-r--r--r--  1 evant       11739 Feb 14 14:30 stdio.h
```

3. Issue the `rcp` command as in the previous example, but preserve the modification times and modes of the source file.

```
$ rcp -p sys002:/usr/include/stdio.h stdio.h
$ ls -l stdio.h
-r--r--r--  1 evant       11739 Jan  6 10:15 stdio.h
```

4. Issue the `rsh` command to list the `/bin` directory on your remote system.

```
$ rsh sys002 ls /bin
Mail
Rsh
X11
.
.
.
```

5. Using the `ftp` command, list the files in the `/usr/include` directory and then copy the file `/usr/include/cpio.h` from the remote system to your user directory on your local system.

```
$ ftp
ftp> open sys002
Connected to sys002.
220 sys002 FTP server (DEC OSF/1 X2.0-8 (Rev. 201.1) ready.
Name (sys002:evant): evant
331 Password required for evant.
Password:
230 User evant logged in.
ftp> ls /usr/include
200 PORT command successful.
150 Opening ASCII mode data connection for file list (13.35.0.2).
/usr/include/cmplrs
/usr/include/X11
/usr/include/tli
.
.
.
226 Transfer complete.
remote: /usr/include
5634 bytes received in 0.45 seconds (12 Kbytes/s)
ftp> get /usr/include/cpio.h  mycpio.h
200 PORT command successful.
150 Opening ASCII mode data connection for /usr/include/cpio.h (13.35.0.68,1041)
 (2790 bytes).
226 Transfer complete.
local: mycpio.h remote: /usr/include/cpio.h
2872 bytes received in 0.02 seconds (1.4e+02 Kbytes/s)
ftp> close
221 Goodbye.
ftp> Ctrl/D
```

Invoking Remote Motif Applications

Overview

Sometimes you may want to execute a Motif application on another system, but display it on the screen of your workstation. This can be done using the same commands that you use to invoke terminal-based applications on another system.

Network Security

In addition to the ordinary network security customized with the `/etc/hosts.equiv` and `$HOME/.rhosts` files, there is an additional security check made before a remote Motif display can be made on your workstation.

You must set up your account on the remote system to grant permission to display on any other system that you want it to display on.

- To see the list of authorized display clients, log in to the remote system and give the command: `/usr/bin/X11/xhost`
- To add display clients, you may use the `xhost` command with allowed host or user names as parameters, but only from a terminal logged in directly to the remote system. You cannot add permission by remote login.
- You can also add display clients directly from the console of the remote system. Log in to the account you want to modify, and choose Security... from the Customize menu of the Session Manager.
- Format is very important: check with your network administrator for the correct commands for your system if you have problems.

Specifying the Display

Many Motif applications provide a `-d` or `-display` option to specify where the output should be displayed. If this option is not specified, the value of the `DISPLAY` environment variable is used.

To set and export the display with the Korn shell, the format is:

```
export DISPLAY=system:screen
```

where:

- `system` is the system name to display on
- `:` indicates that TCP/IP should be used for the connection
- `screen` is the screen on that system to display on, usually 0.0

The version of Motif on your system must be compatible with that on the remote system that you are redirecting the display from.

Invoking Applications on a Remote System

If you have a terminal session running on the remote system, or after you open one with the `rlogin` or `telnet` command, just invoke the Motif application from the command line.

- On the DEC OSF/1 operating system, many Motif applications are stored in the `/usr/bin/X11` directory. If this directory is not in your command search path, you must use the full pathname. You can use the `whereis` command to find out where a program is stored.
- Either set and export your `DISPLAY` variable, or use the display option of the application to redirect the display to your system. You can get the correct option format from the `man` page for the application.
- You may want to use the `&` operator to run the Motif application in the background, so you can use your terminal session for other work.

```
$ rlogin sys002 -l you
Password:
Last login: Mon Mar  7 12:12:20 from sys999

DEC OSF/1 X2.0-8  (Rev. 201.1); Wed Jan 26 13:13:30 EST 1994
DEC OSF/1 X2.0-8 Worksystem Software (Rev. 201)

The installation software has successfully installed your system.

There are logfiles that contain a record of your installation.

sys002$ /usr/bin/X11/dxbook -d sys999:0.0 &
[1]     5731
sys002$
sys002$ export DISPLAY=sys999:0
sys002$ /usr/bin/X11/dxcalendar
```

Invoking a Remote Application with rsh

If you want to invoke only one application on the remote system, you can use the `rsh` command.

```
$ rsh sys002 -l you  /usr/bin/X11/dxbook -d sys999:0
```

Learning Activity: Invoking Remote Motif Applications

To perform these exercises, your system must be connected to a network of UNIX systems using TCP/IP. You must obtain a system name from your system administrator for a remote system that you can log in to and you must have a user account on that system. You must use these commands from a workstation with a Motif display capability.

1. Use a remote login to invoke remote applications:
   a. Using the `rlogin` or `telnet` command, log in to your remote system.
   b. Use the `whereis` command to get the full path for `dxcalendar`, the Calendar application. Invoke it in the background, using the `display` option to direct the display to your workstation. When the Calendar window appears, use it to check a few dates.

c. Set your display variable to your workstation, and export it.

d. Issue the command to invoke Bookreader (dxbook) in the foreground, without a display parameter. Try a few sample Bookreader operations when the window appears.

e. Use Exit from the File menus to close Calendar and Bookreader. Log out from the remote system.

2. Issue the `rsh` command to invoke Bookreader on your remote system. Perform some Bookreader operations, then use Exit from the File menu to close Bookreader.

Learning Activity Solution: Invoking Remote Motif Applications

To perform these exercises, your system must be connected to a network of UNIX systems using TCP/IP. You must obtain a system name from your system administrator for a remote system that you can log in to and you must have a user account on that system. You must use these commands from a workstation with a Motif display capability.

1. Use a remote login to invoke remote applications:

   a. Using the `rlogin` or `telnet` command, log in to your remote system.

```
$ rlogin sys002
Password:
Last login: Tue Dec 28 11:04:25 from you@sys001
DEC OSF/1 T2.0-3  (Rev. 176.1); Wed Dec 22 07:59:25 EST 1993
DEC OSF/1 T2.0-3 Worksystem Software (Rev. 176.1)

The installation software has successfully installed your system.

There are log files that contain a record of your installation.
sys002$
```

   b. Use the `whereis` command to get the full path for `dxcalendar`, the Calendar application. Invoke it in the background, using the `display` option to direct the display to your workstation. When the Calendar window appears, use it to check a few dates.

```
sys002$ whereis dxcalendar
dxcalendar: /usr/bin/X11/dxcalendar /usr/share/man/man1/dxcalendar.1X
sys002$ /usr/bin/X11/dxcalendar -display sys001:0.0 &
[1]     9573
sys002$
```

   c. Set your display variable to your workstation, and export it.

```
sys002$ export DISPLAY=sys001:0
sys002$
```

   d. Issue the command to invoke Bookreader (dxbook) in the foreground, without a display parameter. Try a few sample Bookreader operations when the window appears.

```
sys002$ /usr/bin/X11/dxbook
```

e. Use Exit from the File menus to close Calendar and Bookreader. Log out from the remote system.

```
sys002$ Ctrl/D You have running jobs.
[1] + Done                    /usr/bin/X11/dxcalendar -display sys001:0.0 &
sys002$ Ctrl/D
Connection closed.
$
```

2. Issue the `rsh` command to invoke Bookreader on your remote system. Perform some Bookreader operations, then use Exit from the File menu to close Bookreader.

```
$ rsh sys002 -l you  /usr/bin/X11/dxbook -display sys001:0
$
```

Summary

Accessing Remote Systems

Table 15–6 Remote Access Commands

Command	Function
`rlogin`	Logs in to a remote system
`rsh`	Issues commands on a remote system
`rcp`	Copies files between systems without having to log in
`ftp`	Transfers files from one system to another using the File Transfer Protocol
`telnet`	Connects to a remote system using the TELNET protocol

Invoking Remote Motif Applications

To invoke a Motif application on another system and display it on your system:

- Network security must be set up to allow you to access the remote system, and to display on your system.
- You must specify that the display will be on your system by giving the `-display` parameter to the application, or by setting the Korn shell `DISPLAY` variable.
- You can either `rlogin` to the remote system and invoke the application, or use `rsh` to invoke the application.

16
Printing

Printing

Introduction

Although there have been significant advances in electronic transmission, storage, and retrieval of information, there are still many times when hardcopy documents are preferred. Therefore, computer users must know how to utilize hardcopy printers.

Objectives

To work with printers, you should be able to:

- Determine the names of available printers
- Print files
- Display the status of print jobs
- Delete print jobs

Resources

- *OSF/1 User's Guide*

Printing Overview

Printer Commands

The printer commands covered in this chapter include those shown in the following table.

Table 16–1 Printer Commands

Command	Function
`lpd`	Uses system calls to handle print requests (printer daemon)
`pac`	Provides printer accounting information
`lp`	Sends a print request
`lpr`	Sends a print request
`pr`	Formats and writes a file to standard output
`lpq`	Examines the print spool queue
`lpstat`	Displays printer status information
`lprm`	Removes requests from the print spool queue
`cancel`	Cancels a print request
`lpc`	Controls the operation of the printer system

These commands are all available on DEC OSF/1 operating systems. If you are using a different UNIX system, check the `man` pages on your system for the availability of a particular print command.

DEC OSF/1 operating systems have both System V and BSD commands, so there is more than one command available to perform each printer function. You should learn those commands that will be most useful to you.

Determining Available Printers

Before you can print any files, you must know what printers are available for your use. Available printers can be determined by examining the file `/etc/printcap`, a part of which appears in the example.

Example 16–1 Examining /etc/printcap to Determine Available Printers

```
$ more /etc/printcap
# LN03 in the office area      ❶
lp0|lp|0|officepr:\   ❷
        :lp=:\
        :rm=sys01.zko.dec.com:\
        :rp=1:\
        :sd=/usr/spool/lpd:   ❸
    .
    .
    .
# LN03 in the lab next to the line printer
lp1|1|ansio|labpr:\
        :lp=:\
        :rm=sys01.zko.dec.com:\
        :rp=2:\
        :sd=/usr/spool/lpd1:
    .
    .
    .
$
```

❶ The lines beginning with a pound sign (#) are comments.

❷ The first field in a printer definition gives the printer's names. The different names are separated by the vertical bar (|).

❸ The spool directory for the printer is shown.

Note

If there is no /etc/printcap file on your system, use the lpstat -t command described later.

The Printer Daemon, lpd

The lpd program is the printer spool area handler (daemon) which receives requests from other printer commands.

The lpd daemon is normally started during system startup and runs in the background. It reads the /etc/printcap file to determine the capabilities of existing printers.

The lpd daemon:

- Transfers files to the spooling area
- Prints files in the print queue
- Displays the queue
- Removes jobs from the queue

If the daemon is not running, you can restart it using the /usr/lbin/lpd command.

Using pac to Get Printer Accounting Information

The average user has little need for printer accounting information, but you may want to know how much paper you have used for your print jobs.

The `pac` command reads the printer accounting files in `/var/adm` and provides an account of:

- The number of pages (or length of paper in feet for laser printers, typesetters, and plotters) a user has printed
- The number of times a printer has been used
- The cost of printing

The format of the `pac` command is:

```
/usr/sbin/pac [-Pprinter]
```

Submitting Print Requests

The lp Command

The `lp` command sends the specified files to a printer for printing.

The format of the `lp` command is:

```
lp [options] [files]
```

If you do not provide a file name to the `lp` command, or use the `-` option, input is from standard input. Any information you type at your terminal is printed on a system printer. When you have completed typing the information, press Ctrl/D on an empty line to send the information to the selected printer.

lp Command Options

Some of the most frequently used options for the `lp` command are shown in the table. You can use the `man lp` command to look at other available `lp` options.

Table 16–2 Some lp Options

Option	Function
`-`	Reads input from standard input.
`-c`	Makes a copy of the file to be printed. Normally, files are not copied, but symbolically linked. If the `-c` option is not used, do not delete the file or modify the file until it has finished printing.
`-d printer`	Specifies the printer to do the printing. If `-d` is not specified, the default, in order is: • `LPDEST` environment variable • `PRINTER` environment variable • The default printer
`-m`	Sends mail to the user after the files are printed.
`-n number`	Specifies the number of copies you want to print.
`-t title`	Prints `title` on the banner page.

Using the lp Command

To print two copies of the `phones` file from your working directory, on the default printer, using "This is the Phones file" as a title, and to receive mail when the file is printed, use the command:

```
lp -n2 -t "This is the Phones file" -m phones
```

A file to be printed is placed in the spool directory shown in `/etc/printcap` for that printer, often `/var/spool/lpd`*[printer#]*.

The lpr Command

The `lpr` command places files in the spooling area used by the `lpd` daemon. The files are printed when the printer is available. If a file name is not provided, input is taken from standard input.

The format of the `lpr` command is:

```
lpr [options] [files]
```

lpr Command Options

Some of the `lpr` options are provided in the table.

Table 16–3 Some lpr Options

Option	Function
-	Reads input from standard input.
-i[number]	Indents the output by `number` of spaces. If `-i` is used without `[number]`, the default is 8.
-j	Displays the printer request ID on standard output after the job is entered in the queue.
-J name	Provides `name` on the burst page.
-m	Sends mail when the spooling is complete.
-Pprinter	Prints to the specified `printer`.
-p	Uses `pr` to format the files.
-r	Removes file upon completion of spooling, or completion of printing if the `-s` option is used.
-s	Generates a symbolic link to the file to be printed, rather than copying it to the spool directory. If this option is used, the file should not be modified or removed until printing has completed.
-#number	Produces `number` of copies.

Using the lpr Command

The command to print the file, `myfile` to printer `lp1`, with the banner "MY JOB" printed instead of the file name on the burst page, with a five-space indentation, and with mail being used to provide notification is:

```
lpr -Plp1 -J "MY JOB" -i5 -m myfile
```

The pr Command

The `pr` command is used to format text pages. It paginates files, adds an optional header to each page, and writes to standard output. The header includes the file name, time of last modification, and page number.

The format of the `pr` command is:

```
pr [-option] [file]
```

pr Command Options

Some of the options you may want to use are shown in the table. Other options can be found using `man pr`.

Table 16–4 Some pr Options

Option	Function
`-d`	Changes the output to double space.
`-f`	Uses the formfeed character to advance to a new page instead of a series of line feeds. When output is to the terminal, output is paused at the beginning of the first page. You must press Return to see the first page.
`-h "string"`	Replaces the file name in the header with `"string"`. The `-h` and `"string"` must be separated by a space.
`-l number`	Changes the length of the page from the default of 66 to `number`.
`-p`	Pauses before beginning each page, even the first page, if output is to the terminal. You must press Return to view the next page.
`-w number`	Sets the width of the lines to `number` bytes. The default is 72.
`-`	Reads input from standard input; terminate by pressing Ctrl/D.

Using the pr Command

To use `pr` to provide a page header `"This is a Header"`, change the page length to 44 lines, and print the file, `myfile`, on the default printer, you would use the following commmand:

```
pr -h "This is a Header" -l 44 myfile | lp
```

Controlling Print Queues

Determining Print Status with lpq

After you execute a command to print a file, you may need to know the status of the print request. The `lpq` command examines the spooling area used for printing files, and reports the status of the specified requests or all requests for a user.

The `lpq` command provides:

- Current rank in the queue
- Names of files comprising the request
- Request identification number
- Total size in bytes

If the print request consists of more than one file, the number of file names shown in the `lpq` output for the request depends upon the length of the file name.

The format of the `lpq` command is:

