Fundamentals of Solaris™ 8 Operating Environment for System Administrators

SA-118

Student Guide

Sun Microsystems, Inc.
MS BRM05-104
500 Eldorado Boulevard
Broomfield, Colorado 80021
U.S.A.

Revision A.2, November 2000

Please
Recycle

Contents

About This Course

Course Goal

The *Fundamentals of the Solaris™ 8 Operating Environment for System Administrators* course teaches you how to use basic Solaris™ Operating Environment commands.

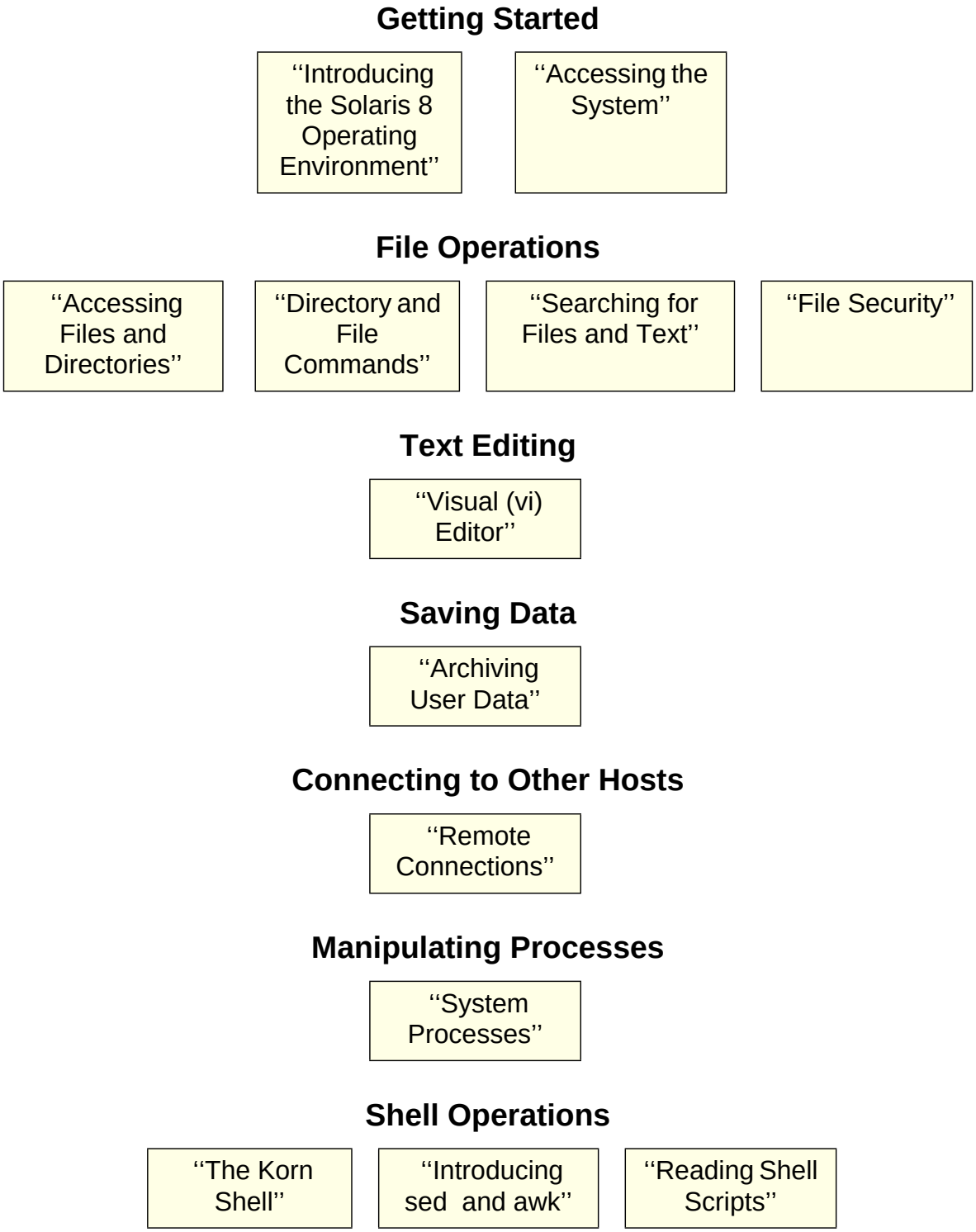
Course Overview

The class is for new users of the Solaris Operating Environment. You will learn fundamental command-line features of the Solaris Operating Environment, including:

- File system navigation
- File permissions
- The `vi` text editor
- Command shells
- Basic network use

Course Map

The following course map enables you to see what you have accomplished and where you are going in reference to the course goal.



Module-by-Module Overview

This course contains the following modules:

- Module 1 – "Introducing the Solaris 8 Operating Environment"

  This module provides a brief introduction to the components of the Solaris Operating Environment.

  Lab exercise – This lab provides a review of the main components of a computer and the Solaris 8 Operating Environment.

- Module 2 – "Accessing the System"

  This module provides instructions on logging in to the system, executing simple commands, and acquiring simple command syntax from online documentation.

  Lab exercise – This lab practices logging in and logging out of the system, as well as using the online documentation.

- Module 3 – "Accessing Files and Directories"

  This module introduces the commands needed to traverse the Solaris Operating Environment directory tree. The concept of absolute and relative path names is introduced as is the use of metacharacters to list files with similar naming characteristics.

  Lab exercise – This lab centers around commands needed to list the contents of specified directories and commands needed to move between directories.

- Module 4 – "Directory and File Commands"

  This module focuses on commands that read the contents of existing files or create and remove new files or directories. Commands to copy or rename files and commands to perform basic print functions are also addressed.

  Lab exercise – This lab addresses copying or renaming files, creating and deleting directories, and manipulating print queues.

- Module 5 – "Searching for Files and Text"

  This module addresses commands used to sort text, compare files, and search through files for regular expressions.

  Lab exercise – This lab demonstrates the use of the `sort`, `cmp`, `diff`, `find`, and `grep` commands.

- Module 6 – "File Security"

  The focus of this module is file and directory permissions.

  Lab exercise – This lab practices use of the commands needed to change file and directory permissions, as well as setting of default permissions through the use of `umask`.

- Module 7 – "Visual (`vi`) Editor"

  This module provides instruction on the use of the `vi` editor.

  Lab exercise – This lab is a guided practice in the use of `vi` commands.

- Module 8 – "Archiving User Data"

  This module introduces various commands and utilities that support archive and compression of user data.

  Lab exercise – This lab demonstrates use of the various archive and data-compression commands and utilities.

- Module 9 – "Remote Connections"

  This module focuses on establishing connections to remote hosts.

  Lab exercise – This lab demonstrates the use of the `rlogin`, `telnet`, and `ftp` commands.

- Module 10 – "System Processes"

  This module introduces basic job control and process management. It also describes the use of signals for controlling process activity.

  Lab exercise – This lab focuses on the `fg` and `bg` commands for job control, the `ps` command for process management, and the use of signals with the `kill` command.

- Module 11 – ''The Korn Shell''

  This module introduces the use of the Korn shell as a command interpreter.

  Lab exercise – This lab demonstrates basic shell operations using the Korn shell.

- Module 12 – ''Introducing sed and awk''

  This module introduces the stream editor (`sed`) and the `awk` command as text processors used for manipulating data and generating reports.

  Lab exercise – This lab uses some simple examples using `sed` and `awk`.

- Module 13 – ''Reading Shell Scripts''

  This module discusses the interpretation of basic shell script programming examples, such as the `if`, `test`, and `case` commands, and the use of shell variables.

  Lab exercise – The lab is a set of practical exercises interpretating basic system administration scripts using the previously introduced shell commands.

Course Objectives

Upon completion of this course, you should be able to:

- Log in and log out of the Solaris Operating Environment and Common Desktop Environment (CDE) systems
- Compose command-line strings to perform Solaris Operating Environment functions
- Navigate the Solaris Operating Environment directory tree
- Create files and directories
- Manipulate text files
- Use commands to search directories and files
- Change permissions of files and directories
- Use the `vi` text editor
- Back up and restore user files and directories
- Use basic network commands
- List active user processes and selectively kill user processes
- Use shell features to streamline command execution
- Identify and modify shell initialization files and read basic shell scripts

Skills Gained by Module

The skills for *Fundamentals of Solaris™ 8 Operating Environment for System Administrators* are shown in the first column of the following matrix. The black boxes indicate the main coverage for a topic; the gray boxes indicate the topic is briefly discussed.

	Module												
Skills Gained	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
Log in and log out of Solaris Operating Environment and CDE systems	■												
Compose command-line strings to perform Solaris Operating Environment functions		■	■	■	■	■	▒	■	■	■	▒	■	
Navigate the Solaris Operating Environment directory tree			■										
Create files and directories			■										
Manipulate text files				■									
Use commands to search directories and files					■								
Change permissions of files and directories						■							
Use the `vi` text editor							■						
Back up and restore user files and directories								■					
Use basic network commands									■				
List active user processes and selectively kill user processes										■			
Use shell features to streamline command execution											■		
Identify and modify shell initialization files and read basic shell scripts													■

Guidelines for Module Pacing

The following table provides a rough estimate of pacing for this course:

Module	**Day 1**	**Day 2**	**Day 3**	**Day 4**
"About This Course"	A.M.			
"Introducing the Solaris 8 Operating Environment"	A.M.			
"Accessing the System"	A.M./P.M.			
"Accessing Files and Directories"	P.M.			
"Directory and File Commands"		A.M.		
"Searching for Files and Text"		A.M.		
"File Security"		P.M.		
"Visual (`vi`) Editor		P.M.		
"Archiving User Data"			A.M.	
"Remote Connections"			A.M./P.M.	
"System Processes"			P.M.	
"The Korn Shell"				A.M.
"Introducing sed and awk"				A.M./P.M.
"Reading Shell Scripts"				P.M.

Topics Not Covered

This course does not cover the following topics; these topics are covered in other courses offered by Sun Educational Services:

- System administration concepts – Covered in SA-238: *Solaris™ 8 Operating Environment System Administration I*
- Detailed shell programming – Covered in SA-245: *Shell Programming for System Administrators*

Refer to the Sun Educational Services catalog for specific information on course content and registration.

Introductions

Now that you have been introduced to the course, introduce yourself to each other and the instructor, addressing the items shown on the overhead.

How to Use the Course Materials

To enable you to succeed in this course, these course materials use a learning model including the following components:

- **Course Map** – An overview of the course content appears in the "About This Course" module so you can see how each module fits into the overall course goal.
- **Objectives** – At the beginning of each module is a list of what you should be able to accomplish after completing the module.
- **Lecture** – The instructor presents information specific to the topic of the module. This information helps you learn the knowledge and skills necessary to succeed with the exercises.
- **Exercise** – Lab exercises give you the opportunity to practice your skills and apply the concepts presented in the lecture.
- **Check Your Progress** – Module objectives are restated, sometimes in question format, so you are sure you can accomplish the objectives of the current module before moving on in the course.

Course Icons and Typographical Conventions

The following icons and typographical conventions are used in this course to represent various training elements and alternative learning resources.

Icons

Additional resources – Indicates additional reference materials are available.

Demonstration – Indicates a demonstration of the current topic is recommended at this time.

Discussion – Indicates a small-group or class discussion on the current topic is recommended at this time.

Exercise objective – Indicates the objective for the lab exercises that follow. The exercises are appropriate for the material being discussed.

Note – This will contain additional important, reinforcing, interesting, or special information.

Caution – This points out a potential hazard to data or machinery.

Warning – This warns of anything that poses personal danger or irreversible damage to data or the operating system.

Typographical Conventions

`Courier` is used for the names of commands, files, and directories, as well as on-screen computer output; for example:

Use `ls -al` to list all files with long listings.
`system% You have mail.`

`Courier bold` is used for characters and numbers you type; for example:

```
system% su
Password:
```

`Courier italic` is used for variables and command-line placeholders that are replaced with a real name or value; for example:

To delete a file, type `rm` *`filename`*.

Palatino italics is used for book titles, new words or terms, or words that are emphasized; for example:

Read Chapter 6 in *User's Guide*.
These are called *class* options.
You *must* be `root` to do this.

Introducing the Solaris 8 Operating Environment 1

Objectives

Upon completion of this module, you should be able to:

- List the four main hardware components of a computer
- Describe the four main components of the Solaris Operating Environment
- State the three main components of the SunOS™ operating system
- Identify the shells available in the Solaris Operating Environment

Additional Resources

Additional resources – The following reference provides additional details on the topics discussed in this module:

- *System Administration Guide,* Part Number 805-7228-10

Introduction to the Solaris Operating Environment

The UNIX® operating system was originally developed at AT&T Bell Laboratories in 1969. It was created as a tool set by programmers for programmers. The early source code was made available to universities all over the country.

Programmers at the University of California at Berkeley made significant modifications to the original source code and called it Berkeley Software Distribution (BSD) UNIX. This version of the UNIX environment was sent to other programmers around the country, who then added tools and code to further enhance BSD UNIX.

Possibly the most important advance made to the operating system by the programmers at Berkeley was the addition of networking software. This allowed the operating system to function in a local area network (LAN).

Sun's original version of the UNIX operating system was known as SunOS, based on BSD UNIX Version 4.2. At that time, AT&T's version of the UNIX environment was known as System V.

In 1988, BSD, AT&T UNIX, and other operating systems were folded into what became System V Release 4 (SVR4) UNIX. This new generation of the operating system was an effort to combine the best features of both BSD and AT&T UNIX, creating an industry standard for the operating system. The new SVR4 became the basis for not only Sun and AT&T versions of the UNIX environment, but also IBM's AIX and Hewlett-Packard's HP-UX.

Main Components of a Computer

The core of all computer systems is the hardware that works with the system software to perform various tasks.

The computer hardware is made up of a number of different components, such as central processing unit (CPU), memory, and disks, each of which has a specific purpose. For these components to work as a team, they require management by the operating system.

The operating system is a collection of programs and files with the primary function of instructing the computer on what to do with the hardware.

The Solaris Operating Environment runs on two types of hardware platforms: SPARC™ and the Intel 32-bit processor architecture (IA).

Hardware Overview

The four main hardware components of a computer are the random access memory (RAM), the CPU, the input/output (I/O) devices, and the hard disk or other mass storage device.

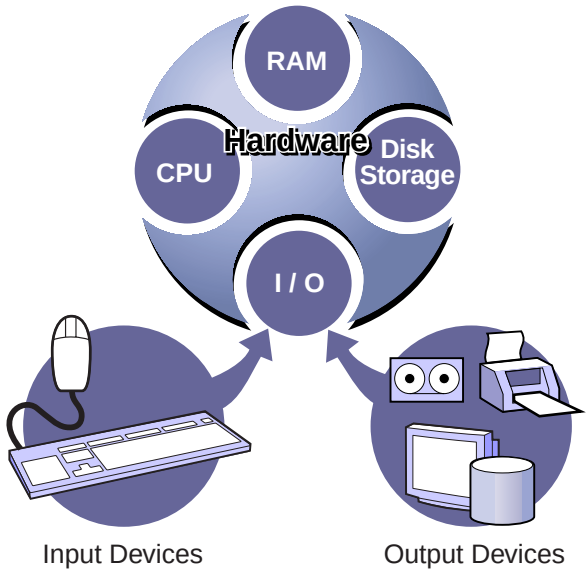


Figure 1-1 Main Hardware Components of a Computer

Random Access Memory

RAM is the main computer memory, which is often referred to as *physical memory*. Programs and data must be loaded into physical memory for the system to process them. The statement, "The system has 64 Mbytes of memory," refers to the amount of RAM or physical memory currently installed.

A software program resides on the *hard disk* and, when activated, an image or copy of that program is loaded into RAM.

Programs remain in RAM as long as needed. When the programs are no longer required, they can be overwritten by copies of other programs. If the system is rebooted or experiences a power loss, all data in physical memory is cleared.

Central Processing Unit

The CPU is the computer logic chip that executes instructions received from physical memory. These instructions are stored in a binary language.

Input/Output Devices

The I/O component reads input from a device, such as a keyboard, into memory, and it writes output from memory to a device, such as a terminal window.

Your input devices include the keyboard and mouse. The monitor, printer, and tape drive are examples of primary output devices.

Hard Disk

The hard disk is a magnetic storage device in which files, directories, and applications are stored.

The Solaris Operating Environment

The Solaris Operating Environment contains the following four components.

- SunOS 5.*x* operating system (based on SVR4 UNIX). This is the heart of the Solaris Operating Environment.
- ONC+™ (Open Network Computing) Technologies, which provide network services. These include Network File System (NFS), which allows file sharing between computers; Network Information System (NIS); and NIS+, which allows the centralization of user accounts and other system information and other underlying technologies.
- Solaris Common Desktop Environment (CDE) graphical user interface (GUI).
- OpenWindows™ graphical environment.

Like all operating systems, SunOS is a collection of software that manages system resources and schedules system operations.

The operating system interprets instructions from the user or an application and tells the computer what to do. It handles input and output, keeps track of data stored on disks, and communicates with peripherals, such as monitors, hard disks, diskette drives, printers, or modems.

The SunOS Operating System

The three main components of the SunOS operating system are:

- The kernel
- The shell
- The directory tree

Note – The directory tree is described in more detail in Module 3, "Accessing Files and Directories."

The Kernel

The kernel is the core of the SunOS operating system. It is the master program that manages all the resources of the computer, including:

- File systems
- Device management
- Process management
- Memory management

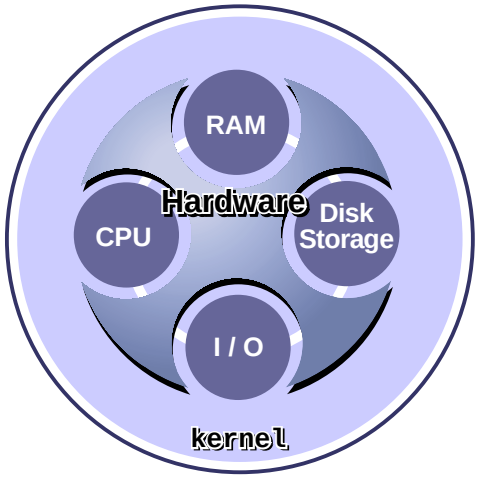


Figure 1-2 The Role of the Kernel

The Shell

The *shell* is an interface between the user and the kernel. The primary function of a shell is to be a command interpreter. That is, the shell accepts commands you enter, it interprets these commands, and then it executes them.

The Solaris Operating Environment supports three primary shells:

- Bourne shell
- C shell
- Korn shell

The Solaris 8 Operating Environment also supports the following shells:

- BASH – The GNU Bourne-Again shell is a Bourne-compatible shell that incorporates useful features from the Korn and C shells.
- Z shell – The Z shell most closely resembles the Korn shell but includes many other enhancements.
- TC shell – The TC shell is a completely compatible version of the Berkeley UNIX C Shell with additional enhancements.

Note – Examples given in this course assume the shell being used is the Korn shell.

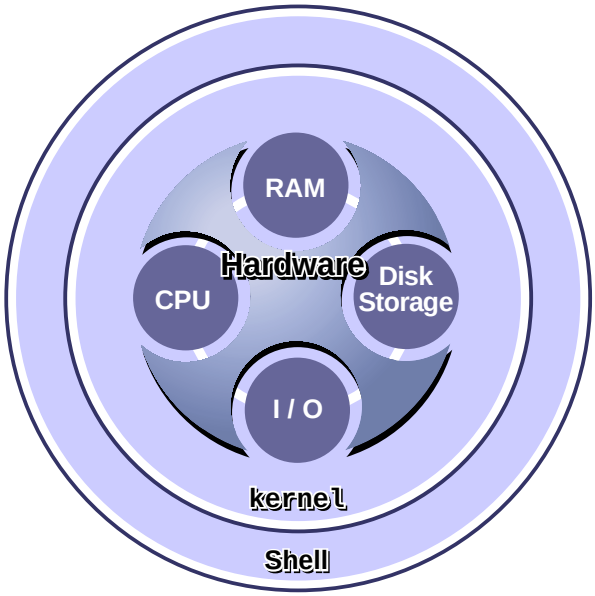


Figure 1-3 The Role of the Shell

The Bourne Shell

The Bourne shell is the original UNIX shell, developed by Steve Bourne at AT&T Bell Laboratories.

It is the recommended shell for programming and is the default shell for the `root` (system administrator) account.

The default Bourne shell prompt for a regular user account is a dollar sign ($).

The C Shell

The C shell was developed by Bill Joy at the University of California at Berkeley.

It is based on the C programming language and has a number of features, such as command line history, aliasing, and job control. This shell has been favored over the Bourne shell by ordinary system users.

The default C shell prompt for a regular user account is the host name followed by a percent sign (`hostname%`).

The Korn Shell

The Korn shell is a superset of the Bourne shell, developed by David Korn at AT&T. This shell had a number of features added to it beyond the enhancements of the C shell.

Additionally, the Bourne shell is almost completely upwardly compatible with the Korn shell, so older Bourne shell scripts can run in this shell.

The Korn shell is considered the most efficient shell and is recommended as the standard shell for regular system users.

The default Korn shell prompt for a regular user account is a dollar sign ($).

Exercise: Using the Solaris 8 Operating Environment

Exercise objective – In this exercise, you review the basics of the computing environment.

Tasks

Answer the following questions:

1. List the four main hardware components of a computer.

   _____

   _____

   _____

   _____

2. Name the three main components of the SunOS operating system.

   _____

   _____

   _____

3. List the three primary shells supported by the Solaris Operating Environment.

   _____

   _____

   _____

4. Match the following terms with their descriptions:

__ Shell	a. Core of the Solaris Operating Environment
__ Kernel	b. Command interpreter

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercises.

- Experiences
- Interpretations
- Conclusions
- Applications

Task Solutions

1. List the four main hardware components of a computer.
   - RAM
   - CPU
   - I/O
   - Hard disk
2. Name the three main components of the SunOS operating system.
   - The kernel
   - The shell
   - The directory tree
3. List the three primary shells supported by the Solaris Operating Environment.
   - Bourne Shell
   - C Shell
   - Korn Shell
4. Match the following terms with their descriptions:

b	Shell	a. Core of the Solaris Operating Environment
a	kernel	b. Command interpreter

Check Your Progress

Before continuing on to the next module, check that you are able to accomplish or answer the following:

- ❑ List the four main hardware components of a computer
- ❑ Describe the four main components of the Solaris Operating Environment
- ❑ State the three main components of the SunOS operating system
- ❑ Identify the shells available in the Solaris Operating Environment

Accessing the System 2

Objectives

Upon completion of this module, you should be able to:

- List the characteristics of an effective password
- Log in and log out of a system from the command line
- Log in and log out of a CDE session
- Execute basic commands
- Change your password
- Identify and describe the components of a command line
- Use control characters to erase a command line, stop the execution of a command, and stop and start screen output
- Display online manual pages
- Search the online manual pages by keyword
- Identify users logged on to the system using the commands `who`, `who am i`, and `id`
- Enter multiple commands on a single command line

Additional Resources

Additional resources – The following references provide additional details on the topics discussed in this module:

- *Solaris Common Desktop Environment: User's Guide,* "Starting a Desktop Session," Part Number 806-1360-10
- *System Administration Guide, Volume 1*, Part Number 805-7228-10

User Accounts

Every user must have a *user account* on the system to login. All user accounts are defined in the `/etc/passwd` file and contain the elements that identify each unique user to the system.

System administrators are responsible for creating and maintaining user accounts.

The `root` Account

The `root` account and password are set up during the Solaris Operating Environment installation process. This login account is used by the system administrator to perform specific administration tasks on the system.

