Compaq Tru64 UNIX V5 Utilities and Commands

Student Guide

NAT001/0999

COMPAQ

Tru64 UNIX V5 Utilities and Commands

Course Guide

NAT001/0999

COMPAQ

Notice

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

COMPAQ COMPUTER CORPORATION SHALL NOT BE LIABLE FOR TECHNICAL OR EDITORIAL ERRORS OR OMISSIONS CONTAINED HEREIN, NOR FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES RESULTING FROM THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL.

The software described in this guide is furnished under a license agreement or nondisclosure agreement. The software may be used or copied only in accordance with the terms of this agreement.

Compaq Tru64 UNIX V5 Utilities and Commands
Course Guide
August 1999

Contents

About This Course

1 Getting Started with Tru64 UNIX

2 Using CDE

3 Customizing Your Workspace

4 Introducing File System Concepts

5 Managing Files with CDE

6 Managing Files with CLI

7 Introducing UNIX Shells

8 Using the vi Editor

9 Redirecting, Filtering, and Piping

10 Controlling Processes

11 Communicating with Other Users

12 Using TCP/IP Networks

13 Printing Using CDE

14 Printing Using CLI

15 Using Disks and Tapes

16 Recalling and Editing Korn Shell Commands

17 Introducing Korn Shell Scripts

18 Test

A stty Command Arguments

B Mail Options Using the set Command

C Korn Shell Command Line Editing Commands

D Using the emacs Editor

Index

Figures

Tables

Examples

About This Course

About This Course

Introduction

This section describes the contents of the course, suggests ways in which you can most effectively use the materials, and sets up the conventions for the use of terms in the course. It includes:

- Course description — a brief overview of the course contents
- Target audience — who should take this course
- Prerequisites — the skills and knowledge needed to ensure your success in this course
- Course goals and nongoals — what skills or knowledge the course will and will not provide
- Course organization — the structure of the course
- Course map — the sequence in which you should take each chapter
- Chapter descriptions — brief descriptions of each chapter
- Time schedule — an estimate of the amount of time needed to cover the chapter material and lab exercises
- Course conventions — explanation of symbols and signs used throughout this course
- Resources — manuals and books to help you successfully complete this course

Course Description

Tru64 UNIX V5 Utilities and Commands is an entry-level course. It provides the fundamental skills needed to effectively use a UNIX type operating system, including Tru64 UNIX, UNIX System V, or a BSD-based system.

The concepts presented here are suitable for users of any UNIX environment, and prepare students to use the Common Desktop Environment (CDE) and the Korn shell on any UNIX based operating system. The examples, exercises, and solutions are based on a Tru64 UNIX Version 5.0 system using CDE and the Korn shell.

Place in Curriculum

This course is the basic user course. It is also the prerequisite for the UNIX system administration curriculum and the UNIX programming curriculum.

Target Audience

The concepts presented in this course are suitable for users of any UNIX environment, for service technicians and customers.

Prerequisites

There are no course prerequisites.

Course Goals

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

- Log in and out
- Invoke desktop applications
- Set up your workspace
- Manage files and directories
- Use shell commands
- Use an editor on the system
- Use redirection, filters, piping, and the `awk` command
- Manage software processes
- Communicate with other users
- Transfer files and execute commands across TCP/IP networks
- Print files and manage your print jobs
- Write files to, and extract files from, tapes
- Write introductory-level Korn shell scripts

Nongoals

This course does not cover the following topics:

- Programming tools or languages
- Detailed shell programming
- System administration

Taking This Course

Course Organization

This Course Guide is divided into chapters designed to cover a skill or related group of skills required to fulfill the course goals. Illustrations are used to present conceptual material. Examples are provided to demonstrate concepts and commands.

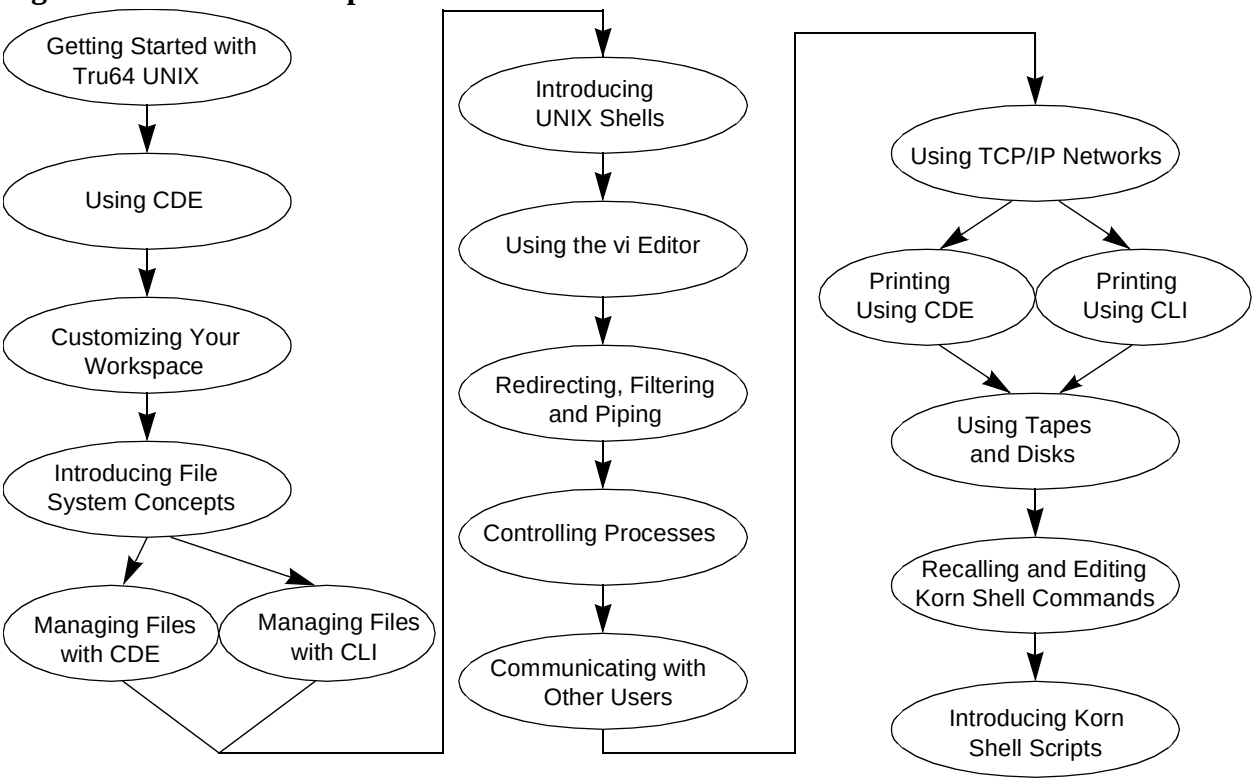
In this course, each chapter consists of:

- An *introduction* to the subject matter of the chapter.
- One or more *objectives* that describe the goals of the chapter.
- A list of *resources*, or materials for further reference. Some of these manuals are included with your course materials. Others may be available for reference in your classroom or lab.
- The *text* of each chapter, which includes outlines, tables, figures, and examples.
- The *summary* highlights the main points presented in the chapter.
- The *exercises* enable you to practice your skills and measure your mastery of the information learned during the course.

Course Map

The Course Map shows how each chapter is related to other chapters and to the course as a whole. Before studying a chapter, you should master all of its prerequisite chapters. The prerequisite chapters are depicted before the following chapters on the Course Map. The direction of the arrows determines the order in which the chapters should be covered.

Figure 0-1: Course Map



Chapter Descriptions

A brief description of each chapter is listed below.

- **Getting Started with Tru64 UNIX**: Contains a brief history of UNIX, a basic description of the Tru64 UNIX operating system, and an introduction to the Common Desktop Environment. Logging in and out, changing passwords and using the Tru64 UNIX documentation is also included.
- **Using CDE**: Describes the CDE Front Panel, how to get help, and how to use the clock, text editor, calendar, and terminal applications in CDE.
- **Customizing Your Workspace**: Describes how to customize a workspace using the Style Manager. For the command line interface, the chapter describes how to set up a terminal.
- **Introducing File System Concepts**: Describes the directory tree structure, types of files and file naming conventions used by the Common Desktop Environment and command line interface; also introduces file access permissions.
- **Managing Files with CDE**: Discusses using File Manager to access, list, find, create, delete, copy, move, rename, link, and control access to the contents of files and directories.
- **Managing Files with CLI**: Discusses basic commands to access, list, find, display, create, delete, copy, move, rename, link, and control access to the contents of files and directories.

- **Introducing UNIX Shells**: Describes the purpose of a shell, the shells available with Tru64 UNIX, and the general features of each of these shells. The Korn shell variables and startup files are also discussed at an introductory level.
- **Using the vi Editor**: Introduces the editors available with Tru64 UNIX, and provides a detailed description of the `vi` text editor.
- **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.
- **Controlling Processes**: Covers the concept of processes and how to manage them. Understanding the information in this chapter 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.
- **Communicating with Other Users**: Teaches the CDE Mailer application and in CLI, teaches commands that show who is logged in to the system. The chapter presents an introductory overview of the mail features of the Tru64 UNIX operating system with additional discussion of the `talk` and `write` utilities.
- **Using TCP/IP Networks**: Describes how to exchange files with, execute commands on, and how to log in to remote systems.
- **Printing Using CDE**: 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 using the Common Desktop Environment.
- **Printing Using CLI**: 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 using the command line interface.
- **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.
- **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.
- **Introducing Korn Shell Scripts**: Provides an introduction to the Korn shell script as well as its creation and execution. This chapter also covers features of the Korn shell programming environment.

Time Schedule

The amount of time required for this course depends on each student's background knowledge, experience, and interest in the various topics. Use the following table as a guideline.

Table 0-1: Course Schedule

Day	Course Chapter	Lecture/ Reading Hours	Lab/ Exercise Hours
1	Getting Started with Tru64 UNIX	1	.5
	Using CDE	1	.5
	Customizing Your Workspace	1	.5
	Introducing File System Concepts	.5	.5
	Managing Files with CDE	1	.5
2	Managing Files with CLI	1	1
	Introducing UNIX Shells	1.5	.5
	Using the vi Editor	1.5	1
3	Redirecting, Filtering, Piping	2	1
	Controlling Processes	1	1
	Communicating with Other Users	1	1
4	Using TCP/IP Networks	1	.5
	Printing Using CDE	.5	.5
	Printing Using CLI	.5	.5
	Using Tapes and Disks	1	1
5	Recalling and Editing Korn Shell Commands	1.5	1
	Introducing Korn Shell Scripts	2	1.5

Course Conventions

This book uses the following conventions.

Table 0-2: Course Conventions

Convention	Description
keyword	Keywords and new concepts are displayed in this typeface.
`example`	Examples, commands, options, and pathnames are displayed in this typeface.

Table 0-2: Course Conventions (Continued)

Convention	Description
`command(n)`	Cross-references to command documentation include the section number in the reference pages. For example, `fstab`(5) means `fstab` is referenced in Section 5.
`$`	A dollar sign represents the user prompt for the Bourne and Korn shells.
`%`	A percent sign represents the user prompt for the C shell.
`#`	A number sign represents the superuser prompt.
`Ctrl/D`	Press and hold the Ctrl key while pressing the D key.
`Esc`	Press the Esc key. As an alternate, press the F11 key or Ctrl/[.
`.` `.` `.`	In examples, an ellipsis indicates that not all lines in the example are shown.
`[ ]`	In syntax descriptions, brackets indicate items that are optional.
`variable`	In syntax descriptions, *italics* indicate items that are variable.

Resources

For more information on the topics in this course, see the following:

- Tru64 UNIX Reference Pages
- *Common Desktop Environment: User's Guide*
- *CDE Companion*
- *Command and Shell User's Guide*

If students want to learn more about the Korn shell, the following texts are recommended for further study. They are not provided in this course.

- *The Kornshell Command and Programming Language*

  Morris I. Bolsky and David G. Korn
  Prentice Hall, Englewood Cliffs, NJ 07632 (c) 1989
  ISBN 0-13-516972-0

- *Kornshell Programming Tutorial*

  Barry Rosenberg
  Addison-Wesley Publishing Co., Reading, MA 01867 (c) 1991
  ISBN 0-201-56324-X

If you have OpenVMS system experience, see *UNIX for VMS Users* by Philip E. Bourne

1

Getting Started with Tru64 UNIX

Unit Overview

Introduction

This unit is an overview. It outlines the history of UNIX. It looks at the Tru64 UNIX operating system as a whole, and the Tru64 UNIX features and characteristics in general.

This unit discusses:

- How to log in and out of the system
- How to change passwords
- Organization of the Tru64 UNIX documentation

Objectives

To get started with Tru64 UNIX, you should be able to:

- Describe the history of UNIX and its relationship to Tru64 UNIX
- Briefly describe the features of the UNIX operating system
- Describe the operating system user interfaces
- Log in and log out of the system using either the Common Desktop Environment (CDE) or the command line interface (CLI)
- Change your password

Resources

For more information on the topics in this unit, see the following:

- *Tru64 UNIX Technical Overview*
- *Common Desktop Environment: User's Guide*, Chapters 1 and 2
- *CDE Companion*, Chapter 1
- *Tru64 UNIX Documentation Overview, Glossary, and Master Index*

Introducing the Tru64 UNIX Operating System

Overview

The Tru64 UNIX operating system is a multiuser, interactive, general-purpose operating system. It allows you to run more than one program at a time while performing other tasks simultaneously.

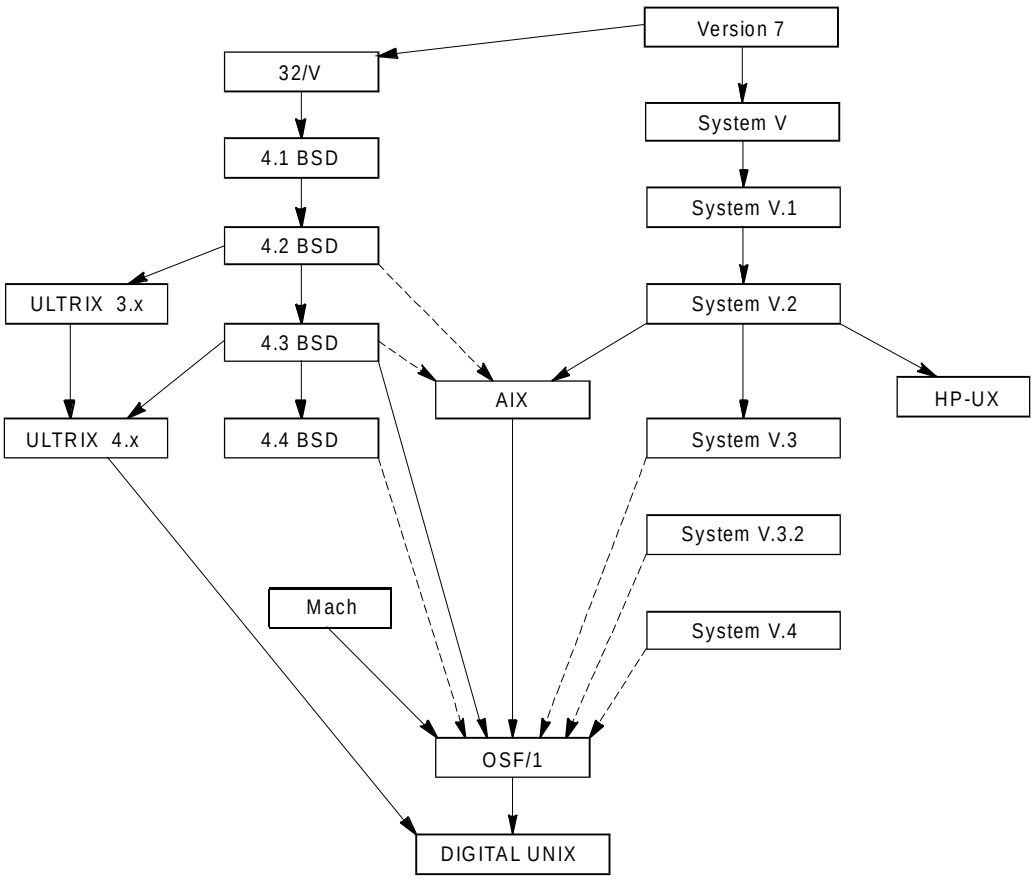
Creators of UNIX

- UNIX was created in 1969 by Ken Thompson at Bell Laboratories, originally written in PDP-7 assembly language.
- Thompson was joined by Dennis Ritchie, developer of the "C" programming language. Thompson and Ritchie rewrote UNIX in C.
- UNIX was one of the first operating systems written in a high-level language.

UNIX Evolution

The following illustration shows the evolution of UNIX since 1969.

Figure 1-1: UNIX Evolution



A Standard UNIX

The Tru64 UNIX operating system is a 64-bit advanced kernel architecture. It is based on Carnegie-Mellon University's Mach V2.5 kernel design with components from Berkeley Software Distribution (BSD) 4.3 and 4.4, UNIX System V, and other sources. Tru64 UNIX is Compaq Computer Corporation's implementation of the Open Software Foundation OSF/1 R1.0, R1.1, and R1.2 technology, and the Motif graphical user interface and programming environment. The full features of the X Window System, Version 11, Release 6 (X11R6) from the X Consortium Inc. are fully supported. Selected features of Release 6.1 (X11R6.1) are also supported.

Tru64 UNIX complies with numerous other standards and industry specifications, including the X/Open XPG4 and XTI, POSIX, FIPS, and System V Interface Definition (SVID). The Tru64 UNIX operating system is compatible with Berkeley 4.3 and System V programming interfaces. Tru64 UNIX conforms with the OSF Application Environment Specification (AES) which specifies an interface for developing portable applications that will run on a variety of hardware platforms. For more information, see the Tru64 UNIX Operating System Software Product Description.

Description

The Tru64 UNIX operating system is a multiuser/multitasking, 64-bit, advanced kernel architecture. It supports symmetric multiprocessing (SMP), and a real-time user and programming environment. It provides a choice of user interfaces, including graphical and command line interfaces.

Tru64 UNIX 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

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

Tru64 UNIX Features

Tru64 UNIX provides symmetric multiprocessing (SMP), real-time support, and numerous features to help application programmers develop applications that use shared libraries, multithread support, and memory mapped files.

- Common Desktop Environment (CDE)

  Tru64 UNIX provides CDE graphical user interface, the new industry standard UNIX desktop.

- Motif and X Window System

  The Motif and X Window System provides a programming environment with an extensive set of libraries and tools for use by application developers.

- Programming environment

  Tru64 UNIX provides alternative programming environments and the programming environment features of both System V and BSD UNIX systems.

- Real-time support

  Tru64 UNIX provides a real-time user and programming environment conforming to the POSIX 1003.1b-1993 standard for real-time.

- File systems

  The Tru64 UNIX hierarchical file system architecture is based on OSF/1 Virtual File System (VFS) and supports a number of file systems:

  — POLYCENTER Advanced File System (AdvFS)

  — UNIX File System (UFS)

  — Network File System (NFS)

  — Memory File System (MFS)

  — ISO 9660 Compact Disk File System (CDFS)

  — File-on-File Mounting File System (FFM)

  — File Based Pipes

  — `/proc` File System

- Internationalization and localization

  Tru64 UNIX provides an internationalization environment and tools as well as the localization features for developing and executing internationalized software without the need to reengineer the application.

- System management

  Tru64 UNIX provides a suite of graphical configuration and administrative applications for managing Tru64 UNIX systems.

- Security

  Tru64 UNIX exceeds the requirements of the C2 evaluation class of DoD 5200.28-STD "Trusted Computer System Evaluation Criteria" including extended passwords, audit and Access Control Lists (ACLs).

Describing the User Interface

Overview

The Tru64 UNIX operating system provides two methods for users to interact with the operating system:

- Graphical user interface (GUI)

  For workstation users or intelligent terminals with bit-mapped displays

- Command line interface (CLI)

  For character cell terminal users or a terminal emulator

It is important to understand the difference between the interfaces. You probably will use both to work on Tru64 UNIX.

Common Desktop Environment

The default graphical user interface for the current version of Tru64 UNIX is the Common Desktop Environment (CDE). CDE is an industry standard, graphical user interface (GUI) which provides a common user interface across multivendor platforms. CDE development was a collaborative effort including technology for HP, IBM, SunSoft, and Novell. It is based on the X Window System (X11R6) and Motif R1.2.3.

When you log in, CDE transforms your display into a desktop backdrop. On the backdrop, you have the Front Panel controls which provide access to various applications and tools. The desktop has more than one workspace. You can configure each workspace to fit the requirements of the work you want to do.

Command Line Interface

Before there were GUIs, the user interface was a command line interpreter (CLI). To interact with the operating system using a CLI, you type commands when the system prompts for input. Each command executes and the system prompts you for more input. Commands generally execute sequentially. You must wait for one command to finish before executing another.

UNIX refers to CLIs as **shells**. Many different shells are available on the UNIX system. Typical shells are:

- C shell
- Korn shell

When to Use a CLI Interface

Two instances when you must use the CLI interface are:

- When using a character cell terminal or dumb terminal

- When accessing the system through a terminal emulator window

DECwindows

Prior to DIGITAL UNIX Version 4.0, the default user interface was DECwindows. DECwindows is based on the OSF Motif interface developed by the Open Software Foundation and released in 1989. Motif is supplied by OSF to hardware vendors, software developers, and end-user organizations.

With DECwindows, you can use more than one window to run applications using mouse buttons, menus and the keyboard. The most noticeable difference between DECwindows and CDE is in the look and feel of the 3-dimensional visuals such as window frames and pushbuttons.

The DECwindows graphical user interface is available as an optional GUI with Tru64 UNIX through the CDE login manager. The DECwindows applications are also available.

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.

Enter information in response to system requests or prompts. The prompts for logging in and system response differ depending upon the type of terminal being used.

NOTE

UNIX is sensitive to uppercase and lowercase letters and treats them as different characters.

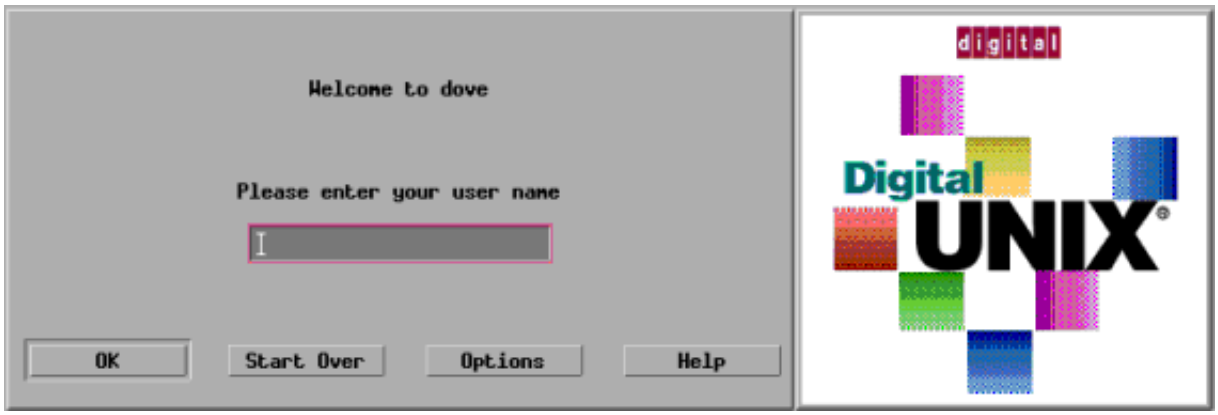
Two types of commands are used in this course.

Enter	Type the command and press Return
Type	Type the characters and do **not** press Return

Logging In Using CDE

To begin a desktop session, log in to CDE. To end the session, log out. The work you do between these events is managed by a CDE application called the Session Manager. On logging out, the Session Manager remembers the state of your session, and on your next login recreates that state in a new session.

Figure 1-2: Login Screen



When the Login Manager displays the login screen, you have the following options:

- Log in to a CDE session.
- Log in to a failsafe session.

  A **failsafe session** provides a single terminal window so you can enter commands using the CLI user interface.
- Log in using a different language.

Now Try This!

To log in and exit CDE, perform the following steps:

1. Type your login name and press Return or click OK.
2. Type your password and press Return or click OK.
3. Log out by clicking the Exit button on the Front Panel.

Need Help?

When you successfully log in, the Session Manager starts your desktop session and displays the Front Panel with backdrop. If this is the first time you have logged in, you will get a new session. If not, the Session Manager restores your previous session.

If the Login Manager rejects your session due to an incorrect name or password, click Start Over and reenter the information.

Now Try This!

Use these steps to log in to and exit a failsafe session.

1. Choose Session from the Options menu.
2. Choose Failsafe Session from the Session submenu.
3. Supply your user name and password.
4. Enter the `exit` command to log out.

Need Help?

If you supply a user name and password, the Session Manager brings up a single terminal window. You may issue commands when you receive the shell prompt.

To exit command line mode, type `exit` at the system prompt and press Return.

Logging In Using a Different Language

Your system administrator sets the default language for your system. If the administrator installs more than one language on your system, you can choose a different language on the Options menu. This operation sets the `LANG` shell environment variable for your session.

Now Try This!

To log in and out using a different language:

1. Choose Language from the Options menu on the login screen.
2. Choose the language group that includes the language you need.
3. Set a language.
4. Log in.
5. Log out by clicking the Front Panel Exit control or choosing Log out from the Workspace menu.

Logging In Using the CLI

Use the command line interpreter to:

- Remotely log in from another system using a terminal emulator.
- Use the failsafe session from the Option Session menu.

NOTE

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

Now Try This!

To log in to and out of a command line session:

1. From the login screen, choose Command Line Login from the Options menu.

   The login screen disappears and is replaced by a console prompt:

   ```
   login:
   ```

2. Type your user name and press Return.
3. Type your password and press Return. The password is not echoed on your terminal.

   ```
   Last login: Mon Jun  15  10:07:00 ...
   $
   ```

   You see the date and time of your last login. 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.

Need Help?

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

If you make a mistake in the login procedure, the system responds with login incorrect and gives you the opportunity to log in again.

Types of Sessions

There are two types of sessions: current and home.

- **Current session**

  When you log in to the system, the Session Manager creates a session for you. This session is called your current session. If you change the characteristics of your desktop during the session, the Session Manager captures this information when you log out and restores it when you next log in.

- **Home session**

  Your home session is one that you explicitly save. You have the option of specifying that the Session Manager always restores the home session when you log in rather than the current session.

Now Try This!

To start the home session at login:

1. Click the Style Manager control on the Front Panel.
2. Click the Startup control in Style Manager.
3. Select Return to home session.
4. Click OK.

You can also choose between the current and home session each time you log in by selecting Ask Me at Logout on the Style Manager's Startup dialog box.

Now Try This!

To save a home session:

1. Click the Style Manager control on the Front Panel.
2. Click the Startup control in Style Manager.
3. Click Set Home Session in the Startup dialog box.
4. Click OK in the confirmation dialog box that appears.

Choosing 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 set or change your password. Your password is confidential. Once you set it, you are 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) password
- Choose a password that is easy to guess, or hard to remember
- Write down your password

Password Guidelines

Passwords may contain nonalphanumeric characters as well as letters and numbers. You should select a password that is easy to remember, but not one that is obvious. For security reasons, it is best to select a password that is greater than six characters in length.

Passwords:

- Must contain at least five characters
- Can be longer than eight characters, but only the first eight are recognized
- Cannot consist entirely of lowercase letters; must include at least one uppercase letter, one digit, or one symbol
- Should be changed frequently

Changing Your Password

Use the `passwd` command to change your password. The system will prompt you to enter:

- Your old password (to verify who you are)
- Your new password
- Your new password again (for confirmation)

If you make an error when confirming your password, your original password is not changed.

Password confirmation ensures that you do not set your password to an unknown word derived from a typing error — unless you make the same mistake twice.

Now Try This!

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

1. At the system prompt enter the `passwd` command:

```
$ passwd
Changing password for [username]
```

2. `Old password:` Enter your current password. It will not echo on the screen.
3. `New password:` Enter your new password. It will not echo on the screen.
4. `Retype new password:` Enter the new password again. It will not echo on the screen.

Some Useful Tips

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

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

If you make an error when confirming your password, the system displays the message `Mismatch - password not changed`.

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

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

Listing Tru64 UNIX Documentation

Overview

Compaq supplies a complete set of manuals with each kit of the Tru64 UNIX operating system. This documentation is available in an online documentation set and in hardcopy in the Software Documentation Kit. Tru64 UNIX commands are also documented on line in the reference pages.

Tru64 UNIX documentation is also available on the World Wide Web at: `http://www.UNIX.digital.com/faqs/publications/pub_page/pubs_page.html`

Using the Documentation

The online documentation is available on the Tru64 UNIX Documentation CD-ROM in a format that is readable with a Web browser. To read this HTML documentation using the Netscape browser included with the Tru64 UNIX operating system, launch the Netscape software from the CDE Applications panel. Then choose the documentation link on the Tru64 UNIX home page.

The Tru64 UNIX documentation set is divided into two categories.

- End User Kit — for general users and system and network managers
- Developer's Kit — for programmers

End User Kit

The End User Documentation Kit contains all the information needed to install and use the Tru64 UNIX system. This kit is divided into three subkits as follows:

- Startup Documentation Kit
  - — *Release Notes*
  - — *Installation Guide*
  - — Update Installation Quick Reference Card
  - — *Technical Overview*
  - — Documentation Map
  - — Quick Reference Card
  - — *Documentation Overview*, *Glossary*, and *Master Index*
- System and Network Management Documentation Kit
  - — *System Administration*

  - — *Performance Manager*
  - — *DECevent Translation and Reporting Utility*
  - — *System Configuration and Tuning*
  - — *Network Administration*
  - — *X Window System Administrator's Guide*
  - — *X Window System Environment*
  - — *Software License Management*
  - — *Logical Storage Manager*
  - — *Kernel Debugging*
  - — *Guide to Prestoserve*
  - — *Sharing Software on a Local Area Network*
- General User Documentation Kit
  - — *Common Desktop Environment: User's Guide*
  - — *Common Desktop Environment: Advanced User's and System Administrator's Guide*
  - — *CDE Companion*
  - — *Command and Shell User's Guide*
  - — *DECwindows User's Guide*
  - — *Security*

Developer’s Documentation Kit

The Developer's Documentation Kit contains information for software developers who are developing applications for Tru64 UNIX. The kit consists of three subkits:

- General Programming Kit

  This kit contains manuals on programming languages, debuggers, support tools, network programming, DECthreads, real-time, internationalization, and calling standards.
- Windows Programming Kit

  This kit contains manuals on programming applications for the Common Desktop and X windows application environments for Tru64 UNIX.
- Writing Device Drivers Kit

This kit contains information for programming developing device drivers for the Tru64 UNIX operating system.

For a complete list of the manuals in the Developer's Documentation Kit, refer to the *Tru64 UNIX Documentation Overview*, *Glossary*, and *Master Index*.

Using Online Documentation

CDE online documentation is furnished with the operating system and provides help in the following ways:

- Help Manager

  A special help application that displays the online help registered on your system

- Application help

  Standard CDE applications, like the File Manager and Calendar have application-specific help

- Man Page Viewer

  An application to view the UNIX reference pages in the CDE environment

- Online reference pages

Use the `man` command to view the UNIX reference pages from the command line.**Now Try This!**

Try to access online help.

If you are running a failsafe session, you should have a system prompt. If you are running CDE, click the Terminal control in the Personal Applications subpanel.

At the prompt, type `man date` and press Return.

Press the spacebar until the prompt returns.

Summary

Introducing the Tru64 UNIX Operating System

The Tru64 UNIX operating system is a multiuser, interactive, general-purpose operating system. It allows you to run more than one program at a time while performing other tasks simultaneously.

Tru64 UNIX is an open system based on the OSF operating system called OSF/1. It has the "look and feel" of UNIX.

Describing the User Interface

The Tru64 UNIX operating system provides two methods for users to interact with the operating system:

- Graphical user interface (GUI)

  For workstation users or intelligent terminals with bit-mapped displays

- Command line interface (CLI)

  For character cell terminal users or a terminal emulator

Logging In and Out of the System

Your system administrator must provide you with username and password information before you can log in to the UNIX operating system.

Enter information in response to system requests or prompts. The prompts for logging in and system response differ depending upon the type of terminal being used.

UNIX is sensitive to uppercase and lowercase letters and treats them as different characters. Two types of commands are used in this course.

Enter	Type the command and press Return
Type	Type the characters and do **not** press Return

- The Tru64 UNIX command line is indicated by a customizable symbol such as $.
- Use the `exit` command to log out of the system.
- Use the `passwd` command to change your password. Keep your password confidential.
- The `man` command displays reference information about commands.

Choosing Passwords

When choosing passwords, do **not**:

- Choose words found in the dictionary
- Use personal information
- Use your default (original) password
- Choose a password that is easy to guess, or hard to remember
- Write down your password

Listing Tru64 UNIX Documentation

Compaq supplies a complete set of manuals with each kit of the Tru64 UNIX operating system. This documentation is available in an online documentation set and in hardcopy in the Software Documentation Kit. Tru64 UNIX commands are also documented on line in the reference pages.

2

Using CDE

Unit Overview

Introduction

The Common Desktop Environment is a new industry standard, user interface based on the X Window System. CDE allows you to start applications and select functions using the mouse or keyboard rather than typing long commands on a command line.

Microsoft Windows and the Apple Macintosh are other graphical user interfaces that provide a windowing environment using icons, menus, and the mouse to interact with the system.

This unit discusses how to invoke some of the desktop applications that you will find on your system using the CDE user interface.

Objectives

To use desktop applications, you should be able to:

- Use the Front Panel
- Obtain help using both the CDE and CLI user interface
- Use the Text Editor to create and edit text files
- Use Calendar to schedule appointments, make To Do lists, browse other calendars and schedule group appointments
- Use the terminal emulator application to execute UNIX commands
- Run applications from the CDE desktop

Resources

For more information on the topics in this chapter, see the following:

- *Common Desktop Environment: User's Guide*, Chapters 3, 4 and 6
- *CDE Companion*, Chapters 1 and 3

Using the CDE Front Panel

Overview

When you begin a CDE session, you see a virtual desktop consisting of a workspace and a Front Panel of controls. This section discusses the elements of the Front Panel, and describes how to customize the desktop to suit your needs.

Front Panel

The **Front Panel** is a special window containing controls that:

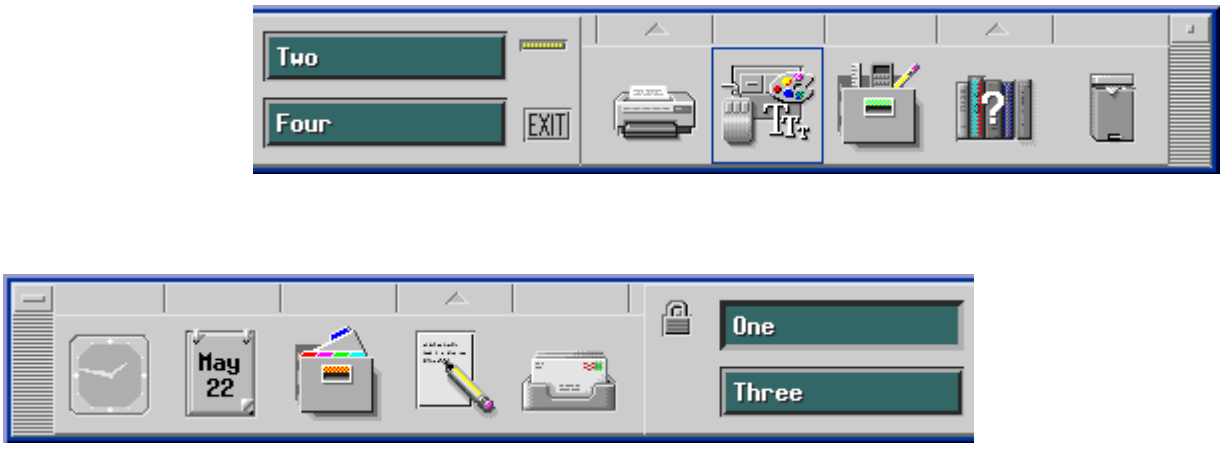
- Start up applications
- Contain buttons for changing to other workspaces
- Indicate system state
- Are drop zones
- Can contain subpanels

The Front Panel:

- Can be positioned anywhere on the desktop
- Can be minimized
- Belongs to all workspaces
- Can be customized

The Front Panel contains the main panel, including workspace switch, and subpanels.

Figure 2-1: Front Panel



Workspace Switch

The **workspace switch** is a portion of the Front Panel containing:

- Buttons to change from one workspace to another
- A control to lock your workspace
- A control to exit the system
- A control that indicates system activity

Each workspace can be customized for a particular use. For example, if you are working on two different projects, you can customize a workspace for each of the projects with the files and tools you need.

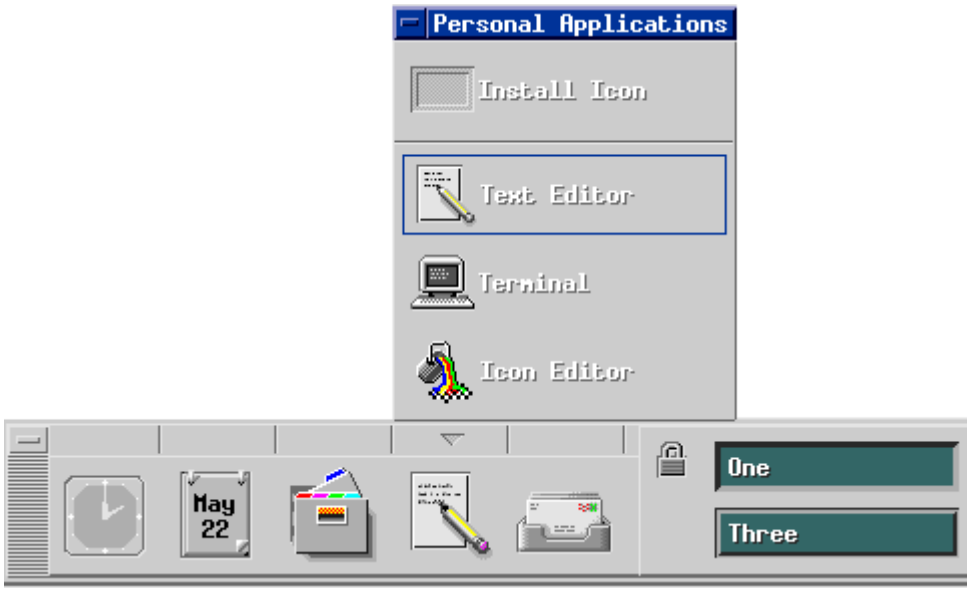
Subpanels

A **subpanel** is a menu associated with a control. It is indicated by an arrow button above the control. A subpanel:

- Contains an Install Icon control to customize the subpanel
- Contains a labeled copy of the Front Panel control
- Can be deleted or added to a Front Panel control

This figure shows a Personal Applications subpanel.

Figure 2-2: Subpanel



Activating a Control

The following table describes how to activate controls, subpanels, and pop-up menus.

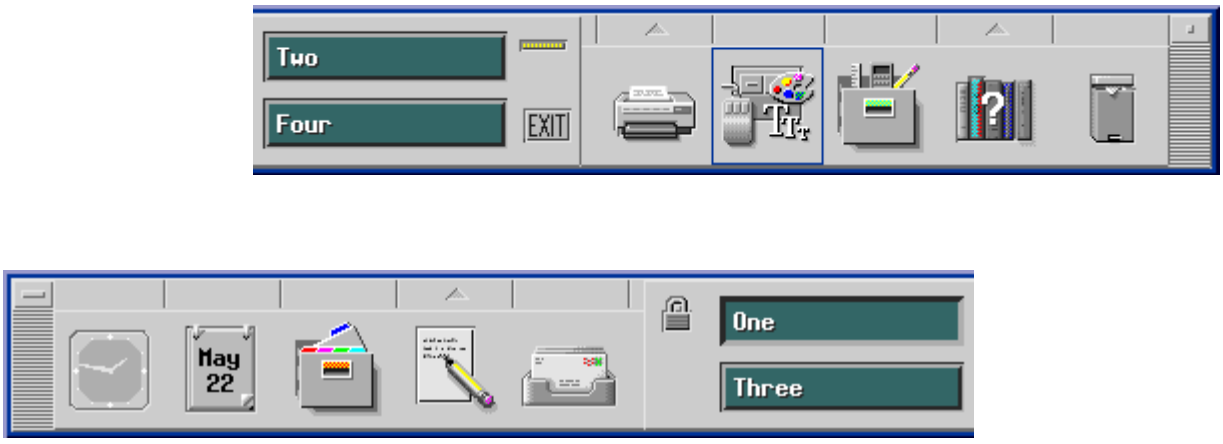
Table 2-1: Activating Controls

To:	Action
Activate a Front Panel control	Click the control
Open a subpanel	Click the arrow button above the control
Close a subpanel	Click the arrow control for subpanel or double-click the window button in the upper left corner of the subpanel
Activate a Front Panel pop-up menu	Point to a control and press and hold MB3
Change workspaces	Click the appropriate workspace button

Using a Front Panel Control

The Front Panel contains several controls, as shown in this figure.

Figure 2-3: Front Panel Controls



Each control has one or more of the following behaviors.

Click	What happens when you click the control
Drop	What happens when you drop an object on the control
Indicator	What the control indicates

This table shows the behavior of each of the Front Panel controls.

Table 2-2: Front Panel Control Behavior

Name	Click Behavior	Drop Behavior	Indicator Behavior
Clock	None	None	Time of day
Calendar	Starts Calendar application	Drops a file to add it to calendar	Date
File Manager	Opens File Manager	Drops a folder to open File Manager view	None
Personal Applications (Text Editor)	Starts Text Editor	Opens a new file in Text Editor window	None
Mailer	Starts Mailer	Opens the file	Indicates new mail arrival
Lock	Locks displays and keyboard; password enabled	None	None
Workspace Switch	Changes workspaces	None	Indicates current workspace
Busy Light	None	None	Blinks when an action is running
Exit	Begins logout from session	None	None
Printer (Default)	Displays status of default printer	Prints the file	None
Style Manager	Opens Style Manager	None	None
Application Manager	Opens Application Manager window	None	None
Help Manager	Opens Help Viewer window	Drops a master volume file to open Help Viewer	None
Trash Can	Opens Trash Can window	Moves file to Trash Can	Indicates when empty and not empty

Now Try This!

The purpose of this exercise is to familiarize you with the Front Panel controls.

1. Click each control and observe the result. What happened when you clicked the Clock control?
2. Click a subpanel indicator above one of the controls.
3. Click the button that minimizes the window.

4. Move the Front Panel to a different location.
5. Click a workspace switch and observe the result.
6. Click the Help control and search for information on the Front Panel.

Solution

1. The Clock control and Busy light are indicators. The Clock shows the current time and the Busy light blinks when the system performs some action.
2. Not all controls have subpanels. You may add or remove a subpanel on a control. The subpanels are not anchored to the control. You can place them anywhere on the desktop.
3. Click the minimize button and the Front Panel closes. Double-click the icon to bring the Front Panel back.
4. No specific solution.
5. The `dtconsole` window is associated with your home workspace.
6. No specific solution.

Personal Applications Control

The personal applications control and its subpanel contain frequently used applications. There are three default applications on the subpanel.

- Text Editor
- Terminal
- Icon Editor

The Text Editor is the default application for the Personal Applications control and its icon appears on the Front Panel.

You can customize this control by adding the applications you use most to the subpanel. You can also change the control's default application.

This table shows the behavior of the Personal Applications subpanel controls.

Table 2-3: Personal Applications Subpanel Control Behaviors

Name	Click Behavior	Drop Behavior	Indicator Behavior
Install Icon	None	Installs an icon dragged to the subpanel	None
Text Editor	Starts Text Editor	Opens the file in a new Text Editor window	None
Terminal	Starts Terminal Editor	None	None
Icon Editor	Starts Icon Editor	Opens a bitmap or pixmap file in Icon Editor	None

Printer Control

The printer control has the following default subpanel controls:

- Install Icon
- Default Printer
- Print Manager

This table shows the behavior of the Printer subpanel controls.

Table 2-4: Printer Subpanel Control Behaviors

Name	Click Behavior	Drop Behavior	Indicator Behavior
Install Icon	None	Installs icon on subpanel	None
Default Printer	Displays the default printer status	Prints the file on the default printer	None
Print Manager	Starts Print Manager	Prints file on default printer	None

Help Control

The Help control and its subpanel provide access to the CDE Help facility. This table shows the behavior of the Help subpanel controls.

Table 2-5: Help Subpanel Control Behaviors

Name	Click Behavior	Drop Behavior	IndicatorBehavior
Install Icon	None	Installs an icon onto the subpanel	None
Help Manager	Opens Help Viewer window	Drops a master volume file (`*.sdl`) to open it	None
Desktop Introduction	Displays "Introducing the Desktop"	None	None
Front Panel Help	Displays the Front Panel help volume	None	None

Changing Workspaces

A **workspace** is the screen display area. Using workspaces, you can set up multiple work areas. For example, one workspace can be used for managing mail, another for project management, and so on.

You can click a workspace button to change to another workspace. Each workspace button has a pop-up menu that allows you to add, delete, or rename a workspace and to get workspace help.

The default number of workspaces is four. Your system administrator sets the default on a systemwide basis. You can add more workspaces as you need them.

Getting Help

Overview

CDE provides help:

- With a context-sensitive Help key
- In an application's Help menu
- Through the Help Manager
- Through the Man Page Viewer

With CLI, you can obtain help by issuing the `man` command.

This section shows you how to use each of these methods to obtain help.

Using the Help Key

CDE provides context-sensitive help. Press the Help key (or function key **F1**) and the application you are using displays the most appropriate topic related to the window, button or dialog box highlighted. The Help key is a shortcut for selecting help on a menu or dialog box.

Using an Application's Help Menu

Most applications have a Help menu that contains:

- An overview of help topics
- Task instructions for most application operation
- Reference summaries for windows, dialog boxes, menus and application resources
- On-item help, which allows you to click an item and view a description of the item
- Help on using Help windows
- Help providing the version and copyright information for the application

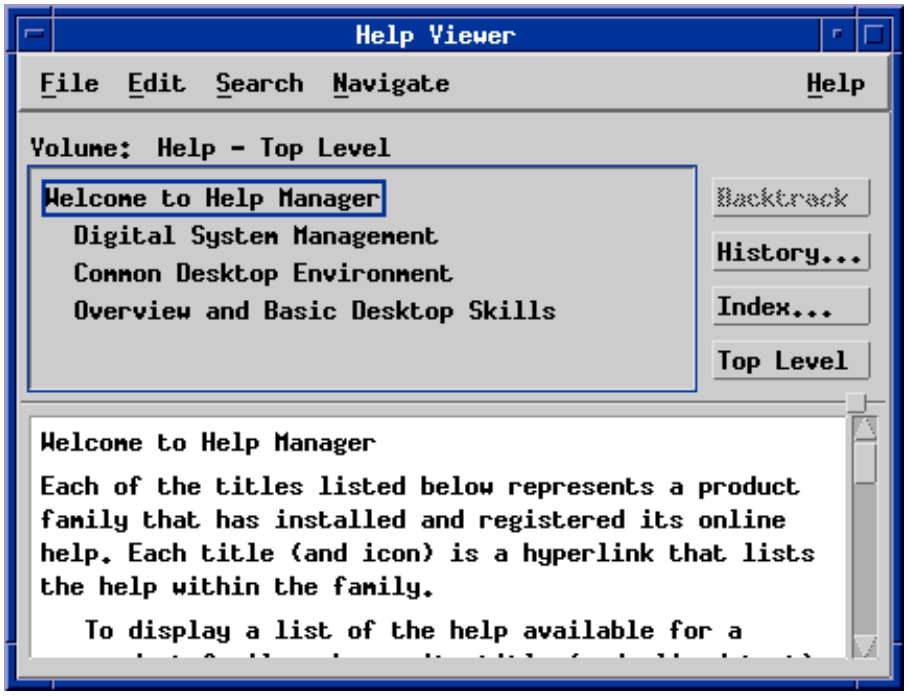
Using the Help Manager

Click the Front Panel Help Manager control to display online help.

The Help Viewer window displays a topic tree. Click a topic to display the related help information. The Help Manager builds a history of where you have been. Choose the Backtrack button to return to a previous topic.

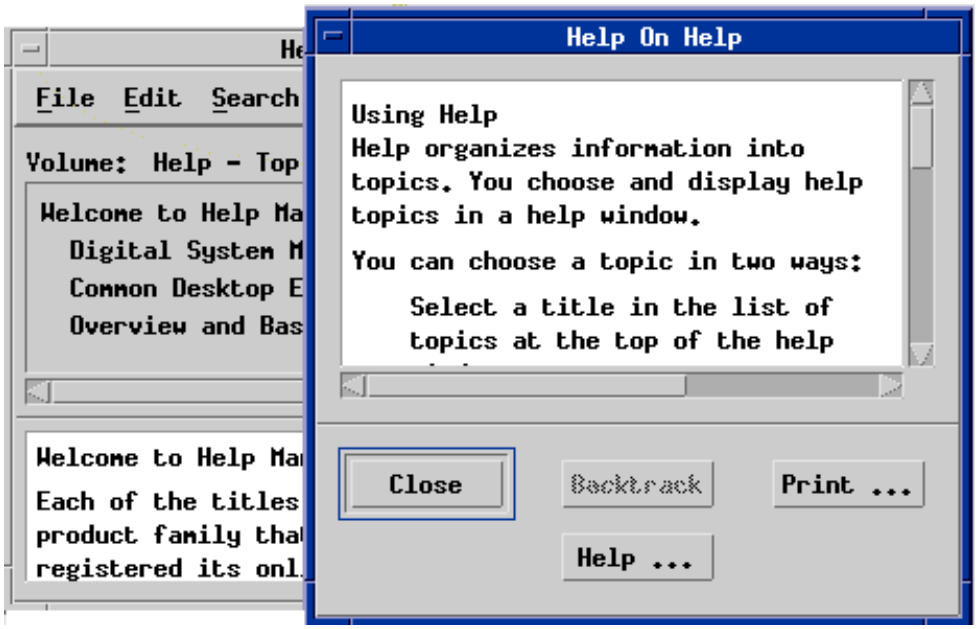
The Help system displays help for the home topic or help for a chosen topic. In this display, hypertext allows you jump to the related help topic.

Figure 2-4: Help Viewer



The Using Help topic provides information on how to obtain help.

Figure 2-5: Help on Help



Now Try This!

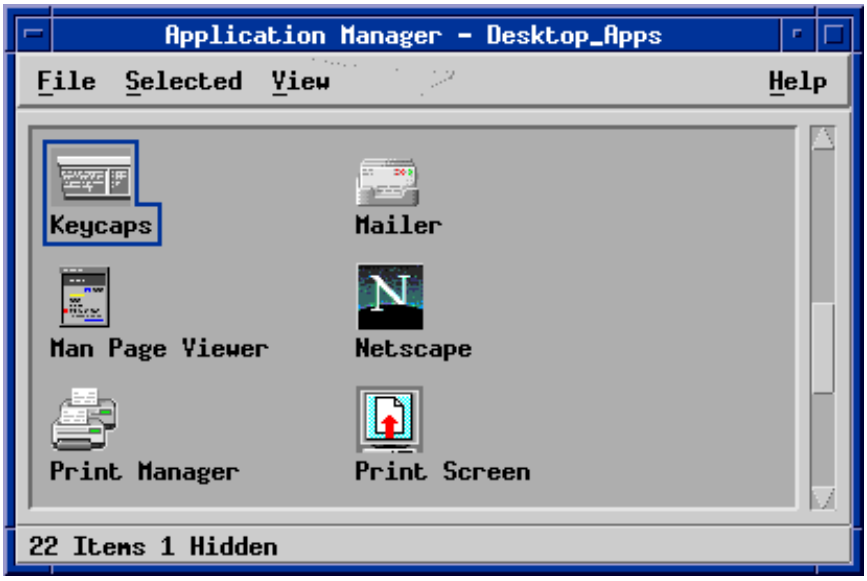
Explore online help:

1. Click Help Manager on the Front Panel.
2. Click the hypertext *Common Desktop Environment*.
3. Click the hypertext *Introducing the Desktop*.
4. Click the Backtrack button.
5. Click the Index button. This provides a search function.
6. Click the Entries with button and type Front Panel.
7. Click the All Volumes button.
8. Click Start Search. It should display several entries.
9. You can continue to explore the Help Manager. Choose Close on the File menu to exit Help.

Using the Man Page Viewer

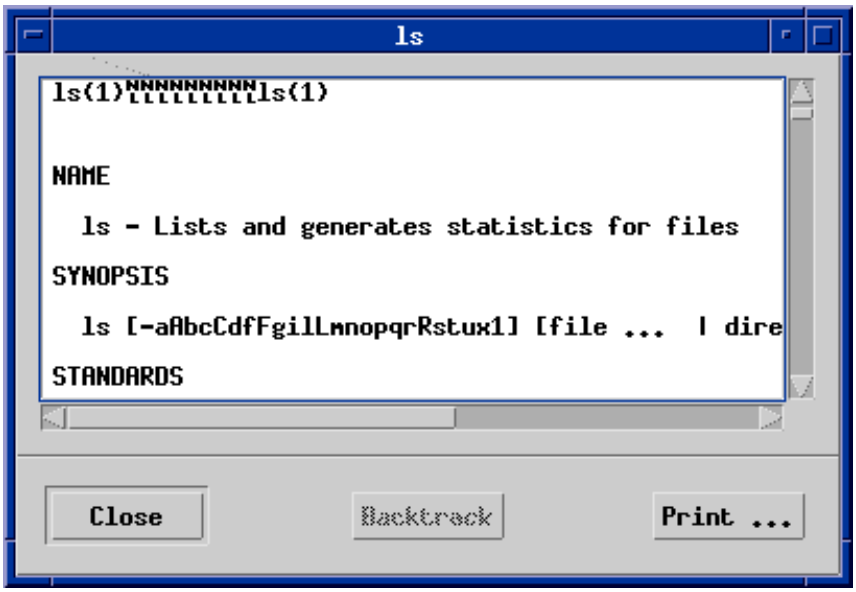
Choose Man Page Viewer from the Application Manager — Desktop_Apps window to view the traditional manual reference pages. Double-click the Man Page Viewer icon and the system prompts you for the reference page.

Figure 2-6: Man Page Viewer



The next figure shows the display for the `ls` reference page.

Figure 2-7: Getting Help on the ls Command



Using the CLI man Command

When you are working in a terminal window using the command line interface, you have access to the standard reference pages through the use of the `man` command.

Organization of Reference Pages

The reference pages are organized into sections according to function, as shown in the following table.

Table 2-6: The man Reference Pages

Section	Topic
`man(1)`	Commands that can be entered by all users
`man(2)`	Operating system program interfaces
`man(3)`	Subroutines
`man(4)`	Include files, program files, and some system files
`man(5)`	Miscellaneous topics, such as text-processing macro packages
`man(7)`	Device special files, related driver functions, and networking support
`man(8)`	System maintenance and operation

Note: there is no `man`(6) section, which is **games** on some UNIX systems. Therefore `cat`(1) means the `cat` command is documented in section 1. Some UNIX words are documented in more than one section; you might need to specify the section number.

Using the man Command

The `man` command provides information about a particular command. For instance, to get information about the `passwd` command, enter `man passwd`. The reference information is displayed one screen at a time.

- Press Return to display additional information, one line at a time
- Press the spacebar to display additional information, one screenful at a time
- Press **Q** or **q** to stop displaying additional information and return to the command line prompt

Entering `man passwd` displays the reference page for the `passwd` command, as shown in abbreviated form in the following example.

Example 2-1: Using the man 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 specified 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. For example, to list the commands related to passwords, enter `apropos password`.

The `man -k` command is equivalent.

Using the help Command

The Tru64 UNIX command line interface does not have a Help facility. However, if you enter the `help` command, the system provides a brief list of helpful commands and their functions, as shown in the following example.

Example 2-2: 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".