```
lpq [+[number]] [-l] [-Pprinter] [request_id...] [user...]
```

The `lpq` command invoked without any arguments provides reports on any requests currently in the queue for the default printer.

lpq Command Options

This table provides information about the use of arguments with the `lpq` command. Spend a few minutes studying each of the various arguments that you can use with the `lpq` command.

Table 16–5 Some lpq Options and Arguments

Option/ Argument	Function
[none]	Reports on all print jobs for the default printer.
`+number`	If the + argument is used, `lpq` reports the status of the spool queue every second until the queue is empty. If `number` is provided, `lpq` displays the queue every `number` of seconds.
`-l`	Causes additional information about each entry to be shown.
`-Pprinter`	Reports the status of print requests for the printer designated by `printer`.
`request_id`	Reports the status of the print job specified by `request_id`.

(continued on next page)

Table 16–5 (Cont.) Some lpq Options and Arguments

Option/ Argument	Function
user	Provides a report of all print jobs queued on the default printer for user user.

Using the lpq Command

The following example shows the output from the lpq command.

Example 16–2 lpq Output

```
$ lpq
                    ❶
Rank   Owner      Job  Files                                 Total Size
active jones       15  pac-man                               1833          ❷
1st    smith       42  pgm-10-15-01.c                        1926 bytes
2nd    smith       43  pgm-10-15-02.c                        3930 bytes
3rd    smith       44  pgm-11-26-b.c                         1656 bytes
4th    you         49  temp, temp1, temp2, temp3, temp4 ...  834 bytes     ❸
```

❶ The job number is important if you want to remove one particular print request from the queue.

❷ The file currently being printed shows up as **active** in the column showing the rank.

❸ In addition to the files temp, temp1, temp2, temp3, and temp4 on the status line for print job 49, it shows an ellipsis, an indication that there are more files to be printed with this job.

To determine the names of all files to be printed as part of job 49, use the lpq -l command to obtain a detailed print status.

Determining Print Status with lpstat

The lpstat command provides another way to get printer status. It provides more information than the lpq command.

The format of the lpstat command is:

```
lpstat [options]
```

This table lists the functions for various options you can use with the lpstat command. You will probably find the lpstat -t command the most useful.

Table 16–6 lpstat Options

Option	Function
-a[printers]	Indicates whether that particular printer queue is enabled to accept print requests. With no list provided, information for all printers is shown.
-d	Provides the system default destination for `lp` or `lpr` commands.
-o[printers]	Provides a display of print requests for all printers with no list of `printers` on the command line. If a list of printers is provided, only the print requests for the specified printers is shown. The output format is the same as the `lpq` command.
-p[printers]	Provides a display of printer status for selected printers if a list of `printers` is supplied. Otherwise, the status for all printers is shown. Information is provided to indicate: queuing is enabled, printing is enabled, the number of entries in the queue, and the presence of the printer daemon.
-r	Provides the status of the printer daemon for the default printer. It is referred to as the scheduler in the report.
-s	Provides a status summary, including the status of the printer, daemon (scheduler), the default destination, and a list of printers.
-t	Provides a list of all printer status information with one command, including print requests.
-u[users]	Provides the status of print requests for all users, or those `users` listed.
-v[printers]	Provides a list of the names of `printers` and the pathnames of the devices associated with them.

Note

When a list of printers (`[printers]`) or users (`[users]`) is provided, the list must be a space or comma-separated list of printer names or user names. The list must be enclosed in double quotes ("), and there cannot be a space between the option and the list. For instance, `lpstat -a"lp, lp2"`.

Deleting Print Requests with lprm

Use the lprm command to remove print requests from the print spooler.

The format of the lprm command is:

lprm [-P*printer*] [-] [*request_id...*] [*user...*]

If the lprm command is used without options, the current active request for the user on the default printer is removed from the queue.

This table provides a list of various lprm options and a description of each function.

Table 16–7 lprm Options

Option	Function
[none]	If no options are specified, the current active request for the user on the default printer is removed from the queue.
-Pprinter	Specifies a specific print queue from which to remove print jobs. Further identification of a print job is required.
-	Removes all print jobs from the designated queue for the user. For the superuser, the spool queue is completely emptied.
[request_id]	Removes the print job with the number request_id from the designated print queue.
[user]	Removes all print jobs from the designated print queue for user. If the superuser executes the command, multiple users may be identified.

Using the lprm Command

The command to remove job 49 from the default printer queue is:

```
lprm 49
```

If you use lprm to remove a print job from a queue other than the default queue, the -P option must be used. You must use the [request_id] option (which requires the job number) to remove a single job from the print queue.

Using the cancel Command

You can also use the cancel command to remove print jobs sent by the lp or lpr commands.

The format of the cancel command is:

cancel [request_id] [printer]

- request_id cancels only the print job with the matching job number.
- printer cancels all print jobs for the selected printer for the user. If the superuser issues cancel with the printer option, all requests for that printer are removed from the queue.

Using the Printer Control Program (lpc)

Use the lpc program (/usr/sbin/lpc) to control the operation of the printers on the system.

- If you execute /usr/sbin/lpc without any arguments, it prompts for commands from standard input.
- If arguments are supplied, lpc interprets the first argument as the command to be executed, and the remaining arguments as parameters to the command.

The format of the lpc command is:

```
/usr/sbin/lpc [command] [argument]
```

lpc Commands

Some lpc commands and arguments available to ordinary users are shown in the table.

Table 16–8 lpc User Commands

Command [argument]	Description
help or ?	With no arguments, provides a list of the available commands. With a command name as an argument, provides a short description of command.
restart [all | printer...]	If an abnormal condition has caused the daemon to quit, leaving jobs in the queue, the restart command will attempt to start a new daemon.
status [printer...]	Displays the status of daemons and queues on the local system. The output from this command may also be seen with the lpstat -t command.
exit or quit	Exits lpc interactive mode.

Learning Activity: Printing

Before you start this lab, check with your course administrator about which printer queues to use. If they disable printing for the applicable queues, you can practice queueing and removing files without wasting paper.

Determining Available Printers

Execute the following command to determine printer availability on your system:

```
more /etc/printcap
```

Printing Files with lp

1. Select a file from your directory. Place it on the appropriate print queue with the lp -d printer command. Use the -m and/or -t title options if you choose.

2. Confirm that your print job is on the appropriate queue with the `lpq` command for the default printer, or the `lpq -P` printer command for any other printers.
3. Repeat the previous print command, but this time instead of printing a file, redirect standard input to the printer. Execute the `lp` command without a file name, or with the - option as `lp -`. Type in a short message. Terminate the message by pressing Ctrl/D at the beginning of an empty line. Verify that the print job is on the queue.
4. Repeat the previous step, this time using the `pr` command. Use the `-h` option to provide a name in the header. Add the `-l` number option to set the page length if you want. When you finish typing, terminate by pressing Ctrl/D at the beginning of a line.
5. Repeat the `pr` command, but this time pipe the `pr` command through the `lp` command to print the information from standard input to the line printer.

   Look at the printer queue status. Notice that, in place of the file name, (you did not print a file), it says `standard input`.

Printing Files with lpr

1. Print a file with the `lpr` command, using the `-j` option. Note that the job number is sent to standard ouptut upon execution of the command.
2. Repeat the `lpr` command, but this time try the `-p` option without providing a file name. Notice that you can print from standard input as you did with `lp`. You can also format using `pr`.

Formatting Files with pr

1. Execute the following command to create a file with a few pages of text for use in the next step.

   ```
   man lpc > lpc-man-pages
   ```

2. Now, use the `pr` command to display the `lpc-man-pages` file on your terminal. Add a title with the `-h` option. Change the page length to 40 lines with `-l40`. Make it pause before each page, even the first using `-p`.

   ```
   pr -h "lpc man pages file" -l40 -p lpc-man-pages
   ```

3. Repeat the previous step, and experiment with the other options. Try double spacing, changing the line width, or removing the title completely.

Getting Print Status

1. Execute each of the following commands and compare the outputs:
   - `lpq`
   - `lpq -l`

- `lpstat -a`
- `lpstat -s`
- `lpstat -t`

Notice that `lpstat -t` provides the most information, and looks like a combination of some of the other status outputs.

2. Experiment with some of the other `lpq` and `lpstat` options.

Deleting Print Requests

1. Check the job numbers of print jobs you have queued up. If you do not have any, use `lp` or `lpr` to queue up a few print jobs.

   Use `lprm` to:

   - Remove one print job from the queue using `lprm[request_id]`
   - Remove one print job from the queue using `cancel[request_id]`
   - Remove all of your remaining jobs using `lprm -`

2. Queue up a few more print jobs, then remove all print jobs using `lprm[user]`

Line Printer Control Program (lpc)

1. Run the `lpc` program `/usr/sbin/lpc`.
2. Experiment with the Help facility. Type `help`.
3. Try to execute the commands. Notice that the program does not let you execute any programs that are reserved for the superuser. You can execute the `restart` and `status` commands.

Learning Activity Solution: Printing

No solution is required.

Summary

Printing Overview

- The printer daemon, `lpd` receives requests from `lp`, `lpr`, `lpq`, `lprm`, or `cancel` to transfer files to the spooling area, print files, display the queues, and remove jobs from the queues.
- The `pac` command reads the printer accounting files to provide information about the number of pages each user has printed.

Submitting Print Requests

- Files can be printed using the `lp` and `lpr` commands.
- The `pr` command is used to format text files for printing.

Controlling Print Queues

- You can obtain printer status information with the `lpq` and `lpstat` commands.
- Print jobs are removed from the queue using the `lprm` or `cancel` command.
- The printer control program, `lpc`, can be used to restart printer daemons and display printer status information.

17

Using Tapes and Disks

Tapes and Disks

Introduction

This chapter introduces the various commands used in the UNIX operating system to copy data between disks and magnetic tapes.

The UNIX operating system offers a choice of data transfer and conversion commands. You can save and restore individual files and directories as well as tree-structured file systems to and from disks and magnetic tapes.

The `tar`, `cpio`, `ltf`, `dd` and `mt` commands are used for tape data transfer and conversion. The `df` and `du` commands provide detail about disk space usage.

Device special files and device naming conventions allow you to identify specific devices when entering commands.

This chapter emphasizes how to use the `tar` command to create backup tapes.

Learning activities allow time for you to become more familiar with the `tar`, `mt`, `df`, and `du` commands.

Objectives

To utilize tape and disk resources, you should be able to:

- Identify disks, tapes, and CDROM device types and names
- Use the `tar` command to save and restore tape archives
- Use the `cpio` command to copy files to and from archives
- Use the `ltf` command to read and write ANSI compatible tapes
- Use the `dd` command to read and write tapes from non-UNIX systems
- Use the `mt` command to manipulate tape devices
- Use the `df` command to display free disk space
- Use the `du` command to display size of files in directories

Resources

- *OSF/1 User's Guide*
- *ULTRIX to DEC OSF/1 Migration Guide*
- DEC OSF/1 `man` pages

Identifying Devices

Overview

This topic introduces commands to help you manage disk, tape, and CDROM devices.

Device Special Files

UNIX treats all devices as special files. Device special files are located in the `/dev` directory and are read and written like ordinary files except they activate the I/O device itself.

Examples of device special files are: `/dev/rmt0h` and `/dev/rra0a`.

Two types of device special files are shown in the following table.

Table 17–1 Device Special Files

File Type	Purpose	Example
Block special	Buffered I/O	rz0g (disk partition g on drive 0 of an RZ disk)
		rz1h (disk partition h on drive 1 of an RZ disk)
Character special	Serial or raw devices	rmt0h (tape drive 0, high density)
		rrz0a (raw device, disk partition a, drive 0, CDROM device)

Understanding Tape Device Names

Tape device names can be in the form: n**rmt**yz

where:

n is no rewind (no n is rewind)
y is tape unit number such as 0, 1, 2
z is h (high density), m (medium density), l (low density), or a (only applicable to SCSC quarter-inch cartridge tapes)

The SCSI quarter-inch cartridge (QIC) tapes (TZK10) are fixed-block tape drives, and all reads and writes must be in a multiple of the block size.

Typical device special files are shown in the following table.

Table 17–2 Device Types

Device	Description	Block Size
rmt0a	QIC-24, read only	512 bytes
rmt0l	QIC-120, read only	512 bytes
rmt0m	QIC-150, read only	512 bytes
rmt0h	QIC-320, read only	1024 bytes

Identifying Disk, Tape and CDROM Device Mnemonics

DEC OSF/1 uses the device **mnemonics** (shorthand names) listed in the table to describe hardware devices on your system. The mnemonics are used by the `/dev/MAKEDEV` utility when creating device special files, in the system configuration file definitions, and in the format for system maintenance commands.

Table 17–3 DEC OSF/1 Disk and Tape Device Mnemonics

Category	Mnemonic	Description
Disks	rznn	SCSI disks (RZ22/RZ23/RZ24/RZ55 /RZ56/RZ57/RZ58)
CDROM	rznn	SCSI disks (RRD40)
Tapes	tznn	SCSI tapes (TLZ04/TLZ05/TZ30 /TZK50)

nn indicates a variable

Learning Activity: Identifying Device Special Files

List your system's devices in `/dev` using the `ls /dev` command. Use `ls /dev/tz*` to list only the tape devices.

No solution required.

The Tape Archive Utility (tar)

Overview

This topic explains how to read and write data on magnetic tape using the `tar`, `cpio`, `ltf`, `dd`, and `mt` commands. The most common usage is backing up files from disk to tape, and then restoring the files from the tape back to disk.

Using Tape Media

To write information to a magnetic tape, **write-enable** the tape:

- Reel-to-reel tapes have a **write-ring**, a removable plastic ring that fits into the inner circle of the back side of the tape reel.

  When the ring is in place, the tape is write-enabled. To prevent a tape from being accidentally overwritten, remove this ring.
- Cartridge tapes for TK50 and TK70 drives have a sliding switch. Push the switch to the right (so the orange indicator does not show) to write-enable the tape. To write-protect the tape, push the switch to the left so the orange indicator shows.

Magnetic tape reels are available in different lengths:

182.8 m (600 ft)
365.6 m (1200 ft)
548.4 m (2400 ft)

Digital tape drives write information at one or more of the following densities:

32.0 rows/mm (800 bits/in)
63.0 rows/mm (1600 bits/in)
246.0 rows/mm (6250 bits/in)

For most reel-to-reel tape drives, the default density is 6250 bpi (h). To select low density, 1600 bpi, use the `f` key option followed by `/dev/rmt0l` .

For TK50 tape cartridge drives, there is only one density - 6666 bpi, whether you use `/dev/rmt0l` or `/dev/rmt0h` .

Using the Tape Archive Utility

The `tar` (**t**ape **ar**chive) utility saves and restores multiple files to and from magnetic tape. The `tar` utility only extracts files and directories from tapes originally made with `tar`.

Users can record private copies of their files and directories using the `tar` command.

The format of the `tar` command is:

tar `[key_options]` *name ...*

Where `name` represents either a file or directory. If `name` is a directory, `tar` recursively copies all files and subdirectories in that directory.

tar Command Options

Commonly used `tar` options are described in the following table.

Table 17–4 Commonly Used tar Options

Key[1] Option	Additional Option	Function
`c`		Creates a new tape archive
`r`		Adds files to the end of the tape
`t`		Lists names of files as they occur on the tape
`u`		Adds new or modified files to the tape
`x`		Extracts files from the tape If the file name given is a directory, all files in the directory are also extracted.
	`f`*device*	Uses `device` as the name of the archive instead of the default device `/rmt0h`
	`p`	Restores files to their original permissions, ignoring the present `umask`
	`v`	Displays the name and other detailed (verbose) information about each file handled on your terminal
	`w`	Queries the user before executing each option

[1]The key options are required options. At least one of them must be used.

One key option must appear as the first command-line argument. Additional options can be combined with the key option. For example:

```
tar xv new_stuff
```

Examples of using the `tar` command are provided after procedures for writing/extracting files to/from tape archives.

Writing Files to Tape Archives

The steps for writing files to tape archives are listed in the following table.

Table 17–5 Writing Files to Tape Archives

Step	Procedure
1	Mount and load a blank tape on the tape drive. Ask your system manager for assistance if you are unfamiliar with the tape drive operating procedure. **Make sure the tape is write-enabled.**
2	Log in at the console terminal using your login name.
3	Use the `cd` command to change to the directory that contains the file(s) you want to record.
4	Enter the `tar` command.
5	After the tape stops, rewind it and unload it.
6	Label the tape and write-protect it. Store the tape in a safe place.

Important Points About the tar Utility

The `tar` utility writes data sequentially until it recognizes the end of tape or the end of information.

When adding files, use the `r` option to prevent overwriting existing files.

You must have read-access permission to extract tape files; therefore do **not** use the `p` option when adding files to a tape.

Caution

Do not use the `tar` command to save and restore symbolic links or directories that contain symbolic links. Corruption of the "linked to" file name can occur if it is used in this manner.

Extracting Files from Tape Archives

The steps for extracting files from tape archives are listed in the following table.

Table 17–6 Extracting Files from Tape Archives

Step	Procedure
1	Mount and load the tape containing your files on the tape drive. Ask your system manager for assistance if you are unfamiliar with the tape drive operating procedure. **Make sure the tape is write-protected.**
2	Log in at the console terminal using your login name.
3	Use the `cd` command to change to the directory where you want to place the file(s) extracted from the tape.

(continued on next page)

Table 17–6 (Cont.) Extracting Files from Tape Archives

Step	Procedure
4	Enter the `tar -x` command.
5	After the tape stops, rewind it and unload it.
6	Use the `ls` command as a final check to verify that the file is in your directory.

Important Points About Using Tapes

Be sure to label (using a paper label) private tapes with the following information:

- Contents
- Dates recorded
- Tape density
- Variation of `tar` command used

You need read-access permission to extract tape files. Users cannot access an extracted `tar` file for which they do not have read permission. To avoid this problem, do not use the `p` option when using `tar` to record files.

To prevent accidental tape reuse, provide blank or recyclable tapes and write-protect rings in a storage area near the tape drive.

All TK50 tapes are considered hardware write-protected by TK70 drives, and therefore are read-only media when used in a TK70 drive.

Using tar for Disk-to-Tape Backup Operations

This example uses the `tar` command to create and list files from tapes.

Example 17–1 Using tar for Disk-to-Tape Operations

```
$ tar cv filename ❶
$ tar t ❷
$ tar tv ❸
$ tar ruv filename ❹
$ tar cvf file.tar filename ❺
```

❶ The `tar` utility creates a new tape using the files designated by `filename`. The names of the files written to tape are displayed on standard output. If `filename` is dot (.) or asterisk (*), all files in the current directory and subdirectories are written to tape. If `filename` is one file name or a list of file names, only those files are written to tape.

❷ The `t` option lists the file names on the tape.

❸ The tv option provides extra (verbose) information about the tape entries other than just the name.

❹ The ruv option appends filename to the end of the tape. The names of the files appended are displayed on standard output.

❺ The cvf option creates the tar file, file.tar on a disk. It contains the files designated by filename. As with tape creation, if filename is dot (.) or asterisk (*), all files in the current directory and subdirectories are written to tape. If filename is one file name or a list of file names, only those files specified are written to tape.

Using tar for Tape-to-Disk Restore Operations

This example uses the tar utility to extract files from tapes to disks, and to manage tar files on disks.

Example 17–2 Using tar for Tape-to-Disk Operations

```
$ tar xv filename ❶
$ tar tvf file.tar ❷
$ tar xvf file.tar filename ❸
$ compress file.tar ❹
$ uncompress file.tar.Z ❺
```

❶ The xv option causes the extraction of files from the tape. If filename is not provided, the entire contents of the tape is extracted. If filename matches a directory whose contents were written onto the tape, this directory is recursively extracted.

❷ Display a verbose list of the files contained in the tar file, file.tar.

❸ Extract the files represented by filename from the tar file, file.tar. If filename is not provided, the entire contents of the tape is extracted. If filename matches a directory, the directory is recursively extracted.

❹ Use compress to reduce the size of the tar file. This is especially useful when tar files must be copied over a network. The new file has the same file name as the tar file with .Z appended to the name (file.tar.Z).

❺ Use uncompress to convert the compressed tar file to the uncompressed version after copying a compressed version of the file over the network. The uncompressed file has the same file name without the Z (file.tar).

Learning Activity: Using the Tape Archive Utility

1. Mount a blank tape on a tape drive. Log in using your login name. Choose a file in your home directory and write it to tape using the command **tar cv filename**. Substitute your actual file name for `filename`. (The `c` option forces `tar` to start writing at the beginning of the tape. The `v` option tells `tar` to display the files as it processes them.)
2. Verify that the file exists on tape.
3. Set the tape to the load point again (by rewinding it). Create a subdirectory named `tardir` and change directory to it.
4. Extract the file you just placed on tape by using the command **tar xv filename**. Substitute your actual file name for `filename`. (The `x` option causes `tar` to restore your file from the tape to your `tardir` directory.)
5. When `tar` is finished, display the files in your `tardir` directory to ensure that the extracted file(s) are there.

Learning Activity Solution: Using the Tape Archive Utility

1. Mount a blank tape on a tape drive. Log in using your login name. Choose a file in your home directory and write it to tape using the command **tar -cv filename**. Substitute your actual file name for `filename`. (The `c` option forces `tar` to start writing at the beginning of the tape. The `v` option tells `tar` to display the files as it processes them.)

   If your file is named `stuff`, the command is:

   ```
   $ tar -cv stuff
   a   stuff    20 blocks
   ```

2. Verify that the file is actually on tape.

   ```
   $ tar t
   stuff
   ```

3. Set the tape to the load point again. Create a subdirectory named `tardir` and change directory to it.

   ```
   $ mkdir tardir; cd tardir
   ```

4. Extract the file you just placed on tape by using the command **tar xv filename**. Substitute your actual file name for `filename`. (The `x` option causes `tar` to restore your file from the tape to your `tardir` directory.)

   If your file is named `stuff`, the command is:

   ```
   $ tar -xv stuff
   x  stuff,  234 bytes, 1 blocks
   ```

5. When `tar` is finished, display the files in your `tardir` directory to ensure that the extracted file is there.

   ```
   $ ls
   stuff
   ```

Other Useful Commands

Overview

Although tar is the most common utility used with tapes, there are several other commands with similar functions, as shown in the following table.

Table 17–7 Commands Used with Tapes

Command	Function
cpio	Copies files to and from archive storage
ltf	Reads and writes tapes for and from other operating systems such as OpenVMS
dd	Reads, writes, and converts tapes from non-UNIX operating systems
mt	Manipulates tapes, moving them forward or backward by a specified number of records

Using the cpio Utility

The cpio (copy file archives in and out) command copies files to and from archive storage. It saves and restores data from traditional format cpio archives. cpio copies a subtree of the file system (directories, and links) to and from archive storage, or another place in the file system.

The cpio command uses the three basic formats shown in the table.

Table 17–8 cpio Command Formats

Command	Description	Function
cpio -i	Copy in	Copies files from cpio archives into the current directory tree
cpio -o	Copy out	Reads pathnames from standard input and copies these files to standard output
cpio -p	Directory copy	Copies files into an already-existing named directory

The cpio utility is especially suited to copy file system subtrees with software links that the dd command cannot.

The format of the three cpio command formats are:

cpio -o[aBcv] [-C*value*] [-M"*string*"] *file_name* ... [-O*device*]

cpio -i[bBcdfmrsStuv6] [-C*value*] [-M"*string*"] [-I*device*] *pattern* ...

cpio -p[adlmruv] *directory* ...

cpio Command Options

This table lists the most commonly used `cpio` options.

Table 17–9 Commonly Used cpio Options

Key[1] Option	Additional Option	Function
`-i`		Copies files that match a specified pattern, for example, `filename*`
`-I`*device*		Specifies the input device containing the archive (required to import data)
`-o` *filename*		Copies the specified files
`-O`*device*		Specifies the output device you will use to store the archive (required to export data)
`-p`		Copies files into the specified directory
`c`		Writes header information in ASCII character form
		Specify this flag when Posix compliance is required and when you are creating or restoring archives for or from another system.
	`d`	Creates directories as needed
	`f`	Copies all files **except** those that match the specified pattern
	`k`	Enables symbolic link handling
	`M`	Specifies the end-of-media message
	`t`	Prints a table of contents of the input (no files are created)
	`u`	Copies files unconditionally
	`v`	Displays detailed (verbose) information during the file copy and file create operations

[1]The key options are required options. At least one of them must be used.

Place a hyphen before the key options. The `-I` and `-O` options must be separated (as in, `-I -O`). If additional options are used, they must follow the key options, with no intervening spaces. Hyphenated (-) key options can be combined with other options. For example, this command copies the file `new_stuff` out to the default tape drive using standard ASCII character form:

```
cpio -ocv new_stuff -O/dev/rmt0
```

Using the cpio Command

This example demonstrates various uses of the `cpio` command.

Example 17–3 Using the cpio Command

```
$ cpio -ov < baseball -O/dev/rmt0h  ❶
$ cpio -itv -I/dev/rmt0h  ❷
$ cpio -iv -I/dev/rmt0h  ❸
$ ls *.c | cpio -ov -O/dev/rmt0h  ❹
$ find . -print | cpio -ov -O/dev/rmt0h  ❺
```

❶ Copy the file `baseball` to tape drive rmt0h.

❷ Display the full (verbose) table of contents of the data previously saved onto `/dev/rmt0h` in `cpio` format. To list only the pathnames, use the `-it` options.

❸ Copy the files previously saved on `/dev/rmt0h` back into the current directory. If there are saved directories, you can use the `d` option to allow the creation of directories as needed.

❹ Copy all files in the current directory that end with `.c` onto magnetic tape.

❺ Copy the current directory and all subdirectories onto magnetic tape.

Using ltf to Create ANSI Standard Tapes

The `ltf` command (an acronym for the **l**abeled **t**ape **f**acility) writes and reads single-volume tapes using formats that are compatible with OpenVMS and other operating systems whose tape formats comply with ANSI standards Versions 3 and 4.

Use `ltf` to move files between OSF/1 and OpenVMS operating systems. Tapes written by the OpenVMS **COPY** command can be read using `ltf` and visa versa.

The `ltf` command:

- Does not support floppy diskettes.
- Handles multivolume save sets, but not multivolume tapes.
- Is much slower than `tar` and should not be used in its place.
- Is not available on all UNIX systems.
- Allows you to read and write single-volume tapes.

The `ltf` command has the format:

ltf [key_options] *name ...*

The `name` can be file name or directory. If you specify a directory name, `ltf` processes the complete directory tree.

ltf Command Options

The following table lists some commonly used `ltf` options.

Table 17–10 Commonly Used ltf Options

Key[1] Option	Other Option	Function
`-c`		Creates a new tape volume
`-H`		Displays help messages
`-t`		Lists files on the tape volume
`-x`		Extracts each named file from the tape to the current directory
	`a`	Outputs to an ANSI-compatible Version 3 format tape; can only be used with the `-c` option (Default is Version 4)
	`f`*device*	Selects drive 0-31 (Default is drive `/dev/rmt0h`)
	`P`*position*	Specifies file sequence and section number at which volume will be positioned; necessary for multivolume format
	`v`	Displays more information about the volume (verbose)
	`V`	Displays verbose information about directories

[1]The key options are required options. At least one of them must be used.

One hyphenated (-) key option must appear as the first command-line argument.

Key options can be combined with additional options. For example, this command copies the file `new_stuff` to a new tape volume on the `/dev/rmt2` device:

```
ltf -cf/dev/rmt2 new_stuff
```

Using the ltf Command

This example shows how to use the `ltf` command.

Example 17–4 Using the ltf Command

```
$ ltf -cf /dev/rmt0h /usr/users/you ❶
$ ltf -t  ❷
$ ltf -x  ❸
$ ltf -H  ❹
$ ltf -tv ❺
```

❶ Copy all files in the directory and subdirectories of `usr/users/you` to `/dev/rmt0h`. Because this is the default, the `-f` option is not really needed.

❷ List the contents of the tape.

❸ Copy the entire volume from the tape into the current directory.

❹ Display a short help message listing the available command options.

❺ Provide a verbose display of the tape contents.

Using dd to Convert and Copy a Tape File

The Convert and Copy a File command (`dd`) is useful for reading and writing tapes for use with non-UNIX processors. The `dd` command can move files between OSF/1 and OpenVMS operating systems.

You can use `dd` to read or write a single file, or append multiple input files to a single output file with the `files=` option.

Compared to the `cpio` command, `dd` is especially suited to I/O on the raw physical devices because it allows reading and writing in arbitrary record sizes. `dd` also reports the number of full (f) and partial (p) blocks copied.

The `dd` command has the following format:

```
dd [option=value ...]
```

dd Command Options

The following table describes commonly used `dd` options.

Table 17–11 Commonly Used dd Options

Option	Function
if=*input file_name*	Specifies the input file_name. The default is standard input.
of=*output file_name*	Specifies the output file_name. The default is standard output.
ibs=*number*	Specifies the input block size in bytes. The default is 512 bytes.
obs=*number*	Specifies the output block size in bytes. The default is 512 bytes.
bs=*number*	Specifies both the input and output block size, superseding ibs and obs.
cbs=*number*	Specifies the conversion buffer size.
skip=*number*	Skips *number* input records before starting the copy.
files=*number*	Copies *number* input files before terminating.
seek=*number*	Seeks to the *number*th record from the beginning of output file before copying.
count=*number*	Copies only *number* input records.
conv=*specification [,specification ...]*	Specifies one or more of the following conversions: ascii—Converts EBCDIC to ASCII block—Converts variable-length records to fixed length ebcdic—Converts ASCII to EBCDIC ibm—Performs a slightly different map of ASCII to EBCDIC unblock— Converts fixed-length records to variable length lcase—Converts all letters to lowercase ucase— Converts all letters to uppercase swab— Swaps every pair of bytes noerror—Does not stop processing on an error sync— Pads every input record to ibs ... , ... Allows several comma-separated conversions

Options must be separated with spaces. For example, the following command converts and copies the contents of the default tape device to the file x. The contents of the tape are converted to lowercase ASCII characters.