The `/etc/passwd` Entry

Each user account entry in the `/etc/passwd` file contains seven fields, each separated by a colon.

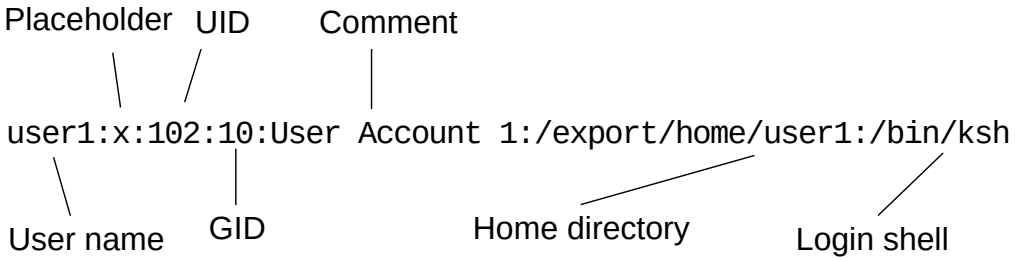


Figure 2-1 Example `/etc/passwd` Entry

- User name – Specifies the name used by the system to identify the user. Depending on the system administrator, user names are usually some combination of a user's first and last names. For example, a user named Bob Wood might be given the user name `bobw`, `bwood`, or `woodb`. The user name must be unique.
- Placeholder – Maintains the field for the password, which is kept in the `/etc/shadow` file. The `/etc/shadow` file contains encrypted passwords and password aging information (for

example, how long before a user must change a password and a date on which the account expires). This file can be read only by the system administrator.

- UID – Identifies the user's unique numerical ID or user identification (UID).
- GID – Identifies the user's unique numerical group ID (GID).
- Comment – Is traditionally the full name of the user.
- Home directory – Specifies the directory in which users create and store their personal files.
- Login shell – Defines the shell in which the user will be working after the user logs into the system.

Logging In

The login process identifies a user to the system.

The Login screen, displayed by the CDE Login Manager, is your entry to the desktop. It provides a place for you to enter your user name and password.

As an alternative to the CDE Login screen, you can log in from a command line by selecting that option from the Options menu.

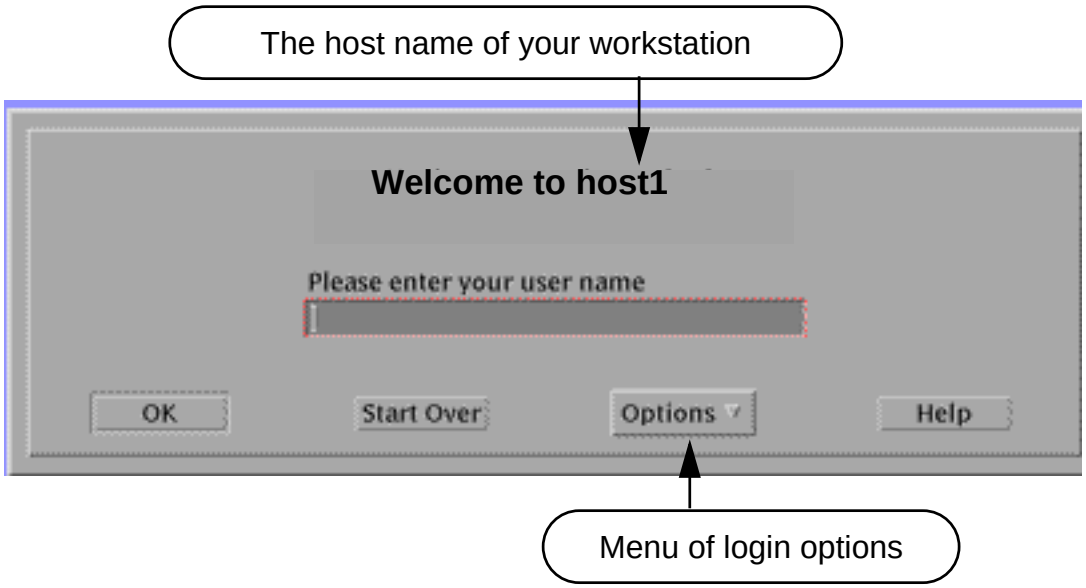


Figure 2-2 The Login Screen

The Options Button

When you select the Options button on the login screen, the Options menu lists your login choices.

- Options
  - Language
  - Session
    - Common Desktop Environment (CDE)
    - OpenWindows Desktop
    - User's Last Desktop
    - Failsafe Session
  - Remote Login
    - Enter Host Name
    - Choose Host From List
  - Command Line Login
  - Reset Login Screen

Language

The Options menu enables you to select a particular language for your session. The default language for your workstation is set by the system administrator.

Session

You can also select in which desktop environment to work (for example, CDE or OpenWindows).

The Failsafe session opens a single terminal window on the workstation screen, instead of starting a full desktop session. This is provided as an alternative method of logging in to fix problems with other sessions. To log out of a Failsafe session, execute the `exit` command.

Remote Login

The Remote Login option enables you to connect to a remote system to start a remote Desktop login. This option allows you to either enter the specific host name of a remote system or to select from a list of available remote systems.

Command-Line Login

The command-line login enables you to work in the more traditional non-GUI environment. This mode is not a desktop session. When the system is in command-line login, the desktop is suspended.

When you logout from a command-line prompt, the CDE Login screen is restarted.

Reset Login Screen

The Reset Login Screen option restarts the CDE Login Screen.

Logging In Using the Login Screen

To log in to a Desktop Session from the CDE Login screen:

1. Enter your user name in the text field, and then press the Return key or click the OK button.

2. Enter your password in the password text field, and then press the Return key or click the OK button.

If the log in attempt fails, a dialog box is shown with the error “Login incorrect; please try again.”

Logging In Using the Command Line

To log in to a command-line session:

1. Display the Options pull-down menu, and select Command Line Login.

   The Login screen disappears and is replaced by a console prompt.

2. Press the Return key to get a prompt for a user name entry.

Note – If you select the Command Line Login option, login within 30 seconds; otherwise, the CDE login screen automatically restarts.

---

3. Enter your user name (or login ID) at the prompt, and press the Return key.

4. Enter your password in the password text field, and press the Return key.

   The password does not appear on the screen when it is entered.

Note – By default, if a user does not have a password, then the user is automatically prompted to enter a new password during the initial login.

Password Requirements

Passwords protect user accounts from unauthorized access. In the Solaris Operating Environment, a user's password:

- Must be six to eight characters in length
- Should contain at least two alphabetic characters and must contain at least one numeric or special character, such as a semicolon (`;`), asterisk (`*`), or dollar sign (`$`)
- Must differ from the login ID name (user name entry)
- May contain spaces

Note – When changing your password, the new password must have at least three characters that differ from the current password.

These password requirements do not apply to the system administrator's `root` account password or to any user password that is assigned by the `root` user.

Changing Your Password

Frequently changing user passwords helps prevent unauthorized access to the system.

Changing Your Password in CDE

To change your password in a CDE session:

1. From the desktop, open a terminal window.
2. Execute the `passwd` command at the shell prompt, and press the Return key.
3. When the prompt `Enter login password:` appears, enter the current password, and press the Return key.
4. When the prompt `New password:` appears, enter the new password, and press the Return key.
5. When prompted, reenter the new password, and press the Return key.

   The system requires this to verify the new password.

Changing Your Password From the Command Line

To change your password from the command line:

1. Execute the `passwd` command at the prompt.
2. Enter the current password.
3. Enter the new password.
4. Reenter the new password for verification. The shell prompt is redisplayed; for example:

```
$ passwd
passwd: Changing password for user1
Enter login password:
New password:
Re-enter new passwd:
passwd (SYSTEM): passwd successfully changed for user1
$
```

Securing Your CDE Session

Securing your CDE session prevents unauthorized users from gaining access to the system. The two ways to secure the system are:

- Locking the screen
- Exiting the session

Locking the Screen

Locking the screen prevents unauthorized users from gaining access to your CDE session, while keeping your session intact.

The padlock icon on the Front Panel is used to secure the screen and apply password protection. To regain access to your CDE session, enter your password, and press the Return key.

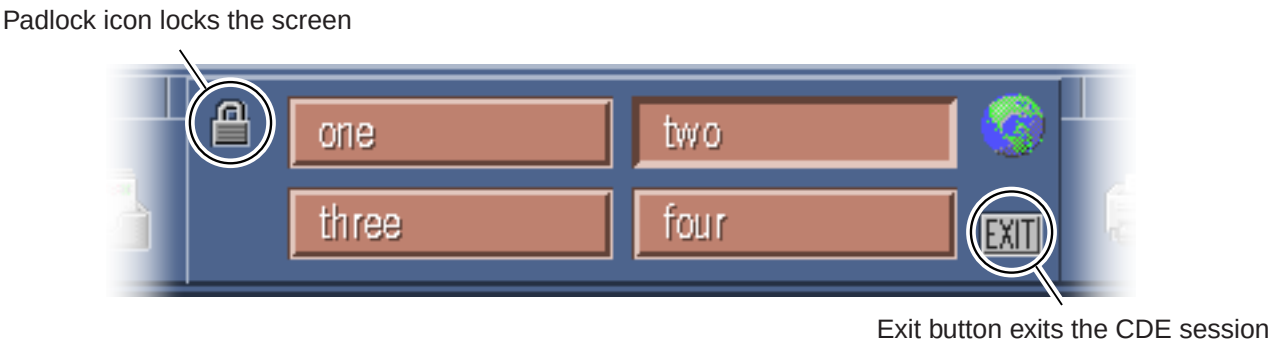


Figure 2-3 Exiting and Locking From the Toolbar

Exiting the Session

Two ways to exit the CDE session include:

- Using the EXIT button
- Using the Logout option from the Workspace Menu

Exiting Using the EXIT Button

Use the EXIT button, on the Front Panel, to log out from a CDE session.

By default, a Logout Confirmation window is displayed. To confirm the logout process, click the OK button or press the Return key when the OK button is highlighted.

The current CDE session is saved, by default, when you log out. The CDE session restores the session and windows when you next log in to a CDE session.

Caution – Any data contained in the current set of open applications is lost when logging out. Be sure to save all data before exiting from a CDE session.

Exiting Using the Workspace Menu

To log out using the Workspace Menu, right-click on the desktop area, and select the Logout option from the Workspace Menu.

By default, a Logout Confirmation window is displayed. To confirm the logout process, click the OK button or press the Return key when the OK button is highlighted.

The home CDE session is saved automatically so that you can return to the same workspace windows at your next CDE session.

Basic UNIX Commands

After you log in to the system, open a terminal window by right clicking on the desktop, selecting `tools` then `terminal` from the subsequent pop-up menus displayed. A Korn shell prompt appears at the start of the command line, indicating that the shell is ready to receive a command.

Note – Examples given in this course assume the shell being used is the Korn shell.

For both the Bourne and Korn shells, the default shell prompt for root is a pound character (#). For the C shell, the prompt for root is the host name and a pound character (`hostname#`).

Using the `uname` *Command*

The `uname` command lists information about the system. By default, entering this command displays the name of the current operating system.

Displaying the Operating System Name

To display the operating system information, execute the following:

```
$ uname
SunOS
$
```

Using the `date` *Command*

The `date` command displays the system's current date and time.

Displaying the Date and Time

To display the date and time, execute the following:

```
$ date
Fri Feb 25 12:55:29 MST 2000
$
```

Using the `cal` *Command*

The `cal` command displays a calendar for the current month and year.

Displaying the Calendar

To display the calendar, execute the following:

```
$ cal
     February 2000
 S  M Tu  W Th  F  S
      1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29

$
```

Command-Line Syntax

You can enhance the capability of commands by using options and arguments. The basic syntax of a UNIX command includes:

```
command option(s) argument(s)
```

command	Executable (specifies *what* the system is to do)
option	Modifies the command (this specifies *how* the command is run). Options start with a - (dash) character.
argument	A file name, directory name, or text

The following are samples of commands and commands using options and arguments.

$ **ls** (Command)

$ **ls -l** (Command and option)

$ **ls dir1** (Command and argument)

$ **ls -l dir2** (Command, option, and argument)

$ **cal 12 2000** (Command and two arguments)

$ **uname -rpns** (Command and multiple options)

$ **uname -r -p -n -s** (Command and multiple options)

Control Characters

Through the use of special control characters, you can stop and start screen output, erase an entire command line, or stop the execution of a command from the keyboard.

To enter a sequence of control characters, hold down the Control key and press the appropriate character on the keyboard for the desired action.

Table 2-1 lists the control characters you can use.

Table 2-1 Control Characters

Control Characters	Purpose
Control-C	Terminates the command currently running
Control-U	Erases all characters on the current command line
Control-S	Stops output to the screen
Control-Q	Restarts output to the screen after Control-S has been pressed
Control-D	Indicates end-of-file or exit
Control-W	Erases the last word on the command line

Viewing Online Documentation

The online UNIX Reference Manuals (also called *man pages*) provide detailed descriptions of commands and their usage. These online manual pages are included in the Solaris Operating Environment.

The man command is primarily used to display the online manual page for any given command.

Command Format

```
man [ -s section ] command_name ...
man -k keyword ...
```

Using the man Command Without Options

You can invoke the man command without options; for example:

```
$ man uname
Reformatting page.  Please Wait... done

User Commands                                              uname(1)

NAME
     uname - print name of current system

SYNOPSIS
     uname [ -aimnprsvX ]

     uname [ -S system_name ]

DESCRIPTION
     The uname utility prints information about the current  sys-
     tem on the standard output. When options are specified, sym-
     bols representing one or more system characteristics will be
     written to the standard output. If no options are specified,
     uname  prints  the  current  operating  system's  name.  The
     options  print  selected  information  returned by uname(2),
     sysinfo(2), or both.

<output omitted>
```

Scrolling in Man Pages

Table 2-2 shows the keys used to control the scrolling capabilities while using the `man` command.

Table 2-2 Keys to Control Scrolling in Man Pages

Key	Action
Spacebar	Displays the next screen of a man page
Return key	Scrolls through a man page one line at a time
`b`	Moves back one screen
`f`	Moves forward one screen
`q`	Quits the `man` command
`/`*`pattern`*	Searches forward for this *`pattern`*
`n`	Finds the next occurrence of *`pattern`*
`h`	Provides a description of all scrolling capabilities

Searching Man Pages by Section

There are many different components of a man page. The `SEE ALSO` part at the bottom of a man page lists alternative references that pertain to the topic addressed. When these references are followed by a number in parentheses, it indicates a section of the `man` pages that you can access using the `-s` option with the `man` command.

For example, executing the command **`man passwd`** displays information on the `passwd` command and provides instructions on how to change a password. The `SEE ALSO` section of this `man` page reads in part as follows:

```
SEE ALSO
     finger(1),  login(1),  nispasswd(1),  nistbladm(1),
     yppasswd(1),  domainname(1M),  eeprom(1M),  id(1M),
     passmgmt(1M),  pwconv(1M),  su(1M),  useradd(1M),  userdel(1M),
     usermod(1M),    crypt(3C),    getpwnam(3C),    getspnam(3C),
     getusershell(3C),  nis_local_directory(3N),  pam(3),  login-
     log(4),  nsswitch.conf(4),  pam.conf(  4),  passwd(4),  sha-
     dow(4), attributes(5), environ(5), pam_unix(5)
```

Executing **man -s4 passwd** displays information on the /etc/passwd file and **man -s3C crypt** displays information on the password encryption process. The -s4 and -s3C options tell the man command to refer to Sections 4 and 3C, respectively, of the online manual.

Using the man *Command With the* -k *Option*

You can invoke the man command with the -k option for a keyword lookup to display a list of the commands that might be relevant.

```
man -k keyword
```

By default, the keyword lookup for searching man pages is not enabled. To configure the system to enable this feature, the system administrator must run the following command:

```
# catman -w
```

Note – See catman(1M) for more details.

Searching Man Pages by Keyword

When you are not sure of the name for a command, you can use the -k option with the man command to specify a keyword as a subject.

```
$ man -k calendar
cal              cal (1)           - display a calendar
calendar         calendar (1)      - reminder service
difftime         difftime (3c)     - computes the difference between two
                                     calendar times
mktime           mktime (3c)       - converts a tm structure to a calendar
                                     time
$
```

Determining Current Users

The `who` command displays a list of users currently logged in to the local system, with their login name, login terminal identifier (TTY) port, login date and time, and the elapsed time since their last activity. When a user is logged in remotely, the remote system name displays for that user.

Command Format

```
who [ am i ]
```

Displaying Users on the System

To display the users on the system, execute:

```
$ who
user1     console      Feb 25 13:50    (:0)
user1     pts/4        Feb 25 14:37    (:0.0)
user1     pts/6        Feb 25 14:54    (:0.0)
user1     pts/7        Feb 25 15:24    (:0.0)
```

Identifying a User

Use the `who am i` command to identify the user name. This command is equivalent to typing the `who -m` command.

Command Format

```
who am i
```

Example

To display the user name, execute the following:

```
$ who am i
user1     pts/7           Feb 25 15:24    (:0.0)
$
```

Identifying User Group Details

Use the `id` command to identify the user ID, user name, group ID, and group name of a system user.

Command Format

```
id [ username ]
```

Identifying a User

To identify your user account information, execute the following:

```
$ id
uid=11001(user1) gid=10(staff)
$
```

To identify a specific user, execute the following:

```
$ id root
uid=0(root) gid=1(other)
$
```

Entering Multiple Commands From a Single Command Line

The semicolon (;) is a special character to the shell and is used as a *command separator.*

The semicolon enables you to enter multiple commands on a single command line. The shell executes each command from left to right when the Return key is pressed.

The following examples demonstrate the use of the semicolon.

```
$ cd;ls
dante     dir2      file.1    file1     file4     practice
dante_1   dir3      file.2    file2     fruit     tutor.vi
dir1      dir4      file.3    file3     fruit2
$


$ date;cal;pwd
Thu Feb 17 16:49:34 MST 2000
   February 2000
 S  M Tu  W Th  F  S
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29

/export/home/user1
$
```

Exercise: Accessing the System

Exercise objective – In this exercise, you practice logging in using the command line, changing your password, executing commands, and logging out.

Tasks

Complete the following steps:

1. Obtain a user name and password from your instructor.
2. Log in to the system using the CDE Login Manager login screen.
   a. Enter your user name, and press the Return key.
   b. Enter your password, and press the Return key.
   c. Select CDE by clicking OK or pressing the Return key.
3. On the CDE Desktop, right-click on the background screen.

   The Workspace menu is displayed.
4. Select Tools from this menu.

   The Tools menu is displayed.
5. Select Terminal from this menu.

   A terminal window is displayed.
6. Use the mouse to move the cursor into the terminal window.
7. Using the `passwd` command, change your password to `mypass1`.

```
$ passwd
Enter login password: (This is your original password)
New password:
Re-enter new password:
passwd (SYSTEM) passwd successfully changed for username
```

8. Exit the CDE desktop by clicking on the EXIT button, located on the Front Panel.

   A Logout Confirmation window is displayed.

9. Click OK or press the Return key to continue to logout.

10. At the CDE Login Manager screen, enter the following incorrect login name and password:

    - Login name: **`nosuchuser`**
    - Password: **`wrong`**

    The message `Login incorrect; please try again` is displayed.

11. Click OK.

12. When the CDE Login Manager screen is displayed, click Options and select the Reset Login screen.

Logging in Using the Command-Line Login Option

1. When the CDE Login Manager screen displays, click Options and select Command Line Login.

2. When the following message appears, press the Return key to display the console login prompt.

```
************************************************
*Suspending Desktop Login ...
*
*If currently logged out, press [Enter] for a console *login prompt.
*
*Desktop Login will resume shortly after you exit console *session.
************************************************
<hostname> console login:
```

3. Enter your user name, and press the Return key.

4. Enter your new password, and press the Return key.

   If you see the message:
   `Starting OpenWindows in 5 seconds (type Control-C to interrupt)`

Hold down the Control key and press the C key. A `^C$` is displayed on the screen. The `^C` is the Control-C key sequence echoed to the screen. The dollar sign (`$`) is your shell prompt.

```
Last login <date> <time> on console
Sun Microsystems Inc. SunOS 5.8 Generic February 2000
```

5. At the shell prompt, type the `exit` command.

   The `<hostname> console login:` prompt reappears. After a short time, the CDE Login Manager screen is redisplayed.

Invoking a Failsafe Session

1. When the CDE Login Manager screen is displayed, click Options, select Session, and select Failsafe Session.
2. Enter your user name, and press the Return key.
3. Enter your password, and press the Return key.

   A single terminal window is displayed.

4. Using the mouse, move the cursor into the terminal window.
5. Execute the `exit` command.

Invoking the User's Last Desktop Session

1. At the CDE login screen, enter your user name and password and start a CDE Desktop session.
2. On the CDE Desktop, using the right mouse button, click the background screen.
3. Open a terminal window.
4. Display the date and time using the `date` command. Display the date and time using the `date -u` command. What is the difference in the information displayed on the screen?

   _____

   _____

5. Display this month's calendar. How was this command entered on the command line?

   _____

6. Display what user name you entered to log in to the system? What command did you use to display this information?

   _____

7. What command displays information about other users currently logged onto your workstation?

   _____

8. What command can be used to discover the user name, UID, group name, and GID of any user on the workstation?

   _____

   a. Enter the `id` command.
      What information displays?
      _____

   b. Enter the `id -a root` command.
      What information displays?
      _____

9. What command can be used to display information about the operating system and the workstation name?

   _____

   a. Execute the `man uname` command and determine what the options `-s`, `-r`, and `-n` do.

   b. Enter the `uname -s` command.
      What information is displayed? _____

   c. Enter the `uname -r` command.
      What information is displayed? _____

   d. Enter the `uname -n` command.
      What information is displayed? _____

   e. Enter the `uname -srn` command.
      What information is displayed? _____

10. Display the online manual pages for the `passwd` command first, and then the `passwd` file.

    a. Execute the `man passwd` command.
    The `man` command moves to Section 1 of the online manual pages to locate information on the `passwd(1)` command.

    b. Move to the end of this online manual page by typing **`/SEE`** (the forward slash [/] character and the word `SEE` are a search string).

    Under the `SEE ALSO` section, locate the `passwd(4)` entry. This indicates information on the `passwd` file is located in Section 4 of the online manual pages.

    c. Execute the `man -s4 passwd` command.
    The `man` command moves to Section 4 of the online manual pages to locate information on the `passwd` file.

11. Using the `-k` option with `man` command, find the online manual page that describes how to clear a terminal window (use the keyword "clear"). How would this command be entered on the command line?

    _____

12. Clear the terminal window.

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercises.

- Experiences
- Interpretations
- Conclusions
- Applications

Task Solutions

Invoking the User's Last Desktop Session

4. Display the date and time using the `date` command. Display the date and time using the `date -u` command. What is the difference in the information displayed on the screen?

   The `date` *command displays the current date and time to the terminal window.*

   The `date -u` *command displays the date and time in Greenwich Mean Time (GMT-universal time) to the terminal window.*

5. Display this month's calendar. How was this command entered on the command line?

   `cal`

6. Display the user name you entered to log in to the system? What command did you use to display this information?

   `who am i`

7. What command displays information about other users currently logged onto your workstation?

   `who`

8. What command can be used to discover the user name, UID, group name, and GID of any user on the workstation?

   `id` or **`id`** ***`username`***

   a. Enter the `id` command.
   What information is displayed?

   Your UID number, current user name, GID number, and group name.

   b. Enter the `id -a root` command.
   What information is displayed?