$
```

Using Text Editor

Overview

The Text Editor (`dtpad`) on the Front Panel is a basic text editor which allows you to perform file creation and editing functions using both the keyboard and a mouse. Some of the editor features include:

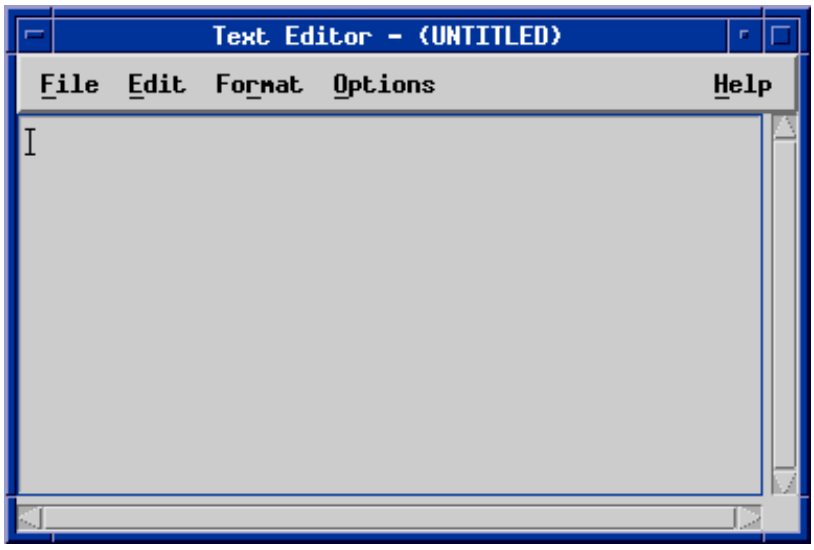
- Pull-down menus for edit and file operations
- Simple formatting
- Search and replace
- Spelling checker
- Status line showing cursor position by line number and the ability to position the cursor by line number
- Automatic save of the editing session if the Text Editor abnormally terminates
- Undo edit operation
- Overstrike mode
- Wrap-to-fit function

This topic introduces the Text Editor and shows how to use some of its features.

Starting the Text Editor

Click the Text Editor control on the Front Panel and it displays the Text Editor main window containing title, menu, scroll bars and an optional status line.

Figure 2-8: Text Editor's Main Window



The title bar displays the name of the current document. If it is a new document, the title bar displays (UNTITLED).

The menu bar provides editing functions on pull-down menus including file access, formatting and option changing functions.

The text input area is where you type the text of your document. This area is blank for a new document; if you read in an existing document, it is displayed here.

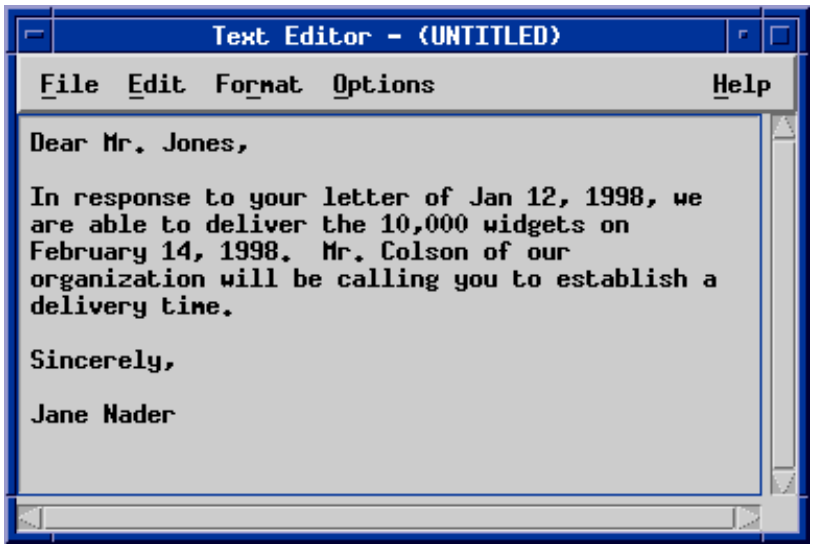
At the bottom of the window, you can optionally display a status line which displays:

- Line number containing the insertion point
- Total number of lines in the document
- Text editor messages
- Overstrike indicator if the editor is in Overstrike mode

Entering Text

When the Text Editor starts up, it provides you with a new document, entitled UNTITLED, with the insert cursor positioned at the top of the window waiting for you to input text. The insert cursor, shaped like an I, always shows the current position for entering text. The status line indicates the line number in the file.

Figure 2-9: Entering Text in Text Editor



To enter text, click in the document and begin typing text. By default, the Text Editor is in insert mode, so text is inserted beginning at the cursor location.

If you select Overstrike mode, when you position the cursor within a text line, any new text will overlay the existing text in the line beginning at the cursor location. Press the Insert key to toggle between Insert and Overstrike modes.

Press the Return key to start a new line.

The Text Editor Wrap To Fit option, when turned on, will automatically wrap a text line at the edge of the window. To use this option, you must press the Return key only to force a new line or create a new paragraph. If you use the Wrap To Fit option, you can resize the Text Editor window and the text automatically adjusts to fit the new window width.

Moving the Cursor

You can move the insertion cursor with the mouse or the arrow keys.

- To move the cursor with the mouse, move the pointer to the new position in the text and press MB1.
- Use the arrow keys or the following:

Home	Move to beginning of the current line
End	Move to end of the current line
Ctrl/Home	Move to beginning of the document
Ctrl/End	Move to end of the document

You can view text outside the current view using the mouse or the keyboard.

- Using the mouse, drag the vertical or horizontal scroll bar to view more text or view longer lines.
- Use the PageUp and PageDown keys to scroll up or down a page.

You can also move the cursor using the status line. After displaying the status line, press Ctrl/G or click in the Line field of the status line. Enter the line number you want to go to and press Return.

Selecting Text

You can select text using either the mouse or the keyboard.

- To select text using the mouse, press and drag MB1 to highlight the text.
- You can also use the mouse to select text in the following ways:
  - — Double-click to select the word
  - — Triple-click to select the line
  - — Quadruple-click to select all text
- To select a text block with the keyboard, move the cursor to the beginning of the text block, hold down the Shift key and use the arrow keys to move the cursor to the end of the text block.

Editing Text

The Edit menu provides the following editing functions:

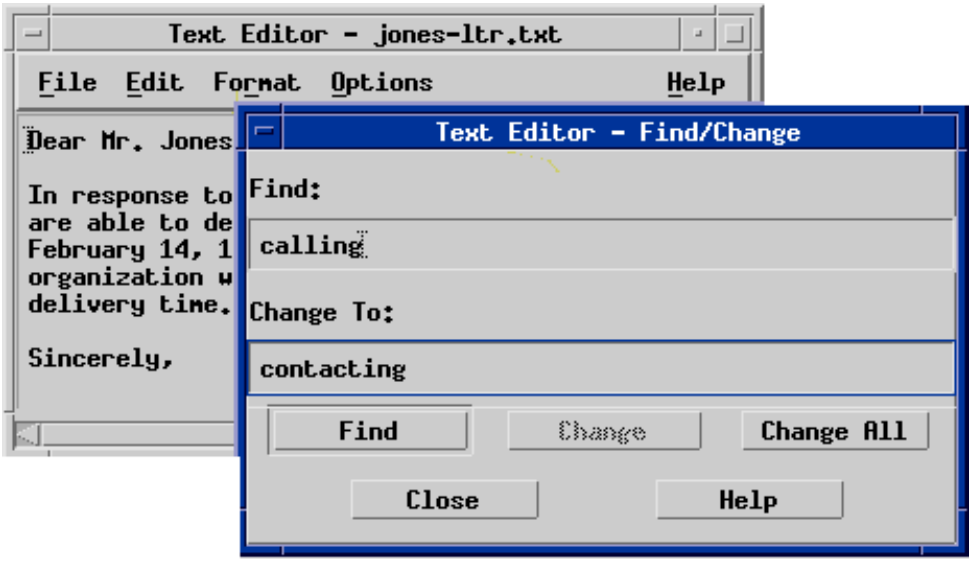
- Cut and paste
- Delete text
- Find and change text
- Check spelling

Finding and Changing Text

Use the Find/Change function from the Edit menu to search for a character string and replace that character string with another character string.

This figure shows the Find/Change dialog box. When you click the Find button, the Text Editor searches for the character string entered into the Find input box on the dialog box. If the Text Editor finds a matching character string, you can change the found string (and all occurrences) to the string in the Change To input box by clicking on Change or Change All.

Figure 2-10: Finding and Replacing Text



You can also use this feature just to find text without performing a replace operation.

Using Drag and Drop

You can use the Text Editor's drag and drop feature to:

- Move/copy text from one location within a document to another location
- Move/copy text from a document to another window
- Include a file into a document

To move or copy text from one location within a document to another location using drag and drop, follow these steps:

1. Select the text you want to move.
2. Move: With the pointer on the selection, press and hold MB1 and drag the selection to where you want to insert the text.

   Copy: With the pointer on the selection, press and hold the Ctrl key and MB1 and drag the selection to where you want to insert the text.
3. Release the mouse button.

To include a file using drag and drop, perform these steps:

1. Click the position in your document where you want to include the contents of the file.
2. Display the icon of the desired file in a File Manager window.

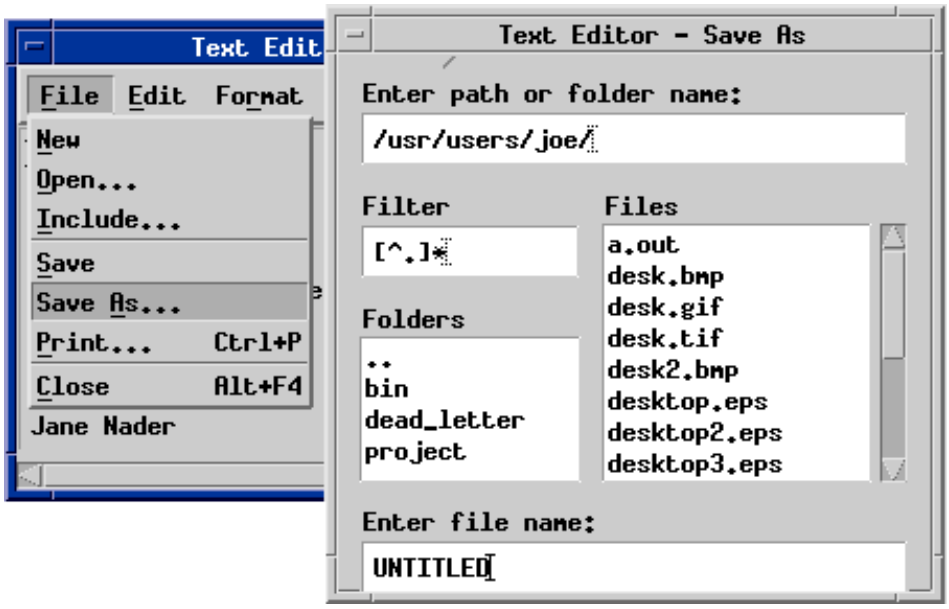
3. Drag the file icon into the Text Editor window and drop it.

Saving Your Work

The File menu, shown in this figure, gives you access to file operations.

- Creating a new document
- Opening an existing document
- Including another document
- Saving a document to a file
- Saving the document with a new name
- Printing a document
- Closing the application

Figure 2-11: Text Editor's File Menu



Before ending your Text Editor session, you must save the document to a file or you will lose your work. You should always save your document periodically while you are working. This will minimize any potential loss if the Text Editor is interrupted.

Recovering a File

The Text Editor performs an automatic save in your current folder if the application is interrupted. This file is a recovery file with the name `#filename#`. If you did not name your file, the recovery file name is `#UNTITLED#`.

To recover the file:

1. Start the Text Editor.
2. Open the recovery file.
3. Save the document using the original file name or provide a new name.

Now Try This!

Start up the Text Editor application and explore the following features.

1. Select Wrap To Fit from the Options menu.
2. Type in two paragraphs from your course book. Press Return only after each paragraph.
3. Select the first sentence using the mouse.
4. Use drag and drop to copy it to the end.
5. Save your document in a file.
6. Use the Include item on the File menu to include another file

Need Help?

Use the Help menu to access online help.

The editor starts up in insert mode. You can just begin typing your paragraphs in the text input area of the screen.

To copy while dragging, press Ctrl with MB1.

Using Calendar

Overview

If you do not have a secretary, an electronic tool like the Calendar application can keep track of your appointments, meetings and any miscellaneous activities you need to do. If you do have a secretary, that person can maintain your calendar as well as the calendar of other group members.

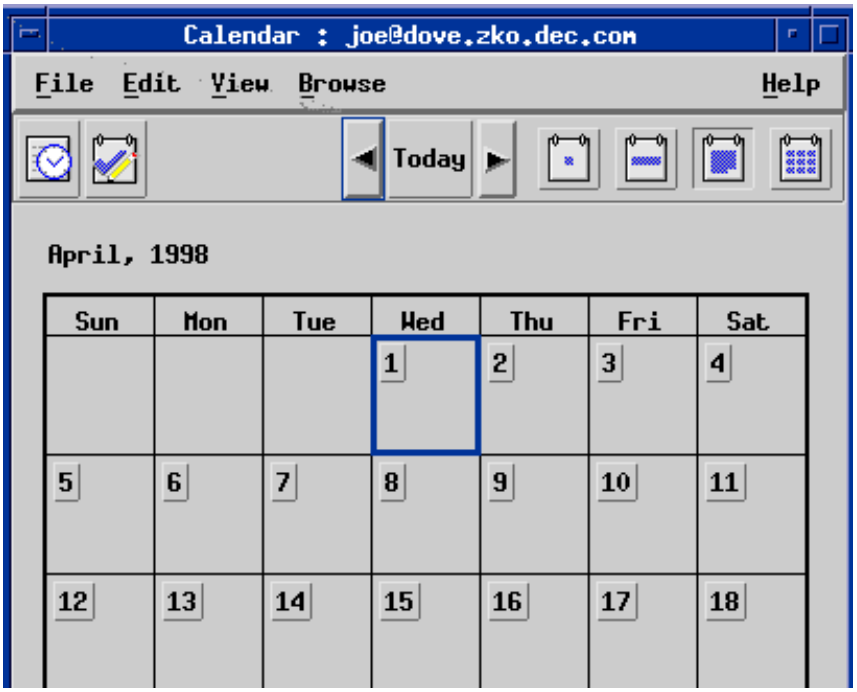
Project leaders will find Calendar's feature of scheduling group appointments useful when setting up meetings to discuss project concerns.

This topic discusses the Calendar application and how to make an entry in your Calendar or To Do list.

Viewing Calendar

When you click the Calendar control on the Front Panel, Calendar displays a main window, as shown in the figure. It contains menus and tool bar controls at the top and the selected Calendar view in the lower part.

Figure 2-12: Calendar's Main Window



Calendar can display the following views:

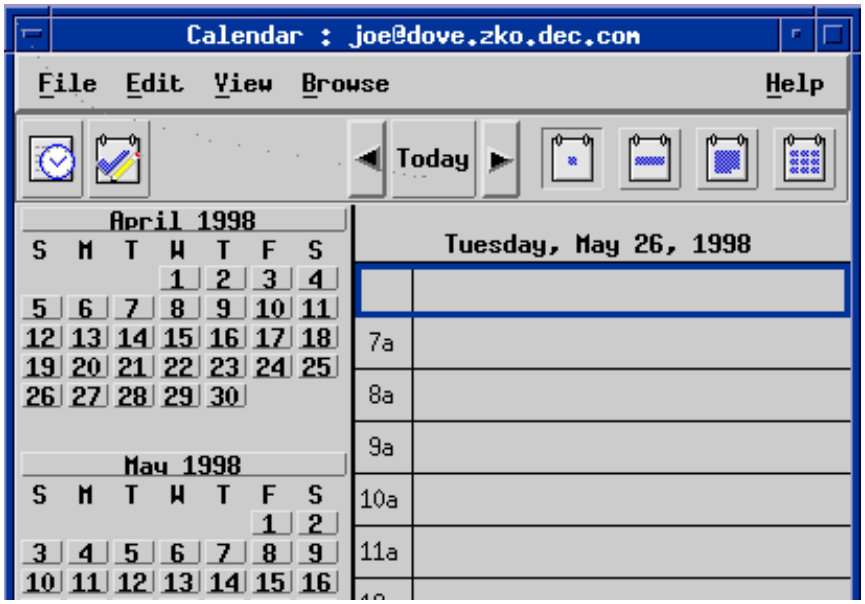
Day	Shows one day's appointments
Week	Shows one week's appointments
Month	Shows one month's appointments (abbreviated)
Year	Does not show appointments

Day View

The day view shows one day's appointments. The left side of the view displays a 3-month calendar including the previous, current and next month. If you click a day in the 3-month calendar, that day becomes the current day in the display.

You can change the display to a different month (month view) by clicking the name of the month in the mini-calendar.

Figure 2-13: Day View



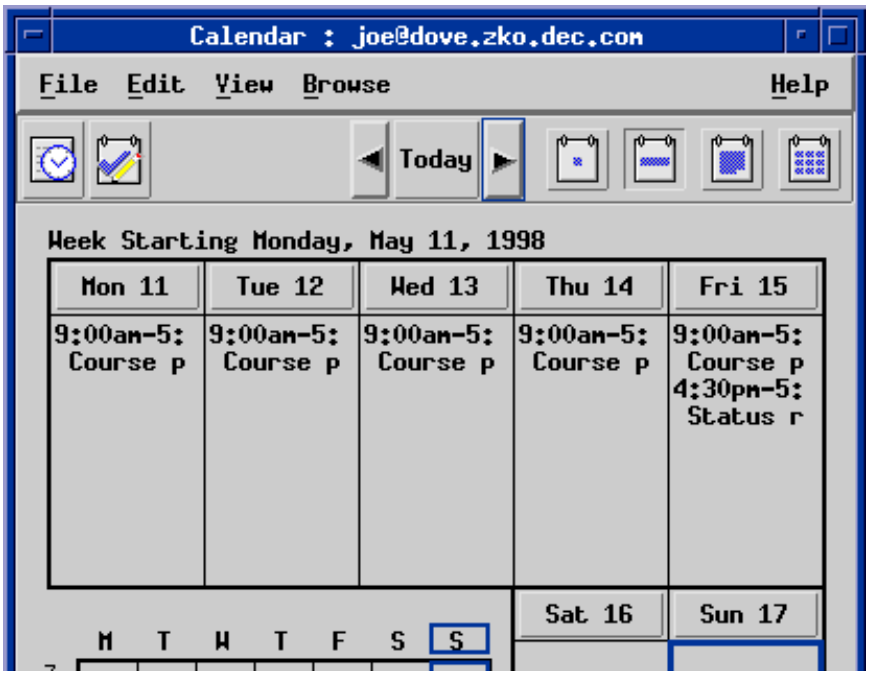
Week View

Calendar's week view displays:

- One week's appointments
- A week grid showing scheduled (shaded) and available (nonshaded) times
- Selected day highlighted

If you click a day name in the week view, the Calendar display changes to day view.

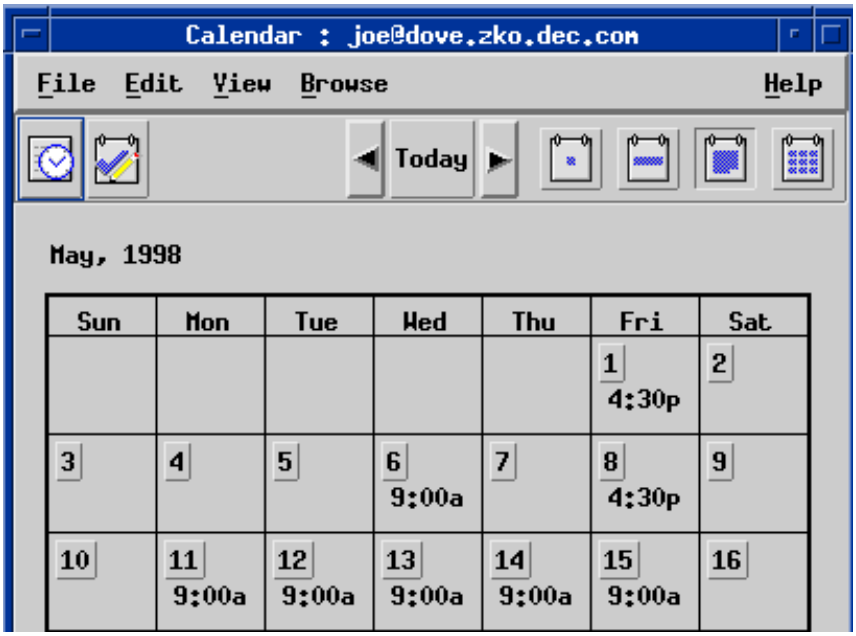
Figure 2-14: Week View



Month View

Calendar's month view shows the appointments in a calendar month. You can click a day number in month view to change the display to day view.

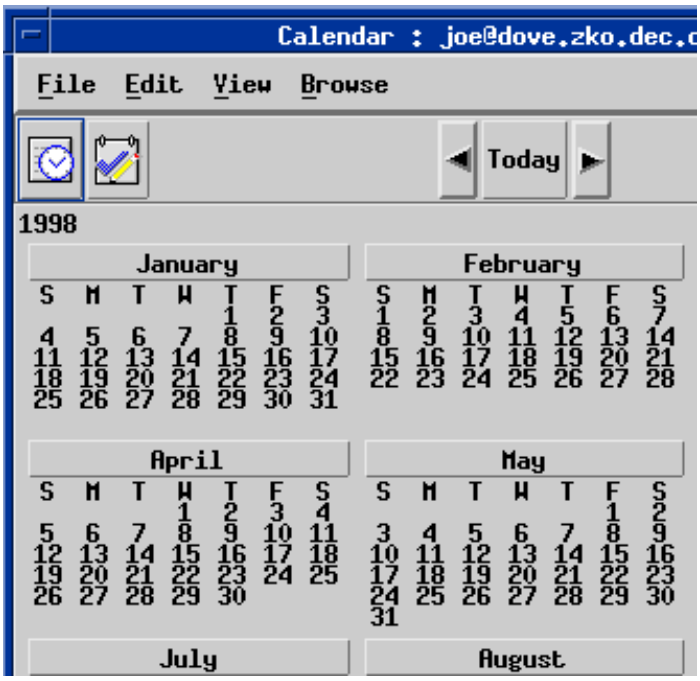
Figure 2-15: Month View



Year View

With the Year view, appointments are not displayed. If you click a month name, Calendar displays the selected month in month view.

Figure 2-16: Year View



Now Try This!

1. Start Calendar by clicking the Calendar control on the Front Panel.
2. Switch views by choosing options on the View menu.
3. Switch views by clicking the view buttons on the right of the toolbar.
4. Display the month view and show next month by clicking the right arrow on the toolbar.
5. Display the day view and show today's appointments by clicking the Today button on the toolbar.

Need Help?

Click the Help menu to access online help.

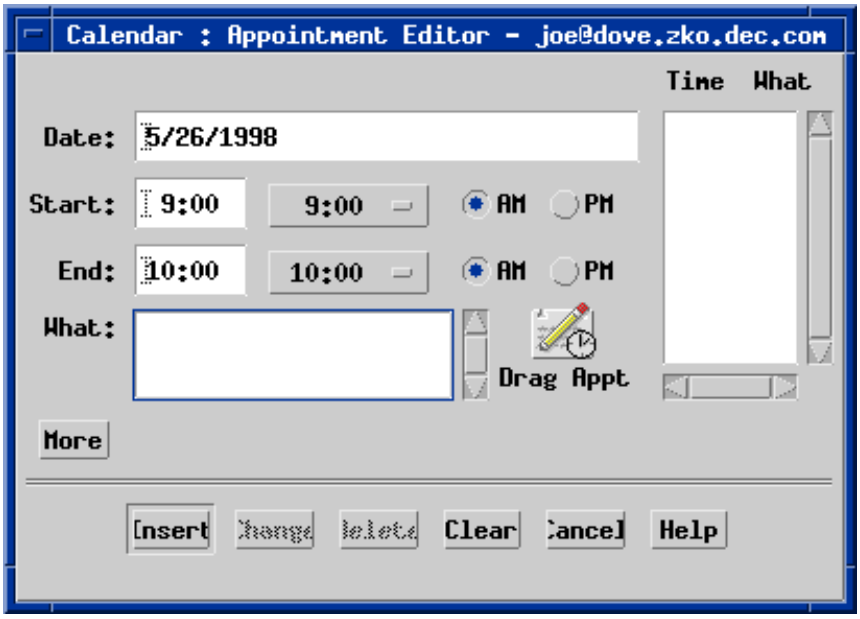
The view buttons on the toolbar are, from left to right, day view, week view, month view, year view.

Adding an Appointment

You can add appointment information from the week or month view by double-clicking the appropriate day, or from day view by double-clicking the time slot.

Calendar displays the Appointment Editor window. This figure shows the extended version of the Appointment Editor window, which appears when you click the More button.

Figure 2-17: Calendar: Appointment Editor



If you enter more than one appointment for a day, you can click Clear to reset the Editor options.

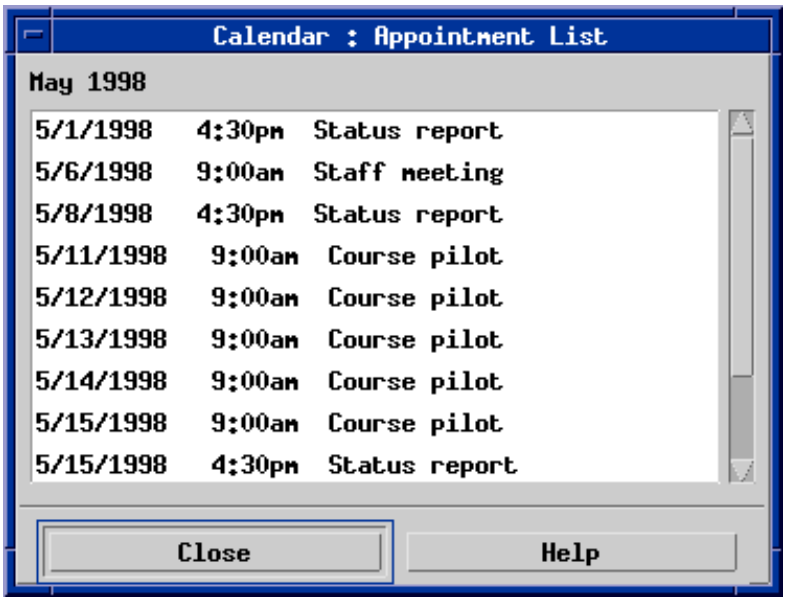
To change appointment information, select the appointment, make your changes and then click the Change button.

You can delete an appointment by selecting the appointment and then clicking the Delete button.

Viewing an Appointment List

To view your appointment list, choose Appointment List from the View menu. This figure reflects typical output.

Figure 2-18: Viewing an Appointment List



Now Try This!

1. Create the following appointments:

   — You are in class Monday-Friday this week, from 9 AM to 5 PM.

   — You have a 1-hour project status meeting next Monday at 10 AM.

2. View your appointment list.

Need Help?

It would be easier to see what you are doing in week or month view.

Double-click the day to open the Appointment Editor.

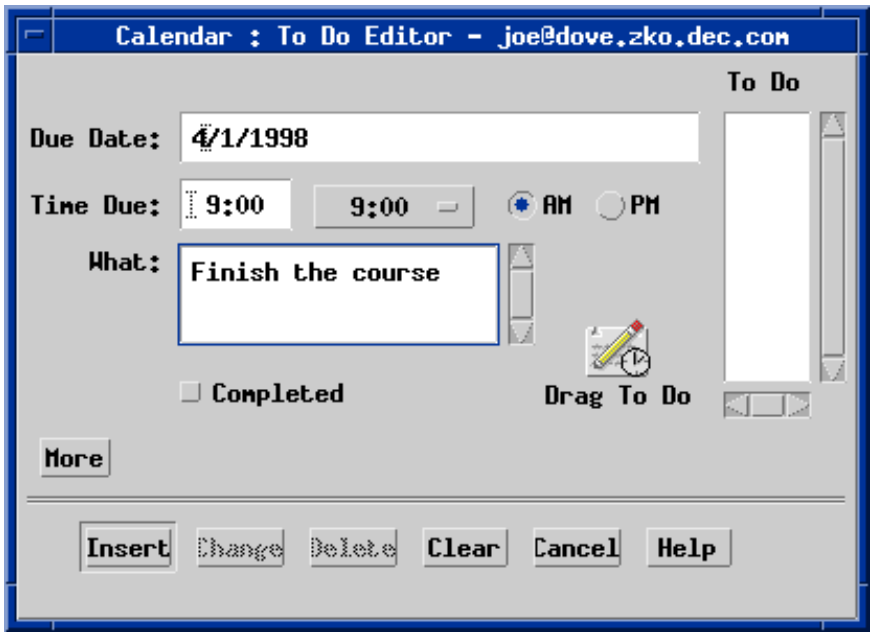
With your class appointment filled in, you can edit the date or use the More button to fill in the rest of the week.

Making a To Do List

To help manage your daily tasks, Calendar provides the To Do Editor for you to create and manage a list of action items. To create a To Do list, click the To Do Editor icon on the toolbar (or the To Do option on the Edit menu).

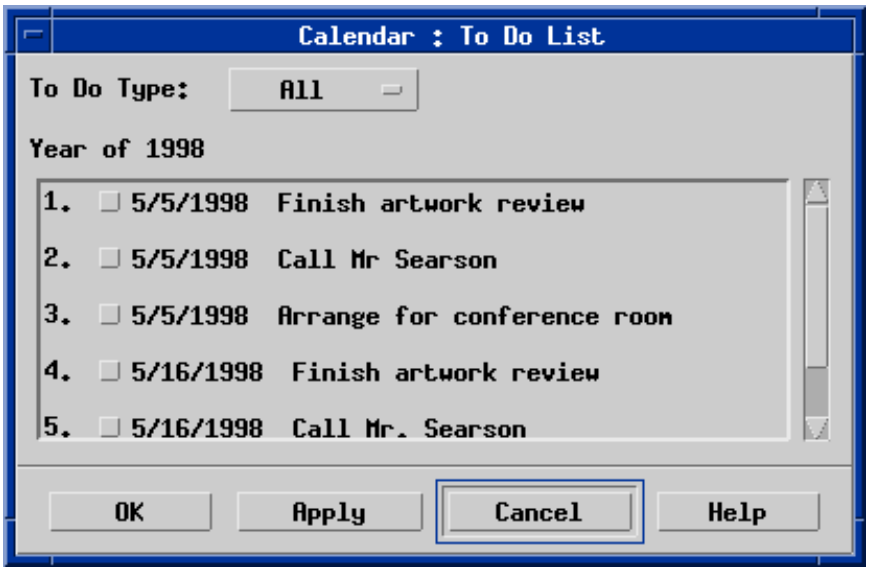
This figure shows the To Do Editor window where you enter information for your To Do List.

Figure 2-19: Making a To Do List



To view your To Do List, choose To Do List from Calendar's View menu. This figure shows a sample To Do List.

Figure 2-20: To Do List



Now Try This!

Create and view a To Do list.

1. Click the To Do Editor icon on the Calendar toolbar.
2. Enter the due date of next Monday, 10 AM.
3. Type in the What field: Write project status report.
4. Click Insert.
5. Click Cancel to close the To Do Editor.
6. Choose To Do List on the View menu.

Need Help?

Click the Help menu to access online help.

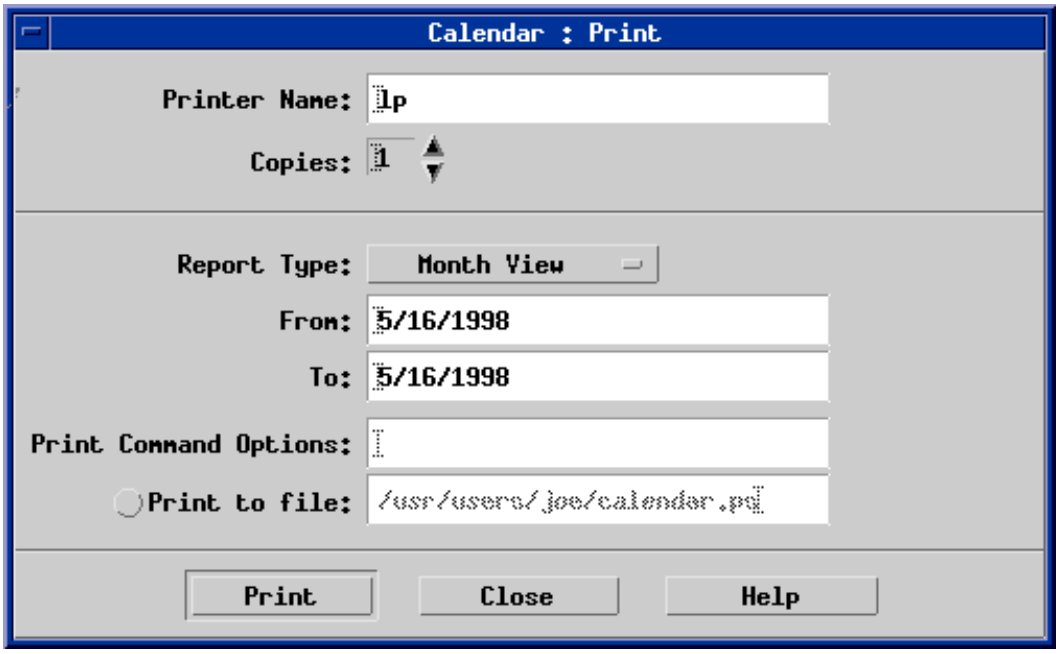
You can also open the To Do Editor by choosing To Do on the Edit menu.

Printing Calendar Views and Lists

When you need a hard copy of your calendar, you can print a view or list by choosing Print from Calendar's File menu. The Calendar: Print dialog box allows you to specify:

- Printer name
- Number of copies to print
- Report type: day, week, month, year, appointment list, or To Do list
- Calendar date range
- Other print command options
- Optionally printing to a file

Figure 2-21: Calendar: Print Dialog Box

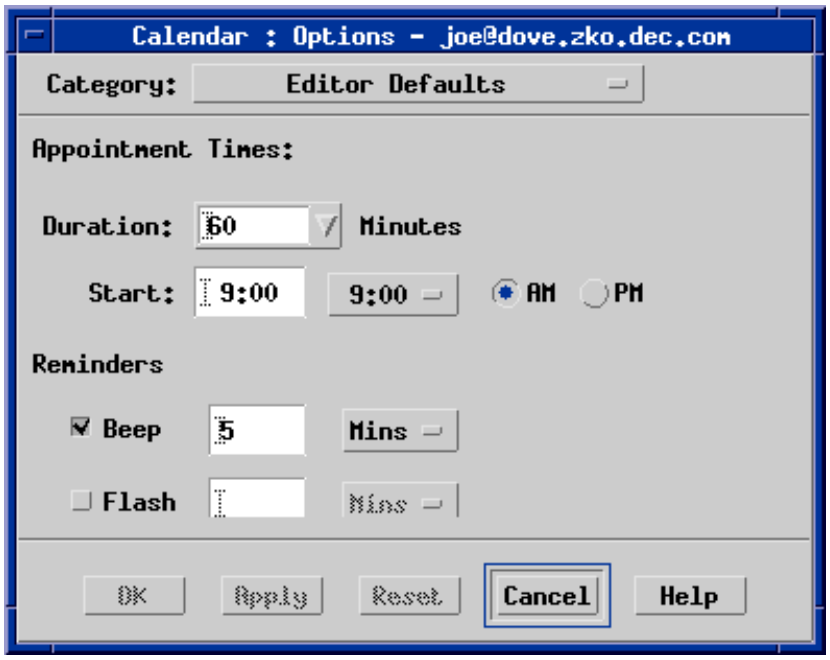


Customizing Calendar

Choose Options on the File menu to open the Options dialog box shown in the figure. You can customize the Calendar application in the following ways:

- Change the date format
- Change default printer settings
- Grant or deny access to your calendar
- Set the default Calendar view
- Change the Appointment Editor defaults

Figure 2-22: Calendar: Options Window



The category menu at the top provides a list of dialog boxes to customize Calendar.

Now Try This!

Grant access to your calendar so someone else can schedule an appointment for you.

1. Choose Options from the File menu.
2. Choose Access List and Permissions from the category menu.
3. In the User Name field, type the calendar name to grant access (*calendar-name@hostname*).
4. Select View and Insert permissions for Public access.
5. Click Add.
6. Click OK.

Need Help?

Click the Help button for online help explaining the dialog box and these steps.

You need the calendar name of another user on your network.

Browsing Calendars

Calendar allows you to look at the calendar of other users on your system or on the network if you are connected to one. This feature is very useful for setting up meetings. To use this feature you need:

- The calendar names that you want to browse

  The format of the calendar name is: *calendar-name@hostname*. You must ask the other users for this information.

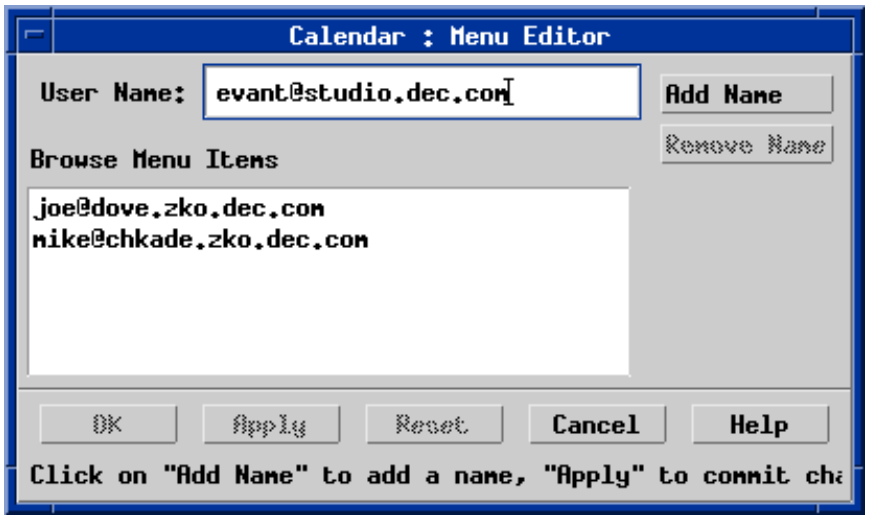
- To add the calendar names to the Browse list
- To determine if you have access to the calendar

Adding Names to the Browse List

To add names to the Browse list, perform the following steps:

1. Choose Menu Editor from the Browse menu.
2. Type the calendar name in the User Name field, as shown in the figure.
3. Click Add Name.
4. Click OK to add the calendar to the Browse list and close the Menu Editor dialog box.

Figure 2-23: Calendar: Menu Editor

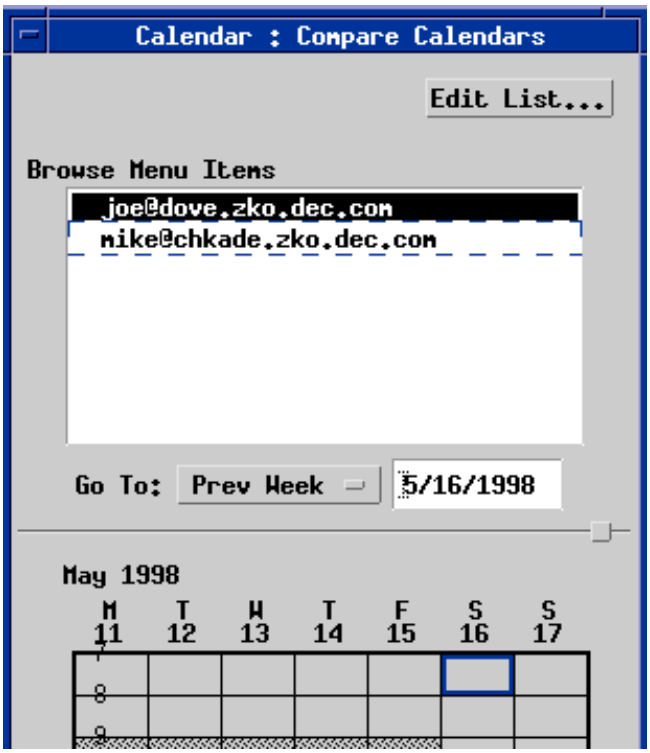


Accessing Other Calendars

Use these steps to determine whether you have access to other calendars.

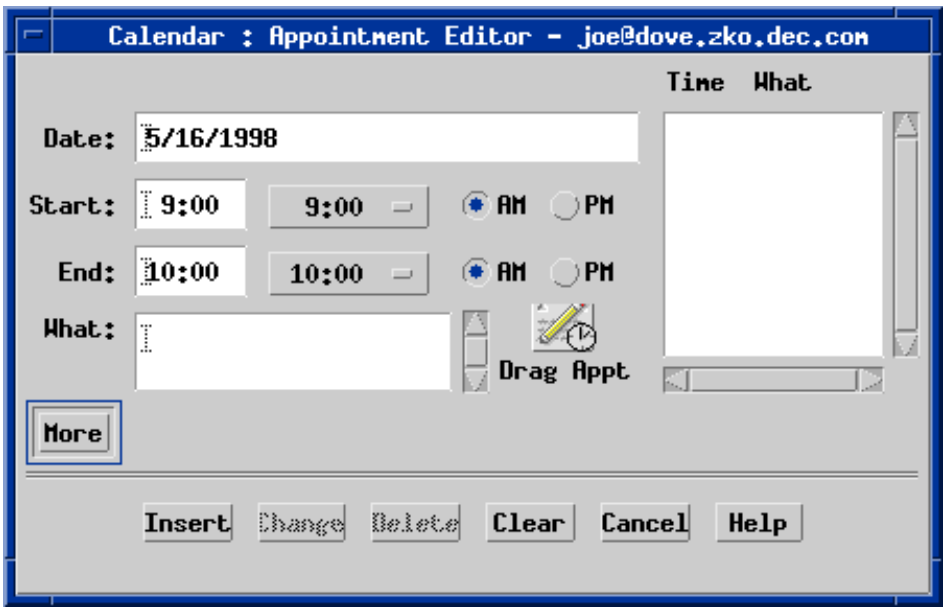
1. Choose Compare Calendars from the Browse menu.
2. Select one or more entries to check for access, as shown in this figure.

Figure 2-24: Calendar: Compare Calendars



3. Click Schedule to open the Group Appointment Editor, as shown in this figure.

Figure 2-25: Calendar: Group Appointment Editor

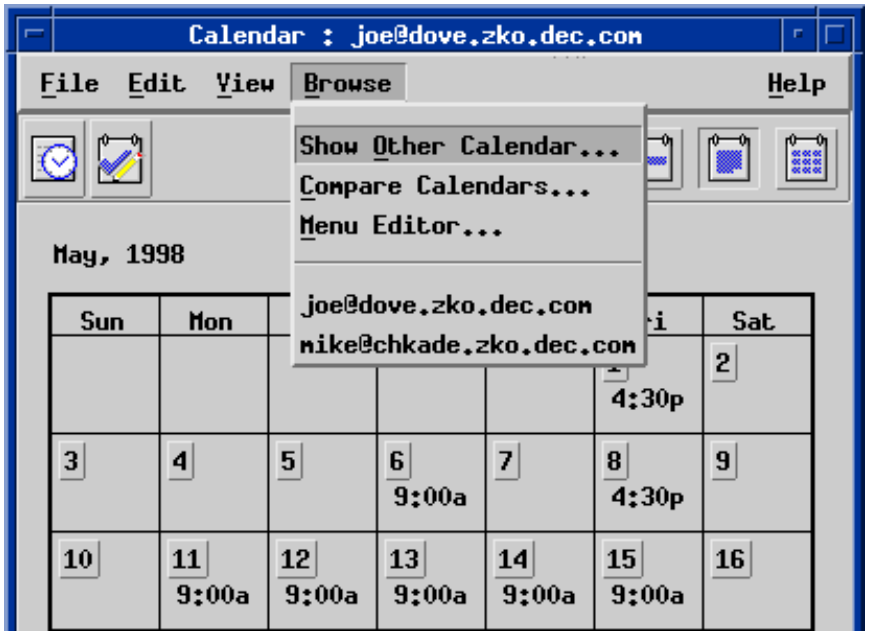


4. Review the Access column in the Calendar Access list; a Y indicates that you have insert access while an N indicates that you do not. If you have insert access, you can insert a group appointment.

Showing Other Calendars

To show another user's calendar, choose Show Other Calendar on the Browse menu shown in this figure. You need to type the calendar name.

Figure 2-26: Showing Other Calendars



If your system is connected to a network and you know the names of other calendars, you can:

- Browse calendars to set up a meeting or appointments for several people
- Check other calendars for free time
- Maintain a list of calendars for browsing called a browse list
- Mail meeting or appointment reminders to a group

Using Terminal

Overview

CDE provides a terminal emulator window where you can enter commands using the UNIX command line interface. The default terminal emulator for the desktop is `dtterm`. The Terminal application provides support for legacy applications written for ANSI and ISO conformant character terminals such as the DEC VT220.

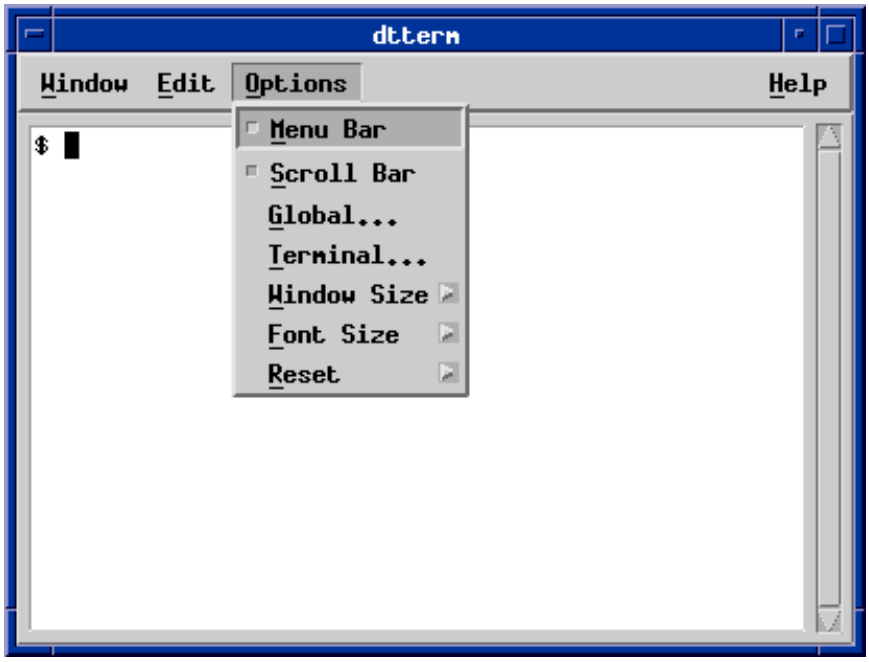
Your system administrator may provide other terminal emulators for your use.

This topic discusses how to use the default terminal emulator for the desktop, `dtterm`.

Starting Terminal

When you click the Terminal control in the Personal Applications subpanel, Terminal displays its main window, as shown in this figure.

Figure 2-27: Terminal's Main Window



When the Terminal emulator supplies a prompt, you may enter commands to start up applications, issue commands or perform copy and paste operations between windows.

Starting an Application

To start up an application, you must type a command at the command line prompt. The syntax of a command is as follows:

```
application [options] &
```

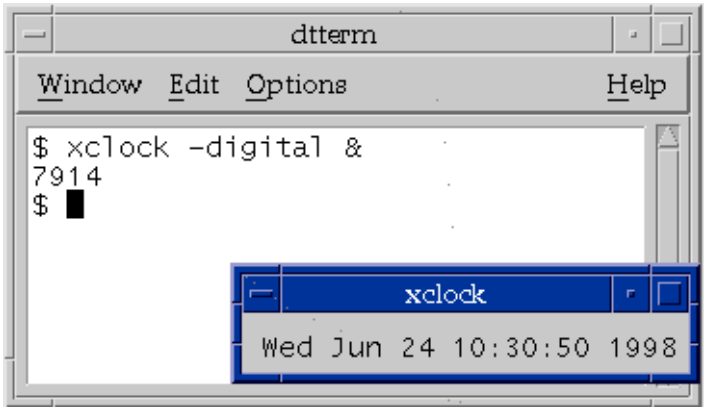
Where *application* is the name of the application you want to run and *options* is information which passes to the application in the form of parameters or switches.

Check the reference pages or online help to find the type of information you can supply on the command line for your application.

The ampersand (&) specifies that you want the application to run in the background. This allows you to continue entering commands in the Terminal window while the application is running.

This figure shows how to run the clock program, `xclock`. The `-digital` switch tells the clock program you want a digital clock instead of an analog clock.

Figure 2-28: Starting an Application in Terminal



After pressing Return to enter the command, the `xclock` application displays the time and date in a separate window which you can place anywhere on the desktop, and the `dtterm` window is ready to take another command.

Now Try This!

Use the Terminal application to start the `xclock` application in the background.

Need Help?

If you get the message `xclock: not found`, use the `/usr/bin/X11/xclock` command.

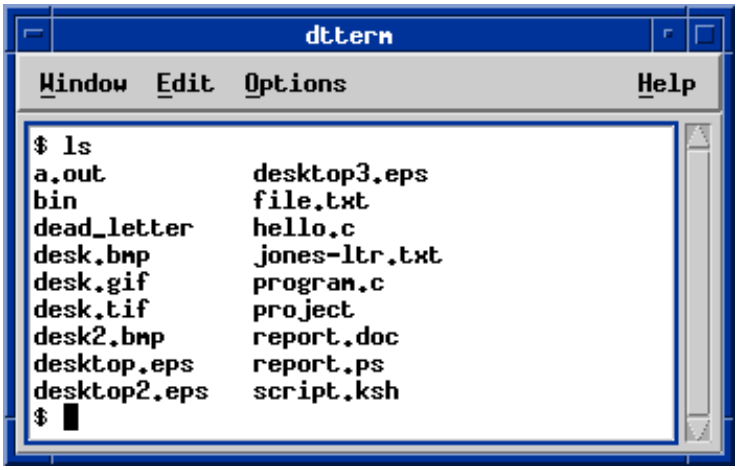
Solution

1. Click the Terminal control on the Personal Applications subpanel.
2. Type `xclock &`.

Entering a Command

This figure shows how to use the `ls` command to list the files in a directory.

Figure 2-29: Entering a Command in Terminal



See the reference pages for syntax information on a command. You may also use the CDE Help system to read reference pages.

Now Try This!

Use the Terminal application to enter the `ls` command. Enter the `man ls` command to read about options to the `ls` command.

Solution

1. Click the Terminal control on the Personal Applications subpanel.
2. Type `ls` .

Copying and Pasting Text

To copy some text within the Terminal window, you can use the mouse:

1. Using MB1, drag the mouse over the desired text.

   Result: Text is highlighted

2. Release MB1 when the desired text is highlighted.

   Result: Highlighted text is moved to the clipboard. The text is available for paste operations until you perform a new copy operation.

3. Click MB2 where you want to insert the text.

   Result: Highlighted text appears in the paste location.

Customizing Terminal

You can customize the Terminal window using the:

- Options menu
- Global Options dialog box
- Terminal Options dialog box

Using the Options Menu

Through Terminal's Options menu you can change the following Terminal characteristics.

- Remove/Restore the Menu Bar.

  If you remove the Menu Bar, to restore it, click MB3 anywhere in the Terminal window to display the pop-up menu. Choose Menu Bar from the Options menu.
- Remove/Restore the Scroll Bar.
- Resize the window to 80 characters by 24 rows or 132 characters by 24 rows.
- Change the font size used for text in the window.

Setting Global Options

By selecting Global from the Options menu, you can change the following characteristics:

- Change the cursor style, blink and blink rate.

  The cursor style is either a box or an underline. You can enable or disable cursor blinking. If you enable blinking, the default blink rate is 250 milliseconds, which you can change.
- Switch foreground and background colors.

  You may choose Normal, which displays the foreground and background colors as usual, or Inverse which switches the foreground and background colors.
- Set scroll behavior.

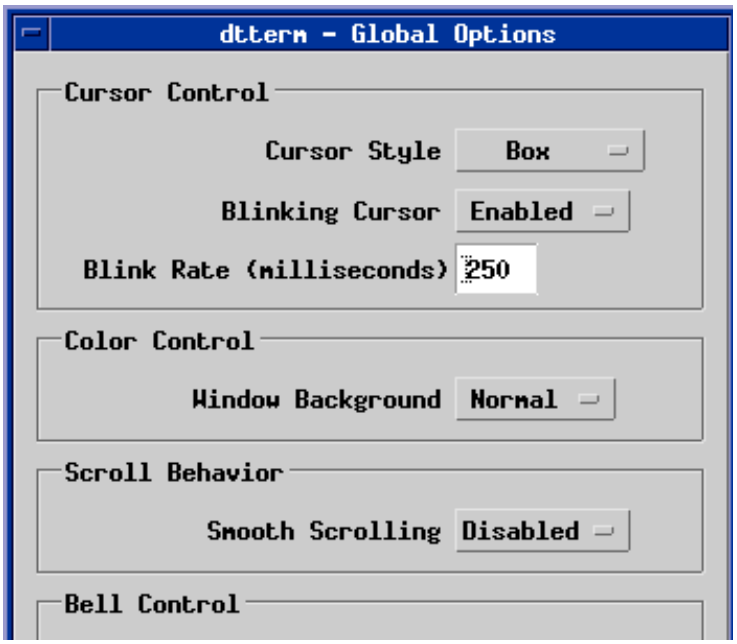
  You can enable or disable smooth scrolling. With smooth scrolling each line is sent to the Terminal window immediately rather than storing the lines in a buffer. The default is smooth scrolling disabled (scroll a page at a time).
- Set the bell type.

  You may choose Audible (default) which causes the bell to make a sound, or Visible which causes the bell to blink the background color.
- Set margin warning and Distance.

  You can enable or disable a visible or sound indication that the cursor is within a specified distance from the right margin. When you enable this feature, you

should set the number of characters from the right margin (default is 10 characters).

Figure 2-30: Global Options Dialog Box



Setting Terminal Options

You can change the behavior of your keyboard and screen by choosing Terminal from the Options menu. The Terminal Options dialog box allows you to change the following:

- Cursor key mode

  In Normal mode, the arrow keys move the cursor in the left, right, up and down directions. In Application mode, the arrow keys generate escape sequences (control sequences) that the application interprets for its own purposes.

- Keypad mode

  In Numeric mode, the keys on the numeric keypad generate the corresponding numeral on the display. In Application mode, the numeric keypad keys generate escape sequences that the application interprets for its own purposes.

- Newline sequence

  Return Only (the default) generates only a carriage return at the end of a line. Return/linefeed generates both a carriage return and a line feed at the end of a line.

- User functions

  Locks or unlocks (default) the user function keys.

- 132 column switching

  When enabled, the system will automatically enlarge the window to accommodate 132 columns when the active application switches to 132 columns. When this feature is disabled (default), the system does not automatically enlarge the window if the active application switches to 132 columns.

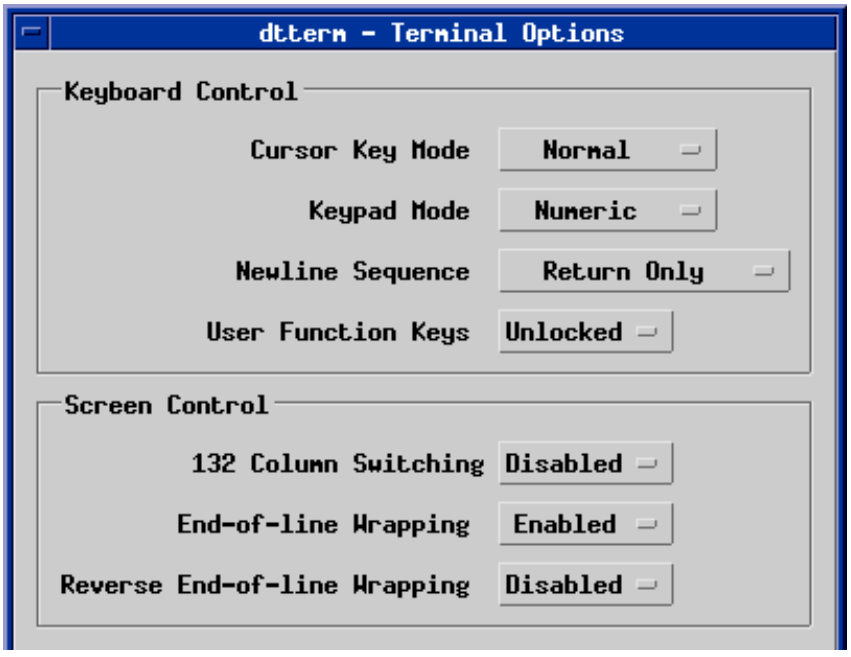
- End-of-Line wrapping

  When enabled, characters wrap to the next line automatically when the end-of-line is reached. When disabled, text does not wrap.

- Reverse end-of-line wrapping

  This option controls the behavior of backspacing at the end of a line. If enabled, backspacing automatically wraps to the previous line when an end-of-line is reached. If disabled, no wrapping occurs.

Figure 2-31: Terminal Options Dialog Box



Resetting Control Characters

Use the Ctrl key in combination with another key to perform a specific function. For example, Ctrl/C cancels the current operation and displays the command line prompt.

The table shows the default settings for control characters set by the Login Manager.

Table 2-7: Terminal Control Characters

Key	Control Name	Function
Ctrl/H	`erase`	Backspace: erases characters
Ctrl/C	`intr`	Interrupt: cancels the current operation and displays the command line prompt
Ctrl/U	`kill`	Stops an operation or application
Ctrl/Q	`start`	Accepts keyboard input; continues an application that has been paused
Ctrl/S	`stop`	Does not accept keyboard input; pauses an application
Ctrl/@	`swtch`	Switches between layers in a shell

Terminal (`dtterm`) emulates a terminal. The control settings may be different for the actual hardware you are using or for another terminal emulator. You can change the settings of the control characters by using the `ttyModes` resource.

The syntax of this statement is:

```
ttyModes: controlname ^key
```

For example:

```
ttyModes:  erase ^H int ^C kill ^U start ^Q stop ^S swtch ^@
```

To set terminal control characters:

1. Add the `ttyModes` resource to the file `.Xdefaults` in your home directory.
2. Double-click the Reload Resources icon in the Desktop_Tools application group (in Application Manager).

Running Applications

Overview

CDE provides a variety of methods for invoking applications which give a great deal of flexibility for how you use the system.

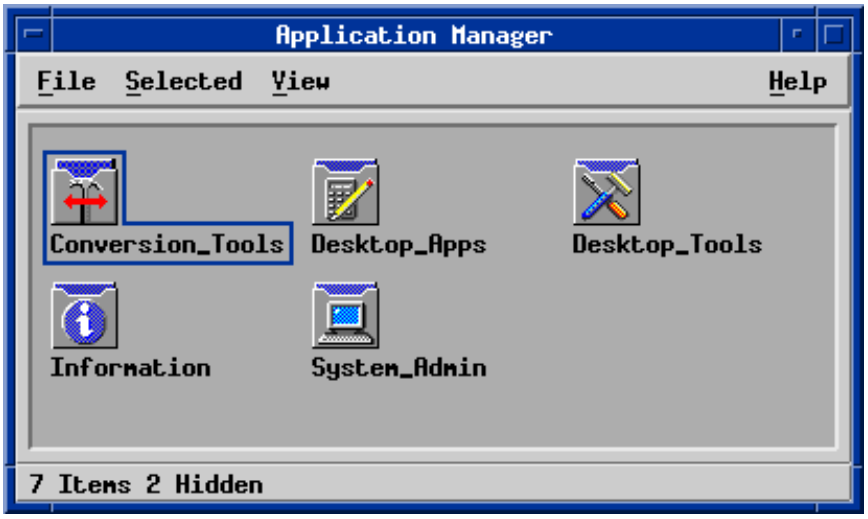
You can invoke an application from the Application Manager, File Manager, Front Panel and the Terminal Emulator application. This topic discusses how to run applications using the Application Manager and File Manager.

Using the Application Manager

The Application Manager control on the Front Panel provides some of the most frequently used tools and applications available on the system. The applications you find there are either built into the desktop or configured by your system administrator.

When you click the Application Manager control, the system displays a File Manager view of a special folder, as shown in this figure.

Figure 2-32: Application Manager Folder



This is not a typical folder because files and folders are not created directly in it. They are gathered into this location automatically when you log in.

The contents of these folders can consist of application files, sample data files, and **read me** files.

Registered Application Groups

Registered applications are those applications that your system administrator registers using the application registration process. This process lets the desktop know that an application is present.

Each application group in the Application Manager is gathered together each time you log in. The application group may be located on your system or on other systems in the network.

Because the Application Manager gathers these applications each time you log in, it is possible that the Application Manager view will display a different set of applications depending upon the work of your system administrator.

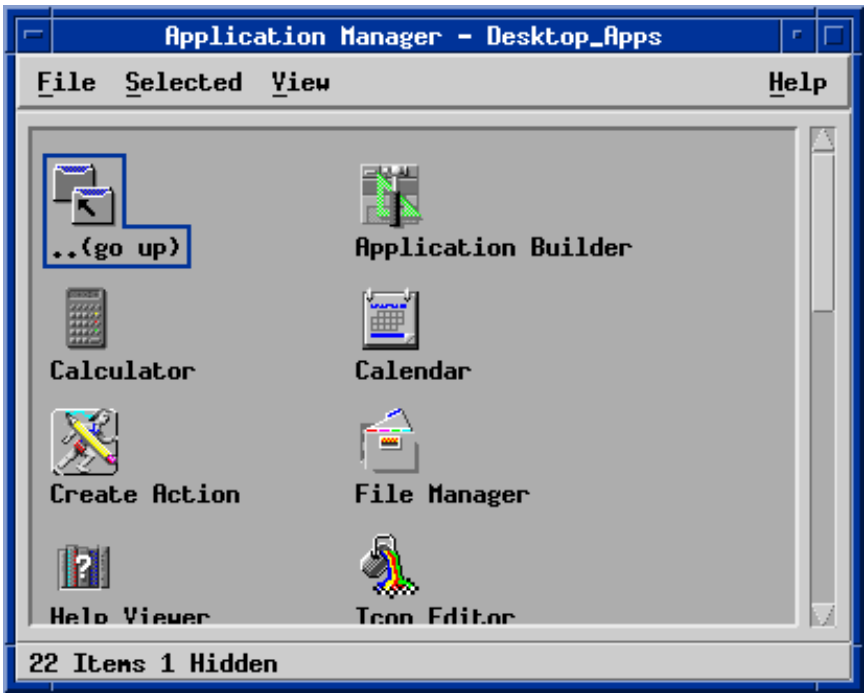
Built-In Application Groups

The desktop provides the built-in application groups shown here.

Application	Description
Desktop_Apps	Applications such as the Calculator, Netscape, and Text Editor
Desktop_Tools	Administration and operating system tools such as Archive, vi Text Editor and Check Spelling
Information	Icons representing frequently used help topics
System_Admin	Tools used to manage the system

When you double-click the Desktop_Apps icon in the Application Manager folder, the system displays the contents of the Desktop_Apps folder, as shown here. This folder contains useful applications such as the Calculator and Man Page Viewer applications.

Figure 2-33: Contents of the Desktop_Apps Application Group



You can start any application by double-clicking its icon.

Running an Application from the Application Manager

To activate an application:

1. Click the Application Manager on the Front Panel.
2. Double-click the application group's icon to display its content.
3. Double-click the application's action icon.

Now Try This!

Start the Calculator application from Application Manager.

1. Click the Application Manager on the Front Panel.
2. Double-click the Desktop_Apps control.
3. Double-click the Calculator control.

Need Help?

Only the controls on the Front Panel need to be single-clicked. The rest must be double-clicked.

Solution

The Calculator should open.

Getting Application Help

You can obtain information about the application by using MB3 on the application's icon and choosing Help from the pop-up menu. You can also choose Help from the Help menu.

Adding an Application to the Front Panel

If you use one of the applications in the Application Manager frequently, you can place the application's icon on any of the Front Panel control subpanels.

To add an application to a subpanel:

1. Click the Application Manager on the Front Panel.
2. Double-click the application group's icon to display its contents.
3. Select the application icon you want to install.
4. Drag the icon to the Install control on the subpanel where you want the application to be.

Adding an Application to the Workspace

You may want frequent access to an application but do not want to place it on the Front Panel. You can place the application on the workspace backdrop and have it available when you do not have an Application Manager window open.

To put an application on the workspace:

1. Open the application group containing the application you want to add to the workspace.
2. Drag the application icon from the Application Manager to the workspace backdrop.

Running an Application Using File Manager

Some applications may be configured to start from the File Manager using any of the application's data files. For example, if you double-click a PostScript file, a PostScript viewer application starts up and displays the PostScript file.

To start an application from File Manager, do one of the following:

- Double-click the application's data file
- Select an application data file and choose Open from the icon's pop-up menu

Now Try This!

Open a file from File Manager.

1. Click the File Manager control on the Front Panel.
2. Enter the `/etc` file path.
3. Find the `hosts` file. It looks like a typed letter (a text file).
4. Double-click the `hosts` file icon.

Need Help?

Choose the Help menu to access online help.

Solution

The `hosts` file should be displayed in Text Editor.

Summary

Using the Front Panel

The CDE desktop consists of:

- Front Panel — special window with controls for starting applications
- Workspace — screen display area

Controls have one or more behaviors:

- Click
- Drop
- Indicator

Getting Help

CDE provides help:

- With a context-sensitive Help key
- In an application's Help menu
- Through the Help Manager
- Through the Man Page Viewer

With CLI, you can use the `man`, `apropos` and `help` commands.

Using Text Editor

The Text Editor allows you to perform file creation and editing functions and has the following features:

- Drag and drop editing
- Find and change
- Spell checking
- Simple formatting
- Save and restore of recovery file

Using Calendar

Calendar is a desktop application used to schedule appointments, make a To Do list, set reminders, browse other calendars and schedule group appointments.

Calendar displays the following views:

Day	Shows one day's appointments
Week	Shows one week's appointments
Month	Shows one month's appointments (abbreviated)
Year	Does not show appointments

Using Terminal

The Terminal application allows you to run applications and commands, and perform copy and paste operations using the UNIX command line interpreter.

Running Applications

The Application Manager provides some of the most frequently used tools and applications available on the system. This is a special view of built-in and registered application groups gathered together when you log in. Double-click an application icon to start the application.

You can also run applications from the File Manager by double-clicking on a data file.

3

Customizing Your Workspace

Unit Overview

Introduction

This unit explains how to set up and customize your CDE workspace for personal preferences. It also describes how to use CLI to set up your terminal or workstation for proper UNIX operation.

Objectives

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

- Customize the Front Panel and workspaces
- Customize the screen appearance and system behavior
- Customize terminal and keyboard setup

Resources

For more information on the topics in this unit, see the following:

- *Common Desktop Environment: User's Guide*
- *CDE Companion*, Chapter 4
- Tru64 UNIX reference pages

Customizing the Workspace

Overview

CDE allows you to customize your desktop. This section explains how to alter the Front Panel and workspaces.

Customizing the Front Panel

You can modify the Front Panel in the following ways:

- Add controls to subpanels
- Switch Front Panel and subpanel controls
- Add and delete subpanels

Now Try This!

Add a new application to the Personal Applications subpanel.

1. Click the Application Manager control.
2. Double-click the Desktop_Apps control.
3. Open the Personal Applications subpanel.
4. Drag and drop the Calculator icon onto the Install icon control.

Now Try This!

Substitute the Text Editor application on the Front Panel with the Terminal control on the Personal Applications subpanel.

1. Open the Personal Applications subpanel.
2. Point to the Terminal control.
3. Press MB3.
4. Choose Copy to Main Panel from the pop-up menu.

Modifying Workspaces

You can modify the workspace switch to:

- Add workspaces
- Delete workspaces
- Rename workspaces
- Add a control

Now Try This!

Add a new workspace to the workspace switch and rename it.

1. Point to an empty area in the workspace switch.
2. Press MB3.
3. Choose Add Workspace from the pop-up menu.
4. Point to the new workspace.
5. Press MB3.
6. Choose Rename from the pop-up menu.
7. Edit the workspace button to read: `Video Project.`

Now Try This!

To remove a workspace:

1. Point to a workspace button.
2. Press MB3.
3. Choose Delete from the pop-up menu.

   If the workspace contains windows, they will move to the next workspace.

Using the Style Manager

Overview

This section discusses how to use the Style Manager to change:

- The appearance of your desktop by changing colors, fonts, and backdrops
- The behavior of your keyboard, mouse and windows
- Your home session, current session and logout confirmation preference

Invoking the Style Manager

When you click the Style Manager control on the Front Panel, the system displays the panel shown here.

Figure 3-1: Style Manager Controls

The Style Manager controls allow you to change these items.

Table 3-1: Style Manager Control Items

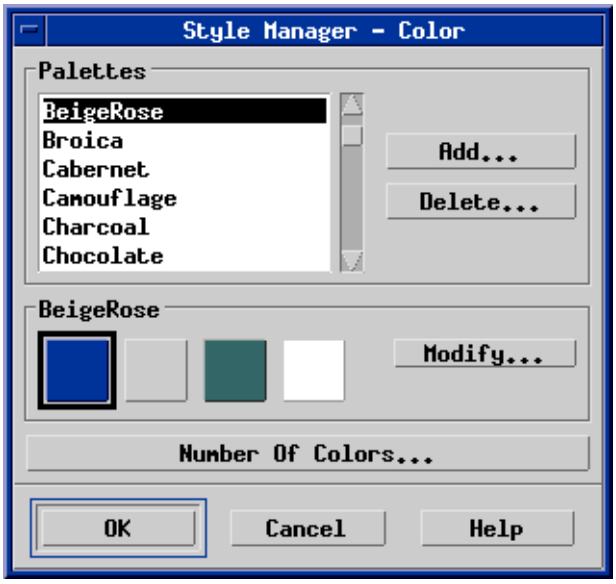
Item	What It Controls
Color	Workspace colors and palettes
Font	Application font sizes
Backdrop	Workspace backdrop patterns
Keyboard	Keyboard click volume and character repeat capability
Mouse	Mouse button click settings
Audio	Beeper volume, tone, and duration
Screen	Time (in minutes) before your screen blanks, and whether or not your system is locked at that time
Window	How a window acquires focus, if the window raises when it receives focus, and where window icons are placed
Startup	How your session begins and ends

Changing Colors

Choose the Style Manager Color control to change the color scheme on your desktop.

The **color palette** is a combination of colors the system uses to draw the elements of your screen. The system provides several color palettes and you can create a custom palette. Click different palette names to see the results.

Figure 3-2: Style Manager — Color

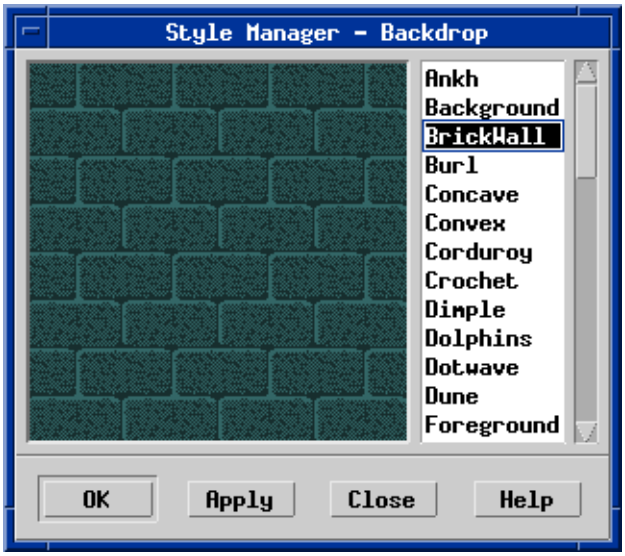


Changing Backdrops

Use the Style Manager Backdrop control to change the appearance of your desktop. You can select a different pattern for each of your workspaces so you can quickly identify them.

In the Backdrop dialog box, use the scroll bar to view the background styles. You can preview the backdrop pattern by clicking the Apply button. Once you are satisfied, click OK.

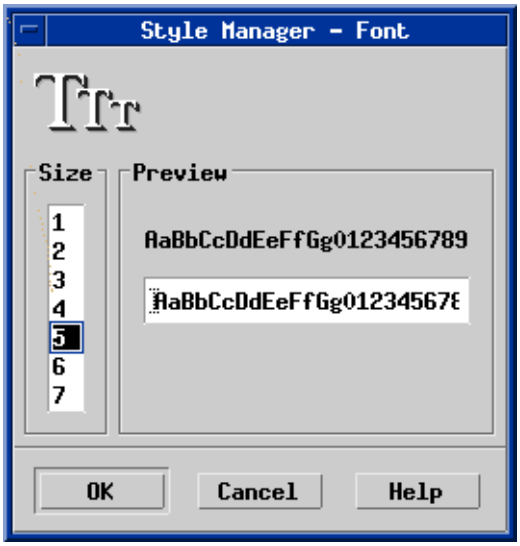
Figure 3-3: Style Manager — Backdrop



Changing Fonts

You can change the size of the font used for window labels and text by using the Style Manager Font control. Select a size and the system displays an example for you to preview. When you start new applications, the system reflects this change. Existing windows do not change.

Figure 3-4: Style Manager — Fonts



Now Try This!

Use the Style Manager controls to change:

- Colors of your desktop

- Backdrop for one of your workspaces
- Size of text

Solution

Use the Color control to set the color palette, the Backdrop control to set the background, and the Font control to set the font size.

Changing Keyboard Settings

To change keyboard characteristics, use the Style Manager Keyboard control. You can change the following:

- Click volume

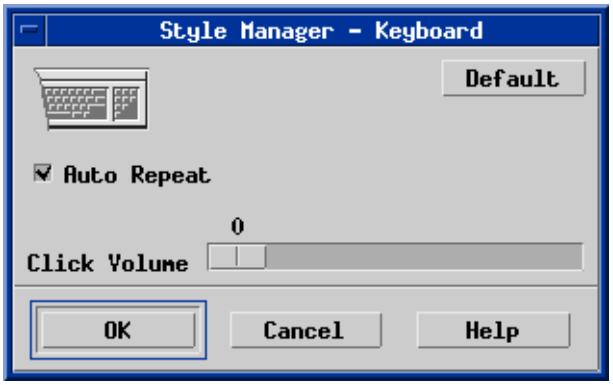
  Use Click Volume to adjust the loudness of the keyboard click. To turn off the click completely, set the scroll bar to a value of zero.

- Key auto repeat

  Click the auto repeat button to turn the auto repeat feature on and off. Turning the feature on causes a key to repeat for as long as it is pressed.

Click Default to return to the default system settings.

Figure 3-5: Style Manager Keyboard Control



Changing Mouse Settings

The Style Manager Mouse control allows you to change:

- Handedness — Settings for a right or left-handed mouse
- Function of mouse button 2 (MB2) — Either Transfer or Adjust

  Transfer means that MB2 will perform a drag and drop operation when you click an object. **Adjust** means that MB2 will allow you to extend your selections to include more than one object. When the function of MB2 is set to Adjust, you can drag and drop with MB1.

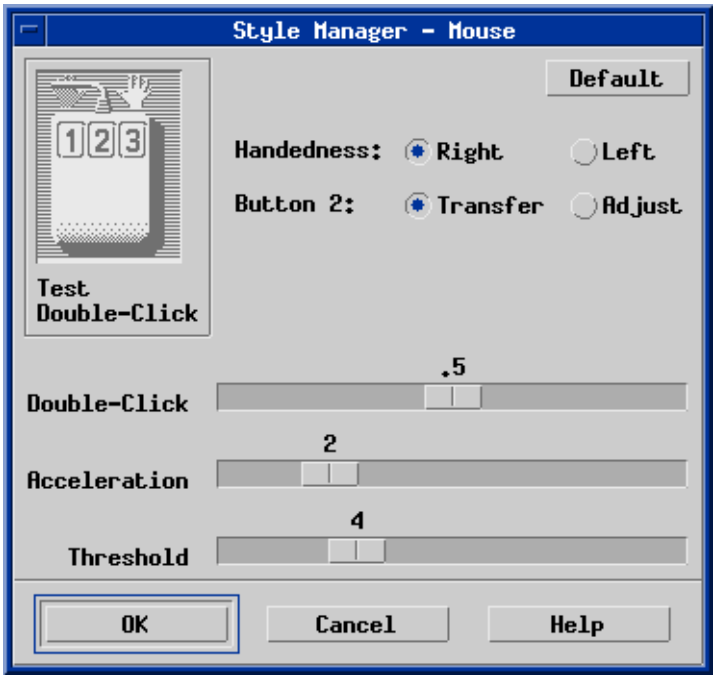
- Double-click — The maximum time between clicks of a double-click

- Acceleration — The speed that the mouse pointer moves across the display
- Threshold — The mouse moves at a slow speed before moving at the accelerated speed

To return to the default system settings, click Default.

The Test Double-Click box allows you to test the double-click setting. It also shows the ordering of mouse buttons for the selected Handedness.

Figure 3-6: Style Manager Mouse Control

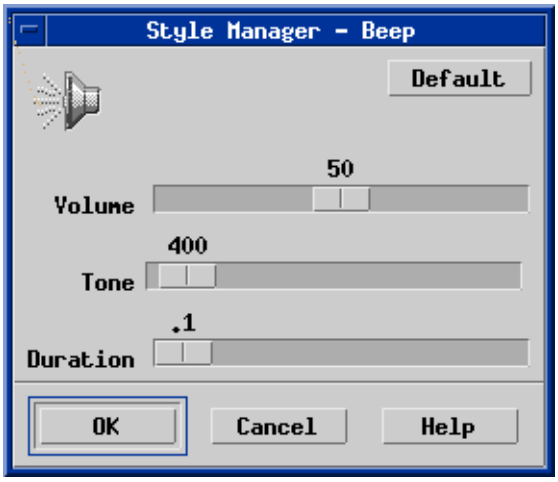


Changing Beep Settings

The Style Manager Beep control allows you to change the Beep settings for:

- Volume
- Pitch
- Duration

Figure 3-7: Style Manager Beep Control

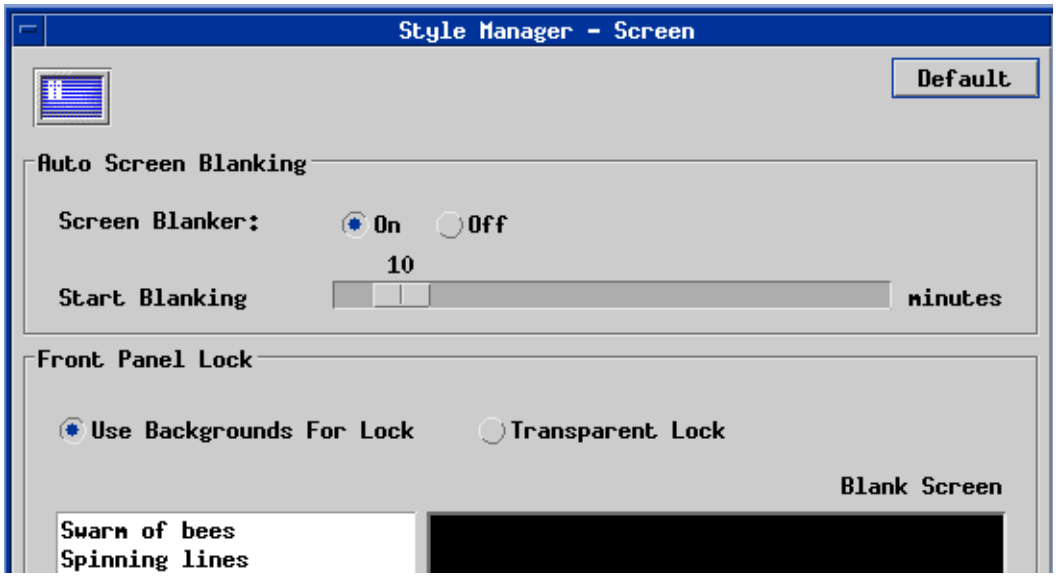


Changing Screen Settings

The Style Manager's Screen control allows you to set:

- One or more screen savers
- The screen saver option on or off
- The time interval before invoking a screen saver
- Automatic locking of your session
- The time interval before locking a session

Figure 3-8: Style Manager Screen Control



Now Try This!

Use the Style Manager's Screen control to set the following and wait a minute to see the results:

1. Exercise group 1:
   - Screen Saver option: off
   - Screen Lock option: on
   - Start Lock: 1 minute
   - Click OK
2. Exercise group 2:
   - Screen Saver option: on
   - Select one or two screen savers
   - Start Saver: 1 minute
   - Time Per Saver: 1 minute
   - Screen Lock option: off
   - Click OK
3. Exercise group 3:
   - Screen Saver option: on
   - Select one or two screen savers

— Start Saver: 1 minute

— Time Per Saver: 1 minute

— Screen Lock option: On

— Click OK

4. Click the Front Panel's Lock control.

Solution

Each operation has a different result. With Screen Lock on and Screen Saver off, the system locks the session, but the desktop is still visible.

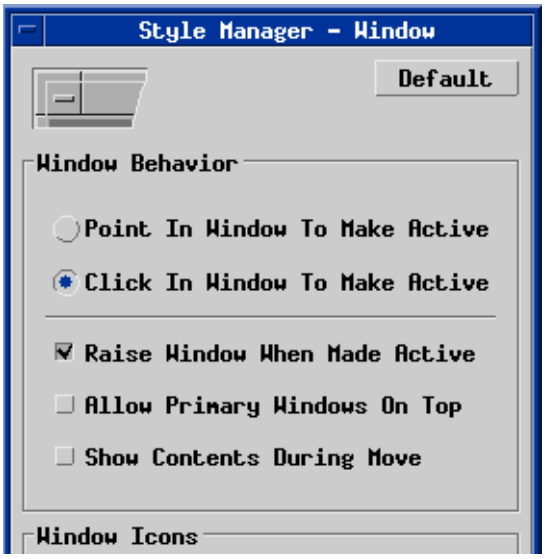
With Screen Lock off and Screen Saver on and a Screen Saver selected, the system hides the desktop and displays the screen saver(s) and does not lock the session.

Changing Window Settings

Use the Style Manager Window control to change:

- Window focus policy
- How active windows are displayed
- Where window icons are displayed

Figure 3-9: Style Manager Window Control



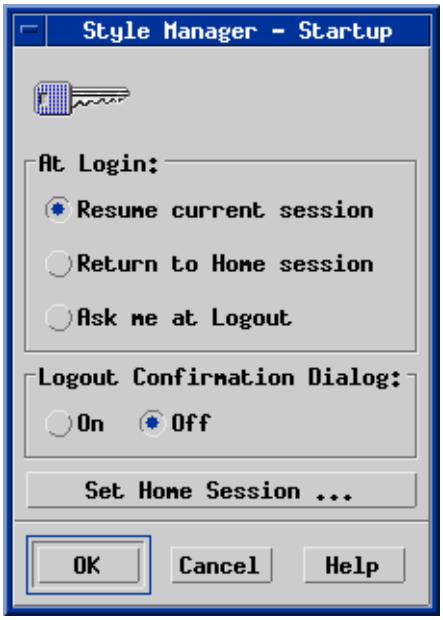
Customizing Startup and Logout

The Style Manager Startup control allows you to configure:

- The session to start up when you log in
- The session you want for your home session

- Your logout confirmation preference

Figure 3-10: Style Manager Startup Control



Setting Up a Terminal

Overview

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

Displaying Keyboard Settings

Use the `stty everything` command to determine terminal settings. Along with keyboard settings, the `stty` (set TTY) command displays the speed of the terminal and settings that affect your screen. (On a Tru64 UNIX system, you can also use `stty -a`.)

Example 3-1: 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
$
```

The caret (^) indicates a control key. For example Ctrl/? is used to erase the previous character. These default keyboard settings may be helpful in modifying incorrect information entered at the keyboard.

Table 3-2: Selected Keyboard Settings

Command	Key	Function
`erase`	Backspace or Ctrl/?	Backs up one space and removes a single character each time it is pressed
`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

When using key functions to erase 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 typed over.

For a complete description of all the information displayed by the `stty everything` command, refer to Appendix A. For more information on the `stty` command and terminal settings, enter the `man stty` command.

Now Try This!

This exercise uses the `stty` command to display your terminal settings. Use the `man stty` command to get additional information.

1. Enter the `stty everything` command to display your terminal settings.
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.

Solution

1. 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. Ctrl/? erases one character; Ctrl/C interrupts (cancels) the command.

3.
```
$ man stty
 stty(1)

NAME
. . .
```

Changing Keyboard Settings

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

When using an operating system with a different keyboard setup, set the `erase`, `kill`, and `intr` functions to something familiar.

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 set it to.

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

Another way to reset a function is to undefine it. For example, to set `kill` undefined, enter:

```
$ stty kill undef
```

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

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

These commands can be put in your `.profile` file to be executed automatically each time you log in.

Now Try This!

This exercise uses the `stty` command to change your keyboard settings. Use the `man stty` command if you need additional information on the `stty` command.

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 the backspace key.

Solution

1. **`$ stty erase ^H`**
2. **`$ stty -a`**

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

3. Pressing Ctrl/H should backspace.
4. **`$ stty erase`** `<backspace>`

Summary

Customizing the Workspace

You can customize the Front Panel in the following ways:

- Add controls to subpanels
- Switch main panel and subpanel controls
- Add and delete subpanels

You can modify the workspace switch to:

- Add workspaces
- Delete workspaces
- Rename workspaces
- Add a control

Using the Style Manager

You can use the Style Manager to change:

- The appearance of your desktop by changing colors, fonts, and backdrops
- The behavior of your keyboard, mouse and windows
- Your home session, current session, and logout confirmation preference

Setting Up a Terminal

Use the `stty -a` command to display your terminal and keyboard settings.

Use `stty` command arguments in your `.profile` file to automatically set up your default terminal and keyboard settings each time you log in.

Summary

4

Introducing File System Concepts

Unit Overview

Introduction

The UNIX operating system uses 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

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 must know limitations regarding naming of files and directories and determine the type of information a file contains.

Objectives

To understand and use UNIX files, you should be able to:

- Identify several file types
- Describe file and directory naming conventions
- Describe the UNIX directory tree structure

Resources

For more information on the topics in this chapter, see the following:

- *Common Desktop Environment: User's Guide*
- *Command and Shell User's Guide*

Identifying Files and Data Types

Overview

A **file** is a container that holds information. This information can be data, records, a spreadsheet, or a document. The format of the data in a file can be arranged in many ways. The format of the file is known as a **data type**. The UNIX file system uses three types of files.

- Ordinary disk files
- Special files
- Directory files

Ordinary Disk Files

An ordinary (or regular) file contains whatever information you place in it.

- ASCII characters or binary data
- A string of bytes of data

The system does not require a particular structure and formatting is left to a user's program or the UNIX command interpreting the file. A file created using an editor is an ordinary file.

A subclass of the ordinary file is the **hidden** file.

- Hidden file names begin with a period (`.profile`, `.kshrc`).
- Each hidden file used by the system performs a special function.

Special Files

UNIX treats all physical devices as if they were special files.

- Each device on a UNIX system has a special file associated with it.
- These special files are normally located in the `/dev` directory.
- Writing to a special file transfers information to the device.

Directory Files

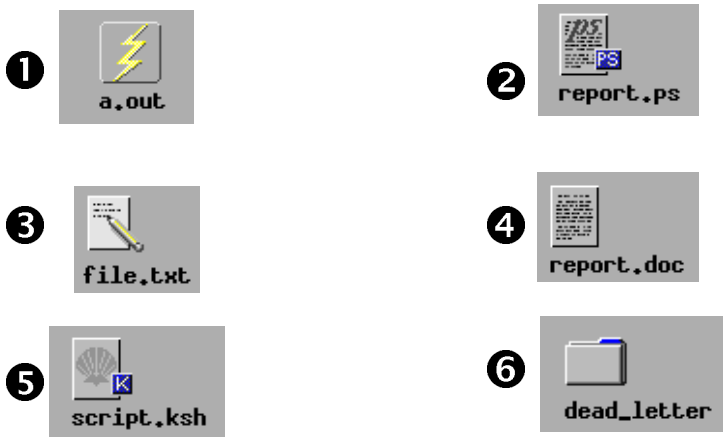
A **directory** is a special file that contains files. Think of it as a file folder containing letters or reports.

Determining Data Types (CDE)

A data type identifies the file's particular format and associates the file with the appropriate application. CDE associates icons with data types. In most cases, a double-click on the icon will open the application and the file.

The icons shown represent the type of data contained in a particular file.

Figure 4-1: Icon Data Types



❶ The lightning bolt represents an executable.

❷ The ps icon represents a PostScript file.

❸ The pencil icon represents a text file.

❹ The text icon is for special and undefined files.

❺ The K and shell icon represents a Korn shell file.

❻ The folder icon represents a folder.

Determining File Types (CLI)

The `file` command shows you the type of a particular file. That is, it provides information such as whether a file contains ASCII text, or program code, 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. The example shows the output of some `file` 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)
$
```

Now Try This!

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

1. `.profile`
2. `/usr`
3. `/`

Solution

1. **`$ file .profile`**
   `.profile:       commands text`
2. **`$ file /usr`**
   `/usr:   directory`
3. **`$ file /`**
   `/:      directory`

Folders

A **folder** is a container for files, similar to a folder in a file cabinet. A folder is the CDE equivalent to a UNIX directory.

A folder can contain subfolders. Within any single folder, each file name must be unique. However, files in different folders may have the same name.

In the CDE environment, as you navigate from folder to folder, your current location is referred to as the **current** folder.

Folders provide the connection between the names of files and the files themselves. They impose a structure on the file system.

- Each folder stores the names of the files in the folder.
- The system maintains several folders for its own use. One of these is the **root** folder, represented by a slash (/). It is the top-level folder.
- Files and folders are often grouped by function.
- All user folders are often grouped together.
- Each user has a folder of files called the **home** folder.

Objects

CDE uses the term **object** to represent a system resource such as a file, directory, or application.

Since files and folders are represented in File Manager as icons, the term object is used to describe them both. Objects are discrete things on the desktop that you can create and manipulate.

On the desktop, applications can also be represented as objects. For example, Application Manager contains objects representing the applications available on your system.

Naming Files and Directories

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 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. This makes the file **hidden**; it does not appear in ordinary directory listings. Many hidden file names are reserved for system use.

The allowable length is system dependent. Traditionally, it can contain up to 255 characters for BSD systems, but only 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 name with a period (.) unless you want it to be hidden.
- Avoid characters with special programming or system meaning, such as those shown.

/	Reserved for root directory
*	Wildcard substitution
?	Single character substitution
[ ]	Selective file name substitution
<	Indicates less than or redirect input
>	Indicates greater than or redirect output
-	Indicates options
$	Indicates argument substitution

'	Turns off special meaning
"	Groups characters into a single argument
&	Performs command in background
!	History substitution
[space]	Confusing in directory listings

You can access files with special naming characters by escaping the special character with a backslash, or putting the entire name in quotes.

- Do not use spaces or periods (.) and (..) alone for file or directory names; these are reserved for system files.
- Upper and lowercase letters are distinct (HELLO, Hello, and hello are three different file or directory names). UNIX is case sensitive.
- The UNIX operating system does not retain old versions of a file if it is updated, although some applications do. For example, the `emacs` editor saves the previous version of a modified file, using the file name with a tilde (~) appended to it.
- You can use part of the file name as a file extension preceded by a period (.) to indicate the contents of the file.

`prog.c`	Source code for a C language program
`prog.out`	Executable file
`report.txt`	Text file
`prog.dat`	Data file

Describing the UNIX Directory Structure

Overview

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

Hierarchical File System

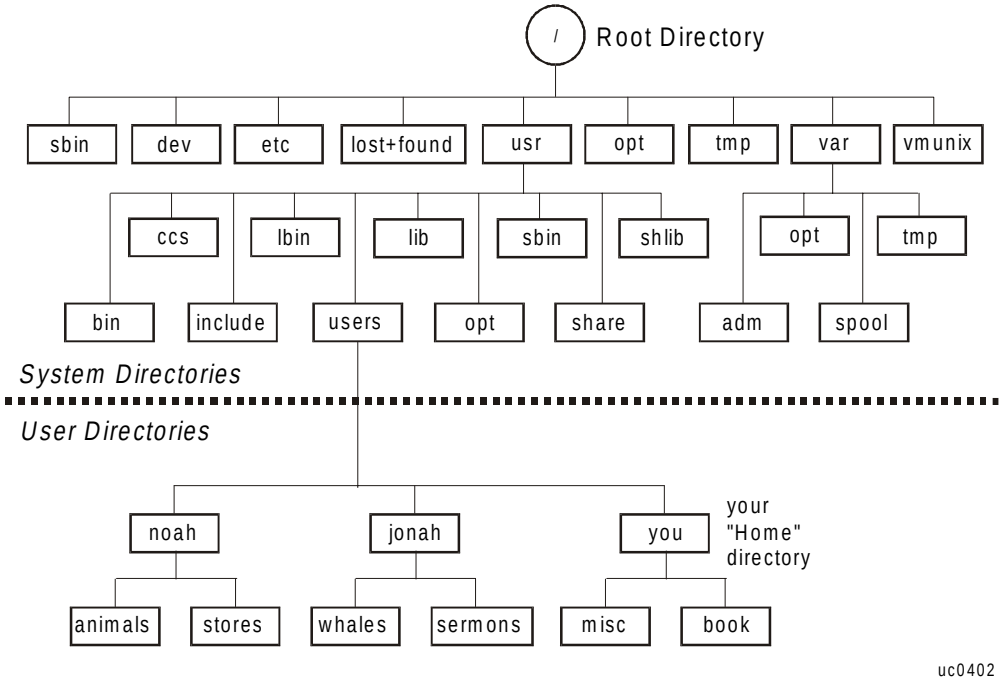
All information stored in the file system hierarchy stems from the **root directory** (/).

- 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.

This figure shows a portion of a representative UNIX file system.

Figure 4-2: Hierarchical File System



Determining the Directories

Directory names do not require an extension and can be indistinguishable from file names.

The CDE File Manager displays a directory as a folder icon.

In command line mode, you can use the `ls` command to display a detailed list of a directory. If the first character of the mode information is a **d**, the file is a directory. In this example, `dogs` and `goats` are directories.

```
$ ls -l
total 2
-rw-r--r--  1 gracie     1990 Nov  4 09:50 cleo
drwxr-xr-x  2 gracie      512 Jul 21 09:54 dogs
-rw-r--r--  1 gracie     1992 Jul 11 08:50 ellie.txt
drwxr-xr-x  2 gracie      512 Sep 28 09:54 goats
-rw-r--r--  1 gracie     1990 Nov  4 09:30 katie.txt
-rw-r--r--  1 gracie     1985 Sep 28 08:50 sally.txt
-rw-r--r--  1 gracie     1987 Oct 15 08:50 sarah.txt
-rw-r--r--  1 gracie     1977 Jul  4 08:50 jessie.txt
-rw-r--r--  1 gracie     1967 May 15 08:50 cookie.txt
```

Pathnames

A file's **pathname** specifies the location of the file within the file system. A pathname consists of a sequence of directory names separated by slashes (/) that end with the file name.

A **full** or **absolute** pathname starts from the root directory. The absolute pathname for the `whales` file (in the preceding figure) is `/usr/users/jonah/whales`.

A **relative** pathname starts from the current directory; for example, `jonah/whales`.

Controlling File Access

Overview

File permissions allow you to restrict access to files or folders. Three groups of users can access files and folders: user (owner), group, and other. File access is divided into three types of permissions: read, write, and execute.

Types of Access

When you try to manipulate a file, the operating system tries to determine what type of permissions the file has. There are three categories of file permissions:

- User

  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

  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

  If neither your user ID nor your group ID matches those of the file, the operating system 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.

Types of Access Permissions

The access permissions on a file specify how that file can be accessed by the owner, group members, and other users.

- Read

  Allows access to retrieve, copy or view the contents of the object.

- Write

  Allows access to change the contents of the object or remove the object.

- Execute

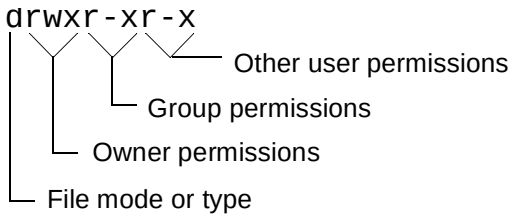
  For a file, allows access to run the file. This includes executable files, scripts, and actions. For a folder or directory, allows access to run commands, scripts and actions within that folder.

Determining Access

With the CDE interface, the folder icons indicate the lack of permissions; for example, a crossed-out pencil indicates no write access. You can also view the permissions in the File Manager by using the Change Permissions option on the Selected menu.

On the command line, the `ls -l` command shows the permissions.

Figure 4-3: File Permissions



Summary

Identifying Files and Data Types (CDE)

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

- Ordinary files, which hold data
- Directories, which hold files
- Special files, which associate physical devices

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

CDE files are represented by icons. The icon type represents the data type of the file. In most cases, you can click the icon to open the application that created the file.

Folders are files used to group and categorize files similar to files contained in folders within a file cabinet.

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 periods (.).
- Avoid spacial characters.
- Upper and lowercase letters are distinct.
- You can use part of the file name as a file extension preceded by a period (.) to indicate the contents of the file.

Describing the UNIX Directory Structure

The UNIX file system is hierarchical. All pathnames 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.

Controlling File Access

File permissions allow you to restrict access to files or folders. Three groups of users can access files and folders: user (owner), group, and other. File access is divided into three types of permissions: read, write, and execute.

5

Managing Files with CDE

Unit Overview

Introduction

File management involves organizing files into directories and subdirectories, as well as moving, copying, renaming, protecting and removing files.

This unit discusses Common Desktop Environment (CDE) methods to:

- Change and list directories
- Locate files based upon file characteristics, such as file name, owner, group, or file size
- Display the contents of a file
- Create, delete, copy and move files
- Change permissions

Objectives

To manage files with CDE, you should be able to:

- Use the File Manager window to locate, access, and manipulate file(s) and folder(s)
- Customize the File Manager display
- Create files and folders
- Delete files and folders
- Copy files and folders
- Move files and folders
- Rename files and folders
- Change ownership and permissions on files and folders
- Link files

Resources

For more information on the topics in this chapter, see the following:

- *Common Desktop Environment: User's Guide*

Using File Manager

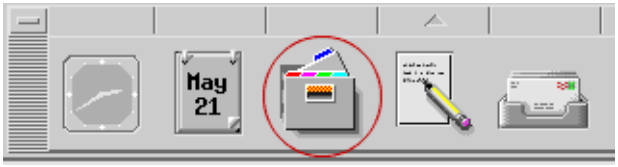
Overview

The File Manager displays the files, folders and applications on your system as icons.

Opening the File Manager

To open a File Manager window, click the File Manager control on the Front Panel.

Figure 5-1: File Manager Control



Now Try This!

1. Click the File Manager control on the Front Panel.
2. Identify the current folder path.

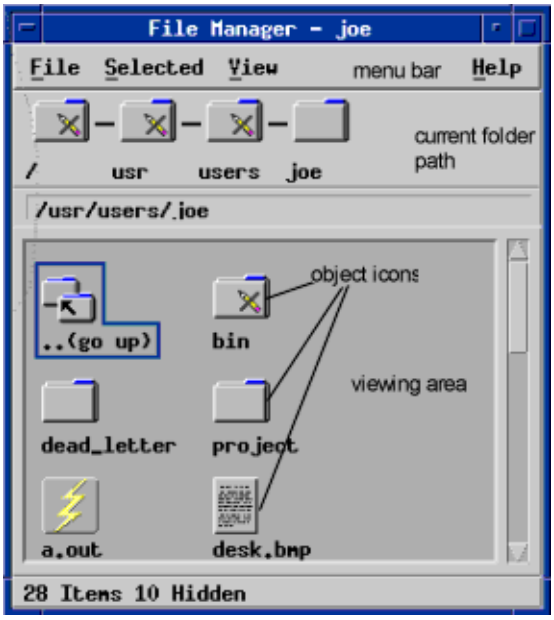
Solution

1. The File Manager window is displayed in the desktop window.
2. The current folder path is likely to be your home directory. It is displayed under the menu bar.

Using the File Manager Window

The File Manager main window is a view of the current folder on your system.

Figure 5-2: The File Manager Window



- Menu bar and menus - Contain the commands available in File Manager
- Current folder path - Displays the path to the current folder
- Object viewing area - Shows the objects (files and folders) in the current working folder
- Object icons - Represent the files and folders in the current folder

Menus

The Menu bar has four menu choices.

File	Creates a file or folder, views a different folder, finds a file or folder
Selected	Manipulates one or more objects
View	Changes the view
Help	Displays help information

Now Try This!

Familiarize yourself with functions within each of the four menu options.

Need Help?

Click a function. Click an option and see what it does. Repeat with a new function/ option combination.

Solution

You can perform many functions using the menu bar.

Changing Folder

To change the current folder, choose one of these methods:

- Double-click a folder in the current folder path or the object viewing area.
- Select one of these menu choices in the File menu.

Go Home	Changes to the home folder
Go Up	Moves up one level in the folder hierarchy
Go To	Prompts for a folder name to display

Selecting a File or Folder

When you select the icon of a file or folder, its name is highlighted. Once an object is selected, several menu options apply to the currently selected object. The contents of the menu change as you select different objects in the view area.

- To select an object using the mouse, click once on the icon using MB1.
- To select an object using the keyboard:
  a. Use the Tab and arrow keys to move the highlight to the icon you want to select.
  b. Press the spacebar to select it.

To deselect a file or folder, select another icon or click an empty area within the File Manager window.

Now Try This!

1. Click the folder labeled `bin` in the object display area.
2. Click the folder labeled `users` in the menu bar area.
3. Display the contents of the `/etc` folder in the File Manager window.

Solution

1. The folder labeled `bin` is highlighted.
2. The object display area displays all the user folders.
3. One way to do this is to use the Go To option on the File menu. The folder `/etc` is displayed in the File Manager window.

Selecting Multiple Files or Folders

You can select multiple icons so you can perform some function on multiple objects. When more than one object is selected, menu options that apply only to a single object are inactive. When multiple icons are selected, dragging any one of the selected icons drags the whole group.

You can use either the mouse or the keyboard to select multiple files or folders.

There are two ways to select multiple objects using a mouse:

- Press MB1 in a blank area of the view, drag the mouse to draw a box around the icons you want to select, then release the mouse button.
- Click MB1 to select the first icon, then hold down the Ctrl key and click MB1 to select each additional icon.

  To remove a single icon from the selected group, hold down the Ctrl key and click the icon you want to remove.

To select multiple objects using the keyboard:

1. Select the first file or folder icon by pressing the spacebar.
2. For each additional icon you want to select, move the highlight to it, then press Ctrl/spacebar.

Dragging and Dropping a File or Folder

To drag and drop a file or folder:

1. Put the mouse pointer over the file or folder.
2. Press and **hold** MB1.
3. Drag the icon to where you want to drop it.
4. Release the mouse button.

To cancel a drag in progress, press Esc before releasing the mouse button.

If you try to drop an icon into a location that does not support dropped objects, the icon snaps back to its original location in File Manager or on the desktop.

Displaying a Pop-up Menu

Each object in File Manager has a pop-up menu. The File Manager window itself has a pop-up menu viewable when the pointer is placed in a location in the view window outside the boundaries of the individual file or folder objects.

To select a pop-up menu using the mouse:

1. Point to the icon whose pop-up menu you want to display and press MB3. On a two-button mouse, press the right mouse button.
2. To choose a command from the menu, drag to the command, then release, or, click the command.

To cancel the menu without choosing a command, click an empty area within the File Manager window.

Opening a File or Folder

The most basic action you can perform on an object is to open it. Opening a data file usually starts the application that created the file and loads the data file. Opening a folder displays the contents.

There are several ways to open a file or folder.

- Double-click the icon.

  Double-clicking an icon executes the object's default action, which is always the first action in the Actions portion of the Selected menu. For most data files, the default action opens the file by starting the appropriate application and loading the file.

- Select the icon, then go to the menu bar and choose Open (or Open In Place or Open New View) from the Selected menu.

- Choose an Open item from the icon's pop-up menu (displayed by pressing MB3).

Now Try This!

1. Double-click one of the folders.
2. Find the folder `/etc` and open a file called `passwd`.

Solution

1. The folder is opened
2. The text editor opens the `passwd` file.

Locating Files

Overview

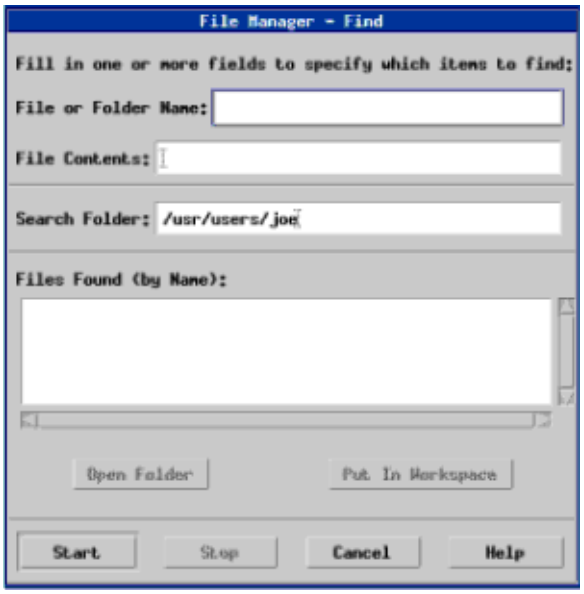
File Manager keeps track of files by allowing you to search for a file or folder by name. You can also search for files based on the contents of the file.

Finding an Object by Name

To find an object based on the name:

1. Choose Find from the File menu.

Figure 5-3: File Manager Find Dialog Box



2. Type the name of the file or folder you want to find in the File or Folder Name field.

   When you specify a file or folder name, you can include wildcard characters, such as asterisk (*) and question mark (?). The * matches **any** string of characters (including no characters), and ? matches any **single** character.

3. Type the folder where you want the search to begin into the Search Folder field. By default, this field contains the name of the current folder.

   Find will search this folder and all of its subfolders.

4. Click Start or press Return.

   File Manager searches the Search Folder and the folders it contains for files that match your request. Matches appear in the Files Found list. Once an object is found, you can select it and click Open Folder to open it or Put in Workspace to place its icon on the current backdrop.

5. To stop the search, click the Stop button or press Return.

Finding a File by Contents

1. Choose Find from the File menu.
2. *Optional*. You can use the File or Folder Name text field to narrow, and therefore speed up, the search.

   If you enter a partial name using wildcards, File Manager examines only the files that match the File or Folder Name field. If you leave the field empty, File Manager searches every file within the search folder.
3. Type the text string you want to search for in the File Contents field.

   Case is ignored for this string. You do not have to use complete words. For example, if you type `fi` it will find both **fish** and **File**.

   Contents can be specified using the same **regular expression** syntax allowed by the `grep` command. Refer to the `grep` reference page for more information.
4. Type the name of the folder where you want the search to begin into the Search Folder field if it is different than the current (default) folder.
5. Click Start or press Return.

   File Manager searches the Search Folder and the folders it contains for files that match your request. Matches appear in the Files Found list. Once an object is found, you can select it and click Open Folder to open it or Put in Workspace to place its icon on the current backdrop.
6. To stop the search, click the Stop button or press Return.

Now Try This!

1. Search the `/usr/etc` folder for the `ping` file.
2. Search the `/usr/users` directory for files containing your first name.
3. Search the `/etc` folder for files named `motd` and display these files on the terminal.

Need Help?

1. Click File. Click Find. In the File or Folder Name field, type `ping`. Tab to the Search Folder field and type `usr/etc`. Click Start or press Return.
2. Click File. Click Find. In the File or Folder Name field, type your first name. Tab to the Search Folder field and type `usr/users`. Click Start or press Return.
3. Click File. Click Find. In the File or Folder Name field, type `motd`. Tab to the Search Folder field and type `etc`. Click Start or press Return.

Solution

1. `ping` is displayed in Files Found.
2. You will receive a message saying no files were found.

3. The files found are:

```
/etc/zoneinfo/motd
/etc/motd
```

Customizing File Manager Views

Overview

The File Manager provides several ways to view the contents of folders. You can:

- Display folders in tree view or folder view
- Change the sort order of objects
- Show or hide certain files or folders
- Change the style used for objects

Displaying the File Tree

The tree view resembles an outline. Files and folders contained in a folder are listed beneath that folder in an indented list.

Figure 5-4: File Manager Tree View

In the tree view, a folder can have three states. The states are shown and changed using the buttons to the left of the folder. You can also open and close the tree branches by selecting a folder and pressing the plus (+) and minus (-) keys on your keyboard, not the numeric keypad.

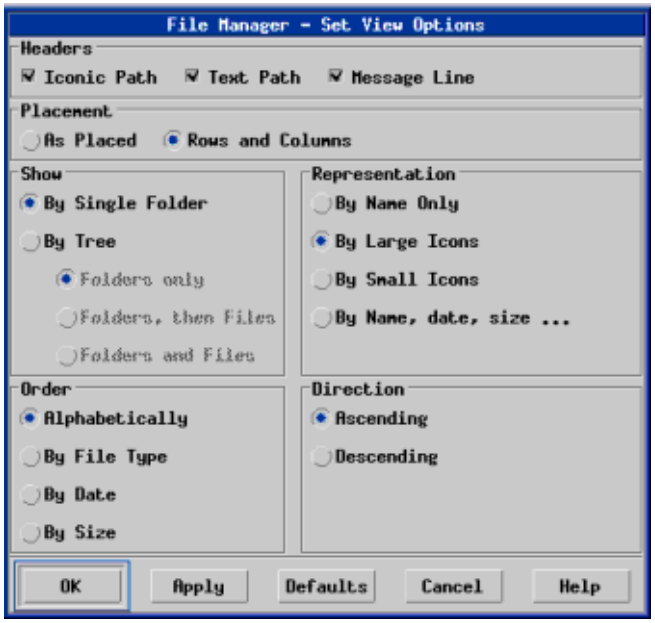
The button states are shown here.

Key	Description
+	The folder is in its closed state. None of the folder's contents are shown. Clicking the button expands the folder partially or fully, depending on the tree view option currently in effect.
-	The folder is in its fully expanded state. All objects are displayed. Clicking the button fully closes the folder.
+/-	The folder is in its partially expanded state. The only contents shown are the folders it contains. Clicking the button expands the folder to show the file names.

Displaying the tree view will show a folder and the files and folders beneath it. To display the tree view:

1. Change to any folder where you want the view to start.
2. From the View menu, choose Set View Options.
3. Select By Tree in the Show box.

Figure 5-5: Set View Options Dialog Box



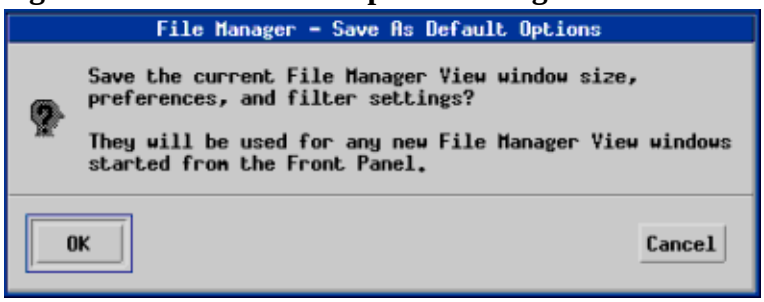
4. Select one of the tree view options.

Option	Description
Folders only	The tree shows folders only. To view files, double-click a folder name. This is the default view.
Folders, then files	Folders only appear at first. Click the + button next to each folder to view its contents. First click shows its subfolders. Second click shows files that are inside. When you click the button the third time, the folders contract again.
Folders and files	The tree shows both folders and files automatically.

5. To implement the chosen options and close the Set View Options dialog box, click OK.

You can change the appearance of files and folders by changing the settings in the Show box in the Set View Options dialog box. If you want the changes to last for more than the current session, choose Save as Default Option from the View menu.

Figure 5-6: Save View Options Dialog Box



If you double-click a folder in tree view, a new view of that folder opens, which is not in tree view mode.

Now Try This!

1. Change the display to show the tree view.
2. Change the display to show the folder view.
3. Change the display to show File Manager folders and files by name and size.
4. Change the display to show File Manager folders and files as large icons.

Need Help?

1. Click File Manager. Click View. Click Set View Options. Select By Tree. Click Apply. Click OK.
2. Click File Manager. Click View. Click Set View Options. Select By Single Folder. Click Apply. Click OK.
3. Click File Manager. Click View. Click Set View Options. Select By Name, date, size. Click Apply. Click OK.
4. Click File Manager. Click View. Click Set View Options. Select By Large Icons. Click Apply. Click OK.

Solution

1. File Manager is displayed in tree view.
2. File Manager is displayed in folder view.
3. File Manager folders and files are displayed by name, in size order.

File Manager folders and files are represented as large icons.

Showing and Hiding Files and Folders

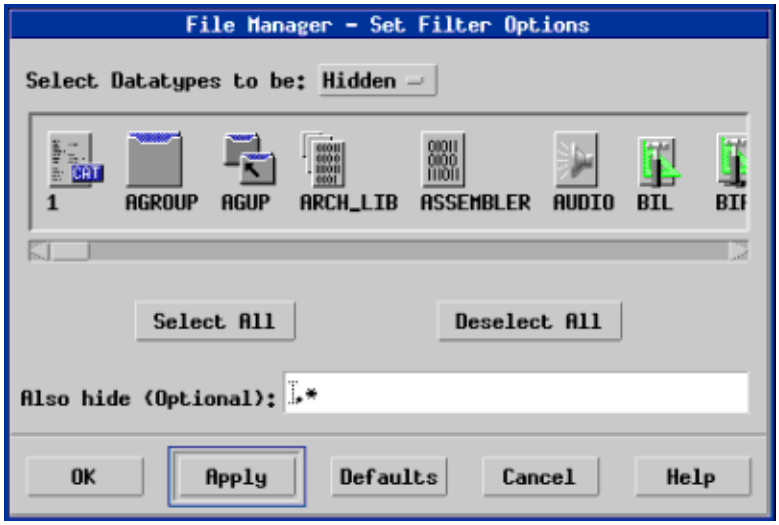
Hidden files and folders are objects whose file types are selected in the filter list.

The criteria for hiding or showing a file or folder is based on its data type. Use the Set Filter Options command to change which data types are shown and hidden.

Specifying which Data Types are Hidden

1. Choose Set Filter Options from the View menu.

Figure 5-7: File Manager-Set Filter Options



Chosen data types to be hidden are highlighted.

2. In the Select File Types to be Hidden box, select the data types you want to be hidden in addition to the objects already selected.

   Or, click Defaults to reset the default filter list.

3. **Optional**. Type a name pattern into the Also Hide (Optional) field specifying additional file and folder names to be hidden.

   The filter list specifies which files are not to be displayed. If you select all object types or type * into the Filter String field, no files are displayed. If you type `.txt`, any file whose name ends in `.txt` will be added to the filter list and not displayed.

4. Click OK.

   To view the results before closing the dialog box, click Apply.

   To reset the default filter list, choose Defaults. This resets the default values but does not apply them until you click Apply or OK.

Creating Files and Folders

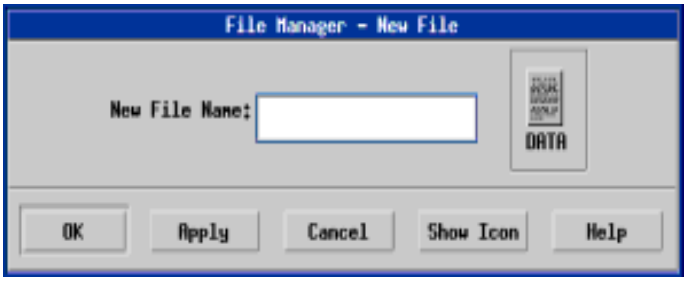
Creating a File

Use the New File menu item to create a new file.

To create a new file in File Manager:

1. Choose New File from the File menu.
2. Type a name into the New File Name: field.

Figure 5-8: Creating a New File



3. Click OK or press Return to save the file **or** to close the New File dialog box without creating a new file, click Cancel or press Esc.

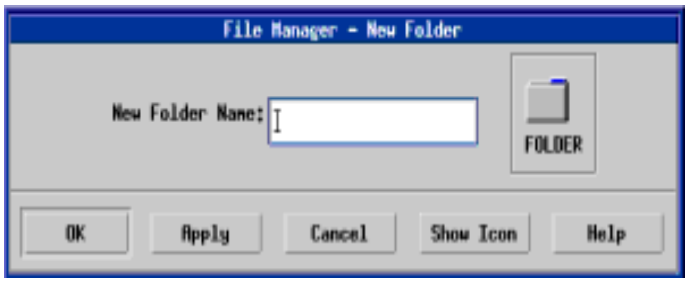
Creating a Folder

Use the New Folder menu item to create new folders.

To create a new folder in File Manager:

1. Choose New Folder from the File menu.
2. Type a name into the New Folder Name: field.

Figure 5-9: Creating a New Folder



3. Click OK or press Return to save the folder name **or** to close the New Folder dialog box without creating a new folder, click Cancel or press Esc.

Now Try This!

1. Create a new file called `fruits.txt` and a file called `vegetables.txt`.

   a. Select New File from the File menu.

   b. Type `fruits.txt` in the New File Name box. Click Apply.

   c. Type `vegetables.txt.` Click OK.

2. Create a new folder called `foods`, another called `drinks` and a third called `snacks`.

   a. Select New Folder from the File menu.

   b. Type `foods` in the New Folder Name box. Click Apply.

   c. Type `drinks` in the New Folder Name box. Click Apply.

   d. Type `snacks` in the New Folder Name box. Click OK.

Solution

1. Files `fruits.txt` and `vegetables.txt` are shown as icons.
2. Folders `foods`, `drinks`, and `snacks` appear as folders.

Deleting Files and Folders

Overview

The Trash Can collects the files and folders that you delete. These objects are not actually removed from the system until the trash is emptied. You can change your mind and restore a file you have put in the Trash Can if the Trash Can has not been emptied.

Using the Trash Can

You can open the Trash Can window by clicking the Trash Can in the Front Panel.

Figure 5-10: Trash Can Icon on the Front Panel

Deleting an Object

To move an object to the Trash Can, perform **one** of the following steps:

- Choose Put in Trash from the object's pop-up menu, displayed by pressing Shift/F10 or MB3.
- Select the object's icon, then choose Put in Trash from the Selected menu on the File Manager menu bar.
- Drag and drop the object's icon onto the Trash Can icon in the Front Panel.
- Drag the object's icon to the open Trash Can window.

Retrieving an Object

To retrieve an object from the Trash Can, perform **one** of the following steps:

- Drag the object from the Trash Can window to File Manager.
- In the Trash Can window, select the object you want to restore and choose Put back from the File menu.
- Choose Put back from the object's pop-up menu, displayed by pressing Shift/F10 or MB3.

Now Try This!

1. Choose New Folder from the File menu in File Manager. Type `prepared_foods` in the New Folder Name box. Click OK.
2. Double-click the `prepared_foods` folder. Now the empty folder is displayed in the File Manager window. Choose New File from the File menu in

File Manager. Type `pizza` in the New File Name box and click Apply. Next, type `liver` in the New File Name box and click OK.

3. Select Go Home from the File menu. Click and hold the folder `prepared_foods` using MB1. Drag the folder on top of the Trash Can icon and release MB1.

4. Double-click the Trash Can icon. Select the file `pizza` and drag the file to the `prepared_foods` folder.

Solution

1. Folder `prepared_foods` is listed.

2. Files `liver` and `pizza` are listed in folder `prepared_foods`.

3. Folder `prepared_foods` is no longer shown in the home directory; it is now part of the Trash Can.

4. File `pizza` is listed in the `prepared_foods` folder.

Permanent Delete

When you delete a file or folder permanently, it cannot be recovered unless you have a backup of the object.

1. Open the Trash Can.

2. Select the objects you want to empty from the Trash Can. Select individual objects, or choose Select All from the File menu.

3. Choose Shred from the File menu or from the object's pop-up menu, displayed by pressing Shift/F10 or MB3.

4. Click OK in the confirmation dialog box.

The Trash Can is automatically emptied when you log out of a session.

Copying and Moving Files and Folders

Overview

You may want to make a backup copy of a file before you make any revisions to the file. When the system makes a copy of a file, the original file remains intact and a new file is created with a new name.

Copying Using the Mouse

To copy a file or folder using a mouse:

1. Make the destination folder visible.
   - — Open a File Manager view of the contents of the destination folder, **or**
   - — Open a File Manager view that shows the icon of the closed destination folder
2. Select the file or folder to be copied.
3. Press and hold the Ctrl key.
4. Drag the file or folder and drop it onto the destination folder.

   Make sure that you release the mouse button before you release the Ctrl key. Otherwise, you will move the file or folder instead of copying it.

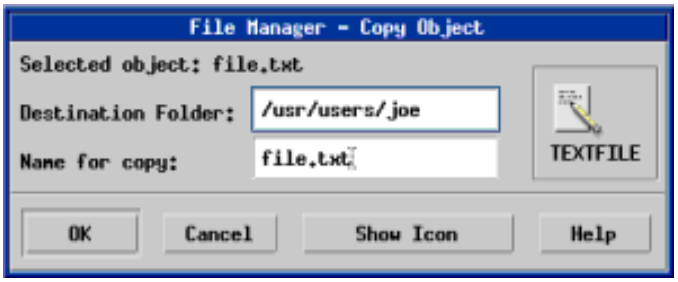
Copying Using the Keyboard

To copy a file or folder using the keyboard:

1. Select the icon.
2. Choose Copy to from the Selected menu.
3. Type a name into the Destination Folder: and Name for copy fields.

   For example, to copy the file `file.txt` to a new location, you could type `/reports/file.txt`, as shown.

Figure 5-11: File Manager-Copy Object



4. Press Return.

   To close the Copy File dialog box without copying a file press Esc.

Now Try This!

Copy the files `fruits` and `vegetables` into the `snacks` folder.

Need Help?

Select the file `fruits`, press the Ctrl key and drag the icon to the `snacks` folders. Repeat with the file `vegetables`.

Solution

The files `fruits` and `vegetables` exist in the folders `snacks` and `foods`.

Moving with the Mouse

To move a file or folder using a mouse:

1. Make sure the source and destination are visible.
   - — Open a File Manager view of the contents of the destination folder, **or**
   - — Open a File Manager view that shows the icon of the closed destination folder
2. Select the file or folder to be moved.
3. Drag the file or folder and drop it onto the destination folder.

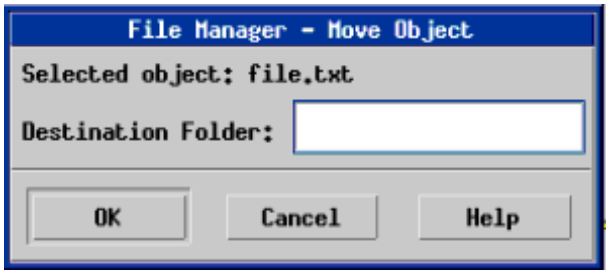
If you move a folder, the entire folder and its contents are moved.

Moving with the Keyboard

To move a file or folder using the keyboard:

1. Use the Tab and direction keys to highlight the file or folder to be moved.
2. Press spacebar to select the object.
3. Choose Move to from the Selected menu.
4. Type a name into the Destination Folder: text field, as shown.

Figure 5-12: File Manager-Move Object



For example, to move the file `file.txt` into the folder `/users/joe`, type `/users/joe`.

5. Click OK or press Return.

If you move a folder, the entire folder and its contents are moved.

Now Try This!

1. Create the following files: `milk`, `soda`, and `juice` in the `foods` folder.
   a. Double-click the `foods` folder.
   b. Choose New File from the File menu.
   c. Type `milk` in the New File Name box. Click Apply.
   d. Type `soda` in the New File Name box. Click Apply.
   e. Type `juice` in the New File Name box. Click OK.
2. Move the files `milk`, `soda`, and `juice` from the `foods` folder into the `drinks` folder.
   a. Double-click the `foods` folder.
   b. To select the files, click `milk`, then
   c. Press and hold the Ctrl key.
   d. Click with MB1 on the `soda` and then the `juice` icon.
   e. Release the Ctrl key.
   f. Choose Move from the Selected menu.
   g. Type in the destination folder and click OK.

Solution

1. Files `milk`, `soda`, and `juice` are listed in folder `foods`.
2. Files `milk`, `soda`, and `juice` are listed in folder `drinks`.

Renaming Files and Folders

Overview

You can rename a file or folder using one of the following methods.

- Select the object and choose Rename from the Selected menu in File Manager.
- Select the object. When the object's name is highlighted, you can type the new name directly into the name field. Press Return when you are finished, or press Esc to cancel the operation.

Figure 5-13: Renaming an Object

Now Try This!

1. Create a file called `soups`.
2. Rename file `soups` and call it `bread`.

Need Help?

Click the `soups` icon. When the object's name is highlighted, type the new name directly into the name field and press Return.

Solution

1. A file is created called `soups`.
2. The file is renamed; it is now called `bread`.

Changing Ownership and Permissions

Overview

File permissions allow you to restrict access to files or folders. Three groups of users can access files and folders: owner, group, and other. File access is divided into three types of permissions: read, write, and execute.

Viewing Permissions

The folder icon represents the different types of access permissions. If you do not have permission to write inside a folder, the folder will look like the next figure.

Figure 5-14: Folder without Write Access

If you do not have read or execute permission for a folder, the folder will look like the next figure.

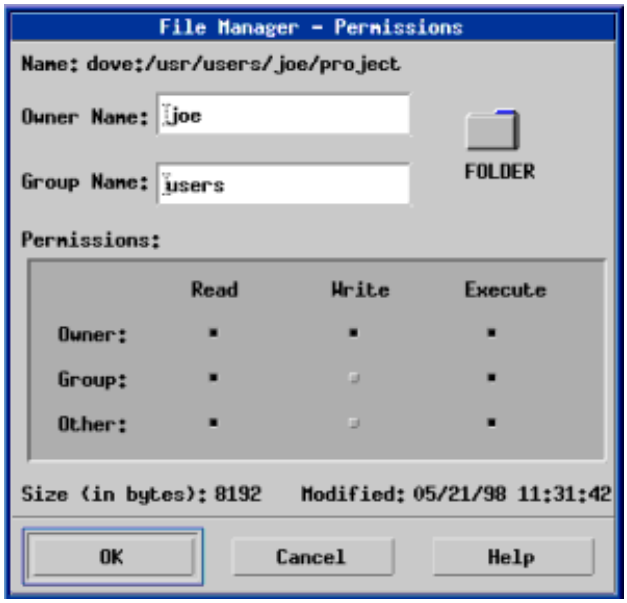
Figure 5-15: Folder without Read and Execute Access

If you have read, write, and execute permission for a folder, the folder will look like the next figure.

Figure 5-16: Folder with Read, Write, and Execute Access

Select a file or folder and choose Change Permissions from the Selected menu. The Permission dialog box opens. The Permissions section lists the types of access in the left column and the types of access permissions in the top row. Cross reference the access types and access permissions to determine the permissions of a file or folder.

Figure 5-17: File Manager Permission Dialog Box



The default permissions are set by the system administrator. To determine your default permissions, create a new file or folder, then open the Permissions dialog box for that file or folder.

Changing the Owner

You must be the owner of the file or a system administrator (root user) to change the ownership of a file. If you have the authority to log in as the root user, you must do so before beginning the following procedure. To change ownership of a file or folder:

1. Select the icon of the file or folder.
2. Choose Change Permissions from the Selected menu or from the icon's pop-up menu displayed by pressing Shift/F10 or MB3.
3. Type the new owner's name into the Owner Name text box.
4. Click OK or press Return.

If you give ownership of a file or folder to another user, you cannot change its permissions again unless that user returns ownership to you.

Changing Permissions

You must be the owner or the system administrator (root user) to change the permissions of a file or folder. To change permissions:

1. Select the icon for the file or folder.
2. Choose the Change Permissions command from the Selected menu or from the icon's pop-up menu displayed by pressing Shift/F10 or MB3.

3. Select the permissions for the file or folder.

   The permissions selected in the Group row indicate the access privileges for any user belonging to the named group. The permissions selected for the Other row apply to all other users.

Read	File can be read
Write	File can be altered
Execute	File can be run

4. Click OK or press Return.

The Group row of permissions shows the permissions for users who are members of the group listed in the Group Name text box. You can specify a different group by typing a new name in this box.

If you do not have permission to change the properties of a file, some of the controls in the File Properties dialog box are inactive.

Now Try This!

1. Copy the file called `group` located in `/etc` to your local directory and call the file `groupfile`.
2. Display the permissions of the file `groupfile`. Click the file `groupfile` and choose Change Permissions from the Selected menu.
3. Change the permissions on the file `groupfile` to the owner having read, write and execute, the group having read, and write, and other has execute permissions. Verify the changes.
4. 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 the changes.

Need Help?

In the Permissions dialog box, in the Owner row, click the Read, Write, and Execute boxes so the appropriate boxes are selected or deselected. In the Group row, click Read and Write. In the Other row, click Execute.

Solution

The `groupfile` file icon changes from an executable (lightning bolt) to an undefined file.

Linking Files and Folders

Overview

A link icon is a copy of an icon that points to the same file or folder as the original icon. Any changes you make after opening the link icon will also appear when you access the file or folder using the original icon.

Creating a Symbolic Link Using the Mouse

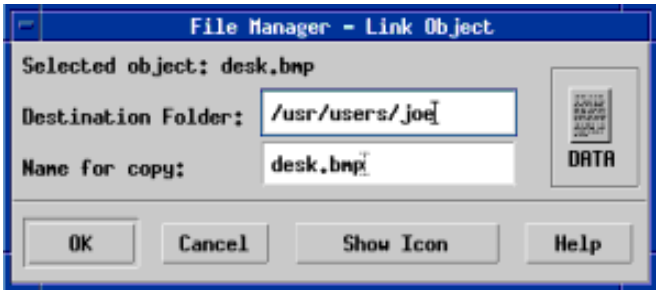
To create a symbolic link using the mouse:

1. Make the source and destination folders visible.
   - — Open a File Manager view of the contents of the destination folder, **or**
   - — Open a File Manager view that shows the icon of the closed destination folder
2. Select the file or folder to be copied.
3. Press and hold the Shift key and the Ctrl key while dragging and dropping the icon of the file or folder onto the icon of a new folder.

When you drop the icon, File Manager creates a symbolic link in the new folder that points to the original file or folder.

Make sure that you release the mouse button before you release the Shift key and the Ctrl key. Otherwise, you will move instead of link the file or folder.

Figure 5-18: Copy as Link



Creating a Symbolic Link Using the Keyboard

1. Use the Tab and arrow keys to highlight the icon of the file or folder to which you want to link.
2. Press spacebar to select the icon.

3. Choose Copy as Link from the Selected menu.
4. Type a name into the Destination Folder: text field including the path for the name of the folder where you want the link icon to appear. To give this link icon a different name than the original icon, type a new name in the Name for copy: text field.
5. Click OK or press Return.

Now Try This!

1. Copy the file `/etc/passwd` to a file in your home directory called `dummy`.
2. Create a folder called `links` in your home directory.
3. Create a link from file `dummy` to folder `links`.
4. Open file `dummy` which now resides in folder `links`.

Need Help?

To create a link, select file `dummy`, press and hold Shift/Ctrl while dragging and dropping it into the folder `links`.

Solution

1. The file `/etc/passwd` is copied into your folder and called `dummy`.
2. A new folder called `links` is created in your area.
3. A link is created from file `dummy` in folder `links` to your working directory.
4. File `dummy` is now open.

Summary

Using File Manager

Start File Manager by clicking the File Manager icon on the Front Panel.

The File Manager window has menus for command options, and shows the current folder path and the object icons in it. You can:

- Change folder path by double-clicking a folder icon.
- Select one or more objects to manipulate.
- Drag files and folders.
- Double-click the file or folder icon to open it.

Locating Files

File Manager keeps track of files by allowing you to search for a file or folder by name. You can also search for files based on the contents of the file.

Use Find from the File menu to search for files.

Customizing File Manager Views

The File Manager provides several ways to view the contents of folders. You can:

- Display folders in tree view or folder view
- Change the sort order of objects
- Show or hide certain files or folders
- Change the style used for objects

File Manager can be customized to display files and folders in tree and folder views. Files and folders can be displayed or hidden from view.

Creating Files and Folders

To create a new file, choose New File from the File Manager menu. Type a file name. Click OK.

To create a new folder, choose New Folder from the File Manager menu. Type a folder name. Click OK.

Deleting Files and Folders

The Trash Can collects the files and folders that you delete. These objects are not actually removed from the system until the trash is emptied. You can change your mind and restore a file you have put in the Trash Can if the Trash Can has not been emptied.

To delete files or folders, drag and drop the object on the Trash Can. To return the file, open the Trash Can and drag and drop the file to File Manager. Files are deleted permanently when the trash is shredded or you log out of a session.

Copying and Moving Files and Folders

Files and folders are copied by selecting the object, then pressing and holding the Ctrl key and dragging and dropping the object into the destination folder.

Files and folders are moved by selecting the object and dragging and dropping the object into a destination folder.

Renaming Files and Folders

You can rename a file or folder using one of the following methods.

- Select the object and choose Rename from the Selected menu in File Manager.
- Select the object and type the new name directly into the name field. Press Return when you are finished, or press Esc to cancel the operation.

Changing Ownership and Permissions

File permissions restrict access to files and folders by owner, group and other. The types of permissions are read, write and execute.

To change the ownership of a file or folder, select the object and choose Change Permissions from the Selected menu, type the new owner, and click OK.

To change permissions, select the object and choose Change Permissions from the Selected menu, select the permissions, and click OK.

Linking Files and Folders

A link icon is a copy of an icon that points to another file. To create a link, select the object to be linked, hold Shift/Ctrl and drag and drop the icon to the new folder. The linked icon points to the original file.

6

Managing Files with CLI

Unit Overview

Introduction

File management involves organizing files into directories and subdirectories, as well as moving, copying, renaming, protecting and removing files.

This unit discusses command line interface (CLI) methods to:

- Change and list directories
- Locate files based upon file characteristics such as file name, owner, group, or file size
- Display the contents of a file
- Create, delete, copy and move files
- Change permissions

Objectives

To manage files with commands, you should be able to:

- Determine the current working directory using the `pwd` command
- Change directories using the `cd` command
- List and find files and directories using the `ls` and `find` commands
- Display files using the `cat`, `more`, `head`, and `tail` commands
- 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 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

For more information on the topics in this chapter, see the following:

- *Command and Shell 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 use the `cd` command, you must first understand the concepts of:

- Home and working directories
- Pathnames

Home Directory

The **home** directory is usually identified by a pathname that includes the user name as its last component. When you log in to your account on a UNIX operating system, you are automatically placed in your home directory. Your home directory never changes.

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.

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

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

Pathnames

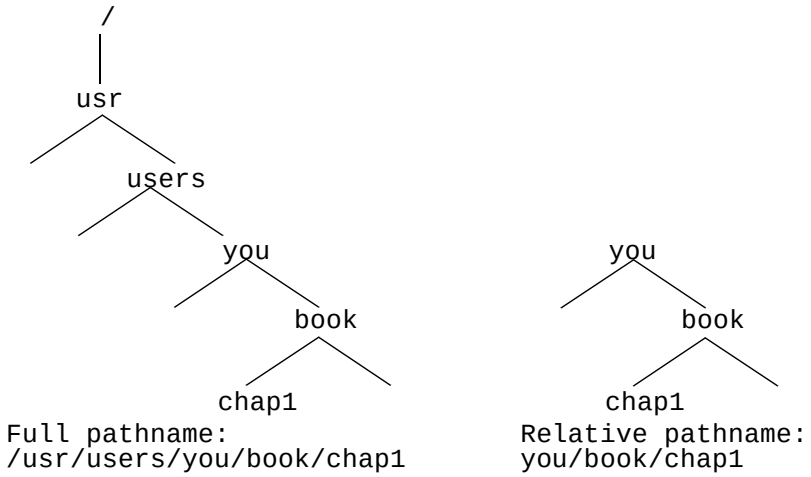
A **pathname** represents the directories and subdirectories that are passed through to reach a specified file or directory. It represents the path of a directory from a specific point in the directory structure. There are two types of pathnames: full pathnames and relative pathnames.

Full pathnames are also known as **absolute** pathnames. Full pathnames always begin at the root (/) directory.

Relative pathnames are based on your current working directory. For example, if your current working directory is your home directory, 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`.

This figure illustrates full and relative pathnames for accessing a file named `chap1` from the `/usr/users` directory.

Figure 6-1: Full and Relative Pathnames



Using the cd Command

There are two special pointer files:

- Dot (.) refers to the current directory.
- Dot dot (..) points to the preceding level in the directory hierarchy (parent directory)

To change to your home directory from anywhere:

```
$ cd
```

To change your working directory to subdirectory `book`:

```
$ cd book
```

To change your working directory to the `/usr/users/misc` directory:

```
$ cd /usr/users/misc
```

To change your working directory to the parent directory:

```
$ cd ..
```

To change your working directory to two directories above the current directory:

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

To change from directory `/usr/users/stu1` to directory `/usr/users/stu2`.

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

Now Try This!

Remember to use the `pwd` command if at any point you are not sure what your current directory is.

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

a. Using the full pathname from your home directory.

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

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

d. Change your working directory to `/etc`. The command `cat passwd` will now display the file.

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

Need Help?

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 displays the contents of a file on your terminal. The format is `cat` *`filename`*..

Solution

```
$ cat /etc/passwd
root:PrFX.Bc1hWLHc:0:1:system PRIVILEGED account:/:
nobody:*Nologin:65534:65534:anonymous NFS user:/:
...