```
dd if=/dev/rmt0 of=x ibs=800 cbs=80 conv=ascii,lcase
```

Using mt to Manipulate Magnetic Tape Devices

Use the mt command to move tapes forward and backward a specified number of files or records. You can also use it to command tape devices to rewind, go off line, or report the device status.

Use mt in conjunction with the dd command to position the tape while extracting files.

The mt command returns a 0 exit status when the operations are successful, a 1 if the command is unrecognized, and a 2 if the operation fails. The mt command does not support Japanese language.

The mt command has the following format:

mt [-f *device*] *command* [*count*]

Use the -f flag to specify the tape device. The device must be a raw (not block) tape device. If you do not specify a device, the default device is the value of the **TAPE** variable. If **TAPE** does not exist, the default is /dev/rmt12.

If a count is not given, the default count is one.

Common mt Commands

The following table lists the commonly used mt commands.

Table 17–12 Commonly Used mt Commands

Command	Function
eof, weof	Writes count End-of-File marks at the current position on the tape
fsf	Spaces forward by count files
fsr	Spaces forward by count records
bsf	Backspaces by count files
bsr	Backspaces by count records
rewind	Rewinds the tape (count is ignored)
offline, rewoffl	Rewinds the tape and place the tape unit off line (count is ignored)
status	Prints status information about the tape unit

Using the mt Command

This example shows how to use the mt command.

Example 17–5 Using the mt Command

```
$ mt fsf 2 -f /dev/rmt0h    ❶
$ mt rewind -f /dev/rmt0h   ❷
$ mt offline -f /dev/rmt0h  ❸
$ mt status                 ❹
```

❶ Move the tape mounted on /dev/rmt0h forward two files.

❷ Rewind the tape.

❸ Rewind the tape and place it off line.

❹ Display a status of the tape drive.
Example 17–6 shows a typical status.

Example 17–6 Tape Status

```
$ mt status -f /dev/rmt0h
DEVIOGET ELEMENT        CONTENTS
---------------         --------
category                DEV_TAPE
bus                     DEV_SCSI
interface               SCSI
device                  TK50
adpt_num                0
nexus_num               0
bus_num                 0
ctlr_num                0
slave_num               5
dev_name                tz
unit_num                5
soft_count              U
hard_count              U
stat                    200
                        DEV_DONE
category_stat           80
                        DEV_6666BPI

MTIOCGET ELEMENT        CONTENTS
----------------        --------
mt_type                 MT_ISSCSI
mt_dsreg                200
mt_erreg                0
mt_resid                0
```

Learning Activity: Using mt to Manipulate Tape Devices

1. Load a write-protected tape on the tape drive. Do not place the tape drive on line. Use the `mt` command to display the status of the tape drive. Verify that its status is DEV_OFFLINE DEV_WRTLOK DEV_DONE.
2. Place the tape drive on line. Repeat the `mt` command to verify the tape drive is now on line.
3. Use the `mt` command to move the tape forward 20 files.
4. When the tape stops, use the `mt` command to rewind the tape and place the tape drive off line.

Learning Activity Solution: Using mt to Manipulate Tape Devices

1. Load a write-protected tape on the tape drive. Do not place the tape drive on line. Use the `mt` command to display the status of the tape drive. Verify that its status is DEV_OFFLINE DEV_WRTLOK DEV_DONE.

   ```
   $ mt -f /dev/rmt0h status
   ```

2. Place the tape drive on line. Repeat the `mt` command to verify the tape drive is now on line.

   ```
   $ mt -f /dev/rmt0h status
   ```

3. Use the `mt` command to move the tape forward 20 files.

   ```
   $ mt -f /dev/rmt0h fsf 20
   ```

4. When the tape stops, use the `mt` command to rewind the tape and place the tape drive off line.

   ```
   $ mt -f /dev/rmt0h rewoffl
   ```

Using Disks

Overview

This topic introduces the `df` and `du` commands for use in displaying free disk space.

- `df`—Displays free disk space.
- `du`—Displays the disk space used by files and directories.

Using df to Display Amount of Free Disk Space

Use the `df` command to display the amount of free disk space on a file system or on the file system where the file exits.

If you do not specify either a file or a file system, `df` displays statistics for all mounted file systems.

If you specify a device that has no file system mounted on it, `df` displays the information for the root file system.

You can use the `df` command to give commands to disk devices with the following format:

`df [-ikn] [-t` *`type`*`] [`*`file ...`*`] [`*`file system ...`*`]`

df Command Options

The following table shows some commonly used `df` options.

Table 17–13 Some df Options

Option	Function
`-e`	Displays information about the mounted file systems.
`-i`	Reports the number of free inodes. The number of inodes controls the number of files that can exist in a file system.
`-k`	Displays numbers in kilobytes. Default is in blocks (512-byte)
`-t`*`type`*	Displays statistics for the specified file system `type`. Types are: cdfs—ISO 9660 CDROM file system mfs—Memory File System nfs—Network File System pc—Xenix sysv—System V File System ufs—UNIX File System (Berkeley Fast File System) (Default)

Using the df Command

This example shows the type of data provided by the `df` command.

Example 17–7 Using the df Command

```
$ df ❶
Filesystem            512-blocks     Used   Avail Capacity  Mounted on
/dev/rz0a                  38750    25140    9734    72%    /
/dev/rz0g                 549328   450738   43656    91%    /usr
/usr/sys01                581660   487244   36250    93%    /usr/sys01
/usr/public@sys01         808254   586026  141404    80%    /sys01/public
$ df -k ❷
Filesystem           1024-blocks     Used   Avail Capacity  Mounted on
/dev/rz0a                  19375    12570    4867    72%    /
/dev/rz0g                 274664   225369   21828    91%    /usr
/usr/sys01                290830   243623   18124    93%    /usr/sys01
/usr/publicsys01          404127   293013   70702    80%    /sys01/public
$ df -t ufs ❸
Filesystem            512-blocks     Used   Avail Capacity  Mounted on
/dev/rz0a                  38750    25140    9734    72%    /
/dev/rz0g                 549328   450738   43656    91%    /usr
```

❶ Display the free disk space on the available devices.

❷ Display the free disk space in kilobytes instead of 512-byte blocks.

❸ Display statistics for the ufs file system.

Learning Activity: Using the df Command

Use the `df` command to display the amount of free disk space available on all file systems currently mounted on your system.

Learning Activity Solution: Using the df Command

Use the `df` command to display the amount of free disk space available on all systems currently mounted on your system.

```
$ df
```

Using du to Display the Size of Files in Directories

The du command displays the size of files, in all directories specified.

- Block count is displayed in 1-kilobyte units.
- du -a displays the disk usage of each file recursively.
- du displays the disk usage of a directory tree and each subtree recursively.

Users can give commands to disk devices using the du command with the format:

```
/usr/bin/du [-alrs] [file ...] [directory ...]
```

du Command Options

The following table shows the commonly used du options.

Table 17–14 Some du Options

Option	Function
-a	Displays the amount of disk usage for each file; default is to report only on directories
-l	Allocates blocks evenly among the links in files with multiple links
-r	Displays an error message when an inaccessible directory or file is encountered
-s	Displays only a summary total

Using the du Command

This example shows how to use the du command. The file size is displayed to the left of each file name.

Example 17–8 Using the du Command

```
$ cd /usr/users      ❶
$ du -s guest        ❷
5       guest

$ du guest           ❸
1       guest/bin
5       guest

$ du -a guest        ❹
1       guest/bin
1       guest/.profile
1       guest/.login
1       guest/.cshrc
5       guest            ❺
```

❶ Change directory to /usr/users.

❷ Display the summary statistics for the guest directory.

❸ Display the sizes of any directories in the guest directory.

❹ Display the disk usage for each file and directory in the guest directory.

❺ The total amount of disk space that `/usr/users/guest` occupies is 5 blocks.

The total amount of disk usage by `/usr/users/guest` is 5 kilobytes.

Learning Activity: Using the du Command

1. Use the `du` command to display a list of the number of blocks used for each file currently owned by each user.
2. Use the `du` command to display a summary list of the total number of blocks occupied by each user directory.

Learning Activity Solution: Using the du Command

1. Use the `du` command to display a list of the number of blocks used for each file currently owned by each user.

```
$ du -a /usr/users/*
```

2. Use the `du` command to display a summary list of the total number of blocks occupied by each user directory.

```
$ du -s /usr/users/*
```

Summary

Identifying Devices

- UNIX treats all devices as special files.
- Use the `ls /dev` command to list available devices on your system.

Using the Tape Archive Utility

- Use the `tar -ruv` command to create private copies of your files and directories.
- Use the `tar -xv` command to extract files and directories from archives originally made with `tar`.

Other Commands

- Use the `cpio` command to copy any type of file to and from magnetic tape.
- Use the `ltf` command to write and read ANSI standard tapes. The `ltf` file formats are compatible with OpenVMS and other operating systems.
- Use the `dd` command to read and write a tape for use on a non-UNIX system such as OpenVMS.
- Use the `dd` command to convert EBCDIC files to ASCII files.
- Use the `mt` command to move tape forward and backward by a number of files or records and to rewind the tape.

Using Disks

- Use the `df` command to display the amount of free disk space.
- Use the `du` command to display the size of files in directories.

18
Test

Introduction

Overview

The following test is provided to help you assess how well you have understood the material in this course. It is intended solely as a self-evaluation tool.

The test is divided into chapters that correspond to the chapters in this course. If, after completing the test, you find that you did not do as well on some chapters as you did on others, you may want to review the material in those specific chapters.

Materials Needed

UNIX Utilities and Commands Test and a pencil.

Self-Evaluation

After you have completed the test, compare your answers to the solutions provided. The passing score for this test is 80% (48 or more correct out of 60 possible responses).

Questions

Introduction to UNIX Utilities and Commands

1. What command would you use to change your password?

   a. `pwd`

   b. `chpass`

   c. `passwd`

   d. `set pass`

2. What command would you type to obtain online information about the `login` command?

   a. `help login`

   b. `?`

   c. `login ?`

   d. `man login`

3. After invoking the `man` command, what key should you press to display another screenful of online documentation text on your screen?

   a. Ctrl/Q

   b. Ctrl/C

   c. Q

   d. Spacebar

Setting Up Your Terminal

4. What command is used to find out which keys on your terminal serve the erase, kill, and interrupt functions?

   a. `stty`

   b. `term`

   c. `print tty`

   d. `sys`

5. The normal way to stop (interrupt) a process that is no longer interesting, or to regain control in an interactive program, is by pressing which keys?

   a. Ctrl/Z

   b. Ctrl/D

   c. Ctrl/S

   d. Ctrl/C

Invoking Motif Applications

6. From what menu can desktop applications be selected?
   a. The **Session Manager** window Session menu.
   b. The **Session Manager** window File menu.
   c. The **DECterm** window Options menu.
   d. The **Session Manager** window Applications menu.

File System Concepts

7. What type of file contains a group of files?
   a. Regular disk file
   b. Directory
   c. Device file
   d. Special file
8. Which command provides the type of information contained in a particular file?
   a. `more` *`filename`*
   b. `type` *`filename`*
   c. `file` *`filename`*
   d. `cat` *`filename`*

Directory and File Access

9. In the DEC OSF/1 operating system, a **full** pathname originates from:
   a. Your home directory
   b. The root directory
   c. Your working directory
   d. Your login directory
10. The command to change your current directory is:
   a. `cd pathname`
   b. `set default pathname`
   c. `sd pathname`
   d. `ccd pathname`

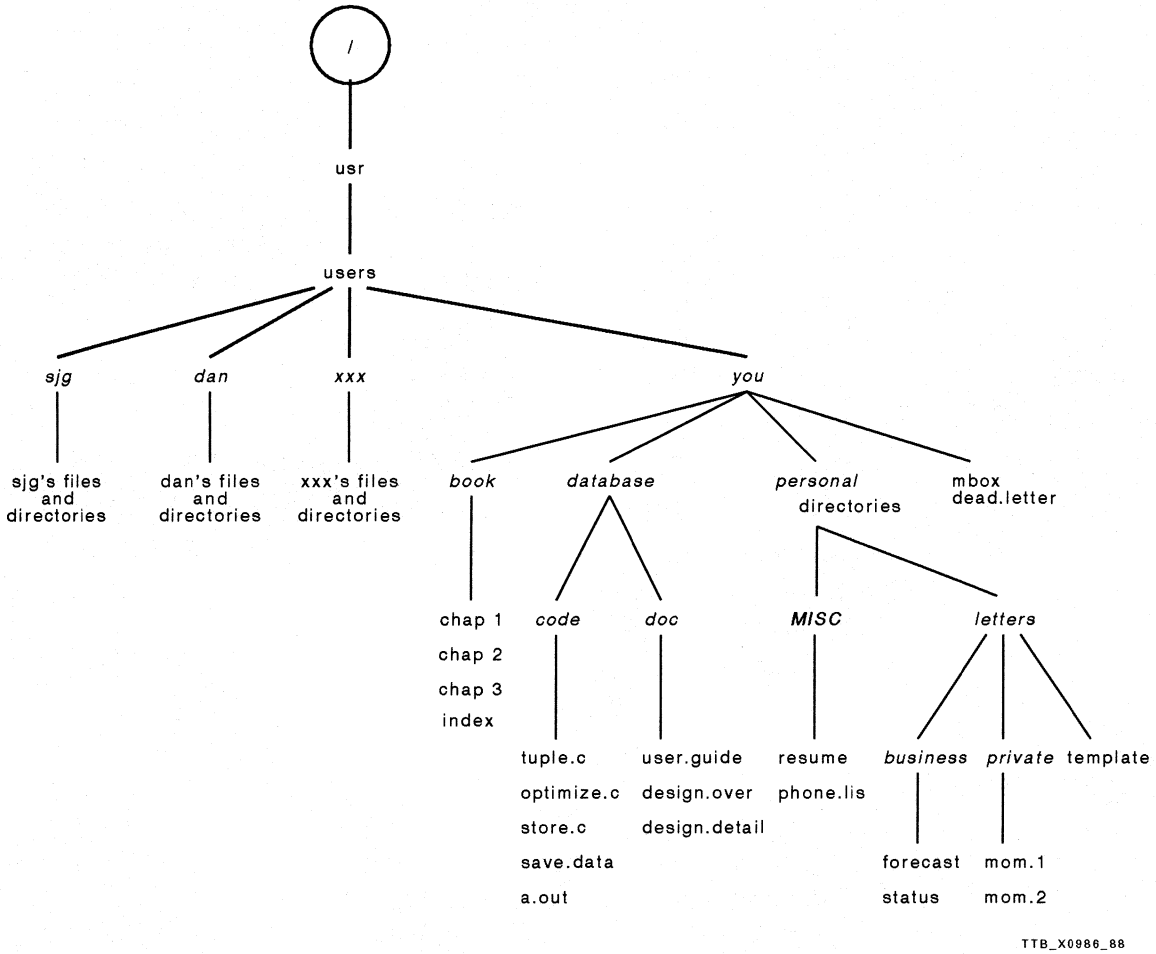
11. You have just finished creating *chap1*. A listing of your working directory shows the four files *chap1*, *chap2*, *chap3*, and *index*. What is the output of the `pwd` command if the directory structure is that of the following figure?

    a. */usr/you/database*

    b. */usr/you/dan*

    c. */usr/users/you/personal*

    d. */usr/users/you/book*



12. Which command displays a listing of all files in a directory with information regarding file access permissions?

    a. `ls -p`

    b. `dir/full`

    c. `ls -F`

    d. `ls -l`

13. What command will search for files and directories based on certain specified criteria?
    a. find
    b. search
    c. grep
    d. locate
14. Which command will display a file one screenful at a time?
    a. more filename
    b. type filename
    c. file filename
    d. cat filename

File Management

15. Assume your current working directory is `/usr/users/you/book`, what command creates a directory `text` under your home directory?
    a. `mkdir /usr/users/you/text`
    b. `mkdir /text`
    c. `mkdir /book/text`
    d. `dir/create /usr/users/you/text`
16. Which of the following commands gives the user **read**, **write**, and **execute** access, the group **read** and **execute** access, and others **no** access to the file named `a.out`?
    a. `chmod 660 a.out`
    b. `chmod 730 a.out`
    c. `chmod 430 a.out`
    d. `chmod 750 a.out`
17. Which command, issued from your home directory, will move the files `/usr/users/you/personal/misc/phones` and `/usr/users/you/personal/misc/resume` to the `/usr/users/you/personal` directory?
    a. `mv personal/misc/phones personal/misc/resume personal`
    b. `mv resume + phones personal`
    c. `mv personal/misc/phones resume personal`
    d. `mv phones resume personal`

18. Issued from your home directory, what command will delete the `/usr/users/you/personal` directory and all the files in that directory?

    a. `rmdir /usr/users/you/personal`

    b. `rmdir -r personal`

    c. `rm -r personal`

    d. `rmdir /personal`

19. Which of the following statements is true of symbolic links?

    a. The file can be accessed through the symbolic link after the original file has been removed

    b. Cannot be made across file systems

    c. Can be made to a directory

    d. The creation of each symbolic link causes the link count to be incremented

Shell Overview and Korn Shell Introduction

20. Which of the following could be considered as a purpose of the shell?

    a. User interface to the DEC OSF/1 operating system

    b. A command language interpreter

    c. A programming language

    d. All of the above

21. Which shells provide a command history buffer?

    a. Bourne shell

    b. C shell

    c. Korn shell

    d. Both C and Korn shells

22. Which of the following is the correct way to set the variable, *myname*, with the Korn shell?

    a. `set myname='Charlie Brown'`

    b. `set myname = "Charlie Brown"`

    c. `myname = 'Charlie Brown'`

    d. `myname="Charlie Brown"`

23. All Korn shell options are displayed with which of the following commands?

    a. `set -o` *`[option]`*

    b. `set -o`

    c. `set +o` *`[option]`*

    d. `set +o`

Introduction to UNIX Editors and the emacs Editor

24. Which of the following editors provides the capability of editing both multiple buffers and multiple windows?

    a. `ed`

    b. `vi`

    c. `emacs`

    d. `sed`

25. Which of the following commands will start `emacs` in the current X window (will not start `emacs` in a separate window)?

    a. `emacs -wn` *`junkfile`*

    b. `emacs -w` *`junkfile`*

    c. `emacs -r` *`junkfile`*

    d. `emacs -nw` *`junkfile`*

26. Which of the following is a valid file name for an `emacs` backup file?

    a. *filename#*

    b. *#filename#*

    c. *~filename~*

    d. *filename~*

27. Which of the following file names is valid for an `emacs` autosave file?

    a. *filename*

    b. *#filename*

    c. *filename#*

    d. *#filename#*

The vi Editor

28. If you use the `vi` editor to edit an existing file named `junk`, and save it to a file with the same name with the `:wq` command, what name does the editor give to the previous (unedited) version of the file?

    a. `junk~`

    b. `junk.bak`

    c. `junk;1`

    d. The previous version of the file is not preserved.

29. Which `vi` mode is used to edit another file, execute a shell command, or execute an `ex` command?
    a. Command mode
    b. Text-input mode
    c. Last-line mode
    d. Exit mode
30. Which of these control sequences should be used to go from `vi` Text-input mode to Command mode?
    a. eof (Ctrl/D)
    b. escape (Ctrl/ [ )
    c. stop (Ctrl/S)
    d. suspend (Ctrl/Z)
31. Which of the following is **not** a valid command to leave the `vi` Command mode and return to the shell prompt?
    a. `Q`
    b. `ZZ`
    c. `:q`
    d. `:q!`

Redirecting, Filtering, and Piping

32. Which command appends a file called `test` to the end of a file called `output`?
    a. `cat output >> test`
    b. `append output > test`
    c. `write output > test`
    d. `cat test >> output`
33. What does the < redirection symbol do?
    a. Redirects standard output
    b. Redirects standard input
    c. Adds output to the end of a file
    d. Redirects standard input and standard output

34. The `dates` file contains three fields: first names, last names, and birthdays, in that order. Which command creates a new file called `dates.new` containing a version of the `dates` file that is in alphabetical order, sorted first by last names, and then by first names?

    a. `sort +1 -2 dates > dates.new`

    b. `sort -o dates +1 -2`

    c. `sort dates +1 -2 > dates.new`

    d. `sort -o dates.new dates +1`

35. Which of the following commands will **NOT** look for the phrase **yellow ribbon** in the file named `song`?

    a. grep yellow ribbon song

    b. grep 'yellow ribbon' song

    c. grep "yellow ribbon" song

    d. grep < song "yellow ribbon"

36. To display a list of all accounts on the system with a login shell of `ksh`, sorted alphabetically by login name and displayed a screen at a time, which of the following commands could be used?

    a. `more /etc/passwd | grep "/usr/users/ksh" | sort`

    b. `grep 'ksh' /etc/passwd ; sort; more`

    c. `sort /etc/passwd > grep "ksh" > more`

    d. `cat /etc/passwd | grep "ksh" | sort | more`

37. Which command runs an `awk` command file named `sum2.awk` to process a file named `weekly.sum`?

    a. `awk -F sum2.awk weekly.sum`

    b. `awk -f weekly.sum sum2.awk`

    c. `awk '[sum2.awk]' weekly.sum`

    d. `awk -f sum2.awk weekly.sum`

Communicating with Other Users

38. Which user information command would you use to find out the real name and phone extension of user *pmj*?

    a. `who pmj`

    b. `whoami pmj`

    c. `users pmj`

    d. `finger pmj`

39. Which Mail command allows you to send a response to the user name of a mail message you just read?

    a. `R`

    b. `~r`

    c. Return

    d. `-`

40. Which Mail command displays a list of commands along with a summary of their meanings?

    a. `usage`

    b. `?`

    c. Ctrl/H

    d. `h`

41. Which of the following statements about the *.mailrc* file is true?

    a. A user must have a *.mailrc* file to be able to send or receive mail.

    b. The *.mailrc* file is a binary file that cannot be modified with an ordinary text editor.

    c. *.mailrc* commands are executed after *Mail.rc* commands.

    d. Your *.mailrc* file contains those mail messages that you have received but not yet read.

Recalling and Editing Korn Shell Commands

42. Which of the following commands may be used to view the commands in the history buffer?

    a. `show history`

    b. `history`

    c. `cat history`

    d. `view commands`

43. The Korn shell has a default history file and default number of retained executed commands. These defaults are:

    a. *.history*, 128

    b. *.sh_history*, 128

    c. *.ksh_history*, 64

    d. *history*, 64

44. There are different ways to turn on the Korn shell built-in command-line editor. If the following three option/variable assignments were made, which built-in editor is used as the Korn shell command-line editor?