   The user's UID number, user name, GID numbers, and group names for all groups to which this user belongs.

9. What command can be used to display information about the operating system and the workstation name?

   `uname -a`

   or

   `uname -ns`

   a. Enter the `uname -s` command.
   What information is displayed?

      The operating system name; for example, SunOS.

   b. Enter the `uname -r` command.
   What information is displayed?

      The operating system version; for example, 5.8.

   c. Enter the `uname -n` command.
   What information is displayed?

      The host name of the workstation.

   d. Enter the `uname -srn` command.
   What information is displayed?

      The operating system name, host name, and version number.

11. Using the `-k` option with `man` command, find the online manual page that describes how to clear a terminal window (use the keyword "clear"). How would this command be entered on the command line?

    `man -k clear`

12. Clear the terminal window.

    `clear`

Check Your Progress

Before continuing on to the next module, check that you are able to accomplish or answer the following:

- ❑ List the characteristics of an effective password
- ❑ Log in and log out of a system from the command line
- ❑ Log in and log out of a CDE session
- ❑ Execute basic commands
- ❑ Change your password
- ❑ Identify and describe the components of a command line
- ❑ Use control characters to erase a command line, stop the execution of a command, and stop and start screen output
- ❑ Display online manual pages
- ❑ Search the online manual pages by keyword
- ❑ Identify users logged on to the system using the commands `who`, `who am i`, and `id`
- ❑ Enter multiple commands on a single command line

Accessing Files and Directories 3

Objectives

Upon completion of this module, you should be able to:

- Demonstrate the difference between absolute and relative path names
- Access files and directories within the directory tree using absolute and relative path names
- Use path name abbreviations to access files and directories within the directory tree
- List the contents of directories and determine file types
- Identify various shell metacharacters to abbreviate file names and path names

Additional Resources

Additional resources – The following references provide additional details on the topics discussed in this module:

- *Solaris Common Desktop Environment: User's Guide,* "Starting a Desktop Session," Part Number 806-1360-10
- *System Administration Guide, Volume 1*, Part Number 805-7228-10

The Directory Tree

Figure 3-1 illustrates a portion of a sample Solaris directory tree, showing the placement of a particular user's files and directories.

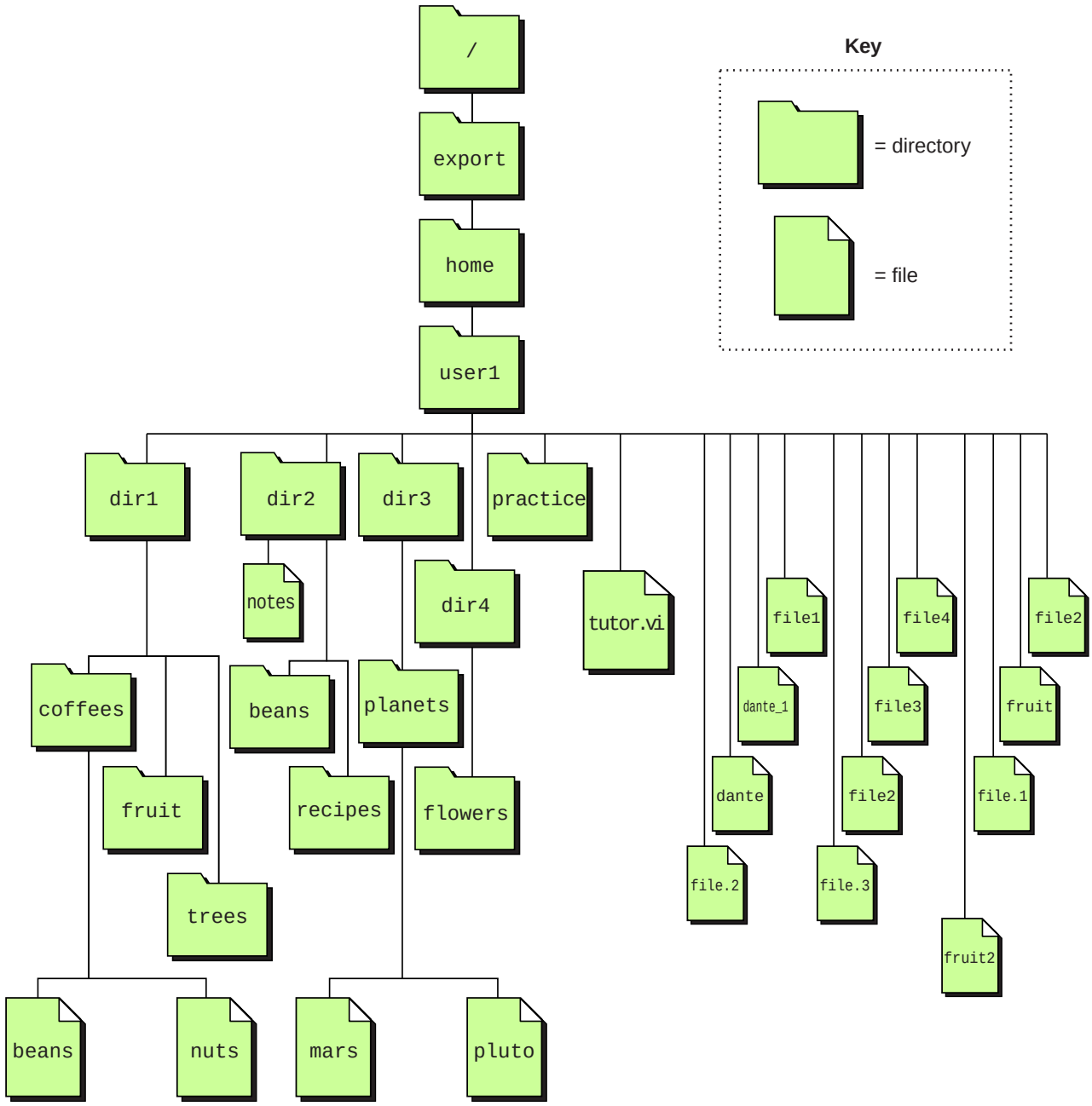


Figure 3-1 Directory Tree

Path Names

A *path name* uniquely identifies a particular file or directory by specifying its location in the directory tree. Path names are similar to road maps, which show how to get from one place in the directory tree to another.

The slashes (/) within a path name are delimiters between object names. An object name can be a directory name or a file name. The slash at the start of a path name always represents the `root` (/) directory; for example:

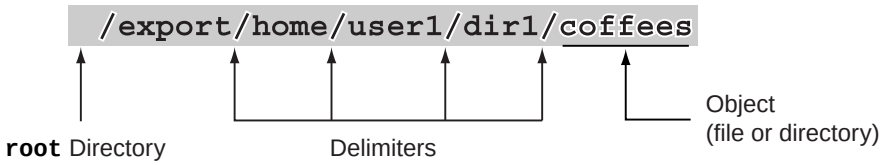


Figure 3-2 Path Name With Delimiters

Note – Depending on the system setup, your home directory can be located in either the `/home` directory or the `/export/home` directory.

Path Name Types

There are two types of path names: absolute and relative.

Absolute Path Name

An *absolute path name* specifies a file or directory in relation to the entire Solaris Operating Environment directory tree. Absolute path names always:

- Start with a slash (/) representing the root directory and then list each directory along the path to the final destination (which may be a file name or another directory). A slash (/) separates multiple directory or file names.

Note – An absolute path name can also be referred to as a *full path name*.

Refer to Figure 3-1 on page 3-3 for a visual representation of the following path names:

- The absolute path name to the `user1` directory is:

  `/export/home/user1`

- The absolute path name to the `dir1` directory is:

  `/export/home/user1/dir1`

- The absolute path name to the `coffees` directory is:

  `/export/home/user1/dir1/coffees`

Relative Path Name

A *relative path name* describes the location of a directory or a file as it relates to the current directory.

A relative path name never begins with a slash (/) character. However, it does use slashes (/) within the path name as delimiters between object names (for example, directory name or file name).

If you are in a directory and you want to move down to access another directory in the hierarchy, you do not have to enter an absolute path name. Simply enter the path starting with the name of the next directory down in the tree structure.

Refer to Figure 3-1 on page 3-3 for a visual representation of the following examples:

- The current directory is:

  `/export/home`

- From `/export/home`, the relative path name to access the `user1` directory is:

  `user1`

- From `/export/home`, the relative path name to access `dir1` is:

  `user1/dir1`

- From `/export/home`, the relative path name to access the `coffees` directory is:

  `user1/dir1/coffees`

File and Directory Naming Conventions

When creating files and directories, you must use the following conventions:

- Directory and file names can be up to 255 characters in length. The characters can be alphanumeric, and non-alphanumeric, such as underscores (_), periods (.), and hyphens (-).
- Special characters, such as asterisks (*), ampersands (&), pipes (|), quotes (""), and dollar signs ($) should not be used. These particular characters hold special meaning to the shell.
- Spaces should not be used in directory or file names.
- File and directory names, as a rule, do not contain extensions. However, if desired, extensions can be used.

Changing Directories

At any given time, you are located in a current working directory within the directory tree. When you initially log in to the system, the current directory is set to your home directory.

You can change your current working directory at any time by using the `cd` command.

Command Format

```
cd directory_name
```

Moving Around the Directory Tree

The following examples show how to change directories within the directory tree:

- Using an absolute path name:

  ```
  $ cd /export/home/user1/dir1/coffees
  ```

- Using a relative path name:

  ```
  $ cd user1/dir1/coffees
  ```

You can always return to your own home directory by typing the `cd` command without an argument; for example:

```
$ cd
```

Displaying the Current Directory

The pwd command, or print working directory command, identifies the directory in which you are currently working.

The pwd command displays the absolute path name of the current working directory.

Command Format

```
pwd
```

Determining the Current Working Directory

Examples of using the pwd command are shown as follows:

```
$ pwd
/export/home/user1
$ cd /export/home
$ pwd
/export/home
$ cd
$ cd practice
$ pwd
/export/home/user1/practice
$
```

Changing Directories With Path Name Abbreviations

Path name abbreviations are used as a quick method for moving to or referring to directories on the command line.

Table 3-1 Path Name Abbreviations

Symbol	Meaning
.	Current or working directory
..	Parent directory; the directory directly above the current working directory

The following examples show how to use the `cd` command to give path name abbreviations to move around the Solaris directory tree.

```
$ pwd
/export/home/user1/dir1
$ cd ..
$ pwd
/export/home/user1
$ cd ../../..
$ pwd
/
$
```

Displaying the Contents of a Directory

To display the contents of a directory, use the `ls` command. This command lists the files and directories within the specified directory.

Using the `ls` command with no argument simply displays the contents of the current directory.

Command Format

`ls [ -options ] pathname ...`

Listing the Contents of a Directory

To list the contents of a directory, execute the following:

```
$ cd
$ pwd
/export/home/user1
$ ls
dante     dir2      file.1    file1     file4     practice
dante_1   dir3      file.2    file2     fruit     tutor.vi
dir1      dir4      file.3    file3     fruit2
$ ls dir1
coffees   fruit   trees
$ ls /var/mail
:saved user1
$
```

Displaying Hidden Files

File names that begin with a period (`.`) are called *hidden files*. Hidden files are frequently used to customize your work environment.

Use `ls -a` to list all files in a directory, including any hidden (`.`) files.

To display hidden files, execute the following:

```
$ pwd
/export/home/user1
$ ls -a
.              .dtprofile    dir1          file.2        file4
..             .sh_history   dir2          file.3        fruit
.TTauthority   .solregis     dir3          file1         fruit2
.Xauthority    dante         dir4          file2         practice
.dt            dante_1       file.1        file3         tutor.vi
$
```

Displaying File Types

Use `ls -F` to display file types. Use the symbols in Table 3-2 to interpret the `ls -F` output:.

Table 3-2 File Type Symbols

File Type	Symbol
Directory	/
Executable	*
Plain text file/ASCII	(none)
Symbolic link	@

Execute the following to show a file type:

```
$ pwd
/export/home/user1

$ ls -F
dante      dir2/      file.1     file1      file4      practice/
dante_1    dir3/      file.2     file2      fruit      tutor.vi
dir1/      dir4/      file.3     file3      fruit2
$
```

Note – A *symbolic link* is a special type of file that points to another file or directory. A symbolic link file contains the path name of the file or directory to which it points.

Displaying a Long Listing

To see detailed information about the contents of a directory, use the `ls -l` command.

To get detailed information using the `ls -l` command, execute the following:

```
$ ls -l
total 88
-rw-r--r--   1 user1    staff       1320 Feb 22 14:51 dante
-rw-r--r--   1 user1    staff        368 Feb 22 14:51 dante_1
drwxr-xr-x   5 user1    staff        512 Feb 22 14:51 dir1
drwxr-xr-x   4 user1    staff        512 Feb 22 14:51 dir2
drwxr-xr-x   3 user1    staff        512 Feb 22 14:51 dir3
drwxr-xr-x   3 user1    staff        512 Feb 22 14:51 dir4
-rw-r--r--   1 user1    staff          0 Feb 25 12:54 file.1
-rw-r--r--   1 user1    staff          0 Feb 25 12:54 file.2
-rw-r--r--   1 user1    staff          0 Feb 25 12:54 file.3
-rw-r--r--   1 user1    staff       1696 Feb 22 14:51 file1
-rw-r--r--   1 user1    staff        105 Feb 22 14:51 file2
-rw-r--r--   1 user1    staff        218 Feb 22 14:51 file3
-rw-r--r--   1 user1    staff        137 Feb 22 14:51 file4
-rw-r--r--   1 user1    staff         56 Feb 22 14:51 fruit
-rw-r--r--   1 user1    staff         57 Feb 22 14:51 fruit2
drwxr-xr-x   2 user1    staff        512 Feb 22 14:51 practice
-rw-r--r--   1 user1    staff      28738 Feb 22 14:51 tutor.vi
$
```

The `ls -l` command provides the file information as shown in Figure 3-3.

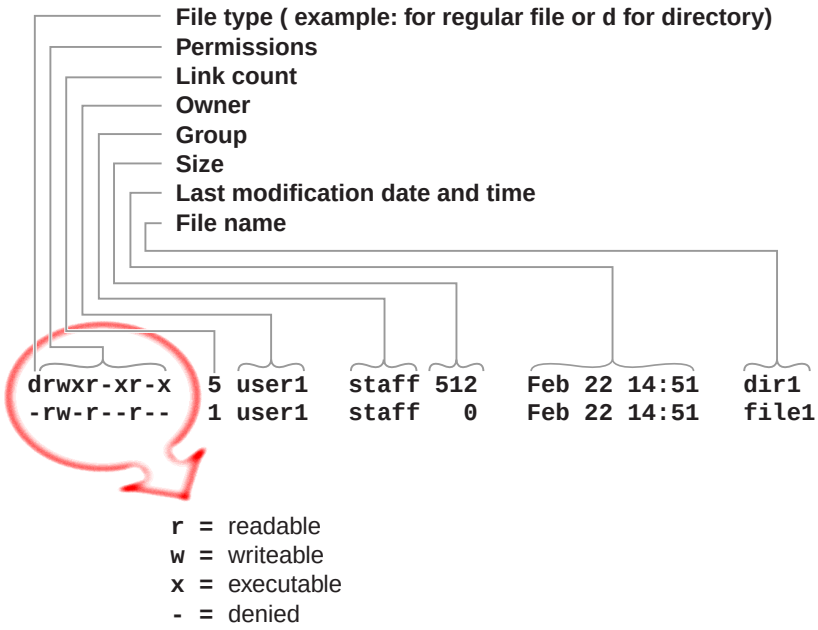


Figure 3-3 Long Listing File Information

Listing Individual Directories

Use `ls -ld` to display detailed information for the directory only, not its contents.

To obtain detailed directory information, execute the following:

```
$ cd
$ ls -l dir1
total 6
drwxr-xr-x 2 user1 staff 512 Feb 22 14:51  coffees
drwxr-xr-x 2 user1 staff 512 Feb 22 14:51  fruit
drwxr-xr-x 2 user1 staff 512 Feb 22 14:51  trees
$ ls -ld dir1
drwxr-xr-x 5 user1 staff 512 Feb 22 14:51  dir1
$
```

Use `ls -R` to display the contents of a directory and all of its subdirectories.

This is also known as a *recursive* listing.

To display a recursive listing, execute the following:

```
$ pwd
/export/home/user1
$ ls -R dir1
dir1:
coffees  fruit    trees

dir1/coffees:
beans    nuts

dir1/fruit:

dir1/trees:
$
```

To display a listing sorted by the file's last modification time, with the latest modified file appearing first in the list, execute the following:

```
$ ls -lt
drwx--x--x  3 user1 staff      96 Aug 14 16:17 dir3
drwx--x--x  3 user1 staff      96 Aug 14 16:17 dir4
drwx--x--x  2 user1 staff      96 Aug 14 16:17 practice
drwx--x--x  5 user1 staff      96 Aug 14 16:17 dir1
drwx--x--x  4 user1 staff      96 Aug 14 16:17 dir2
-rwx--x--x  1 user1 staff    1610 Jul 25 14:55 file1
-rwx--x--x  1 user1 staff   28738 May 31 16:45 tutor.vi
-rwx--x--x  1 user1 staff     218 May 31 16:45 file3
-rwx--x--x  1 user1 staff     137 May 31 16:45 file4
-rwx--x--x  1 user1 staff      56 May 31 16:45 fruit
-rwx--x--x  1 user1 staff      57 May 31 16:45 fruit2
-rwx--x--x  1 user1 staff       0 May 31 16:45 file.1
-rwx--x--x  1 user1 staff       0 May 31 16:45 file.2
-rwx--x--x  1 user1 staff       0 May 31 16:45 file.3
-rwx--x--x  1 user1 staff     105 May 31 16:45 file2
-rwx--x--x  1 user1 staff     368 May 31 16:45 dante_1
-rwx--x--x  1 user1 staff    1320 May 31 16:44 dante
$
```

To display a listing showing the latest file modification time in reverse order, execute the following:

```
$ ls -ltr
-rwx--x--x  1 user1 staff      1320 May 31 16:44 dante
-rwx--x--x  1 user1 staff       368 May 31 16:45 dante_1
-rwx--x--x  1 user1 staff       105 May 31 16:45 file2
-rwx--x--x  1 user1 staff         0 May 31 16:45 file.3
-rwx--x--x  1 user1 staff         0 May 31 16:45 file.2
-rwx--x--x  1 user1 staff         0 May 31 16:45 file.1
-rwx--x--x  1 user1 staff        57 May 31 16:45 fruit2
-rwx--x--x  1 user1 staff        56 May 31 16:45 fruit
-rwx--x--x  1 user1 staff       137 May 31 16:45 file4
-rwx--x--x  1 user1 staff       218 May 31 16:45 file3
-rwx--x--x  1 user1 staff     28738 May 31 16:45 tutor.vi
-rwx--x--x  1 user1 staff      1610 Jul 25 14:55 file1
drwx--x--x  4 user1 staff        96 Aug 14 16:17 dir2
drwx--x--x  5 user1 staff        96 Aug 14 16:17 dir1
drwx--x--x  2 user1 staff        96 Aug 14 16:17 practice
drwx--x--x  3 user1 staff        96 Aug 14 16:17 dir4
drwx--x--x  3 user1 staff        96 Aug 14 16:17 dir3
$
```

Shell Metacharacters

Shell metacharacters are specific characters, generally symbols, used to represent a special meaning to the shell. Some examples of shell metacharacters include:

```
~ - + * ? [ ]
```

Using the Tilde (~) Character

The shell substitutes the tilde (~) character with the home directory of the current user. It is an abbreviation of the absolute path name; for example:

```
$ cd /etc
$ pwd
/etc
$ cd ~/dir1
$ pwd
/export/home/user1/dir1/
$
```

Note – The tilde (~) character is available in all shells except the Bourne shell.

Using ~username

Attaching a user name to the tilde (~) character refers to the home directory of the specified user; for example:

```
$ cd ~user2
$ pwd
/export/home/user2
$
```

Using ~+ and ~-

The tilde and plus string (~+) refers to the current working directory. The tilde and dash string (~-) refers to the previous working directory.

For example:

```
$ pwd
/export/home/user1
$ cd dir2
$ ls ~+
beans notes recipes
$ cd ~-
$ pwd
/export/home/user1
$ cd ~-
$ pwd
/export/home/user1/dir2
$
```

Using the Dash

To switch quickly between two specific directories, use the Korn shell dash (-) symbol. The Korn shell automatically displays the current directory path with this particular option.

```
$ pwd
/export/home/user1
$ cd /tmp
$ pwd
/tmp
$ cd -
/export/home/user1
$ cd -
/tmp
$
```

Using the Asterisk

The asterisk (*) represents zero or more characters, excluding the leading period (.) on a hidden file. The asterisk is often referred to as a *wildcard* character.

For example:

```
$ ls f*
file.1  file.3  file2  file4  fruit2
file.2  file1   file3  fruit

$

$ ls d*
dante    dante_1

dir1:
coffees  fruit    trees

dir2:
beans    notes    recipes

dir3:
planets

dir4:
flowers
$

$ ls *3
file.3  file3

dir3:
planets
$
```

Using the Question Mark

The question mark (?) matches any single character, excluding the leading period (.) on a hidden file.

For example:

```
$ ls dir?
dir1:
coffees  fruit    trees

dir2:
beans    notes    recipes

dir3:
planets

dir4:
flowers
$
```

The following example shows the error message that is displayed if there are no file names matching the wildcard character.

```
$ ls z?
z?: No such file or directory
$
```

Using the Square Brackets

Use square brackets ([ ]) to match a set or range of characters for a single character position.

When looking for a set of characters, the characters inside the brackets do not generally need to be in any order; for example, `[abc]` is the same as `[cab]`.

However, when looking for a range of characters, they must be in proper order; for example, `[a–z]` or `[0–9]`.

To search for all alphabetic characters, whether lowercase or uppercase, use `[A–Z]` or `[a-z]` as the pattern to match.

For example:

```
$ ls [a-f]*
dante    file.1   file.3   file2    file4    fruit2
dante_1  file.2   file1    file3    fruit

dir1:
coffees  fruit    trees

dir2:
beans    notes    recipes

dir3:
planets

dir4:
flowers
$

$ ls [af]*
file.1  file.2  file.3  file1   file2   file3   file4   fruit   fruit2
$
```

Note – You should not use these metacharacters when creating file and directory names. These particular characters hold special meaning to the shell.

Exercise: Accessing Files and Directories

Exercise objective – In this exercise, you use the commands described in this module to list and change directories.

Tasks

Referring to Figure 3-1 on page 3-3 identify the path names for the following objects:

1. Specify the absolute path names for:
   - `user1`_____
   - `coffees`_____
   - `dir4`_____

Assume `/export/home/user1` is the current directory for the next two questions.

2. Specify the relative path names for:
   - `dir3`_____
   - `flowers`_____
   - `recipes`_____
3. Specify the relative path names for the `dir1` subdirectories and files.

_____

_____

_____

_____

_____

Using Figure 3-1 as a reference, perform each of the following tasks on your system. Use path name abbreviations whenever possible.

4. Change to your home directory.

   _____

5. Change to the `dir1` directory.

   _____

6. Change to the `fruit` directory.

   _____

7. Change to the `planets` directory.

   _____

8. Change to your home directory.

   _____

9. Change to the `/etc` directory.

   _____

10. Change to the `recipes` directory.

    _____

11. Change to the `flowers` directory.

    _____

Use the `ls` and `cd` commands to complete the following steps. Refer to Figure 3-1 on page 3-3 for Steps 12 through 17, if needed.