$ cd /
$ cat etc/passwd
root:PrFX.Bc1hWLHc:0:1:system PRIVILEGED account:/:
nobody:*Nologin:65534:65534:anonymous NFS user:/:
...

$ cat passwd
cat: cannot open passwd

$ cd /etc
$ cat passwd
root:PrFX.Bc1hWLHc:0:1:system PRIVILEGED account:/:
nobody:*Nologin:65534:65534:anonymous NFS user:/:
...

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

Listing Files

Overview

The system keeps track of the names of all your files. You can see a list of the names of all files in a directory using the `ls` command.

The format of the `ls` command is:

`ls` *`options pathname`*

Using the ls Command

Here are some examples of the `ls` command assuming the working directory is `/usr/users/you`.

To display a list of files in the working directory:

```
$ ls
```

To display a list of files in the `/usr/users/you/book` using a full pathname:

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

To display a list of files in the directory `/usr/users/you/book` using a relative pathname:

```
$ ls book
```

To display a list of files in the working directory using the special symbol . (dot):

```
$ ls .
```

To display a list of files in the parent directory using the special symbol .. (dot dot):

```
$ ls ..
```

To display the files in more than one directory, separate each directory name with a space:

```
$ ls book project
book:
chap1     chap2     contents.lis
project:
plan      report.txt
```

Now Try This!

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 root 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.

Solution

1. **`$ ls`**
2. **`$ ls . ..`**
3. **`$ cd /`**

   `$ ls bin etc`
4. **`$ cd /usr/bin`**

   `$ ls`
5. If your home directory is, for example, `/usr/users/you`, you could use:

   `ls /usr /usr/users/you` **or** **`ls .. ../users/you`**
6. $ **`cd`**

Listing Details

There are many options to the `ls` command. Here are just a few.

`-a`	Lists **all** files in directory including those that begin with a period; file names that begin with a period are usually hidden
`-F`	Flags certain types of files with special symbols
`-l`	Provides a long listing of the files in a directory, including file type, access rights, owner, group, size, and date of creation or modification
`-R`	Recursively lists all subdirectories

You must have execute permission on the directory to get a long listing, while only read permission is required for an ordinary listing.

You can combine these options. Using the `-l` and `-a` opitons together provides a long listing of all files, including hidden files.

Example 6-1: Listing All Files

```
$ ls -al
total 8
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  5 09:38 book
-rwxrw-r--   1 you  users   88 Sep  3 10:30 letters
drwxr-xr-x   2 you  users  512 Nov  5 11:16 misc
-rw-r--r--   1 you  users  178 Nov  5 09:30 time.status
```

Now Try This!

1. Display a listing of all files in your home directory, the root directory, and the `/usr` directory.

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

2. Display a long listing of all the files in your home directory.

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

Solution

1. **`$ ls -a . / /usr`**
2. **`$ ls -al`**
3. **`$ ls -aF`**

   Directory names are marked with a slash (/).

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. The `whereis` and `find` commands are useful for these situations.

Locating Files

The `whereis` command locates the directory that contains the source, binary and reference page files for a program. With no options, `whereis` looks in the default directories for all three types of files. After looking in the default directory, `whereis` then searches the Path variable directories.

Example 6-2: Using the whereis Command

```
$ whereis emacs ❶
emacs: /usr/bin/emacs /usr/lib/emacs /usr/share/man/man1/emacs
$ 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 reference page directories

❷ Searches for `emacs` executable programs

❸ Searches for `emacs` reference pages

❹ Searches for `emacs` source files; none are found on this system

See the `whereis` reference page for more information and options.

Using the find Command

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

The format for the `find` command is:

`find` *`pathname expression`*

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

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

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

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

In this 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
```

Using find Command Expressions

Some of the more common `find` command expressions, examples of their use and their functions are listed here.

- The `-print` expression causes the file name(s) matching the criteria to be displayed on the screen.

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

- The `-name` expression tells the system to find files based on the file name.

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

- The `-ls` expression causes the file name(s) matching the criteria to be displayed on the screen in the format of a long list.

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

- The `-user` expression finds files based on the owner.

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

- The `-group` expression finds files based on the group.

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

- The `-size` expression finds files based on size in blocks. A block is 512 bytes. You can search for files greater than a size by preceding the size with a plus (+).

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

- The `-atime` expression finds files based on the number of days ago they were last accessed.

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

Pattern Matching

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 (*) (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`.

Now Try This!

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 include the characters `tt` and display those file names in the form of a long listing.

Solution

1. `$ find /etc -name motd -print`
2. `$ find /usr/users -user you -print`
3. `$ find /dev -name \*tt\* -ls`

   or

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

Viewing Files

Overview

UNIX provides many ways to display file contents on your terminal screen. Commands that provide this functionality are:

`cat`	Text of file scrolls by
`more`	File displays one page at a time
`head`	First 10 lines of file are displayed on the screen
`tail`	Last 10 lines of file are displayed on the screen

Viewing Entire Files

The `cat` command displays the entire contents of a file on your screen, with the first lines scrolling off if all contents do not fit.

The format for the `cat` command is:

`cat` *`filename`*

Example 6-3: 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
Kalamazoo MI (616)384-8310
Littleton MA (508)952-4839
Augusta ME (207)271-2211
Burlington VT (802)657-2895
Charleston SC (803)744-9046
Philadelphia PA (215)246-7028
Rochester NY (716)385-8240
$
```

Viewing Portions of Files

When you use the `cat` command to view the contents of a file, the entire file is sent to your screen, leaving the last screenful of the file on your screen. However, there may be times when you only want to see certain portions of a file.

The `head` and `tail` commands provide this functionality. By default, the `head` and `tail` commands display the first 10 or last 10 lines of a file.You can specify how many lines of the file you want to display.

This example displays the first six lines of the phones file.

Example 6-4: Using the head Command

```
$ 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
Kalamazoo MI (616)384-8310
$
```

This example displays the last four lines of the phones file.

Example 6-5: Using the tail Command

```
$ tail -4 phones
Burlington VT (802)657-2895
Charleston SC (803)744-9046
Philadelphia PA (215)246-7028
Rochester NY (716)385-8240
$
```

Viewing a Screenful of a File

The more command is also useful when viewing the contents of a file. This command displays one screenful at a time starting at the beginning of a file. After each screenful is displayed, the file name and the percentage of the file displayed appear at the bottom of your screen, and the system waits for your response.

Valid responses to the file name: (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 is not displayed on your terminal.
- Press spacebar to display the next screenful.
- Press B to display the previous screenful.
- Press Q to finish viewing the file and return to the system prompt.
- Press H to get help.
- Press / to get a prompt to search for a string.
- Press N to search for the next occurrence of the string.

This example displays the phones file one page at a time.

Example 6-6: Using the more Command

```
$ 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
Kalamazoo MI (616)384-8310
Littleton MA (508)952-4839
Augusta ME (207)271-2211
Burlington VT (802)657-2895
Charleston SC (803)744-9046
Philadelphia PA (215)246-7028
Rochester NY (716)385-8240
phones: END
```

Now Try This!

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.

Solution

1. `$ ls /etc`
2. `$ cat /etc/passwd`
3. `$ more /etc/passwd`
4. `$ head -6 /etc/passwd`

   `$ tail -9 /etc/passwd`

Creating Files

Overview

The UNIX operating system provides several ways to create files. One method is using an editor such as `vi` and `emacs`. Another method, used to create short files, is the `cat` command. You can also create empty files with the `touch` command.

Using the touch Command

The `touch` command can:

- Create an empty file
- Update the access time of a file
- Update the modification time of a file

The format for the `touch` command is:

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

The options are described in the following table.

Table 6-1: touch Options

Option	Function
`touch (no option)`	Updates both the access and modification times to the time specified, or the present time
`touch -a`	Changes only the access time
`touch -c`	Prevents the creation of the file
`touch -f`	Attempts to force file access in spite of file permissions
`touch -m`	Changes only the modification time
`touch -r`	Changes the modification time to that of the reference file
`touch -t`	Changes the modification time to the specified time in the form: [[CC]YY]MMDDhhmm[.SS]

The following examples demonstrate how to use the `touch` command.

```
$ touch poker.c
```

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

```
$ touch -m -c poker.c
```

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

```
$ touch -a -t 07070707 poker.c
```

Sets the access date of `poker.c` to July 7, 7:07 AM of the current year.

```
$ touch empty
```

Creates an empty file named `empty`.

Using the cat Command

In addition to using the `cat` command to view the contents of files, you can also use it to:

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

Using cat to Create a File

The format of the `cat` command to create a file is:

```
cat > filename
```

The > symbol indicates that the output of the command should be placed in the file following the > symbol.

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

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

❶ Redirect the output of the `cat` command to the new file `homephones`.

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

❸ On a blank line, press Ctrl/D to save the text and close the file.

You can now view the contents of the file at any time by entering the following command:

```
$ cat homephones
```

Now Try This!

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

Solution

Sample solution.

```
$ cat > homephones
Elizabeth  927-6016
John       838-1234
Lisa       884-5779
<Ctrl/D>
```

Using cat to Add Text to a File

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

This format of the `cat` command is:

```
cat >> filename
```

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

```
$ 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. Simple typing mistakes can be corrected using the erase key. However, you cannot move the cursor to a previous line to correct a typing error.

❸ On a blank line, press Ctrl/D to save the text into a file.

Now, when you view the contents of the `homephones` file, it contains the added information.

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

Using cat to Combine Files

File contents are appended to another file using the `cat` command, similar to how text lines are added to files.

This format of the `cat` command is:

```
cat filename1 >> filename2
```

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

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

Now Try This!

1. 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.
2. Append the fruits file to the vegetables file.

Solution

1. **$ cat > fruits**

```
apples
bananas
cantaloupes
dates
. . .
<Ctrl/D>

$ cat > vegetables
asparagus
brussel sprouts
broccoli
cauliflower
. . .
<Ctrl/D>
```

2. **$ cat fruits >> vegetables**

Creating Directories

Overview

After you use a UNIX operating system for awhile, your home directory may become cluttered. When you list your files, they fill the screen. The logical solution is to organize your files into directories.

Using the mkdir Command

The `mkdir` command creates a directory.

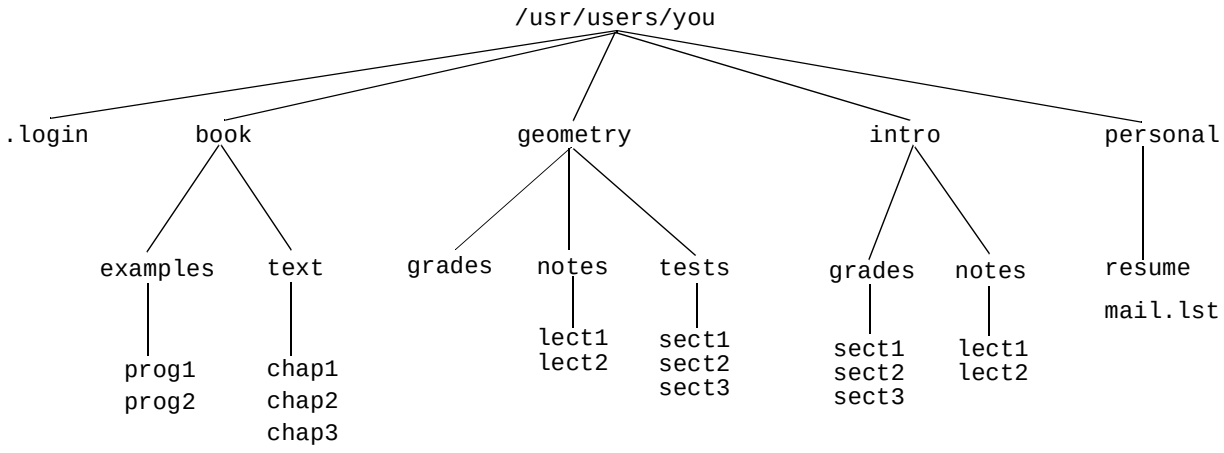
The format for the `mkdir` command is:

`mkdir` *`directory-name`*

When creating subdirectories, it is very important to know the name of your current working directory. Unless an alternate pathname is given for the new directory, it is automatically a subdirectory of your current working directory.

This figure represents the organization of files into four directories; book, geometry, intro, and personal.

Figure 6-2: Sample Directories



Assuming the working directory is `/usr/users/you`, the commands used to create the four directories are:

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

Using mkdir to Create Directories

Here are some examples of using `mkdir`.

This example sets the working directory to `/usr/users/you`, creates subdirectory `examples` in the `book` directory, and creates subdirectory `text` in the `book` directory.

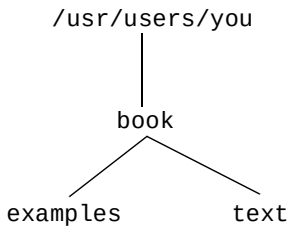
```
$ cd /usr/users/you
$ mkdir book/examples
$ mkdir book/text
```

This example changes working directory to `/usr/users/you/book`, creates subdirectory `examples` in the `book` directory, and creates subdirectory `text` in the `book` directory.

```
$ cd /usr/users/you/book
$ mkdir examples
$ mkdir text
```

The result of these two examples is the same, shown in this figure.

Figure 6-3: Results of mkdir Command (1)

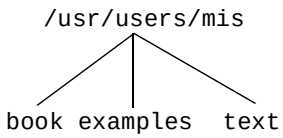


This example changes working directory to `/usr/users/you`, creates subdirectory `examples` in the `you` directory, and creates subdirectory `text` in the `you` directory.

```
$ cd /usr/users/you
$ mkdir examples
$ mkdir text
```

The result of the last example is shown in this figure.

Figure 6-4: Results of mkdir Command (2)



Now Try This!

1. In your home directory, create a directory called `exercises`.
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.

Solution

1. `$ mkdir exercises`
2. `$ ls -a exercises`
3. `$ mkdir exercises/sub1`
4. `$ cd exercises`

   `$ mkdir sub2`
5. `$ ls -a`

   ```
   .    ..   sub1  sub2
   ```
6. `$ cd`

   `$ pwd`

   ```
   /usr/users/you
   ```

Changing Permissions

Overview

File permissions allow you to restrict access to files or folders. Three groups of users can access files and folders: user, group, and other. File access is divided into three types of permissions: read, write, and execute.

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

Displaying the Permissions

You can display the permissions a file has by using the `-l` option with the `ls` command.

```
$ ls -l
-rwxr-x--x  1 you  users    45 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 are the group's permissions, and the last three are the others' permissions.

Within each set of three characters are the types of permissions. A dash (-) means no permission, and the following characters indicate a permission.

`r`	Users can read the file
`w`	Users can write to (change) the file; write access implies delete access as well
`x`	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 describes the file permissions for this sample entry.

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

Table 6-2: Interpreting File Permission Fields

Field	Name	Description
-	File type	The 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 (you) has read, write and execute access.
rw-	Group access	The next three characters indicate the file access for the group. Group (students) 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.

Changing File Permissions Using chmod

Use the chmod command to change file permissions. There are two different methods of changing file permissions.

- Using letters and operation symbols
- Using octal numbers

The chmod command can specify permissions using letters and operation symbols.

The format for the chmod command is:

chmod *class operation permission filename*

Table 6-3: chmod Options

Parameter	Description
class	Specifies which group's permission you are changing. Possible values are: u User's (owner's) permission g Group's permission o Other's or world permission a All (user, group and world)

Table 6-3: chmod Options (Continued)

Parameter	Description
operation	Specifies whether you are adding or removing a permission. Possible values are: + Add permission - Remove permission = Assign permission regardless of previous settings
permission	Specifies the type of permission you are adding or removing. Possible values are: r Read w Write x Execute
filename	Specifies the name of the file whose permissions you want to change.

This example shows how to use the `chmod` command with letters and operation symbols.

1. Display the current permissions on the file `a.out`.

```
$ ls -l a.out
-rw-r-----  1 you  users    23951 Jan 08 13:02 a.out
```

2. Give group and others write access to the file `a.out`. Separate multiple class-operation-permission arguments with a comma.

```
$ chmod g+w,o+w a.out
```

3. Display the permissions on the file `a.out`.

```
$ ls -l a.out
-rw-rw--w-  1 you  users    23951 Jan 08 13:02 a.out
```

4. Remove read access from the group and adds read access to others on the file `a.out`.

```
$ chmod g-r,o+r a.out
```

5. Display the permissions on the file `a.out`.

```
$ ls -l a.out
-rw--w-rw-  1 you  users    23951 Jan 08 13:02 a.out
```

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

```
$ chmod g+rx a.out
```

7. Display the permissions on the file `a.out`.

```
$ ls -l a.out
-rw-rwxrw-  1 you  users    23951 Jan 08 13:02 a.out
```

8. Give user, group, and other read and write only access to the file `a.out`.

```
$ chmod a=rw a.out
```

9. List the permissions on the file `a.out`.

```
$ ls -l a.out
-rw-rw-rw-  1 you  users    23951 Jan 08 13:02 a.out
```

Now Try This!

1. In your home directory, enter the command: `cp /etc/group 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.
   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.

Solution

```
$ cp /etc/group groupfile
$ ls -l groupfile
-rwxr-x---   1 you   users      268 Jan 09 08:43 groupfile
$ chmod g+w-x,o=x groupfile
$ ls -l groupfile
-rwxrw---x   1 you   users   268 Jan 09 08:43 groupfile
$ chmod uo-x,g-rw groupfile
$ ls -l groupfile
-rw-------   1 you   users   268 Jan 09 08:43 groupfile
```

Using chmod with Octal Numbers

The `chmod` command can also specify permissions using octal numbers.

This format for the `chmod` command is:

`chmod` *`octalnumber filename`*

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.

Table 6-4: Octal 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 and describes step-by-step how to use the `chmod` command with octal numbers.

1. Display the current permissions on the file `a.out`.

```
$ ls -l a.out
-rw-r-----  1 you  users    23951 Jan 08 13:02 a.out
```

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

```
$ chmod 662 a.out
```

3. Display the permissions on the file `a.out`.

```
$ ls -l a.out
-rw-rw--w-  1 you  users    23951 Jan 08 13:02 a.out
```

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

```
$ chmod 626 a.out
```

5. Display the permissions on the file `a.out`.

```
$ ls -l a.out
-rw--w-rw-  1 you  users    23951 Jan 08 13:02 a.out
```

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

```
$ chmod 777 a.out
```

7. Display the permissions on the file `a.out`.

```
$ ls -l a.out
-rwxrwxrwx  1 you  users    23951 Jan 08 13:02 a.out
```

8. Give the user read and write access, and the group and others no access to the file `a.out`.

```
$ chmod 600 a.out
```

9. Display the permissions on the file `a.out`.

```
$ ls -l a.out
-rw-------  1 you  users    23951 Jan 08 13:02 a.out
```

Changing File Group Using chgrp

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

The format for 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 above 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.

Now Try This!

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.

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.

Solution

```
$ chmod 755 groupfile
$ ls -l groupfile
-rwxr-xr-x   1 you   users   268 Jan 09 08:43 groupfile
$ chmod 111 groupfile
$ ls -l groupfile
 ---x--x--x   1 you   users   268 Jan 09 08:43 groupfile
 $ chmod 644 groupfile
$ ls -l groupfile
-rw-r--r--   1 you   users   268 Jan 09 08:43 groupfile
```

Moving and Renaming Files

Overview

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

The format of the `mv` command is:

`mv` *`file1 file2`* or `mv` *`file ... directory`*

Using the mv Command

When a file is moved 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 changed.

When the `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.

- Move the file `mail.lst` to the subdirectory `personal` under the current working directory

  ```
  $ mv mail.lst personal
  ```

- Move the files `res` **and** `skills` at the same time to the subdirectory `personal` under the working directory.

  ```
  $ mv res skills personal
  ```

- Rename the file `mail.lst` in the current working directory to `mail`.

  ```
  $ mv mail.lst mail
  ```

- Move the file `res` to the subdirectory `intro` under the working directory **and** renames it to `resume`.

  ```
  $ mv res intro/resume
  ```

- Move all the files in the directory `personal` to the directory `intro`, keeping the same file names. This command empties the `personal` directory.

  ```
  $ mv personal/* intro
  ```

Moving Files Safely

When you move a file into a file name that already exists, the original file is overwritten. 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 to have the system prompt you for confirmation any time a file might be overwritten.Type `y` to continue the copy or `n` to cancel the copy.

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

Now Try This!

1. In your home directory, put the list of files in your home directory into a file called `listing`. (Hint: use the > symbol to redirect the output of the `ls` command.)
2. Rename the file `listing` to `dir.list`.
3. In your home directory, create 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.

Solution

```
$ ls > listing
$ mv listing dir.list
$ mkdir temp
$ mv dir.list temp/listing
$ cd temp
$ ls
listing
$ cd
```

Copying Files

Overview

You may on occasion want to make a copy of a file rather than rename it. Always make a backup copy of a file before you make 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

- To create a copy of the `mail` file named `memo` in the working directory:

  ```
  $ cp mail memo
  ```

- To create a copy of the `resume` file named `skills` in the `pers` directory:

  ```
  $ cp resume pers/skills
  ```

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. The `-i` option asks you for confirmation before overwriting a file. The file is overwritten only if your response is `y` or `yes`.

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

Copying Subdirectories

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

The following example 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`.

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

Now Try This!

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` when prompted to overwrite the existing file.

Solution

```
$ cd
$ cp /etc/passwd pass1
$ mkdir practice
$ cp pass1 practice
$ cd  practice
$ ls
pass1
$ cp -i /etc/passwd pass1
overwrite 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...`*

- To remove the file named `book` from the working directory:

  ```
  $ rm book
  ```

- To remove the two files named `book` and `intro` from the working directory:

  ```
  $ rm book intro
  ```

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

Example 6-7: Removing Files with rm

```
$ 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
```

Removing Files Safely

The `-i` option 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? y
```

NOTE

Be very careful. Once a file is removed, you cannot retrieve it. Make a habit of using the `-i` option with the `rm` command.

Removing Files and Directories

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

The format for using 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 the files in the directory `/usr/users/you/test` and then the `test` directory as well, enter:

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

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

Example 6-8: Removing Files and Directories

```
$ 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

You can remove empty directories on the system that you no longer need using the `rmdir` command. It will warn you if the directory is not empty.

The format for the `rmdir` command is:

```
rmdir directory-name...
```

This example deletes a directory named `personal` from the current working directory if the `personal` directory is empty.

```
$ rmdir personal
```

This example deletes three directories named `proj1`, `proj2`, and `proj3` from the working directory, if they are empty.

```
$ rmdir proj1 proj2 proj3
```

Now Try This!

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 the `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.

Solution

```
$ mkdir practice1 practice2
$ cd practice1
$ cat > friends
Clark
Marion
Linda
Rob
<Ctrl/D>
$ cp friends friends1
$ cp friends friends2
$ cp friends friends3
$ rm friends2
$ rm -i friends3
rm: remove friends3? y
$ cd
$ rm -ri practice1
rm: remove practice1/friends? y
rm: remove practice1/friends1? y
rm: remove practice1? y
```

Linking Files

Overview

Links provide a mechanism for sharing files.

- A link allows more than one user to access the same data.
- A link is a directory entry referring to a file.
- A file may have multiple links to it.

About Inodes

An **inode** is a file system data structure that contains information about a file such as the:

- Type and access mode of a file
- File's owner
- Group access identifier
- Number of links to a file
- Size of the file in bytes
- Number of physical blocks used by the file
- Time the file was created, accessed, or modified
- Pointers to the data blocks on the disk

You can display the inode number with the `ls -i` command.

Using the ln Command

The two types of links, hard and symbolic (or soft link), are generated using the `ln` command. By default `ln` makes 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. Commonly used link options are:

`-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

About 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

Generating Hard Links

The following example shows how to generate a hard link to user Smif's mortgage program. The results of `ls -il` commands for both the source and target files are also shown.

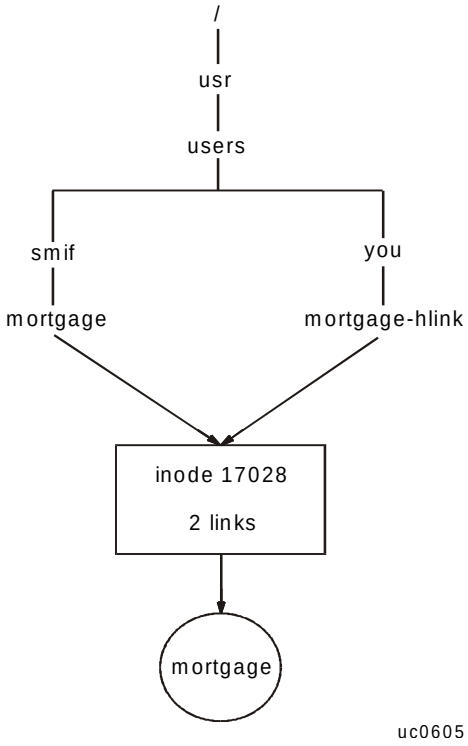
Example 6-9: Creating a Hard Link

```
$ pwd
$ /usr/users/you
$ ln /usr/users/smif/mortgage mortgage-hlink
$
$ ls -il mortgage-hlink
17028 -rwxr-xr-x   2 smif users    39772 Apr 06 10:40 mortgage-hlink
$
$ ls -il /usr/users/smif/mortgage
17028 -rwxr-xr-x   2 smif users    39772 Apr 06 10:40 mortgage
```

Relationship Between Hard Link and File

The following figure shows the relationship between a hard link and the file to which it is linked.

Figure 6-5: Hard Links



Now Try This!

1. 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? Compare the directory listings for the link and the original file. What directory entries are the same? Why?
6. Delete the file `dummy`.
7. Use `cat hard_link` or `more hard_link` to display the file. Can you display the file (after deleting the original file)? Why?
8. Delete the link file `hard_link`.
9. Try to create a hard link `hard_link` to `/etc/passwd`. What happens?
10. 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.

11. Get a directory listing of your working directory using `ls -l`. What is unusual about `hard_link`?

12. 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.

Solution

1. **`$ cp /etc/passwd dummy`**

2. **`$ ls -il dummy`**

```
80537 -rw-r--r--   1 you    users     2623 Dec 23 13:01 dummy
```

3. **`$ ln dummy hard_link`**

4. **`$ 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. The number of links increased from one to two. The inode numbers, link count, owner, group, size, and dates are all the same. The only difference is the file names themselves.

6. **`$ rm dummy`**

7. **`$ cat hard_link`**

```
root:PrFX.Bc1hWLHc:0:1:system PRIVILEGED account:/:
nobody:*Nologin:65534:65534:anonymous NFS user:/:
   .
   .
   .
```

8. **`$ rm hard_link`**

   When the file was deleted, the link count was not reduced to zero. Therefore, the file is still accessible.)

9. **`$ 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.

10. **`$ ln /usr/users/them/sample hard_link`**

11. **`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.

```
12. $ rm hard_link

rm: override protection 755 for hard_link? y
```

About Symbolic (Soft) Links

A **symbolic** or soft link provides a path to a file.

This format of the `ln` command is:

`ln -s` *`source_file target_file`*

Symbolic links can be made:

- Across file systems
- To a directory

The original file and soft link have different inode numbers.

Creation of a symbolic link has no effect on the link count. Delete the original file and the soft link still exists, but the original file is gone.

In the following example, a symbolic link is made to `/usr/users/smif/mortgage`. Directory listings for the link and linked files are shown.

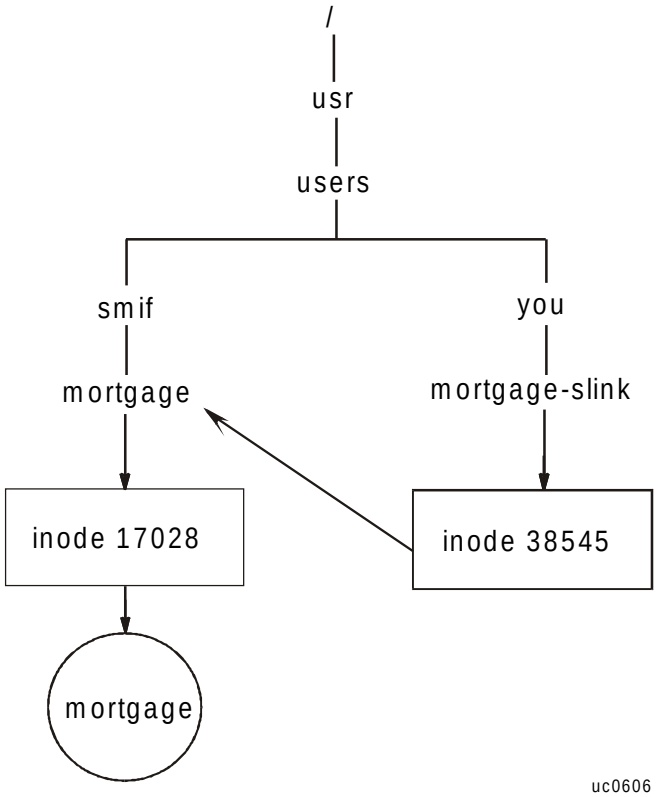
Example 6-10: Creating a Soft Link

```
$ 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 Link and File

The following figure shows the relationship between a soft link and the file to which it is linked.

Figure 6-6: Symbolic Links



Now Try This!

1. Copy the file `/etc/passwd` to a file named `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 indicate 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`.

Solution

1. **$ cp /etc/passwd dummy**

```
$ ls -il dummy

 80551 -rw-r--r--  1 you   users   2623 Dec 23 13:19 dummy
```

2. **$ ln -s dummy soft_link**

3. **$ 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. 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. **$ rm dummy**

6. **$ cat soft_link**

```
cat: cannot open soft_link
```

7. **$ rm soft_link**

8. **$ ln /etc/passwd soft_link**

9. **$ 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. **$ rm soft_link**

Summary

Changing Directories

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 use the `cd` command, you must first understand the concepts of home and working directories and pathnames.

The home directory is usually identified by a pathname that includes the user name as its last component. Your home directory never changes.

Your working directory is defined as your current location in the directory tree.

Listing Files

The system keeps track of the names of all your files. You can see a list of the names of all files in a directory using the `ls` command.

Finding Files

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. The `whereis` and `find` commands are useful for these situations.

Viewing Files

UNIX provides many ways to display file contents on your terminal screen. Commands that provide this functionality are:

`cat`	Text of file scrolls by
`more`	File displays one page at a time
`head`	First 10 lines of file are displayed on the screen
`tail`	Last 10 lines of file are displayed on the screen

Creating Files

Use the `cat` command to create a file by redirecting the output with the > or >> symbols.

The `touch` command creates an empty file.

Creating Directories

After you use a UNIX operating system for awhile, your home directory may become cluttered. When you list your files, they fill the screen. The logical solution is to organize your files into directories.

The command for creating a directory is: `mkdir` *`filename`*.

Changing Permissions

File permissions allow you to restrict access to files or folders. Three groups of users can access files and folders: user, group, and other. File access is divided into three types of permissions: read, write, and execute.

Use the `chmod` command to change file permissions.

Moving and Renaming Files

Use the `mv` command to move or rename a file. If the destination name is a directory, the file is moved; otherwise the file is renamed.

Copying Files

You may on occasion want to make a copy of a file rather than rename it. Always make a backup copy of a file before you make any major revisions to it.

The `cp` (copy) command performs this function.

Removing Files and Directories (CLI)

Use the `rm` command to remove files. Use the `rm -r` command to remove a directory and all its contents. Use the `rmdir` command to remove an empty directory.

Linking Files

Use the `ln` command to create a hard link:

- Indistinguishable from the original entry
- Cannot span file systems
- Cannot refer to directories

Use the `ln -s` command to create a symbolic (soft) link:

- Can be made to a directory
- Can be made across file systems
- Delete a file which has an existing symbolic link, and there is no way to access the file through the link

7

Introducing UNIX Shells

Unit Overview

Introduction

This unit discusses the typical shells available on a UNIX system, as well as the basic features of the different shells on a Tru64 UNIX system. This unit also covers how to:

- Change shells
- Determine which shell is your default login shell
- Customize your Korn shell environment

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 Tru64 UNIX
- Determine your default login shell and change your shell
- Use wildcard operators to specify lists of file names
- Define the use of shell user startup files
- Describe typical contents of the Korn shell user startup files

Resources

For more information on the topics in this unit, see the following:

- Tru64 UNIX reference pages for `ksh`
- *Common Desktop Environment: User's Guide*
- *The Korn shell Command and Programming Language* — Chapters 2, 6 and 13
  Morris I. Bolsky and David G. Korn
  Prentice Hall, Englewood Cliffs, New Jersey 07632 (c) 1989
  ISBN 0-13-516972-0
- *Korn shell Programming Tutorial* — Chapters 11 and 14
  Barry Rosenberg
  Addison-Wesley Publishing Company, Inc. Reading, MA 01867 (c) 1991
  ISBN 0-201-56324-X

Introducing the UNIX Shell

Overview

This lesson provides an introduction to the UNIX shells, describing the purpose of shells, which shells are available on a Tru64 UNIX system, the general features of the shells, and how to change from one shell to another.

NOTE

Rather than providing examples of the command syntax for all UNIX shells, this unit focuses on the Korn shell. The syntax may be different for other shells.

Purpose of Shells

A UNIX **shell** 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
- Means to execute a shell script

Available Shells in Tru64 UNIX

The Tru64 UNIX operating system provides the following shells:

- C shell
- Bourne shell (including the restricted Bourne shell)
- Korn shell

The default prompts for the Tru64 UNIX shells are:

```
%    C shell
$    Bourne shell
$    Korn shell
```

Tru64 UNIX Shell Features

The following table summarizes the features of each Tru64 UNIX shell.

Table 7-1: Tru64 UNIX 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 shell	X			Offers a controlled shell environment to limit user access
Common 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 arithmatic		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 Your 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 the following command:

```
more /etc/passwd
```

Look for your account name. The entry after the last colon (`:`) is the pathname of the shell to execute. The pathnames for the three available shells are as follows:

`/bin/csh`	C shell
`/bin/sh`	Bourne shell
`/bin/ksh`	Korn shell

An example `/etc/passwd` entry is shown in the following example. In the example, the pathname of the default login shell is `/bin/ksh`, the Korn shell.

Example 7-1: /etc/passwd Entry

```
you:Gux27ZVJ.QtV6:1745:15:User for U&C:/usr/users/you:/bin/ksh
```

Changing Your Login Shell

Use the `chsh` (change shell) command to change your login shell. For example, to change your login shell from the Korn shell to the C shell, use the command shown in the example.

Example 7-2: Changing Your Login Shell with chsh

```
$ chsh
Changing login shell for joe
Old shell: /bin/ksh
New shell: /bin/csh
$
```

Your /`etc/passwd` entry is changed, and the next time you log in, your default shell will be the C shell.

Temporarily Changing Your Login Shell

To temporarily change your shell (start a subshell), enter the pathname of the requested shell. For example, to temporarily change shells from the Korn shell to the C shell, enter the command shown in the example.

Example 7-3: Changing Shells Temporarily

```
$ /bin/csh
%
```

To return to the previous shell, either:

- Enter `exit`

  or

- Press Ctrl/D (the end of file character)

Now Try This!

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 in 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.

5. Display `/etc/passwd` and verify that your login shell has changed to the selected shell.

6. If your login shell is not now the Korn shell, use `chsh` to change your login shell to the Korn shell.

7. To use the **new** login shell you just changed to, you must log out, then log back in again.

Solution

1.

```
$ more /etc/passwd
root:PrFX.Bc1h:0:1:system PRIVILEGED account:/:
nobody:*Nologin:65534:65534:anonymous NFS user:/:
you:kdtLvA2:1745:15:User:/usr/users/you:/bin/ksh
```

2.

```
$ cat /etc/shells
/bin/sh   /bin/csh   /bin/ksh
```

3. For example, if your present shell is the Korn shell, to change to the C shell, you would enter:

```
$ /bin/csh
%
% exit
$
```

4. 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
$
```

5.

```
$ more /etc/passwd
root:PrFX.Bc1hWLHc:0:1:system PRIVILEGED account:/:
nobody:*Nologin:65534:65534:anonymous NFS user:/:
you:kdtLvA2:1745:15:User:/usr/users/you:/bin/csh
```

6. This sample assumes that your login shell is now the C shell.

```
% chsh
ld shell: /bin/csh
New shell: /bin/ksh
%
```

7. No solution required.

Using 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 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.

Wildcard operators cannot be used to match the leading period in hidden file names.

Using the * Operator

Use the asterisk (*) operator 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

Using the ? Operator

Use the question mark (?) operator 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 three 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 (`'`)

For example, you can delete a file named `*wow*` with the command `rm \*wow\*` or the command `rm '*wow*'` .

Now Try This!

1. Use the `touch` command to create three files, `a`, b, and `c`. (If you already have files with these names, the command 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 only your hidden files.

Solution

1.

```
$ 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.

```
$ 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
$
```

Describing the Korn Shell Environment

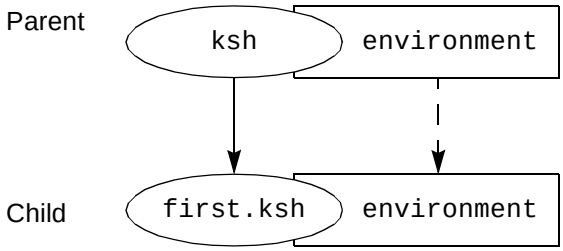
Overview

This lesson provides an introduction to the concept of the parent and child process and the environment.

The Parent, the Child, and the Environment

Assume that you are logged in to a UNIX system and are using the Korn shell to run 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. 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.

Figure 7-1: Processes and the Environment



Every parent or child process has an environment. An **environment** is a collection of privileges, resources, and traits.

- A parent cannot inherit environmental traits from its child.
- A child cannot damage the environment of its parent.
- By default, a parent process passes rights and privileges to its children.

Inherited Environmental Characteristics

By default, a parent does not pass variables, functions, or aliases to its child. The following environmental characteristics are passed from a parent to a child:

- 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

- Value of the `IFS` variable

A child does not inherit the following environmental characteristics from its parent:

- Aliases defined by the parent
- Functions defined by the parent
- Variables defined by the parent
- Values of the parent's Korn shell reserved variables
- Values of the parent's option settings

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

The Korn shell has many reserved variables, which fall into two classes; variables whose values the Korn shell:

- Sets and updates
- Does **not** automatically set

The following table lists some Korn shell variables that are set and updated by the Korn shell.

Table 7-2: Variables Set and Updated by the Korn Shell

Variable	Function
`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, and a nonzero error status indicates failure

The next table lists some Korn shell variables **not** set by the Korn shell.

Table 7-3: 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 / (absolute 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 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; 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 (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 you are using
VISUAL	Another variable that defines the command line editor

Setting Korn Shell Variables

Use an assignment statement to set variables. The format of an assignment statement is:

name=*value*

If *value* has a space in it, you must enclose the entire string with double quotes.

As shown in the following example, you can create the variables myname and address and assign a value to the EDITOR and FCEDIT variables.

Example 7-4: Setting Korn Shell Variables

```
$ myname="Charlie Brown"
$ address="2 Tired Way"
$ EDITOR=emacs
$ FCEDIT=emacs
```

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
$
```

Displaying 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 following example.

Example 7-5: Displaying All Variables with 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'
```

You can also display an individual variable with the `print` command, which has the format: `print $name`.

Example 7-6: Displaying Individual Variables with set

```
$ 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
```

Using typeset with Korn Shell Variables

The Korn shell supports some data types, and you may want to declare the data types of the variables you use for Korn shell programming.

Use a Korn shell built-in `typeset` to:

- Set variable attributes
- Unset variable attributes
- Set values

- Display variables and their attributes. Do not specify a variable name, and specify:

- (minus)	Displays names and values of all variables that have the attributes you specify
+ (plus)	Displays only the names
Nothing (neither plus nor minus)	Displays names and attributes of all variables

Some of the attributes shown in the following table can be specified for `typeset`.

Table 7-4: 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-justified; *n* specifies field length
`-LZ`*n*	Left-justified and stripped of leading zeros; *n* specifies field width
`-R`*n*	Right-justified; n specifies field length
`-RZ`*n*	Right-justified and stripped of leading zeros; *n* specifies field width

The following example shows how to use the `typeset` command as another way to display variables.

Example 7-7: Displaying Variables with 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 PTARG
export USER
function RANDOM
export SHELL
integer TMOUT
```

```
export SAVEPID
export HOME
export FCEDIT
export TERM
export PWD
export TZ
export ENV
integer MAILCHECK
$
```

The following example shows how to use the `typeset` command to define variables.

Example 7-8: typeset Command Usage

```
$ 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 Define the variable `uper` and set the attributes to uppercase. Assign a value to it.

❷ Define the variable `lower` and set the attributes to lowercase. Assign a value to it.

❸ Display the variables `uper` and `lower`, with some spaces between them. Note that `uper` is forced to uppercase and `lower` is forced to lowercase.

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. To 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`.

More `typeset` command examples are shown here.

Example 7-9: More typeset Command Usage

```
$ typeset -L9 first=Ambrosia  ❶
$ typeset -L3 mi="B."    ❷
$ typeset -L10 last="Salad"  ❸
$ print "$first$mi$last" ❹
Ambrosia B. Salad
$ first="Leaf-lettuce"   ❺
$ print "$first$mi$last" ❻
Leaf-lettB. Salad
$ typeset -lur   ❼
konstant=96420
lower='lowercase this'
uper="THIS IS AN UPPERCASE TEST"
$
```

❶ Define the variable `first`, and set the attributes to left-justified with a field width of 9. Assign a value to it.

❷ Define the variable `mi` and set the attributes to left-justified with a field width of 3. Assign a value to it.

❸ Define the variable `last` and set the attributes to left-justified with a field width of 10. Assign a value to it.

❹ 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.

❺ Assign a value to `first`, which contains more characters than the attributes will allow.

❻ Display the values of `first`, `mi`, and `last`. Notice that `Leaf-lettuce` is truncated to nine characters.

❼ Display all variables that have the attributes of `lowercase`, `uppercase`, and `read only`.

Korn Shell Options

Several options affect how the Korn shell processes commands. The Korn shell turns on some options and allows you to specify others. The Korn shell assigns defaults for the options, but you can change most of them.

Use the `set` command to turn on, turn off, or display Korn shell options. The formats for the `set` command are shown here.

Example 7-10: set Command Formats

```
set -o          # Display all options
set -o {option} # Turn on an option
set +o {option} # Turn off an option
```

The following example shows how to turn off the `ignoreeof` option and turn on `gmacs`.

Example 7-11: Setting Options with the set Command

```
$ set +o ignoreeof
$ set -o gmacs
```

The next example shows two displays from the `set -o` command placed side by side. The one on the left is the output prior to the modifications in the previous example. 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.

Example 7-12: Displaying Options with the set Command

```
(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
```

Now Try This!

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 `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.

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 `print` command.

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 the `unset` command 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.

Solution

1.

```
$ set
ERRNO=0
FCEDIT=/usr/bin/ed
HOME=/usr/users/you
. . .
$ typeset
export PATH
function ERRNO
. . .                  .                    .
```

2.

```
$ print $RANDOM
14768
$ print $RANDOM
22034
$ print $RANDOM
13180
$ print $RANDOM
1729
$
```

3. No solution required.

4.

```
$ 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.

```
$ unset myname address age
$ print $myname $address $age
$
```

6.

```
$ set -o
Current option settings
allexport        off
. . .
```

7.

```
$ set -o allexport
$ set -o
Current option settings
allexport        on
. . .
$ set +o allexport
$ set -o
Current option settings
allexport        off
. . .
```

8.

```
$ set -o
Current option settings
allexport        off
bgnice           on
emacs            off
gmacs            off
vi               off
$ set -o emacs
$ set -o
Current option settings
allexport        off
bgnice           on
emacs            on
gmacs            off
vi               off
$ set -o vi
$ set -o
Current option settings
allexport        off
bgnice           on
emacs            off
gmacs            off
vi               on
```

Defining 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 lesson provides examples of the Korn shell startup files and describes some of the variables and options that can be set.

The Korn shell startup files are:

- `/etc/profile`
- `$HOME/.profile`
- The file indicated by the expansion of the `ENV` variable (we will use `$HOME/.kshrc`)

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 needs.

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.

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.

The assignment of the file name to the `ENV` variable is done in the `$HOME/.profile` file, to be described shortly.

Korn Shell .profile File

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
- Set options that you want to apply to your login shell only
- Specify a script file to execute when you log out

A sample Korn shell `.profile` file is shown in the following example.

Example 7-13: Korn Shell .profile File

```
CDPATH=.:..:$HOME
ENV=$HOME/.kshrc
EDITOR=emacs
FCEDIT=emacs
PS1='! ksh> '
export PATH ENV PS1
trap $HOME/.kshlogout  EXIT
```

Korn Shell Environment Script File

Each time the Korn shell is invoked, it expands the `ENV` variable, and if not null, executes that script file. We will use `$HOME/.kshrc` as defined in the `.profile` of the preceding example.

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 commands that might be included in a Korn shell environment file are shown in the following table.

Table 7-5: Some .kshrc Commands

Command	Function
`set -o monitor`	Enables the Korn shell to display the job number when a background job is started, and displays a status message when the job completes
`set -o trackall`	Reduces 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`	Prevents the Korn shell from writing over an existing file when using the > (redirection) operator
`alias lal='ls -al`	Defines an alias for getting a long list of all files
`alias h=history`	Defines an alias that provides a history of commands
`alias c=clear`	Defines an alias that clears the screen
`HISTSIZE=64`	Defines the maximum number of previously entered commands that you can access through the Korn shell as 64; default is 128
`umask 022`	Creates 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

Summary

Introducing the UNIX Shell

The Tru64 UNIX available shells, their default prompts, and file names are:

`/bin/csh (%)`	C shell
`/bin/sh ($)`	Bourne shell
`/bin/ksh ($)`	Korn shell

Shells can be used as a programming language to provide:

- A user interface to the Tru64 UNIX operating system
- A command language interpreter

Using Wildcard Operators

- The asterisk (`*`) operator can match any string of zero or more characters.
- The question mark (`?`) operator can match any single character.
- The special meanings of wildcard operators can be escaped (turned off) by preceding the wildcard operator with a backslash (\), or by enclosing the string in single quotes.

Describing 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
- Does not automatically set

Korn shell variables are defined with an assignment statement that has the format `name=value`. Variables are displayed with the `set`, `typeset`, or `print $name` commands. The `typeset` command sets the attributes of a variable.

Defining Korn Shell Startup Files

The system and user login scripts 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 set up shell and environment variables and options:

- `/etc/profile`
- `$HOME/.profile`
- Environment file designated by the expansion of the `ENV` variable

8

Using the vi Editor

Unit Overview

Introduction

This unit introduces the various UNIX editors and provides instruction for the `vi` editor. One major advantage of the `vi` editor over the `emacs` editor is that most UNIX systems offer the `vi` editor. If you use `vi`, and move from one UNIX system to another, you might not have to learn any other editor.

This unit shows you how to start and exit `vi`, move the cursor, find and edit text, and copy and move text.

Objectives

To edit files on a UNIX system, you should be able to:

- Compare the various UNIX editors
- Start and exit an edit session
- Move around in a file
- Find and change text
- Copy and move text

Resources

For more information on the topics in this unit, see the following:

- *Command and Shell User's Guide*, Appendix A
  `http://www.unix.digital.com/faqs/publications/base_doc/DOCUMENTATION/V40D_HTML/APS2HDTE/TITLE.HTM`
- `vi` reference page

Introducing the UNIX Editors

Overview

The table shows the standard editors available on most UNIX systems.

Table 8-1: Standard UNIX Editors

Editor	Description
`ed`	Line editor that operates on one line at a time; `ed` is the only editor available in single user mode
`ex`	Line editor similar to `ed`, but more powerful; provides a subset of the `vi` screen editor commands (`vi` editor can be invoked from `ex`)
`edit`	Line editor designed for beginning users; a simplified version of `ex`
`sed`	Stream line editor that modifies lines based on script commands from the command line or from a script file
`vi`	Full screen display editor based on the `ex` line editor; uses different operational modes and options that can be customized
`view`	Read-only version of `vi`
`emacs`	Powerful, extensible, screen editor which allows editing in multiple windows and has an extensive help facility; GNU emacs editor is available from the Free Software Foundation and the Tru64 UNIX Freeware CD-ROM

The Common Desktop Environment (CDE) also includes a basic text editor with a graphical user interface. See the Text Editor icon on the Front Panel.

Editing with vi

Overview

This lesson discusses the `vi` modes of operation as well as how to start and end an editing session.

Setting Up Your Terminal

If you are using a Tru64 UNIX system from a remote host, set up your workstation window or terminal for VT100 mode. Otherwise, when you press the Escape (Esc) key to return to command mode, the case of the last letter typed may be changed (uppercase to lowercase or vice versa).

For example, on a DECterm window, choose General... from the Options Menu, and choose VT100 mode.

Some keyboards do not contain a labeled Esc (Escape) key. Alternate keystrokes that may work on your system include:

- Press the F11 key

  In a DECterm window, pull down the Options menu and choose Keyboard. Then choose "F11 key sends Escape".

- Press Ctrl/[

Editing Modes

The `vi` editor has three operational modes:

- Command mode

  The default mode entered upon editing a file with `vi`. Used to move the cursor, and delete or change text.

  Entered from text input mode by pressing Esc. Entered from last line mode by pressing Return. Pressing Esc in command mode cancels a partial subcommand. Pressing the interrupt key (usually Ctrl/C) will also cancel a partial subcommand.

- Text input mode

  Entered from command mode by typing the letters a, A, c, C, i, I, o, O, R, s, or S.

  After entering one of these commands, you can enter text into the buffer at the current cursor position.

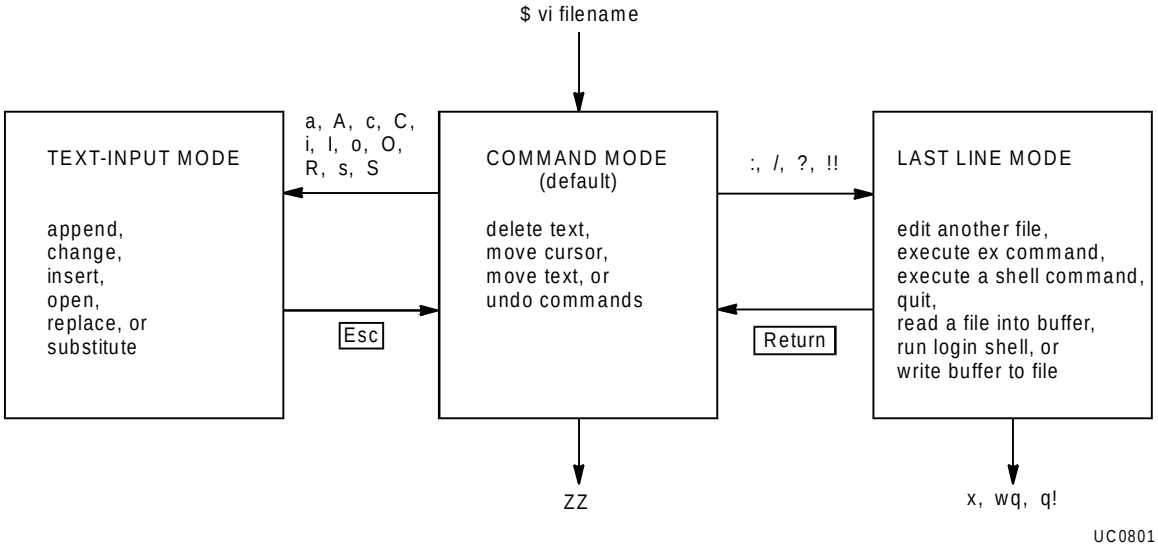
- Last line mode

  Entered from command mode by typing colon (:), slash (/), question mark (?), or exclamation marks (!!).

  A prompt appears on a line at the bottom of the screen. Type a command and press the Return key.

The relationship between the three operational modes is shown in greater detail in this figure.

Figure 8-1: vi Modes



If you are not sure what mode you are in, press the Esc key twice and you will hear a beep, which means you have returned to command mode.

Starting vi

You can start the `vi` editor using the name of an existing file to edit or a new file you want to create, or without a file name.

The format of the vi command is:

`vi [`*`options`*`] [`*`filename`*`]`

If you start `vi` with the `-R` option, the file is opened in read-only mode. You can use the `view` command to accomplish the same thing.

If the file name is an existing file, the first lines of the file are displayed on the screen. If the number of lines is not enough to fill the screen, all lines that do not contain characters have a tilde (~) in the first column. The last line shows the name of the file and number of lines and characters in the file.

Editing an Existing File

This example shows how to edit the `vegetables` file in the current directory.

Example 8-1: Starting vi with 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.

❸ The last line indicates the name of the file and number of lines and characters in the file.

Creating 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 8-2: Starting vi for a New File

```
$ vi junk
~
~
~
...
~
"junk" [New file]
```

Exiting vi

The following table lists the commands used to quit `vi`, execute a command, or run a shell command from within `vi`.

Table 8-2: Interrupting, Cancelling and Exiting vi

To	Use
Exit `vi`, saving changes, if any.	`:wq or 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 returns you to `vi`.	`:sh`
Suspend `vi`	`Ctrl/Z`

Note that several of these commands start with a colon, which invokes last line mode.

Now Try This!

1. Create a sample file with the following command:

   ```
   man vi > myfile.txt
   ```

2. Start the `vi` editor with `myfile.txt`.
3. Quit `vi` with the `:q!` command.

Solution

1. **`$ man vi > myfile.txt`**

2. **`$ vi myfile.txt`**

```
vi(1)           vi(1)


NAME
...
```

3. **`:q!`**

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 use 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.

This section introduces a few of the many cursor movement commands in `vi`.

Moving the Cursor a Character or Line

There are a number of commands to move the cursor near the line you are currently editing.

Table 8-3: Moving the Cursor a Character or a Line

To 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 that the `vi` editor is case sensitive and the use of uppercase and lowercase for the same letter often causes different results.

Repeating Cursor Movement

You can prefix most `vi` commands with the number of times you want the command to execute.

For example, press the `k` key to move the cursor up one line. Type `6k` to move the cursor up 6 lines.

This works only if there are six existing lines above the present cursor position.

Moving the Cursor Within a Screen

There are a number of commands to navigate within a screen.

Table 8-4: Moving the Cursor Within a Screen

To Move the Cursor	Use
To the beginning of the current line	`0`
To the end of the current line	`$`
To the beginning of the next word	`w`

Table 8-4: Moving the Cursor Within a Screen (Continued)

To Move the Cursor	Use
To the beginning of the previous word	`b`
To the beginning of the next sentence	`)`
To the beginning of the previous sentence	`(`

Paging

There are a number of commands to page in the file you are editing.

Table 8-5: Cursor Movement: Paging and Scrolling

To Move the Cursor	Use
Scroll up	`Ctrl/U`
Scroll down	`Ctrl/D`
Page forward one screen	`Ctrl/F`
Page backward one screen	`Ctrl/B`
Place the cursor at the first character in the file	`H`
Place the cursor at line number *n* or at the last line	*n*`G`

The amount that the screen scrolls depends upon:

- The `stty rows` value (the number of lines in the terminal display)
- The `vi window` value (the number of lines in one window of text)

Now Try This!

1. Edit the `myfile.txt` file and enter the following cursor movement commands: `3jl`

   You should be over the letter **i** in `vi`.
2. Enter the following cursor movement commands: `155G`

   You should be on the first character in line 155 ([).
3. Enter the following cursor movement commands: `)w`

   You should be on the second word in the next sentence.
4. You can practice with other movement commands.
5. Quit `vi` with the `:q!` command.

Hint

If the cursor movement commands appear on the screen, press the Esc key to return to command mode.

Finding, Inserting, and Changing Text

Overview

This section discusses how to search for, insert and change text in a file.

Searching for Patterns

You can search for patterns within a file.

Table 8-6: 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 in the same direction	`n`
Repeat the last search but in the opposite direction	`N`

When you type the slash (/), it is displayed at the bottom of the screen.

Now Try This!

1. Edit the `myfile.txt` file and use this command to search for the word mode: `/mode`

   It matches the pattern in the word modes in line 62.
2. Use the `n` command to find the next occurrence.
3. Use the `N` command to find the previous occurrence.
4. Quit `vi` with the `:q!` command.

Hint

As soon as you type the slash (/), it appears on the bottom line of the screen. Press Return after typing the pattern.

If the commands appear on the screen, other than the bottom line, press the Esc key to return to command mode.

Inserting Text

The following commands place the `vi` editor in text input mode so you can add text. All keys typed will then be entered as normal text until you press the Esc key.

Table 8-7: 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`
Open an empty line for text insertion below the current line	`o`
Open an empty line for text insertion above the current line	`O`

Now Try This!

1. Edit the `myfile.txt` file and use the `i` command to insert the following line:

   ```
   The quick brown fox jumped over the fence.
   ```

2. Press the Esc key to return to command mode. Move to the **`i`** of `vi` and use the `a` command to append a space and the word command.

3. Press the Esc key to return to command mode. Quit `vi` with the `:wq` command.

Hint

If the cursor movement commands appear on the screen, press the Esc key to return to command mode.

If the Esc key does not work, try typing Ctrl/[.

Solution

1. `The quick brown fox jumped over the fence.`

2. `vi command(1)  vi(1)`

   `NAME`

   `...`

3. `:wq`

Changing Text

Use these commands to change text while in command mode.

Table 8-8: Changing Text

To	Use
Change the current word	`cw`
Change the previous word	`cb`
Change to the end of the line	`c$ or C`
Change to the beginning of the line	`c0`
Change the entire line	`cc`
Delete text from the present cursor position to the end of the word (and following space)	`dw`
Delete to the end of the line	`d$ or D`
Delete to the beginning of the line	`d0`
Delete the current line	`dd`
Replace the present character with the specified character	`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`
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	`~`

When you use the change command (`c`), a dollar sign ($) is displayed at the end of the area being changed.

Now Try This!

1. Edit the `myfile.txt` file and find the word `quick`.
2. Use the `cw` command to change the word.
3. Type the word fast-moving. Press the Esc key to return to command mode.
4. Move down one line and delete the line with the `dd` command.
5. Move to the line beginning with the words `vi command` and use the `S` command to substitute the following line:

   ```
   This is an explanation of the vi command.
   ```

6. Press the Esc key to return to command mode. Quit `vi` with the `:wq` command.

Hint

In command mode, use */pattern* to find the pattern.