```
EDITOR=emacs
VISUAL=gmacs
set -o vi
```

a. /bin/ed

b. emacs

c. gmacs

d. vi

45. Which of the following correctly defines the editor to use with the Korn shell `fc` built-in command to edit a command in a temporary buffer?

a. `set -o emacs`

b. `EDITOR=emacs`

c. `set +o emacs`

d. `FCEDIT=emacs`

46. You use **emacs** to edit, and modify, multiple Korn shell commands in a temporary buffer. When you exit **emacs**, you tell **emacs** not to save the buffer in the **/tmp** directory. Which commands get executed?

a. No commands are executed

b. The original contents of the buffer

c. The modified commands, using your current working directory

d. The modified commands, but they are executed with your home directory as your working directory instead of `/tmp`

Controlling Processes

47. You can continue entering a long command on the next line by entering what character?

    a. ;

    b. \

    c. /

    d. -

48. You can include several commands on the same command line by separating them with what character?

    a. \

    b. ;

    c. ,

    d. TAB

49. What command would you enter to find out what processes are currently in execution, including those processes not connected to a terminal?

    a. `stat`

    b. `jobs`

    c. `bg`

    d. `ps -e`

50. The default signal sent by the `kill` command is:

    a. `INT`

    b. `KILL`

    c. `TERM`

    d. `USR1`

Introduction to Korn Shell Scripts

51. What character is used to introduce comments in Korn shell scripts?

    a. :

    b. \

    c. ;

    d. #

52. To make the Korn shell variable MYNAME available in child shells, you should use the command:

    a. `inherit $MYNAME`

    b. `export MYNAME`

    c. `use $MYNAME`

    d. No command is necessary, child shells always inherit all the parent shell's variables.

53. What command could be used to check for the existence of a file *'junk'* in a Korn shell script?

    a. `if [[ -f $junk ]] then print "junk exists"`

    b. `if [[ -f junk ]] ;then print "junk exists"`

    c. `if [[ -e junk ]] ;then print "junk exists"`

    d. `if [[ junk==$HOME ]] then print "junk exists"`

54. Which word is used to end a conditional loop in the Korn shell?

    a. end

    b. enddo

    c. endif

    d. done

Using TCP/IP Networks

55. Which of the following commands would you use to copy files from a remote DEC OSF/1 system to your local DEC OSF/1 system over a TCP/IP network?

    a. `copy`

    b. `dcp`

    c. `rcp`

    d. `finger`

56. Which of the following commands would you use to log in to a remote system using a different user ID?

    a. `rsh newsys df`

    b. `login newsys -l thomas`

    c. `rcp newsys`

    d. `rlogin newsys -l thomas`

Printing

57. The primary purpose of the `pr` print command is to:
    a. Print files to the default printer
    b. Print files to the printer designated by the `-P` option
    c. Format print pages for standard output
    d. Provide a list of print requests in the printer queue
58. Which of the following commands provides status information for printer lp1?
    a. `lpq -Plp1`
    b. `lpstat -t`
    c. `lpc status`
    d. All of the above

Using Tapes and Disks

59. What command extracts original copies of files and directories from a tape archive?
    a. The `tar -cv` command
    b. The `tar -rv` command
    c. The `tar -t` command
    d. The `tar -xv` command
60. What command displays the amount of free disk space?
    a. The `du` command
    b. The `dd` command
    c. The `df` command
    d. The `ls -l` command

Answers

Introduction to UNIX Utilities and Commands

1. What command would you use to change your password?
   a. `pwd`
   b. `chpass`
   c. `passwd`
   d. `set pass`
2. What command would you type to obtain online information about the `login` command?
   a. `help login`
   b. `?`
   c. `login ?`
   d. `man login`
3. After invoking the `man` command, what key should you press to display another screenful of online documentation text on your screen?
   a. Ctrl/Q
   b. Ctrl/C
   c. Q
   d. Spacebar—This is the correct answer

Setting Up Your Terminal

4. What command is used to find out which keys on your terminal serve the erase, kill, and interrupt functions?
   a. `stty`
   b. `term`
   c. `print tty`
   d. `sys`
5. The normal way to stop (interrupt) a process that is no longer interesting, or to regain control in an interactive program, is by pressing which keys?
   a. Ctrl/Z
   b. Ctrl/D
   c. Ctrl/S
   d. Ctrl/C—This is the correct answer

Invoking Motif Applications

6. From what menu can desktop applications be selected?
   a. The **Session Manager** window Session menu.
   b. The **Session Manager** window File menu.
   c. The **DECterm** window Options menu.
   d. The **Session Manager** window Applications menu.

File System Concepts

7. What type of file contains a group of files?
   a. Regular disk file
   b. Directory
   c. Device file
   d. Special file
8. Which command provides the type of information contained in a particular file?
   a. `more` *`filename`*
   b. `type` *`filename`*
   c. `file` *`filename`*
   d. `cat` *`filename`*

Directory and File Access

9. In the DEC OSF/1 operating system, a **full** pathname originates from:
   a. Your home directory
   b. The root directory
   c. Your working directory
   d. Your login directory
10. The command to change your current directory is:
   a. `cd pathname`
   b. `set default pathname`
   c. `sd pathname`
   d. `ccd pathname`

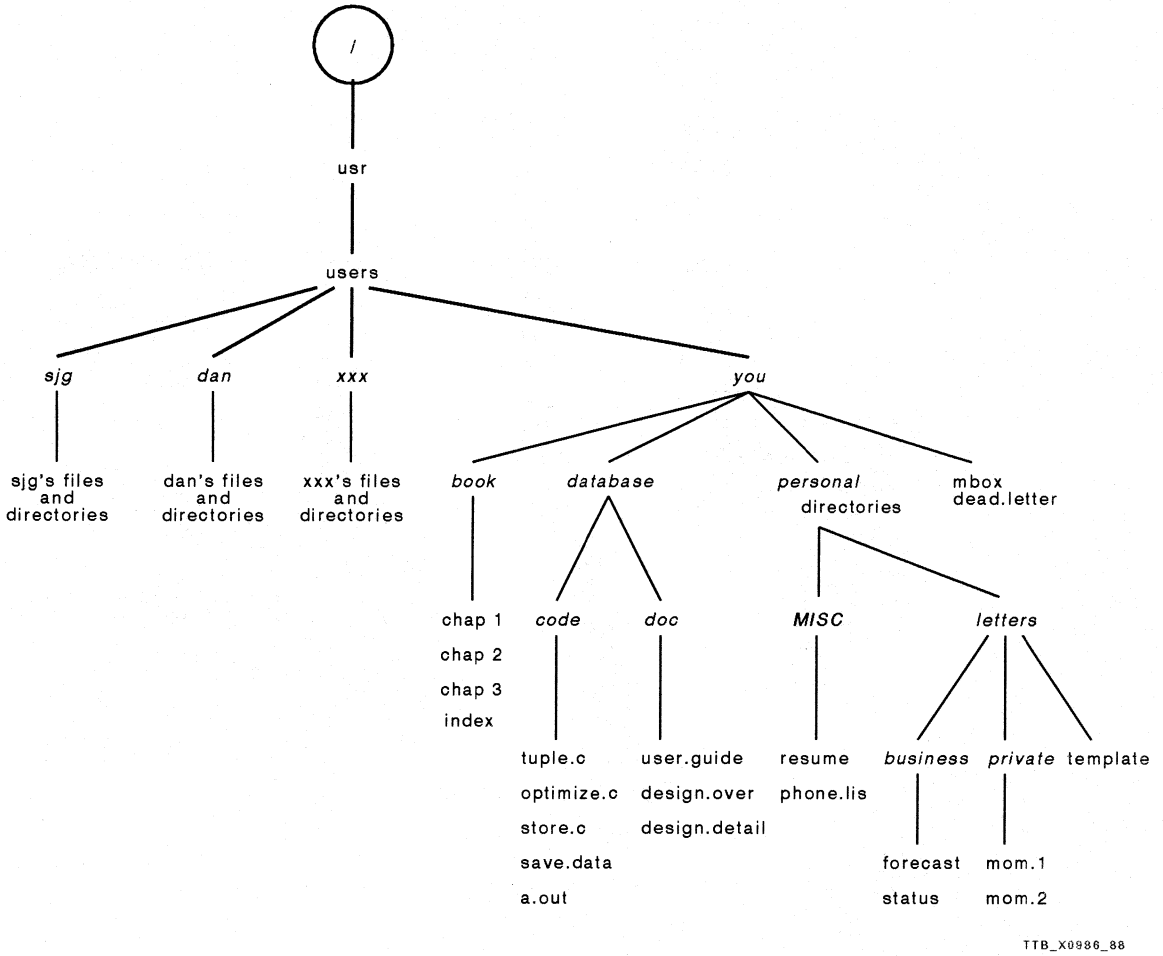
11. You have just finished creating *chap1*. A listing of your working directory shows the four files *chap1*, *chap2*, *chap3*, and *index*. What is the output of the `pwd` command if the directory structure is that of the following figure?

    a. */usr/you/database*

    b. */usr/you/dan*

    c. */usr/users/you/personal*

    d. */usr/users/you/book*



12. Which command displays a listing of all files in a directory with information regarding file access permissions?

    a. `ls -p`

    b. `dir/full`

    c. `ls -F`

    d. `ls -l`

13. What command will search for files and directories based on certain specified criteria?

    a. find

    b. search

    c. grep

    d. locate

14. Which command will display a file one screenful at a time?

    a. more filename

    b. type filename

    c. file filename

    d. cat filename

File Management

15. Assume your current working directory is `/usr/users/you/book`, what command creates a directory `text` under your home directory?

    a. `mkdir /usr/users/you/text`

    b. `mkdir /text`

    c. `mkdir /book/text`

    d. `dir/create /usr/users/you/text`

16. Which of the following commands gives the user **read**, **write**, and **execute** access, the group **read** and **execute** access, and others **no** access to the file named `a.out`?

    a. `chmod 660 a.out`

    b. `chmod 730 a.out`

    c. `chmod 430 a.out`

    d. `chmod 750 a.out`

17. Which command, issued from your home directory, will move the files `/usr/users/you/personal/misc/phones` and `/usr/users/you/personal/misc/resume` to the `/usr/users/you/personal` directory?

    a. `mv personal/misc/phones personal/misc/resume personal`

    b. `mv resume + phones personal`

    c. `mv personal/misc/phones resume personal`

    d. `mv phones resume personal`

18. Issued from your home directory, what command will delete the `/usr/users/you/personal` directory and all the files in that directory?

    a. `rmdir /usr/users/you/personal`

    b. `rmdir -r personal`

    c. `rm -r personal`

    d. `rmdir /personal`

19. Which of the following statements is true of symbolic links?

    a. The file can be accessed through the symbolic link after the original file has been removed

    b. Cannot be made across file systems

    c. Can be made to a directory

    d. The creation of each symbolic link causes the link count to be incremented

Shell Overview and Korn Shell Introduction

20. Which of the following could be considered as a purpose of the shell?

    a. User interface to the DEC OSF/1 operating system

    b. A command language interpreter

    c. A programming language

    d. All of the above

21. Which shells provide a command history buffer?

    a. Bourne shell

    b. C shell

    c. Korn shell

    d. Both C and Korn shells

22. Which of the following is the correct way to set the variable, *myname*, with the Korn shell?

    a. `set myname='Charlie Brown'`

    b. `set myname = "Charlie Brown"`

    c. `myname = 'Charlie Brown'`

    d. `myname="Charlie Brown"`

23. All Korn shell options are displayed with which of the following commands?

    a. `set -o` *`[option]`*

    b. `set -o`

    c. `set +o` *`[option]`*

    d. `set +o`

Introduction to UNIX Editors and the emacs Editor

24. Which of the following editors provides the capability of editing both multiple buffers and multiple windows?

    a. `ed`

    b. `vi`

    c. `emacs`

    d. `sed`

25. Which of the following commands will start `emacs` in the current X window (will not start `emacs` in a separate window)?

    a. `emacs -wn` *`junkfile`*

    b. `emacs -w` *`junkfile`*

    c. `emacs -r` *`junkfile`*

    d. `emacs -nw` *`junkfile`*

26. Which of the following is a valid file name for an `emacs` backup file?

    a. *filename#*

    b. *#filename#*

    c. *~filename~*

    d. *filename~*

27. Which of the following file names is valid for an `emacs` autosave file?

    a. *filename*

    b. *#filename*

    c. *filename#*

    d. *#filename#*

The vi Editor

28. If you use the `vi` editor to edit an existing file named `junk`, and save it to a file with the same name with the `:wq` command, what name does the editor give to the previous (unedited) version of the file?

    a. `junk~`

    b. `junk.bak`

    c. `junk;1`

    d. The previous version of the file is not preserved.

29. Which `vi` mode is used to edit another file, execute a shell command, or execute an `ex` command?
    a. Command mode
    b. Text-input mode
    c. Last-line mode
    d. Exit mode
30. Which of these control sequences should be used to go from `vi` Text-input mode to Command mode?
    a. eof (Ctrl/D)
    b. escape (Ctrl/ [ )
    c. stop (Ctrl/S)
    d. suspend (Ctrl/Z)
31. Which of the following is **not** a valid command to leave the `vi` Command mode and return to the shell prompt?
    a. `Q`
    b. `ZZ`
    c. `:q`
    d. `:q!`

Redirecting, Filtering, and Piping

32. Which command appends a file called `test` to the end of a file called `output`?
    a. `cat output >> test`
    b. `append output > test`
    c. `write output > test`
    d. `cat test >> output`
33. What does the < redirection symbol do?
    a. Redirects standard output
    b. Redirects standard input
    c. Adds output to the end of a file
    d. Redirects standard input and standard output

34. The `dates` file contains three fields: first names, last names, and birthdays, in that order. Which command creates a new file called `dates.new` containing a version of the `dates` file that is in alphabetical order, sorted first by last names, and then by first names?

    a. `sort +1 -2 dates > dates.new`

    b. `sort -o dates +1 -2`

    c. `sort dates +1 -2 > dates.new`

    d. `sort -o dates.new dates +1`

35. Which of the following commands will **NOT** look for the phrase **yellow ribbon** in the file named `song`?

    a. `grep yellow ribbon song`

    b. grep 'yellow ribbon' song

    c. grep "yellow ribbon" song

    d. grep < song "yellow ribbon"

36. To display a list of all accounts on the system with a login shell of `ksh`, sorted alphabetically by login name and displayed a screen at a time, which of the following commands could be used?

    a. `more /etc/passwd | grep "/usr/users/ksh" | sort`

    b. `grep 'ksh' /etc/passwd ; sort; more`

    c. `sort /etc/passwd > grep "ksh" > more`

    d. `cat /etc/passwd | grep "ksh" | sort | more`

37. Which command runs an `awk` command file named `sum2.awk` to process a file named `weekly.sum`?

    a. `awk -F sum2.awk weekly.sum`

    b. `awk -f weekly.sum sum2.awk`

    c. `awk '{sum2.awk}' weekly.sum`

    d. `awk -f sum2.awk weekly.sum`

Communicating with Other Users

38. Which user information command would you use to find out the real name and phone extension of user *pmj*?

    a. `who pmj`

    b. `whoami pmj`

    c. `users pmj`

    d. `finger pmj`

39. Which Mail command allows you to send a response to the user name of a mail message you just read?

    a. R

    b. ~r

    c. Return

    d. -

40. Which Mail command displays a list of commands along with a summary of their meanings?

    a. usage

    b. ?

    c. Ctrl/H

    d. h

41. Which of the following statements about the *.mailrc* file is true?

    a. A user must have a *.mailrc* file to be able to send or receive mail.

    b. The *.mailrc* file is a binary file that cannot be modified with an ordinary text editor.

    c. *.mailrc* commands are executed after *Mail.rc* commands.

    d. Your *.mailrc* file contains those mail messages that you have received but not yet read.

Recalling and Editing Korn Shell Commands

42. Which of the following commands may be used to view the commands in the history buffer?

    a. show history

    b. history

    c. cat history

    d. view commands

43. The Korn shell has a default history file and default number of retained executed commands. These defaults are:

    a. *.history*, 128

    b. *.sh_history*, 128

    c. *.ksh_history*, 64

    d. *history*, 64

44. There are different ways to turn on the Korn shell built-in command-line editor. If the following three option/variable assignments were made, which built-in editor is used as the Korn shell command-line editor?