12. Return to your home directory.

    _____

13. Change to the `dir1` directory.

    _____

14. List the contents of the `dir1` directory.

_____

15. Display a recursive listing of the contents of the `dir2` directory.

_____

16. Use the `ls` command to display a detailed listing of your home directory, including hidden files.

_____

17. Use the `ls` option that recursively displays all contents in your home directory.

_____

Is there a directory in the `root` directory (`/`) called `kernel`?

_____

Is there a directory in `/var/spool` called `cron`?

_____

18. Without changing directories, execute the `ls` command that displays all the file names that end with the number 1 in your home directory.

_____

19. On one command line, change to your home directory, and list the contents of the directory.

_____

20. Issue the `ls` command that displays the file and directory names of any length beginning with the letters `d` or `f`.

_____

21. Issue the `ls` command that displays all files starting with `file` followed by any one character.

_____

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercises.

- Experiences
- Interpretations
- Conclusions
- Applications

Task Solutions

1. Specify the absolute path names for:

   - `user1`

     `/export/home/user1`

   - `coffees`

     `/export/home/user1/dir1/coffees`

   - `dir4`

     `/export/home/user1/dir4`

2. Specify the relative path names for:

   - `dir3`

     `dir3`

   - `flowers`

     `dir4/flowers`

   - `recipes`

     `dir2/recipes`

3. Specify the relative path names for the `dir1` subdirectories and files.

   `dir1/coffees`; `dir1/coffees/nuts`; `dir1/coffees/beans`
   `dir1/fruit`; `dir1/trees`

4. Change to your home directory.

   `cd`
   or
   `cd ~`

5. Change to the `dir1` directory.

   `cd dir1`

6. Change to the `fruit` directory.

   `cd fruit`
   or
   `cd ~/dir1/fruit`

7. Change to the `planets` directory.

   `cd ~/dir3/planets`
   or
   `cd ../../dir3/planets`

8. Go back to your home directory.

   `cd`
   or
   `cd ~`

9. Change to the `/etc` directory.

   `cd /etc`

10. Change to the `recipes` directory.

    `cd ~/dir2/recipes`
    or
    `cd /export/home/user1/dir2/recipes`

11. Change to the `flowers` directory.

    `cd ../../dir4/flowers`
    or
    `cd ~/dir4/flowers`

12. Return to your home directory.

    `cd`
    or
    `cd ~`

13. Change to the `dir1` directory.

    `cd dir1`

14. List the contents of the `dir1` directory.

    `ls`

15. Display a recursive listing of the contents of the `dir2` directory.

    `ls -R ../dir2`
    or
    `ls -R ~/dir2`

16. Use the `ls` command to display a detailed listing of your home directory, including hidden files.

    `ls -la ~`

17. Use the `ls` option that recursively displays all contents in your home directory.

    `ls -R ~`

    Is there a directory in the `root` directory (`/`) called `kernel`?

    Yes;
    `ls -F /`
    or
    `ls -ld /kernel`

    Is there a directory in `/var/spool` called `cron`?

    Yes; **`ls -F /var/spool`**

18. Without changing directories, execute the `ls` command that displays all the file names that end with the number 1 in your home directory.

    `ls ~/*1`

19. On one command line, change to your home directory, and list the contents of the directory.

    `cd;ls`

20. Issue the `ls` command that displays the file and directory names of any length beginning with the letters `d` or `f`.

    `ls [df]*`

21. Issue the `ls` command that displays all files starting with `file` followed by any one character.

    `ls file?`

Check Your Progress

Before continuing on to the next module, check that you are able to accomplish or answer the following:

- ❑ Demonstrate the difference between absolute and relative path names
- ❑ Access files and directories within the directory tree using absolute and relative path names
- ❑ Use path name abbreviations to access files and directories within the directory tree
- ❑ List the contents of directories and determine file types
- ❑ Identify various shell metacharacters to abbreviate file names and path names

Directory and File Commands 4

Objectives

Upon completion of this module, you should be able to:

- Determine file types with the `file` command
- Display the contents of text files using the `cat`, `more`, `pg`, `head`, and `tail` commands
- Determine character, word, and line counts using the `wc` command
- Create empty files or update modification times on existing files using the `touch` command
- Use the `tee` command to create text within a file
- Create and remove directories using the `mkdir` and `rmdir` commands
- Manage files and directories using the `mv`, `cp`, and `rm` commands
- Use commands to print a file, check print queue status, and cancel a print request
- Format and print the contents of files using the `pr` command

Additional Resources

Additional resources – The following reference provides additional details on the topics discussed in this module:

- *System Administration Guide, Volume 1*, Part Number 805-7228-10

Determining File Type

There are many different file types found in the Solaris Operating Environment. Using the `file` command allows you to determine some file types easily.

This information is important when you want to open or read a file. Knowing the file type helps you decide which command or program you need to use.

Command Format

`file` *`filename(s)`*

The output from the `file` command is most often one of the following:

- Text – Examples include American Standard Code for Information Interchange (ASCII) text, English text, commands text, and executable shell scripts.
- Data – Data files are those that are created by an application. In some cases, the type of data file is indicated; for example, a FrameMaker document. When the `file` command cannot determine the file type, it simply indicates that the file is a data file.
- Executable or binary – Examples include 32-bit executable and extensible linking format (ELF) code files and other dynamically linked executables. This file type indicates that the file is a command or program.

Example Text File

The following is an example of a text file:

```
$ file dante
   dante:                commands text
```

Example Data File

The following is an example of a data file:

```
$ cd /export/home/user1/dir1/coffees
$ file beans
  beans:                Frame Maker Document
```

Example Executable File

The following is an example of an executable file:

```
$ file /usr/bin/cat
/usr/bin/cat:  ELF 32-bit MSB executable SPARC
Version 1, dynamically linked, stripped
```

Displaying the Contents of a Text File

The `cat` (catenate) command displays the contents of one or more text files on the screen. It is often used to display short text files because `cat` displays the entire contents of the file to the screen without pausing.

You can also use the `cat` command to create short text files without having to use an editor.

Command Format

```
cat [ filename ... ]

cat > filename
```

Using the `cat` Command to Display a Short Text File

To display a short text file, execute the following:

```
$ cat dante
The Life and Times of Dante
by Dante Pocai

Mention “Alighieri” and few may know about whom you
are talking. Say “Dante,” instead, and the whole
world
knows whom you mean. For Dante Alighieri, like
Raphael, Michelangelo, Galileo, etc. is usually
referred to by his first name. ...
$
```

If the contents of the file fills more than one screen, the top of the file scrolls off the screen. If using a scrolling window, such as a terminal window within CDE, scroll the contents back using the scroll bars to view the entire file.

Using the `cat` *Command to Create a Short Text File*

To create a short text file, execute the following:

```
$ cat > newfile
```

You can then begin typing text in this new file. To save the contents of the file, press Control-D on an empty line.

Caution – If the file name already exists, and the `noclobber` option has not been set, the new file overwrites the existing file.

Joining Multiple Files

Use the `cat` command to join the contents of two files into a new file; for example:

```
$ cat filename1 filename2 > newfile1
```

Note – The file name after the “>” character should not be the same as any of the file names before the “>”; otherwise, the `cat` command displays the following error message: `cat: input/output files 'filename' identical`.

Extracting Printable Strings

The `strings` command finds printable strings in a binary data file. This enables you to read text strings imbedded within a binary file that can be useful for programming.

For example:

```
$ strings /usr/bin/cat
SUNW_OST_OSCMD
usvtebn
usage:  cat [ -usvtebn ] [-|file] ...
cat:  Cannot stat stdout
cat:  cannot open %s
cat:  cannot stat %s
cat:  input/output files '%s' identical
cat:  close error
cat:  close error
cat:  close error
cat:  cannot read %s:
cat:  write error:
cat:  mmap error
cat:  no memory
cat:  output error (%d/%d characters written)
cat: input error on %s:
standard input
$
```

Note – Someone with a programming background might be able to interpret the output produced by the `strings` command. The command is introduced here solely as a method for demonstrating the printable characters of an executable file.

More Metacharacters

Two useful metacharacters are the redirect (>) character and the pipe (|) character.

The redirect (>) character takes a command output and directs it to a specified file.

The pipe (|) character is used on the command line to take the output of one command and pass it as input to another command.

Browsing the Contents of a File

To browse, or page, through the contents of a long text file, use the `more` command. With this command, the contents of a text file display one screen at a time, and the following message appears at the bottom of the screen:

```
--More--(n%)
```

The `n%` is the percentage of the file already displayed. When the entire file content has been displayed, the shell prompt appears.

The online man pages use the `more` utility for display purposes, so the scrolling keys in Table 4-1 on page 4-10 are the same as those used to control the display of man pages.

Note – Using `cat` or `more` to read binary files can cause a terminal or window to freeze. If this occurs, close the window and open a new window, or select Reset -> Soft Reset from the CDE Options menu.

Command Format

```
more [ filename ... ]
```

Use the `more` command to read text files, not binary files.

Scrolling Keys

At the `--More--` prompt, you can use the keys described in Table 4-1 to control scrolling capabilities.

Table 4-1 Scrolling Keys for the `more` Command

Scrolling Keys	Purpose
Spacebar	Scrolls to the next screen
Return	Scrolls one line at a time
`b`	Moves back one screen
`f`	Moves forward one screen
`h`	Displays a help menu of features
`q`	Quits and returns to the shell prompt
`/string`	Searches forward for *`string`*
`n`	Finds the next occurrence of *`string`*

Viewing Long Files

The pg command also provides you the ability to view a file that is longer than one screen.

This command pauses after displaying a screen of text and displays the : prompt at the bottom of the page. Press the Return key to display another screen of text, or use one of the scrolling keys listed in Table 4-2.

The pg command displays an (EOF): prompt when the end of the file is reached. Press the Return key to return to the shell prompt.

Command Format

```
pg filename(s)
```

Scrolling Keys

Table 4-2 shows the keys used to control screen scrolling.

Table 4-2 Scrolling Keys for the pg Command

Scrolling Keys	Purpose
Return	Scrolls to the next screen
l <Return>	Displays the next line
d <Return>	Displays half a page more
.<Return>	Re-displays current page
+/*pattern*/<Return>	Searches forward for *pattern*
$<Return>	Moves to the last page
h<Return>	Displays the help menu of features
q<Return>	Quits and returns to the shell prompt

Note – The pg command does not display binary files.

Displaying the First Few Lines of a File

The `head` command displays the first 10 lines of a file. You can change the number of lines displayed by using the `-n` option.

Command Format

```
head [ -n ] [ filename ... ]
```

Displaying a Specific Number of Lines at the Beginning of a File

To display a specific number of lines at the beginning of a file, execute the following:

```
$ head -6 /usr/dict/words
10th
1st
2nd
3rd
4th
5th
$
```

In this example, the `head` command with the `-6` option displays the first six lines of the `/usr/dict/words` file.

Displaying the Last Few Lines of a File

The `tail` command displays the last 10 lines of one or more files. You can change the number of lines displayed by using the *-n* or *+n* options.

If you use the *-n* option, the `tail` command counts relative to the end of the file. When you use the *+n* option, the `tail` command displays from the nth line to the end of the file.

Command Format

```
tail [ -n ] [ filename ]
tail [ +n ] [ filename ]
```

Displaying a Specified Number of Lines at the End of a File

To display a specified number of lines at the end of a file, execute the following:

```
$ tail -5 /usr/dict/words
zounds
z's
zucchini
Zurich
zygote
$
```

In this example, the `tail` command with the `-5` option displays the last five lines of the `/usr/dict/words` file.

Displaying Lines From a Specific Point in the File

To display lines from a specific point in a file, execute the following:

```
$ tail +23 /usr/dict/words
```

In this example, the `tail` command with the `+23` option displays Line 23 through to the end of the `/usr/dict/words` file.

Displaying Lines, Words, and Characters in a File

The `wc` command displays the number of lines, words, and characters contained in a file.

Command Format

```
wc [ -lwcm ] [ filename ... ]
```

Using the `wc` Command With Options

You can use the following options with the `wc` command:

`-l`	Prints line count
`-w`	Prints word count
`-c`	Prints byte count
`-m`	Prints character count

Using the `wc` Command Without Options

Using the `wc` command without options displays the number of lines, words, and characters contained in the file; for example:

```
$ wc dante
33     223    1320  dante
$
```

Determining the Number of Lines in a File

To determine the number of lines in a file, execute the following:

```
$ wc -l dante
    33  dante
$
```

Creating Empty Files

One use of the `touch` command is creating an empty file. If the file name or directory name already exists, then the `touch` command simply updates its modification and access time to the current date and time.

Command Format

`touch` *`filename ...`*

You can specify absolute and relative path names on the command line when creating new files.

Creating Empty Files

To create an empty file, execute the following:

```
$ cd ~/practice
$ touch mailbox project projection research
$ ls
mailbox            project
projection         research
$
```

Capturing and Displaying Output Using the `tee` *Command*

The `tee` command copies standard input to standard output, writing a duplicate to zero or more files.

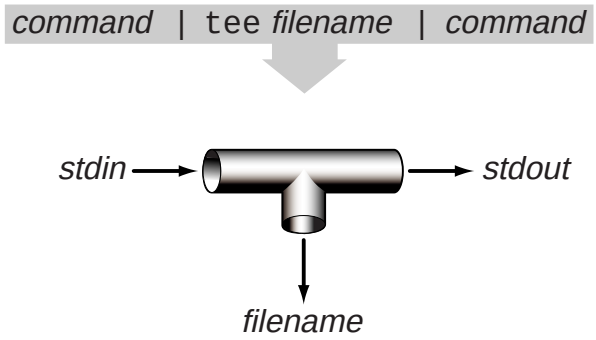


Figure 4-1 The `tee` Command

Command Format

```
tee [ -a ] filename ...
```

Replicating Data

In the following example, the output from the `ls` command is captured in a file called `logfile` and is also displayed on the screen, one screenful at a time.

```
$ ls -lR | tee logfile | more
.:
total 208
-rwxr-----   1 user1 staff       1320 Jun 31 16:44 dante
-rwxr-----   1 user1 staff        368 Jun 31 16:45 dante_1
drwx--x--x   5 user1 staff         96 Jun 14 16:17 dir1
drwx--x--x   4 user1 staff         96 Jun 14 16:17 dir2
drwx--x--x   3 user1 staff         96 Jun 14 16:17 dir3
drwx--x--x   3 user1 staff         96 Jun 14 16:17 dir4
-rwxr-----   1 user1 staff          0 Jun 31 16:45 file.1
-rwxr-----   1 user1 staff          0 Jun 31 16:45 file.2
-rwxr-----   1 user1 staff          0 Jun 31 16:45 file.3
-rwxr-----   1 user1 staff      14502 Jun 14 17:05 file1
-rwxr-----   1 user1 staff       7251 Jun 14 17:05 file2
-rwxr-----   1 user1 staff        218 Jun 31 16:45 file3
-rwxr-----   1 user1 staff        137 Jun 31 16:45 file4
-rwxr-----   1 user1 staff         56 Jun 31 16:45 fruit
-rwxr-----   1 user1 staff         57 Jun 31 16:45 fruit2
drwx--x--x   2 user1 staff         96 Jun 14 16:17 practice
-rwxr-----   1 user1 staff      28738 Jun 31 16:45 tutor.vi
```

Appending Data to a File

If you use the `-a` option with the `tee` command, the new output appends to the file, rather than overwriting the original contents.

For example:

```
$ cal | tee -a logfile
   August 2000
 S  M Tu  W Th  F  S
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31
```

Creating Directories

Use the `mkdir` command to create directories. You can create directories using either absolute or relative path names.

This command also gives you the ability to specify more than one directory name on the same command line, creating several directories at the same time.

Command Format

`mkdir [ -p ] directory_name ...`

Creating a New Directory

In this example, the `mkdir` command creates a new directory in the `home` directory of `user1`.

```
$ cd
$ pwd
/export/home/user1
$ mkdir Reports
$ ls -dl Reports
drwxr-xr-x 2 user1 staff 512 Mar 1 16:24 Reports
$ mkdir Reports/Weekly
$ ls Reports
Weekly
$ cd Reports/Weekly
$ mkdir dir1 dir2 dir3
$ ls -F
dir1/ dir2/ dir3/
$ mkdir ~/games
$ cd
$ ls -F
Reports/   dir1/     dir4/     file.3*   file3*    fruit2*  practice/
dante*     dir2/     file.1*   file1*    file4*    games/   tutor.vi*
dante_1*   dir3/     file.2*   file2*    fruit*    logfile
$
```

Note – Remember the tilde (~) character is available in all shells, except the Bourne shell, to indicate your home directory.

You must have the appropriate permissions to create a directory. (Permissions are described in Module 6, "File Security.") If you do not have the correct permissions, an error message similar to the one shown is displayed:

```
$ mkdir /export/home/Olympic
mkdir: Failed to make directory "/export/home/Olympic"; Permission denied
```

Creating Multiple Levels of Directories

To create multiple levels of directories at one time, use the `-p` option. In the following example, `mkdir -p` creates a directory named `practice2` as a subdirectory of the current directory.

At the same time, it creates a subdirectory named `dir1` as a subdirectory of `practice2`, and a directory named `admin` as a subdirectory of `dir1`.

```
$ mkdir -p practice2/dir1/admin
$ ls -lF
drwx--x--x   3 user1 staff        96 Aug 25 10:38 Reports/
-rwxr-----   1 user1 staff      1320 May 31 16:44 dante*
-rwxr-----   1 user1 staff       368 May 31 16:45 dante_1*
drwx--x--x   5 user1 staff        96 May 31 16:45 dir1/
drwx--x--x   4 user1 staff        96 May 31 16:45 dir2/
drwx--x--x   3 user1 staff        96 May 31 16:45 dir3/
drwx--x--x   3 user1 staff        96 May 31 16:45 dir4/
-rwxr-----   1 user1 staff         0 May 31 16:45 file.1*
-rwxr-----   1 user1 staff         0 May 31 16:45 file.2*
-rwxr-----   1 user1 staff         0 May 31 16:45 file.3*
-rwxr-----   1 user1 staff      1610 Jul 25 14:55 file1*
-rwxr-----   1 user1 staff       105 May 31 16:45 file2*
-rwxr-----   1 user1 staff       218 May 31 16:45 file3*
-rwxr-----   1 user1 staff       137 May 31 16:45 file4*
-rwxr-----   1 user1 staff        56 May 31 16:45 fruit*
-rwxr-----   1 user1 staff        57 May 31 16:45 fruit2*
-rw-------   1 user1 staff      2342 Aug 25 10:36 logfile
drwx--x--x   2 user1 staff      8192 Aug 25 10:36 practice/
drwx--x--x   3 user1 staff        96 Aug 25 10:41 practice2/
-rwxr-----   1 user1 staff     28738 May 31 16:45 tutor.vi*
```

Copying Files and Directories

The cp command copies files and directories.

Copying Files

The cp command copies the contents of a file to another file, or it copies multiple files while preventing overwrites to existing files.

Command Format

```
cp [ -ir ] source_file   destination_file
cp [ -ir ] source_file ...   destination_directory
```

Copying a File to Another File Within a Directory

The following example shows how to copy one file to a new file in the same directory.

```
$ cd
$ pwd
/export/home/user1
$ cp file3 feathers
$ ls
Reports    dir1       dir4       file.2     file2      fruit
practice
dante      dir2       feathers   file.3     file3      fruit2
practice2
dante_1    dir3       file.1     file1      file4      logfile
tutor.vi
$ cp feathers feathers_6
$ ls
Reports     dir2        feathers_6  file1       fruit       practice2
dante       dir3        file.1      file2       fruit2      tutor.vi
dante_1     dir4        file.2      file3       logfile
dir1        feathers    file.3      file4       practice
```

Copying Multiple Files

The following example demonstrates how to copy multiple files into a directory other than the current directory:

```
$ pwd
/export/home/user1
$ ls dir1
coffees  fruit    trees
$ cp feathers feathers_6 dir1
$ ls dir1
coffees   feathers   feathers_6   fruit   trees
$
```

Preventing Overwrites to an Existing File While Copying

Use the `cp` command with the `-i` option as a precaution. The `-i` option prompts for confirmation before overwriting any existing file with the new file.

- Answering with a `yes` response means the copy should proceed.
- Answering with a `no` response prevents `cp` from overwriting the target.

For example:

```
$ cp -i feathers feathers_6
cp: overwrite feathers_6 (yes/no)? n
$
```

Copying Directories

Use the `cp` command with the `-r` option to copy a directory and its contents to another directory. If that directory does not exist, it is created by the `cp` command.

Command Format

`cp -ir` *`source_directory(s) destination_directory`*

When used with the `-i` option, `cp` prompts for verification before overwriting an existing directory or file.

Copying the Contents of a Directory to a New Directory

The following example demonstrates how to copy an existing directory and all of its contents to a new directory in the current working directory.

If you do not use the `-r` option, you will receive the error message: `cp: directoryname: is a directory`.

```
$ cd
$ pwd
/export/home/user1
$ ls dir3
planets
$ cp dir3 new.dir
cp:  dir3: is a directory
$ cp -r dir3 new.dir
$ ls new.dir
planets
$
```

The following example demonstrates how to copy a directory to another directory that is not in the current working directory:

```
$ cd
$ pwd
/export/home/user1
$ cd dir3
$ cp -r planets ../dir1/constellation
$ cd
$ cp -ri dir1 new.dir /tmp
cp: overwrite /tmp/dir1/coffees/beans (yes/no)? y
cp: overwrite /tmp/dir1/constellation/mars (yes/no)? y
cp: overwrite /tmp/dir1/constellation/pluto (yes/no)? y
$ ls -F /tmp
dir1/ new.dir/ ...
$
```

Moving and Renaming Files and Directories

Use the `mv` (move) command to move or rename a file or directory. This command does not affect the contents of the file or directory, it simply changes its location or name from the old name to the new name.

The old name equates to the source, and the new name equates to the target. If the target directory does not exist, it is created.

Command Format

```
mv [ -i ] source target_file

mv [ -i ] source... target_directory
```

The `-i` option prompts for confirmation whenever the move would overwrite an existing target.

- Answering with a `yes` response means the move should proceed.
- Answering with a `no` response prevents `mv` from overwriting the target.

Renaming Files in the Current Directory

The following example demonstrates how to rename a file in the current directory:

```
$ cd ~/dir1/coffees
$ ls
beans    newfile    nuts

$ mv nuts brands

$ ls
beans    brands
$
```

Moving Files to Another Directory

The following example demonstrates how to move a file into another directory:

```
$ pwd
/export/home/user1/dir1/coffees
$ ls
beans   brands
$ mv brands ~
$ ls
beans
$ ls ~/b*
/export/home/user1/brands
```

If you are moving a single directory and the target directory does not exist, the directory is renamed to the target directory.

If you are moving multiple directories and the target directory does not exist, then you will get the error message:
`mv: target_directory not found`.

Renaming Directories

The following example demonstrates how to use the `mv` command to rename directories within the current directory:

```
$ cd
$ mkdir maildir
$ ls
Reports     dir2        file.1      file3       maildir
brands      dir3        file.2      file4       new.dir
dante       dir4        file.3      fruit       practice
dante_1     feathers    file1       fruit2      practice2
dir1        feathers_6  file2       logfile     tutor.vi
$
$ mv maildir monthly
$ ls
Reports     dir2        file.1      file3       monthly
brands      dir3        file.2      file4       new.dir
dante       dir4        file.3      fruit       practice
dante_1     feathers    file1       fruit2      practice2
dir1        feathers_6  file2       logfile     tutor.vi
```

Moving a Directory and its Contents

The following example demonstrates how to use the `mv` command to move a directory and its contents into a new directory:

```
$ pwd
/export/home/user1
$ ls practice
mailbox     project     projection  research
$ mv practice letters
$ ls letters
mailbox     project     projection  research
```

Renaming Files in Another Directory

The following example demonstrates how to use the `mv` command to rename a file in a directory other than the current directory:

```
$ pwd
/export/home/user1
$ mv letters/project letters/project2
$ ls letters
mailbox     project2    projection  research
```

Removing Files and Directories

After a file is no longer needed, you can permanently remove it by using the `rm` command.