If the cursor movement commands appear on the screen, press the Esc key to return to command mode.

Solution

```
The fast-moving brown fox jumped over the fence.

This is an explanation of the vi command.

NAME
...
:wq
```

Search and Replace

You can use the `ex` editor substitute command in `vi` to perform a global search and replace, with this command:

`:%s/`*search_pattern*`/`*replace_pattern*`/g`

You can add a `c` after the `g` to prompt for confirmation before each substitution.

Now Try This!

1. Edit the `myfile.txt` file and use the following command:

   ```
   :%s/editor/bathtub/gc
   ```

   When you press Return, it displays the first occurrence of the word `editor`.

   ```
   The vi command is a display editor that is based
                                  ^^^^^^
   ```

2. Type `y` and press Return to substitute. It displays the next occurrence.
3. Type `n` and press Return to not substitute.
4. Quit `vi` with the `:wq` command.

Hint

The affirmative and negative responses are determined by your locale. For the default or English locale, use **y** or yes and **n** or no.

If things are not working, press the Esc key to return to command mode and start the command again.

Solution

```
The vi command is a display bathtub that is based
on an underlying line editor (ex).
```

Copying, Moving, and Saving Text

Overview

This lesson discusses how to use buffers with the `yank` or `delete` and `put` commands to copy or move text. It also covers commands to save text by writing the buffer to a file.

Copying and Moving Text

When you delete or copy text in the `vi` editor, it is put in a temporary buffer. You can then move to another location and put the text back in.

For example, to move the current line down two lines, first delete it with the `dd` command. Move down one line with the `j` command and paste the line below the cursor with the `p` command.

delete (`d`)	Cuts text to the buffer; delete word (`dw`), delete line (`dd`)
yank (`y`)	Copies text to the buffer; yank word (`yw`), yank line (`yy`) You can prefix the command with a number; `5yy` copies 5 lines.
put (`p`)	Pastes text from the buffer **after** the cursor
put (`P`)	Pastes text from the buffer **before** the cursor

Now Try This!

1. Edit the `myfile.txt` file and use the `yy` command to copy the first line.
2. Move down to line 5 and use the `p` command to paste the line.
3. Quit `vi` with the `:wq` command.

Hint

One way to move to line 5 is typing `5G`.

If the commands appear on the screen, press the Esc key to return to command mode.

Solution

```
The fast-moving brown fox jumped over the fence.

This is an explanation of the vi command.


The fast-moving brown fox jumped over the fence.
...
:wq
```

Restoring and Repeating Changes

You can undo or repeat a command.

- To undo the last command, type `u`. The cursor need not be on the line where the original edit was made.
- To undo all edits on a single line, type `U` if the cursor is still on that line.
- To repeat the last command, type a period (`.`).

Saving Changes

Use the `:w` command to save changes to the original file. The colon brings you to last line mode where the commands are executed.

`w` *`file`*	Saves the contents to the specified file
`w!`*`file`*	Overwrites the specified file

Now Try This!

1. Edit the `myfile.txt` file and use the `dd` command to delete the first line.
2. Use this command to write to a new file: `:w myfile2.txt`
3. Quit `vi` with the `:q` command.
4. Use this command to compare the first three lines of the two files: `head -n 3 myfile.txt myfile2.txt`

Hint

The `head` command displays the first few lines of a file.

Solution

```
$ head -n 3 myfile.txt myfile2.txt
==> myfile.txt <==
The fast-moving brown fox jumped over the fence.

This is an explanation of the vi command.

==> myfile2.txt <==

This is an explanation of the vi command.
```

Editing Another File

Use the commands shown here to edit another file.

`e` *`file`*	Edits a different file; note the space before the file name
`e!`	Reedits the current file, discarding any changes
`r` *`file`*	Reads the specified file into the editing buffer below the cursor; note the space before the file name
`r!`	Runs a shell command, placing the command output in the file below the cursor position

Controlling the Screen

Use the `Ctrl/L` command to clear and redraw the screen.

`z`	Redraws the screen with the current line at the top
`z-`	Redraws the screen with the current line at the bottom
`z.`	Redraws the screen with the current line at the center

Recovering an Edit Session

If a system failure occurs while you are editing a file, any changes 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 any recover buffers are available, execute the `vi` command with the `-r` option. It displays a list of all your editing buffers that the system has saved.

```
$ vi -r

On Mon Mar 16 at 9:05 saved 5 lines of file "test.txt"
```

To recover the buffer, use the `-r` option with the file name.

```
$ vi -r test.txt
```

(`vi` opens the file)

Now Try This!

1. Edit the `myfile.txt` file in a terminal window.
2. Insert a line of text at the top, and press Return. Do not save the file.
3. Exit the window.
4. Open a terminal window and use this command to display the first 3 lines of the file (to verify your line was not saved).

```
head -n 3 myfile.txt
```

5. Use the `vi -r myfile.txt` command to restore your changes.

6. Quit `vi` with the `:wq` command.

Hint

To exit the terminal window, double-click the window menu button (upper-right corner of the window).

Solution

```
$ head -n 3 myfile.txt
The fast-moving brown fox jumped over the fence.

This is an explanation of the vi command.
$ vi -r myfile.txt
This is the text I typed and did not save
The fast-moving brown fox jumped over the fence.

This is an explanation of the vi command.
...
:wq
```

Customizing vi

The `vi` editor does not automatically wrap long lines. You must press the Return key. When searching for text, the text may 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 options that you can customize. You can display the options from command mode with the `ex` command `:set all`.

The output may look similar to that shown in the example.

Example 8-3: vi Set Options

```
:set all
noautoindent    nonumber                noslowopen
autoprint       nonovice                nosourceany
nitrite         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=2                warn
nolisp          sections=NHSHH HUuhsh+c window=47
nolist          shell=/bin/ksh          wrapscan
magic           shiftwidth=8            wrapmargin=0
mesg            noshowmatch             nowriteany
nomodeline      noshowmode
```

Setting Options

Two types of options can be changed using the `:set` command.

- Toggle options, which are either on or off

  `:set` *`option`* (on) and `:set no`*`option`* (off)

  For example, to ignore the case of the words during a search, type `:set ignorecase`.

  To match the case of the words during a search, type `:set noignorecase`.

- Options that take a numeric or string value

  `:set` *`option`*`=`*`value`*

  For example, to automatically insert a carriage return 10 columns from the right margin, type `:set wrapmargin=10`.

It may be tedious having to customize `vi` each time a file is edited. You can add these options to a file named `.exrc` in your home directory. When using an `.exrc` file to customize `vi`, leave off the colon (:). The commands are executed by `ex` before it enters `vi`.

Summary

Introducing the UNIX Editors

UNIX systems include a number of editors, including line editors such as `ex`, and screen editors such as `vi` and `emacs`. The Common Desktop Environment (CDE) also includes a basic text editor with a graphical user interface.

Editing with vi

Use the `vi` command with a file name to create a new file or edit an existing file.

The `vi` editor has three modes:

- Command mode, which enters commands to manipulate text
- Text input mode, which enters text
- Last line mode, which enters `ex` commands

Moving the Cursor

There are numerous commands to move the cursor around a text file. The `vi` commands can be preceded with a count to specify the number of times the command is executed.

Finding, Inserting, and Changing Text

- A file can be searched based on a pattern
- Text can be inserted or appended
- Text can be changed

Copying, Moving, and Saving Text

- Text can be copied and moved to another location
- The last change can be undone or repeated
- Files can be overwritten or changes discarded
- More than one file can be edited
- `vi` can be customized

Additional Exercises

Introducing the UNIX Editors: Exercise

These questions provide a review of the material.

1. List several line editors available on most UNIX systems.
2. List several screen editors available on most UNIX systems.
3. When should someone use the `ed` editor?
4. What is the advantage of the `vi` editor?

Introducing the UNIX Editors: Solution

1. Line editors available on most UNIX systems include `ed`, `ex`, `edit`, and `sed`.
2. Screen editors available on most UNIX systems include `vi`, `emacs` and the CDE text editor.
3. `ed` is the only editor available in single user mode.
4. The `vi` editor includes the powerful commands of the `ex` editor, while providing a full screen display.

Editing with vi: Exercise

1. Use the `vi` editor to create a file containing the lines Line 1, Line 2, through Line 10. Save the file as `lines.txt`.
2. Move the lines so they are numbered from 10 to 1. Save this version as `lines2.txt`.
3. Use the global replace command to change the word Line to line.
4. Write the changes and exit the editor.

Editing with vi: Solution

1. There are a number of ways to accomplish this, from inserting the ten lines, to inserting one line, copying it and pasting it ten times, then correcting the numbers.

```
Line 1
Line 2
...
Line 10
```

2. Use the `dd` command to cut a line, and the `p` command to paste it. Use the `:w lines2.txt` command to save the file.

```
Line 10
Line 9
...
Line 1
```

3. The `:%s/Line/line/g` command results in the following

```
line 10
line 9
...
line 1
```

4. The `:wq` command writes the changes and exits the editor.

9

Redirecting, Filtering, and Piping

Unit Overview

Introduction

The Tru64 UNIX operating system includes three very powerful tools that allow you to control the way commands handle data input and output.

- Redirection of input and output
- Filters that read from standard input and print to standard output
- Piping the output of one process to the input of another

This unit explains the use of these tools and demonstrates some of their power.

Objectives

To use these Tru64 UNIX tools, 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

Resources

For more information on the topics in this unit, see the following:

- *Common Desktop Environment: User's Guide*
- *Programming Support Tools Reference*, Chapter 2

Redirecting Input and Output

Overview

You can redirect standard input or output if you want input to come from somewhere other than the terminal keyboard, or output to go somewhere other than the terminal screen.

With the Korn shell, you can redirect standard input and output using the following symbols.

>	Redirects standard output to a specified file
>>	Redirects standard output by appending output to a file
<	Redirects standard input

Using the > Symbol

Use the > symbol to replace the contents of a file with the output of a command, even if the file already exists (except if the `noclobber` option is set).

The > symbol:

- 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` produces an empty file)

Here are some examples of the use of the > symbol.

`$ cat chap1 > newchap1`	Directs output of `cat chap1` to file `newchap1`
`$ ls > listfile`	Directs output of `ls` command to file `listfile`
`$ man cd > cdref`	Directs output of `man cd` command to file `cdref`
`$ date > date_file` `2> date_err`	Directs output of `date` command to `date_file` and directs any error messages to `date_err`

Using the >> Symbol

The >> symbol also redirects standard output. This symbol appends the output of a command to the named file.

These examples show the use of the >> symbol in various contexts.

`$ ls -l >> dirlist`	Appends a long listing of the current directory to a file named `dirlist`

$ **cat workphones >> homephones**	Appends output of `cat workphones` to a file named `homephones`
$ **cat report1 report2 >> report3**	Appends output of `date` command to `date_file` and directs any error messages to `date_err`

Using the < Symbol

The < symbol redirects standard input. It tells the system to look for input from the file that is specified after the < symbol. The syntax is as follows:

command-name < *input-filename*

command-name	Any command that accepts standard input
input-filename	File that contains the input for the specified command

The following example shows the use of the < symbol.

Example 9-1: Using the < Symbol

```
$ cat < chap3
Chapter 3
.
.
.
```

In this example, the `cat` command displays the contents of `chap3` on your terminal screen. If you did not supply a file name, `cat` would have read its input from the keyboard.

Now Try This!

Display the contents of your home directory `.cshrc` file using the `cat` command.

Solution

You can use the < operator to redirect standard input so that `cat` displays the contents of `.cshrc`.

```
$ cat < .cshrc
set notify
```

Using Filters

Overview

A **filter** is a command that reads standard input, processes the input, and prints the result to standard output.

Using the wc Filter

The `wc` (word count) 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
- Provides a count of the number of lines and the number of characters in the file(s)

The `wc` output, reading from left to right, displays the number of lines, words, and characters in the input. If a file is specified, the file name is displayed at the extreme right.

The following example shows the output generated when `wc` is applied to the `newfile` file.

Example 9-2: Using the wc Command

```
$ wc newfile
     10      27      139     newfile
$
```

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 9-1: wc Command Options

Option	Function
`-l`	Prints only the number of lines in a file
`-w`	Prints only the number of words in a file
`-c`	Prints only the number of characters in a file

If you run the `wc` command on standard input (without specifying a file name), you must press Ctrl/D to display the output of the command.

Now Try This!

Determine the number of lines, words, and characters in the `/etc/passwd` file.

Solution

The `wc` command allows you to determine the number of lines, words, and characters in an entry you make at the keyboard, or in an existing file. To determine the number of characters, words, and lines in the `/etc/password` file, enter the following command:

```
$ wc /etc/passwd
26    79    1681    /etc/passwd
```

This `passwd` file contains 26 lines, 79 words, and 1681 characters.

Sorting

Overview

Another useful filter is the `sort` command, which alphabetizes its input according to the first letter in each line, and prints the sorted list to standard output. The following example displays the result of using the `sort` filter on the `hot` and `cool` files.

Example 9-3: Using the sort Command

```
$ cat hot
orange
yellow
scarlet
magenta

$ cat cool
 aqua
 turquoise
 violet
 chartreuse

$ sort hot cool
aqua
chartreuse
magenta
orange
scarlet
turquoise
violet
yellow
```

sort Command Options

Common options to the `sort` command are shown in the following table.

Table 9-2: Common sort Command Options

Option	Example	Function
`-f`	`sort -f cool hot`	Sorts files `hot` and `cool`, disregarding whether the letters in the file are uppercase or lowercase
`-r`	`sort -r cool`	Sorts the file `cool` in reverse alphabetical order
`-u`	`sort -u hot cool`	Sorts files `hot` and `cool`, eliminating any duplicates; ensures that each item in the output is unique
`-m`	`sort -m file1 file2`	Reads files `file1` and `file2` and merges the results of each on the screen; works only on files that are already sorted
`-o`	`sort -o warm hot`	Sorts the file `hot` and places the output in the file name immediately following the `-o` flag (`warm`); enables sorting a file onto itself

Sorting by Different Fields

You may need to sort files that are more complex than the `hot` and `cool` files. Often, files of data contain more than one type of information, like the `family` and `baseball` files shown in the following example.

Example 9-4: Data Files with Multiple Fields

```
$ 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
```

```
$ 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
```

Sorting by Field Position

The following table displays the options to the `sort` command that relate to field position.

Table 9-3: Field Position Options for the sort Command

Option	Result
`+pos`	Starts sort key in specified field; note that numbering of fields begins at zero
`-pos`	Stops sort key in specified field position
`-n`	Sorts fields according to numeric values instead

The following examples show two methods of sorting the `family` file by fields.

Example 9-5: Sorting by Fields

```
$ 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      101-4019
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
```

Results of Sorting Numerically by Fields

The following example shows the result of sorting the `baseball` file numerically by fields.

Example 9-6: Sorting Numerically by Fields

```
$ sort +3n baseball
```

```
Rod Carew          .331      87
Ty Cobb            .367     118
Joe DiMaggio       .325     361
Reggie Jackson     .268     493
Lou Gehrig         .340     493
Ted Williams       .344     521
Mickey Mantle      .298     536
Willie Mays        .302     660
Babe Ruth          .342     714
Hank Aaron         .305     755
```

Now Try This!

1. Create a file called `familydates` that contains the first names, last names, and birthdays of a few members of your family. Each name should be on a separate line, and the birthdays should be in the format `m/d/yr`.
   John Doe 3/15/50

2. Use the `sort` command to print out your birthday list in alphabetical order by last name.

3. Sort the `familydates` file in order of birth dates. First try using the `+2` option; any October, November, and December (10, 11, and 12) dates will precede those in February. Sort the files again, using the appropriate option so that the birth dates are sorted in numerical order.

Solution

1. First, create a file called `familydates`.

```
$ cat > familydates
firstname  lastname    birthday
firstname  lastname    birthday
firstname  lastname    birthday
firstname  lastname    birthday
<Ctrl/D>
```

2. To sort `familydates` according to last name, use the `+1` option.

```
$ sort +1 familyfiles
```

3. To sort `familydates` according to month of birth, use the +2n option.

```
$ sort +2n familydates
```

Searching

Overview

Another filter, called `grep` (global regular expression printer), 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...
```

where *expression* is a pattern of characters, and *filename...* 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.

Finding Patterns

The following example displays the contents of the `price.fruit` file, then shows how to use the `grep` command to search `price.fruit` for the prices of fruits that are berries.

Example 9-7: Searching for Patterns with grep

```
$ cat price.fruit
apples              .49
oranges             .59
grapefruit         0.59
lemons              .25
limes               .33
cantaloupe         1.19
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
```

```
$ grep berries price.fruit
    raspberries        3.39
    strawberries       2.38
    blueberries        1.89
    cranberries        1.59
```

Finding Patterns in Specific Positions

The following series of examples shows how to use `grep` to find patterns in specific positions.

To find the price of a pound of apples, you could use the command:

```
$ grep 'apples' price.fruit
```

However, be aware that this command prints out the price of any fruit containing the pattern `'apples'`, not just the price of the apples.

```
$ grep 'apples' price.fruit
apples             .49
pineapples        1.49
```

If you want only the price of apples (not pineapples or crabapples), you can specify that you only want `grep` to print the pattern if it occurs at the beginning of the line. To anchor a pattern to the beginning of a line, use the caret (^) character, enclosing the entire expression in quotation marks.

```
$ grep '^apples' price.fruit
apples         .49
```

You can also anchor patterns to the end of the line using a dollar sign ($), enclosing the entire expression in quotation marks.

```
$ grep '59$' price.fruit
oranges            .59
grapefruit        0.59
cranberries       1.59
```

Finding Nonfixed Patterns

The `grep` command 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
- 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. A choice, or range, of characters is specified with square brackets (`[]`).

The following example prints any line in the `price.fruit` file that begins with the letter `l`, the letter `o`, or the letter `s`.

Example 9-8: Searching Files for Regular Expression

```
$ grep '^[los]' price.fruit
lemons                    .29
limes                     .33
oranges                   .59
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.

The next example prints every line in the `price.fruit` file that begins with letters `a` through `g`.

Example 9-9: Specifying a Range with 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
```

Using Negation with grep

You can use `grep` 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 following example.

Example 9-10: Using Negation with grep

```
$ grep -v '9$' price.fruit
lemons                .25
limes                 .33
bananas               .87
strawberries         2.38

$ grep '[^9]$' price.fruit
lemons                .25
limes                 .33
bananas               .87
strawberries         2.38
```

Other Options to the grep Command

Some other options to the `grep` command are shown in the following table.

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

It is also possible to use `grep` to match arbitrary patterns. With regular expressions, a period (`.`) matches any character in the ASCII set. Each period matches exactly one character. Consider the situations listed in the following table.

Table 9-4: Matching Arbitrary Characters

To Find all Lines in a File that...	Use the Expression...	To Match...
Are exactly 10 characters long	`'^..........$'`	Who knows? telephones Red Blue John Smith
Start with 4-character words beginning with the character R	`'^R...'`	Rick says "hello" Ride 'em Cowboy\ Red Blue
End with 6-character words	`'......$'`	Bring extra shoes Who knows Ride 'em Cowboy Red Blue?

Finding Special Characters

Escaped metacharacters are known as **literal characters**. When a metacharacter is escaped, it is accepted literally and not given special meaning. You can escape (`\`) the metacharacters listed in the following table by placing them within single quotation marks in an expression.

Use this Format	To Search for this Literal Character
`\[`	`[`
`\]`	`]`
`\.`	`.`
`\*`	`*`
`\$`	`$`
`\?`	`?`
`|`	`|`
`\^`	`^`
`\\`	`\`

For example, to find all items in the `price.fruit` file whose prices end in .59, use the following command.

Example 9-11: Finding Special Characters with grep

```
$ grep '\.59$' price.fruit
oranges                .59
grapefruit            0.59
cranberries           1.59
```

Finding Repeated Characters

Use an asterisk (*) inside regular expressions to indicate that the character immediately preceding it may be repeated any number of times, including zero.

Using metacharacters to match repeating patterns is called **closure**. The following format specifies closure in regular expressions.

```
$ grep character* filename
```

where *character* is 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 9-12: Specifying Repeating Characters with grep

```
$ 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 the example.

Now Try This!

Create a file called `veg.price` with the following contents:

```
lettuce        .89        carrots         .69
tomatoes       .69        celery          .59
broccoli       .79        string-beans   1.29
cauliflower    .87        onions          .29
parsley       2.14        asparagus       .89
avocado       1.19        corn            .85
```

1. Print out all the vegetables 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 the vegetables whose names start with c, a, or p, with prices under $1.

Solution

1. `$ grep '^c' veg.price`
2. `$ grep '9$' veg.price`
3. `$ grep -v '9$' veg.price, or`
   `$ grep '[^9]$' veg.price`
4. `$ grep '^[a-m]' veg.price`
5. `$ grep '[0 ]\.[0-7][0-9]$' veg.price`
6. `$ grep '^[acp].*[0 ]\...$' veg.price`

Using Pipelines

Overview

There are situations when you want to use the output from one command or program as input to another, thereby constructing a **pipeline** of commands. Normally the commands used in pipelines are commands that are filters.

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 the Pipeline Operator

Use the vertical bar (|) symbol to pipe output from one program or command into another. The | operator allows you to combine several operations in a single command line, optimizing processor performance. The following example compares not using and using pipelines to perform the following series of tasks.

- Generate a file that contains a long listing of files in your current directory
- Sort that listing numerically according to the size of the files
- Search the sorted list for all files that end in .txt

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
```

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
```

Using the tee Command

The `tee` command reads from standard input and writes the input both to standard output and to a specified file name. The format of the `tee` command is:

```
tee [-a] filename
```

If the `-a` option is used, the output is appended to the file instead of replacing the old contents.

Now Try This!

1. From your home directory, list all the files in /`user`. This listing is very long and difficult to read because it quickly scrolls off the screen.
2. Again, list the files in `/user` and its subdirectories. This time, pipe the output to the `more` command. 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 quit displaying the listing.
3. Without creating any new files, generate a listing in order of size of all the files in your home directory.

Solution

1. **`$ ls -R /usr`**
2. **`$ ls -R /usr | more`**
3. **`$ ls -al | sort +4n`**

Using 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 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 one of the most powerful filters. 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

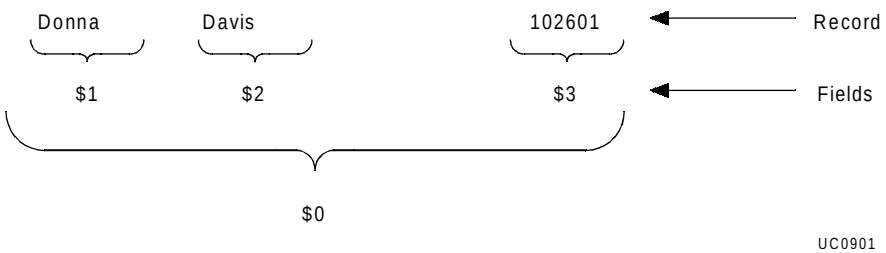
Introduction to awk Terminology

A **record** refers to a line of input. The different parts of a record are called **fields**.

- Fields are separated by a field separator, which, by default, is a space.
- In awk, fields are numbered differently than the way they are numbered in sort.
  - — Numbering 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.
  - — The notation to refer to the fifth field in a record is $5.

The following figure shows the way fields are numbered in awk.

Figure 9-1: awk Field Numbering Conventions



Using the awk Program Structure

The 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> }
```

Using awk

There are two ways to use the awk utility, from the command line, or through a program.

You can invoke awk on the command line, as shown in the following example.

Example 9-13: Invoking awk from the Command Line

```
$ cat names
  Donna Davies
  Peter Parker
  Paul Baker
$ awk '{print $2, $1}' names
  Davies Donna
  Parker Peter
  Baker Paul
```

Alternatively, you can create an awk program and run awk with the -f option. Use this method for more complex commands and reports. The following example uses the /etc/passwd file. The result may vary from system to system.

Example 9-14: Using an awk Program

```
$ cat > list.awk
{
FS = ":"
OFS = "<TAB>"
}
{print $1, $5}
<Ctrl/D>

$ 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 shows some of the expressions used in writing awk programs. These expressions are really predefined variables that can be changed to a new value within an awk program.

Table 9-5: awk Program Expressions

Expression	Significance
BEGIN <action>	<action> will be performed once, before any other <action> in the program, and before any input is processed
END <action>	Causes <action> to be performed after other <action>s in the program, and after all input has been processed
NR	Number of records that have been read at the time of evaluation
NF	Number of fields in the current record
FS	Input field separator
OFS	Output field separator

Comparison Operators

The symbols shown in the following table are used by awk to evaluate and compare expressions. They are the same symbols used in the C programming language.

Table 9-6: Comparison Operators for awk

Symbol	Definition
==	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

Using Logical Operators with awk

Logical operators are Boolean statements that compare two or more expressions.

AND (&&)	Both condition 1 **AND** condition 2 are true
OR (||)	Either condition 1 **OR** condition 2 is true

The phones file shows the use of logical operators with awk.

Example 9-15: Using Logical Operators with awk

```
$ cat phones
Augusta ME (207)271-2211
Maynard MA (508)493-4123
Burlington VT (802)657-2895
Nashua NH (603)884-2413
Cambridge MA (617)654-9531
Charleston SC (803)744-9046
Stamford CT (203)258-3232
Syracuse NY (315)453-1057
Kalamazoo MI (616)384-8310
Boston MA (617)106-5472
Littleton MA (508)952-4839
Providence RI (401)107-8932
```

The next example shows the use of && (the AND logical operator) in awk programs.

Example 9-16: Using && with awk

```
$ 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
Boston --- 106-5472
```

The next example shows the use of || (the OR logical operator) in awk programs.

Example 9-17: Using || with awk

```
$ 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
Boston MA---617-106-5472
Littleton MA---508-952-4839
Providence RI---401-107-8932
$
```

Now Try This!

The following exercises give you an opportunity to use some of the awk commands and program features. The phones file will be used in some exercises.

```
$ cat phones
Augusta ME (207)271-2211
Maynard MA (508)493-4123
Burlington VT (802)657-2895
Nashua NH (603)884-2413
```

```
Cambridge MA (617)654-9531
Charleston SC (803)744-9046
Stamford CT (203)258-3232
Syracuse NY (315)453-1057
Kalamazoo MI (616)384-8310
Boston MA (617)106-5472
Littleton MA (508)952-4839
Providence RI (401)107-8932
```

1. Use `awk` to print the list of phone numbers from the `phones` file in the first field, followed by the city and state.
2. Use `awk` to search the `/etc/passwd` file for any users with a password field of just `*`, and display their username and real name.
3. Write an `awk` program that searches the `phones` file for numbers with a state of MA or NY and print the numbers by city or state. Use the program to produce a directory of phones in MA and NY, sorted by phone number, with city and state.

Solutions

1. **$ awk ' {print $3, $1, $2 } ' phones**
2. **$ awk '{FS=":"} ($2=="*") { print $1, "->", $5 } ' / etc/passwd**
3. **$ cat > phones.awk**
   ($2=="NY") || ($2=="CT") { print $3, $1, $2 } '
   <Ctrl/D>

   $ awk -f phones.awk phones | sort

Summary

Redirecting Input and Output

With the Korn shell, you can redirect standard input and output as follows:

>	Redirects standard output to a specified file
>>	Redirects standard output by appending output to a file
<	Redirects standard input

Using Filters

The `wc` filter:

- Performs a word count on its input file(s), or on standard input if no file name is given
- Gives a count of the number of lines and the number of characters in the file(s)

Sorting

The `sort` command alphabetizes its input according to the first letter in each field, and prints the sorted list to standard output.

The `sort` command options allow you to designate different sequences for the `sort` command to follow when it examines and reformats the input.

Searching

The `grep` command searches its input for a pattern of characters, and outputs any line containing that pattern. The `grep` utility has the ability to look for patterns that are not fixed, called regular expressions.

You can also use `grep` to look for lines that do not contain a specific pattern, or to match arbitrary characters.

Using Pipelines

Pipelines allow you to use the output of one command as input to another. You can use the vertical bar (`|`) symbol to pipe output from one program or command into another.

Using the awk Utility

The `awk` utility processes ASCII text files and generates reports involving selective retrieval and string manipulation. It is used primarily to generate tables and reports from files of raw data.

Summary

10

Controlling Processes

Unit Overview

Introduction

A process is a running program or command. This unit discusses the concept of a process and how to manage user processes. When you enter a command, the command executes as a subordinate process (child) of the current process (parent). All user processes are children of the login shell process.

Objectives

To control processes, you should be able to:

- Define the concepts of process and job
- Obtain the status of a process
- Control and manage processes
- Schedule processes to run at other times

Resources

For more information on the topics in this unit:

- Tru64 UNIX reference pages

Describing the Process Concept

Overview

The operating system manages both hardware and software resources. One of the software resources that it manages is an executing program. We call an executing program a **process**.

A process:

- Can be created and destroyed
- Has allocated resources
- Has an associated environment that:
  - Is inherited from the parent process
  - Consists of all information relative to the process
  - Can be changed by issuing shell commands
- Creates other processes
- Communicates with other processes

Process Environment

A process environment includes:

- 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

Creating a Process

When you start the operating system, the `init` process starts. The `init` process creates login processes that wait for terminal input.

The superuser owns the `init` process and the console controls it. The console is the terminal device the kernel uses to record errors in the event of serious system problems.

When a user logs in, 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.

Grouping Shell Commands

You can submit more than one command at a time by:

- Using the semicolon (;) to separate the commands that are to execute in sequence in the current shell
- Using parentheses () to surround commands that are to execute together in a subshell

This example shows how to group commands using semicolons.

Example 10-1: Grouping Commands with Semicolons

```
$> pwd; date; cd /etc; pwd; ls -al | grep usr; pwd
/usr/users/joe
Mon May 18 11:10:50 EDT 1998
/etc
lrwxrwxrwx   1 root     system    14 May  4 11:34 dt -> ../usr/var/.dt
lrwxrwxrwx   1 root     system    15 May  4 11:44 rmt -> ../usr/sbin/rmt
lrwxrwxrwx   1 root     system    24 May  4 11:17 termcap -> ../usr/share/lib/
termcap
/etc
$
```

Now Try This!

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.

Solution

```
$> who
joe          console     May 14 11:50
joe          pts/1       May 14 11:50
joe          pts/2       May 18 11:08
$> date
Mon May 18 11:12:51 EDT 1998
$> who; date
joe          console     May 14 11:50
joe          pts/1       May 14 11:50
joe          pts/2       May 14 14:56
Mon May 18 11:12:51 EDT 1998
```

This example shows how to group commands using parentheses.

Example 10-2: Grouping Commands with Parentheses

```
$ pwd; date; (cd /etc; ls -al | grep usr); pwd
/usr/users/joe
Mon May 18 11:15:08 EDT 1998
lrwxrwxrwx   1 root     system        14 May  4 11:34 dt -> ../usr/var/.dt
lrwxrwxrwx   1 root     system        15 May  4 11:44 rmt -> ../usr/sbin/rmt
lrwxrwxrwx   1 root     system        24 May  4 11:17 termcap -> ../usr/share/
lib/termcap
/usr/users/joe
$
```

Notice that the example starts and ends in the `/usr/users/you` directory. The `cd` command in the subshell does not affect the current shell.

Now Try This!

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. Use the `cd` command to return to your home directory.

   c. Enter the command: (`cd /bin ; ls`).

   d. Use `pwd` to verify that you are still in your home directory.

Solution

1. This is the solution using semicolons to group commands.

```
$ cd /bin ; ls
Mail       dxfc        mig            strip Rsh        dxmail
mkcatdefs      strip2.20
  . . .
```

2. The is the solution using a subshell.

```
$ pwd
/bin
$ cd
$ pwd
/usr/users/jones
$ (cd /bin; ls)
Mail       dxfc         mig           strip
Rsh        dxmail       mkcatdefs     strip2.20
. . .
$ pwd
/usr/users/jones
```

Foreground Jobs

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 runs in the **foreground**, the shell cannot accept any further input until the job finishes. In other words, you cannot type other commands to the shell while you wait for the job to finish.

Background Jobs

To allow more than one job to run at a time, jobs can 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 (spreadsheet calculations, complex mathematical calculations)
- Does not require interactive input (sorts, compilations)

To process a job in the background, append an ampersand (&) to a shell command.

```
command &
```

Jobs that are run in the background need to have their input (if any) and output (if any) redirected either **from** or **to** a file. This precludes any confusion with an application running interactively in the foreground at the same time.

This example shows two commands executing in the background.

Example 10-3: Running Jobs in the Background

```
$ ls -al /usr | grep ^d > user-directories &
[1]     1768
$ find / -name emacs -print > find-emacs &
[2]     1765
[1] - Done ls -al /usr | grep ^d > user-directories &
$
```

When you start a background job, the system displays a job number and PID(s). You must know the job number to bring the job to the foreground or otherwise manage the process.

Now Try This!

To run a job in the background that performs a listing of the `/etc` directory:

1. Enter the following command:

   ```
   $ls -l /etc > etc_dir.txt &
   ```

2. Use the `ls -l` command to show that the output file was created.

3. When the job is complete, display the output file.

Solution

Your results should look similar to this.

```
$ ls -l /etc > etc_dir.txt &
13953
$ ls -l etc_dir.txt
-rw-r--r--   1 joe      users      10906 May 18 11:23 etc_dir.txt
$ cat etc_dir.txt
total 1444
. . .
-rw-rw-r--   1 root     system         5 Mar  4  1997 exports
-rw-r--r--   1 root     system       291 Aug  4  1997 fstab
-rwxr-xr-x   1 bin      bin         4058 Nov 16  1996 gettydefs
-rw-r--r--   1 root     system       555 Aug  5  1997 group
-rw-rw-r--   1 root     system      2703 Mar  4 14:50 hosts
-rwxr-xr-x   1 bin      bin         2370 May 20  1996 hosts.equiv
-rw-rw-r--   1 root     system      2535 Apr 17  1997 hosts~
. . .
```

Obtaining Process Status

Overview

To determine a job's process status:

- Use the `ps` command to obtain the status of all your processes.
- Use the `jobs` command to obtain the status of a stopped or background job.

Using the ps Command

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 active child processes.

The format for the `ps` command is:

```
ps [-option]
```

The `ps` command has many options which provide customized output. For options, see `ps`(1).

Now Try This!

1. Enter the `ps` command without options.

   What is the process identification number (PID) of your shell process?

2. Enter the command:

   ```
   $ ps -l
   ```

What additional process information is available with this form of the process command?

Solution

1. In this example, the process ID of the shell process is 13938.

   ```
   $ ps
     PID TTY      S           TIME CMD
   13938 pts/4    S  +     0:00.23 -ksh (ksh)
   ```

2. Additional information includes the parent process ID (PPID), the user ID (UID), and priority of the job (PRI).

```
$ ps -l
F S           UID   PID   PPID    C   PRI  NI  ADDR  SZ  WCHAN  TTY      TIME CMD
80808001 S  +   205  13938  13869  0.0  44   0     0 200K wait  pts/4    0:00.24 sh
$ $
```

Status Information

With no options on the command line, the `ps` command provides process status with the information categories shown in the table.

Table 10-1: Process Status Information Categories

Header	Description
PID	Process identification number
TTY	Name of the terminal device that controls the process; in most cases, this is the terminal you were using when you entered a command
S	State of the process indicated by a sequence of up to four letters
TIME	Accumulated execution time of the process
CMD	Name of the command and its arguments

Using ps Options

Some common options to the `ps` commands are shown in this table.

Table 10-2: Selected Options for the ps Command

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
`-f`	Provides additional information including: user PPID, percentage of CPU usage, and the time the process started
`-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
`-u` *`ulist`*	Displays information about processes with the user IDs specified in `ulist`

This example shows the output obtained by entering a `ps` command with no options.

Example 10-4: Using the ps Command with no Options

```
$   ps
   PID TTY      S           TIME CMD
 13938 pts/4    S  +     0:00.23 -sh (sh)
$
```

This example uses the `ps` command with the `-ef` options. This display provides all processes and includes the user name.

Example 10-5: Using the ps Command with the -ef Options

```
$ ps -ef
UID         PID   PPID    C STIME    TTY            TIME CMD
root          0      0  7.1   May 12 ??         10:25:06 [kernel idle]
root          1      0  0.0   May 12 ??          0:01.85 /sbin/init -a
. . .
joe        7191   7150  0.0   May 14 ??          0:37.82 dtwm
joe       13750  13829  0.0 11:11:54 ??          0:00.07 /bin/sh -c dtpad -ser
joe       13827  13750  0.0 11:11:54 ??          0:05.95 dtpad -server
joe       13829      1  0.0 11:08:03 ??          0:00.63 /usr/dt/bin/ttsession
joe       13865  13725  0.0 11:08:10 ??          0:06.35 dtwm
joe       13869  13870  1.6 11:20:45 ??          0:05.96 /usr/dt/bin/dtterm -l
joe       13870  13865  0.0 11:20:45 ??          0:00.31 /usr/dt/bin/dtexec -o
joe       13875  13725  0.0 11:08:21 ??          0:00.04 sh -c /usr/bin/X11/dx
joe        7481   7463  0.0   May 14 pts/3       0:00.31 -ksh (kh)
joe       13938  13869  0.4 11:20:48 pts/4       0:00.40 -ksh k(sh)
root      13957  13938  0.0 11:28:00 pts/4       0:00.07 ps -ef
$
```

Listing Background Jobs

The `jobs` command displays a list of stopped or background jobs.

The format of the jobs command is:

```
jobs [-lp]
```

The `jobs` command without options displays a list of background or stopped jobs with job number and the job status. With the `-l` option, `jobs` also shows the PID, as shown in this example. With the `-p` option, `jobs` shows only the PID.

Example 10-6: 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
$
```

Controlling and Managing Jobs

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 Jobs

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.

This example shows the `vi` editor started and then suspended with Ctrl/Z.

Example 10-7: 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 `fg` command format is:

```
fg job_id
```

If `job_id` is not used, the most current job, as indicated by the plus (+) in the jobs output, is placed in the foreground. You can specify the job number by preceding it with a percent sign (%), or you can specify the PID. If you do not specify the job, the system uses the current job.

This example demonstrates the `fg` command.

Example 10-8: Using the fg Command

```
$ fg          Places the current job in the foreground
$ fg %3       Places job 3 in the foreground
$ fg 10147    Places the job with PID 10147 in the foreground
```

Now Try This!

1. Start up the `vi` editor by issuing the following command:

   ```
   $ vi file.txt
   ```

2. Stop the process by typing Ctrl/Z.
3. Determine the job number of the process.
4. Restart the editing job.
5. Exit the editing job by typing ZZ.

Solution

1. $ **vi file.txt**

   ```
   . . .
     ~
   ~
   "file.txt" [New file]
   ```

2. **<Ctrl/Z>**
3. [2] + Stopped vi file.txt
4. $ **fg %2**

   ```
   . . .
   ~
   ~
   ```

5. **ZZ**

   ```
   $
   ```

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_id
```

When used without the job ID, 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 10-9: Using the bg Command

```
$ bg %2
[2]     ls -l /etc > files.txt &
```

Now Try This!

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 background.

Solution

1. The job number is 1; the `PID` is 4529. Your results will vary but should be similar.

```
$ (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
$
```

2. You must wait for the command to complete.

```
$ (sleep 20; ls) &
[2]     4532
$ fg
(sleep 20; ls)
```

Using the kill Command

You can communicate with a foreground process by 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 with the `trap` command to direct a script to carry out appropriate action. An application can choose to ignore signals or to catch them and attend to them at a later time. If this is not the case, the `kill` command will terminate the process.

The format of the `kill` command is:

```
kill [-signal] job_name
```

You display the `kill` signals with the `kill -l` command. The table shows the list of signals from the `kill-l` command.

Table 10-3: Tru64 UNIX kill Signals

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

Table 10-3: Tru64 UNIX kill Signals (Continued)

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

Specify the signal with either a name or number. The default `kill` signal is `TERM`, which terminates any process that does not ignore or catch the signal.

This example shows how to use the `kill` command.

Example 10-10: 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 terminates a process that does not ignore or catch the signal.

❷ Sends the `QUIT` signal to job 1.

❸ Sends the `KILL` signal to the job with `PID` 10652. This is the surest kill, and often terminates the process even if `TERM` does not terminate it.

Process Priority

The operating system assigns each process a process **priority**. This priority schedules 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 allows you to run a command at a lower priority. The `nice` priority value:

- Controls how fast your process executes
- Ranges from -20 (highest priority) to 19 (lowest priority)
- Default is 10

The superuser can use the highest priorities (negative numbers).

The operating system maps the nice value to an appropriate priority on that system.

The format of the `nice` command is:

```
nice [-n priority] command [argument ...]
```

In this example, the script file `find.ksh` is run in the background with a decreased priority.

Example 10-11: Using the nice Command

```
$ nice -n 15 find.ksh &
```

Nonsuperusers can only decrease their own priority, they cannot increase their priority. This table shows some useful priority levels.

Table 10-4: Useful Priority Levels

Priority Level	Description
+19	Runs only when nothing else in the system is running
+10	Default level when no number is specified with `nice`
0	Base scheduling priority
-1 to -20	Makes processes run very fast (superuser only)

Now Try This!

To reduce the priority of a command:

1. Enter 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.

Solution

1. The NI column shows the process priority.

```
$ 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
```

2. The process priority is now changed by 10.

```
$ nice -n 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
```

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 some 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 execute long jobs when user interaction is unnecessary. However, you may have to wait for a background job to finish executing before you can complete other tasks. The `wait` utility waits until all process IDs known to the invoking shell have terminated.

The format of the `wait` command is:

```
wait [job_name]
```

Suppose you enter the `ls` command using the `-R` flag 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 completes.

This example demonstrates the use of the `wait` command in an interactive situation.

Example 10-12: 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` 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.

Options to the `wait` command are shown in this table.

Table 10-5: Korn Shell Options to the wait Command

Option	Command	Waits for:
(No option)	wait	All background commands 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

Shell commands, shell scripts, and system commands either:

- Succeed and display the results on the screen, or
- Fail and the shell prints some type of diagnostic error message

The shell also sets a reserved variable, ?, with the exit status of the command. It is normally set to:

- Zero (0) to indicate successful execution
- Nonzero to indicate failure

The user documentation describes the meanings of the status codes for specific commands.

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 10-13: Checking the 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
$
```

Now Try This!

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 anonymous /etc/passwd
```

Print out the exit status code.

Solution

1. A successful exit status code.

```
$ grep stu1 /etc/passwd
stu1:xrusfP3Jy5TiE:1713:15:Student One:/usr/class/stu1:/bin/ksh
$ print $?
0
```

2. An unsuccessful exit status code.

```
$ grep anonymous /etc/passwd
$ print $?
1
```

Scheduling Jobs to Run at Appropriate Times

Overview

To schedule jobs to run at appropriate times, you need commands that will schedule jobs to run

- **Once** at some time in the future
- **Periodically** at some time in the future

Scheduling Jobs to Run Once

There are two methods for executing commands just once at some time in the future. The commands are:

- The `at` command executes commands once at a specified time
- The `batch` command:
  - — Executes commands once when the system load permits
  - — Input comes from standard input

The `at` command has the following formats:

```
at [-cskm] time [date] [+increment] [command | file] ...
at -l[user]
at -n[user]
at [-fi] -r job_number ...  [-u user] [command | file] ...
```

The `batch` command has the format:

```
batch
command
<Ctrl/D>
```

Command Options

This table shows `at` command options.

Table 10-6: Command Options

Option	Function
`c`	Executes job in the C shell
`k`	Executes job in the Korn shell
`s`	Executes job in the Bourne shell
`m`	Mails a message on successful command execution
`l`	Reports your scheduled jobs

Table 10-6: Command Options (Continued)

Option	Function
r	Removes a job previously scheduled by at or batch, where *job_number* is the number assigned by at or batch; you can remove only your own jobs
n	Requests the number of files in the queue for the current user
f	Suppresses delete verification
i	Specifies an interactive delete
u	Deletes all jobs for the specified user; must be used with the -r option

Some at and batch commands are shown in the following examples.

Example 10-14: Using the at Commands

```
$ at -km 2300 cleanup.ksh              ❶
job Cindy.827553600.e at Fri Mar 20 23:00:00 1998

$ at -km now +2 days cleanup.ksh       ❷
job Cindy.827687090.e at Sun Mar 22 12:04:50 1998

$ at -km 2300 April 12 cleanup.ksh     ❸
job Cindy.829364400.e at Sun Apr 12 23:00:00 1998

$ at -km N mar 25 cleanup.ksh          ❹
job Cindy.827773200.e at Wed Mar 25 12:00:00 1998

$ at -km M mar 25 cleanup.ksh          ❺
job Cindy.827730000.e at Wed Mar 25 00:00:00 1998
```

Example 10-15: Using batch

```
$ batch      ❶
cleanup.ksh
<Ctrl/D>

job Cindy.827514957.b at Sun Mar 22 12:15:57 1998
```

❶ Use batch to execute cleanup.ksh when the system load allows.

Example 10-16: Managing at and batch Requests

```
$ at -l      ❶
Cindy.827553600.e Sun Mar 22 23:00:00 1998
Cindy.827687090.e Tue Mar 24 12:04:50 1998
Cindy.829364400.e Sun Apr 12 23:00:00 1998
Cindy.827600401.e Mont Mar 23 12:00:01 1998
Cindy.827773200.e Wed Mar 25 12:00:00 1998
Cindy.827730000.e Wed Mar 25 00:00:00 1998

$ at -r Cindy.827773200.e  ❷
at file: Cindy.827773200.e deleted
```

```
$ at -ru Cindy     ❸
at file: Cindy.827553600.e deleted
at file: Cindy.827600400.e deleted
at file: Cindy.827600401.e deleted
at file: Cindy.827600402.e deleted
at file: Cindy.827730000.e deleted
$ at -l    ❹
$
```

❶ Check the scheduled `at` and `batch` jobs.

❷ Use the `at -r` option to remove job `Cindy.827773200.e`.

❸ Delete the remaining jobs with the `at -ru Cindy` command.

❹ No jobs left.

Now Try this!

Create a script file that will execute the `ls` command in your home directory. Use the `date` command to put a date stamp on the operation. After you have completed this step, submit the file using the `at` command to run within 5 minutes. After the job has executed, check your mail to see the results.

Solution

```
$ cat listing.ksh
     ls
 date
 $ chmod 755 listing.ksh
 $ at -m now + 5 minutes listing.ksh
 job you.757608283.a at Sat Jan 03 09:44:43 1998
 $ Mail
 Mail $Revision: 4.2.4.2 $  Type ? for help.
 "/usr/spool/mail/you": 2 messages 2 new
 >N  1 you   Sat Jan  3 09:44  9/279
  N  2 root  Sat Jan  3 09:44  56/698
 ? 1
 Message  1:
 From you Sat Jan  3 09:44:44 1998
 Date: Sat, 3 Jan 1998 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 Sat Jan  3 09:44:45 1998
 Received: by sys1.dec.com; id AA05137; Sat, 3 Jan 1998 09:44:44 -0500
 Date: Sat, 3 Jan 1998 09:44:44 -0500
 From: system PRIVILEGED account <root>
 Message-Id: <9401031444.AA05137@sys1.dec.com>
 Apparently-To: you

 TUTORIAL
```

```
 for_dir.ksh
 . . .
     Mon Sat  3 09:44:44 EST 1998

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

Now Try this!

Modify the script file created in the previous activity to redirect the output to a file, then submit the job to run in a few minutes.

Solution

```
$ cat listing.ksh
date > daily-log.txt
ls >> daily-log.txt
$ at now + 2 minutes listing.ksh
job you.757610265.a at Sat Jan 03 10:17:45 1998
$ cat daily-log.txt
Mon Sat  3 10:17:46 EST 1998
TUTORIAL
daily-log.txt
for_dir.ksh
 . . .
```

Scheduling Periodic Jobs

The `cron` daemon runs shell commands at a specified date and time on a periodic basis. The `cron` daemon:

- Starts up when the operating system initializes and continues running until system failure or shutdown
- Creates a log of activities
- Checks `crontab` and `at` files in the directory `/usr/spool/cron/crontabs` periodically for jobs
- Starts jobs contained in the `/usr/spool/cron/crontabs` directory by:
  - — Invoking a subshell from the invoking process' $HOME directory
  - — Not running the `.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

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 `crontab` command has the format:

```
crontab [file]
crontab -l | -v
crontab -r
```

Command Options

The `crontab` command options are shown in the table.

Table 10-7: Some crontab Command Options

Option	Function
`l`	Displays the contents of your `crontab` file
`e`	Edits a copy of your `crontab` entry; invokes the editor specified by the `EDITOR` environment variable or uses `vi` by default.
`r`	Removes the `crontab` file from the `crontab` directory
`v`	Verbose mode; 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.

Example 10-17: Using the crontab Command

```
$ crontab weekly_cleanup
$ crontab -l
30 13 * 1 1-5 echo "I was cronned"
$ crontab -v
crontab file: your submission time: Fri Jan 02 13:00:29 1998
```

Creating a crontab File

Each user can have one `crontab` file active at a time. The `crontab` file may contain more than one entry line. Each line in the file consists of six fields separated by spaces or tabs.

This example shows the format of a `crontab` file.

Example 10-18: A 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)
```

The example 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)
- Two numbers separated by a dash to indicate an inclusive range
- A list of numbers, separated by commas, that selects all numbers in the list
- An asterisk, which indicates all legal values

This example shows a sample `crontab` file.

Example 10-19: A Typical crontab File

```
# @ (#) crontab
#
#
0  5 1 *   * /usr/adm/billing
0 20 * * 1-5 /usr/adm/cleanup
```

Now Try This!

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 the job executes correctly, then remove your `crontab` file.

Solution

```
$ date
Sat Jan  3 10:36:47 EST 1998
$ cat mycron
40,45,50 10 * * * date >> crontab.txt
$ crontab mycron
$ crontab -l
```

Summary

Describing the Process Concept

A process is a program that is executing. We call the associated information the process environment. You create processes by running programs and commands.

Jobs can be run in the background by appending an & sign at the end of the command line.

Obtaining Process Status

You can check the status of a job using the `jobs` or `ps` command.

Controlling and Managing Jobs

- 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.

Scheduling Jobs to Run at Appropriate Times

Use the `at` and `batch` commands to run commands once at a later time. To run commands periodically at a later time, use the `crontab` command

11

Communicating with Other Users

Unit Overview

Introduction

Communicating with other users is a function you perform every day. This unit discusses how to:

- Obtain information about other system users
- Send and receive electronic messages with a mail utility

Objectives

To electronically communicate with others, you should be able to:

- Use Mailer to send, receive and manage your electronic mail
- Execute commands such as `who`, `users`, and `finger`
- Use the `mailx` utility to send, receive, select, and maintain mail messages
- Customize your mail environment

Resources

For more information on the topics in this unit, see the following:

- *Common Desktop Environment: User's Guide*, Chapter 8
- *Command and Shell User's Guide*, Chapter 11

Using Mailer (CDE)

Overview

CDE provides the Mailer application for you to communicate with other users.

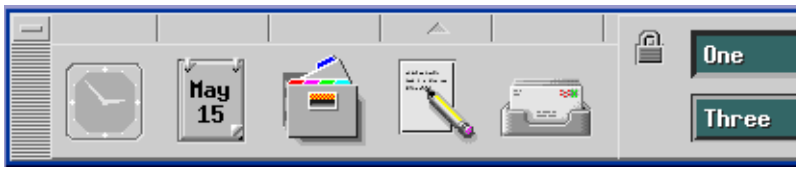
Using Mailer, you can:

- Create, send and receive messages
- Manage your incoming mail
- Customize the Mailer application for your working environment

Starting Up Mailer

To start Mailer, click the Mailer control on the Front Panel.

Figure 11-1: Mailer on the Front Panel



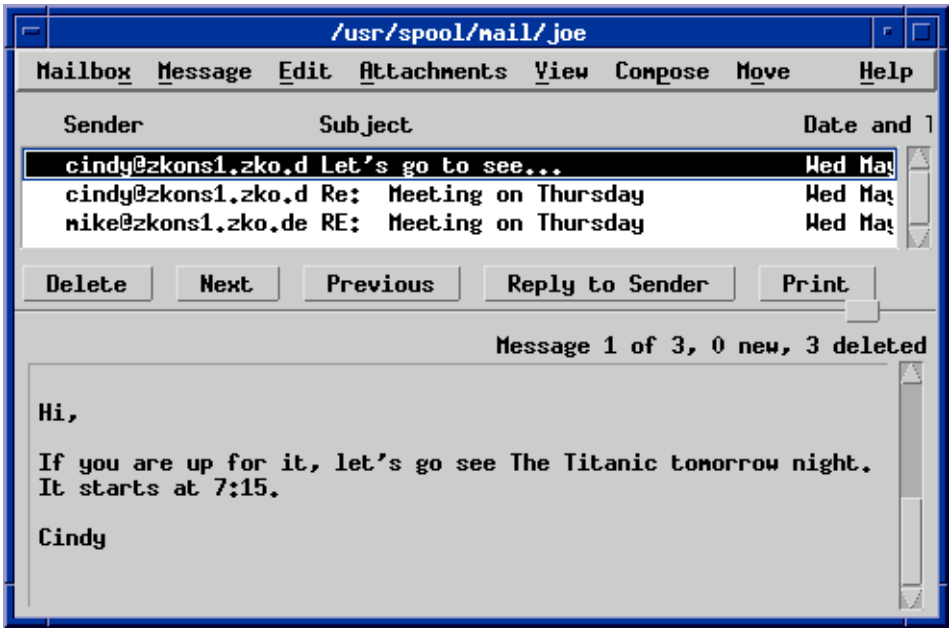
After clicking the control, Mailer displays the main window.

Main Window

Mailer's Main window consists of the following components.

Message Header list	Lists any message you receive
Message View area	Displays the body of a message
Sash	Draggable window element that resizes the Message Header List and View areas
Attachment window	Appears if you receive or create a message with an attachment; contains icons representing attachments

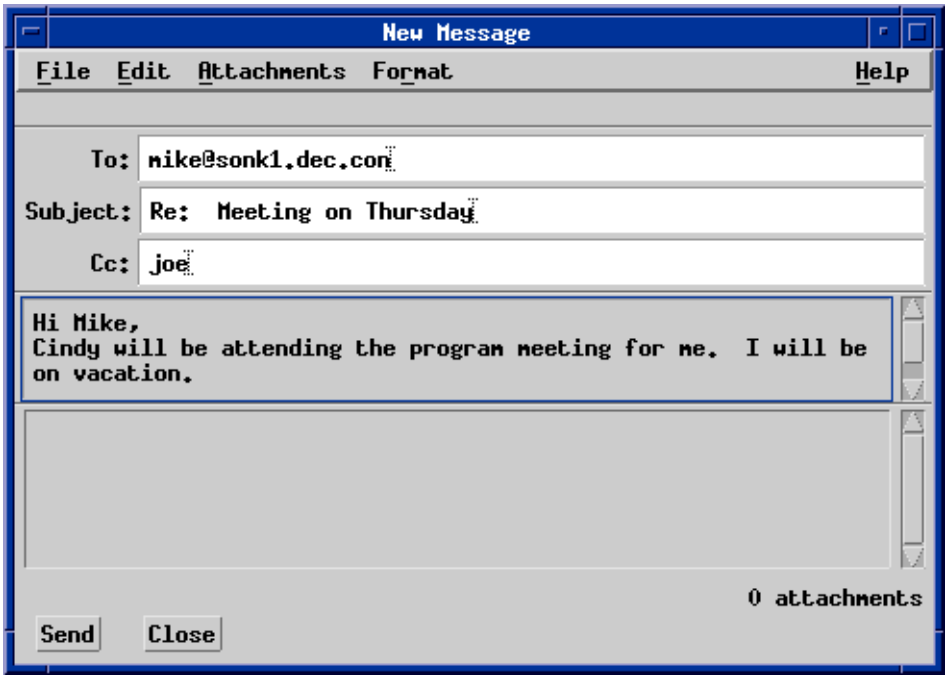
Example 11-1: Mailer's Main Window



Creating a Mail Message

To create a mail message, choose New Message from the Compose menu. Mailer displays the New Message window as shown.

Example 11-2: New Message Window



Now Try This!

Compose a mail message to remind yourself of a meeting for next week. Take the following steps:

1. Choose New Message from the Compose menu.
2. Type the Email address of the recipient in the To: field and press Return.
3. Type the title of the message or topic in the Subject: field and press Return.
4. Type the Email address of users to receive a carbon copy of the message in the CC: field and press Return.
5. Compose your message in the text area.
6. Click the Send button or choose Send from the file menu.

Need Help?

In the Help Table of Contents, click:

To Compose, Address, and Send A Message

Other Useful Features

When composing a message, you can also perform the functions shown in the table.

Table 11-1: Useful Mail Functions

Function	Menu Choice
Check spelling	Edit ->Check Spelling
Use a template	Format->Template
Perform a global replace of text	Edit ->Find/Change
Include another text file	File ->Include
Attach another file (image, document or executable file)	Attachments ->Add File

Now Try This!

Compose a new mail message to yourself, include a text file of information like a reference page and then spell check the message.

1. Choose New Message from the Compose menu.
2. Type the Email address of the recipient in the To: field and press Return.
3. Type the title of the message or topic in the Subject: field and press Return.
4. Type the Email address of users to receive a carbon copy of the message in the CC: field and press Return.
5. Choose Include from the File menu.

6. Select a file from the Mailer - Include screen.
7. Position your cursor at the beginning of the message and then choose Check Spelling from the Edit menu.
8. Click the Send button or choose Send from the file menu.

Need Help?

In the Mailer Help Table of Contents, click:

To Include a Text File in a Mail Message

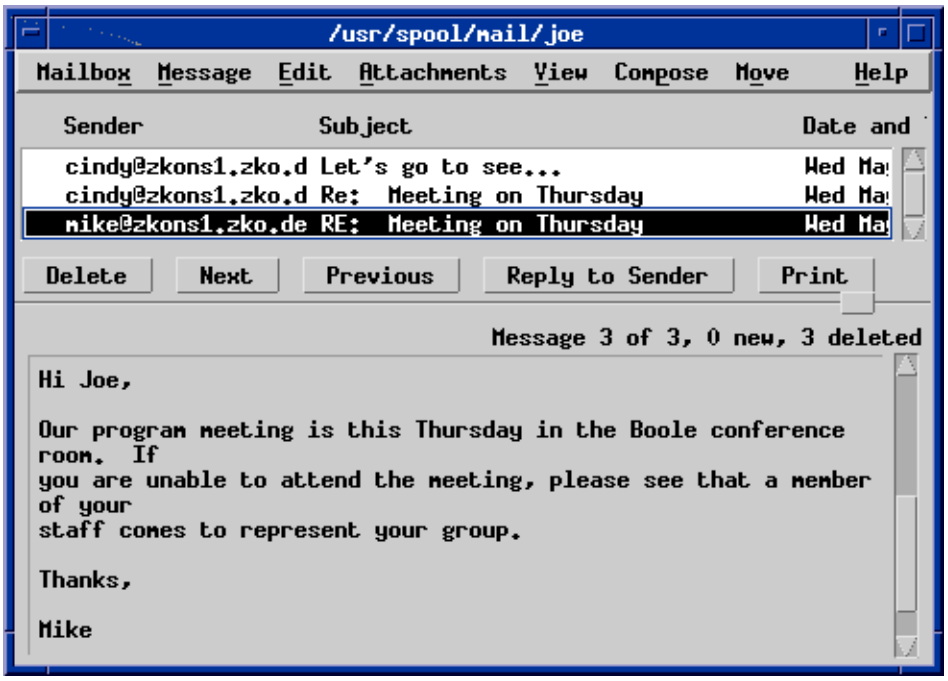
Viewing Your Mail

When new mail arrives for you, Mailer will indicate this by changing the Front Panel Mailer control, as shown here.

Figure 11-2: Mail Arrives

When you select a message for viewing from the Main window, Mailer displays the message in the Message view area of the Mail window.

Figure 11-3: Viewing Your Mail



Now Try This!

Look at the message you sent to yourself from the last activity. View the mail message using these steps.

1. Click on the Front Panel Mailer control. Result: The mailbox main window appears.
2. Choose Check for New Mail from the Mailbox menu to retrieve any messages waiting for delivery to your mailbox.
3. From the Message Header list, click the message to view.

   Result: Mailer displays the message in the Message view area of the window.

Need Help?

In Mailer's Help Table of Contents click:

To Read a Mail Message

Other Viewing Features

In addition to viewing your messages, you can perform the tasks shown in the table.

Table 11-2: Mail Viewing Functions

Mail Feature	Menu Choice
Sort your messages	View->By Date/Time, Sender, Subject, Size and Status
Reply to a message	Compose->Reply to Sender
Forward a message	Compose->Forward Message
Print a message	Message->Print or Click the Print button
Save a message in text format	Message->Save As Text
Delete a message	Message ->Delete or Click the Delete button
Find a message in a mailbox	Message->Find

Email and Alias Addresses

The Email address you use when composing your mail message can have more than one form. You may use the standard Email address format or you may use an alias.Review the format of each form.

Table 11-3: Email Addresses

Mail Address	Format	Example
Standard	`username@location`	`bobbyjo@ufl.edu`
Alias	`alias@location`	`bob@ufl.edu`

You can create an alias for lengthy or hard-to-spell Email addresses, distribution lists or a group of aliases. Once you have created the alias, you can use it in the To: field when composing a mail message.

Now Try This!

Create an alias for members of your project team. Call it `proj_members`. Follow these steps:

1. Choose Mail Options from the Mailbox menu. This brings up the Mail Options dialog box.
2. Choose Aliases from the Category menu of the Mail Options dialog box.
3. Type the name for your alias in the Alias field.
4. Type the user addresses that will make up the alias in the Addresses field. Each address should be in the form *username@location* and be separated by a comma, space or both.

5. Click Add to add the information in the Alias and Addresses fields to the Aliases list.

6. Click OK or Apply to make the addition.

Need Help?