```
EDITOR=emacs
VISUAL=gmacs
set -o vi
```

a. /bin/ed

b. emacs

c. gmacs

d. vi

45. Which of the following correctly defines the editor to use with the Korn shell `fc` built-in command to edit a command in a temporary buffer?

a. `set -o emacs`

b. `EDITOR=emacs`

c. `set +o emacs`

d. `FCEDIT=emacs`

46. You use **emacs** to edit, and modify, multiple Korn shell commands in a temporary buffer. When you exit **emacs**, you tell **emacs** not to save the buffer in the **/tmp** directory. Which commands get executed?

a. No commands are executed

b. The original contents of the buffer

c. The modified commands, using your current working directory

d. The modified commands, but they are executed with your home directory as your working directory instead of `/tmp`

Controlling Processes

47. You can continue entering a long command on the next line by entering what character?

    a. ;

    b. \

    c. /

    d. -

48. You can include several commands on the same command line by separating them with what character?

    a. \

    b. ;

    c. ,

    d. TAB

49. What command would you enter to find out what processes are currently in execution, including those processes not connected to a terminal?

    a. `stat`

    b. `jobs`

    c. `bg`

    d. `ps -e`

50. The default signal sent by the `kill` command is:

    a. `INT`

    b. `KILL`

    c. `TERM`

    d. `USR1`

Introduction to Korn Shell Scripts

51. What character is used to introduce comments in Korn shell scripts?

    a. :

    b. \

    c. ;

    d. #

52. To make the Korn shell variable MYNAME available in child shells, you should use the command:

    a. `inherit $MYNAME`

    b. `export MYNAME`

    c. `use $MYNAME`

    d. No command is necessary, child shells always inherit all the parent shell's variables.

53. What command could be used to check for the existence of a file *'junk'* in a Korn shell script?

    a. `if [[ -f $junk ]] then print "junk exists"`

    b. `if [[ -f junk ]] ;then print "junk exists"`

    c. `if [[ -e junk ]] ;then print "junk exists"`

    d. `if [[ junk==$HOME ]] then print "junk exists"`

54. Which word is used to end a conditional loop in the Korn shell?

    a. end

    b. enddo

    c. endif

    d. done

Using TCP/IP Networks

55. Which of the following commands would you use to copy files from a remote DEC OSF/1 system to your local DEC OSF/1 system over a TCP/IP network?

    a. `copy`

    b. `dcp`

    c. `rcp`

    d. `finger`

56. Which of the following commands would you use to log in to a remote system using a different user ID?

    a. `rsh newsys df`

    b. `login newsys -l thomas`

    c. `rcp newsys`

    d. `rlogin newsys -l thomas`

Printing

57. The primary purpose of the `pr` print command is to:
    a. Print files to the default printer
    b. Print files to the printer designated by the `-P` option
    c. Format print pages for standard output
    d. Provide a list of print requests in the printer queue
58. Which of the following commands provides status information for printer lp1?
    a. `lpq -Plp1`
    b. `lpstat -t`
    c. `lpc status`
    d. All of the above

Using Tapes and Disks

59. What command extracts original copies of files and directories from a tape archive?
    a. The `tar -cv` command
    b. The `tar -rv` command
    c. The `tar -t` command
    d. The `tar -xv` command
60. What command displays the amount of free disk space?
    a. The `du` command
    b. The `dd` command
    c. The `df` command
    d. The `ls -l` command

A

stty Command Arguments

Setting Terminal Characteristics

Overview

Use the arguments in the following tables to set the terminal characteristics.

Table A–1 stty Command Arguments: Control Modes

Argument	Function
`parenb ( -parenb )`	Enables (disables) parity generation and detection.
`parodd ( -parodd )`	Selects odd (even) parity.
`cs5 cs6 cs7 cs8`	Selects character size, if possible.
`number`	Sets terminal baud rate to the number given, if possible. If the baud rate is set to zero, modem control is no longer asserted.
`ispeed number`	Sets terminal input baud rate to the number given, if possible. If zero is specified, the input baud rate is set to equal the output baud rate.
`ospeed number`	Sets terminal output baud rate to the number given, if possible. If the output baud rate is set to zero, modem control is no longer asserted.
`hupcl ( -hupcl )`	Stops (does not stop) asserting modem control on last close.
`hup ( -hup )`	Performs the same as `hupcl (-hupcl )`.
`cstopb ( -cstopb )`	Uses two (one) stop bits per character.
`cread ( -cread )`	Enables (disables) the receiver.
`clocal ( -clocal )`	Assumes a line without (with) modem control.
`crtscts ( -crtscts )`	Uses (does not use) RTS/CTS hardwired flow control.
`nokerninfo ( -nokerninfo )`	Disables (enables) the printing of kernel-generated status information when the info control character is entered.

Table A–2 stty Command Arguments: Input Modes

Argument	Function
`echoctl ( -echoctl )`	Echoes control characters as ^X and ⌫ as ^?. Prints two backspaces following the End-of-File character. (Special characters are echoed as themselves.)
`ignbrk ( -ignbrk )`	Ignores (does not ignore) break on input.
`brkint ( -brkint )`	Signals (does not signal) `intr` on break.
`ignpar ( -ignpar )`	Ignores (does not ignore) parity errors.
`parmrk ( -parmrk )`	Marks (does not mark) parity errors.
`inpck ( -inpck )`	Enables (disables) input parity checking.
`istrip ( -istrip )`	Strips (does not strip) input characters to seven bits.
`inlcr ( -inlcr )`	Maps (does not map) new line to carriage return on input.
`igncr ( -igncr )`	Ignores (does not ignore) carriage return on input.
`icrnl ( -icrnl )`	Maps (does not map) carriage return to new line on input.
`imaxbel ( -imaxbel )`	Rings (does not ring) bell on terminal when input buffer is full.
`iuclc ( -iuclc )`	Maps (does not map) uppercase letters to lowercase.
`ixon ( -ixon )`	Enables (disables) Start/Stop output control. Output from the system is stopped when the system receives Stop, and started when the system receives Start.
`ixany ( -ixany )`	Allows any character (allows only Ctrl/Q) to restart output.
`ixoff ( -ixoff )`	Requests that the system send (not send) Start/Stop characters when the input queue is nearly empty/full.

Table A–3 stty Command Arguments: Output Modes

Argument	Function
opost (-opost)	Post-processes output (does not post-process output) and ignores all other output modes.
olcuc (-olcuc)	Maps (does not map) lowercase letters to uppercase on output.
onoeot (-onoeot)	Discards (keeps) End-of-Text on output.
onlcr (-onlcr)	Maps (does not map) new line characters to carriage return/new line characters.
ocrnl (-ocrnl)	Maps (does not map) carriage return/new line characters to new line characters.
onocr (-onocr)	Outputs (does not output) carriage return characters at column 0 (zero).
onlret (-onlret)	Causes (does not cause) new line to perform the carriage return function on the terminal.
ofill (-ofill)	Uses fill characters (uses timing) for delays.
ofdel (-ofdel)	Uses delete (uses null) characters for fill characters.
tabs (-tabs)	Maintains (expands to spaces) any tab characters in the output.
cr0 cr1 cr2 cr3	Selects style of delay for carriage return characters.
nl0 nl1 nl2 nl32	Selects style of delay for new line characters.
tab0 tab1 tab2 tab3	Selects style of delay for horizontal tabs.
bs0 bs1	Selects style of delay for backspaces.
ff0 ff1	Selects style of delay for form feeds.
vt0 vt1	Selects style of delay for vertical tabs.

Table A–4 stty Command Arguments: Local Modes

Argument	Function
altwerase (-altwerase)	Uses (does not use) the altwerase mode, which defines a word as containing only alphanumeric characters and underscore (_).
isig (-isig)	Enables (disables) the checking of characters against the special control characters intr, quit, and susp.
icanon (-icanon)	Enables (disables) canonical input (erase and kill processing).
crtkill (-crtkill)	Echoes (does not echo) the kill character by erasing the line in place like echo.
mdmbuf (-mdmbuf)	Uses (does not use) carrier as a flow control flag rather than sending a HANGUP signal.
prterase (-prterase)	Prints (does not print) erased characters backward within backslash (\) and slash (/).
tostop (-tostop)	Stops (allows) output from background jobs to the terminal.
xcase (-xcase)	Echoes (does not echo) uppercase letters on input, and displays uppercase letters on output with a preceding backslash.
iexten (-iexten)	Enables (disables) any implementation-defined special control characters not currently controlled by icanon, isig, or ixon.
echo (-echo)	Echoes back (does not echo back) every character typed.
echoe (-echoe)	Causes the erase character to visually erase (not visually erase) the last character in the current line from the display, if possible.
echok (-echok)	Echoes (does not echo) new line after the kill character.
echonl (-echonl)	Echoes (does not echo) new line, even if echo is disabled.
noflsh (-noflsh)	Disables (enables) flush after intr, quit, susp.

(continued on next page)

Table A–4 (Cont.) stty Command Arguments: Local Modes

Argument	Function
special_character string	Sets special_character to string. The special character is set to the first character in string and subsequent characters are ignored, with the following exceptions: The strings undef and — set the special character to {_POSIX_VDISABLE} if it is in effect for the device. The string ^? sets the special character to ⌫. Any other string beginning with the character ^ sets the special character to the control character corresponding to the second character of string (subsequent characters are ignored). For example, the string ^c sets the special character to Ctrl/C; the string ^zq sets the special character to Ctrl/Z. Note that you can set a special character to a control character in two ways: by entering the control character itself or by entering ^ and another character. This allows you to enter a control character already assigned to a special character without entering that special character. For example, you can enter ^C, even if it is already assigned to the intr special character, by entering ^ and then c. Recognized special characters include dsusp, eof, eol, eol2, erase, discard, status, intr, kill, lnext, quit, reprint, start, stop, susp, and werase.
saved settings	Sets the current terminal characteristics to the saved settings produced by the -g option.
min number or time number	Sets the value of min or time to number. Both min and time are used in noncanonical mode input processing (-icanon).
line number	Sets the line discipline to the specified number.

Table A–5 stty Command Arguments: Combination Modes

Argument	Function
evenp or parity	Enables parenb and cs7; disables parodd.
oddp	Enables parenb, cs7, and parodd.
parity, evenp, oddp	Disables parenb, and sets cs8.
raw (-raw | cooked)	Enables (disables) raw input and output (no erase, kill, intr, quit, eot, or output processing).
nl (-nl)	Enables (disables) icrnl and onlcr. -nl also unsets inlcr, igncr ocrnl, and onlret.
lcase (-lcase) or LCASE (-LCASE)	Sets (unsets) xcase, iuclc, and olcuc. Used for terminals with uppercase letters only.
sane	Resets all modes to some reasonable values.

Table A–6 stty Command Arguments: Compatibility Mode

Argument	Function
ek	Resets erase and kill characters back to system defaults.
lfkc (-lfkc)	Same as echok (-echok).
flow (-flow)	Same as ixon (-ixon).
tandem (-tandem)	Same as ixoff (-ixoff).
decctlq (-decctlq)	Same as ixany (-ixany).
dec	Sets all modes suitable for terminals supplied by Digital Equipment Corporation. The erase control character is set to ^? .
crterase (-crterase) or crtbs (-crtbs)	Same as echoe (-echoe).
ctlecho (-ctlecho)	Same as echoctl (-echoctl).
crt (-crt) or newcrt (-newcrt)	Sets (clears) echoe, echoke, and echoctl.
litout (-litout)	Sends output characters without (with) output processing.
xtabs (-xtabs) or oxtabs (-oxtabs)	Expands (does not expand) tabs to spaces.
fill (-fill)	Same as ofill (-ofill).
everything	Same as -a.
echoke (-echoke) or nohang (-nohang)	Sends (does not send) HANGUP signal if carrier drops.

(continued on next page)

Table A–6 (Cont.) stty Command Arguments: Compatibility Mode

Argument	Function
`nul-fill`	Performs character fill and uses `null` character.
`del-fill`	Performs character fill and uses `delete` character.
`tty33`	Sets modes suitable for the Teletype Corporation Model 33 terminal.
`tty37`	Sets modes suitable for the Teletype Corporation Model 37 terminal.
`vt05`	Sets modes suitable for the Digital Equipment Corporation Mode VT05 terminal.
`tn300`	Sets modes suitable for the General Electric TermiNet 300.
`ti700`	Sets modes suitable for the Texas Instruments 700 series.
`tek`	Sets modes suitable for the Tektronix 4014 terminal.
`speed`	Prints the line speed only.
`size`	Prints the terminal size only.

General Notes

- If no options are specified, an unspecified subset of the information displayed for the `-a` flag is displayed.
- If the terminal input speed and output speed are the same, the speed information is displayed as follows:

  speed speed baud

  Otherwise speeds are displayed as follows:

  ispeed ispeed baud; ospeed ospeed baud
- Characters for control functions are displayed as follows:

  `control_function = value`

  where `value` is either the character, or some visual representation of the character if it is nonprinting, or `<undef>` if the function is disabled.)