Removing Files

You can use the `rm` command to remove a single file or several files at once.

Command Format

```
rm [ -ir ] filename ...
```

Removing Several Files

You can remove several files at the same time; for example:

```
$ cd ~/letters
$ pwd
/export/home/user1/letters
$ ls
mailbox     project2    projection  research
$ rm research project2
$ ls
mailbox     projection
$
```

Use the `rm` command with the `-i` option as a precaution. The `-i` option prompts for confirmation before removing any file.

- Answering with a `yes` response means the deletion should proceed.
- Answering with a `no` response prevents `rm` from overwriting the target.

For example:

```
$ rm -i projection
rm: remove projection: (yes/no) ? y
$ ls
mailbox
$
```

Removing Directories

You can remove unwanted directories by using the `rmdir` and `rm` commands.

- The `rmdir` command deletes empty directories only.
- The `rm -r` command removes a directory that contains files.

Command Format

`rmdir` *`directory_name ...`*

`rm [ -ir ]` *`directory_name ...`*

Removing an Empty Directory

Use the `rmdir` command to remove an empty directory; for example:

```
$ cd
$ pwd
/export/home/user1
$ mkdir -p newdir/empty
$ cd newdir
$ ls -F
empty/
$ rmdir empty
$ ls
$
```

Removing a Directory With Contents

Use `rm -r` to remove a directory that is not empty; for example:

```
$ cd
$ pwd
/export/home/user1
$ ls letters
mailbox
$ rm -r letters
$ ls letters
letters: No such file or directory
$
```

Interactively Removing Directories

Use `rm -ir` to interactively remove directories; for example:

```
$ mkdir -p ~/practice/dir1
$ ls practice
dir1
$ rm -ir ~/practice
rm: examine files in directory practice (yes/no)? y
rm: examine files in directory practice/dir1
(yes/no)? y
rm: remove practice/dir1: (yes/no)? y
rm: remove practice: (yes/no)? y
$
```

Note – You cannot remove a directory while you are in it. You must move to the parent directory to remove a subdirectory.

Command-Line Printing

The `lp` command queues text files for printing.

From the command line, you can print ASCII text or PostScript™ files. Do not use this method to print data files, such as binary files or files created in applications, such as FrameMaker.

Command Format

```
lp [options] filename ...
```

Options

Use the following options with the `lp` command:

`-d` *`destination`*	Specifies the desired printer. If printing to the default printer, the `-d` option is not necessary.
`-o nobanner`	Specifies that the banner page is not to be printed.
`-n` *`number`*	Prints a specified number of file copies.
`-m`	Sends a mail message to you after a job is complete.

Sending Files to a Printer

To print the file `feathers` located in your home directory on the default printer, execute the following command:

```
$ lp ~/feathers
request id is printerA-419 (1 file(s))
$
```

This example demonstrates how you can specify a printer other than the default printer.

To specify another printer (for example, `printerB`), you must use the `-d` option.

```
$ lp -d printerB ~/feathers
request id is printerB-93 (1 file(s))
$
```

Displaying Printer Status and Queues

The `lpstat` command displays the status of the printer queue.

Command Format

```
lpstat [ -podtsa ]
```

Options

Use the following options with the `lpstat` command:

- `-p` Displays the status of all printers
- `-o` Displays the status of all output requests
- `-d` Displays which printer is the system default
- `-t` Displays complete status information for all printers
- `-s` Displays a status summary for all printers
- `-a` Displays which printers are accepting requests

Note – When a print request has been sent to the printer, the output of `lpstat` can show the print request as *filtered*. Filtering indicates that a print request is in the process of printing.

Displaying the Status of All Print Requests

To display the status of all print requests, execute the following:

```
$ lpstat -o
printerA-7    user1   391   Aug 10 16:30     on printerA
printerB-1    user2   551   Aug 10 16:45     filtered
$
```

Displaying Requests on a Specific Printer's Queue

To display requests on a specific printer's queue, execute the following:

```
$ lpstat printerA
printerA user2 551 Aug 10 16:45
$
```

Determining the Status of All Configured Printers

To determine the status of all configured printers, execute the following:

```
$ lpstat -t
scheduler is running
system default destination: printerA
system for printerB: host2
system for _default: host1 (as printer printerA)
system for printerA: host1
printerB accepting requests since Aug 7 09:43 2000
_default accepting requests since Aug 2 08:20 2000
printer printerB is idle. enabled since Aug 7 09:43 2000. available.
printer _default is idle. enabled since Aug 2 08:20 2000. available.
printer printerA is idle. enabled since Aug 2 08:20 2000. available.
$
```

Determining Which Printers Are Configured on the System

To determine which printers are configured on the system, execute the following:

```
$ lpstat -s
scheduler is running
system default destination: printerA
system for printerB: host2
system for _default: host1 (as printer printerA)
system for printerA: host1
$
```

Displaying Which Printers Are Accepting Requests

To display which printers are accepting requests, execute the following:

```
$ lpstat -a
printerB accepting requests since Aug 7 09:43 2000
_default accepting requests since Aug 2 08:20 2000
$
```

Removing a Print Request

The `cancel` command enables you to cancel print requests previously sent with the `lp` command. To do this, first use the `lpstat` command to identify the *`request-ID`* associated with your print request.

Command Format

```
cancel Request-ID ...

cancel -u username
```

Canceling a Print Request

For example, as `user3`, you would execute the following commands to first identify the *`request-ID`* associated with your print request and then cancel it.

```
$ lpstat printerB
printerB-2 user2 551       Aug 10 16:45
printerB-5 user2 552       Aug 10 16:45
printerB-6 user3 632       Aug 10 16:47
$ cancel printerB-6
printerB-6: cancelled
$
```

Use the `cancel -u` *`username`* to remove all requests owned by a specific user; for example:

```
$ cancel -u user2
printerB-5: cancelled
$
```

As the `root` user, you can cancel all print requests owned by all users.

Note – As a regular user, you can only remove your own print jobs. When using the CDE Printer Manager, it appears that you can cancel another person's print job, but the job is redisplayed in the Print Manager the next time it is refreshed.

Format and Print a File

Use the `pr` command to format and print the contents of a file according to different format options. This command automatically prints the file contents to the terminal screen for viewing.

By default, the `pr` command prints a header, up to 66 lines of text, and a trailer that contains five blank lines, for each page.

The header of each page includes the name of the file, the date and time last modified, and a page number.

Command Format

```
pr [ options ] filename ...
```

Options

You can use the following options when executing the `pr` command:

`+page`	Begins printing on page number specified.
`-column`	Prints file in multi-column format. The default is one column.
`-d`	Prints file in double-spaced format.
`-h header`	Replaces the file name contained in the page header with the pattern *header*.
`-l lines`	Resets page length to *lines*, the default is 66 lines.
`-m`	Merges files side-by-side into text columns. The columns are of equal width. Lines of data that do not fit within a column are truncated.
`-t`	Prints files without the header.
`-n`	Numbers each line.

Format and Print Files to the Screen

The following examples print the file contents to the terminal screen, using the various `pr` format options.

- To begin formatting a file on Page 4:

```
$ pr +4 tutor.vi | more

Mar  1 14:58 2000  tutor.vi Page 4

  1. Press  <ESC>  to make sure you are in Command Mode.

  2. Move the cursor to the line below marked --->.

  3. Move the cursor to the end of the correct line (AFTER the first . ).

  4. Type d$ to delete to the end of the line.
<output omitted>
```

- To format a file into three text columns:

```
$ pr -3 /usr/dict/words | more

Sep  1 05:01 1998  /usr/dict/words Page 1

10th                 abject               abstinent
1st                  ablate               abstract
2nd                  ablaze               abstracter
3rd                  able                 abstractor
4th                  ablution             abstruse
5th                  Abner                absurd
6th                  abnormal             abuilding
```

- To send a file to the default printer:

```
$ pr file3 | lp
```

- To merge three files side-by-side in text columns:

```
$ pr -m fruit fruit2 file2 | more

Aug 21 17:09 2000   Page 1

lemon                   lemon                   bill
orange                  orange                  joe
apple                   apple                   tom
banana                  banana                  don
pear                    tomato                  liz
mango                   guava                   kim
tomato                  mango                   ann
pomegrante              pomegrante              darlene
                                                syndey
                                                kathy
                                                henry
                                                richard

                                                All of the above are me
```

- To print two columns, double-spaced, with the header "TREES" for `fruit` and `fruit2` on the default printer:

  ```
  $ pr -2h "TREES" fruit fruit2 | lp
  ```

Exercise: Using Directory and File Commands

Exercise objective – In this exercise, you use the commands described in this module to display the contents of files; compare files; determine the file type; and create, move, and remove files and directories.

Tasks

Complete or answer the following:

1. Determine the file type of `/etc/passwd`, and display the contents of the file.

   _____

2. Display the contents of the file `/usr/dict/words` one screen at a time using the `more` command. Exit after displaying two screens.

   _____

3. Display the contents of the file `/usr/dict/words` on the screen using the `pg` command.

   a. Display half a page more. _____

   b. Display the next line. _____

   c. Move to the last page. _____

   d. Quit and return to the shell prompt. _____

4. Display only the first five lines of the file `/usr/dict/words` on the screen.

   _____

5. Display only the last eight lines of the file `/usr/dict/words` on the screen.

   _____

6. Print the file `/usr/dict/words` to the screen in an eight-column format.

_____

7. Determine the total number of lines contained in the file `/usr/dict/words`.

_____

8. What command would you most likely use to read the contents of the binary file `/usr/bin/cp`?

_____

9. Return to your `home` directory (if you need to), and list the contents.

_____

10. Copy the `dir1/coffees/beans` file into the `dir4` directory, and call it `roses`.

_____

11. Create a directory called `vegetables` in `dir3`.

_____

12. Move the `dir1/coffees/beans` file into the `dir2/recipes` directory.

_____

13. From your home directory, make a directory called `practice1`.

_____

14. Copy `dir3/planets/mars` to the `practice1` directory, and name the file `addresses`.

_____

15. Create a directory called `play` in your `practice1` directory, and move the `practice1/addresses` file to the `play` directory.

_____

16. Copy the `play` directory in the `practice1` directory to a new directory in the `practice1` directory called `appointments`.

_____

17. Recursively list the contents of the `practice1` directory.

_____

18. In your home directory and with one command, create a directory called `house` with a subdirectory of `furniture`.

_____

19. Create an empty file called `chairs` in the new directory `furniture`.

_____

20. Using one command, create three directories called `letters`, `memos`, and `misc` in your home directory.

_____

21. Using one command, delete the directories called `memos` and `misc` in your home directory.

_____

22. Try to delete the directory called `house/furniture` with the `rm` (no options) command. What happens?

_____

23. Identify the command to delete a directory that is not empty. Delete the directory `house/furniture`. List the contents of `house` to verify that `furniture` has been deleted.

_____

24. From your home directory, redirect the output of the `ls` command to a file called `file.list`.

_____

25. Display the contents of `file.list` using the `cat` command.

_____

26. Display a calendar and, using the `tee` command, append the output to `file.list`.

_____

27. Check the print queue to display the status of all print requests.

_____

28. Cancel a print request, removing it from the print queue.

_____

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercises.

- Experiences
- Interpretations
- Conclusions
- Applications

Task Solutions

1. Determine the file type of `/etc/passwd`, and display the contents of the file.

   ```
   file /etc/passwd
   cat /etc/passwd
   ```

2. Display the contents of the file `/usr/dict/words` one screen at a time, using the `more` command. Exit after displaying two screens.

   ```
   more /usr/dict/words
   ```

   To display two screens, press the spacebar twice.
   To exit, type `q` *or Control-C.*

3. Display the contents of the file `/usr/dict/words` on the screen using the `pg` command. Display half a page more. Display the next line. Move to the last page. Quit and return to the shell prompt.

   ```
   pg /usr/dict/words
   ```

   To display half a page more, type **`d`** *at the* `:` *prompt.*
   To display the next line, type **`l`** *at the* `:` *prompt.*
   To move to the last page, type **`$`** *at the* `:` *prompt.*
   To quit and return to the shell prompt, type **`q`**..

4. Display only the first five lines of the file `/usr/dict/words` on the screen.

   ```
   head -5 /usr/dict/words
   ```

5. Display only the last eight lines of the file `/usr/dict/words` on the screen.

   ```
   tail -8 /usr/dict/words
   ```

6. Print the file `/usr/dict/words` to the screen in an eight-column format.

   ```
   pr -8 /usr/dict/words
   ```

7. Determine the total number of lines contained in the file `/usr/dict/words`.

   ```
   wc -l /usr/dict/words
   ```

8. What command would you most likely use to read the contents of the binary file `/usr/bin/cp`?

   `strings /usr/bin/cp`

9. Return to your home directory (if you need to), and list the contents.

   `cd; ls`

10. Copy the `dir1/coffees/beans` file into the `dir4` directory, and call it `roses`.

    `cp dir1/coffees/beans dir4/roses`

11. Create a directory called `vegetables` in `dir3`.

    `mkdir dir3/vegetables`

12. Move the `dir1/coffees/beans` file into the `dir2/recipes` directory.

    `mv dir1/coffees/beans dir2/recipes`

13. From your home directory, make a directory called `practice1`.

    `mkdir practice1`

14. Copy `dir3/planets/mars` to the `practice1` directory, and name the file `addresses`.

    `cp dir3/planets/mars practice1/addresses`

15. Create a directory called `play` in your `practice` directory, and move the `practice1/addresses` file to the `play` directory.

    `mkdir practice/play`
    `mv practice1/addresses practice1/play`

16. Copy the `play` directory in the `practice1` directory to a new directory in the `practice1` directory called `appointments`.

    `cp -r practice1/play practice1/appointments`

17. Recursively list the contents of the `practice1` directory.

    `ls -R practice1`

18. In your home directory and with one command, create a directory called `house` with a subdirectory of `furniture`.

    `cd; mkdir -p house/furniture`

19. Create an empty file called `chairs` in the new directory `furniture`.

    `touch house/furniture/chairs`

20. Using one command, create three directories called `letters`, `memos`, and `misc` in your home directory.

    `mkdir letters memos misc`

21. Using one command, delete the directories called `memos` and `misc` in your home directory.

    `rmdir memos misc`

22. Try to delete the directory called `house/furniture` with the `rm` (no options) command. What happens?

    `rm house/furniture`
    `rm: house/furniture is a directory`

23. Identify the command to delete a directory that is not empty. Delete the directory `house/furniture`. List the contents of `house` to verify that `furniture` has been deleted.

    `rm -r house/furniture; ls house`

24. From your home directory, redirect the output of the `ls` command to a file called `file.list`.

    `ls > file.list`

25. Display the contents of `file.list` using the `cat` command.

    `cat file.list`

26. Display a calendar and, using the `tee` command, append the output to `file.list`.

    `cal | tee -a file.list`

27. Check the print queue to display the status of all print requests.

    `lpstat -o`

28. Cancel a print request, removing it from the print queue.

    `cancel` ***`request-id`***

Check Your Progress

Before continuing on to the next module, check that you are able to accomplish or answer the following:

- ❑ Determine file types with the `file` command
- ❑ Display the contents of text files using the `cat`, `more`, `pg`, `head`, and `tail` commands
- ❑ Determine character, word, and line counts using the `wc` command
- ❑ Create empty files or update modification times on existing files using the `touch` command
- ❑ Use the `tee` command to create text within a file
- ❑ Create and remove directories using the `mkdir` and `rmdir` commands
- ❑ Manage files and directories using the `mv`, `cp`, and `rm` commands
- ❑ Use commands to print a file, check print queue status, and cancel a print request
- ❑ Format and print the contents of files using the `pr` command

Searching for Files and Text 5

Objectives

Upon completion of this module, you should be able to:

- Use the `find` command to locate files in the Solaris Operating Environment directory tree using specific search criteria
- Use the `cmp` and `diff` commands to compare the contents of files for differences
- Sort the content of text files in alphabetical and numerical order using the `sort` command
- Search for regular expressions in the contents of one or more files using the commands `grep`, `egrep`, and `fgrep`

Additional Resources

Additional resources – The following reference provides additional details on the topics discussed in this module:

- *System Administration Guide, Volume 1*, Part Number 805-7228-10

Locating Files Using the `find` *Command*

To locate a file in the directory tree, use the `find` command. This command gives you the ability to locate files based on specific criteria, such as file name, size, owner, modification time, or type.

The `find` command recursively descends the directory tree in the path name list, seeking those files that match the criteria.

As `find` locates the files that match those values, the path to each file is displayed on the screen.

Command Format

```
find pathname(s) expression(s) action(s)
```

The first argument on the command line is the path name where the search starts; this can be an absolute or a relative path name.

The remainder of the arguments specify the criteria by which to find the files, and what action to take after they are located.

Table 5-1 through Table 5-3 on page 5-4 show the arguments, expressions, and actions that can be used with the `find` command.

Table 5-1 Arguments Used With the `find` Command

Argument	Definition
pathname	The directory path in which the search originates.
expression	The search criteria specified by one or more options. Specifying multiple options causes `find` to treat the statement as an "AND" request, so all listed expressions must be verified as true.

Table 5-2 Expressions Used With the `find` Command

Expression	Definition
`-name` *`filename`*	Finds files matching the specified *`filename`*. Metacharacters are acceptable if placed inside quotes.
`-size [+|-]`*`n`*	Finds files that are larger than *`+n`*, smaller than *`–n`*, or exactly *`n`*. The *`n`* represents 512-byte blocks.
`-atime [+|-]`*`n`*	Finds files that have been accessed more than *`+n`*, less than *`-n`*, or exactly *`n`* days.
`-mtime [+|-]`*`n`*	Finds files that have been modified more than *`+n`*, less than *`-n`*, or exactly *`n`* days.
`-user` *`loginID`*	Finds all files that are owned by the *`login ID`* name.
`-type`	Finds a file type; for example, `f` (file) or `d` (directory).
`-perm`	Finds files that have certain access permission bits.

Table 5-3 Actions Used With the `find` Command

Action	Definition
`-exec` *`command`* `{} \;`	Executes the specified *`command`* on each file located, automatically. A set of braces, "`{}`", delimits where the file name is passed to the command from the preceding expressions. A space, backslash, and semicolon, "` \;`", delimits the end of the command. There must be a space before the backslash (\).
`-ok` *`command`* `{} \;`	Interactive form of `-exec`. It requires input before `find` applies the *`command`* to the file; otherwise, it behaves as the "`-exec`" action.
`-print`	Instructs `find` to print the current path name to the terminal screen. This is the default.
`-ls`	Prints the current path name, together with its associated statistics, such as inode number, size in kilobytes, protection mode, number of hard links, and user.

The following examples illustrate the power of the `find` command.

- To search for `core` files starting at the `root` (`/`) directory:

  ```
  $ find / -name core
  ```

- To search for `core` files starting at your home directory and delete each one as it is found:

  ```
  $ find ~ -name core -exec rm {} \;
  ```

- To look for all files, starting at the current directory that have not been modified in the last 90 days:

  ```
  $ find . -mtime +90
  ```

- To find files larger than 57 blocks (512-byte blocks) starting at the home directory:

  ```
  $ find ~ -size +57
  ```

- To search for files that end with the characters "`tif`," starting at the `/usr` directory:

  ```
  $ find /usr -name *tif
  ```

Finding Differences Between Files

There are several utilities that compare two files and report differences, if any, between the two files.

Locating Differences Using the `cmp` Command

Comparing files to determine differences is easily accomplished using the `cmp` command. This command prints results only if there are differences between the files. If no results are shown, the files are exactly the same.

The `cmp` command performs a byte-by-byte comparison of each file. If the bytes within the files differ, then `cmp` prints the byte count and the line number of the first difference and then stops.

This command works with both binary and ASCII files.

Command Format

```
cmp filename filename
```

Using the `cmp` Command to Compare Files That Appear the Same

```
$ cmp fruit fruit2
fruit fruit2 differ: char 27, line 5
```

This output identifies the first occurrence of a difference between the two files. The difference occurred at the 27th character position and was found on Line 5.

Locating Text Differences Using the `diff` Command

The `diff` command is another command used for finding differences between files.

The results of this command display line-by-line differences between two text files, providing you with instructions on how to edit one file to make it the same as the other.

Command Format

```
diff -option filename filename
```

Table 5-4 shows the options that can be used when using the `diff` command.

Table 5-4 Options for the `diff` Command

Options	Definition
`-i`	Ignores the case of letters; for example, `A` is equal to `a`.
`-c`	Produces a contextual listing of differences.

Using `diff` *With the* `-c` *option*

When using `diff` with the `-c` option to compare files, the results are displayed in three sections.

The first section shows the names of the files being compared and their creation dates, followed by a string of asterisks (*).

The second section shows, for `file1`, a line count of those lines that differ from `file2`, followed by a comma and a total line count for all lines contained in `file1`. Actual text lines from `file1` are printed with each line that differs from `file2` being tagged with a dash (`-`) character. At most, three lines preceding the first difference are displayed as contextual information.

The third section shows, for `file2`, a line count of those lines that differ from `file1`, followed by a comma and a total line count for all lines contained in `file2`. Actual text lines from `file2` are printed with each line that differs from `file1` being tagged with a plus (`+`) character.

Note – In the `diff -c` example that follows, up to three lines previous to the first difference are displayed so that the difference is displayed in its proper context.

For example, execute the following to use `diff` to compare files:

```
$ cat fruit
lemon
orange
apple
banana
pear
mango
tomato
pomegranate
$ cat fruit2
lemon
orange
apple
banana
tomato
guava
mango
pomegranate
$ diff -c fruit fruit2
 *** fruit       Wed May 31 16:45:05 2000
 --- fruit2      Wed May 31 16:45:05 2000
 ***************
 *** 2,8 ****
   orange
   apple
   banana
 - pear
 - mango
   tomato
   pomegranate
 --- 2,8 ----
   orange
   apple
   banana
   tomato
 + guava
 + mango
   pomegranate
$
```

Sorting Data

The `sort` command sorts text lines in one or more files and prints the results to the screen.

The `sort` command provides a quick and easy method to organize data in numeric or alphabetic order.

By default, `sort` relies on white space to delimit the various fields within the data of a file.

Command Format

```
sort -options filename(s)
```

Options

The options available with the `sort` command are used to define the type of sort to be performed, and identify which field to begin the sorting.

Table 5-5 Options for Using the `sort` Command

Option	Definition
`-n`	Performs a numeric sort.
`(+|-)`*`n`*	Begins (`+`*`n`*) or end (`-`*`n`*) the sort with the field following the *`n`* field.
`-r`	Reverses the order of the sort.
`-f`	Folds uppercase and lowercase characters together; ignores the case in the sort order.
`-M`	Sorts the first three characters of the field as an abbreviated month name.
`-d`	Uses dictionary order. Only letters, digits, and spaces are compared; all other symbols are ignored.
`-o` *`filename`*	Prints results into the file *`filename`*.
`-b`	Ignores leading blank characters when determining the starting and ending positions of a restricted sort key.
`-t` *`char`*	Uses *`char`* as the field separator character. If `-t` is not specified, blank characters are used as default field separators.