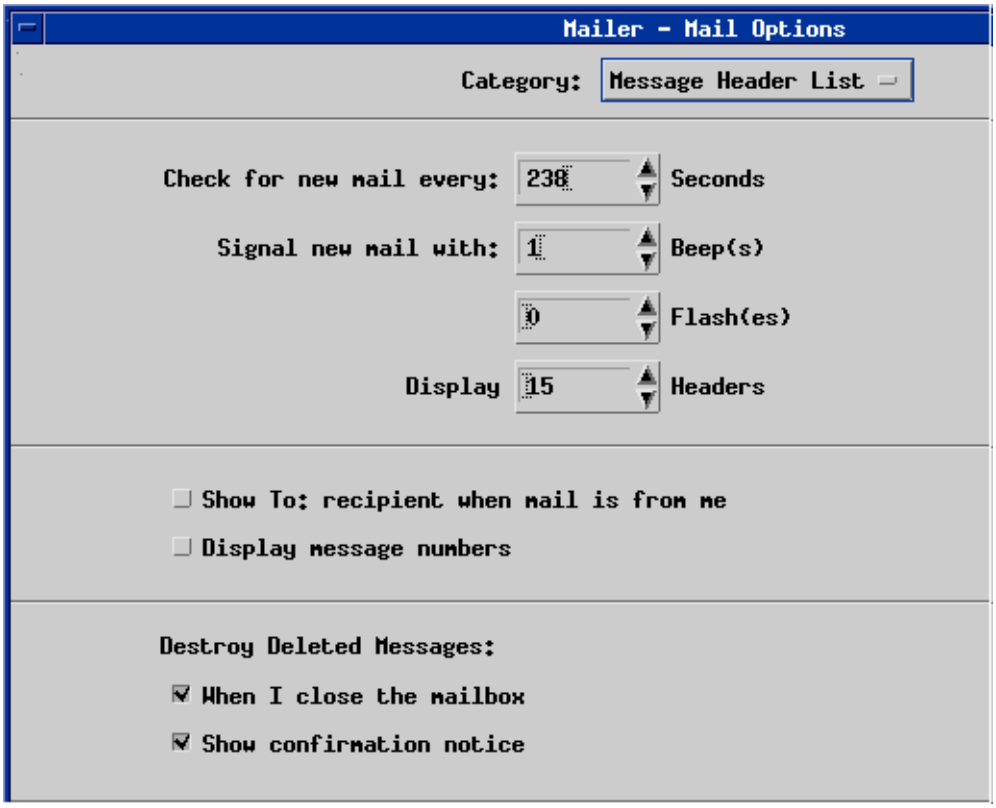
In Mailer's Help Table of Contents click:

To Create an Alias

Customizing Mailer

You can change the behavior of the Mailer application by choosing Mail Options on the Mailbox menu. Mail Options are grouped into categories.

Figure 11-4: Mailer - Mail Options



By clicking the Category pull-down list, you can alter the following Option categories.

Table 11-4: Option Categories

Option	Allows you to:
Message Header list	Set how often Mailer checks for mail, how to signal when new mail arrives, how to display message headers and when to delete messages

Table 11-4: Option Categories

Option	Allows you to:
Message view	Change the number of display lines that appear in the message view, the number of characters on a line, and the fields to be hidden when displaying a message
Compose window	Set how attachments are shown, the string used to indicate a reply message line, the path used for the Dead Letter Folder and how to add custom header fields to the Format menu
Message filing	Specify path and file for storing mail files, the mailboxes that appear in the Move and Copy To menus, the path where Mailer starts looking for mailboxes and the number of recently visited mailboxes to display
Vacation	Specify a vacation message and to turn on the Vacation feature
Templates	Add template file containing frequently used text
Aliases	Create distribution lists
Advanced items	Set defaults for confirmation notices, MIME character encoding, mail file locking, ignoring host name in addresses, and use of local names

You can customize when to destroy a deleted message as well as:

- How often Mailer checks for receipt of new mail
- How many beeps to signal that new mail has arrived
- How many headers to display
- Whether or not to display message numbers
- An automatic reply (vacation mail)

Now Try This!

Create a message to be used as an automatic reply when you receive mail during your vacation. Follow these steps.

1. Select Mail Options from the Mailbox menu.
2. Select Vacation from Category pull-down menu on the Mail Options dialog box.
3. On the Vacation dialog box, click Vacation.
4. In the Subject Input text box, you can choose to change the subject.
5. Type your vacation message in the message area.
6. Click Apply.
7. Click OK.

Need Help?

In Mailer's Table of Contents, click:

To Set Mailer Properties

Obtaining User Information (CLI)

Overview

The following CLI commands allow you obtain user information.

- `who`
- `users`
- `finger`

Finding Out Who is Logged In

The `who` command shows you:

- Who is logged in
- Where users are logged in

The `who` command has three forms.

- `who am i`
  - — Displays information only about the user logged in at the terminal where the command is entered. Use this command to determine your initial login name.

```
$ who am i
smith         pts/5       May 14 16:05            (:0.0)
```

- `whoami`
  - — Examines the system file `/etc/passwd` and then displays the current user's login name based on that file. Use this command after the `su` command to change your current user name. Remember to use `who am i` to determine who you really are.

```
$ whoami
smith
$
```

- `who` [*options*]
  - — The `who` command examines the system file `/var/adm/utmp` to obtain user information.

```
$who
smith         console     May 14 11:50
smith         :0          May 14 11:50
smith         pts/1       May 14 11:50
smith         pts/2       May 14 14:56
smith         pts/3       May 14 16:01
$
```

Now Try This!

To observe the difference in output from the three versions of the `who` command:

1. Enter the `who` command to see who is currently logged in to the system.
2. Enter the `who am i` command to determine your initial login name and terminal ID.
3. Enter the `whoami` command to see your current login name.

Need Help?

Read the reference page for the `finger`(1) command.

Solution

1. Issue the `who` command to see who is currently logged in to the system.

```
$ who
mcflower    pts/0        May 14 10:11
evant       pts/1        May 14 11:25
```

2. Issue the `who am i` command to determine your initial login name and terminal ID.

```
$ who am i
evant       pts/1        May 14 11:25
```

3. Issue the `whoami` command to see your current login name.

```
$ whoami
evant
```

Obtaining a Quick User List

The `users` command displays a one-line list of the login names of all the current users.

```
$ users
anderson
johnston
smith
thomas
$
```

Now Try This!

To see how many users are logged in to your system, enter the `users` command.

Solution

```
$ users
evant
mcflower
```

Obtaining More Information About Users

The `finger` command displays a detailed list of users including:

- Their real names
- Elapsed time since they last entered a command
- Personal informationThe following table shows some useful options to the `finger` command

Table 11-5: Some finger Command Options

Option	Function
`finger`	Lists users
`finger -i`	Produces a quick listing with idle times
`finger -q`	Produces a quick listing, including only login name, terminal name, and login time
`finger -w`	Forces narrow, short format listing

Now Try This!

To find out the full names of people using your system, enter:

```
$ finger
```

To find out more information about a specific user, enter:

```
$ finger username
```

Need Help?

See `finger`(1) for other useful options.

Solution

```
$ finger
Login       Name              TTY Idle    When            Office
evt       Evan Tyler        co   2d Thu 11:50  ZKO1-2/D53  884-1765
joe       Joe User          *st     Thu 16:17  RKO-2D/54   555-8765
joe       Joe User          pt      Thu 16:18  RKO-2D/54   555-8765
$ finger joe
Login name: joe         (messages off)  In real life: Joe User
Office: RKO-2D/54, 555-8765             Home phone: 555-4213
Directory: /usr/users/joe               Shell: /bin/sh
On since May 14 16:17:46
   on loner:0 from loner:0
On since May 14 16:18:35             2 minutes 59 seconds Idle Time
   on pts/4 from loner:0.0
No Plan.
$
```

Introducing the Mail Utility (CLI)

Overview

The CLI `mailx` and `Mail` commands let you:

- Compose, edit and send messages
- Receive and read messages
- Delete or store messages in a mailbox or in folders

Sending Mail

The formats of the `mailx` and `Mail` commands are:

```
mailx [-dinvF] [-h number] [-r address] [-s subject] user ...
Mail  [-dinvF] [-h number] [-r address] [-s subject] user ...
```

The following mail session shows how to send a message to three system users: `jar`, `kls`, and `sjg` and a carbon copy to `ske`.

Example 11-3: Sending Mail

```
$ mailx 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>
$
```

Now Try This!

To send a mail message to yourself:

1. At the system prompt ($), enter: **`mailx`** *`username`* Return, using your username.
2. Mail then prompts with Subject:. Enter the subject of your message and press Return.
3. Mail is now waiting for you to enter your message. Enter the body of your message, then press Return to move the cursor to the beginning of the next line. Then press Ctrl/D to end the message.
4. If Mail prompts: Cc:, enter usernames (separated with commas or spaces) to whom carbon copies/additional copies should be sent and press Return.

5. The system then prompts: $. This indicates you have successfully sent your mail message.

Editing a Message

When you are entering the body of your mail message, the line-oriented editor allows you to enter each line of the message. You then press Return to get a new line. To edit a line, you must make changes before entering Return. There are more efficient ways to create and edit a message, including:

- Creating a file before using mail
- Invoking an editor while in mail
- Using ~r to send a file

Creating a File Before Mailing

To send a message using an existing file:

- Use a text editor to create and edit a file entering the information you want to mail. Save the message to a file.
- Use the `Mail` command to send the file to a distribution list.

The following example sends the document, `letter.mail`, to users `dan`, `kls`, `ar` and `rln`. The message file, `letter.mail`, was created with a text editor before entering the `mail` command. Notice the redirection symbol (`<`). This indicates to the mail program to take input from the file, `letter.mail`.

Example 11-4: Using Mail to Send a File

```
$ mailx dan kls ar rln < letter.mail
```

Invoking an Editor in Mail

To invoke an editor from within mail, enter `~v` or `~e` on a line by itself.

The following example shows how to create a message using the tilde escape sequence, `~v`. When you enter `~v` on a line by itself, `mail` invokes the editor defined by the shell environment variable, `VISUAL`.

Example 11-5: Editing a Message with ~v

```
$ mailx 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.
```

The following table describes the tilde escape sequences you can use to invoke an editor.

Table 11-6: Selecting Your Editor in Mail

Tilde Escape Sequence	Description
~e	Uses the editor defined by the **EDITOR** shell environment variable. If **EDITOR** is not defined, the default editor is the `ed` line editor.
~v	Uses the editor defined by the **VISUAL** shell environment variable. If **VISUAL** is not defined, the default is `vi`.

Now Try This!

To edit a message by invoking a text editor in mail:

1. Enter the `mail` command and provide a subject.
2. Invoke an editor by typing a `~e` or `~v` tilde escape sequence at the beginning of a blank line.
3. Exit the editor in the usual manner.

Need Help?

If the editor you expect does not run, check the value of the VISUAL and/or EDITOR shell environment variables.

Using ~r to Send a Named File

If the file already exists, it can be read into the message buffer with `~r`, as shown in the following example.

Example 11-6: Using ~r to Mail a File

```
$ mailx msg
Subject: Here are the prices! <Return>
~r pricelist.txt <Return>
   .
   .
   file contents are displayed
   .
   .
<Ctrl/D>
Cc: sjk <Return>
$
```

Tilde Escape Sequences

Other commonly used tilde escapes are shown in the following table.

Table 11-7: Tilde Escape Functions

Option	Function
`~b`	Adds username(s) to the list of people to username(s) receive blind copies of the message
`~c`	Adds username(s) to the list of carbon copy username(s) 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`	Adds usernames to the list of direct username recipient
`~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

This example shows the use of some tilde escape sequences.

Example 11-7: Using Tilde Escapes When Sending Mail

```
$ mailx jar kls sjg
❶~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?
❷~t pnh
❸~c ard
❹~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>
$
```

❶ The `~s` provides the string for the subject field

❷ The `~t` adds `pnh` to the recipient list.

❸ The ~c adds ard to the Cc list.

❹ The ~p prints the entire message including header information.

Now Try This!

To send a message using the Tilde Escape Sequences to add a subject field and additional recipients:

1. At the system prompt, enter the mailx command with a user ID.
2. In the body of the message on a line by itself, enter ~s and a text string for the subject.
3. On another line by itself, enter ~t and a list of users to add additional recipients.
4. On another line by itself, enter ~p to view the headers and body of the message.
5. Enter Ctrl/D to send the message.

Need Help?

Refer to the section, Sending Mail, in mailx(1).

Solution

```
$mailx msg
~s The equipment has arrived!
~t evt tlb
~p
-------
Message contains:
To: msg evt tlb
Subject: The equipment has arrived!

(continue)
Let Tom know when he can install the drives and the modem!
.
EOT
```

Reading Mail

To read your mail, type the command Mail or mailx and press Return, as shown in this example.

Example 11-8: Mail Menu

```
$ mailx
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 "Do not 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.

Mail uses these symbols when displaying a list of mail messages.

Table 11-8: Mail Symbols

Symbol	Meaning
?	Mail prompt
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 > prompt will point to the first new message on the list

To read a particular mail message, use the commands in this table.

Table 11-9: Reading Mail Commands

Command	Result
<Return>	**Current** message is displayed
p,t	**Current** message is displayed or redisplayed
n, +	**Next** message is displayed
-	**Previous** message is displayed
number	Selected ***numbered*** message is displayed

Sending Mail Commands

The following table contains most of the commands you will need to send mail.

Table 11-10:Sending Mail Commands

Function	Command	Result
Send a message	m *user_list*	Sends mail to all users named in the message list user_list. Can be used with the ~f escape to forward messages. The ~f escape reads the contents of the specified file into the message you are sending.
	file#	Returns you to your mailbox file.
Save a message	RET, s, w	Message is saved in the file mbox in your home directory.
Reply to a message	R	Allows you to reply to the user who sent you the message. The subject field of the message contains the subject of the message you are responding to, preceded by RE: You may use the tilde escapes just as you would when sending mail.
	r	Same as R, but sends a response to all recipients of the original message as well as the sender.

Deleting Mail

You can delete messages and retrieve those messages deleted by mistake using the `d` and `u` commands, as shown in the following table.

Table 11-11: Delete and Undelete Commands

Command	Function
`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
`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 may be undeleted

Exit and Help Commands

Mail provides the following commands to obtain help and exit the mail program.

Table 11-12: Exiting and Help Commands

Function	Command	Result
Help	`?, help`	Displays a list of commands along with a summary of their meanings
Exit mail	`q`	Mail is exited; all changes made to your mailbox (for example, deletions) are final
	`x, ex`	Mail is exited and you are returned to the $ prompt; your mailbox is left as it was at the beginning of the mail session
List mail variables	`set`	Lists mail variables

Customizing Mail (CLI)

Overview

You can customize your mail session by:

- Using a `.mailrc` file to override mail's initial configuration
- Creating folders to organize saved mail
- Using an alias to simplify mail distribution

System Mail File /usr/share/lib/mail.rc

The `/usr/share/lib/mail.rc` file:

- Contains commands that set the initial mail configuration and are applicable to all users
- Is executed by the mail utility before it turns control over to you
- Is maintained by the system administrator

Individual users can alter or augment the settings by using a `$HOME/.mailrc` file.

Creating a .mailrc File

The `.mailrc` file:

- Must be located in your home directory
- Contains mail `set`, `unset`, `alias`, and `ignore` commands

If the `.mailrc` file is not present when you log in for the first time, create it using an editor or the `cat` command.

The following `.mailrc` file contains two set commands.

- `set ask`, which tells mail to prompt you for a subject line whenever you send a message
- `set askcc`, which prompts you for carbon copies when you finish typing a message

Example 11-9: Creating a .mailrc File

```
$ cat > .mailrc
set ask
set askcc
<Ctrl/D>
$
```

Customizing Your Mail Option File

Two types of options can be set: binary and valued.

Table 11-13: Binary Options (on or off)

Option	Function
`[un]set ask`	Causes `mailx` to prompt you for the subject of each message you send
`[un]set askbb`	Causes `mailx` to prompt you for email address of people to receive a blind carbon copy
`[un]set askcc`	Causes `mailx` to prompt you for Email addresses of people to receive copies of the message
`[un]set metoo`	Causes the sender to be included in the alias expansion and therefore sender also receives message
`[un]set hold`	Causes message to be held in the system mailbox by default

Table 11-14: Valued Options (*option=value*)

Option	Function
`set crt=`*`number`*	Causes paging program to be invoked from messages that exceed number of lines
`set EDITOR=`*`pathname`*	Defines text editor invoked by the `~e` and `edit` subcommands
`set folder=`*`pathname`*	Defines the name of directory to use for storing folders for messages
`set VISUAL=`*`pathname`*	Specifies pathname of a text editor to use in the `visual` and `~v` subcommands

Solution

1. **`$ vi .profile`**

```
i
MAIL=/var/spool/mail/$USER
:wq
".profile" 20 lines, 340 characters
```

2. **`$ cat /usr/share/lib/Mail.rc`**

```
set append dot save
ignore Received Message-Id Resent-Message-Id Status Mail-From
Return-Path Via
$
```

3. **$ vi .mailrc**

```
i
set ask askcc
set cdenotooltalklock='f'
alias UandC evt@loner.dec.com
set folder=mail_folder
:wq
".mailrc" 8 lines, 205 characters
```

Folders

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 called **folders** for organizing groups of messages together.

All mail folders are stored in the user directory specified with the `mailx set folder` option. You can have one or more folders. The first time you save a message to a folder, `mailx` creates a folder file under the directory you created for this purpose. Thereafter, each time you save a message to the folder, `mailx` appends the message to the folder file.

Now Try This!

To create a folder:

1. Enter the `mkdir` command at the system prompt to create a new directory to store your mail folders. Give it a name you like.

   $mkdir *folder_directory*

2. Edit your `.mailrc` file and enter the following line:

   `set folder=`*`folder_directory`*

   where *folder_directory* is the name you gave in step 1.

3. Enter the `mailx` command and select a new message.

4. Use the `mailx save` command to save the message and the folder.

   `save 1 +`*`folder_name`*

   where *`folder_name`* is a name that you supply.

   Mail will create the new folder with the supplied name and save the message in the folder.

Need Help?

Look at the examples in `mailx`(1).

Solution

1. **$ mkdir evt_mail**

2. **$ vi .mailrc**

```
i
set folder=evt_mail
:wq
".mailrc" 8 lines, 201 characters
```

3. **$ mailx**

```
Mail $Revision: 4.2.4.2 $  Type ? for help.
"/usr/spool/mail/evt": 1 message 1 new
>N  1 evt   Mon Feb 23 13:09  11/385  "The equipment has arrived!"
```

4. **? save 1 +equipment**

```
"+equipment" [New file] 11/385
? q
```

Using Folders

You can use a folder name, preceded with a plus sign (+) anywhere a file name is expected with a `mailx` command. The following table shows various commands for working with folders.

Table 11-15: Working with Folders

To:	Enter at the ? Prompt
Select folder *folder_name*	`folder +folder_name`
Select the previous folder	`folder #`
Select the system mailbox	`folder %`
Select your personal mailbox	`folder &`
Determine the current folder and number of messages it contains	`folder`
Display the names of all folders	`folders`
Copy the current message to *folder_name*	`copy [msg_list] +folder_name`
Save the current message in *folder_name*	`save [msg_list] +folder_name`

Now Try This!

To read mail messages from a folder called `practice`:

1. Enter `mailx`, and save a mail message to a folder called `practice`. Mail will automatically create the folder.
2. Copy a mail message you have received to the folder called `practice`.
3. Leave `mail` using the `quit` command and reenter `mail` using the command:

```
$ mailx -f +practice
```

This command starts `mailx` with `practice` as the current folder.

4. Read the messages in your `practice` folder using the regular mail commands.

Need Help?

Look at the examples in `mailx`(1).

Solution

1. **$ mailx**

```
Message  1:
From evt Mon Feb 23 13:09:00 1998
Date: Mon, 23 Feb 1998 13:08:59 -0500
From: Evan Tyler <evt>
To: evt msg tlb
Subject: The equipment has arrived!

Let Tom know when he can install the drives and the modem!
```

2. **? save 1 +practice**

```
"+practice" [New file] 11/385
```

3. **?q**

4. **$ mailx -f +practice**

```
From evt Mon Feb 23 13:09:00 1998
Received: by loner.dec.com (5.65v3.2/1.1.10.5/23Jan97-0140AM)
        id AA07212; Mon, 23 Feb 1998 13:08:59 -0500
Date: Mon, 23 Feb 1998 13:08:59 -0500
From: Evan Tyler <evt>
Message-Id: <9802231808.AA07212@loner.dec.com>
To: evt msg tlb
Subject: The equipment has arrived!
Status: R

Let Tom know when he can install the drives and the modem!

? q
```

Using Aliases

An **alias** is a name that stands for one or more user names and provides a quick way to:

- Send mail to a distribution list.
- Abbreviate a user name that is complicated or hard to spell.

After creating the alias in your `.mailrc` file, you can use the alias instead of the user name when mailing messages.

Now Try This!

To define an alias, (called `project`) for members of your project team (sam, sally, steve and susan), do the following:

1. Edit your `.mailrc` file by adding the following line:

   ```
   alias project sam sally steve susan
   ```

2. Send mail to all the project members using the alias you just created.

   ```
   mailx project
   ```

Summary

Using Mailer (CDE)

CDE provides the Mailer application for you to communicate with other users on your network systems. Using Mailer, you can create, send and receive messages, work with attachments, manage your incoming mail and customize the Mailer application for your working environment.

Obtaining User Information (CLI)

The following commands display helpful user information.

`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

Introducing the Mail Utility (CLI)

The CLI `mailx` and `Mail` commands let you:

- Compose, edit and send message
- Receive and read messages
- Delete or store message in a mailbox or in folders

Customizing Mail (CLI)

You can customize your mail session by:

- Using a `.mailrc` file to override mail's initial configuration
- Creating folders to organize saved mail
- Using an alias to simplify mail distribution

Summary

12

Using TCP/IP Networks

Unit Overview

Introduction

This unit examines some of the commands that allow access to other systems using TCP/IP as the network transport.

TCP/IP stands for **T**ransmission **C**ontrol **P**rotocol/**I**nternet **P**rotocol - a suite of network protocols used by applications to provide network functions such as:

- Remote login
- File Transfer
- Name/address resolution
- Network file systems

Objectives

To access other systems, you should be able to:

- Log in and execute commands
- Display Motif applications from remote systems

Resources

For more information on the topics in this unit, see the following:

- UNIX reference 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 start up programs. This section covers the commands you can use to gain access to a machine other than your own. Once you have access to the other machine, you can enter commands just as you do on your local host system.

Network Security

For many network applications, the remote host allows 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.

Some applications such as `rlogin` will prompt for a password if none of the above is true, while others such as `rcp` will fail.

Logging In

You can use the `rlogin` command to log in to a remote system.

The format of the `rlogin` command is:

`rlogin` *`rhost`* `[-e`*`character`*`] [-8] [-l` *`username`*`]`

This table shows some of the options you can use with the `rlogin` command.

Table 12-1: 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 host are not standard, `rlogin` uses a 7-bit data path, and the 8th (high) bit of each byte is stripped.
-e*character*	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.

This example shows two uses of the `rlogin` command.

Example 12-1: Using the rlogin Command

```
$ rlogin sys001
Last login: Mon Mar  30 01:00:59 from sys999
Tru64 UNIX X5.0-14(Rev. 490);Mon Mar 30 03:09:17EST 1998

sys001$ whoami
joe
sys001$ exit
$ rlogin sys002 -l smith
Password:
Last login: Mon Mar  30 01:00:59 from sys999
Tru64 UNIX X5.0-14(Rev. 490);Mon Mar 30 03:09:17EST 1998

sys002$ whoami
smith
sys002$ <Ctrl/D>
$
```

Now Try This!

To perform the exercises in this unit, your system must be connected to a network of UNIX systems using TCP/IP. You must obtain a system name from your instructor/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. Enter the `hostname` and `whoami` commands.
3. Log out.

Need Help?

If you receive a password prompt, the local system does not have an entry in the remote `host.equiv` or `.rhosts` file. Create a `.rhosts` file in your remote home directory listing the local system.

Solution

```
$ rlogin sys002
Password:
Last login: Mon Mar  30 01:00:59 from you@sys001
Tru64 UNIX X5.0-14(Rev. 490);Mon Mar 30 03:09:17EST 1998
sys002$ hostname
sys002
sys002$ whoami
you
sys002$ <Ctrl/D>
$
```

Using the rsh Command

One way to enter commands on a remote system is to use the `rsh` (remote shell) command. The `rsh` command connects to the remote host and executes a specified command or, if no command is specified, performs a login on the remote host.

The same restrictions apply for the `/etc/hosts.equiv` and `$HOME/.rhosts` files as for the `rlogin` command.

The format of the `rsh` command is:

```
rsh [-dn] [-l user] remote_host [command] [argument ...]
```

This table shows some options for the `rsh` command.

Table 12-2: rsh 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

This example shows two uses of the `rsh` command.

Example 12-2: Using the rsh Command

```
$ rsh sys001 users
evant garcia
$
$ rsh sys002
Last login: Mon Mar  30 02:10:59 from sys999
Tru64 UNIX X5.0-14(Rev. 490);Mon Mar 30 03:09:17EST 1998

sys002$
```

Now Try This!

Enter the `rsh` command to list the `/bin` directory on the remote system.

Need Help?

If you get the message, "Permission denied", create the `.rhosts` file on the remote system.

Solution

```
$ rsh sys002 ls /bin
 Mail
 Rsh
 X11
   .
   .
   .
```

Copying Files

One way to copy files between a local and a remote or two remote systems is to use the `rcp` command. As with the `rlogin` command, the name of the local system must be in the `/etc/hosts.equiv` file on the remote system, or the system name and user name must be in the `$HOME/.rhosts` file in the remote account.

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.

This table shows some options you can use with the `rcp` command.

Table 12-3: 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; the destination must be a directory

This example shows how to enter the `rcp` command to copy files from a remote system.

Example 12-3: Using the rcp Command

```
$ rcp  sys001:/usr/users/smith/file.txt file.txt

$ rcp  smith@sys001:file.txt file.txt

$ rcp  -r smith@sys001:/usr/users/smith/project  \
          smith@sys002:/production
```

Now Try This!

1. Enter the `rcp` command to copy the following file from your remote system to your user directory on your local system.

   `/usr/include/stdio.h`

2. Enter the `rcp` command again but preserve the modification times and modes of the source file.

Solution

1. `$ rcp sys002:/usr/include/stdio.h stdio.h`

```
$ ls -l stdio.h
-r--r--r--  1 evant      11739 May 30 14:30 stdio.h
```

2. `$ 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
```

File Transfer

A second method of transferring files between systems is to use the `ftp` application. `ftp:`

- Allows the transferring of files between hosts that use dissimilar file systems
- Does not attempt to preserve file attributes that are specific to a particular file system
- Does not allow recursive coping of subdirectories

Once you invoke the `ftp` application, you can enter other commands to accomplish tasks.

The format of the `ftp` command is:

`ftp [`*`options`*`] [`*`host`*`]`

If you do not specify a host, `ftp` prompts for a command. Refer to `ftp`(1) for a list of options. Some `ftp` commands are shown in this table.

Table 12-4: Some ftp Commands

Command	Function
`open rsys`	Connects to remote system `rsys`
`close`	Closes the current connection
`get`	Receives a file
`put`	Sends a file
`ls`	Lists contents of remote directory
`quit` **or** `Ctrl/D`	Exits `ftp`
`?` **or** help	Displays list of `ftp` commands

Refer to `ftp`(1) for a complete list of commands.

This example shows how to use the `ftp` command to log in to a remote system.

Example 12-4: Using ftp

```
$ ftp
ftp> open sys001
Connected to sys001
220 sys001 FTP server (Tru64 UNIX 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
$
```

Now Try This!

Using the `ftp` command, list the files in the remote system's `/usr/include` directory and then copy the file `/usr/include/cpio.h` from the remote system to your user directory on the local system.

Need Help?

You need an account with a password on the remote system. Enter the `ftp` help command to get a list of commands.

Solution

```
$ ftp
ftp> open sys002
Connected to sys002.
220 sys002 FTP server (Tru64 UNIX Version 5.60) 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>
```

Telnet

Another way to connect to a remote system is to use the `telnet` application. Once you connect to the host and log in, you receive a system prompt and may execute any of the available commands on the host system.

The format of the `telnet` command is:

`telnet [`*host*`[`*port*`]]`

If you do not specify a host, `telnet` prompts for a command. This table shows some `telnet` commands.

Table 12-5: Some telnet Commands

Command	Function
`open rsys`	Connects to remote system `rsys`
`close`	Closes current connection, or allows you to log out from remote system
`quit` **or** `Ctrl/D`	Exits `telnet`
`?`	Displays list of `telnet` commands

This example shows how to use `telnet` to connect to a remote system.

Example 12-5: Entering telnet Commands

```
$ telnet
telnet> open sys002
Trying 16.30.0.2...
Connected to sys002.
Escape character is '^]'.
Tru64 UNIX (sys002) (ttyp3)
login: you
Password:
Last login: Mon Mar  30 01:00:59 from sys999
Tru64 UNIX X5.0-14(Rev. 490);Mon Mar 30 03:09:17EST 1998
$ hostname
sys002
$ exit
Connection closed by foreign host.
$
```

Now Try This!

1. Start `telnet`.
2. Use the `?` command to list `telnet` commands.
3. Log into a remote system.
4. Use the escape character (Ctrl/]) to return to `telnet`.
5. Use the `status` command to print status.
6. Close the connection and quit `telnet`.

Solution

```
gertie> telnet
telnet> ?
Commands may be abbreviated.  Commands are:
close       close current connection
display     display operating parameters
mode        try to enter line-by-line or character-at-a-time mode
open        connect to a site
quit        exit telnet
send        transmit special characters ('send ?' for more)
set         set operating parameters ('set ?' for more)
unset       unset operating parameters ('unset ?' for more)
status      print status information
toggle      toggle operating parameters ('toggle ?' for more)
slc         change state of special characters ('slc ?' for more)
z           suspend telnet
!           invoke a subshell
environ     change environment variables ('environ ?' for more)
?           print help information
telnet> open dove
login: joe
Password:
Last login: Mon Jun 29 15:08:03 from gertie
dove> <Ctrl/]>
TELNET> status
Session  1 Active  Host gertie Port 23
    Operating Mode: Character-at-a-time
    Escape character: '^]'
    Options:
        Echo - Remote
        Terminal Type - Local
        Terminal Type - VT200
        Suppress Go Ahead - Local
        Suppress Go Ahead - Remote
    Terminal Dataoveruns:     0
    Suspended Network I/Os:   0
TELNET> close
%TELNET-S-LCLCLOSED, Local connection closed
-TELNET-I-SESSION, Session 01, host gertie, port 23
TELNET> quit
$
```

Invoking Remote Motif Applications

Overview

Sometimes you may want to execute a Motif application on another system, but display the output on your workstation screen. 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 that the `/etc/hosts.equiv` and `$HOME/.rhosts` files provide, there is an additional security check made before a remote Motif application can display its output on your workstation.

You must grant permission on the local system (server) to receive display from a remote system (client).

- To see the list of authorized display clients, enter the command `/usr/bin/X11/xhost` on the local system. The output will show which remote systems have permission to display output on the local system.
- To add display clients, use the `xhost` command with the host or user names you want to display output.

Specifying the Display

Many Motif applications provide a `-d` or `-display` option to specify where the application should display output. If you do not specify this option, the application uses the value of the `DISPLAY` environment variable.

To set and export the display with the KornShell, 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 Motif version on your system must be compatible with the remote system's version.

Invoking Applications

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.

Tru64 UNIX locates many of the Motif applications in the `/usr/bin/X11` directory. If this directory is not in your command search path, you must use the full pathname.

Either set and export your `DISPLAY` variable, or use the display option of the application to redirect the display to your system. 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.

This example shows how to run a remote Motif application with the display output directed to your local system.

- First by specifying the display system on the command line with the `-d` option
- Second by using the `DISPLAY` environment variable to specify the display system

Example 12-6: Invoking Motif Applications

```
$ rlogin sys002 -l you
Password:
Last login: Mon Mar  30 12:12:20 from sys999
        . . .
sys002$ /usr/bin/X11/dxbook -d sys999:0.0 &
[1]     5731
sys002$ export DISPLAY=sys999:0.0
sys002$ /usr/bin/X11/xclock
```

Now Try This!

1. Using the `rlogin` or `telnet` command, log in to your remote system.
2. Use the `whereis` command to get the full path for `xclock`, the clock application. Invoke it in the background, using the display option to direct the display to your workstation.

Solution

1.

```
$ rlogin sys002
Password:
Last login: Mon Mar  30 01:00:59 from you@sys001
Tru64 UNIX X5.0-14(Rev. 490);Mon Mar 30 03:09:17EST 1998
```

2.

```
sys002$ whereis xclock
xclock: /usr/bin/X11/xclock /usr/share/man/man1/xclock.1X
sys002$ /usr/bin/X11/xclock -display sys001:0.0 &
[1]     9573
sys002$
```

Invoking with rsh

To invoke only one application on the remote system, you can use the `rsh` command.

```
$ rsh sys002 -l you  /usr/bin/X11/xclock -d sys999:0
```

Now Try This!

Enter the `rsh` command to invoke `xclock` on your remote system.

Need Help?

If you get a message stating:

```
Client is not authorized to connect to Server:  Can not open display
```

use the `xhosts` command on your local system to grant permission to the remote system.

Solution

```
$ rsh sys002 -l you  /usr/bin/X11/xclock -d sys001:0
$
```

Summary

Accessing Remote Systems

Table 12-6: Remote Access Commands

Command	Function
`rlogin`	Logs in to a remote system
`rsh`	Enters 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 `DISPLAY` environment variable.
- You can either `rlogin` to the remote system and invoke the application, or use `rsh` to invoke the application.

13

Printing Using CDE

Unit Overview

Introduction

In spite of the advances in electronic transmission storage, and information retrieval, 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 use CDE to:

- Determine and change the default printer
- Print files and set print file options
- Use printing applications to perform various printer functions

Resources

For more information on the topics in this unit, see the following:

- *Common Desktop Environment: User's Guide*, Chapter 9

Setting Your Default Printer

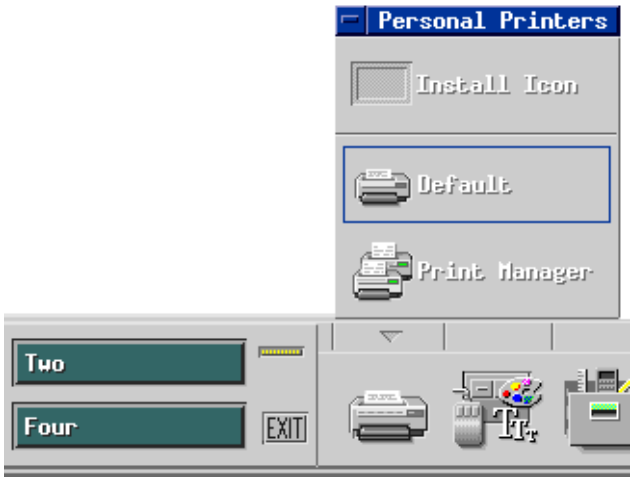
Overview

Your system may have access to several printers and provide many printing options. You might have more than one type of printer (such as PostScript, PCL or plotter), which may be either directly connected to your system (local) or located somewhere on the network (remote). One printer is generally designated as your default printer.

Determining the Default Printer

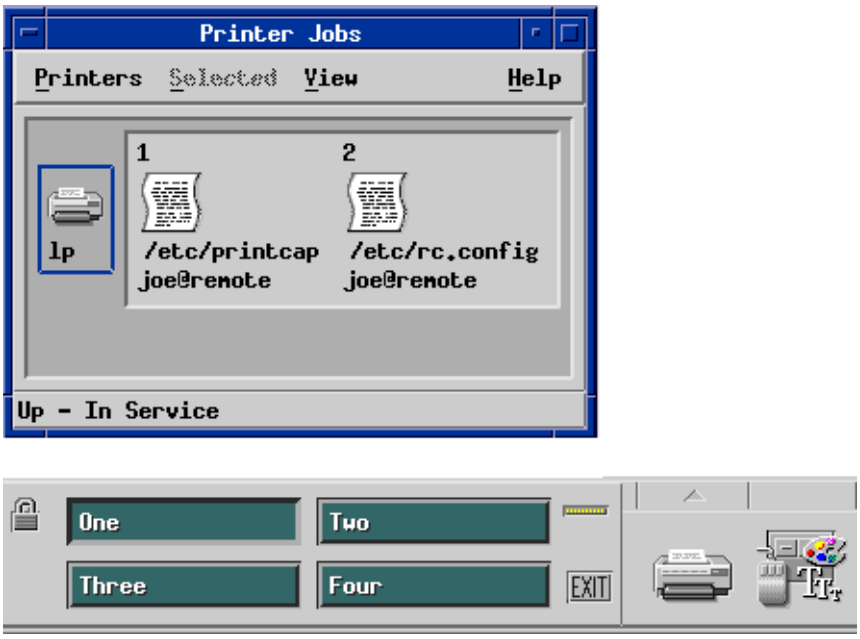
To determine your default printer, double-click the printer control on the Front Panel or the Default printer control in the Personal Printers subpanel. (See Figure 13-1)

Figure 13-1: Printer Control on Front Panel



Result: The Printer Jobs application starts up in another window showing information on your default printer.

Figure 13-2: Determining the Default Printer



Changing the Default Printer

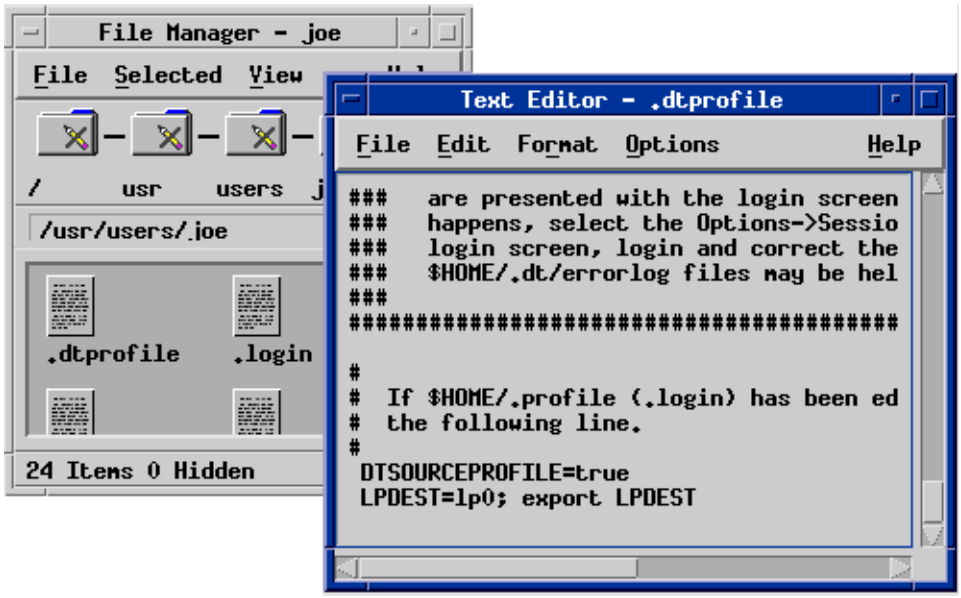
To change your default printer:

1. Using the File Manager, select your home directory.
2. Double-click the `.dtprofile` file. If you cannot see the file, change your File Manager View to show hidden files.
3. Add a line to set the value for the `LPDEST` environment variable to the printer you want to use as your default. Check the `/etc/printcap` file for possible printers.

   ```
   LPDEST=printer_name; export LPDEST
   ```

4. Log out of the system and then log back in. The change should be in effect.

Figure 13-3: Editing .dtprofile



Now Try This!

To change your default printer to another printer on your system:

1. Use the File Manager to select the `/etc` folder.
2. Double-click the `printcap` icon.
3. Select a printer to use as your default. Close the file.
4. Use the File Manager to select your home folder.
5. Open the `.dtprofile` file. Edit the value of the `LPDEST` environment variable to your new choice for the default printer.
6. Save the changes.
7. Log out, then log back in to have the change take effect.

Need Help?

Double-clicking a text file opens it in the Text Editor so you can view it or make changes.

Pick a printer listed in the `/etc/printcap` file and add it to the `.dtprofile` file.

Solution

Sample text line.

```
LPDEST=turky; export LPDEST
```

Printing Files

Overview

With CDE, you can print a file using the:

- Printer control on the Front Panel
- Print selection on the File Manager's Selected menu
- Print Manager
- Print option on application menus

File Types

CDE will print the following types of files.

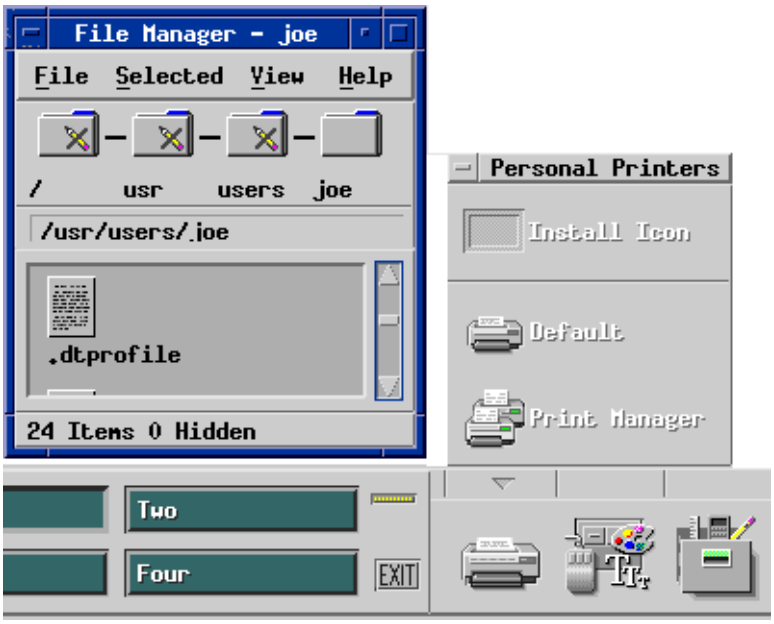
- Text
- PostScript
- PCL
- Data

The Printer Control

For a convenient way to print a file, follow these steps.

1. Drag one or more files from File Manager and drop them on the Printer control on the Front Panel or to a printer icon in the Personal Printers subpanel.

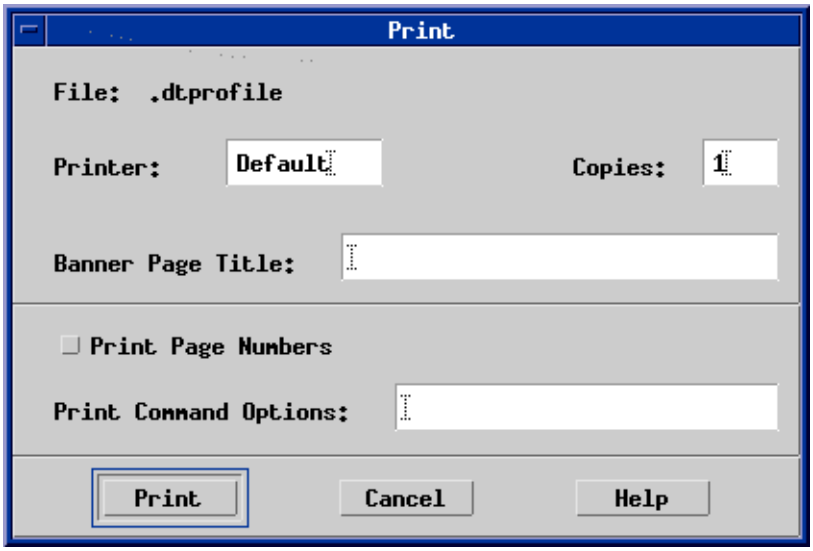
Figure 13-4: Using the Printer Control



Result: The system displays a Print dialog box for each file that you drag to the Printer control.

2. Set any of the following options on the Print dialog box for each file.

Figure 13-5: Setting Print Options



3. Click the Print button to send the job to the designated printer.

   You may supply additional print command options on the Print dialog box. See `lp`(1) for a list of options.

Printing Using the File Manager

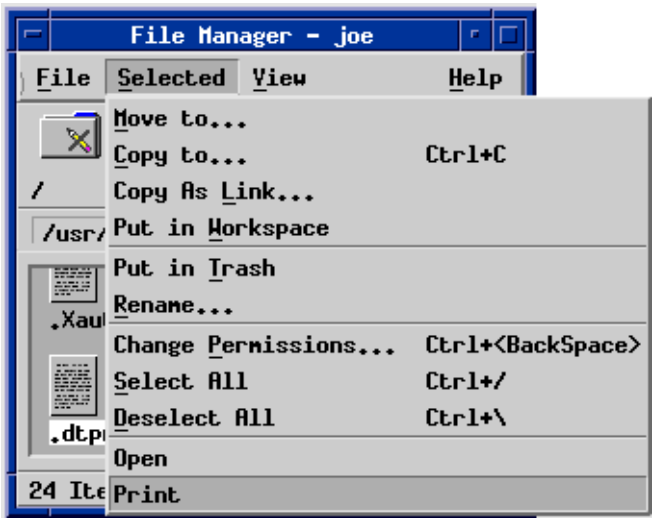
To print from the File Manager window:

1. Open a File Manager window.
2. Select a file to print.
3. Choose Print from the Selected menu or the file's pop-up menu.

   Result: The system displays a Print dialog box.

4. Set options on the Print dialog box such as the number of copies or the print command options.
5. Click the Print button to send the job to the designated printer.

Figure 13-6: Printing Using the File Manager



Printing with the Print Manager

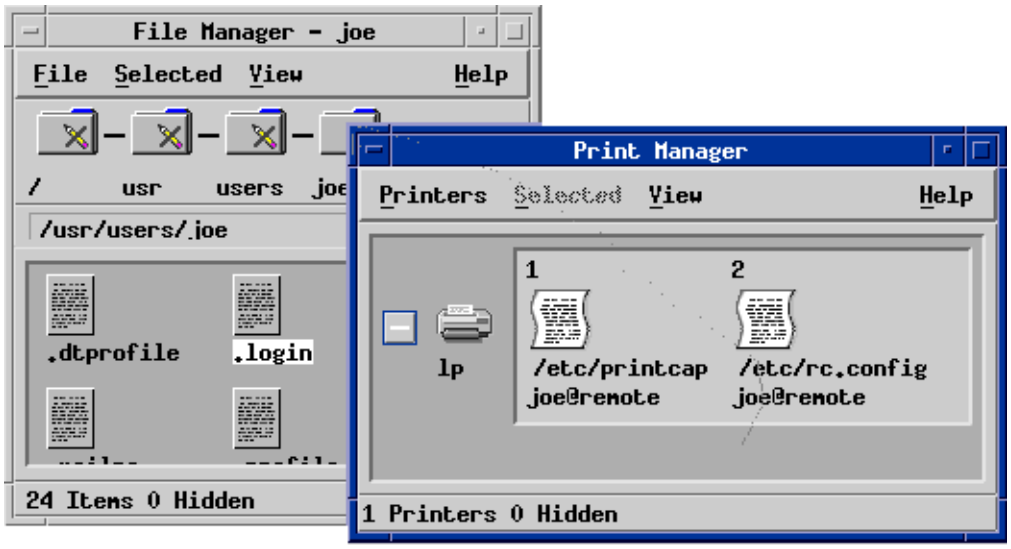
To print using the Print Manager:

1. Start the Print Manager from the Personal Printers subpanel.
2. Drag a file from the File Manager to a printer icon in the Print Manager window.

Result: The system displays a Print dialog box.

3. Set any of the following options on the Print dialog box for the file.
4. Click the Print button to send the job to the designated printer.

Figure 13-7: Printing with the Print Manager



Printing Within Applications

You can also print a file from within an application if the print function is supplied on one of the application's menus.

Now Try This!

Using the following steps, print two copies of your `.dtprofile`.

1. Using the File Manager, select your home directory.
2. Select the `.dtprofile` icon.
3. From the File Manager Selected menu, select Print.
4. On the Print dialog window, enter the printer name and the number of copies.
5. Click Print.

Need Help?

If you do not get a printout, check with your system administrator to be sure you have a printer configured correctly and the printer is working.

Managing Print Requests

There are two applications that display the progress of print requests.

- Print Manager
- Printer Jobs

The Print Manager displays the status of all printers known to your system. Printer Jobs displays the status of the default printer.

Using the Print Manager

You can use the Print Manager to:

- Check the status of a print job
- Delete a print job
- Find a particular print job
- Check the properties of a print job
- Change the Print Manager display characteristics

The Print Manager keeps track of all printers known to the system.

Checking the Status of a Print Job

To check the status of a print job:

1. Click the Print Manager control in the Personal Printer subpanel.

   Result: The system opens up the Print Manager window.

2. Open up the Printer icon by doing one of the following:

   — Clicking the Open [+] button to the left of the printer icon

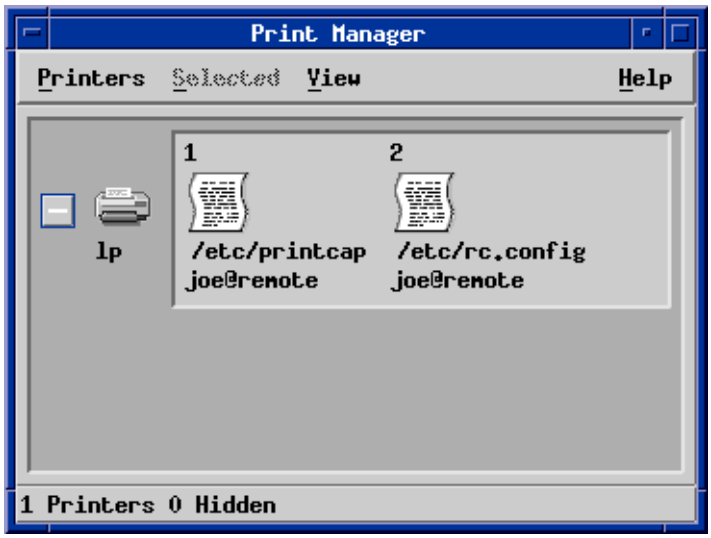
   — Double-clicking the printer icon

   — Selecting a printer, then choosing Open from the Selected menu or from the printer's pop-up menu (displayed by pressing Shift/F10 or MB3).

   Result: The Print Manager displays all print jobs waiting to be printed

   (See Figure 13-8)

Figure 13-8: Print Manager Status



Deleting a Print Job

To remove a print job:

1. Click the Print Manager control in the Personal Printer subpanel.

   Result: The system opens up the Print Manager window.

2. Select a print job.
3. Choose Cancel from the Selected menu or from the print job's pop-up menu.
4. Click Yes in the confirmation dialog box.

Changing the Print Manager Display

You can change how the Print Manager application:

- Displays printers and print jobs
- Updates the frequency of the display
- Displays printer jobs
- Shows printers
- Displays the status of printers
- Displays the Print Manager message line

Figure 13-9: Print Manager Display Options

Printer Jobs Application

The Printer Jobs application shows the status of print jobs for one printer that you opened from the Front Panel. The Printer Jobs functions behave in the same manner as they do in the Print Manager.

Displaying Print Job Properties

For various reasons, you may need to know the owner of a print job, its size or when it was submitted. You can obtain this information by displaying the properties of a print job using this procedure.

1. Click the Print Manager control in the Personal Printer subpanel.

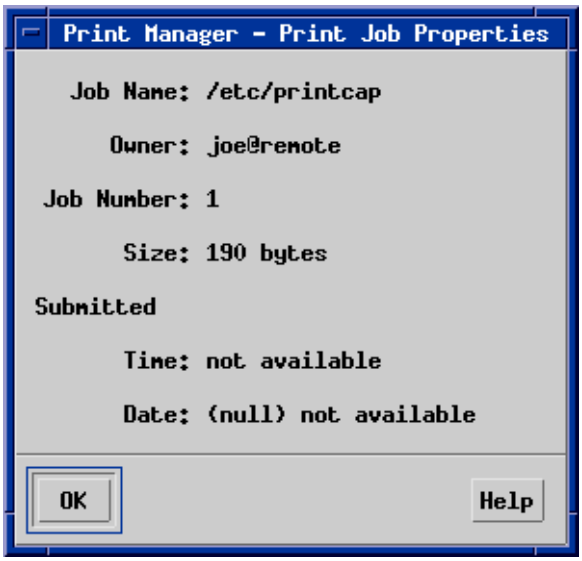
   Result: The system opens up the Print Manager window.

2. Select a print job.

3. Choose Properties from the Selected menu or from the print job's pop-up menu.

   Result: The Print Manager displays the following job properties.

Figure 13-10: Print Jobs Properties



Finding a Print Job

To locate a print job in Print Manager:

1. Choose Find from the Printer menu.
2. Type the name or a partial name of the print job you want to find in the Job Name field.
3. Click Start Find.
4. To go to the file, select the job and click Goto.
5. To cancel a found job, select the job and click Cancel Print Jobs.

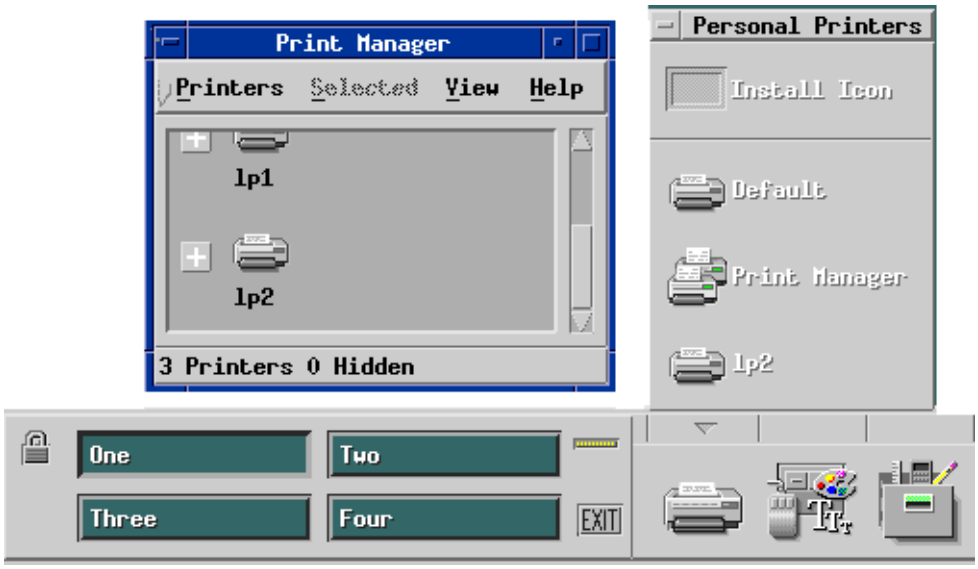
Adding Printers to the Front Panel

The printer associated with the Front Panel is generally your default printer. With CDE, you can change that printer to any printer displayed by the Print Manager. You can also add to your list of personal printers on the Personal Printers submenu.

To change the printer associated with the Front Panel:

1. Double-click the Print Manager located on the Personal Printers subpanel.
2. Verify that the printer you want is represented. If it is not, you must add the printer to the Personal Printers subpanel.
3. In the subpanel, point to the printer icon you want to add and select Put In Main Panel from the icon's pop-up menu (Shift/F10 or MB3).

Figure 13-11: Adding Printers to the Front Panel



Now Try This!

1. Using the Print Manager, display the print jobs for each of the printers shown.
2. Display the printer properties of one of the print jobs shown in the Print Manager display.

Solution

1. To display the print jobs for each printer:
   a. Click the Print Manager control in the Personal Printer subpanel.
   b. Double-click the printer icon.
2. To display printer properties of a print job:
   a. Click the Print Manager control in the Personal Printer subpanel.
   b. Select a print job.
   c. Choose Properties from the Selected menu.

Summary

Setting Your Default Printer

You can determine which printer is the default printer by double-clicking the printer control on the Front Panel. To change the default printer to another printer, edit the `.dtprofile` file in your home directory to modify the value of the LPDEST environment variable.

Printing Files

You can print a file using the:

- Printer control on the Front Panel
- Print selection on the File Manager's Selected menu
- Using the Print Manager
- Print option on application menus

Managing Print Requests

There are two applications that display the progress of print requests.

- Print Manager
- Printer Jobs

With the Print Manager you can:

- Check the status of a print job
- Delete a print job
- Find a particular print job
- Check the properties of a print job
- Change the Print Manager display characteristics

The Printer Jobs application provides the status of one printer.

14

Printing Using CLI

Unit Overview

Introduction

If you do not have CDE configured, or you prefer to use the command line interface, you need to know how to configure a printer, print files and manage your print jobs using commands.

Objectives

To work with printers, you should be able to use CLI to:

- Set up your default printer
- Print files
- Manage print requests

Resources

For more information on the topics in this unit, see the following:

- *Command and Shell User's Guide*, Chapter 3

Setting Your Default Printer

Overview

To set up your default printer, you need to:

- Determine the available printers.
- Set up the PRINTER environment variable.

Determining Available Printers

Before you can print any files, you must know what printers are available for your use. You can determine this by examining the printer description file, `/etc/printcap`. This is a partial example of an `/etc/printcap` file.

Example 14-1: Examining /etc/printcap

```
$ more /etc/printcap
# LN03 in the office area
lp0|lp|0|officepr:\
 :lp=:\
 :rm=sys01.zko.dec.com:\
 :rp=1:\
 :sd=/usr/spool/lpd:
   . . .
# 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:
   . . .
 $
```

Your system administrator maintains the `/etc/printcap` file and adds an entry for each local or remote printer available for use. An entry consists of several lines which describe a printer's characteristics. The first noncomment line of each entry lists the printer's name and its aliases separated by a vertical bar (|) and terminated with a colon (:).

You may use any one of the names to reference the printer in a print command. In the previous example, `lp0` is the name of one printer, `lp`, `0` and `officepr` are aliases for that printer.

PRINTER Environment Variable

The next step in setting up your default printer is to initialize the PRINTER environment variable.

If you are using the C shell, enter the following command at the system prompt:

```
% setenv PRINTER printer_name
```

where *printer_name* is the name or alias of the printer you want to use.

Or if you are using the Korn shell, enter:

```
$ PRINTER=printer_name
```

LPDEST Environment Variable

If you do not set the PRINTER environment variable, print commands use the printer specified by the LPDEST environment variable. To determine the value of this variable, enter the following command:

C shell:

```
% setenv
```

Korn shell:

```
$ set
```

Now Try This!

To set your default printer:

1. Use the more command to display the contents of your /etc/printcap file.
2. Choose a printer to use as your default printer.
3. Use the appropriate shell command to set the PRINTER environment variable to the printer you choose to be the default printer.

Need Help?

Read the reference pages on lp(1), lpr(1), csh(1), and ksh93(1CDE)

Solution

```
1. $ more /etc/printcap
# Created on Mon Jan 27 22:48:40 EST 1997 by root

2. turky|turky|lp|turky (PrintServer turky):\
   :af=/var/adm/turky.acct:\
   .
   .
   .
   chi|chi|chi (PrintServer chi):\
   :lp=/dev/null-chi:\
   .
   .
3. $ PRINTER=turky
```

Printing Files

Overview

With CLI, you have two print commands.

- `lp`
- `lpr`

Printing Files with lp

The `lp` command sends the specified files to a printer for printing.

The format of the `lp` command is:

`lp [-option] [-] [file]`

This table shows some of the options for the `lp` command.

Table 14-1: lp Command Options

Option	Function
`-`	Reads input from standard input
`-d`	Specifies the printer to do the printing; if `-d printer` is not specified, the default, in order is: • PRINTER environment variable • LPDEST environment variable
`-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

Now Try This!

To print a text file from your working directory:

1. Use your default printer which you created in the previous lesson.
2. Print two copies of the text file.
3. Give your print job a title.
4. Have the system send you mail when the file is printed.

Need Help?

If you did not receive a job number, your print command did not work. If you received an error message, fix the error. If you are still having problems, talk to your system administrator after reading the reference page on `lp`(1).

Solution

The command you enter should look similar to this:

```
$ lp -n2 -t"Program Example V1.0" -m example.txt
Job Number is:  215
```

Printing Files with lpr

The `lpr` command provides more printing options than the `lp` command including:

- One or two-sided printing
- Landscape or portrait orientation
- Multiple pages on a side of paper
- International support

The format of the `lpr` command is:

`lpr [-option] [file]`

This table shows some `lpr` options.

Table 14-2: lpr Command Options

Option	Function
`-`	Reads input from standard input
`-i[number]`	Indents the output by number of spaces; default number of spaces 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
`-Ksides`	Prints one-sided, two-sided, tumble, simplex or duplex
`-m`	Sends mail when the spooling is complete
`-Nnumber`	Prints one or more pages on one sheet of paper
`-Pprinter`	Prints to the specified printer
`-#number`	Produces the specified number of copies

Now Try This!

Using the `lpr` command, print a text file from your home directory with the following options:

1. Print the file to a printer other than your default printer.
2. Print with the banner MY JOB.
3. Have the text indented five spaces.

4. Have the mail program notify you when your print job is complete.

Need Help?

Read the reference page on `lpr`(1).

Solution

The command you enter should look similar to this:

```
$ lpr -Plp1 -J "MY JOB" -i5 -m examples.txt
```

Managing Print Requests

Overview

Once you print your file, you may want to check on the status of your print job. UNIX provides the following commands for checking print job status.

- `lpq`
- `lpstat`
- `lpc`

Checking Print Status with lpq

The `lpq` command reports the status of jobs in the print queues by:

- Printer
- User
- Request ID

The format of the `lpq` command is:

`lpq [+][`*`number`*`] [-l] [-P`*`printer`*`] [`*`request_id`*`...] [`*`user`*`...]`

The following table provides information about the use of arguments with the `lpq` command.

Table 14-3: lpq Command Options

Option	Function
`(No option)`	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
`-P`*`printer`*	Reports the status of print requests for the specified printer
`request_id`	Reports the status of the print job specified by *`request_id`*
`user`	Provides a report of all print jobs queued for user *`user`*

Using lpq

`lpq` provides the:

- Current rank in the queue
- Names of files comprising the request

- Identification number for the print request
- Total size in bytes

If the print request consists of more than one file, the number of file names shown in the `lpq` output depends upon the length of the file name.

This example shows the information you obtain with the `lpq` command.

Example 14-2: lpq Output

```
$ lpq

Rank   Owner    Job  Files                                 Total Size
active jones     15  pac-man                               1833 bytes
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
```

Now Try This!

Compare the output of the following two commands:

- `lpq`
- `lpq -l`

Need Help?

Read the reference page on the `lpq`(1) command.

Solution

```
$ lpq
Mon Mar  2 13:07:34 1998: Attempting to print dfA215chkade
Rank   Pri Owner      Job  Files                          Total Size
active 0   evt       215  vmstat.txt                     5936 bytes
1st    0   evt       216  wrl.txt                        1033 bytes
$ lpq -l
Mon Mar  2 13:07:36 1998: Attempting to print dfA215chkade
evt: active                                [job 215union]
        2  copies of vmstat.txt        2968 bytes

evt: 1st                                [job 216union]
        wrl.txt                        1033 bytes
$
```

Print Status with lpstat

The `lpstat` command provides more information than the `lpq` command.

The format of the `lpstat` command is:

```
lpstat [options]
```

The following table lists some `lpstat` options.

Table 14-4: 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`*`]`	Displays print requests for all printers or for specified printers
`-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

This example shows the information output by the options of the `lpstat` command.