- The `stty` utility exits with one of the following values:
  - 0 if the terminal options were read or set successfully
  - >0 if an error occurred

B

Korn Shell Command-Line Editing Commands for emacs and vi

emacs and vi Command-Line Editing Commands

Overview

This appendix consists of tables containing command-line editing commands for the Korn shell. Each table contains commands for both the `emacs` and `vi` command-line editors. The commands are separated into tables by function.

The following is a list of tables to be used as a quick reference to subjects in this appendix.

- Table B–1, Fetching Commands from the History File
- Table B–2, Command Execution
- Table B–3, Moving the Cursor
- Table B–4, Deleting or Replacing Characters, Words, and Lines
- Table B–5, Entering Insert Mode
- Table B–6, Using Uppercase and Lowercase Letters
- Table B–7, Pathname Expansion or Completion
- Table B–8, Miscellaneous Commands

For `emacs`:

- A command is repeated Esc [count] times
- In general, `emacs` and `gmacs` are not case sensitive

For `vi`:

- The default for `vi` is insert mode. You must use Esc to get to command mode to recall commands
- A command is repeated [count] times
- The `vi` editor is case sensitive

If F11 does not function as Esc on your terminal, use Ctrl/ [ ].

Table B–1 Fetching Commands from the History File

To	From emacs type	From vi, type [1]
Recall previous command or [count] commands back	Esc [count] Ctrl/P	[count] k or [count] -
Recall next command or [count] commands ahead	Esc [count] Ctrl/N	[count] j or [count] +
Search backward through the history file to find the first occurrence of a command line containing `string`	Ctrl/R `string`	/ `string`
Search backward through the history file to find the first occurrence of a command line containing `string` at the beginning of the command line	Ctrl/R ^ `string`	/ ^ `string`
Search forward through the history file to find the first occurrence of a command line containing `string`	Esc 0 Ctrl/R `string`	? `string`
Search forward through the history file to find the first occurrence of a command line containing `string` at the beginning of the command line	Esc 0 Ctrl/R ^ `string`	? ^ `string`
Repeat the most recent / or ? directive	—	n
Repeat the most recent / or ? directive but in the opposite direction	—	N
Replace the current command with the oldest command from the history file, or, for `vi`, the command indicated by `number`	Esc <	[number] G
Replace the current command with the newest command from the history file	Esc >	—

[1]vi must be in command mode, Esc prior to fetching a command from the history file.

Table B–2 Command Execution

To	From emacs, type	From vi, type
Execute the current command (`vi` in any mode)	Return, Ctrl/J or Ctrl/M	Return, Ctrl/J or Ctrl/M
Execute the current command line, then fetch the next command	Ctrl/O [1]	—

[1]The vi command-line editor works on one command, even if it is multiline, whereas emacs works on one line at a time. For emacs multiline commands, use Ctrl/O to cause the first line to be processed and to fetch the second and subsequent lines. By default, *discard* is defined as Ctrl/O, so must be redefined to another key or undefined (stty discard 'undef') if Ctrl/O is needed.

Table B–3 Moving the Cursor

To	From emacs type	From vi, type
Move cursor right	Esc [count] Ctrl/F	[count] l or [count] Spacebar
Move cursor left	Esc [count] Ctrl/B	[count] h
Move cursor to beginning of line	Ctrl/A	0
Move cursor to first character on the line that is not a space or a tab	—	^
Move cursor to end of line	Ctrl/E	$
Move cursor to first space beyond end of current word	Esc [count] Esc F	—
Move cursor left to beginning of word	Esc [count] Esc B	[count] b; (B) [1]
Move cursor to end of word	–	[count] e; (E) [1]
Move cursor to beginning of next word	–	[count] w; (W) [1]
Move cursor to next character c	Ctrl/] c	[count] f c
Move cursor to previous character c	–	[count] F c
Move cursor to character before next character c after the current cursor position	–	[count] t c
Move cursor to character after next character c prior to the current cursor position	–	[count] T c
Repeat the most recent f, F, t, or T command. Previous count is ignored	–	[count] ;
Repeat the most recent f, F, t, or T command, but in the reverse direction. Previous count is ignored.	–	[count] ,
Move cursor to [count] character	—	[count] |

[1]Commands b, e, and w recognize the vi definition of a small word: letters, digits, and underscores, delimited at both ends by characters other than letters, digits, and underscores, or by the beginning or end of a line or file. Commands B, E and W recognize the vi definition of big word: a sequence of nonblank characters preceded and followed by blank characters or the beginning or end of a line or file.

Table B–4 Deleting or Replacing Characters, Words, and Lines

To	From emacs, type	From vi, type
Delete character at current cursor position	Esc [count] Ctrl/D	[count] x
Delete character before current cursor position	<X	[count] X
Delete the current word, placing deleted text in the save buffer	Esc D	dw
Delete from cursor to beginning of word, placing deleted text in the save buffer	Esc <X or Esc H	db
Delete entire command, regardless of cursor position, (vi in insert mode, stays in insert mode)	Ctrl/U	Ctrl/U
Delete entire command, regardless of cursor position, vi in command mode, enters insert mode	—	S
Delete entire command, regardless of cursor position, vi in command mode, remains in command mode	—	dd
Delete characters from cursor to end of line, placing deleted text in the save buffer (vi in command mode, stays in command mode)	Ctrl/K	d$ or D
Delete characters from cursor to end of line with vi in command mode, changes to insert mode	—	C
Set mark	Ctrl/Spacebar, Ctrl/@, or Esc Spacebar	—
Delete from cursor to mark, placing text in save buffer	Ctrl/W	—
Exchange cursor and mark	Ctrl/X Ctrl/X	—
Save characters between cursor and mark in save buffer. Do not delete from command.	Esc P	—
Insert characters from save buffer at current cursor position	Ctrl/Y [1]	p
Insert characters from save buffer before current cursor position	—	P
Replace current character with character c, or [count] characters with character c	—	[count] r c
Delete any characters between cursor position and new cursor position established by motion. Enter insert mode. A cc deletes entire command.	—	[count] c motion
Delete any characters between cursor position and new cursor position established by motion. Place deleted characters in save buffer.	—	[count] d motion
Yank the characters from the current cursor position to the new cursor position established by motion into the save buffer	—	[count] y motion

[1]susp cannot be defined as Ctrl/Y.

(continued on next page)

Table B–4 (Cont.) Deleting or Replacing Characters, Words, and Lines

To	From emacs, type	From vi, type
Yank the characters from the current cursor position to the end of the line into the save buffer	—	Y

Table B–5 Entering vi Insert Mode

To	From emacs, type	From vi, type
Enter insert mode after current cursor position	—	a
Enter insert mode at end of current command line	—	A
Enter insert mode at the current cursor position	—	i
Enter insert mode at beginning of current command line	—	I
Enter insert mode and replace characters beginning at current cursor position	—	R

Table B–6 Using Uppercase and Lowercase Letters

To	From emacs, type	From vi, type
Change current character to uppercase	Ctrl/C [1]	—
Change characters from current cursor position to end of word to uppercase	Esc C	—
Change characters from current cursor position to end of word (or count words) to lowercase	Esc [count] Esc L	—
Change case of count characters	—	[count] ~

[1]INTR cannot be defined as Ctrl/C.

Table B–7 Pathname Expansion or Completion

To	From emacs, type	From vi, type
Display the possible pathnames that match the current word as if an * were appended to the word.	Esc =	=
Append characters to the word under the cursor to complete the pathname of an existing file. Characters are appended up to the point where they would match more than one pathname.	Esc Esc	\
Append an * to the end of the current word. Perform pathname expansion on the word. If any pathnames match, replace the word with the pathnames that match the pattern.	Esc *	*

Table B–8 Miscellaneous Commands

To	From emacs, type	From vi, type
Redraw line	Ctrl/L	—
Transpose characters: For emacs, transpose current character with next character; For gmacs, transpose previous two characters.	Ctrl/T	—
Search the alias list for an alias with the name _letter. If it is found, place its value in the command at the cursor position.	Esc letter	@ letter
Undo the last command that modified the text on the current command line.	—	u
Undo all changes to the current command line since command mode was entered.	—	U
Append the last word, our [count] word from the previous command to the end of the present command. User adds a space for emacs or gmacs, vi adds its own space.	Esc [count] . or Esc [count] _	[count] _
Escape the next character. Used for entering control characters on the command line. Will not escape control characters for I, J, M, O, Q, S, V, or Z.	\ [control-character]	—
Escape the next character. Used for entering control characters on the command line. Will not escape kill or erase.	—	Ctrl/V [control-character]
Escape the next kill or erase character.	—	\ [control-character]
Place a # at the beginning of the command, and place the command into the history file. Do not execute the command.	—	#
Display the version of Korn shell.	Ctrl/V	—

C

Mail Options Using the set Command

Mail Options Using the set Command

Binary and Value Options

Below are listed the binary and value options available with the `set` command when using the `Mail` utility. The `set` command is discussed in Chapter 11, Communicating with Other Users. The options are listed in alphabetical order. To avoid confusion, please note that the options are either all lowercase letters or all uppercase letters.

- `EDITOR`

  Valued option which defines the pathname of the text editor to be used in the `edit` command and `~e`. If not defined, a standard editor is used.

- `SHELL`

  Valued option which gives the pathname of your shell. Used for the `!` command and `~!` escape. In addition, this shell expands file names with shell metacharacters like * and ? in them.

- `VISUAL`

  Valued option which defines the pathname of your screen editor for use in the `visual` command and `~v` escape. A standard screen editor is used if you do not define one.

- `append`

  Binary option which causes messages saved in mbox to be appended to the end rather than prepended. Normally, the Mail utility stores messages in mbox in the same order that the system puts messages in your system mailbox. By setting `append`, you are requesting that mbox be appended to regardless. It is quicker to append.

- `ask`

  Binary option which causes mail to prompt you for the subject of each message you send. If you respond with simply a newline, no subject field is sent.

- `askcc`

  Binary option which causes you to be prompted for additional carbon copy recipients at the end of each message. Responding with a new line shows your satisfaction with the current list.

- `autoprint`

  Binary option which causes the `delete` command to funchtion like `dp`. After deleting a message, the next one is typed automatically. Useful for quickly scanning and deleting messages in your mailbox.

- `crt`

Valued option used as a threshold to determine how long a message must be before `more` is used to read it. `more` prints only one screenful of the message at a time allowing the user to press Spacebar when ready to continue.

- `debug`

  Binary option which causes debugging information to be displayed. Same as using the `-d` command-line option.

- `dot`

  Binary option which, if set, causes mail to interpret a period alone on a line as the terminator of a message you are sending.

- `escape`

  Valued option which allows you to change the escape character used when sending mail. Only the first character of the `escape` option is used, and it must be doubled if it is to appear as the first character of a line of your message. If you change your escape character, then ~ loses all its special meaning, and no longer needs to be doubled at the beginning of a line.

- `folder`

  Directory name to use for storing folders of messages. If this name begins with a slash (/) mail considers it to be an absolute pathname; otherwise, the folder directory is found relative to your home directory.

- `hold`

  Binary option which causes messages that have been read (but not manually dealt with) to be held in the system mailbox. Prevents such messages from being automatically swept into your mbox.

- `ignore`

  Binary option which causes RUBOUT characters from your terminal to be ignored and echoed as `@s` while you are sending mail. RUBOUT characters retain their original meaning in mail command mode. Setting the `ignore` option is equivalent to supplying the `-i` option on the command line.

- `ignoreeof`

  Option related to `dot` which makes mail refuse to accept a Ctrl/D as the end of a message. Also applies to mail command mode.

- `keep`

  Option which causes mail to truncate your system mailbox instead of deleting it when it is empty. Useful if you choose to protect your mailbox, which you would do with the shell command:

  ```
  chmod 600 /usr/spool/mail/you
  ```

where you is your login name. If you do not do this, anyone can read your mail, although most people will not.

- keepsave

  When you save a message to a folder or a file, mail usually discards its copy when you quit. To retain all saved messages, set the keepsave option.

- metoo

  When sending mail to an alias, mail makes sure that if you are included in the alias, that mail will not be sent to you. This is useful if a single alias is being used by all members of the group. If, however, you want to receive a copy of all the messages you send to the alias, you can set the binary option metoo.

- noheader

  Binary option which suppresses the printing of the version and headers when mail is first invoked. Same as using -N on the command line.

- nosave

  Normally, when you abort a message with two RUBOUTs, mail copies the partial letter to the file dead.letter which it creates in your home directory. Setting the binary option nosave prevents this.

- quiet

  Binary option which suppresses the printing of the version when mail is first invoked.

- record

  Valued option which can be set to the name of a file to save your outgoing mail. Each new message you send is appended to the end of the file.

- screen

  When mail initially prints the message headers, it determines the number to print by looking at the speed of your terminal. The faster your terminal, the more it prints. The valued option screen overrides this calculation and specifies how many message headers you want printed. This number is also used for scrolling with the z command.

- toplines

  Valued option which defines the number of lines that the top command will print out instead of the default five lines.

- verbose

  Binary option which causes mail to invoke sendmail with the -v option, causing it to go into verbose mode and announce expansion of aliases, and so forth. verbose option is equivalent to invoking mail with the -v option.

Glossary

&&

Logical operator (**AND**) used in conditional command execution. The command after the **&&** operator executes only if the command before the **&&** operator is successful.

||

Logical operator (**OR**) used in conditional command execution. The command after the || operator executes only if the command before the || operator is not successful.

absolute pathname

See full pathname.

alias

(1) A Korn shell name used to abbreviate one or more commands. (2) A name used in the Mail utility file `.mailrc` to define a list of recipient user names for a mail message.

argument

Additional information that the shell needs to perform your command. For example, the file name `demo1` is the argument in the command: `cat demo1`.

awk

The **A**ho, **W**einberger, **K**ernighan report generator that processes ASCII text files and generates reports involving selective retrieval and string manipulation.

background process

A process that is running without having an assigned terminal and therefore without interfering with the normal command-line entries is running in the "background." A background process is submitted by adding the ampersand character (&) at the end of the command.

block

A unit of allocation, in bytes of data, in the file system.

block device

A random-access, mass-storage device that supports a block-oriented interface; for example, a disk drive.

block I/O

I/O to a block device.

Bourne shell

A command-language interpreter originally developed by Steven Bourne that contains a programming language and a command interpreter. Versions are available on nearly all UNIX systems. See also shell.

byte

A unit of measure applied to data; usually the smallest unit of addressable memory. A byte is almost always 8 bits.

C shell

A command-language interpreter that contains a programming language with C-like syntax. Contains more interactive features but fewer scripting features than the Bourne shell. See also shell.

cascade menu

A submenu or menu-within-a-menu, that appears when you highlight a menu function that has an arrow next to its name. Cascade menus are used to group similar functions together.

character device

A device that provides either a character-stream oriented I/O interface, or an unstructured (raw) interface.

character I/O

I/O to a character device.

child process

A process created by an original process called the parent process. The child process is an exact duplicate of the parent process except for the PID and the parent PID.

CISC

Complex Instruction Set Computer.

click

To press and release a mouse button.

command history

A buffer or file containing the most recently executed shell commands. Another way of referring to the history buffer.

command-line editing

The ability to edit a Korn shell command on the command line.

command-line editor

The built-in Korn shell editor selected to edit shell commands on the command line, either `gmacs`, `emacs`, or `vi`.

command-line prompt

Indicates that the computer is ready to accept your commands. Each terminal emulation window has a command-line prompt. Usually the command-line prompt is a `$` for the Bourne and Korn shells or a `%` for the C shell.

command recall

The ability to recall previously executed shell commands.

conditional execution

Command execution based on the success or failure of a previous command.

conditional statement

A statement that is executed based on the evaluation of an expression. For example, `if ( expression ) statement`.

console

The console is the terminal teletype device to which error messages are written by the kernel in the event of serious system errors.

construct

Special commands that alter the flow of control of a shell script to execute commands conditionally or repeatedly.

crontab file

The system examines the `crontab` file every minute while the system is up, and automatically executes commands at specified dates and times according to the instructions in the file. It is a handy place to add those commands you want to have run periodically.

cron daemon

A system program that automatically executes commands at specific times and dates as specified in the `crontab` file. The `cron` program is started up at system initialization and runs continuously until the system is brought down. Each user may submit one `crontab` file which may have multiple entries.

daemon

A system program that is always running. For example, `cron` and the clock.

debugging

The process of finding and correcting errors. For example, using a trace statement `set -x` command, embedded at the beginning of a shell script, to test each line of the script.

detached process

A background command that continues running after you log out.

device

A hardware peripheral connected to the CPU. Examples are disk drives, tape drives, and terminals.

device special file

A file through which processes can access hardware devices on a system.

directory

A file that points to other files or directories. Directories can be nested to any depth.

disk partition

A contiguous region of a disk drive, used as a swap area or to hold a file system.

double-click

To press and release a mouse button twice in rapid succession.

drag

To press and hold down a mouse button while moving the mouse on the desktop (and the pointer on the screen). Typically, dragging is used with menu selecting, and window moving and resizing.

ed

A standard text line editor.

emacs

(1) An extendible interactive full-screen editor available on a wide variety of operating systems. DEC OSF/1 uses the `GNU emacs` version developed by the Free Software Foundation.
(2) One of the Korn shell built-in command-line editors which is a subset of the `emacs` editor.

executable

A file that can be invoked by name. It can contain machine language instructions or it can be a text file of shell commands that has been given execute permission by the `chmod` command.

exit status

Every program or command returns the status of executing the command to the shell. This value, either success or failure, is stored in the shell variable **?** and can be used to conditionally execute another command.

exporting a variable

A shell sending the value of one of its variables to a shell script or command that it invokes. See also global variable.

field

A group of characters. One or more fields make up a record.

file

A named collection of information. On UNIX systems, a file is a stream of bytes; there are no predetermined, system-supported file or record structures.

filename

A group of characters that are used to refer to a file. File names can be up to 255 characters long and can include letters, numbers, periods and underscores.

file permission

The sum of all system processes and procedures designed to inhibit unauthorized access, modification, or destruction of a file.

file system

An initialized partition on a disk; a collection of files.

filter

A command used to read standard input and print the result to standard output.

flow control

A programming feature that allows a program to stop, start, or loop through a group of statements to reach a desired result. See also loop.

full (absolute) pathname

Indicates the path of a directory starting from the root directory. See also relative pathname.

foreground process

Executing a command or script file such that the shell is waiting for it to finish before prompting for another command. Foreground processes can be stopped by entering commands or pressing control keys at the keyboard.

fork

A system call that causes creation of a new process. The new process is called a child process and is an exact copy of the parent process, with the exception of the ID number.

GID

A unique integer assigned to a set of users, Group ID.

global variable

A predefined or user-defined programming construct that allows all processes created by a user to see, and be able to use, the variable. See also exporting a variable and shell variable.

gmacs

One of the Korn shell built-in command-line editors. It is the same as `emacs` except in response to Ctrl/T. The `gmacs` command-line editor is a subset of the `emacs` editor.

GNU emacs

An extendible interactive full-screen editor developed by the Free Software Foundation. It is commonly referred to as `emacs`.

grep

The **g**lobal **r**egular **e**xpression **p**rinter. Searches files for every occurrence of a specific word, phrase, or pattern.

group

A set of users assigned a unique group name and GID in the file `/etc/group`.

hard link

A link to a file that is indistinguishable from the original directory entry (which is itself a hard link to the inode for the file). Any changes to the file are independent of the name used to reference the file. Hard links cannot exist between files on different file systems. See also symbolic link.

hidden file

Files that have names that start with a period (.). These files do not appear in a listing of files unless the `-a` option is specified with the `ls` command.

hierarchical file system

The storage of files in the form of a tree structure. All directories and files branch from one location, the root (/).

history buffer

See command history.

home directory

The directory in which the system automatically places you when you log in. This is specified in the `/etc/passwd` file.

icon

A small, graphical representation of a window or displayed object.

in-line comment

A comment on the same line as a command.

init process

An operating system process that spawns login processes for each terminal on a machine. The `init` process begins execution when the system starts up and is responsible for creating login processes to wait for input from terminals. All user processes are child processes of the `init` process.

Input/Output (I/O)

The transfer of data between memory and peripheral devices.

interprocess communication (IPC)

The transfer of data between processes.

interrupt

An event external to the currently executing process that causes a change in the normal flow of instruction execution. Interrupts are often generated by hardware devices external to the CPU.

job

One process, or a collection of processes, that get started when a command is issued.

job number

The number associated with a process that is under job control. Use the job number to control the process.

kernel

The core of the operating system responsible for all machine-level work.

Korn shell

A command-language interpreter that enhances features of the C shell and the Bourne shell. See also shell.

label

A location marker in a shell script consisting of a name followed by a colon (:).

link

A directory entry referring to a file. A file may have several links to it. There are two kinds of links: hard links and symbolic (soft) links. See also hard link and symbolic link.

literal string

A string of characters enclosed in quotation marks so that they will be interpreted literally.

login shell

The default shell executed each time you log in. This is specified in the `/etc/passwd` file.

logical operator

An operator that uses logical (true/false) operands. Binary logical operators are **&&** (logical **AND**) and **| |** (logical **OR**).

loop

A sequence of instructions that is executed repeatedly until a specified condition is satisfied.

man pages

Online reference information for commands and system routines accessed using the `man` command.

mbox

File containing the personal mail messages for a user of the UNIX personal mail utility, stored in the default login directory of the user.

metacharacter

Nonalphanumeric characters that have special meaning to the shell or certain commands like `grep`. Metacharacters are accepted literally and not given special meaning when they are escaped using a backslash or contained within quotation marks.

Motif

The graphical user interface (GUI) defined for the Open Software Foundation for use in their windowing environment and portable to other operating systems.

nesting

The placement of one set of instructions inside another, such as a loop within a loop.

parent process

A process that creates another process.

password

A word or string of characters that, in conjunction with other information, such as the user name, uniquely confirms a user's identity.

pathname

A file name. The pathname tells the shell where to look for the file in the directory tree. Files may be referenced by full pathnames or relative pathnames. See also full and relative pathnames.

permission

For file access, the authorization for user, group, or other to read, write, or execute the file.

PID

A unique identification number assigned to a process at its creation; Process ID.

pipe

A connection that allows one command to get its input from another.

pipeline

A sequence of one or more shell commands separated by a pipe (|) symbol. The standard output of a command is connected to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate.

pointer

Sometimes called the mouse cursor, the pointer shows the location of the mouse. The pointer's shape may change depending on its screen location. It may also become a stop-watch to show when processing is occurring.

POSIX

The portable operating system interface for computer environments standards group, P1003, established by the IEEE. Their first established standard was the operating system interface.

print spooler

The line printer daemon, `lpd`, responsible for transferring files to be printed to the spooling directory, printing, displaying and removing jobs from the print queue.

process

Running program (or command). UNIX assigns each process a unique number when it is started, the process identification number (PID). The PID numbers can be used to terminate individual processes. The user identification number (UID) is also attached to every process by the operating system. When you enter any command, that command is executed as a subordinate process, or "child process" of a "parent process." Every process on the system has a parent except the `init` process. All processes that a user creates are children of the login shell.

process environment

Information necessary to execute and control a process including: PID, open files, working directory, file creation mask (umask), GID, signal-action settings and shell variables.

process priority

A parameter used by the kernel to schedule the execution of processes. The priority for a process changes dynamically according to the operation of the process, and the value of the nice parameter.

pull-down menu

A menu accessed by clicking on a window menu item. Pull-down menus may display as a pop-up menu.

push button

A control that causes an immediate action. To select (push) a button on the screen, point to it then click the mouse button.

radio button

A small diamond-shaped or square push button on the screen that allows you to make on/off selections with the mouse.

raw I/O

Unstructured or character I/O; does not use the buffer cache.

record

A group of fields treated as one unit.

redirection

Designating the source or output of standard input, standard output, or standard error to somewhere other than the keyboard or terminal screen. Redirection symbols are >, >>, and <.

regular expression

A method of specifying patterns of character strings. They allow you to "describe" a set of strings instead of having to list every member of the set. Strings that fit the specified pattern **match** the regular expression.

regular files

A file that contains whatever information you place in it with no particular structure.

relative pathname

The path of a directory starting from the current directory.

restore

To change an icon back into its original window.

RISC

Reduced Instruction Set Computer.

root

(1) The name of the user account with UID=0.
(2) The file system containing the root (/) directory.

root directory

The top level of the directory hierarchy (/).

script file

Also known as a shell program or shell procedure, it is a file containing ordinary system commands.

shell

The shell is a command-language interpreter that allows the commands you enter to operate a UNIX system. Every user has his/her own copy of the shell. The DEC OSF/1 system includes three different shell interpreters: the C shell (`csh`), the Bourne shell (`sh`), and the Korn shell (`ksh`).

shell script

A text file containing a series of shell commands.

shell variable

A named storage location that contains a value. A value is assigned to a variable and you can return a variable value. For example, use `set` to display the contents of the Korn shell built-in variables such as **HOME**, **PATH**, **SHELL**, and **USER**.

signal

A message, represented by a numerical value, which indicates a specific action. A signal is sent to a process to change the action of a process. Typical signals are: **QUIT**, **HUP**, **TERM**, and **KILL**.

sort key

Refers to the portion of a record that is compared to other records when a file is sorted.

special file

A file that controls system input/output.

spool directory

The directory that contains the files queued to be printed. There is a spool directory for each printer.

spooling

The process of copying files to be printed to a spool directory. See also spool directory.

standard error

The device where error messages are displayed. By default, it is the terminal screen, but it can be redirected.

standard input

The device from which a program reads its normal input. By default, it is the terminal keyboard. Standard input can be redirected.

standard output

The device to which a program writes its normal output. By default, it is the terminal screen. Standard output can be redirected.

statement

The text that constitutes a source program or a command in a shell script.

sticky bit

A file permission bit used for frequently executed programs to retain swap space even when it is not running. When set on a shareable, executable file, the file will not be removed from the swap area. When set on a directory, only file owners, the directory owner, or the superuser can remove and rename files in the directory.

SUID

Sets effective user ID of process equal to the owner's user ID, mode 4000.

superuser

An account on a UNIX operating system which has special privileges, including the right to boot (start up) the system and to stop it, and set up user accounts and special user groups. The **root** account has superuser privileges, and is often reserved for the system manager.

suspended

Temporarily stopped job. It can be restarted in the foreground or background, or terminated.

symbolic link

A file that contains a pathname. Symbolic links can point to directories and to files on other file systems. Also called soft link. See also hard link.

system manager

The person responsible for installing, configuring and maintaining the operating system and associated file systems.

terminal

In computer systems, a device used to enter and receive data interactively from a computer.

tilde

A special character (~) used in the Mail utility to do mail editing commands within mail, and in the Korn shell for online editing and directory pathname expansion.

UID

The user ID number; each process has a user ID associated with the process. Your UID is set when you log in and is inherited by all processes you create.

user

For file permissions, the owner of the file.

username

The name a user types to log in to the system.

utility programs

Programs included as part of the operating system such as communications software, editors, text-formatting commands, file manipulation commands, a mail utility, and special program development tools, and much more.

vi

(1) A full-screen editor developed at Berkeley.
(2) One of the Korn shell built-in command-line editors, which is a subset of the `vi` editor.

working directory

The current directory associated with a process. The process uses the current directory to translate relative file pathnames.