Using `sort` *With Different Options*

The following examples show various ways to use the `sort` command with different options.

```
$ cat fileA
Annette         48486
Jamie           48481
Fred            48487
Sondra          48483
Janet           48482

$ sort fileA
Annette         48486
Fred            48487
Jamie           48481
Janet           48482
Sondra          48483

$ sort +1n fileA
Jamie           48481
Janet           48482
Sondra          48483
Annette         48486
Fred            48487
```

The first example shows the contents of `fileA` using the `cat` command.

The first `sort` command produces an alphabetic sort, beginning with the first character of each line.

The second sort produces a numeric sort on the second field (`sort` skips one separator with the `+1` syntax).

Using `sort` *on Different Fields Within a File*

The following examples show how to use the `sort` command on different fields within a file.

```
$ ls -l f* > list
$ cat list
-rw-r--r--   1 user1    staff          0 Feb 25 12:54 file.1
-rw-r--r--   1 user1    staff          0 Feb 25 12:54 file.2
-rw-r--r--   1 user1    staff          0 Feb 25 12:54 file.3
-rw-r--r--   1 user1    staff       1696 Feb 22 14:51 file1
-rw-r--r--   1 user1    staff        156 Mar  1 14:48 file2
-rw-r--r--   1 user1    staff        218 Feb 22 14:51 file3
-rw-r--r--   1 user1    staff        137 Feb 22 14:51 file4
-rw-r--r--   1 user1    staff         56 Feb 22 14:51 fruit
-rw-r--r--   1 user1    staff         57 Feb 22 14:51 fruit2
$
$ sort -rn +4 list -o num.list
$ cat num.list
-rw-r--r--   1 user1    staff       1696 Feb 22 14:51 file1
-rw-r--r--   1 user1    staff        218 Feb 22 14:51 file3
-rw-r--r--   1 user1    staff        156 Mar  1 14:48 file2
-rw-r--r--   1 user1    staff        137 Feb 22 14:51 file4
-rw-r--r--   1 user1    staff         57 Feb 22 14:51 fruit2
-rw-r--r--   1 user1    staff         56 Feb 22 14:51 fruit
-rw-r--r--   1 user1    staff          0 Feb 25 12:54 file.3
-rw-r--r--   1 user1    staff          0 Feb 25 12:54 file.2
-rw-r--r--   1 user1    staff          0 Feb 25 12:54 file.1
$
$ sort +5M +6n list -o update.list
$ cat update.list
-rw-r--r--   1 user1    staff         56 Feb 22 14:51 fruit
-rw-r--r--   1 user1    staff         57 Feb 22 14:51 fruit2
-rw-r--r--   1 user1    staff        137 Feb 22 14:51 file4
-rw-r--r--   1 user1    staff        218 Feb 22 14:51 file3
-rw-r--r--   1 user1    staff       1696 Feb 22 14:51 file1
-rw-r--r--   1 user1    staff          0 Feb 25 12:54 file.1
-rw-r--r--   1 user1    staff          0 Feb 25 12:54 file.2
-rw-r--r--   1 user1    staff          0 Feb 25 12:54 file.3
-rw-r--r--   1 user1    staff        156 Mar  1 14:48 file2
$
```

The first example, shown previously, takes the output of the `ls` command and places it in the file named `list`. The contents of this file are then viewed with the `cat` command.

The first `sort` command produces a reverse numeric order sort on the fifth field and places the results into a file called `num.list`.

The second `sort` example represents a multi-level sort on Fields 6 and 7 in the file named `list`.

- The option `+5M` performs an alphabetic sort on the month in the sixth field.
- The `+6n` option performs a second-level numeric sort on the day in the seventh field and places all results in the `update.list` file.

Searching for Text in Files

The Solaris Operating Environment provides a family of commands used to search the contents of one or more files for a specific character pattern. A *pattern* can be a single character, a string of characters, a word, or a sentence.

By definition, the pattern of characters used to match the same characters in a search is called a regular expression (RE).

- The `grep` command globally searches for regular expressions in files and prints all lines containing the regular expression to standard output.

Note – The `grep` command is derived from the phrase "globally search for a regular expression and print if found." Originally the command was `g/re/p`.

- The `egrep` and `fgrep` commands are variants of `grep`; `egrep` uses extended regular expressions, and `fgrep` uses fixed strings rather than regular expressions

Using the `grep` *Command*

The `grep` command searches the contents of one or more files for a regular expression or character pattern. If found, `grep` prints every line containing the pattern to the screen and does not change the file content in any way.

Command Format

```
grep option(s) pattern filename(s)
```

Options

The `grep` command provides a number of options to modify the way it does its search or displays lines.

Some useful options are described in Table 5-6.

Table 5-6 Options for `grep`

Option	Definition
`-i`	Ignores case. Uppercase and lowercase characters are considered identical.
`-l`	Lists only the names of files with matching lines.
`-n`	Precedes each line with its relative line number in the file.
`-v`	Inverts the search to display only lines that do not match the *`pattern`*.
`-c`	Prints only a count of the lines that contain *`pattern`*.
`-w`	Searches for the expression as a word, ignoring those matches that are substrings of larger words.

Examples of Searching for Regular Expressions With `grep`

The following are examples of using `grep` to search for regular expression:

- To search for all lines containing the pattern "root" in the `/etc/group` file, execute the following:

```
$ grep -n root /etc/group
1:root::0:root
3:bin::2:root,bin,daemon
4:sys::3:root,bin,sys,adm
5:adm::4:root,adm,daemon
6:uucp::5:root,uucp
7:mail::6:root
8:tty::7:root,tty,adm
9:lp::8:root,lp,adm
10:nuucp::9:root,nuucp
12:daemon::12:root,daemon
$
```

- To search for all lines that do not contain the pattern "root" in the `/etc/group` file, execute the following:

```
$ grep -v root /etc/group
other::1:
staff::10:
sysadmin::14:
nobody::60001:
noaccess::60002:
nogroup::65534:
...
$
```

- To search for just the names of the files that contain the pattern "root," execute the following:

```
$ cd /etc
$ grep -l root group passwd hosts
group
passwd
$
```

Note – For multiple file searches, the results are listed with the file name in which the pattern was found. For single file searches, only the results are displayed.

- To search for the pattern "the" in all files in the `/etc` directory, listing only the names of files with lines that match the pattern "the" or "The," execute the following:

```
$ cd /etc
$ grep -li the *
aliases
asppp.cf
dacf.conf
device.tab
devlink.tab
dgroup.tab
fmthard
format

<output omitted>

syslog.conf
system
termcap
TIMEZONE
ttysrch
umountall
$
```

- To search for the pattern "root" in the `/etc/group` file, printing only a count of the lines that contain the pattern, execute the following:

```
$ grep -c root group
10
$
```

- To search for the pattern "mar 1" in the output from the `ls -la` command, execute the following:

```
$ ls -la | grep -i 'mar  1'
prw-------   1 root     root           0 Mar  1 11:05 initpipe
-r--r--r--   1 root     root         806 Mar  1 13:39 mnttab
prw-------   1 root     root           0 Mar  1 11:06 utmppipe
$
```

If the date is a single numeral, the `grep` pattern must have two spaces placed between the month and day. For example: `Mar  1.`

Regular Expression Metacharacters

The `grep` command supports several regular expression (RE) metacharacters to further define a search pattern. Table 5-7 describes some useful metacharacters.

Table 5-7 Regular Expression Metacharacters

Metacharacter	Purpose	Sample	Result
`^`	Beginning of line marker	`'^pattern'`	Matches all lines beginning with "pattern"
`$`	End of line anchor	`'pattern$'`	Matches all lines ending with "pattern"
`.`	Matches one character	`'p.....n'`	Matches lines containing a "p," followed by five characters, followed by an "n"
`*`	Matches preceding item zero or more times	`'[a-z]*'`	Matches lowercase alphanumeric characters
`[ ]`	Matches one character in the pattern	`'[Pp]attern'`	Matches lines containing "Pattern" or "pattern"
`[^]`	Matches one character not in the pattern	`'[^a-m]attern'`	Matches lines not containing "a" through "m," followed by "attern"

Examples

The following are examples of using regular expression metacharacters:

- To print all lines that begin with the word “root” in the `/etc/passwd` file:

  ```
  $ grep '^root' /etc/passwd
  ```

- To print all lines containing an “A,” followed by three characters, followed by an “n” in the `/etc/passwd` file:

  ```
  $ grep 'A...n' /etc/passwd
  ```

- To print all lines that end with the word “adm” in the `/etc/group` file:

  ```
  $ grep 'adm$' /etc/group
  ```

Using the `egrep` *Command*

The `egrep` command searches the contents of one or more files for a pattern using extended regular expression metacharacters.

The `egrep` command uses some new regular expression metacharacters in addition to all the metacharacters used by `grep`.

Command Format

```
egrep [ -options ] pattern filename ...
```

Extended Regular Expression Metacharacters

Table 5-8 show the regular expression metacharacters that are new with `egrep`.

Table 5-8 Extended Regular Expression Metacharacters

Metacharacter	Purpose	Sample	Result
`+`	Matches one or more of the preceding characters	`'[a-z]+ark'`	Matches one or more lowercase letters followed by "`ark`" (for example, `airpark`, `bark`, `dark`, `landmark`, `shark`, `sparkle`, `trademark`)
`x|y`	Matches either `x` or `y`	`'apple|orange'`	Matches for either expression
`( | )`	Groups characters	`'(1|2)+'` `'search(es|ing)+'`	Matches for one or more occurrences (for example, `1` or `2`, `searches`, or `searching`)

Examples Using `egrep`

The following examples show various ways to use the `egrep` command:

- To search for all lines containing the pattern" N," followed by either an "e" or "o" one or more times, execute the following:

```
$ egrep 'N(e|o)+' /etc/passwd
listen:x:37:4:Network Admin:/usr/net/nls:
nobody:x:60001:60001:Nobody:/:
noaccess:x:60002:60002:No Access User:/:
```

- To search for lines containing the pattern "Network Admin" or "uucp Admin," execute the following:

```
$ egrep '(Network|uucp) Admin' /etc/passwd
uucp:x:5:5:uucp Admin:/usr/lib/uucp:
nuucp:x:9:9:uucp Admin:/var/spool/uucppublic:/usr/lib/uucp/uucico
listen:x:37:4:Network Admin:/usr/net/nls:
```

Using the `fgrep` *Command*

The `fgrep` command searches a file for a fixed string. It differs from `grep` and `egrep` because it regards all characters literally, and it does not interpret any RE metacharacters specified on the command line. It recognizes only the literal meaning of these characters; treating a (?) as a question mark, and a ($) as a dollar sign.

Use `fgrep` to search for a specific pattern in a file that includes metacharacter symbols.

Command Format

`fgrep` ***`option(s) pattern filename(s)`***

Example Using `fgrep`

The following example shows how to use the `fgrep` command:

- To search for all lines in the file containing the string of text and symbols, execute the following:

  ```
  $ fgrep '*' /etc/system
  ```

Exercise: Locating Files and Text

Exercise objective – In this exercise, you practice searching for files and directories using the `find` command and displaying and manipulating text in files.

Tasks

Complete the following steps. In the space provided, write the commands you would use to perform each task.

1. Starting in your home directory, find all files ending with a "2."

   _____

2. Use the `find` command to search the `/usr` directory, and display the file names of any length that end with "ln."

   _____

3. Starting in your home directory, find all files of type `file`, printing the full path name of each file located.

   _____

4. In your home directory, find all files of type `directory`.

   _____

5. Find all files in the `/etc` directory that have access permissions 644 (read and write for the owner and read for the group and others).

   _____

6. From your home directory, find ordinary files of size 0 (zero) in the `/tmp` directory, and ask if it is permissible to remove them.

   _____

7. Display all the files created in your `home` directory using `ls`, `grep`, and today's date.

_____

8. Search for the text string "other" in the `/etc/group` file, and have it display to the screen.

_____

9. Using the `grep` command, look for all lines in the `file4` file located in your home directory that do not contain the letter "M."

_____

10. Display all lines in the files `dante`, `file1`, and `dante_1` that contain the pattern "he."

_____

11. Display all the lines in the file `file4` that contain the pattern either "Sales" or "Finance."

_____

12. Sort the lines in the file `fruit` alphabetically.

_____

13. Sort the file `fruit` in reverse order.

_____

Note – To create a file for use in the next step, execute:
`$ ls -la > ls.output`

---

14. Sort the `ls.output` file. Produce a numerical listing by the size of the files, in reverse order. What command did you use?

_____

15. Perform a multilevel sort of the `ls.output` file that places the data in chronological order, then alphabetically by name. What command did you use?

_____

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercises.

- Experiences
- Interpretations
- Conclusions
- Applications

Task Solutions

Complete the following steps. In the space provided, write the commands you would use to perform each task.

1. In your home directory, find all files ending with a "2."

   ```
   $ find ~ -name '*2'
   ```

2. Use the `find` command to search the `/usr` directory, and display the file names of any length that end with "ln."

   ```
   $ find /usr -name '*ln'
   ```

3. Starting in your home directory, find all files of type `file`, printing the full path name of each file located.

   ```
   $ find ~ -type f
   ```

4. In your home directory, find all files of type `directory`.

   ```
   $ find ~ -type d
   ```

5. Find all files in the `/etc` directory that have access permissions 644 (read and write for the owner and read for the group and others).

   ```
   $ find /etc -perm 644
   ```

6. From your home directory, find ordinary files of size 0 (zero) in the `/tmp` directory, and ask if it is permissible to remove them.

   ```
   $ find /tmp -type f -size 0 -ok rm {} \;
   ```

7. Display all the files created in your `home` directory using `ls`, `grep`, and today's date; for example:

   ```
   $ ls -la | grep 'Jun  9'
   ```

8. Search for the text string "other" in the `/etc/group` file, and have it display to the screen.

   ```
   $ grep 'other' /etc/group
   ```

9. Using the `grep` command, look for all lines in the `file4` file located in your home directory that do not contain the letter "M."

   ```
   $ grep -v 'M' file4
   ```

10. Display all lines in the files `dante`, `file1`, and `dante_1` that contain the pattern "he."

    ```
    $ grep he dante file1 dante_1
    ```

11. Display all the lines in the file `file4` that contain the pattern either "Sales" or "Finance."

    ```
    $ egrep 'Sales|Finance' file4
    ```

12. Sort the lines in the file `fruit` alphabetically.

    ```
    $ sort fruit
    ```

13. Sort the file `fruit` in reverse order.

    ```
    $ sort -r fruit
    ```

Note – To create a file for use in the next step, execute:
`$ ls -la > ls.output`

---

14. Sort the `ls.output` file. Produce a numerical listing by size the of the files, in reverse order. What command did you use?

    ```
    $ sort -rn +4 ls.output
    ```

15. Perform a multilevel sort of the `ls.output` file that places the data in chronological order, then alphabetically by name. What command did you use?

    ```
    $ sort +5M +6n +7 +8d ls.output
    ```

    If the time of day created is an issue, the answer would be:

    ```
    $ sort +5M +6n +7n +7.3n +8d ls.output
    ```

Check Your Progress

Before continuing on to the next module, check that you are able to accomplish or answer the following:

- Use the `find` command to locate files in the Solaris Operating Environment directory tree using specific search criteria
- Use the `cmp` and `diff` commands to compare the contents of files for differences
- Sort the content of text files in alphabetical and numerical order using the `sort` command
- Search for regular expressions in the contents of one or more files using the commands `grep`, `egrep`, and `fgrep`

File Security 6

Objectives

Upon completion of this module, you should be able to:

- Display file and directory permissions
- Define the standard permission types (read/write/execute)
- Use the `chmod` command to change permissions with symbolic mode or octal mode values
- Determine the default permissions assigned to newly created files and directories with `umask`

Additional Resources

Additional resources – The following reference provides additional details on the topics discussed in this module:

- *System Administration Guide, Volume 1*, Part Number 805-7228-10

Security Overview

The most important function in a secure system is the ability to deny access to unauthorized users while maintaining access for authorized users. Maintaining a secure system is a primary function of the system administrator, but it is a responsibility of the user as well.

The Solaris Operating Environment provides two basic measures to prevent unauthorized access to a system and to protect its data.

The first measure is to authenticate a user's login by verifying that the login ID (user name) and password exist in `/etc/passwd` and `/etc/shadow`.

The second measure is to automatically protect file and directory access by placing a standard set of access permissions when files and directories are created.

Note – The Solaris Operating Environment also provides a special user account on every system called `root`. The `root` user, often referred to as *superuser*, has complete access to every user account and all files and directories. The `root` user can override the permissions placed on files and directories.

Viewing File and Directory Permissions

To view the permissions on files and directories, use the `ls -l` command.

The first field of information that this command displays defines the type of file, followed by three distinct classes of users and their access permissions.

- File type – Identifies whether the item is a directory or a file
- User – Lists access permissions for the owner
- Group – Lists access permissions for a group of users; defined by the system administrator
- Others (world) – Lists access permissions for all other users

Figure 6-1 illustrates permissions for each class of user.

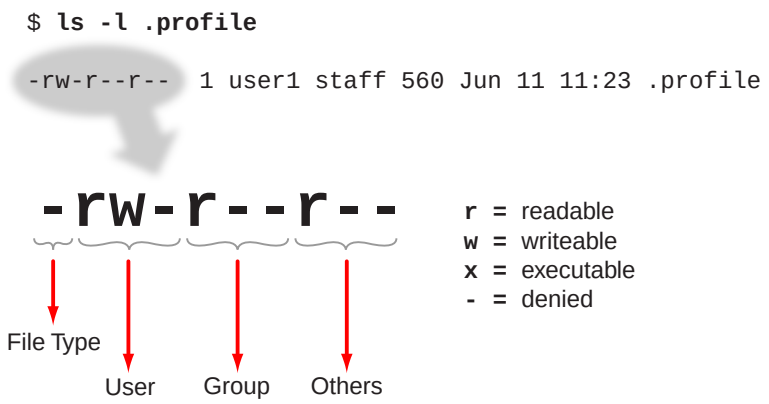


Figure 6-1 Permissions for Each Class of User

Permission Categories

The following sections describe the permission categories.

File Type

The first character in the `ls -l` listing defines the *file type*.

A directory file type is represented by the letter `d`.

An ordinary file's file type is represented by a hyphen (–).

A hyphen, sometimes referred to as a dash, located anywhere else in the permission set indicates that a particular permission is denied.

User (Owner) Permissions

The next three characters are the *owner's permissions*. These indicate what type of access the owner has on the file. In Figure 6-1 on page 6-4, `user1`, the owner of this file, has read and write permissions.

Group

The second set of three characters are the *group permissions.* These identify the permissions being granted or denied for each user who is a member of the group that owns this file.

A group is a set of users who need to access the same files. All users in the same group can access each others' files based on these group permissions.

The system administrator creates and maintains the groups in the `/etc/group` file and assigns users to groups according to shared file access.

In Figure 6-1 on page 6-4, the file belongs to a group called `staff`, and all users who are members of this group have permission to read this file.

Others (World)

The third set of three characters are the *other permissions*. These define the permissions for everyone else.

Other is any user who is not the file owner. nor a member of the group that owns the file, but who has access to the system. In Figure 6-1 on page 6-4, others have read-access to the file.

Determining Access to a File or Directory

Access to a file or a directory is determined by the user identification number (UID) and the group identification number (GID).

- UID – Identifies the user who created the directory or file and determines ownership.
- GID – Identifies the group of users who own the directory or file. A file or directory can belong to only one group at a time.

All files and directories contain a UID and GID number. The Solaris Operating Environment uses these numbers to track file and directory ownership and group membership.

To view these UID and GID numbers, use the `ls -n` command:

```
$ ls -n
total 108
-rw-r--r--   1 11001    10             0 Feb 22 14:51 brands
-rw-r--r--   1 11001    10          1320 Feb 22 14:51 dante
-rw-r--r--   1 11001    10           368 Feb 22 14:51 dante_1
```

Process for Determining Permissions

When a user attempts to access a file or directory, the Solaris Operating Environment compares the UID of the user to the UID of the file or directory being accessed.

If the UID values match, then the owner permissions are used to determine if access to the file or directory is granted.

If the UID numbers do not match, then the user's GID and the GID number of the file or directory are compared. If these values match, the group permissions apply.

If the GID numbers do not match, then the *other* category of permissions are used to determine file or directory access.

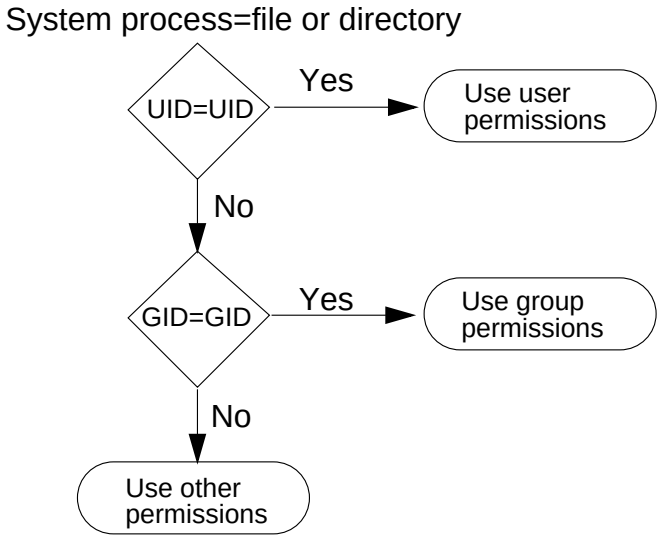


Figure 6-2 Process for Determining Permissions

Types of Permissions

File and directory access is protected by a standard set of default permissions, which are automatically assigned by the Solaris Operating Environment when a file or directory is created.

Permissions control who can do what to a file or directory and are represented by the characters `r` (read), `w` (write), `x` (execute), and `-` (denied).

When a user creates a new file or directory, by default, the Solaris Operating Environment automatically assigns the permissions on a file as `rw-rw-rw-` and on a directory as `rwxrwxrwx`.

Note – Execute permissions can be placed on files by the user with the `chmod` command, but these permissions are not assigned by default when a file is created.

The read/write/execute permissions are interpreted differently when assigned to an ordinary file than when assigned to a directory. See Table 6-1 describes the differences.

Table 6-1 Permissions and Corresponding Symbols

Permission	Permission Symbol	File	Directory
Read	`r`	The file can be displayed or copied	Contents can be listed with the `ls` command
Write	`w`	The file contents can be modified	If the user also has execute access, then files can be added or deleted
Execute	`x`	The file can be executed (shell scripts or executables only)	The user can `cd` to the directory. If the user also has read access, then the user can execute the `ls` command on the directory

Note – For a directory to be of general use, you must set the read and execute permissions.

The following are samples of different types of permissions set on files and directories.

- This file is read/write/execute for the file owner only. All other permissions for group and others are denied:

  ```
  -rwx------
  ```

- This directory is read/execute for the directory owner and the group only:

  ```
  dr-xr-x---
  ```

- This file is read/write/execute for file owner and read/execute for the group and all others:

  ```
  -rwxr-xr-x
  ```

Changing Permissions

You can modify the permissions set on newly created files or directories using the `chmod` command. Either the owner of the file or directory or superuser can use this command to change permissions.

The `chmod` command can modify permissions specified in either *symbolic* mode or *octal* mode.

- Symbolic mode uses combinations of letters and symbols to add, remove, or set permissions for each class of users.
- Octal mode uses numbers to represent each permission, often referred to as *absolute* mode.

Figure 6-3 on page 6-11 illustrates the relationship between permissions and files.