Example 14-3: lpstat Examples

```
sys001> lpstat -d     ❶
System default destination: serial
sys001> lpstat -r     ❷
Scheduler is running
sys001> lpstat -s     ❸
System default destination: serial
Scheduler is running
Output for printer lp is sent to remote printer lp0 on sys002
Output for printer lp1 is sent to remote printer lp5 on sys002
Output for printer serial is sent to /dev/tty01
Output for printer parallel is sent to /dev/lp0
sys001> lpstat -t     ❹
System default destination: serial
Scheduler is running
Output for printer lp is sent to remote printer lp0 on sys002
Output for printer lp1 is sent to remote printer lp5 on sys002
Output for printer serial is sent to /dev/tty01
Output for printer parallel is sent to /dev/lp0
lp:
   printer is on remote host sys002 with name lp0
   queuing is enabled
   printing is enabled
   no entries
   no daemon present
lp1:
   printer is on remote host sys002 with name lp5
```

```
   queuing is enabled
   printing is enabled
   no entries
   no daemon present
serial:
   printer is on device '/dev/tty01' speed 4800
   queuing is enabled
   printing is enabled
   no entries
   no daemon present
parallel:
   printer is on device '/dev/lp0' speed -1
   queuing is enabled
   printing is enabled
   no entries
   no daemon present
Requests on lp:
sys001: Mon Feb 23 17:13:48 1998:
Warning: no daemon present
Rank   Owner      Job  Files                          Total Size
1st    joe          3  /etc/printcap                   294 bytes
2nd    Kathy        4  /tmp/CMpaafBba                 8320 bytes
3rd    Kathy        5  (standard input)                207 bytes
4th    joe          6  /usr/users/joe/restore.txt      966 bytes
5th    joe          7  /usr/users/joe/syslog.txt       739 bytes
6th    evt          8  /tmp/CMpaalpua                15271 bytes
7th    Kathy1       1  (standard input)                109 bytes
8th    Cindy       12  /tmp/CMpaadwia                 12282 bytes
9th    evt         15  /usr/users/evt/ksh.txt        112517 bytes
10th   evt        230   dtterm.txt                     41158 bytes
Requests on lp1:
sys001: Mon Feb 23 17:01:18 1998:
no entries

Requests on serial:

no entries

Requests on parallel:

no entries

sys001> lpstat -v     ❺
Output for printer lp is sent to remote printer lp0 on sys002
Output for printer lp1 is sent to remote printer lp5 on sys002
Output for printer serial is sent to /dev/tty01
Output for printer parallel is sent to /dev/lp0
sys001>
```

❶ The `-d` option provides the name of the default system printer. In this case, the default printer name is `serial`. If you use the `cat` command on the `/etc/printcap` file, you will see an entry for a printer with the default printer name.

❷ The `-r` option provides the status of the printer daemon. In this example, the daemon is running.

❸ The `-s` option provides a list of printers in addition to the default printer and the status of the daemon. Notice that the output for the `lp` and `lp1` printers goes to a printer on a remote system.

❹ The `-t` option provides status information for all printers. Notice that queuing and printing are enabled for all printers, however, no daemon is present for any of the printers. No daemon is present because there are no jobs.

❺ To determine the printers available on the system, use the `lpstat -v` option. The output from this command also shows if a printer is local or remote. The `serial` printer in this example is a local printer (one directly connected to the system). `lp`, `lp1`, `parallel` are remote printers (located somewhere on the network).

Printer Control Program (lpc)

The `lpc` command is normally used by the system administrator to control the operation of the printers on the system. However, users can issue the `lpc status` and `restart` commands. Use these two commands to check the status of your print job and restart the printer daemons.

The format of the `lpc` command is:

```
/usr/sbin/lpc [command [argument] ]
```

If you enter `/usr/sbin/lpc` without arguments, `lpc` prompts for commands from standard input. If you supply arguments, `lpc` interprets the first argument as the command to be executed, and the remaining arguments as parameters to the command.

lpc User Commands

This table lists some `lpc` commands available to users.

Table 14-5: lpc User Commands

Command	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 caused the daemon to quit, leaving jobs in the queue, the `restart` command attempts 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.

Now Try This!

Compare the output of the `lpstat -t` and `lpc status` commands.

Solution

```
$ lpstat -t
system default destination: turky
Scheduler is running
Output for printer turky is sent to /dev/null-turky
Output for printer chi is sent to /dev/null-chi
turky:
   printer is on device '/dev/null-turky' speed -1
   queuing is enabled
   printing is enabled
   2 entries in spool area
   Mon Mar  2 13:07:34 1998: Attempting to print dfA215sys001.dec.com
chi:
   printer is on device '/dev/null-chi' speed -1
   queuing is enabled
   printing is enabled
   no entries
   no daemon present
Mon Mar  2 13:07:34 1998: Attempting to print dfA215sys001.dec.com
Rank   Pri Owner      Job  Files                                 Total Size
active 0   evt        215  vmstat.txt                           5936 bytes
1st    0   evt        216  wrl.txt                              1033 bytes
Mon Mar 2 13:45:53 1998:
no entries
System default destination: turky
Scheduler is running
Output for printer turky is sent to /dev/null-turky
Output for printer chi is sent to /dev/null-chi
Requests on turky:

Requests on chi:

$ lpc status
turky:
   printer is on device '/dev/null-turky' speed -1
   queuing is enabled
   printing is enabled
   2 entries in spool area
   Mon Mar  2 13:07:34 1998: Attempting to print dfA215sys001.dec.com
chi:
   printer is on device '/dev/null-chi' speed -1
   queuing is enabled
   printing is enabled
   no entries
   no daemon present
```

Deleting Print Requests

Overview

Use either of the following commands to delete a print request:

- `lprm`
- `cancel`

Deleting a Print Job with lprm

You can remove print requests from the print queue with the `lprm` command.

The format of the `lprm` command is:

`lprm [options]`

The following table lists some options to the `lprm` command

Table 14-6: lprm Options

Option	Function
(No option)	Removes from the queue the current active request for the user on the default printer if no options are specified
-P*printer*	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 users; for the superuser, the spool queue is completely emptied
request_ id	Removes the print job with 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 following example shows how to remove job 49 from the default printer queue.

Example 14-4: Deleting a Print Job with lprm

```
$ lprm 49
```

Deleting a Print Job with cancel

The `cancel` command can also remove print jobs spooled by the `lp` or `lpr` commands.

The format of the `cancel` command is:

`cancel [request_id] [printer]`

The *request_id* option cancels only the print job with that job number. The *printer* option cancels all print jobs for the selected printer for the user. If the superuser issues the `cancel` command with the printer option, all requests for that printer are removed from the queue.

Now Try This!

1. Spool a large file to a printer.
2. Use the `lpq` command to find the job number.
3. Use the `cancel` command to cancel the job.

Solution

Sample solution:

```
$ lpr -Pforpub core
$ lpq -Pforpub
Thu Jul  2 10:55:37 1998: Attempting to print dfA002gertie
Rank   Pri Owner   Job  Files                 Total Size
active 0   msg     2    core                1212416 bytes
$ cancel 2 forpub
dfA002gertie dequeued
cfA002gertie dequeued
```

Summary

Setting Up Your Default Printer

You can determine available printers by displaying the contents of the `/etc/printcap` file.

Use the `PRINTER` environment variable to establish your default printer.

Printing Files

You can use the `lp` or `lpr` commands to print files. The `lpr` command provides more printing options including:

- One or two-sided printing
- Landscape or portrait orientation
- Multiple pages on a side of paper
- International support

Managing Print Requests

You can obtain printer status information with the `lpq` and `lpstat` commands. You delete print jobs using the `lprm` or `cancel` command.

Use the printer control program, `lpc` to restart printer daemons and display printer status information.

Deleting Print Requests

To delete a print job, you can use the `lprm` or `cancel` commands.

15

Using Disks and Tapes

Unit Overview

Introduction

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.

Device special files and device naming conventions allow you to identify specific devices when entering commands.

The `df` and `du` commands provide detail about disk space usage. The `tar`, `ltf`, `dd`, and `mt` commands are used for tape data transfer and conversion.

The primary emphasis is on using the `tar` command to create backup tapes.

Objectives

To save and restore disk files and file systems on magnetic tape, you should be able to:

- Identify device types and names
- Use the `df` and `du` commands to display free disk space
- Use the Tape Archive utility

Resources

For more information on the topics in this chapter, see the following:

- Tru64 UNIX reference pages, `du`(1), `df`(1), `tar`(1) and `tar`(4)

Identifying Devices

Overview

This topic introduces concepts to help you manage disk, tape and CD-ROM devices.

Device Special Files

UNIX treats all devices (console, mouse, keyboard, disks, tape drives, printers, terminals pseudoterminals) as special files. The device special files are read and written like ordinary files, except they activate the I/O device itself and are generally located in the directory `/dev`.

Examples of device special files are shown in the table.

Table 15-1: Device Special Files

File	Description
`/dev/ttyp1`	Terminal
`/dev/lp0`	Printer
`/dev/disk/dsk3g`	Third Disk found when the system was started, partition **g**, block I/O
`/dev/rdisk/dsk3g`	Third Disk found when the system was started, partition **g**, character I/O
`/dev/tape/tape0_d0`	FirstTtape unit found, highest supported density
`/dev/rz0a`	SCSI disk, logical unit `0`, partition **a**
`/dev/rmt0h`	Tape drive, logical unit `0`, high density

Disk and type drives have two types of special device files.

- Block
  - — Reads and writes to the device one block of bytes at a time (usually 512 bytes)
- Character
  - — Reads and writes to the device one character at a time

Device Names (Prior to V5.0)

Prior to Version 5.0, Tru64 UNIX systems used the device mnemonics listed in this table to name disk and tape devices.

Table 15-2: Tru64 UNIX Tape, Disk, CD-ROM Device Mnemonics

Category	Mnemonic	Description
Disks	rz*nn*	SCSI disks (RZ22/RZ23/RZ24/RZ55/RZ56/RZ57/RZ58)
CD-ROM	rz*nn*	SCSI disks (RRD40)
Tapes	tz*nn*	SCSI tapes (TLZ04/TLZ05/TZ30/TZK50)

The format of a device name is:

`[r]`*`device-mnemonic lun [attribute]`*

The table details the device name elements.

Table 15-3: Device Name Elements

Element	Description
`r`	Indicates a character device special file; not used for block special files or tape devices
`device-mnemonic`	Disks
`lun`	CD-ROM
`attribute`	Tapes

Device Names (V5.0)

Tru64 UNIX Version 5.0, introduces a new format for device names and new directory locations for the device special files. The new names remove all notion of device type and physical location from the device name. They now reflect the order in which the device is found during the installation process.

The disk device name format is:

`deviceup`

where: *`device`* is `dsk` or `cdrom`

`u` is the device unit number

`p` is the partition letter

For example, the device special file for the third disk found at installation might have the pathname:

`/dev/disk/dsk3g`

A character special file for this disk would have the following pathname:

`/dev/rdisk/dsk3g`

The tape device name format is:

`deviceu_dn`

where: *`device`* is tape

`u` is logical unit number

`dn` is the tape density supported

For example, the device special file for the first tape drive found during installation might have the pathname:

`/dev/tape/tape0_d0` (rewind tape device - highest density supported)

The no rewind special device file for this tape drive might have the pathname:

`/dev/ntape/tape0_d0`

Both versions of the device names can co-exist on the system.

Now Try This!

List the system devices in the `/dev` directory. The output from this command may be long. You might want to pipe the output through the `more` command.

Solution

```
$  ls /dev
$MAKEDEV     ptypb        rrz11e       rrz6c        rz1a         rz9g
MAKEDEV.log  ptypc        rrz11f       rrz6d        rz1b         rz9h
SYSV_PTY     ptypd        rrz11g       rrz6e        rz1c         sad
audit        ptype        rrz11h       rrz6f        rz1d         snmpinfo
binlogdmb    ptypf        rrz12a       rrz6g        rz1e         streams
cam          ptyq0        rrz12b       rrz6h        rz1f         tape
console      ptyq1        rrz12c       rrz9a        rz1g         tty
disk         ptyq2        rrz12d       rrz9b        rz1h         tty00
kbinlog      ptyq3        rrz12e       rrz9c        rz2a         tty01
kcon         ptyq4        rrz12f       rrz9d        rz2b         ttyp0
kevm         ptyq5        rrz12g       rrz9e        rz2c         ttyp1
kevmpterm    ptyq6        rrz12h       rrz9f        rz2d         ttyp2
keyboard0    ptyq7        rrz1a        rrz9g        rz2e         ttyp3
klog         ptyq8        rrz1b        rrz9h        rz2f         ttyp4
kmem         ptyq9        rrz1c        rz0a         rz2g         ttyp5
lockdev      ptyqa        rrz1d        rz0b         rz2h         ttyp6
log          ptyqb        rrz1e        rz0c         rz3a         ttyp7
lp0          ptyqc        rrz1f        rz0d         rz3b         ttyp8
mem          ptyqd        rrz1g        rz0e         rz3c         ttyp9
mouse0       ptyqe        rrz1h        rz0f         rz3d         ttypa
none         ptyqf        rrz2a        rz0g         rz3e         ttypb
nrmt0a       rdisk        rrz2b        rz0h         rz3f         ttypc
nrmt0h       rmt0a        rrz2c        rz10a        rz3g         ttypd
```

```
nrmt0l      rmt0h       rrz2d       rz10b       rz3h        ttype
nrmt0m      rmt0l       rrz2e       rz10c       rz4a        ttypf
ntape       rmt0m       rrz2f       rz10d       rz4b        ttyq0
null        rrz0a       rrz2g       rz10e       rz4c        ttyq1
pfcntr      rrz0b       rrz2h       rz10f       rz4d        ttyq2
pipe        rrz0c       rrz3a       rz10g       rz4e        ttyq3
prf         rrz0d       rrz3b       rz10h       rz4f        ttyq4
printer     rrz0e       rrz3c       rz11a       rz4g        ttyq5
ptm         rrz0f       rrz3d       rz11b       rz4h        ttyq6
ptmx        rrz0g       rrz3e       rz11c       rz6a        ttyq7
ptmx_bsd    rrz0h       rrz3f       rz11d       rz6b        ttyq8
pts         rrz10a      rrz3g       rz11e       rz6c        ttyq9
ptyp0       rrz10b      rrz3h       rz11f       rz6d        ttyqa
ptyp1       rrz10c      rrz4a       rz11g       rz6e        ttyqb
ptyp2       rrz10d      rrz4b       rz11h       rz6f        ttyqc
ptyp3       rrz10e      rrz4c       rz12a       rz6g        ttyqd
ptyp4       rrz10f      rrz4d       rz12b       rz6h        ttyqe
ptyp5       rrz10g      rrz4e       rz12c       rz9a        ttyqf
ptyp6       rrz10h      rrz4f       rz12d       rz9b        ws0
ptyp7       rrz11a      rrz4g       rz12e       rz9c        zero
ptyp8       rrz11b      rrz4h       rz12f       rz9d
ptyp9       rrz11c      rrz6a       rz12g       rz9e
ptypa       rrz11d      rrz6b       rz12h       rz9f
```

Now Try This!

List the system devices in the `/dev/disk` directory. The output from this command may be long. You might want to pipe the output through the `more` command.

Solution

```
$ ls /dev
$drom0a   dsk1b     dsk2e     dsk3h     dsk5c     dsk6f     dsk8a     dsk9d
cdrom0c   dsk1c     dsk2f     dsk4a     dsk5d     dsk6g     dsk8b     dsk9e
dsk0a     dsk1d     dsk2g     dsk4b     dsk5e     dsk6h     dsk8c     dsk9f
dsk0b     dsk1e     dsk2h     dsk4c     dsk5f     dsk7a     dsk8d     dsk9g
dsk0c     dsk1f     dsk3a     dsk4d     dsk5g     dsk7b     dsk8e     dsk9h
dsk0d     dsk1g     dsk3b     dsk4e     dsk5h     dsk7c     dsk8f     floppy0a
dsk0e     dsk1h     dsk3c     dsk4f     dsk6a     dsk7d     dsk8g     floppy0c
dsk0f     dsk2a     dsk3d     dsk4g     dsk6b     dsk7e     dsk8h
dsk0g     dsk2b     dsk3e     dsk4h     dsk6c     dsk7f     dsk9a
dsk0h     dsk2c     dsk3f     dsk5a     dsk6d     dsk7g     dsk9b
dsk1a     dsk2d     dsk3g     dsk5b     dsk6e     dsk7h     dsk9c
```

Using Disks

Overview

This topic introduces two commands that display information on disk use.

`df` Displays free disk space on mounted file systems

`du` Displays the disk space used by files and directories

Using the df Command

Use the `df` command to display the mounted file systems and determine the amount of free disk space.

The format for the `df` command is:

`df [-option] [file ...] [file system ...]`

The following table shows some commonly used `df` options.

Table 15-4: Commonly Used 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 CD-ROM 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)

This example shows the type of data provided by the `df` command.

Example 15-1: Using the df Command

```
$ df
   ❶                    ❷         ❸       ❹       ❺       ❻
Filesystem         512-blocks    used    avail capacity Mounted on
/dev/rz0a           38750  25140   9734   72%  /
/dev/rz0g          549328 450738  43656   91%  /usr
/users@sys02         581660 487244  36250   93%  /usr/users
/public@sys01      808254 586026 141404   80%  /public
$ df -k  ❼
Filesystem   1024-blocks    used  avail capacity Mounted on
```

```
/dev/rz0a          19375   12570   4867    72%   /
/dev/rz0g         274664  225369  21828    91%   /usr
/users@sys02      290830  243623  18124    93%   /usr/users
/public@sys01      404127  293013  70702    80%   /sys01/public

$ df -t ufs ❽
Filesystem  512-blocks   used  avail capacity Mounted on
/dev/rz0a        38750  25140   9734    72%   /
/dev/rz0g       549328 450738  43656    91%   /usr
$
```

❶ Displays the complete fileset name of the file system.

❷ Displays the total disk space in the file system in either 512K byte blocks or 1024K byte blocks. The total disk space is calculated as: used + free + reserved. Reserved space is for Metadata. If fileset quotas are imposed, this field represents the limit set by the fileset quota.

❸ Displays the total space used by the file system.

❹ Displays the total amount of unused space available to the fileset. If fileset quotas are established, this value represents space remaining until the quota limit is reached.

❺ How full the fileset is; represented as a percentage.

Capacity = used/used+available. This value can be greater than 100% if a domain has multiple filesets. See `df`(1).

❻ Displays the directory on which the file system is mounted.

❼ Outputs space information in kilobytes.

❽ Displays information for a specific file system type. `ufs` is the UNIX File System, `advfs` is the Advanced File System, `cdfs` is the Compact Disk Read-only Memory File System, `nfs` is the Network File System. See `df`(1) for more file system types.

Now Try This!

Enter the `df` command to display a list of free disk space on all mounted file systems.

Need Help?

See `df`(1) for other useful options

Solution

Your output should look similar to this:

```
$ df
  Filesystem    512-blocks    Used   Avail Capacity  Mounted on
  /dev/rz1a         126478   76900   36930    68%    /
  /dev/rz1g         424076  198060  183608    52%    /var
```

. . .

Using the du Command

The du command displays the size of files, in number of blocks, in all directories (listed recursively) specified. You can give commands to disk devices using the du command. The format for the du command is:

```
du [-option] [file ...] [directory ...]
```

This table lists some commonly used du options.

Table 15-5: Commonly Use du Options

Option	Function
-a	Displays amount of disk usage for each file; default is to not report on files
-l	Allocates blocks evenly among the links in file with multiple links
-r	Displays an error message when an inaccessible directory or file is encountered
-s	Displays a summary total only

The following example shows how to use the du command.

Example 15-2: Using the du Command

```
$ cd /usr/users
$ du -s guest     ❶
5 guest
$ du guest        ❷
1 guest/bin
5 guest
$ du -a guest     ❸
1 guest/bin
1 guest/.profile
1 guest/.login
1 guest/.cshrc
5 guest
$
```

❶ The total amount of disk usage by `/usr/users/guest` is 5 kilobytes.

❷ Displays the number of blocks in the `guest` directory and each subdirectory.

❸ Displays the number of blocks for each file in the `guest` directory and each subdirectory.

Now Try This!

Using the du command, display a list of the number of blocks used for each file in `/usr/users`.

Solution

Your output should look similar to this:

```
$ du -a /usr/users
    . . .
  4       /usr/users/you/.cshrc
  4       /usr/users/you/.login
  4       /usr/users/you/.profile
  224     /usr/users/you/uandc-files.tar
  238     /usr/users/you
  268904  /usr/users
```

Now Try This!

Using the du command, display a summary list of the total number of blocks used in /usr/users.

Solution

Your output should look similar to this:

```
$ du -s /usr/users
268904  /usr/users
```

Using the Tape Archive Utility

Overview

The `tar` utility is most commonly used to back up files from disk to tape, and then restore the files from the tape back to disk.

Tape Density

Typical device special files for tape drives with their corresponding densities are:

- `/dev/rmt0l`: drive 0, low density
- `/dev/rmt0m`: drive 0, medium density

`/dev/rmt0h`: drive 0, high densityFor most reel-to-reel tape drives, the default density is 6250 bpi (high density). To select low density, 1600 bpi, the device special file for low density must be selected.

- For TK50 tape cartridge drives, there is only one density - 6666 bpi, whether you select the device special file for high or low density.

Tape Archive Utility

The `tar` (tape archive) utility saves and restores multiple files to and from a magnetic tape. `tar` 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 for the `tar` command is:

```
tar [key_option] [options] name ...
```

`name` is either a file or directory. If *`name`* is a directory, `tar` recursively copies files and subdirectories in that directory.

This table shows some commonly used `tar` options. One key option must appear as the first command line argument. Additional options can be combined with the key option.

Table 15-6: Commonly Used tar Options

Key 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

Table 15-6: Commonly Used tar Options (Continued)

Key Option	Additional Option	Function
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

Writing Files to Tape

To write files to tape:

1. 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 to create a new tape or add files to the tape.
5. After the tape stops, rewind and unload it.
6. Label the tape and write protect it. Store the tape in a safe place.

Keep in mind that:

- tar writes data sequentially until it recognizes the end of tape or the end of information.
- When adding files, use the u option to prevent duplicating existing files.

See tar(1) and tar(4) for more information.

Tape Guidelines

The following guidelines are important to remember:

- Be sure to label (using a paper label) private tapes with the following information:
  - — Contents
  - — Dates recorded

  - — Tape density
  - — Variation of `tar` command used

- Write-access permission to the tape device is required to write to a tape. Read-access permission is required for users to extract files from a tape.
- To prevent accidental tape reuse, provide blank or recyclable tapes and write-protect rings in a storage area near the tape drive.
- All TK tapes are considered hardware write-protected by TK70 drives, and therefore are read-only media when used in a TK70 drive.

Extracting Files

To extract files from tape archives:

1. 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 files read from the tape.
4. Enter the `tar x` command to extract the file or files from the tape.
5. After the tape stops, rewind and unload it.
6. Use the `ls` command as a final check to verify that the file or files are in your directory.

Backup Operations

This example uses `tar` to create and list files from tapes.

Example 15-3: Using tar for Disk-to-Tape Operations

```
$ tar cv file_name          ❶
$ tar t                     ❷
$ tar tv                    ❸
$ tar ruv file_name         ❹
$ tar cvf file.tar file_name ❺
```

❶ `tar` creates a new tape using the files designated by `file_name`. The names of the files written to tape are displayed on standard output. If any `file_name` is a directory, all files in that directory and subdirectories are written to tape.

❷ The `t` option lists the names of the files on the tape to the terminal.

❸ The `tv` option provides extra information about the tape entries other than just the name.

❹ The `ruv` option appends `file_name` 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 `file_name`. As with tape creation, if `file_name` is dot (`.`) or asterisk (`*`), all files in the current directory and subdirectories are written to tape. If `file_name` is one file name or a list of file names, only those files specified are written to tape.

Restore Operations

This example uses `tar` to restore an archive to disk. The `tar` archive can be copied to other systems and extracted just as if they were being read from a tape.

Example 15-4: Using tar for Disk-to-Disk Operations

```
$ tar -tvf file.tar           ❶
$ tar -xvf file.tar file_name ❷
$ tar -xv file_name           ❸
$ compress file.tar           ❹
$ uncompress file.tar.Z       ❺
```

❶ Displays a verbose listing of the files contained in the `tar` archive, `file.tar`.

❷ Extracts the files represented by `file_name` from the `tar` archive, `file.tar`. If `file_name` is not provided, the entire contents of the tape is extracted. If `file_name` matches a directory, the directory is recursively extracted.

❸ The `xv` option causes the extraction of files from the archive. If `file_name` is not provided, the entire contents of the tape is extracted. If `file_name` matches a directory whose contents were written onto the tape, this directory is recursively extracted.

❹ `compress` reduces the size of the `tar` archive. This is especially useful when `tar` archive must be copied over a network. The new archive has the same file name as the `tar` file with `.Z` appended to the name (`file.tar.Z`).

❺ `uncompress` converts the compressed `tar` file to the uncompressed version after copying a compressed version of the file over a network. The created file has the same file name without the `.Z` (`file.tar`).

Now Try This!

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 `tar` command. The `c` option forces `tar` to start writing at the beginning of the tape. The `v` option asks `tar` to display the files it is working on.
2. Enter: `$ tar -t` to verify the file is actually on tape.
3. Set the tape to the load point again. Create a subdirectory named `tardir` and change directory to it.
4. Extract the file you just placed on tape by using the `x` option of `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 is there.

Solution

In this sample solution, we archive the file `stuff`.

1. `$ tar -cv stuff`

   ```
   a stuff 14 Blocks
   ```
2. `$ tar -t`

   ```
   stuff
   ```
3. `$ mkdir tardir`

   `$ cd tardir`
4. `$ tar -xv`

   ```
   blocksize = 20
   x stuff, 6849 bytes, 14 tape block
   ```
5. `$ ls`

   ```
   stuff
   ```

Summary

Identifying Devices

UNIX treats all devices (console, mouse, keyboard, disks, tape drives, printers, terminals pseudoterminals) as special files. The device special files are read and written like ordinary files, except they activate the I/O device itself and are generally located in the directory `/dev`.

Disk and type drives have two types of special device files.

- Block
  - — Reads and writes to the device one block of bytes at a time (usually 512 bytes)
- Character
  - — Reads and writes to the device one character at a time

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.

Using the Tape Archive Utility

The `tar` utility is most commonly used to back up files from disk to tape, and then restore the files from the tape back to disk.

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`.

16

Recalling and Editing Korn Shell Commands

Unit Overview

Introduction

The Korn shell allows the storing, viewing, and reexecution of previously executed shell commands. Each command that is issued can be stored in a history file.

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 feature saves you time because it allows you to reuse commands instead of retyping them.

The Korn shell also allows you to edit commands, either on the command line or in a temporary buffer.

Objectives

To efficiently execute Korn shell commands, you should be able to:

- Define the use of variables associated with command recall
- Recall commands from command history
- Edit commands in a temporary buffer using the `emacs` or the `vi` editor

Resources

For more information on the topics in this unit, see the following:

- *Common Desktop Environment: User's Guide*
- *The KornShell Command and Programming Language*
  Morris I. Bolsky and David I. Korn; published by Prentice Hall, Englewood Cliffs, New Jersey 07632 (c) 1989 ISBN 0-13-516972-0

Korn Shell Command History File

Overview

Every time you execute a command on the Korn shell command line, the Korn shell automatically stores a copy of the command in a history file. The history file contains a record of all the commands you have invoked.

Commands stored in the history file are available to you any time during the session. The history file is maintained when you log out, so previously executed commands are available the next time you log in.

- The history file is created by default
- You can modify:
  - — Number of recallable commands
  - — Name of the history file

Korn Shell Command History Variables

The following table shows variables you can use to alter the number of commands you may recall, or change the name of the history file.

Table 16-1: Korn Shell History Buffer and Command Recall Variables

Variable	Description
`HISTSIZE=`*`n`*	*`n`* determines the number of 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 to change it.

If you have multiple sessions running on one system, you should create 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 are archived in one history file, `$HOME/.sh_history`. The following example illustrates commands you can place in the `.profile` file to set the number of recallable commands to 96, an arbitrary number, and to set the `HISTFILE` variable to provide a separate history file for each session running the Korn shell.

Example 16-1: 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, edit in a temporary buffer, or recall and reexecute the latest `HISTSIZE` commands.

To display the command history buffer, use the Korn shell built-in command `fc` (fix command). The `fc` command has several formats. In this topic we discuss the first format, used to view previously executed Korn shell commands.

The syntax of the first format of the `fc` command is:

```
fc [-lnr] first last
```

The command options are shown in the table.

`-l`	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`, used to display the last 16 commands.
`-n`	When used with the `-l` option, causes commands to be displayed without command numbers.
`-r`	When used with the `-l` option, causes commands to be displayed in reverse order.
`first` and *`last`*	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

The following example uses the `fc` command to view previously executed Korn shell commands from the history file.

Example 16-2: Viewing Korn Shell Commands

```
$ fc -l 32 36  ❶
32 ls -al
33 more tempfile
34 cd /bin
35 ls -al k
36 echo $myname
$ fc -ln 32 36  ❷
   ls -al
   more tempfile
   cd /bin
   ls -al k
   echo $myname
$ fc -lr 32 36  ❸
36 echo $myname
```

```
35 ls -al k
34 cd /bin
33 more tempfile
32 ls -al
$ fc -l ls cd  ❹
35 ls -al k
34 cd /bin
$ history ls cd  ❺
35 ls -al k
34 cd /bin
$
```

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 next `cd` command

❺ Displays the command lines from the latest `ls` to the next `cd` command, but this time using the `history` command, the Korn shell built-in alias for `fc -l`

Recalling Previously Executed Korn Shell Commands

Previously executed Korn shell commands can be recalled and reexecuted, or modified and reexecuted. The second format of the `fc` command is used to recall and modify a previously executed command.

The syntax of 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, just type `r`.

The following example shows a short history of commands.

Example 16-3: history Command Output

```
$ history 49 53
49 $ date
50 $ first="Charlie"
51 $ mi="M."
52 $ last="Brown"
53 $ print $first $mi $last
```

The next example shows some samples of recalling and modifying commands.

Example 16-4: Recalling Commands from the History File

```
$ fc -e - Charlie=Susan 50     ❶
first="Susan"
$ fc -e - M=L mi      ❷
mi="L"
$ r Brown=Smith last     ❸
last="Smith"
$ r print       ❹
print $first $mi $last
Susan L. Smith
$
```

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 that started with `last` and substitutes `Smith` for `Brown`; uses the default alias, `r`

❹ Reexecutes, without modifications, the last `print` command

Now Try This!

1. Korn Shell Command History Variables

   a. By default, the Korn shell retains the last ___ commands in the history buffer. The default history file is ___.

   b. Why must this file be occasionally deleted?

   c. To control the number of commands you can access in the history buffer as well as the name of the history file, what variables would you set?

      Note: Do **not** set either of these variables at this time, as some of the following exercise steps would not work properly.

2. Korn Shell Command History Set Up

   a. Edit your `.profile` and add `PS1='! $ '`, if you have not already done so. In addition, execute the command on the command line. It will provide you with the command number.

   b. Execute the following sequence of commands. They will set up a known sequence for the following steps. Some commands are chosen at random just to add something to the history file.

```
$ more .profile
$ date
$ set
$ stty -a
$ name="your name"          provide your name
$ address="some address"  provide your address here
$ echo $name
```

3. Korn Shell `fc -l` Function

   Use the Korn shell built-in function `fc` (fix command) with the format `fc [-lnr] first last` to execute each of the following commands and answer any questions.

   a. `$ fc -l` — How many commands are displayed?

   b. `$ history` — How does this display compare with the `fc -l` display?

   From the series of commands we executed, pick one as *n* for the next few steps.

   c. `$ fc -l n n+4` — Display five commands from the history file.

   d. `$ fc -lr n n+4` — Display the same five commands, but in reverse order.

   e. `$ fc -ln n n+4` — Display the same five commands, but without the command numbers.

   f. `$ fc -l set echo` — Display the commands between the latest `set` and `echo` commands.

Exercises 3c through 3f used `fc -l`. You could also use `history` instead of `fc -l`. For example, you could enter `$history set echo` to display the commands between `set` and `echo`.

4. Korn Shell `fc -e` Function

   Use the Korn shell built-in `fc` function with the format: `fc -e -[old=new] [string]` to execute the following commands and answer any questions.

   a. `$ history` — Note the command numbers for future reference.

   b. `$ fc -e - name=address echo` — Repeat the last `echo $name` command, substituting `address` for `name`.

   c. `$ 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. `$ 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. `$ r SHELL=HISTFILE` — Repeat the previous command, substituting `HISTFILE` for `SHELL`. Why does the `r` command do the same thing as `fc -e -`?

   f. Repeat the last `more` command, substituting `.login` for `.profile`.

   g. Try some of these commands again, using command numbers instead of command names. No solution to this step is provided.

Solution

1. Korn Shell Command History Variables

   a. By default, the Korn shell retains the last 128 commands in the history buffer. The default history file is `$HOME/.sh_history`.

   b. This file must be occasionally deleted because it could grow very large if not automatically purged.

   c. To control the number of commands in the history buffer, set HISTSIZE. To control the name of the `history` file, set HISTFILE.

2. Korn Shell Command History Set Up

   a. Change the prompt to include the command number.

```
$ cat >> .profile
PS1='! $ '
<Ctrl/D>
$ PS1='! $ '
95 $
```

   b. Execute the following sequence of commands.

```
95 $ more .profile
. . .
PS1='! $ '
$ date
Mon Jul  6 12:08:19 EDT 1998
97 $ set
ERRNO=10
. . .
98 $ stty -a
#2 disc;speed 9600 baud; 48 rows; 80 columns
99 $ name="Joe User"
100 $ address="1 Main Street"
101 $ echo $name
Joe User
```

3. Korn Shell `fc -l` Function

   a. **`$ fc -l`** — Displays 16, or fewer commands if that is all you have used.

   b. $ **`history`** — Also displays the previous 16 commands. `history` is the default alias for `fc -l`.

   c. **$ `fc -l n n+4`**

```
104 $ fc -l 96 100
96      date
97      set
98      stty -a
99      name="Joe User"
100     address="1 Main Street"
```

   d. **$ `fc -lr n n+4`**

```
105 $ fc -lr 96 100
100     address="1 Main Street"
99      name="Joe User"
98      stty -a
97      set
96      date
```

e. $ **fc -ln n n+4**

```
106 $ fc -ln 96 100
      date
      set
      stty -a
      name="Joe User"
      address="1 Main Street"
```

f. $ **fc -l set echo**

```
107 $ fc -l set echo
97      set
98      stty -a
99      name="Joe User"
100     address="1 Main Street"
101     echo $name
```

4. Korn Shell `fc -e` Command

a. $ **history**

```
108 $ history
. . .
95      more .profile
96      date
97      set
98      stty -a
99      name="Joe User"
100     address="1 Main Street"
101     echo $name
. . .
```

b. $ **fc -e - name=address echo**

```
109 $ fc -e - name=address echo
echo $address
1 Main Street
```

c. $ **fc -e - name=USER *n***

```
110 $ fc -e - name=USER 101
echo $USER
joe
```

d. $ **fc -e - USER=SHELL**

```
111 $ fc -e - USER=SHELL
echo $SHELL
/bin/ksh
```

e. $ **r SHELL=HISTFILE**

```
112 $ r SHELL=HISTFILE
echo $HISTFILE
```

The `r` command does the same thing as `fc -e -` because `r` is a built-in alias for `fc -e -`.

f. $ **fc -e - .profile=.login more**

```
113 $ fc -e - .profile=.login more
more .login
#
. . .
```

Korn Shell Command Line Editing

Overview

The Korn shell offers 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 section 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 can 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, from 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, the `VISUAL` variable would override the other two commands and `emacs` would be the command line editor used.

Example 16-5: Setting the Command Line Editor

```
$ set -o vi
$ EDITOR=gmacs
$ VISUAL=emacs
```

You need to use only one method of selecting your command line editor, and should place that command in your `.profile` file.

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`. These editors 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 C. The commands are separated into tables by function as follows:

- Fetching commands from the history file

- Command execution
- Moving the cursor
- Deleting or replacing characters, words, and lines
- Entering `vi` insert mode
- Using uppercase and lowercase letters
- Pathname expansion or completion
- 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.

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. Use insert mode 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 the following table.

Table 16-2: vi Insert Mode Commands

Command	Function
Esc (escape)	Places the `vi` command line editor into command mode
Backspace (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 by entering 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 C contains tables of commands to be used with `emacs`, `gmacs`, or `vi` command line editing. The commands for `vi` are for command mode only.

Now Try This!

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 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 response 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

   a. Using the `emacs` option, turn on the `emacs` command line editor.

   b. 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 C. Do not edit commands yet; editing is discussed later.

Function	Command
Recalls previous command or [count] commands back	`Esc [count] Ctrl/P`
Recalls next command or [count] commands ahead	`Esc [count]  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`

   Note: Use `Esc [count]`only to provide a count. If the F11 key does not work as Esc on your terminal, use Ctrl/[.

   c. 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 C. Again, do not 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 indicating there is more text.

The following notation indicates that there is more text to the left and/or right.

Right — >

Left — <

Right and Left — *

The Korn shell also sends the bell character when you attempt to execute a command 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 C. 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 C. 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 "is not that old after all!"
> fi
```

a. Select `emacs` as your command line editor.

Before proceeding further, make sure that Ctrl/O does not interfere. By default, discard is defined as Ctrl/O (`stty -a`). Pressing Ctrl/O turns off the output and does not execute the command on the command line. To prevent this, you must **undefine** discard. Execute the following:

```
$ 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, press 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 `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.

Solution

1. Korn Shell Command Line Editor Selection

a. `Display options`

```
$ set -o
Current option settings
allexport       off
bgnice          on
emacs           off
gmacs           off
vi              on
. . .
```

To turn off an option, use `set +o option .`

```
$ set +o vi
```

b. Execute the following commands.

```
$ print $EDITOR
$ print $VISUAL
```

To clear a variable, use `unset variable`.

```
$ unset $VISUAL
```

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

a. Using the `emacs` option, turn on the `emacs` command line editor.

```
$ set -o emacs
```

b. Use the commands listed in the table. No solution provided.

c. After practicing the different `emacs` commands, turn off the `emacs` command line editor and turn on the `vi` command line editor.

```
$ set -o vi
```

Now, use the commands from the table. No solution provided.

3. Editing Long Korn Shell Command Lines

a. Editing Long Korn Shell Commands with `emacs`

```
$ set -o emacs
```

See Appendix C for commands.

b. Editing Long Korn Shell Commands with `vi`

```
$ set -o vi
```

See Appendix C for commands.

4. Editing Multiple-Line Commands

```
$ name="Joe"
$ age=40
$ if (($age == 40))
> then print $name "is over the hill!"
> else print $name "is not that old after all!"
> fi
Joe is over the hill!
```

a. Select `emacs` as your command line editor.

```
$ set -o emacs
<Ctrl/P> <Ctrl/P>
$ if (($age == 40))
$ if (($age == 50))
<Ctrl/O>
> then print $name "is over the hill!"
<Ctrl/O>
> else print $name "is not that old after all!"
<Ctrl/O>
> fi <Return>
Joe is not that old after all!
```

b. Select `vi` as your command line editor.

```
$ set -o vi
<Esc> 1k
$ if (($age == 50))^Jthen print $name "is over the hill!">
11l
r4
$ if (($age == 40))^Jthen print $name "is over the hill!">
<Return>
Joe is over the hill!
```

Editing Korn Shell Commands in a Temporary Buffer

Using a Temporary Buffer

One command, or multiple commands, can be edited in a temporary buffer using the `fc` command with the following format when using the `fc` command to edit commands in a temporary buffer:

```
fc -e editor first last
```

- The editor to be used is supplied on the command line by using the `-e` option, or is defined by the `FCEDIT` variable if no editor value is specified.
- *`first`* and *`last`* are the numbers or names of the first and last commands in the sequence to be edited. If *`last`* is not used, only one command is edited.
- The selected commands are placed in a temporary buffer for editing.
- After making necessary changes, writing the temporary file causes command execution.
- Exiting the editor without writing the file causes the original commands in the buffer to be executed.

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 will cause 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, 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.

The following example shows how to use the `fc` command to edit commands in a temporary buffer.

Example 16-6: Editing Korn Shell 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

Now Try This!

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 `emacs` as the editor. Make the same changes that you did with `vi`, and exit with Ctrl/X Ctrl/C . Save the file.

   Do the modified or unmodified commands get executed? Why?

Solution

1. Sample solution.

```
160 $ more .profile
. . .
161 $ name="Joe"
162 $ address="1 Main Street"
163 $ echo $name
Joe
164 $ fc -l
. . .
160     more .profile
161     name="Joe"
162     address="1 Main Street"
163     echo $name
164     fc -l
```

2. Edit command number `n` through `n+3` in a temporary buffer using the `vi` editor.

```
$ fc -e vi 160 163
more .profile
name="Joe"
address="1 Main Street"
echo $name
~
. . .
~
"/tmp/sh36155.1" 4 lines, 60 characters

(Use the vi editor to make it look like this)

more .login
name="Sue"
address="5 Main Dunstable Road"
echo $address
```

   If you quit `vi` with `:q!`, the original commands are executed because the temporary file is not written.

3. Assign the `FCEDIT` variable the value `emacs`, then use `fc` without the `-e` option.

```
$ FCEDIT=emacs
$ print $FCEDIT
emacs
$ fc 160 163
more .profile
name="Joe"
address="1 Main Street"
echo $name

. . .
-----XEmacs: sh36155.2     (Nroff)----All----------------------
```

```
Loading nroff-mode...done

(Use the emacs editor to make it look like this)

more .login
name="Sue"
address="5 Main Dunstable Road"
echo $address

<Ctrl/X><Ctrl/C>
Save file /tmp/sh36155.2? (y, n, !, ., q, or C-h) y
```

The modified commands get executed. 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.

Summary

17

Introducing Korn Shell Scripts

Unit Overview

Introduction

This chapter contains introductory level material that describes how to work with Korn shell scripts. The following topics are discussed:

- Creating and executing Korn shell scripts
- Defining and using variables
- Korn shell built-in variables
- Relational operators

Objectives

To use Kron shell scripts, 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

For more information on the topics in this chapter, see the following:

- *Common Desktop Environment: User's Guide*
- *Korn shell Programming Tutorial*
  Barry Rosenberg; published by Addison-Wesley Publishing Company, Inc. Reading, MA 01867 (c) 1991 ISBN 0-201-56324-X
- *The Korn shell Command and Programming Language*
  Morris I. Bolsky and David I. Korn; published by Prentice Hall, Englewood Cliffs, New Jersey 07632 (c) 1989 ISBN 0-13-516972-0
- *Ultrix to Tru64 UNIX Migration Guide*
  Compaq Order Number: AA-PJUAA-TE

Writing Simple Korn Shell Scripts

Overview

Shell scripts are useful for:

- Customizing your environment using your `.profile` file
- Performing repetitive tasks of several commands
- Submitting jobs in background mode

This chapter discusses basic features of Korn shell scripts for Tru64 UNIX users.

Creating Korn Shell Scripts

Korn shell scripts are easy to create, debug, and maintain. You can use any editor to create a file containing commands to be executed. The following example contains a simple Korn shell script.

Example 17-1: Simple Korn Shell Script

```
#!/bin/ksh    ❶
#
# intro-script.ksh     ❷
# 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         ❸
pwd
ls
cd book
pwd
ls
cd
```

❶ The first line of the script indicates the shell type.

❷ After the first line, any text following the pound sign (#) on a line is a comment.

❸ Commands executed by the script are placed on new lines using normal syntax.

To see what happens when you execute the script, enter:

```
$ ksh intro-script.ksh
```

The output from executing the command is as follows:

```
/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:

  `$ ksh filename`

- Run the script in your current shell:

  `$ . filename`

- Make the script executable, and invoke it as a command:

  `$ chmod u+x filename`
  `$ filename`

To invoke a subshell, you must have read privileges set on the file. Any shell specified on the first line of the script is ignored. The "`dot filename`" method is a Korn shell built-in feature, and will not work with the C shell.

Korn Shell Environment

The first line of a Korn shell script should contain the line `#!/bin/ksh`, both to inform the program reader, and to force execution of the script in the Korn shell environment if no subshell is specified.

You can execute Korn shell scripts in your current directory or in remote directories by specifying the search path on the command line. Make sure that the correct path variable, PATH, is set before you try to execure the script, since PATH contains the search path for your command line.

The following example shows how to modify the PATH variable to execute a Korn shell script that has been made executable.

Example 17-2: Modifying PATH to Execute a Script

```
$ 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    intro-script.ksh    misc    time.status
/usr/users/you/book
chap1   chap2   contents.dis    intro.txt
$
```

Now Try This!

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 your current directory
   - — Shows 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.

Solution

1.

```
$ 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.

a.

```
$ cat ps.ksh
#!/bin/ksh
#
# Korn shell script to display active processes
#
ps
print $hi
$
```

b.

```
$ hi=HELLO
$
```

c.

```
$ ksh ps.ksh
  PID TT  S   TIME COMMAND
  573 p5  S0:03.53 -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
$
```

Using Variables in Korn Shell Scripts

Overview

The Korn shell lets you use variables interactively and within shell scripts to modify information that has changing values. For example, you can modify the value of the PATH variable to include new directories. Variables such as PATH are reserved by the shell.

Using shell and user-created variables in Korn shell scripts allows you to read new information or change the output of your scripts to meet changing needs. Assignment of variable values can be done in a variety of ways, as shown in the following examples.

Variable	Function
$ nvalue=100	Assigns a numerical value to the `nvalue` variable; by default, all variables are strings
$ typeset -i nvalue=50	Declares `nvalue` to be an integer and `nvalue=50` 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

The next example shows the results of executing a series of commands for setting, displaying, and unsetting variables.

Example 17-3: Script for Modifying 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

Variables provide a way to easily modify the arguments that get passed to a command, as shown in the following example.

Example 17-4: Script for Modifying Arguments

```
$ cat greeting.ksh.
#!/bin/ksh
#
# greeting.ksh#
# This script uses variables to issue a welcome message
message="Good Morning"   ❶
application="Shell Programming"   ❷
print  ""     ❸
print "$message, $USER!"      ❹
print "bb"
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 dollar sign ($), apostrophe (‘), quotation marks ("), and backslash (\).

❻ Execute the `greeting.ksh` shell script.

Reading Variables

There are times when the value of a variable must be supplied by the user at run time. A shell script that allows its user to enter information at run time is known as an **interactive** script.

The following example uses the `read` command, followed by the variable name, to prompt the user to enter information interactively.

Example 17-5: Interactive Script

```
$ cat printname.ksh
#!/bin/ksh
#
# printname.ksh
#
# This Korn shell script prompts the user for their
# first, middleand 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 variables are returned by placing the $ in front of variable name.

Korn Shell Built-In Variables

The Korn shell provides some built-in variables that have special meaning to the shell. Issuing the `set` command returns the values of the built-in variables, as shown in the following example.

Example 17-6: Returning Values of 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
TMOUT=0
USER=you               ❼
_=set                  ❽
nvalue=100             ❾
y=100
```

❶ The name of the user

❷ Colon-separated command search path

❸ The user's parent process identification number

❹ PS1 is the primary prompt string for the shell (default is $). PS2 is the secondary prompt (default is >). PS3 is the selection prompt string used within a select loop (default is #). PS4 is the parameter substitution prompt and is used as the first character in each line of an execution trace (default is +)

❺ The user's current working directory

❻ The pathname of the user's shell

❼ The login name of the user

❽ A temporary variable that contains the last argument of the previous shell command; a portion of the previous shell command executed

❾ A user-created variable

To have a customized environment, users can define the built-in variables that are modifiable such as `PATH`, `PS1`, `CDPATH`, and `HISTSIZE` in the `.profile` file in their login directory. The following example provides a sample of the commands used to set these variables.

Example 17-7: Commands Used to Set Variables

```
$ 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 (>). Caution - pathnames can be very long. A long pathname in a prompt would leave little room for command input.

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 only available 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:

Example 17-8: Defining Global Variables

```
$ dog=cat
$ export dog
$ print $dog
cat
$
```

Use the `ksh` command to fork a child shell. Global variables can be seen at a child level as follows.

Example 17-9: Global Variables in a Child Shell

```
$ 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 Create 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 the child shells

❻ Invoke another subshell

❼ The global variable is now available to the child shell

Now Try This!

These lab exercises explore the use of variables in Korn shell 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 using just `=` 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.

Solution

1.

```
$ typeset first=you
$ set
. . .
first=you
. . .
```

2.

```
$ cat welcome.ksh
#!/bin/ksh
#
first="Tom"
print "Welcome to the World of $first"
```

```
$ cat >> .profile
welcome.ksh
<Ctrl/D>
```

3.

a.

```
$ typeset -L10 first=Charlie
```

b.

```
$ typeset -L10 last=MacArthy
```

c.

```
$ print “$first$last”
Charlie   MacArthy
$ typeset -L8 first=Charlie
$ typeset -L8 last=MacArthy
$ print "$first$last"|
Charlie MacArthy
```

d.

```
$ first=Pugnacious
$ last=Pugilist
$ print "$first$last"
PugnacioPugilist
```

In this case, the value of `first` is truncated.

e.

```
$ /bin/ksh
$ print “$first$last”
$ exit
$
```

The variables are not available to a subsequent shell.

f.

```
$ typeset -xL11 first=Pugnacious
$ typeset -xL11 last=Pugilist
$ /bin/ksh
$ print “$first$last”
Pugnacious Pugilist
$ exit
$
```

Using Flow Control and Loops

Overview

As you write more complex shell programs, you must be able to verify input and results. Conditional and loop statements allow you to execute commands such as `if` and `while`, or until specified conditions are true.

Relational Operators

The Korn shell script relational operators let you make comparisons that a numeric or string value based on a condition being met. The table provides numerical relational operators.

The format for a numeric test is: `(( number1 operator number2 ))`. The following table shows numeric relational operators.

Table 17-1: Numeric Relational Operators

Operator	Comparison
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

The format for string tests is: `[[string1 operator string2]]`. The following table shows string relational operators.

Table 17-2: String Relational Operators

Operator	Comparison
=	Equal to
!=	Not equal to
<	Greater than
>	Less than

Flow Control: If Statement

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. The basic syntax is as follows.

```
if (( condition ))
then
      commands
else
      commands
fi
```

In the following example, the `checkname.ksh` script demonstrates the use of the relational operators.

Example 17-10: Relational Operators in ksh Scripts

```
$ 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
USAGE:  checkname.ksh first middle last
$
$ checkname.ksh Garfield the Cat
Your name is Garfield the Cat.
$
```

This script also illustrates the use of command line arguments commonly called **positional** parameters, shown in the following table.

Table 17-3: Positional Parameters

PositionalParameter	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

Table 17-3: Positional Parameters (Continued)

PositionalParameter	Meaning
$n	Name of *n*th argument to script, function, or set. For argument numbers greater than nine, you must use braces around the argument number. For example: ${10} indicates the 10th parameter.

Logical Operators

Logical operators test whether or not a group of arguments meet a specified condition. These operators evaluate to either TRUE or FALSE. Korn shell logical operators are shown in the following table.

Table 17-4: Korn Shell 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

The following example shows a script that prompts for your age and then uses the OR operator to return a ticket price based on your input.

Example 17-11: Logical Operators in ksh Scripts

```
$ 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

$ age.ksh

Enter your age --
45
Ticket price is $5.00
```

Flow Control: while Loop

The following `while.ksh` script shows the use of the `while` statement which executes as long as a condition is true.

Example 17-12: ksh Script While Loops

```
$ 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 loops as long as a condition is false.

Example 17-13: ksh Script Until Loops

```
$ 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
$
```

Flow Control: for Loop

The most common loop command is the `for` command, which specifies a list of values for a variable. The following example assigns three values to `n` as it loops through the list of elements: 1 2 3.

Example 17-14: ksh For Loops

```
$ cat for.ksh
 #!/bin/ksh
 #
 # for.ksh
 #
 # Loop as long as there is another element
 # in the list.  “in” is a key word to
 # separate n from the list of elements
 #
 integer total=0 n=0
 for n in 1 2 3
     do total=total+n
 done
 print Total from 1 to 3 is $total
 $ for.ksh
 Total from 1 to 3 is 6
 $
```

Now Try This!

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.

Solution

```
$ 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
$
```

Conditional Execution Based on the Exit Status

Shell commands, shell scripts, and system commands either:

- Succeed and display the results on the screen, or
- Fail and the shell prints some type of diagnostic error message

The shell programmer can use the exit status to determine whether or not to execute the next command in a shell script.

Execution Based on Success

To execute the next command if the first command is successful, the format is:

```
<first command> && <second command>
```

The following example shows a second command providing a success message if the previous command is successful in deleting a file.

Example 17-15: Conditional Execution Based on Success

```
$ rm $1 && print "$1 has been deleted."
$
```

Execution Based on Failure

To execute the next command if the first command fails, the command format is:

```
<first command> || <second command>
```

The next example shows a command that is executed if the preceding command fails.

Example 17-16: Conditional Execution Based on Failure

```
$ grep “$1” datafile || print "$1 is not in the database."
$
```

Signal Trapping

Signals are provided in a UNIX operating system to control the system. Signals are used in programming to send a message to a script to stop it altogether, or momentarily.

To see the signals available on your system, enter: `kill -l`. Most signals sent to a script will terminate the script. To prevent this from happening, programmers try to capture signals rather than let them interfere with their program.

Korn shell programming provides the `trap` command to hold signals and events and capture error information. The following 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 you start the script, enter numerous `<Ctrl/C>` and `<Ctrl/\>` commands during the 15 seconds it takes to run the script.

Example 17-17: ksh Script Signal Trapping

```
$ 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   ❶
<Ctrl/ \> <Ctrl/ \> <Ctrl/C> ouch   ❷
<Ctrl/C> ouch   ❸
<Ctrl/ \> <Ctrl/ \> <Ctrl/ \> <Ctrl/C> ouch   ❹
All done: 4 ^Cs
$
```

The `kill` signals are displayed with the `kill -l` command. The following table shows signals on a Tru64 UNIX system. They might be different on your system.

Table 17-5: Tru64 UNIX kill Signals

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

The `ksh -v` option to the `ksh` command prints each executable line of the script before substitutions are made. The `-x` option of the command prints each executable line after substitutions are made with a + to highlight the execution.

The following example uses `age.ksh` to illustrate the use of `ksh -x`.

Example 17-18: Debugging Korn Shell Scripts with ksh -x

```
$ ksh -x age.ksh
ksh -x elif.ksh
+ print Enter your age --
Enter your age --
+ read age
32
+ (( age <= 6 || age >= 64 ))
+ (( age >=40 && age <=43 ))
+ (( 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 next example uses the `while.ksh` file to demonstrate the use of `nl`.

Example 17-19: 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"
$
```

Now Try This!

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/C 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.

Solution

1.

```
#!/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.

```
#!/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.

```
$ PS4=’at $LINENO - ‘
$ export PS4
```

4.

```
#!/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.

```
$ ksh -xv misspell.ksh chap*
```

Summary

Writing Simple 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.

Using Flow Control and Loops

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

Summary

18

Test

Questions

In the space provided, write the letter corresponding with the best answer to each of the following multiple-choice questions.

1. _____ UNIX was the first operating system to be written in which high-level language?

   a. FORTRAN

   b. C

   c. Bliss

   d. Assembler

2. _____ UNIX has which of the following file systems?

   a. Flat

   b. Hierarchical

   c. Transversal tree

   d. Sequential

3. _____ Which of the following are standard user interfaces supplied with Tru64 UNIX?

   a. DECwindows and the Command Line Interface

   b. DECwindows and the Common Desktop Environment

   c. Common Desktop Environment and Command Line Interface

   d. Motif windows and the Command Line Interface

4. _____ Which login option would you use to execute a few commands and then use CDE?

   a. Logging in using the Command Line

   b. Logging in using CDE

   c. Logging in using a different language

   d. Logging into a Failsafe session

5. _____ What command changes your password?

   a. `pwd`

   b. `chpass`

   c. `passwd`

   d. `set pass`

6. _____ In which documentation kit would you find information on how to use the Common Desktop Environment?

   a. Startup Documentation Kit

   b. Developer's Kit

   c. System and Network Management Documentation Kit

   d. General User Documentation Kit

7. _____ Which statement is not true?

   a. Controls can have one or more of the following behaviors: click, drop and indicator

   b. Each workspace has its own Front Panel

   c. The Front Panel consists of the Main Panel, subpanels and the Workspace switch area

   d. The Help control and its subpanel provide access to information on the Help manager, Desktop and Front Panel

8. _____ To remove your user customizations of the Front Panel, the most appropriate action would be:

   a. Change the entry in the `.dtprofile` file

   b. Use Restore Front Panel in the Application Manager's Desktop_tools folder

   c. Log out

   d. Make a request to your system administrator

9. _____ Which of the following can you accomplish with the Style Manager?

   a. Change the appearance of your desktop

   b. Set your home and current sessions

   c. Change the behavior of your keyboard, mouse and windows

   d. All of the above

10. _____ What command determines which keys on your terminal serve the erase, kill, and interrupt functions?

a. `stty`

b. `term`

c. `print tty`

d. `sys`

11. _____ 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

12. _____ What type of file contains a group of files?

a. Regular disk file

b. Directory

c. Device file

d. Special file

13. _____ Which command provides the type of information contained in a particular file?

a. `more filename`

b. `type filename`

c. `file filename`

d. `cat filename`

14. _____ The UNIX file system structure logically looks like an inverted tree. This type of file system is called:

a. Root structure

b. Bottom-up

c. First-level

d. Hierarchical

15. _____ To view your files with a CDE tool, click:

- **a.** Help Manager
- **b.** File Manager
- **c.** Application Manager
- **d.** Workspace switch

16. _____ Which is not a component of the File Manager?

- **a.** Current folder path
- **b.** Menu bar and menus
- **c.** Application area
- **d.** Object icons

17. _____ Which is not a basic drag and drop task?

- **a.** Move files to another folder
- **b.** Select an object
- **c.** Place a file or folder in your desktop workspace
- **d.** Print a file

18. _____ One way to open a file is:

- **a.** Double-click its icon
- **b.** Drag and drop the icon to the Print Manager
- **c.** Select the icon and press Return
- **d.** Click File Manager on the Front Panel

19. _____ Which does not meet the critria to find a file?

- **a.** Text string
- **b.** Directory path
- **c.** Pattern of characters
- **d.** Size of the file

20. _____ Which is not a valid way to display files and folders?

a. By name only

b. By icon

c. By name and file information

d. By permission

21. _____ What does the command `pwd` result in?

a. Your home directory becomes your current directory

b. The path to your current directory is displayed

c. Your password is displayed

d. The path to your home directory is displayed

22. _____ In the UNIX operating system, a full pathname originates from:

a. Your home directory

b. The root directory

c. Your working directory

d. Your login directory

23. _____ The command to change your current directory is:

a. `cd pathname`

b. `set default pathname`

c. `sd pathname`

d. `ccd pathname`

24. _____ What command changes your working directory to your home directory?

a. `home`

b. `cd`

c. `dir/h`

d. `clrpathname`

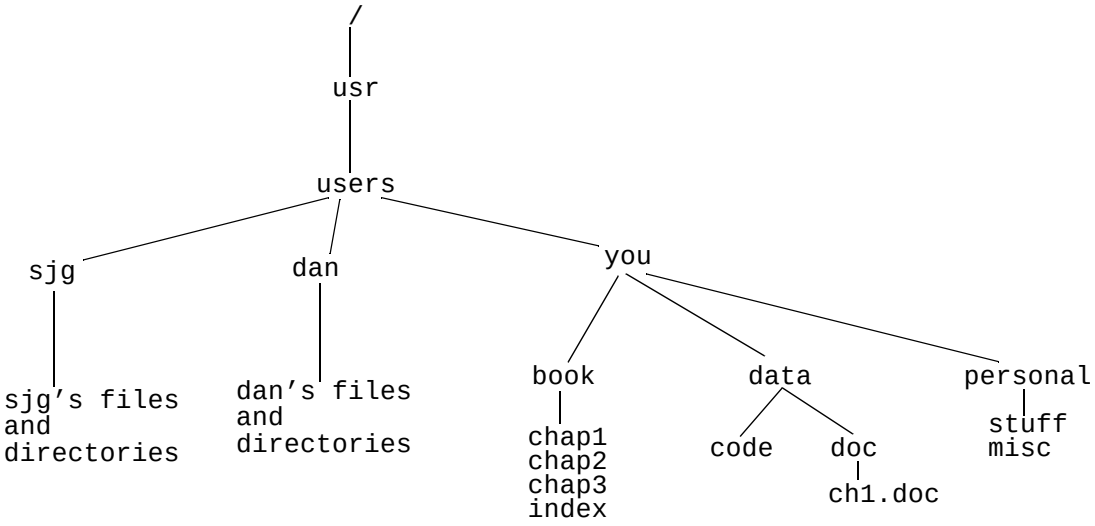
25. _____ You have just finished creating the `chap1` file. A listing of your working directory shows the four files `chap1`, `chap2`, `chap3` and `index`. What is the output of the `pwd` command if the directory structure is that of the following figure?

a. `/usr/you/data`

b. `/usr/you/dan`

c. `/usr/users/you/personal`

d. `/usr/users/you/book`



26. _____ What command displays the files you have?

a. `dir`

b. `files`

c. `cp`

d. `ls`

27. _____ 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`

28. _____ What command searches for files and directories based on certain specified criteria?

a. `find`

b. `search`

c. `grep`

d. `locate`

29. _____ Which expression, when used with the `find` command, displays file names in the format of a long listing?

a. `-exec`

b. `-print`

c. `-display`

d. `-ls`

30. _____ Which command displays a file, one screenful at a time?

a. `more filename`

b. `type filename`

c. `file filename`

d. `cat filename`

31. _____ Which command displays an entire file on the screen?

a. `more filename`

b. `type filename`

c. `file filename`

d. `cat filename`

32. _____ Your working directory is your home directory. How would you display the last 10 lines of the file `chap3` in the directory `/usr/users/you/book`?

a. `cat book/chap3`

b. `tail book/chap3`

c. `tail /book/chap3`

d. `cat /usr/users/you/book/chap3`

33. _____ Which method does not create a file?

a. Clicking the New File menu item

b. Copying a file

c. Moving a file

d. Using a text editor

34. _____ Which statement is not true?

a. After the trash is shredded, your files are removed from the trash can

b. You can open the trash can by double-clicking the trash can

c. To move files to the trash can, drag and drop the file on the trash can icon

d. Trash in the trash can is shredded every hour

35. _____ Which does not describe the purpose of copying files?

a. Backing up files

b. Making links to other files

c. Renaming a file to another location

d. Copying a file to another location

36. _____ Which command allows you to create a file named `test2`?

a. `create test2`

b. `cat test2`

c. `touch test2`

d. `make test2`

37. _____ Which command creates a directory?

a. `credir`

b. `create`

c. `dir/create`

d. `mkdir`

38. _____ 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`

Use the following example to answer the next three questions.