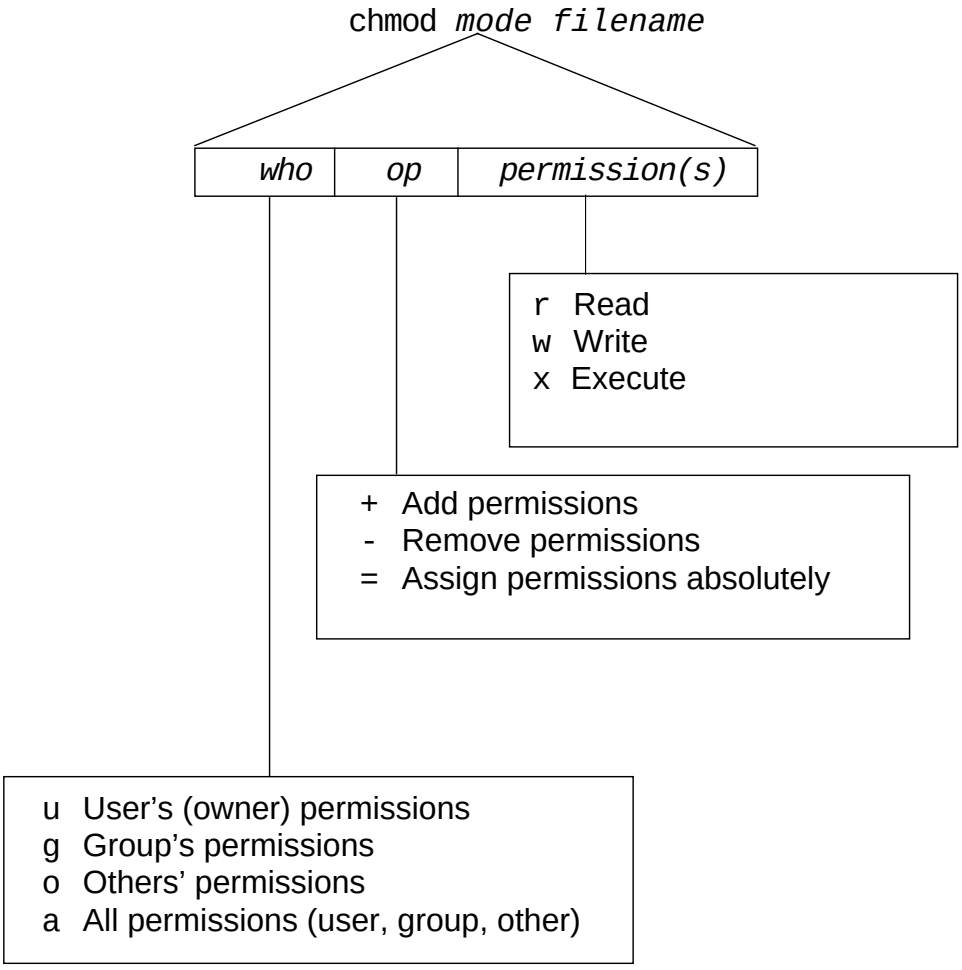


Figure 6-3 Symbolic Mode Command Format

Changing Permissions With Symbolic Mode

The following examples show how to modify permissions on files and directories using symbolic mode.

- Remove the read permission for other:

```
$ ls -l dante
-rw-r--r--   1 user1    staff       1320 Feb 22 14:51 dante
$ chmod o-r dante
$ ls -l dante
-rw-r-----   1 user1    staff       1320 Feb 22 14:51 dante
$
```

- Remove the read permission on group:

```
$ chmod g-r dante
$ ls -l dante
-rw-------   1 user1    staff       1320 Feb 22 14:51 dante
$
```

- Add an execute permission for the user (owner), and a read permission for the group and other:

```
$ chmod u+x,go+r dante
$ ls -l dante
-rwxr--r--   1 user1    staff       1320 Feb 22 14:51 dante
$
```

- Assign read and write permissions for user, group, and other:

```
$ chmod a=rw dante
$ ls -l dante
-rw-rw-rw-   1 user1    staff       1320 Feb 22 14:51 dante
$
```

Octal (Absolute) Mode

You specify octal mode using a combination of octal numbers. The numbers used include 0 to 7.

Command Format

`chmod` *`octal_mode filename`*

Each permission is represented by its own octal number.

Table 6-2 Permissions Assigned Octal Value

Octal Value	Permissions
4	Read
2	Write
1	Execute

Each octal number represents a permission set as shown in Table 6-3.

Table 6-3 Octal Digits for Permission Sets

Octal Value	Permission Sets
7	`r w x`
6	`r w -`
5	`r - x`
4	`r - -`
3	`- w x`
2	`- w -`
1	`- - x`
0	`- - -`

By combining octal numbers, a user can quickly modify the permissions for each class of users. The first octal number defines *owner permissions*, the second octal number defines *group permissions*, and the third octal number defines *other permissions*.

Table 6-4 Combined Values and Permissions

Octal Mode	Permissions
644	`rw-r--r--`
751	`rwxr-x--x`
775	`rwxrwxr-x`
777	`rwxrwxrwx`

When using octal mode with the `chmod` command, chmod automatically fills in any missing digits to the left with zeros.

Changing Permission With Octal Mode

The following examples show how to modify permissions on files and directories using octal (absolute) mode.

Note – Each example builds on the resulting permissions from the previous example.

- To give owner, group, and others read and execute access only:

```
$ ls -l dante
-rw-rw-rw-   1 user1    staff       1320 Feb 22 14:51 dante
$ chmod 555 dante
$ ls -l dante
-r-xr-xr-x   1 user1    staff       1320 Feb 22 14:51 dante
$
```

- To change user and group permissions to include write access:

```
$ chmod 775 dante
$ ls -l dante
-rwxrwxr-x   1 user1    staff       1320 Feb 22 14:51 dante
$
```

- To change group permission to read and execute only:

```
$ chmod 755 dante
$ ls -l dante
-rwxr-xr-x   1 user1    staff       1320 Feb 22 14:51 dante
$
```

Default Permissions

The following sections describe the different types of default permissions.

The umask Filter

The umask filter controls the default permissions assigned to newly created files and directories. The umask is a three-digit octal value that refers to read/write/execute permissions for owner, group, and other.

Displaying the umask

```
$ umask
022
$
```

In the Solaris Operating Environment, the default umask value is 022.

The umask operates as a filter to affect the initial permission values specified by the system during the creation of a file or directory.

The first digit determines the default permissions for the owner, the second digit determines the default permissions for the group, and the third digit determines the default permissions for other.

The initial permission value specified by the system for a file creation is 666 (rw-rw-rw-).

The initial permission value specified by the system for a directory creation is 777 (rwxrwxrwx).

The umask value is automatically filtered or subtracted from the initial permission value to determine the default permissions assigned to newly created files and directories.

Understanding the `umask` *Filter*

Another way to determine what the default permissions will be when creating new files is to take the initial value specified by the system; represented by symbolic mode:

```
rw-rw-rw-
```

which corresponds to read/write access for the user, group, and other, and represented in octal mode as:

```
42-42-42-
```

Use the default `umask` value of 022, which removes (or denies) write permission for group and other.

For example:

`rw-rw-rw-`	Initial value specified by the system for a new file.
`----w--w-`	Default `umask` filter value to be subtracted.
`rw-r--r--`	Default permissions assigned to newly created files.

When the access permissions to be denied are masked out from the initial value, the default permissions assigned to the new directories remain.

All newly created files are assigned read/write access for the user, and read access for group and other:

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

You can apply this same process when determining what the default permissions are when creating new directories.

In this case, take the initial value, specified by the system, represented by symbolic mode as:

```
rwxrwxrwx
```

which corresponds to read/write/execute access for the user, group, and others, and represented in octal mode as:

```
421421421
```

Use the default `umask` value of 022, which removes (or denies) write permission for group and other.

For example:

`rwxrwxrwx` Initial value specified by the system for a new directory.

`----w--w-` Default `umask` filter value to be subtracted.

`rwxr-xr-x` Default permissions set for newly created directories.

When the access permissions to be denied are masked out from the initial value, the default permissions assigned to the new directories remain.

All newly created directories are assigned read/write/execute access for the user, and read/execute access for group and other:

```
rwxr-xr-x
```

Changing the `umask` *Value*

Some user's require a more secure `umask` value of 027, which assigns the following access permissions to newly created files and directories.

- Files are given read/write permissions for the owner; read permission for the group; all permissions are denied for other:

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

- Directories are given read/write/execute permissions to the owner and read/execute permissions to the group; all access permissions are denied for other:

  ```
  rwxr-x---.
  ```

Changing umask

You can change the umask to a new value on the command line.

Changing the umask value on a file to 027 (rw-r-----) gives read/write permission to the owner of a new file, read permissions to group, and no permissions to others.

Changing the umask value on a directory to 027 (rwxr-x---) gives all permissions to the owner, read/execute permissions to group, and no permissions to others.

For example, to change the umask value to 027, and then verify the new value has been set, execute the following:

```
$ umask 027
$ umask
027
$
```

This new umask value affects only those files and directories that are created from this point forward. However, because the umask value was changed on the command line, if the user logs out of the system, the new value (027) is replaced by the old value (022) on subsequent logins.

To retain the new umask value, place it in one of the shell initialization files.

Note – Shell initialization files are covered in Module 11, "The Korn Shell."

Exercise: Changing File Permissions

Exercise objective – In this exercise, you practice reading permissions on files and changing permissions using symbolic or octal notation.

Tasks

Complete or answer the following:

1. Execute the following commands:
   ```
   $ mkdir ~/perm
   $ cd /etc
   $ cp group passwd motd vfstab shadow ~/perm
   $ cd
   $ cp -r /etc/skel perm
   ```

   When trying to copy `/etc/shadow`, an error message was displayed. Why?

   _____

   _____

2. Change the directory to `perm`, and complete the following table:

File or Directory	User Permissions	Group Permissions	Other Permissions	Octal Value
group	rw-			
passwd			r--	
vfstab	rw-			
skel				755

3. Create a new file and a new directory.

   ▼ What are the default permissions given to the new file?

   _____

   ▼ What are the default permissions given to the new directory?

   _____

4. In a directory with permissions of `drwxr-xr--`, who can perform the following actions with the files shown below? Put an X next to each allowed action.

   `-rw-r--r--`
   User: read___ modify___ delete___ execute___

   Group: read___ modify___ delete___ execute___

   Others: read___ modify___ delete___ execute___

   `-rwxrwxr-x`
   User: read___ modify___ delete___ execute___

   Group: read___ modify___ delete___ execute___

   Others: read___ modify___ delete___ execute___

5. Using symbolic mode, add write permission for the group to the `motd` file.

   _____

6. Using octal mode, change the permissions on the `motd` file to `-rwxrw----`.

   _____

7. Using octal mode, change the permissions on the `group` file to add write permission for others.

   _____

8. Why is execute not a default permission for a newly created file?

   _____

   _____

9. Create a new file called `memo` in your `practice` directory.

   _____

10. Remove the read permission for the owner from the `memo` file in the `practice` directory. Use either symbolic or octal mode.

    _____

    What happens when you try to use the `cat` command to view the `memo` file?

    _____

    What happens when you try to copy the `memo` file?

    _____

    _____

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercises.

- Experiences
- Interpretations
- Conclusions
- Applications

Task Solutions

Complete or answer the following:

1. Execute the following commands:

```
$ mkdir ~/perm
$ cd /etc
$ cp group passwd motd vfstab dumpdates shadow ~/perm
$ cd
$ cp -r /etc/skel perm
```

   When trying to copy `/etc/shadow`, an error message was displayed. Why?

   Because the only user who has any permissions on this file is the owner root.

2. Change the directory to `perm`, and complete the following table:

```
$ cd perm
$ ls -l
```

File or Directory	User Permissions	Group Permissions	Other Permissions	Octal Value
group	rw-	r--	r--	644
passwd	rw-	r--	r--	644
vfstab	rw-	r--	r--	644
skel	rwx	r-x	r-x	755

3. Create a new file and a new directory.

   - What are the default permissions given to the new file?

     `rw-r--r--`

   - What are the default permissions given to the new directory?

     `rwxr-xr-x`

4. In a directory with permissions of `drwxr-xr--`, who can perform the following actions with the files shown below? Put an X next to each allowed action.

   `-rw-r--r--`

User:	read **X**	modify **X**	delete **X**	execute___
Group:	read **X**	modify___	delete___	execute___
Others:	read **X**	modify___	delete___	execute___

   `-rwxrwxr-x`

User:	read **X**	modify **X**	delete **X**	execute **X**
Group:	read **X**	modify **X**	delete **X**	execute **X**
Others:	read **X**	modify___	delete___	execute **X**

5. Using symbolic mode, add write permission for the group to the `motd` file.

   ```
   $ chmod g+w motd
   ```

6. Using octal mode, change the permissions on the `motd` file to `-rwxrw----`.

   ```
   $ chmod 760 motd
   ```

7. Using octal mode, change the permissions on the `group` file to add write permission for others.

   ```
   $ chmod 646 group
   ```

8. Why is execute not a default permission for a newly created file?

   Most files are not binary or executable scripts.

   Having execute as a default permission for a file would cause the system to see all new files as executables.

9. Create a new file called `memo` in your `practice` directory.

   ```
   $ touch ~/practice/memo
   ```

10. Remove the read permission for the owner from the `memo` file in the `practice` directory. Use either symbolic or octal mode.

```
$ chmod u-r ~/practice/memo
```
or
```
$ chmod 244 ~/practice/memo
```

What happens when you try to use the `cat` command to view the `memo` file?

You cannot use the `cat` *command because the read permission has been removed for the user. Even though you are part of the group, the permissions are looked at in the order they appear.*

What happens when you try to copy the `memo` file?

You cannot copy the file because the user has no read permission.

Check Your Progress

Before continuing on to the next module, check that you are able to accomplish or answer the following:

- ❑ Display file and directory permissions
- ❑ Define the standard permission types (read/write/execute)
- ❑ Use the `chmod` command to change permissions with symbolic mode or octal mode values
- ❑ Determine the default permissions assigned to newly created files and directories with `umask`

Visual (`vi`) Editor 7

Objectives

Upon completion of this module, you should be able to:

- Define the three modes of operation used by the `vi` editor
- Start the `vi` editor
- Position and move the cursor in `vi`
- Switch between `vi` modes
- Create and delete text
- Copy or move text
- Set `vi` options
- Perform search and replace functions within `vi`
- Exit the `vi` editor

Additional Resources

Additional resources – The following reference provides additional details on the topics discussed in this module:

- *Solaris™ 8 Reference Manual Collection: User Commands,* Part Number 806-0624-10

Introducing `vi`

The visual display (`vi`) editor is an interactive editor used to create or modify text files.

All text editing with the `vi` editor takes place in a buffer. Changes can either be written to the disk or be discarded.

For those who intend on becoming system administrators, it is important to know how to use `vi`. You need to know how to use the `vi` editor in case the windowing system is not available. `vi` is also the only text editor that can be used to edit certain system files without changing the files' permissions.

Figure 7-1 shows the initial vi display.

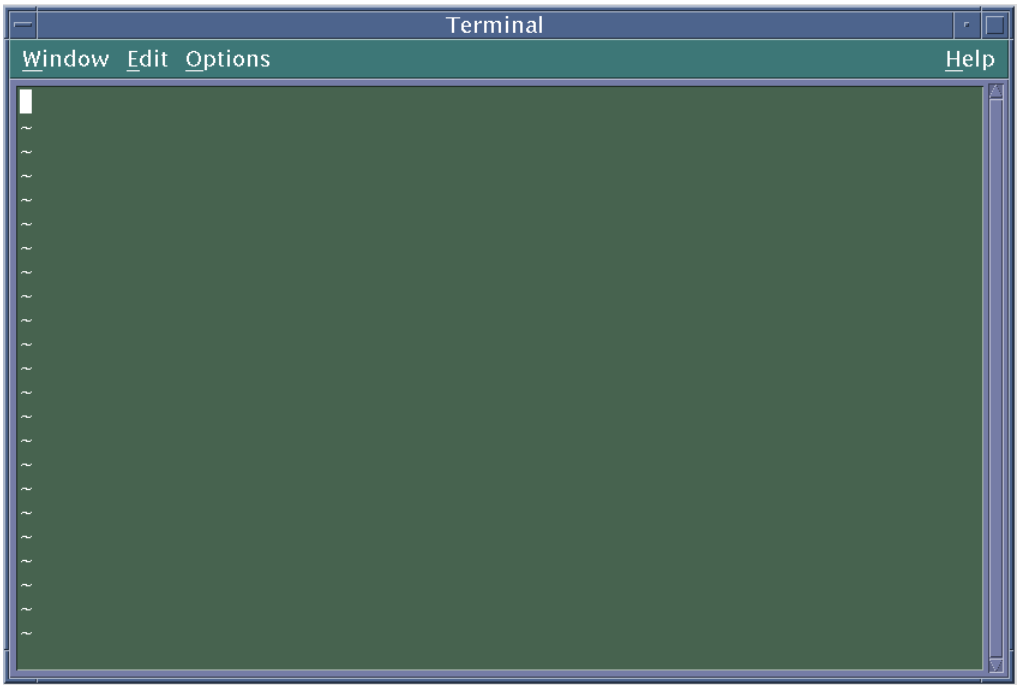


Figure 7-1 Initial `vi` display

vi Modes

The vi editor is a command-line editor that has three basic modes of operation:

- Command mode
- Edit mode
- Last line mode

Command Mode

This is the default mode for vi. In this mode, you can enter commands to delete, change, copy, and move text; position the cursor; search for text strings; or exit vi.

Edit Mode

In this mode, you can enter text into a file. To instruct vi to enter edit mode, enter one of the following three commands:

- i (insert)
- o (open)
- a (append)

Last Line Mode

While in command mode, you can use advanced editing commands by typing a colon (:), which places you at the bottom line of the screen. This is called last line mode. However, all commands are initiated from command mode.

Switching Modes

By typing an `i`, `o`, or `a` command, `vi` leaves the default command mode and enters edit mode.

In edit mode, text is not interpreted as commands by `vi`. Now, everything you type is entered into the file as text.

When you have finished entering text in the file, you can return `vi` to command mode by pressing the Escape key. When you are back in command mode, you can then save the file and quit `vi`.

For example:

1. Type `vi` *`filename`* to create a file.
2. Type the `i` command to insert text.
3. Press the Escape key to return to command mode.
4. Type `:wq` to write and save the file and exit `vi`.

Invoking `vi`

To create a new file, invoke `vi` with a new file name by typing commands to create, edit, or view a file.

Command Format

```
vi option(s) filename
view filename
```

Input Commands

To insert or append text, use the commands in Table 7-1.

Table 7-1 Append and Insert Commands for `vi`

Command	Meaning
`a`	Appends text after the cursor
`A`	Appends text at the end of the line
`i`	Inserts text before the cursor
`I`	Inserts text at the beginning of the line
`o`	Opens a new line below the cursor
`O`	Opens a new line above the cursor

Note – The `vi` editor is case-sensitive, so use the specified case when using these cursor input commands.

Positioning Commands

Table 7-2 shows the key sequences that control cursor movement in the `vi` editor.

Table 7-2 Key Sequences

Command	Meaning
`h`, ←, or Back Space	Moves left one character
`j` or ↓	Moves down one line
`k` or ↑	Moves up one line
`l`, →, or Space bar	Moves right (forward) one character
`w`	Moves forward one word
`b`	Moves back one word
`e`	Moves to the end of the current word
`$`	Moves to the end of the line
`0` (zero) or ^	Moves to the beginning of the line
Return	Moves down to the beginning of the next line
Control-F	Pages forward one screen
Control-D	Scrolls down one-half screen
Control-B	Pages back one screen
Control-U	Scrolls up one-half screen
Control-L	Refreshes the screen

Editing Commands

The following sections describe the editing commands in the `vi` editor.

Deleting Text

To delete text, use the options in Table 7-3.

Table 7-3 Text Deletion Commands for `vi`

Command	Meaning
`x`	Deletes a character at the cursor
`dw`	Deletes a word (or part of the word to the right of the cursor)
`dd`	Deletes the line containing the cursor
`D`	Deletes the line to the right of the cursor (from cursor position to the end of the line)
`:5,10d`	Deletes Lines 5 through 10

Undoing, Repeating, and Changing Text Commands

To change text, undo a change, or repeat an edit function, use the commands in Table 7-4. Many of these commands change `vi` to edit mode. To return to command mode, press the Escape key.

Table 7-4 Edit Commands for `vi`

Command	Meaning
`cw`	Changes a word (or part of a word) at the cursor location to the end of the word
`R`	Overwrites or replaces characters on the line
`C`	Changes from cursor to end of the line
`s`	Substitutes string for characters
`r`	Replaces the character at the cursor with one other character

Table 7-4 Edit Commands for `vi` (Continued)

Command	Meaning
`J`	Joins the current line and the line below
`xp`	Transposes the character at the cursor and the character to the right
`~`	Changes the case of the letter (uppercase or lowercase) at the cursor
`u`	Undoes the previous command
`U`	Undoes all changes to the current line
`u`	Undoes the previous last-line command
`:r` *`filename`*	Inserts (reads) the file at the line after the cursor

To search and replace text, use the following options in Table 7-5.

Table 7-5 Search and Replace Commands

Command	Meaning
/string	Searches forward for the *string*
?string	Searches backward for the *string*
`n`	Finds the next occurrence of the string
`N`	Finds the previous occurrence of the string
`:%s/`*`old`*`/`*`new`*`/g`	Searches and replaces globally

Copying and Pasting Text

The copy commands write the copied text into a temporary buffer. The paste commands read the text from the temporary buffer and write the text into the current document at the specified location.

To copy and paste text, use the options in Table 7-6.

Table 7-6 Copy and Paste Commands

Command	Meaning
`yy` (lowercase)	Yanks a copy of a line
`p` (lowercase)	Puts yanked or deleted text after the current position
`P` (uppercase)	Puts yanked or deleted text before the current position
`:1,3 co 5`	Copies Lines 1 through 3 and puts them after Line 5
`:4,6 m 8`	Moves Lines 4 through 6 to Line 8 (Line 6 becomes Line 8; Line 5 becomes Line 7, and Line 4 becomes Line 6)

Note – Both delete and yank write to a buffer. When yanking, deleting, and pasting, the `put` commands insert the text differently depending on whether you are pasting words or lines.

Saving and Quitting Files

To save and quit a file, use the options in Table 7-7.

Table 7-7 Save and Quit Commands

Command	Meaning
`:w`	Saves the changes (write buffer)
`:w` *`new_filename`*	Writes the contents of the buffer to *`new_filename`*
`:wq`	Saves the changes and quits `vi`
`:x`	Saves the changes and quits `vi`
`ZZ`	Saves the changes and quits `vi`
`:q!`	Quits without saving changes
`:wq!`	Saves the changes and quits `vi` (`!` overrides read-only permissions for the owner of the file only)

Customizing Your `vi` *Session*

The `vi` editor includes options for customizing edit sessions, such as:

- Displaying line numbers
- Displaying invisible characters, such as tab and end-of-line characters

Use the `set` command in command mode to control these options, as shown in Table 7-8.

Table 7-8 Edit Session Customization Commands

Command	Meaning
`:set nu`	Shows line numbers
`:set nonu`	Hides line numbers
`:set ic`	Instructs searches to ignore case
`:set noic`	Instructs searches to be case-sensitive
`:set list`	Displays invisible characters, such as tab and end-of-line
`:set nolist`	Turns off the display of invisible characters
`:set showmode`	Displays current mode of operation
`:set noshowmode`	Turns off mode display
`:set`	Displays all `vi` variables set
`:set all`	Displays all possible `vi` variables and their current settings

You can also place these options in a file you create in your `home` directory called `.exrc`. The set options are placed in this file, without the preceding colon, one command to a line. After the `.exrc` file has been created, it is read by the system each time you open a `vi` session.