```
drwxr-x--x   5 you   users  512 Nov  5 11:04 .
drwxr-xr-x  26 root  users  512 Oct 18 15:41 ..
-rwxr-x--x   1 you   users  373 Jun 29 15:22 .cshrc
-rwx--x---   1 you   users  145 Jun 29 15:22 .login
drwxr-xr-x   2 you   users  512 Nov  5 09:38 book
-rwx---r--   1 you   users   88 Sep  3 10:30 letters
drwx------   2 you   users  512 Nov  5 11:16 misc
-rw-rw-r--   1 you   users  178 Nov  5 09:30 time.status
```

39. _____ How many files are inaccessible to members of the group called users?

a. 1

b. 4

c. 2

d. All of them

40. _____ How many files or directories are executable by the owner?

a. 4

b. 7

c. 5

d. All of them

41. _____ Which command changes the permission on the `time.status` file so that only you can read or write it?

a. `set permission [rw,rw,,] time.status`

b. `chmod g-rw,w-r time.status`

c. `chmod g-rw,o-r time.status`

d. `chmod u+rw time.status`

42. _____ Which command 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`

43. _____ Which command, issued from your home directory, moves 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`

44. _____ Your current working directory is `/usr/users/you/database/code`. Which command copies the file `design.detail` from the `/usr/users/you/database/doc` directory into a file in your current directory called `design`?

a. `cp /usr/users/you/database/doc/design.detail design`

b. `cp /usr/users/you/database/doc/design.detail detail`

c. `cp /database/doc/design.detail /database/code/design`

d. `cp design.detail /usr/users/you/database/code/design`

45. _____ Which command removes a file named `work1`?

a. `delete work1`

b. `del work1`

c. `rm work1`

d. `remove work1`

46. _____ Issued from your home directory, what command deletes 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`

47. _____ Assume your working directory is `/usr/users/you/work` and there is an empty subdirectory in the working directory named `projects`. What command removes that empty directory?

a. `rm projects`

b. `rmdir projects`

c. `rm /usr/users/you/projects`

d. `rmdir work/projects`

48. _____ Which statement is true of hard links?

a. May refer to directories

b. Indistinguishable from the original file

c. May span file systems

d. Does not share inodes

49. _____ Which statement 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

50. _____ You can make appointment entries from which Calendar Application views?

a. Day, Week, Year

b. Week, Month, Year

c. Day, Week, Month

d. Day, Week

51. _____ With the Text Editor you can add new text into a file by:

a. Copying and pasting from one window to another

b. Using the Text Editor's drag and drop feature

c. Typing in the information

d. All of the above

52. _____ To enter commands while using CDE, you would click which control?

a. Calendar

b. Mailer

c. Terminal

d. Desktop_Apps

53. _____ The system administrator places frequently used applications in which system folder?

a. Personal Applications folder

b. User's folder

c. System Administrator's folder

d. Application Manager's folder

54. _____ Which of the following could be considered as a purpose of the shell?

a. User interface to the UNIX operating system

b. A command language interpreter

c. A programming language

d. All of the above

55. _____ Which shell(s) provide a command history buffer?

a. Bourne shell

b. C shell

c. Korn shell

d. Both C and Korn shells

56. _____ What is not a file name for a Tru64 UNIX shell?

a. `/bin/sh`

b. `/bin/bsh`

c. `/bin/ksh`

57. _____ The Korn shell wildcard operator `?` can be used to match:

a. Either zero or one arbitrary characters

b. Exactly one character

c. One or more characters

d. Numeric characters only

58. _____ To list all the files in the current working directory with a `.txt` extension, use the command:

a. `ls .txt?`

b. `ls ?.txt`

c. `ls *.txt`

d. `ls .txt`

59. _____ You can verify the setting of all Bourne, C, and Korn shell variables with which command?

a. `show variables`

b. `set`

c. `set variables`

d. `show -all`

60. _____ The value of any particular shell variable can be displayed in the Bourne, C, and Korn shell with which command?

a. `print variable_name`

b. `variable_name =`

c. `echo $variable_name`

d. `print $variable_name`

61. _____ What 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"`

62. _____ All Korn shell options are displayed with which command?

a. `set -o {option}`

b. `set -o`

c. `set +o {option}`

d. `set +o`

63. _____ Which command turns on `emacs` as the Korn shell command line editor?

a. `set -o emacs`

b. `set +o emacs`

c. `set emacs`

d. `set emacs on`

64. _____ Which is not a default shell login script?

a. `.login`

b. `.cshrc`

c. `.profile`

d. `.kshrc`

65. _____ The file name of one of the Korn shell login script files is based upon expansion of which variable?

a. `ENV`

b. `LOGIN`

c. `KSHSTART`

d. `SHELL`

66. _____ If you are using the C shell, and you start a subshell, which startup file is executed?

a. `.login`

b. `.profile`

c. `.kshrc`

d. `.cshrc`

67. _____ Which statement describes the purpose of the Korn shell environment script file?

a. Define aliases and functions that apply to interactive use and for scripts invoked from `ksh`

b. Set default options that you want to apply to all `ksh` invocations

c. Set variables that you want to apply to the current session

d. All of the above

68. _____ Which editor is available in single user mode?

a. `emacs`

b. `ed`

c. `vi`

d. `ex`

69. _____ Which of the following is a read-only version of the `vi` editor?

a. `ed`

b. `ex`

c. `sed`

d. `view`

70. _____ Which editor allows you to edit both multiple buffers and multiple windows?

a. `ed`

b. `vi`

c. `emacs`

d. `sed`

71. _____ 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

72. _____ Which answer best describes the `vi` editor modes?

a. Command mode; Edit mode; Text-input mode

b. Command mode; Text-input mode; Last-line mode

c. Cursor mode; Text-input mode; Last-line mode

d. Command mode; Text-input mode; Exit-mode

73. _____ 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

74. _____ Which control sequence 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)

75. _____ Which answer 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!`

76. _____ 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`

77. _____ What does the command `ls -l > listing` accomplish?

a. Places the output from `ls -l` into a file named `listing`

b. Uses the contents of `listing` as input to the command `ls -l`

c. Uses the output from the command `ls -l` as input to the command `listing`

d. Lists the contents of a file named `listing`

78. _____ 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

79. _____ Which command displays how many lines there are in a file named `flowers`?

a. `sort -l flowers`

b. `sort -w flowers`

c. `wc -l flowers`

d. `lines flowers`

80. _____ Which command sorts a file called `flowers` in reverse alphabetical order without regard to case?

a. `sort -f flowers`

b. `sort -r -f flowers`

c. `sort -r flowers`

d. `sort -r -m flowers`

81. _____ 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 data from 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`

82. _____ What does the `-m` option to `sort` accomplish?

a. Eliminates duplicate items when merging files

b. Merges two or more sorted files

c. Sorts and merges any files larger than 500 lines

d. Sorts and merges multiple files

Use the following example to answer the next four questions.

```
1)  go.c
2)  date
3)  go.bin
4)  go.lst
5)  a.out
6)  lifeguard
7)  glob.in
8)  layout.c
9)  layout.h
10) totals
11) .login
```

83. _____ Which strings are matched by the regular expression `'t$'`?

a. 10

b. 4, 5

c. 4, 5, 8, 9

d. 2, 4, 5, 8, 9, 10

84. _____ Which strings are matched by the regular expression `'...'`?

a. 2

b. 2, 5

c. 1, 2

d. All strings

85. _____ Which string is not matched by the regular expression `'^[la]'`?

a. `.login`

b. `layout.c`

c. `a.out`

d. `lifeguard`

86. _____ Which string is matched by the regular expression `'.........$'`?

a. `a.out`

b. `go.bin`

c. `layout.c`

d. `lifeguard`

87. _____ The single character that can be typed to use the output from one command as the input of another is:

a. Question mark (?)

b. Vertical bar (|)

c. Asterisk (*)

d. At sign (@)

e. Slash (/)

88. _____ Which command creates a listing of files, then sorts that listing in reverse alphabetical order and puts the results into a file named `sortedlist`?

a. `ls | sort -r > sortedlist`

b. `ls > list | sort`

c. `ls > sort -r sortedlist`

d. `ls > sortedlist > sort -r`

89. _____ Which command could be used 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?

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`

90. _____ Which command could be used to create a file called `phonenums` that contains a list of the phone numbers from a file called `phones`, if the phone number is the third field?

a. `awk phones [3] >! phonenums`

b. `awk "{print $3}" phones >| phonenums`

c. `awk '{print $3}' phones > phonenums`

d. `awk -o phones phonenums > $3`

91. _____ 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`

92. ____ The `awk` command was derived from:

a. The first letters of the first names of its creators

b. The first letters of the last names of its creators

c. The ability of the command to be awkward in its syntax

d. A South American bird

93. ____ Which command displays the real name and phone extension of user `pmj`?

a. `who pmj`

b. `whoami pmj`

c. `users pmj`

d. `finger pmj`

94. ____ Which command can view the commands in the history buffer?

a. `show history`

b. `history`

c. `cat history`

d. `view commands`

95. ____ 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

96. ____ 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`

97. ____ Which answer 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`

98. ____ 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`

99. ____ You can continue entering a long command on the next line by entering what character?

a. `;`

b. `\`

c. `/`

d. `-`

100.____ You can include several commands on the same command line by separating them with what character?

a. `\`

b. `;`

c. `,`

d. Tab

101.____ What command determines which processes are currently in execution, including those processes not connected to a terminal?

a. `stat`

b. `jobs`

c. `bg`

d. `ps -e`

102.____ The default signal sent by the `kill` command is:

a. `INT`

b. `KILL`

c. `TERM`

d. `USR1`

103.____ What character introduces comments in Korn shell scripts?

a. `:`

b. `\`

c. `;`

d. `#`

104.____ 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

105.____ What command could 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"`

106.____ Which word ends a conditional loop in the Korn shell?

a. `end`

b. `enddo`

c. `endif`

d. `done`

107.____ Which command copies files from a remote UNIX system to your local UNIX system over a TCP/IP network?

a. `copy`

b. `dcp`

c. `rcp`

d. `finger`

108.____ Which command logs you 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`

109.____ Which file would you edit to designate a different printer as your default printer?

a. `/etc/printcap`

b. `.dtprofile`

c. `.profile`

d. `.login`

110.____ Which environment variable sets the value for your default printer?

a. `PTRNAME`

b. `DISPLAY`

c. `LPDEST`

d. `PTRDEVICE`

111.___ 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

112.____ Which command provides status information for printer lp1?

a. `lpq -Plp1`

b. `lpstat -t`

c. `lpc status`

d. All of the above

113.____ 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

114.____ What command reads and writes tapes for use on a non-UNIX system such as OpenVMS, and can convert EBCDIC and ASCII files?

a. `dd`

b. `tar`

c. `cpio`

d. `mt`

115.____ What command can move tape forward and backward by number of files or records, and also rewinds tape?

a. `df`

b. `du`

c. `dd`

d. `mt`

116.____ What command displays the amount of free disk space?

a. `du`

b. `dd`

c. `df`

d. `ls -l`

117.____ What command displays the size of files in directories?

a. `du -s`

b. `du`

c. `df`

d. `dd`

Answers

1. ___b__ UNIX was the first operating system to be written in which high-level language?

   a. FORTRAN

   b. C

   c. Bliss

   d. Assembler

2. ___b__ UNIX has which of the following file systems?

   a. Flat

   b. Hierarchical

   c. Transversal tree

   d. Sequential

3. ___c__ Which of the following are standard user interfaces supplied with Tru64 UNIX?

   a. DECwindows and the Command Line Interface

   b. DECwindows and the Common Desktop Environment

   c. The Common Desktop Environment and Command Line Interface

   d. Motif windows and the Command Line Interface

4. ___d__ Which login option would you use to execute a few commands and then use CDE?

   a. Logging in using the Command Line

   b. Logging in using CDE

   c. Logging in using a different language

   d. Logging into a Failsafe session

5. __c___ What command changes your password?

   a. `pwd`

   b. `chpass`

   c. `passwd`

   d. `set pass`

6. __d___ In which documentation kit would you find information on how to use the Common Desktop Environment?

   a. Startup Documentation Kit

   b. Developer's Kit

   c. System and Network Management Documentation Kit

   d. General User Documentation Kit

7. __b__ Which statement is not true?

   a. Controls can have one or more of the following behaviors: click, drop and indicator

   b. Each workspace has its own Front Panel

   c. The Front Panel consists of the Main Panel, subpanels and the Workspace switch area

   d. The Help control and its subpanel provide access to information on the Help manager, Desktop and Front Panel

8. __b___ To remove your user customizations of the Front Panel, the most appropriate action would be:

   a. Change the entry in the `.dtprofile` file

   b. Use Restore Front Panel in the Application Manager's Desktop_tools folder

   c. Log out

   d. Make a request to your system administrator

9. ___d__ Which of the following can you accomplish with the Style Manager?

   a. Change the appearance of your desktop

   b. Set your home and current sessions

   c. Change the behavior of your keyboard, mouse and windows

   d. All of the above

10. ___a__ What command determines which keys on your terminal serve the erase, kill, and interrupt functions?

    a. `stty`

    b. `term`

    c. `print tty`

    d. `sys`

11. ___d__ 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

12. ___b__ What type of file contains a group of files?

a. Regular disk file

b. Directory

c. Device file

d. Special file

13. ___c__ Which command provides the type of information contained in a particular file?

a. `more filename`

b. `type filename`

c. `file filename`

d. `cat filename`

14. __d___ The UNIX file system structure logically looks like an inverted tree. This type of file system is called:

a. Root structure

b. Bottom-up

c. First-level

d. Hierarchical

15. ___b__ To view your files with a CDE tool, click:

a. Help Manager

b. File Manager

c. Application Manager

d. Workspace switch

16. __c___ Which is not a component of the File Manager?

a. Current folder path

b. Menu bar and menus

c. Application area

d. Object icons

17. ___b__ Which is not a basic drag and drop task?

a. Move files to another folder

b. Select an object

c. Place a file or folder in your desktop workspace

d. Print a file

18. ___a__ One way to open a file is:

a. Double-click its icon

b. Drag and drop the icon to the Print Manager

c. Select the icon and press Return

d. Click File Manager on the Front Panel

19. ___d__ Which does not meet the critria to find a file?

a. Text string

b. Directory path

c. Pattern of characters

d. Size of the file

20. __d___ Which is not a valid way to display files and folders?

a. By name only

b. By icon

c. By name and file information

d. By permission

21. __b___ What does the command `pwd` result in?

a. Your home directory becomes your current directory

b. The path to your current directory is displayed

c. Your password is displayed

d. The path to your home directory is displayed

22. ___b__ In the UNIX operating system, a full pathname originates from:

a. Your home directory

b. The root directory

c. Your working directory

d. Your login directory

23. ___a__ The command to change your current directory is:

a. `cd pathname`

b. `set default pathname`

c. `sd pathname`

d. `ccd pathname`

24. ___b__ What command changes your working directory to your home directory?

a. `home`

b. `cd`

c. `dir/h`

d. `clrpathname`

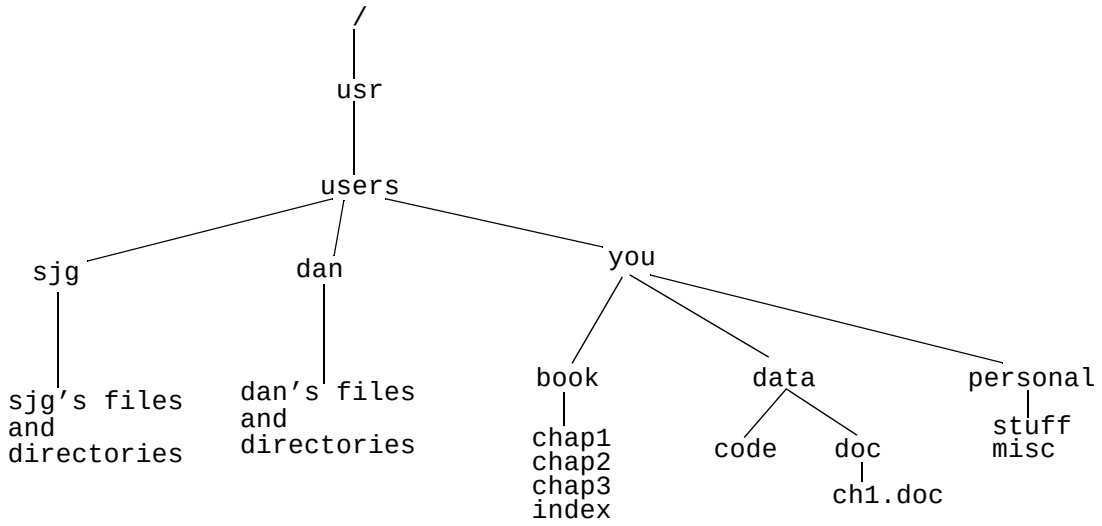
25. ___d__ You have just finished creating the `chap1` file. A listing of your working directory shows the four files `chap1`, `chap2`, `chap3` and `index`. What is the output of the `pwd` command if the directory structure is that of the following figure?

a. `/usr/you/data`

b. `/usr/you/dan`

c. `/usr/users/you/personal`

d. `/usr/users/you/book`



26. ___d__ What command displays the files you have?

a. `dir`

b. `files`

c. `cp`

d. `ls`

27. ___d__ 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`

28. ___a__ What command searches for files and directories based on certain specified criteria?

a. `find`

b. `search`

c. `grep`

d. `locate`

29. __d___ Which expression, when used with the `find` command, displays file names in the format of a long listing?

a. `-exec`

b. `-print`

c. `-display`

d. `-ls`

30. ___a__ Which command displays a file, one screenful at a time?

a. `more filename`

b. `type filename`

c. `file filename`

d. `cat filename`

31. __d___ Which command displays an entire file on the screen?

a. `more filename`

b. `type filename`

c. `file filename`

d. `cat filename`

32. ___b__ Your working directory is your home directory. How would you display the last 10 lines of the file `chap3` in the directory `/usr/users/you/book`?

a. `cat book/chap3`

b. `tail book/chap3`

c. `tail /book/chap3`

d. `cat /usr/users/you/book/chap3`

33. __c___ Which method does not create a file?

a. Clicking the New File menu item

b. Copying a file

c. Moving a file

d. Using a text editor

34. ___d__ Which statement is not true?

a. After the trash is shredded, your files are removed from the trash can

b. You can open the trash can by double-clicking the trash can

c. To move files to the trash can, drag and drop the file on the trash can icon

d. Trash in the trash can is shredded every hour

35. ___b__ Which does not describe the purpose of copying files?

a. Backing up files

b. Making links to other files

c. Renaming a file to another location

d. Copying a file to another location

36. ___c__ Which command allows you to create a file named `test2`?

a. `create test2`

b. `cat test2`

c. `touch test2`

d. `make test2`

37. ___d__ Which command creates a directory?

a. `credir`

b. `create`

c. `dir/create`

d. `mkdir`

38. ___a__ 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`

Use the following example to answer the next three questions.

```
drwxr-x--x   5 you   users  512 Nov  5 11:04 .
drwxr-xr-x  26 root  users  512 Oct 18 15:41 ..
-rwxr-x--x   1 you   users  373 Jun 29 15:22 .cshrc
-rwx--x---   1 you   users  145 Jun 29 15:22 .login
drwxr-xr-x   2 you   users  512 Nov  5 09:38 book
-rwx---r--   1 you   users   88 Sep  3 10:30 letters
drwx------   2 you   users  512 Nov  5 11:16 misc
-rw-rw-r--   1 you   users  178 Nov  5 09:30 time.status
```

39. ___c__ How many files are inaccessible to members of the group called users?

a. 1

b. 4

c. 2

d. All of them

40. ___b__ How many files or directories are executable by the owner?

a. 4

b. 7

c. 5

d. All of them

41. ___c__ Which command changes the permission on the `time.status` file so that only you can read or write it?

a. `set permission [rw,rw,,] time.status`

b. `chmod g-rw,w-r time.status`

c. `chmod g-rw,o-r time.status`

d. `chmod u+rw time.status`

42. __d___ Which command 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`

43. __a___ Which command, issued from your home directory, moves 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`

44. ___a__ Your current working directory is `/usr/users/you/database/code`. Which command copies the file `design.detail` from the `/usr/users/you/database/doc` directory into a file in your current directory called `design`?

a. `cp /usr/users/you/database/doc/design.detail design`

b. `cp /usr/users/you/database/doc/design.detail detail`

c. `cp /database/doc/design.detail /database/code/design`

d. `cp design.detail /usr/users/you/database/code/design`

45. __c___ Which command removes a file named `work1`?

a. `delete work1`

b. `del work1`

c. `rm work1`

d. `remove work1`

46. ___c__ Issued from your home directory, what command deletes 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`

47. ___b__ Assume your working directory is `/usr/users/you/work` and there is an empty subdirectory in the working directory named `projects`. What command removes that empty directory?

a. `rm projects`

b. `rmdir projects`

c. `rm /usr/users/you/projects`

d. `rmdir work/projects`

48. ___b__ Which statement is true of hard links?

a. May refer to directories

b. Indistinguishable from the original file

c. May span file systems

d. Does not share inodes

49. ___c__ Which statement 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

50. ___c__ You can make appointment entries from which Calendar Application views?

a. Day, Week, Year

b. Week, Month, Year

c. Day, Week, Month

d. Day, Week

51. ___d__ With the Text Editor you can add new text into a file by:

a. Copying and pasting from one window to another

b. Using the Text Editor's drag and drop feature

c. Typing in the information

d. All of the above

52. ___c__ To enter commands while using CDE, you would click which control?

a. Calendar

b. Mailer

c. Terminal

d. Desktop_Apps

53. __d___ The system administrator places frequently used applications in which system folder?

a. Personal Applications folder

b. User's folder

c. System Administrator's folder

d. Application Manager folder

54. ___d__ Which of the following could be considered as a purpose of the shell?

a. User interface to the UNIX operating system

b. A command language interpreter

c. A programming language

d. All of the above

55. ___d__ Which shell(s) provide a command history buffer?

a. Bourne shell

b. C shell

c. Korn shell

d. Both C and Korn shells

56. ___b__ What is not a file name for a Tru64 UNIX shell?

a. `/bin/sh`

b. `/bin/bsh`

c. `/bin/ksh`

57. ___b__ The Korn shell wildcard operator `?` can be used to match:

a. Either zero or one arbitrary characters

b. Exactly one character

c. One or more characters

d. Numeric characters only

58. ___c__ To list all the files in the current working directory with a `.txt` extension, use the command:

a. `ls .txt?`

b. `ls ?.txt`

c. `ls *.txt`

d. `ls .txt`

59. ___b__ You can verify the setting of all Bourne, C, and Korn shell variables with which command?

a. `show variables`

b. `set`

c. `set variables`

d. `show -all`

60. ___c__ The value of any particular shell variable can be displayed in the Bourne, C, and Korn shell with which command?

a. `print variable_name`

b. `variable_name =`

c. `echo $variable_name`

d. `print $variable_name`

61. ___d__ What 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"`

62. ___b__ All Korn shell options are displayed with which command?

a. `set -o {option}`

b. `set -o`

c. `set +o {option}`

d. `set +o`

63. ___a__ Which command turns on `emacs` as the Korn shell command line editor?

a. `set -o emacs`

b. `set +o emacs`

c. `set emacs`

d. `set emacs on`

64. ___d__ Which is not a default shell login script?

a. `.login`

b. `.cshrc`

c. `.profile`

d. `.kshrc`

65. ___a__ The file name of one of the Korn shell login script files is based upon expansion of which variable?

a. `ENV`

b. `LOGIN`

c. `KSHSTART`

d. `SHELL`

66. ___d__ If you are using the C shell, and you start a subshell, which startup file is executed?

a. `.login`

b. `.profile`

c. `.kshrc`

d. `.cshrc`

67. ___d__ Which statement describes the purpose of the Korn shell environment script file?

a. Define aliases and functions that apply to interactive use and for scripts invoked from `ksh`

b. Set default options that you want to apply to all `ksh` invocations

c. Set variables that you want to apply to the current session

d. All of the above

68. ___b__ Which editor is available in single user mode?

a. `emacs`

b. `ed`

c. `vi`

d. `ex`

69. ___d__ Which of the following is a read-only version of the `vi` editor?

a. `ed`

b. `ex`

c. `sed`

d. `view`

70. ___c__ Which editor allows you to edit both multiple buffers and multiple windows?

a. `ed`

b. `vi`

c. `emacs`

d. `sed`

71. ___d__ 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

72. ___b__ Which answer best describes the `vi` editor modes?

a. Command mode; Edit mode; Text-input mode

b. Command mode; Text-input mode; Last-line mode

c. Cursor mode; Text-input mode; Last-line mode

d. Command mode; Text-input mode; Exit-mode

73. ___c__ 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

74. ___b__ Which control sequence 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)

75. __a___ Which answer 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!`

76. ___d__ 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`

77. ___a__ What does the command `ls -l > listing` accomplish?

a. Places the output from `ls -l` into a file named `listing`

b. Uses the contents of `listing` as input to the command `ls -l`

c. Uses the output from the command `ls -l` as input to the command `listing`

d. Lists the contents of a file named `listing`

78. ___b__ 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

79. ___c__ Which command displays how many lines there are in a file named `flowers`?

a. `sort -l flowers`

b. `sort -w flowers`

c. `wc -l flowers`

d. `lines flowers`

80. ___b__ Which command sorts a file called `flowers` in reverse alphabetical order without regard to case?

a. `sort -f flowers`

b. `sort -r -f flowers`

c. `sort -r flowers`

d. `sort -r -m flowers`

81. ___a__ 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 data from 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`

82. ___b__ What does the `-m` option to `sort` accomplish?

a. Eliminates duplicate items when merging files

b. Merges two or more sorted files

c. Sorts and merges any files larger than 500 lines

d. Sorts and merges multiple files

Use the following example to answer the next four questions.

```
1)  go.c
2)  date
3)  go.bin
4)  go.lst
5)  a.out
6)  lifeguard
7)  glob.in
8)  layout.c
9)  layout.h
10) totals
11) .login
```

83. __b___ Which strings are matched by the regular expression `'t$'`?

a. 10

b. 4, 5

c. 4, 5, 8, 9

d. 2, 4, 5, 8, 9, 10

84. ___d__ Which strings are matched by the regular expression `'...'`?

a. 2

b. 2, 5

c. 1, 2

d. All strings

85. ___a__ Which string is not matched by the regular expression `'^[la]'`?

a. `.login`

b. `layout.c`

c. `a.out`

d. `lifeguard`

86. ___d__ Which string is matched by the regular expression `'.........$'`?

a. `a.out`

b. `go.bin`

c. `layout.c`

d. `lifeguard`

87. ___b__ The single character that can be typed to use the output from one command as the input of another is:

a. Question mark (?)

b. Vertical bar (|)

c. Asterisk (*)

d. At sign (@)

e. Slash (/)

88. __a___ Which command creates a listing of files, then sorts that listing in reverse alphabetical order and puts the results into a file named `sortedlist`?

a. `ls | sort -r > sortedlist`

b. `ls > list | sort`

c. `ls > sort -r sortedlist`

d. `ls > sortedlist > sort -r`

89. __d___ Which command could be used 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?

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`

90. __c___ Which command could be used to create a file called `phonenums` that contains a list of the phone numbers from a file called `phones`, if the phone number is the third field?

a. `awk phones [3] >! phonenums`

b. `awk "{print $3}" phones >| phonenums`

c. `awk '{print $3}' phones > phonenums`

d. `awk -o phones phonenums > $3`

91. ___d__ 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`

92. __b__ The `awk` command was derived from:

a. The first letters of the first names of its creators

b. The first letters of the last names of its creators

c. The ability of the command to be awkward in its syntax

d. A South American bird

93. __d__ Which command displays the real name and phone extension of user `pmj`?

a. `who pmj`

b. `whoami pmj`

c. `users pmj`

d. `finger pmj`

94. __b__ Which command can view the commands in the history buffer?

a. `show history`

b. `history`

c. `cat history`

d. `view commands`

95. __c__ 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

96. __c__ 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`

97. __d__ Which answer 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`

98. __b__ 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`

99. _c__ You can continue entering a long command on the next line by entering what character?

a. `;`

b. `\`

c. `/`

d. `-`

100.__b__ You can include several commands on the same command line by separating them with what character?

a. `\`

b. `;`

c. `,`

d. Tab

101.__d__ What command determines which processes are currently in execution, including those processes not connected to a terminal?

a. `stat`

b. `jobs`

c. `bg`

d. `ps -e`

102.__c__ The default signal sent by the `kill` command is:

a. `INT`

b. `KILL`

c. `TERM`

d. `USR1`

103.__d__ What character introduces comments in Korn shell scripts?

a. `:`

b. `\`

c. `;`

d. `#`

104.__b__ 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

105.__b__ What command could 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"`

106.__d__ Which word ends a conditional loop in the Korn shell?

a. `end`

b. `enddo`

c. `endif`

d. `done`

107.__c__ Which command copies files from a remote UNIX system to your local UNIX system over a TCP/IP network?

a. `copy`

b. `dcp`

c. `rcp`

d. `finger`

108.__d__ Which command logs you 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`

109. __b__ Which file would you edit to designate a different printer as your default printer?

a. `/etc/printcap`

b. `.dtprofile`

c. `.profile`

d. `.login`

110. __c__ Which environment variable sets the value for your default printer?

a. `PTRNAME`

b. `DISPLAY`

c. `LPDEST`

d. `PTRDEVICE`

111. __c_ 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

112. __d__ Which command provides status information for printer lp1?

a. `lpq -Plp1`

b. `lpstat -t`

c. `lpc status`

d. All of the above

113. __d__ 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

114.__a__ What command reads and writes tapes for use on a non-UNIX system such as OpenVMS, and can convert EBCDIC and ASCII files?

a. `dd`

b. `tar`

c. `cpio`

d. `mt`

115.__d__ What command can move tape forward and backward by number of files or records, and also rewinds tape?

a. `df`

b. `du`

c. `dd`

d. `mt`

116.__c__ What command displays the amount of free disk space?

a. `du`

b. `dd`

c. `df`

d. `ls -l`

117.__b__ What command displays the size of files in directories?

a. `du -s`

b. `du`

c. `df`

d. `dd`

Answers

Appendix A

stty Command Arguments

Using stty Command Arguments

Overview

Use the arguments in the following tables to set the terminal characteristics.

Table 18-1: 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 18-2: stty Command Arguments: Input Modes

Argument	Function
`echoctl (-echoctl)`	Echoes control characters as `^X` and `Backspace` 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 18-3: 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 18-4: stty Command Arguments: Local Mode

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) `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

Table 18-4: stty Command Arguments: Local Mode (Continued)

Argument	Function
`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`
`special_character string`	Sets *`special_character`* to *`string`*. The *`string`* 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 `Backspace`. 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 current terminal characteristics to saved settings produced by `-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 18-5: 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 18-6: 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`

Table 18-6: stty Command Arguments: Compatibility Mode (Continued)

Argument	Function
`echoke (-echoke)` or `nohang (-nohang)`	Sends (does not send) HANGUP signal if carrier drops.
`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 TeletypeCorporation 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.

- 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

Appendix B

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 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 function 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.

Appendix C

Korn Shell Command Line Editing Commands

Using emacs and vi 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 18-7, Fetching Commands from the History File
- Table 18-8, Command Execution
- Table 18-9, Moving the Cursor
- Table 18-10, Deleting or Replacing Characters, Words, and Lines
- Table 18-11, Entering `vi` Insert Mode
- Table 18-12, Using Uppercase and Lowercase Letters
- Table A-7, Pathname Expansion or Completion
- Table 18-14, 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 18-7: 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 0Ctrl/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 0Ctrl/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 18-8: 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 18-9: 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` `[`*`count`*`]E`
Move cursor to next character	`Ctrl/e`	`[`*`count`*`]e` `[`*`count`*`]E`
Move cursor to beginning of next word	—	`[`*`count`*`]w;` `[`*`count`*`]W`
Move cursor to next character *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: letter, 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 18-10: 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	Backspace	[*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 Backspace 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 (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*

Table 18-10: Deleting or Replacing Characters, Words, and Lines (Continued)

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

1.susp cannot be defines as Ctrl/Y

Table 18-11: 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 18-12: 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 countwords) to lowercase	Esc [*count*]Esc L	—
Change case of count characters	—	[count]~

1.INTR cannot be defined as Ctrl/C.

Table 18-13: 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 18-14: 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, or [*`count`*] words 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-`
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 the Korn shell.	`Ctrl/V`	—

Appendix D

Using the emacs Editor

Overview

Introduction

`emacs` is the most powerful editor in the UNIX world today, and is a complete working environment. You can use `emacs` to:

- Get a listing
- 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

This appendix introduces the `emacs` editor.

Resources

For more information on the topics in this appendix, see the following:

- `emacs` reference pages
- *GNU emacs Manual*
  Richard Stallman
  available in the `/usr/lib/emacs/doc` directory when you install the FSFemacsSRC software subset
- *Learning GNU emacs*
  Debra Cameron and Bill Rosenblatt
  published by O'Reilly and Associates, Inc., Sebastopol, CA (c) 1991
  ISDN 0-937175-84-6

Introducing the emacs Editor

Overview

The original emacs was written by Richard Stallman for the PDP-10 as a set of macros for the TECO line editor. 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.

The emacs software and documentation is free. The emacs software is often packaged with distributions of UNIX systems; Tru64 UNIX includes GNU emacs in the OSFemacs software subset (and documentation and sources in the FSFemacs SRC subset).

The documentation for GNU emacs is available online and can be viewed using Info, a subsystem of the emacs facility. The emacs editor has an extensive interactive help facility, however, you must know how to manipulate emacs windows and buffers to use it.

Using emacs Files and Buffers

emacs does not edit the actual file, but puts the contents of the file into a temporary buffer and edits that buffer. Any changes are made to the buffer, not the original file. The file on disk is not modified until the file is written. The buffer name is usually the same as the file name.

emacs may have many buffers active at one time. One of the buffers is the help buffer, *Help*. You can open more than one buffer, and move back and forth between buffers at will.

emacs Modes

emacs has different operating **modes**. The modes allow emacs to operate differently based upon the contents of the file being edited. There are major modes and minor modes. Some examples of major modes are shown in the table.

Mode	Function
Fundamental	Default mode; no special operation
Text	Writes text
Indented text	Indents all the text you write
Picture	Creates simple line drawings
C	Writes C programs
FORTRAN	Writes FORTRAN programs
emacs LISP	Writes emacs LISP functions

emacs selects the major mode based upon the file extension or the contents of the file. If it cannot determine a major mode, it defaults to the fundamental mode.

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 shown in the table.

Mode	Function
Abbrev	Allows the use of word abbreviations
Fill	Enables word wrap
Overwrite	Allows typing over characters instead of inserting them
Autosave	Saves your file automatically every so often

Starting emacs

On a UNIX workstation, you can start emacs within the current terminal window, or in a separate X window as a background process.

Starting emacs in a Terminal 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; perform the editing in the present window

For example, to start emacs on a terminal without using an initialization file:

```
$ emacs -q junkfile
```

Starting emacs in an Xterm Window

The format of the emacs command is:

```
emacs [-option] filename &
```

Some of the options you may want to use with a workstation are shown here.

Option	Function
`-wn` *`name`*	Specifies the name of the `emacs` window
`-i`	Displays the **kitchen sink** bitmap icon (looks like a deep sink) when making the `emacs` window an icon
`-geometry` *`geometry`*	Sets the `emacs` window's width, height, and position. The width and height are specified in characters, and the default is 80 by 24
`-fg` *`color`*	Sets the color of the text on color displays; colors can be found in the file `/usr/lib/X11/rgb.txt`
`-bg` *`color`*	Sets the color of the window's background on color displays

Using 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**, the area of the screen for the text
- A **mode line**, two lines up from the bottom of the screen
- The **minibuffer** area, just below the mode line

The cursor marks the place where the next character will be entered, and you can start typing right away. `emacs` does not have separate modes for inserting text and giving commands.

This example shows a typical mode line, containing the following information.

- 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 the `emacs` editor.
- The name of the buffer you are editing, `junkfile`.
- 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.
- Where you are in the file (Top, Bot, All, or % into the file).

Example 18-1: emacs Mode Line

```
--**-emacs: junkfile          (Fundamental)----------------------Top--------
```

Below the mode line is the minibuffer. This is the area where:

- `emacs` echoes the commands you enter
- You specify file names for `emacs` to find

- You specify values for `emacs` to search for
- You type the name of a command to be executed

Introducing emacs Commands

Before talking about specific `emacs` commands, we will discuss `emacs` commands from a more general point of view.

- There are hundreds of `emacs` commands, but you need to learn only a few to get started.
- `emacs` commands have long command names. However, the commands used most often are bound to key sequences.
- You can always use the command name to execute a command, even if it is bound to a key.
- The most commonly used commands (cursor movement commands) are bound to Ctrl/*char*.
- Slightly less used commands are bound to Esc *char* key sequences. If you do not have an Esc key, try the F11 key or Ctrl/ [.
- `emacs` allows you to bind a command to a sequence of keys, something you should do after using `emacs` awhile.

You can execute any command by pressing Esc X then typing the command name. This is the normal way to execute the commands that do not have key sequences bound to them. An example is the goto-line command, which would be executed as: Esc X `goto-line 100`

Leaving emacs

After editing the buffer, you can temporarily suspend `emacs` by pressing Ctrl/Z. This places you back at the shell prompt. You can get back into `emacs` by typing `fg`. Suspending `emacs` is very useful, but be careful not to start another session of `emacs` on the same file; you could corrupt the file. You should get in the habit of saving your buffer before suspending `emacs`.

Terminate the `emacs` session by pressing Ctrl/X Ctrl/C. If you 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` forces you to verify that the changes should be discarded.

Now Try This!

1. Start the `emacs` editor with the `emacs -q file5` command.
2. Note the mode line near the bottom of the window specifying the file name.
3. Type this line: `Now is the time to say goodbye`
4. Terminate `emacs` by pressing Ctrl/X Ctrl/C.

5. Save your file when prompted.

Need Help?

If the `emacs` command is not found, check with your system administrator that the `emacs` software is installed.

Solution

The `emacs` editor should start up with this line at the bottom:

```
-----emacs: file5           (Fundamental)--All-------------------------------
```

Once you start typing, the line should change to:

```
--**-emacs: file5           (Fundamental)--All-------------------------------
```

Using the emacs Help Facility

Overview

emacs has one of the most useful, and complete help facilities of any text editor. emacs help consists of:

- Completion: emacs helps you complete typing the names of things
- The help key: allows you to get help on many emacs topics

Name Completion

You can type in the shortest, unambiguous sequence of characters and tell emacs to figure out the rest of the name. The partial name has to contain enough characters, starting from the beginning, to distinguish it from any other name. emacs can complete:

- 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. When you type in a name, press 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, you might have a C program called poker.c in your default directory. When you press Ctrl/X Ctrl/F to visit the file, emacs prompts for the file name. The following are the responses you would get if you type pok and use name completion.

Press Tab	emacs completes the file name poker.c if no other files begin with pok
Press spacebar	emacs completes the file name to poker
Press ?	emacs generates a *Completion* window and provides the names of all files that start with pok

Now Try This!

1. Start the `emacs` editor with the `emacs -q file5` command.
2. Move down by pressing the down arrow and type a second line: `This is the second line`
3. Use name completion to help save the file. Type Ctrl/X Ctrl/W `fi` Tab

   The minibuffer should show the name `file5`. Press Return.
4. Terminate `emacs` by pressing Ctrl/X Ctrl/C.

Solution

When you press Ctrl/X Ctrl/W, the `emacs` minibuffer shows: Write file: ~/

When you type `fi` and press Tab, `emacs` should complete the file name.

Getting emacs Help

`emacs` provides help through the `emacs` help key, Ctrl/H, followed by the character that represents the type of help you want. Some of the help commands are shown in the table.

Command	Function
Ctrl/H a *expr*	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 f *cmd-name*	Provides a description of the command provided by *cmd-name*.
Ctrl/H k *key-sequence*	Provides the name of the command executed by *key- sequence*, and provides a description of the command.
Ctrl/H w *cmd-name*	Provides the key sequence, if any, bound to *cmd-name*.

If you cannot remember what to type to get help on a particular topic, type Ctrl/H Ctrl/H. `emacs` opens a *Help* window which provides a short description of all the help options.

Here are some general information help key sequences.

Ctrl/H i	Starts the information documentation reader
Ctrl/H t	Provides an `emacs` tutorial

Now Try This!

1. Start the `emacs` editor with the `emacs -q file5` command.
2. Start the `emacs` tutorial by typing: Ctrl/H t.
3. Follow the steps in the tutorial.

Need Help?

If Esc or F11 do not work as escape, use Ctrl/ [.

You may also need to turn off Ctrl/S and Ctrl/Q as flow control commands, or they cannot be used as defined by `emacs`.

Performing Basic Editing

Reading and Writing Files

You can read in a file other than the one you are editing, insert a file into the buffer, or save a modified file.

Command Sequence	Function
Ctrl/X Ctrl/F *file* find-file command	Finds and reads a file
Ctrl/X `i` *file* insert-file command	Inserts the file at the cursor
Ctrl/X Ctrl/S save-buffer command	Saves the buffer in the file with the same name; if Ctrl/S is used for keyboard flow control, this key sequence will not work
Ctrl/X Ctrl/W *file* write-file command	Writes the buffer to the file

Now Try This!

1. Edit a new file with: `emacs -q file6`
2. Insert `file5` with: Ctrl/X `i file5`
3. Add a third line: `This is file6`
4. Save the file with: Ctrl/X Ctrl/S
5. Read `file5` with: Ctrl/X Ctrl/F `file5`

   This switches to the `file5` file.

Need Help?

If Ctrl/S is used for keyboard flow control, this key sequence will not work. You must turn off flow control, or assign another key sequence.

You can use the command name with Esc X `save-buffer`.

You can also use the Ctrl/X Ctrl/W `file6` sequence to save the file.

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.

Do not turn fill mode on until after you have completed the `emacs` tutorial lab.

Repeating Commands

You may often want to repeat a command a number of times, especially when it comes to cursor motion. `emacs` provides two ways to repeat commands.

- Precede the command with Esc *number*, where *number* is 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. Ctrl/U *number* works the same as Esc *number*. Ctrl/U without a number uses the number four.

Moving the Cursor

There are many commands to move the cursor. To move the cursor one character or line at a time, use the commands shown here.

Command	Function
Ctrl/F or right arrow	Moves forward one character (right)
Ctrl/B or left arrow	Moves backward one character (left)
Ctrl/P or up arrow	Moves to previous line (up)
Ctrl/N or down arrow	Moves to next line (down)

Some other cursor movement commands include those shown here.

Command	Function
Ctrl/A	Moves to the beginning of the line
Ctrl/E	Moves to the end of the line
Ctrl/V	Moves forward one screen
Esc V	Moves backward one screen
Esc >	Moves to the end of the file
Esc <	Moves to the beginning of the file
Esc X `goto-line` *n*	Goes to line *n* in the file

Now Try This!

1. Create a file with: `man emacs | sed 's/.<Ctrl/V><Ctrl/H>//g' > emacs.txt`
2. Edit the file with: `emacs -q emacs.txt`
3. Move the cursor down to the 17th line (hint: press down arrow 17 times or press Esc 17 <down arrow>)
4. Move to the end of the line with Ctrl/E

5. Move to the beginning of the line with Ctrl/A
6. Scroll forward one screen with Ctrl/V
7. Scroll back one screen with Esc V
8. Go to the end with Esc >
9. Go to the beginning with Esc <

Marking Text

To copy or delete a phrase, several sentences, or parts of two paragraphs, using the line deletion command is unwieldy. It is often easier to mark the area you want to delete. To define a region, use a secondary pointer called a **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/spacebar or Ctrl/@, then move the cursor to the other end of the region.

Use Ctrl/X Ctrl/X to swap the positions of point and mark because:

- You cannot see the location of the mark
- The text in the marked region is not highlighted
- You should verify that the region is really marked properly

Deleting Text

`emacs` provides many ways to delete text, the easiest being the use of the backspace key . Some `emacs` delete commands are shown in the table.

Command	Function
Ctrl/D	Deletes the character under the cursor
Backspace	Deletes the previous character
Esc Z *char*	Deletes up to next occurrence of *char* (including that character)
Ctrl/K	Deletes from the cursor to the end of the line
Esc K	Deletes from the cursor to the end of the sentence
Ctrl/Y	Restores the last deletion
Ctrl/W	Deletes a marked region
Esc W	Copies a marked region

Moving or Copying Text

Sometimes you need to move text. 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:

1. Place the cursor at the beginning of the region.
2. Set a mark using Ctrl/spacebar. `emacs` responds with "Mark set" in the minibuffer region.
3. Move the cursor to the end of the region you want to move.
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.

To copy a marked region rather than cut it, use Esc W rather than Ctrl/W.

Now Try This!

1. Edit the file with: `emacs -q emacs.txt`
2. Move to the 22nd line (hint: Esc 22 down arrow)
3. Mark this paragraph with Ctrl/spacebar.
4. Move to the end of this paragraph.
5. Use Ctrl/X Ctrl/X to verify the mark.
6. Delete the region with Ctrl/W.
7. Move to the blank line below the next paragraph.
8. Use Ctrl/Y to restore the region.

emacs Kill Ring

When you delete and copy text, the deleted or copied region of marked text is placed in the **kill ring**. The kill ring:

- Stores the past 30 deletions of one word or more.
- A yank (Ctrl/Y) command removes the most recently deleted text from the kill ring.
- If you made several deletions, you can retrieve all the deletions from the kill ring:
  - — The Esc Y (yank-pop) command retrieves successive deletions from the kill ring.
  - — It can only be used immediately after a yank (Ctrl/Y) command.

Now Try This!

1. Edit a new file with: `emacs -q file7`
2. Add 5 lines:

```
line 1
line 2
line 3
line 4
line 5
```

3. Go to the top of the buffer and use Ctrl/K to delete each line.
4. Use Ctrl/Y to restore the last deletion. This should be line 5.
5. Use Ctrl/Y Esc Y to restore the previous deletion. This should be line 4.
6. Use Ctrl/Y Esc Y Esc Y to skip a deletion. This should be line 2.
7. Use Ctrl/Y Esc Y Esc Y again. It circles back to line 5.

Undoing Changes

All too often we make typing mistakes, or execute a command that we did not mean to execute. A command can be aborted with the keyboard-quit command, Ctrl/G. `emacs` indicates that a command has been stopped with "Quit" in the minibuffer.

Other undo commands:

Ctrl/X u	Undoes the last edit
Esc X `revert-buffer`	Restores the buffer to the state when the file was last saved (or autosaved)

Use the `revert-buffer` command to restore the buffer to the state as it is stored on disk, or from the last autosave. Autosave file names are marked in your directory with a leading and trailing number sign (#). For example, if you are editing the file `junkfile`, and it has been autosaved, the autosave file is `#junkfile#`.

Searching and Replacing

emacs Search Operation

Like any good editor, emacs provides search commands. In fact, there are three different search commands, as shown in this table.

Search Command	Function
Incremental search	As soon as you type a character, emacs starts searching for the next occurrence of that character. As you type more characters, it searches for that combination of characters.
Nonincremental or **simple** search	Provide emacs with a search string and it will find the next occurrence. The search terminates at the found string, or fails if the string is not found.
Word search	Like a simple search, except that it only searches for full words and phrases. For example, if you are searching for the word **mental**, you do not have to worry about finding the word **incremental**.

emacs searches are, by default, not case sensitive. If you search for **home**, emacs will find **Home**, **HOME**, and **hoMe**.

Incremental Search

To do an incremental search:

1. Press Ctrl/S. emacs displays "I-search" in the minibuffer.
2. Type the first character to search for. emacs displays that character in the minibuffer and moves to its first occurrence.
3. Type the next characters in the string. emacs displays the characters and finds the first match.

Notes:

- Press Ctrl/S to search again forward.
- Terminate the search by pressing Esc or cursor motion commands that do not start with Esc, such as arrow keys.
- Press Ctrl/G to cancel a nonfailing search and move the cursor back to the original position.
- To change the search string, press the backspace key and type the correct characters.
- If the search fails to find the character string you have typed in, emacs echoes "Failing I-Search" in the minibuffer.
- Use Ctrl/R to search backward.

Simple Search

To do a simple search:

1. Press Ctrl/S. `emacs` displays "I-search" in the minibuffer.
2. Press Esc. `emacs` displays "Search" in the minibuffer.
3. Type the characters in the string. `emacs` displays the characters and finds the first match.

Notes:

- Use Ctrl/R Esc *string* to start a simple search backward.

Word Search

To do a word search:

1. Press Ctrl/S. `emacs` displays "I-search" in the minibuffer.
2. Press Esc. `emacs` displays "Search" in the minibuffer.
3. Press Ctrl/W. `emacs` displays "Word Search" in the minibuffer.
4. Type the characters in the string. `emacs` displays the characters and finds the first match.

Notes:

- Use Ctrl/R Esc Ctrl/W string to start a word search backward.

Now Try This!

1. Edit the file with: `emacs -q emacs.txt`
2. Press Ctrl/S. `emacs` displays "I-search" in the minibuffer.
3. Type slowly and watch the minibuffer display and the cursor: `editing`
4. Press Ctrl/S again to find the next occurrence.
5. Press Ctrl/G to cancel the search.
6. Press Ctrl/S and type `emaq`. `emacs` displays "Failing I-search: emaq" in the minibuffer.

Need Help?

If Ctrl/S is used for keyboard flow control, this key sequence will not work. You must turn off flow control, or assign another key sequence. If you are using the `emacs` initialization file, use Ctrl/\ instead of Ctrl/S.

Search and Replace

You may want to combine a search with a replace operation. For example, you may have problems when typing certain words, like "teh" instead of "the," or often misspell a word like "recieve" instead of "receive." `emacs` provides search and replace operations that allow you to make changes like this very quickly.

`emacs` has two types of search and replace operations:

- Simple search and replace: Replaces every occurrence
- Query replace: Verifies each occurrence before replacing

Search and replace, and query replace operate only in the forward direction.

Simple Search and Replace

To replace every occurrence of one string with another, use the simple search and replace. The command format is:

```
Esc X replace-string <Return> old-string <Return> new-string
<Return>
```

1. Press Esc X, type the `replace-string` command, and press Return. `emacs` displays "Replace string:" in the minibuffer.
2. Type the string to find and press Return. `emacs` displays "with:" in the minibuffer.
3. Type the replacement string and press Return. `emacs` finds and replaces any occurrences, then says "Done" in the minibuffer.

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.

Query Replace

If you are not sure that you want to replace each and every occurrence, use query replace Esc %. It lets you decide on text replacement on a case-by-case basis.

1. Press Esc %. `emacs` displays "Query replace:" in the minibuffer.
2. Type the string to find and press Return. `emacs` displays "with:" in the minibuffer.
3. Type the replacement string and press Return. `emacs` finds the next occurrence and waits for your response.

y or spacebar	Replaces old-string with new-string; searches for next occurrence of the string
n or backspace	Does **not** replace; goes to next instance
q or Return	Quits

.	Replaces the current instance and quits
,	Replaces the current instance, but does not move on
!	Replaces all remaining occurrences without asking for confirmation
^	Backs up to the previous instance

Now Try This!

1. Edit the file with: `emacs emacs.txt`
2. Press Esc %. `emacs` displays "Query replace:" in the minibuffer.
3. Type `editor` and press Return.
4. Type `elephant` and press Return.
5. Replace the first two occurrences (press **y**) and skip the next five (press **n**).
6. Press ^ to back up to the previous occurrence.
7. Press ! to replace the remaining occurrences.

Recursive Edit

While you are doing a query replace, you may see something else that needs fixing. Try and remember the change, or write it down, so you can go back and fix it later. When you finish the query replace, if you did not make a note, many times you will not remember what else 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 remember it later.

Type Ctrl/R to start a recursive edit, then use the normal `emacs` commands to move the cursor around and make changes. When the changes are complete, use Esc Ctrl/C or Ctrl/] to exit recursive edit and 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 are placed around the modes on the mode line, for example, [(Fundamental)].

Ctrl/R	Enters recursive edit
Esc Ctrl/C	Exits recursive edit
Ctrl/]	Exits recursive edit and exits query replace

Using Buffers and Windows

Overview

One of the most useful features of `emacs` is its 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 create 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 often only one or two buffers are 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 be able to manipulate them.

Some buffer commands are listed here.

Command	Function
Ctrl/X b *buffer-name*	Moves to the specified buffer
Ctrl/X Ctrl/B	Displays the buffer list
Ctrl/X k *buffer-name*	Deletes the specified buffer
Ctrl/X s	Asks if you want to save each modified buffer
Esc X kill-some-buffers	Asks before deleting a buffer
Esc X rename-buffer	Changes the name of the buffer to the specified name

Getting a List of Buffers

If an `emacs` editing session has been active for a long time, the list of buffers can grow so long you cannot keep track of all of them. You can get a list of buffers by pressing Ctrl/X Ctrl/B. `emacs` creates another window on the screen and creates the buffer *Buffer List*. This example shows a sample buffer list.

Example 18-2: Buffer List

```
MR Buffer          Size  Mode           File
-- ------          ----  ----           ----
.% docs            6994  Fundamental    /ProjX/docs
 * contacts        737   Fundamental    /ProjX/contacts
.* proj-dlist      31    Fundamental    /ProjX/distribution
 * distribution    28    Fundamental    /ProjX/distribution
    ordering-emacs 12253 Fundamental     /usr/local/ordering-emacs
   *Help*          517   Fundamental
 * *Buffer List*  252   Buffer Menu
   *scratch*       0     Lisp Interaction

--%%-emacs: *Buffer List*         (Buffer Menu)----All-----------------------
```

The MR (Modified or Read-Only) column shows the buffer status.

Table 18-15: 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

Press Ctrl/X `o` to move the cursor to the window containing the buffer list, then use these commands to mark buffers for saving or deletion, or to select buffers for display.

Table 18-16:Buffer List Commands

Type	To:	It will happen:
Down arrow	Move the cursor to the next buffer in the list	Immediately
Up arrow	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, k, s, and u)	Immediately
`~`	Mark the buffer as unmodified	Immediately
`1`	Display this buffer in a full screen	Immediately
`f`	Replace the buffer list with this buffer	Immediately
`o`	Replace the other window with this buffer	Immediately
`m`	Mark a buffer to be displayed in a window	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`, place the cursor on the line containing the name of the other buffer, then press `q`.

Saving Buffers

We have already discussed how to save buffers with:

- Ctrl/X Ctrl/S (save-buffer command)
- Ctrl/X Ctrl/C (save-buffers-kill-`emacs` command)

You can also save buffers by pressing Ctrl/X s. `emacs` cycles through all modified buffers, and asks if you want to save the buffer. The buffer is saved if you type **`y`**.

Killing Buffers

You can kill the buffer you are currently editing by pressing Ctrl/X k. 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 you to verify.
- If you provide a buffer name, `emacs` prepares to kill the named buffer as it would the default buffer.

Use Esc `X kill-some-buffers` to selectively kill buffers. `emacs` steps through the buffer list, and informs you if a buffer has been modified to allow you to keep the buffer. It then asks if you want to kill the buffer. If you respond **yes**, `emacs` gives you another chance to change your mind by informing you that the file has been modified, and asks if you want to kill the buffer anyway.

`emacs` even lists its internal buffers, allowing you to kill them if you want.

Renaming Buffers

Imagine the situation where you have multiple directories, each containing a file by the same name, `distribution-list`. If you edit this file from one directory, use Ctrl/X Ctrl/F to bring in a file with the same name from another directory, `emacs` names the buffer for the second file `distribution-list2` 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 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 and save the files, `emacs` saves them to the proper files.

Read-Only Buffers

You will occasionally want to copy text from one file to another without changing the file you are copying from. It is easy to hit the wrong keys and unknowingly add text. You may want to make a buffer read only, preventing 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

Now Try This!

1. Edit `file5` with: `emacs file5`
2. Go to file6 with: Ctrl/X Ctrl/F
3. List the buffers with: Ctrl/X Ctrl/B

4. Move to the buffer list window with: Ctrl/X
5. Move down to `file5` by pressing the down arrow
6. Display this buffer in the full screen with: `1`

Solution

The buffer list initially looks like this:

```
 MR Buffer         Size   Mode           File
 -- ------         ----   ----           ----
.   file6          68     Text          file6
    file5          54     Text          file5
    *scratch*      0      Lisp Interaction
 *  *Buffer List*  240    Text
```

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. You can also have all 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.
- `emacs` windows can be split horizontally or vertically. We will not discuss vertical windows as few will find it useful.
- Use Ctrl/X `2` to divide the current window horizontally into two windows.
- Use Ctrl/X `o` (other window command) to move the cursor from one window to another. The cursor moves down the screen, one window at a time.
- Use Esc *number* Ctrl/X `o` to move more than one window at a time. Esc 3 Ctrl/X o moves the cursor to the third window from the present window.
- Use Esc *number* Ctrl/X `^` to enlarge the window that number of lines.
- Use Ctrl/X `0` (zero) to delete the window the cursor is in.
- Use Ctrl/X `1` (one) to delete all windows except the window the cursor is in.

Now Try This!

1. Create several small files as follows:

```
$ cat > file1
file1
<Ctrl/D>
$ cat > file2
file2
<Ctrl/D>
$ cat > file3
file3
<Ctrl/D>
```

2. Edit the first file with: `emacs file1`
3. Split the screen into two windows with: Ctrl/X `2`
4. Edit `file2` with: Ctrl/X Ctrl/F `file2`
5. Switch to the other window with: Ctrl/X `o`
6. Split that window with: Ctrl/X `2`
7. Edit `file3` with: Ctrl/X Ctrl/F `file3`
8. Delete all other windows with: Ctrl/X `1`

Correcting Mistakes

Transposing Characters and Words

One of the most common typing mistakes is transposing characters. `emacs` provides commands to transpose characters, words, sentences, or paragraphs, as shown here.

Command	Function
Ctrl/T	Transposes two characters, the character the cursor is on and the previous
Esc T	Transposes two words, the two words the cursor is between
Ctrl/X Ctrl/T	Transposes two lines, the line the cursor is on and the previous
Esc X transpose-sentences	Transposes two sentences, the sentence the cursor is on and the next
Esc X transpose-paragraphs	Transposes two paragraphs, the paragraph the cursor is on and the next

Which characters, words, and lines are transposed depends upon cursor position.

Capitalization

Another mistake users often make is in capitalization. `emacs` provides special commands for fixing capitalization problems.

Esc C	Capitalize the first letter of the word
Esc U	Uppercase the word
Esc L	Lowercase the word

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 workstation, you can move the cursor, cut and paste, and manipulate `emacs` windows with the mouse. The following table provides direction for using the mouse with `emacs`.

Table 18-17:Using the Mouse with emacs

Mouse Click	Equivalent emacs Command
MB1	Moves cursor to location mouse points to
MB2 or Shift/MB3	Yank, Ctrl/Y

Table 18-17:Using the Mouse with emacs (Continued)

Mouse Click	Equivalent emacs Command
MB3 or Shift/MB2	Copy-region-as-kill, Esc W
Ctrl/MB2	Kill-region, Ctrl/W
Ctrl/MB3	Split-window-vertically, Ctrl/X 2
Ctrl/Shift/MB3	Delete-other-windows, Ctrl/X 1
Ctrl/Shift/MB1	Creates a pop-up menu with buffer names, similar to Ctrl/X Ctrl/B, list-buffers; click any mouse button on buffer name to select that buffer
Ctrl/Shift/MB2	Creates a stack of pop-up menus for `emacs` help

Summary

Introducing the emacs Editor

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

Terminate the `emacs` session by pressing Ctrl/X Ctrl/C.

Using the emacs Help Facility

`emacs` has a very extensive and useful Help facility. You can get 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.

Performing Basic Editing

There are many `emacs` commands, such as:

- Use the sequence Ctrl/X Ctrl/F *file* (find-file command) to read a file.
- Precede a command with Esc *number* to repeat the command.
- Move the cursor with the arrow keys.
- Set a mark at one end of a region with Ctrl/spacebar or Ctrl/@, then move the cursor to the other end of the region and press Ctrl/X Ctrl/X.
- Ctrl/W deletes a marked region.
- Esc W copies a marked region.
- Ctrl/Y yanks the deleted or copied text into the buffer.
- Ctrl/X u undoes the last edit.

Searching and Replacing

`emacs` provides commands to search for a text string and optionally replace it. Start a search with Ctrl/S.

Using Buffers and Windows

`emacs` allows you to edit multiple buffers at one time. A window on the screen displays a buffer.

- Ctrl/X Ctrl/B lists the buffers.
- Ctrl/X b *buffer-name* switches buffers.
- Ctrl/X 2 splits the window.
- Ctrl/X o moves the cursor to the other window.
- Ctrl/X 1 deletes all but one window.

Correcting Mistakes

`emacs` provides commands to transpose characters, words, sentences, or paragraphs. `emacs` provides commands to change the capitalization of text.

Exercises

1. Using the `emacs` tutorial:

   a. Start `emacs` with `junkfile`, then start the `emacs` tutorial with Ctrl/H T. Only the TUTORIAL buffer will be visible. Follow the steps in the tutorial.

      Remember, you may be using Ctrl/ \ for incremental searches instead of Ctrl/S.

      Also remember, if Esc does not work, try 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. Some odd combination of characters, like qqq, is easy to remember. Use Ctrl/X Ctrl/C to exit the `emacs` editor. When `emacs` asks you if you want to save the file ~/TUTORIAL, type `yes`. `emacs` will save the file in your home directory.

      When you want to resume the tutorial at the point you left off, edit the file TUTORIAL that you saved. When the mode line is on the screen, use incremental search (Ctrl/S) to find the qqq, or whatever you used as a marker. Use Esc to terminate the search when `emacs` finds the qqq. You are now ready to resume the tutorial.

   b. When you have completed the tutorial portion of the lab, exit `emacs` with Ctrl/X Ctrl/C. This time, type `no` when `emacs` asks if you want to save the file TUTORIAL. You can always get back into the tutorial with Ctrl/H T.

2. Cutting and pasting from one file to another:

   a. Edit `junkfile` again, and start the tutorial.

   b. Create a second window with Ctrl/X 2.

   c. Use Ctrl/X B Return to switch to the buffer `junkfile`, which is the default. You should now have `junkfile` in one window, and the tutorial in the other window.

   d. Switch the cursor to the window containing the tutorial with Ctrl/X O.

   e. Use incremental search, Ctrl/S or Ctrl/ \ , to find the text string "several windows".

   f. 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.

   g. Set the mark with Ctrl/spacebar.

   h. Move the cursor to the end of the paragraph with Esc].

i. 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.

j. Delete the marked text with Ctrl/W.

k. Move the cursor to the other window with Ctrl/X O.

l. Yank the text into the buffer from the kill ring with Ctrl/Y.

3. Using Query Replace and Recursive Edit

a. Move the cursor to the other window (TUTORIAL) with Ctrl/X O.

b. Move the cursor to the top of the tutorial with Esc <.

c. Use Esc % to query replace "M-" with "Esc-".

d. 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...". Let's assume that we notice a mistake and want to replace "characters" with "one character".

e. To enter recursive edit, type Ctrl/R.

f. 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..."

g. Exit recursive edit with Esc Ctrl/C. Notice that the cursor moves to the next "M-", right where you left off.

h. Enter y or spacebar to replace a few more occurrences of M-.

i. Enter . to exit the query replace.

Index

Symbols

A

B

C

D

E

O

P

R

S

T

U

V

W

X