To find a particular line, use the options in Table 7-9.

Table 7-9 Positioning Commands

Command	Meaning
`G`	Goes to the last line of the file
`1G`	Goes to the first line of the file
`:21`	Goes to Line 21
`21G`	Goes to Line 21

To clear the screen or insert files, use the options in Table 7-10.

Table 7-10 Refreshing Commands

Command	Meaning
Control-L	Refreshes the screen

Exercise: Using the `vi` *Editor*

Exercise objective – In this exercise, you practice creating and modifying the files using the `vi` editor.

Tasks

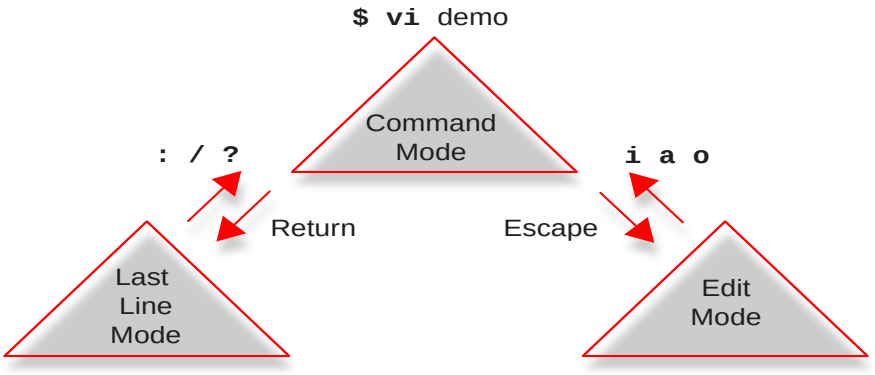
Complete the following steps:

1. In your home directory, there should be a file called `tutor.vi`. Make sure you are currently in your home directory, and then open this file with the command:

   ```
   $ vi tutor.vi
   ```

   This opens a `vi` tutorial file.

2. Complete the lessons outlined in this tutorial.



Search Functions

/exp	go forward to exp
?exp	go backward to exp

Move and Insert Text

:3,8d	delete line 3-8
:4,9m 12	move lines 4-9 to 12
:2,5t 13	copy lines 2-5 to 13
:5,9w file	write lines 5-9 to file

Save Files and Exit

:w	write to disk
:w newfile	write to newfile
:w! file	write absolutely
:wq	write and quit
:q	quit editor
:q!	quit and discard
:e!	re-edit current file, discard buffer

Control Edit Session

:set nu	display line number
:set nonu	turn off an option
:set all	show all settings
:set list	display invisible characters
:set wm=5	wrap lines 5 spaces from right margin

Screen/Line Movement

j	move cursor down
k	move cursor up
h	move cursor left
l	move cursor right
0	go to line start (zero)
$	go to line end
G	go to last file line

Word Movement

W	go forward 1 word
B	go backward 1 word

Search Functions

n	repeat previous search
N	reverse previous search

Delete Text

x	delete 1 character
dw	delete 1 word
dd	delete 1 line
D	delete to end of line
d0	delete to start of line
dG	delete to end of file

Cancel Edit Function

u	undo last change
.	do last change again

Copy and Insert Text

Y	yank a copy
5Y	yank a copy of 5 lines
p	put below cursor
P	put above cursor

Add/Append Text

a	append after cursor
A	append at line end
i	insert before cursor
5i	insert text 5 times
I	insert at beginning of line

Add New Lines

o	open a line below cursor
O	open a line above cursor

Search Functions

n	repeat previous search
N	reverse previous search

Change Text

cw	change a word
3cw	change 3 words
C	change line
r	replace one character
R	replace/type over a line

Figure 7-2 Quick Reference Chart for Using `vi`

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercises.

- Experiences
- Interpretations
- Conclusions
- Applications

Check Your Progress

Before continuing on to the next module, check that you are able to accomplish or answer the following:

- ❑ Define the three modes of operation used by the `vi` editor
- ❑ Start the `vi` editor
- ❑ Position and move the cursor in `vi`
- ❑ Switch between `vi` modes
- ❑ Create and delete text
- ❑ Copy or move text
- ❑ Set `vi` options
- ❑ Perform search and replace functions within `vi`
- ❑ Exit the `vi` editor

Archiving User Data 8

Objectives

Upon completion of this module, you should be able to:

- Determine which commands are suitable for storing, viewing, or retrieving different types of files
- Demonstrate how to reduce the size of files and directories and store them to tape using the `compress` and `tar` commands
- Describe the steps for uncompressing or viewing a compressed file with the `uncompress` and `zcat` commands
- Use the `gzip` and `gunzip` commands to compress and uncompress files
- Use the `zip` command to package and compress multiple files and use `unzip` to uncompress a zipped archive file
- Compress and copy multiple files to a single archive file in one step using the `jar` command
- Copy and extract files from an archive file or tape device with the `cpio` command
- Understand how to use volume management to access CD-ROMs and diskettes
- Use the `eject` command to remove CD-ROMs and diskettes from device drives

Overview of Archive Commands

To safeguard your files and directories, archive copies of them onto some form of removable media, such as a cassette tape. You need the archived tapes to retrieve lost, deleted, or damaged files.

The commands available to easily store, locate, and retrieve files on a tape device or an archive file include:

- `tar` – Creates and extracts files from a tape device or file archive
- `compress` and `uncompress` – Compresses and uncompresses a file
- `zcat` – Uncompresses a compressed file and sends the output to the screen without changing the compressed file
- `gzip` and `gunzip` – Compresses and uncompresses a file
- `gzcat` – Uncompresses a gzipped file and sends the output to the screen without changing the gzipped file
- `zip` and `unzip` – Packages and compresses files and uncompresses files
- `jar` – Packages and compresses multiple files to a single archive file
- `cpio` – Copies and extracts files from a file archive or tape device

---

Note – When archiving, use relative path names.

Archiving Files Using the `tar` *Command*

The `tar` command archives and extracts files to and from a single file called a *tar file*. A tar file defaults to a magnetic tape device, but it can be any file.

Command Format

```
tar function(s) archivefile filename(s)
```

Functions

Table 8-1 lists of the various `tar` functions.

Table 8-1 Functions for the `tar` Command

Function	Definition
`c`	Creates a new `tar` file. Writing starts at the beginning of the file rather than the end.
`t`	Lists the table of contents of the `tar` file.
`x`	Extracts the specified files from the `tar` file.
`f`	Specifies the *`archive_file`* or tape device. The default tape device is `/dev/rmt/0`. If *`archive_file`* is "`-`", then the `tar` command reads from standard input or writes to standard output, whichever is appropriate.
`v`	Executes in verbose mode.
`r`	Replace. The named files are written at the end of the tape archive.
`u`	Update. If the named files do not exist within the archive or they have been updated since being written to the archive, then the files are written to the end of the archive.
`e`	Exit immediately if any expected errors occur.
`h`	Follow symbolic links as if they were normal files or directories.

Table 8-1 Functions for the `tar` Command (Continued)

Function	Definition
`m`	When extracting files from an archive, the file modification timestamp is the time of extraction.
`o`	When extracting from an archive, the file owner is set to the user executing `tar`.
`w`	The prompt for confirmation of every file to be extracted from or added to the archive.

Creating, Viewing, and Retrieving a Directory From a Tape

To create a tape archive of your home directory using the default tape device, you can execute the following commands (in this example, `user1` is creating a tape archive of the `user1` home directory):

```
$ cd
$ tar cvf /dev/rmt/0 .
a ./ 0 tape blocks
a ./dante 106 tape blocks
a ./dante_1 1 tape blocks
a ./logfile 5 tape blocks
a ./file2 1 tape blocks
<output omitted>
```

To view the contents of the `user1 home` directory copied to tape, execute the following command:

```
$ tar tf /dev/rmt/0
./
./dante
./dante_1
./logfile
./file2
./file3
./file4
./fruit
./fruit2
./tutor.vi
./dir1/
./dir1/coffees/
./dir1/coffees/beans
./dir1/coffees/nuts
```

```
./dir1/coffees/beans.backup
<output omitted>
```

If your `home` directory was deleted, you could retrieve the entire directory by extracting its contents from the archive tape by executing the following commands (in this example `user1` is retrieving the `user1` directory from the archive tape):

```
$ cd
$ tar xvf /dev/rmt/0 .
x user1, 0 bytes, 0 tape blocks
x ./dante, 54120 bytes, 106 tape blocks
x ./dante_1, 368 bytes, 1 tape blocks
x ./logfile, 2483 bytes, 5 tape blocks
x ./file2, 105 bytes, 1 tape blocks
x ./file3, 218 bytes, 1 tape blocks
x ./file4, 137 bytes, 1 tape blocks
x ./fruit, 56 bytes, 1 tape blocks
x ./fruit2, 57 bytes, 1 tape blocks
x ./tutor.vi, 28738 bytes, 57 tape blocks
x ./dir1, 0 bytes, 0 tape blocks
x ./dir1/coffees, 0 bytes, 0 tape blocks
<output omitted>
```

Note – Typically, home directories are scheduled for tape archive on a nightly basis by the system administrator. However, you can also archive the contents of your home directory on a regular basis. If a file is accidentally deleted, you can restore it quickly.

The following examples describe the steps for creating, viewing, and retrieving a directory from an alternate tape drive attached to the system, rather than the default tape drive.

```
$ cd user1
$ tar cvf /dev/rmt/1 .

$ tar tvf /dev/rmt/1

$ cd user1
$ tar xvf /dev/rmt/1 .
```

Creating, Viewing, and Retrieving Files From an Archive File

You can also use the `tar` command to create single archive files on disk to share with other users or for attaching to mail messages.

```
$ cd
$ tar cvf files.tar file1 file2 file3
a file1 2K
a file2 1K
a file3 1K
```

In this example, `file1`, `file2`, and `file3` are stored in an archive file called `files.tar`.

To view the contents of this archive file, execute the following command:

```
$ tar tf files.tar
file1
file2
file3
```

The following example shows how to extract files from this archive file for placement back into the current directory.

```
$ tar xvf files.tar
tar: blocksize = 11
x file1, 1696 bytes, 4 tape blocks
x file2, 156 bytes, 1 tape blocks
x file3, 218 bytes, 1 tape blocks
```

Compressing Files Using the `compress` *Command*

Use the `compress` command to reduce the size of a file. This is particularly useful when working with large files, which can consume disk space and take longer than smaller files to transfer from one system to another over a network.

Note – The amount of compression depends on the type of file being compressed, but, typically, a text file is reduced by 60 to 80 percent.

When a file is compressed, the named file is replaced by a new file with a `.Z` extension. The ownership and modification time of the original file remains the same, even though the file's contents are completely changed.

Command Format

```
compress [ -v ] filename ...
```

Compressing a File

This example compresses a file called `files.tar`:

```
$ compress -v files.tar
files.tar: Compression: 70.20% -- replaced with files.tar.Z
```

The newly compressed file, which replaces `files.tar`, is now called `files.tar.Z`.

This naming convention (`.Z`) indicates that the file is compressed and should not be viewed or printed without first uncompressing it.

Caution – Compressing files that have already been compressed makes file sizes larger—not smaller.

Uncompressing Files Using the uncompress *Command*

The uncompress command restores a compressed file back to its original state.

Command Format

```
uncompress [ -vc ] filename ...
```

Uncompressing a File

The following example uncompresses the files.tar.Z file and replaces it with the original file named files.tar.

```
$ uncompress files.tar.Z
```

Viewing the Contents of a Compressed File

Using the uncompress command with the -c option sends the contents of a compressed file to the screen or to another program if used with a pipe (|), without changing the original compressed (.Z) file.

```
$ uncompress -c file.tar.Z | tar tvf -
```

Note – The "-" at the end of the command line indicates that tar reads the data from the piped output of the uncompress command rather than a tar file or tape.

Viewing Files Using the `zcat` *Command*

The `zcat` command also allows you to view a file that has been compressed with the `compress` command.

The `zcat` command interprets the compressed data and displays the contents of the file as though it were not compressed. After running `zcat`, the contents of the compressed file are unchanged; it is still stored on the disk in compressed form.

Command Format

`zcat` *`filename ...`*

The command `zcat` *`filename`* is functionally identical to the `uncompress -c` *`filename`* command.

Viewing the Contents of a Compressed File

To extract the contents of the compressed tar file, without uncompressing the file first, execute the following:

```
$ zcat file.tar.Z | tar xvf -
```

Note – The "`-`" at the end of the command line indicates that `tar` reads the data from the piped output of the `uncompress` command rather than a tar file or tape.

Compressing a File With the gzip *Command*

When gzip compresses a file, it adds the extension .gz to the file name.

Command Format

```
gzip filename filename filename
```

For example:

```
$ gzip file1 file2 file3 file4
$ ls
file1.gz file2.gz file3.gz file4.gz
```

Note – The gzip command performs the same function as the compress command but, generally, produces smaller files.

Restoring a gzip *File With the* gunzip *Command*

To restore a file that has been compressed with gzip, use the gunzip command.

```
$ gunzip file.gz
```

Note – Although the gzip and gunzip executables are included in the Solaris 8 Operating Environment, they were not included in the Solaris 7 Operating Environment and earlier versions. However, you can download these executables from the gzip web home page at http://www.gzip.org/.

Viewing Files Using the `gzcat` *Command*

Use the `gzcat` command to view files that have been compressed with either the `gzip` or the `compress` command.

The `gzcat` command interprets the compressed data and displays the contents of the file as though it were not compressed. After running `gzcat`, the contents of the compressed file are unchanged; it is still stored on the disk in compressed form.

Command Format

`gzcat` *`filename ...`*

Viewing the Contents of a Compressed File

To view the contents of the compressed tar file, without uncompressing the file first, execute the following:

```
$ gzcat file.gz
```

Compressing Multiple Files With the `zip` Command

The `zip` command is similar to the `jar` command in that it compresses multiple files into a single archive.

When `zip` compresses a file, it adds the extension `.zip` to the file name if it is not given a file name with an extension.

Note – You can type `zip` or `unzip` on the command line to see a list of options used with each command.

Command Format

```
zip target_filename source_filename(s)
```

For example:

```
$ zip file.zip file1 file2 file3
$ ls
file.zip
file1
file2
file3
```

where `file.zip` is the packaged and compressed zip file.

Restoring a `zip` File With the `unzip` Command

You can unpack the contents of a `zip` file using the `unzip` command:

```
$ unzip file.zip
```

Note – Both `jar` and `zip` create compatible files. This means that `unzip` will unpack a `.jar` file, and `jar` will unpack a `.zip` file.

Compressing Files Using the `jar` *Command*

The `jar` command is used like the `tar` command, but it compresses the named files in the same step. This command copies and compresses multiple files into a single archive file. Zip programs can read `jar` files.

Note – This was originally created for Java™ programmers working with Java technology to allow downloading multiple files at one time rather than issuing a download for each separate file. The `jar` command is standard with the Solaris 8 Operating Environment, and it is available on any system that has a Java Virtual Machine™ (JVM™) installed.

Command Format

The syntax for the `jar` command is almost identical to the syntax for the `tar` command.

```
jar options output_file filename(s)/directory(s)
```

Options

Table 8-2 lists the options used with the `jar` command.

Table 8-2 Options for the `jar` Command

Option	Definition
`c`	Creates a new `jar` file.
`t`	Lists the table of contents of a `jar` file.
`x`	Extracts the specified files from the `jar` file.
`f`	Specifies the `jar` file (for example, `/tmp/file.jar`) or tape drive (for example, `/dev/rmt/#`). The `jar` command sends the data to the screen if the "`f`" option is not used.
v	Executes in verbose mode.

Adding All the Files in a Directory to an Archive

The following example copies and compresses multiple files into the single archive file called `bundle.jar`.

```
$ ls
Reports     dir1        feathers    file.3      file4       new.dir
brands      dir2        feathers_6  file1       fruit       newdir
dante       dir3        file.1      file2       fruit2      practice2
dante_1     dir4        file.2      file3       monthly     tutor.vi
$
$ jar cvf /tmp/bundle.jar *
adding: Reports/ (in=0) (out=0) (stored 0%)
adding: Reports/Weekly/ (in=0) (out=0) (stored 0%)
adding: Reports/Weekly/dir1/ (in=0) (out=0) (stored 0%)
adding: Reports/Weekly/dir2/ (in=0) (out=0) (stored 0%)
adding: Reports/Weekly/dir3/ (in=0) (out=0) (stored 0%)
adding: Reports/Weekly/games/ (in=0) (out=0) (stored 0%)
adding: brands (in=0) (out=0) (stored 0%)
adding: dante (in=1320) (out=744) (deflated 43%)
adding: dante_1 (in=368) (out=242) (deflated 34%)
adding: dir1/ (in=0) (out=0) (stored 0%)
adding: dir1/coffees/ (in=0) (out=0) (stored 0%)
adding: dir1/coffees/beans (in=12288) (out=3161) (deflated 74%)
adding: dir1/fruit/ (in=0) (out=0) (stored 0%)
adding: dir1/trees/ (in=0) (out=0) (stored 0%)
adding: dir1/feathers (in=218) (out=162) (deflated 25%)
adding: dir1/feathers_6 (in=218) (out=162) (deflated 25%)
adding: dir1/constellation/ (in=0) (out=0) (stored 0%)
adding: dir1/constellation/mars (in=68) (out=61) (deflated 10%)
adding: dir1/constellation/pluto (in=42) (out=44) (deflated -4%)
adding: dir2/ (in=0) (out=0) (stored 0%)
adding: dir2/recipes/ (in=0) (out=0) (stored 0%)
adding: dir2/beans/ (in=0) (out=0) (stored 0%)
adding: dir2/notes (in=0) (out=0) (stored 0%)
adding: dir3/ (in=0) (out=0) (stored 0%)
adding: dir3/planets/ (in=0) (out=0) (stored 0%)
<output omitted>
```

Note – The `jar` command does not back up symbolic links as links. If a link is archived, it is stored as a normal file with the contents being what is pointed to by the symbolic link.

Using the `cpio` *Command*

The `cpio` (copy in/copy out) command archives or extracts a file to a tape or a single archive file.

Some advantages of the `cpio` command are:

- It packs data onto the tape more efficiently than the `tar` command
- It skips over any bad spots in a tape when restoring files
- It has options that change the output format to increase portability among different system types
- It creates multiple tape volumes
- It archives files without changing the files' access time

Command Format

```
cpio options filename(s)
```

Options

Table 8-3 shows the options available for use with the `cpio` command.

Table 8-3 Options for the `cpio` Command

Options	Definition
`o`	Creates a file archive. Copies a list of files or path names to the tape device or file (copy out).
`i`	Extracts the file archive from the tape device or file (copy in).
`p`	Reads from the tape device or file to get path names.
`c`	Reads or writes header information in ASCII character form for portability.
`t`	Lists the table of contents for the files on the tape in the tape drive specified.

Table 8-3 Options for the `cpio` Command (Continued)

`v`	Prints a list of file names in a format similar to the `ls -l` command.
`a`	Resets the access times of files after they have been copied.
`M` *`message`*	Defines a message to inform the user when the end of the media is reached and what action is needed.
`I` *`filename`*	Reads the contents of a file as an input archive.
`O` *`filename`*	Directs the output of the `cpio` command to a file.

Note – You must specify one of the following options: `o`, `i`, or `p` on the `cpio` command line.

Creating File Archives

The following example demonstrates how to use the `find` command with `cpio` to create an archive of the current directory contents and copy it to a file called `dir.cpio`.

```
$ find . | cpio -ocv -O dir.cpio
```

You can also use the `find` command with the `cpio` command to create an archive of only those files that have been changed within the last week, and copy it to a file called `modify.cpio`.

```
$ find . -mtime -7 | cpio -ocv -O modify.cpio
```

The following example shows how to view the list of file names in the archive files created previously.

```
$ cpio -ivt -I dir.cpio

$ cpio -ivt -I modify.cpio
```

Copying All Files in a Directory to a Tape

This example describes how to use the `cpio` command to copy files and directories to tape.

```
$ cd /export/home/user1
$ ls | cpio -oc -O /dev/rmt/0
16 blocks
$
```

Listing the Files on a Tape

The following example shows how to list the table of contents of a tape.

```
$ cpio -icvt -I /dev/rmt/0
-rwxr-xr-x  1  user1  staff     1320  Jun  1 08:45 2000, dante
-rwxr-xr-x  1  user1  staff      368  Jun 1 08:45 2000, dante_1
drwxr-xr-x  5  user1  staff        0  Jun  1 08:45 2000, dir1
drwxr-xr-x  4  user1  staff        0  Jun  1 08:45 2000, dir2
drwxr-xr-x  3  user1  staff        0  Jun  1 08:45 2000, dir3
drwxr-xr-x  3  user1  staff        0  Jun  1 08:45 2000, dir4
-rwxr-xr-x  1  user1  staff        0  Jun  1 08:45 2000, file.1
-rwxr-xr-x  1  user1  staff        0  Jun  1 08:45 2000, file.2
-rwxr-xr-x  1  user1  staff        0  Jun  1 08:45 2000, file.3
-rwxr-xr-x  1  user1  staff     1696  Jun  1 08:45 2000, file1
-rwxr-xr-x  1  user1  staff      105  Jun  1 08:45 2000, file2
-rwxr-xr-x  1  user1  staff       21  Jun  1 08:45 2000, file3
-rwxr-xr-x  1  user1  staff      137  Jun  1 08:45 2000, file4
-rwxr-xr-x  1  user1  staff       56  Jun  1 08:45 2000, fruit
-rwxr-xr-x  1  user1  staff       57  Jun  1 08:45 2000, fruit2
drwxr-xr-x  2  user1  staff        0  Jun  1 08:45 2000, practice
-rwxr-xr-x  1  user1  staff    28738  Jun  1 08:45 2000, tutor.vi
```

The table of contents displays eight fields of information for each file contained on the tape. The first field shows the permissions in octal mode; the second field shows the owner of the file; the third field displays the number of characters (bytes) in the file; the fourth, fifth, sixth, and seventh fields show the month, date, time, and year the file was last modified, and the last field shows the name of the file.

Retrieving All Files From a Tape

To retrieve all files from a tape, you must first change to the directory in which the files are to be placed, and then you can execute the following command:

```
$ cpio -icv -I /dev/rmt/0
```

The Volume Management Feature

To store and retrieve files on diskettes and CD-ROMs, the Solaris Operating Environment provides the volume management feature.

Volume management provides ordinary users with a standard method for handling data on diskettes and CD-ROMs. This feature gives you access to these types of devices automatically.

Note – If volume management is not running on a system, only the superuser can have access to diskettes and CD-ROMs.

The volume management service is invoked by a system daemon named `vold`. The superuser can enable and disable this feature on any system by starting or stopping the `vold` process.

By default, volume management is always running on the system to automatically manage CD-ROMs, diskettes, and removable small computer system interface (SCSI) disks, such as Zip and Jaz disks, for ordinary users.

Detecting Removable Media Devices

When you insert a CD-ROM or diskette into the appropriate drive on the system, the volume manager must detect its presence to provide access.

Volume management provides automatic detection of CD-ROMs. However, the volume manager does not detect the presence of a new diskette. It must be informed every time you insert a diskette in the drive.

Note – Automatic detection of diskettes would cause excessive reads, which would quickly wear out the diskette drive.

To inform the volume manager that you have inserted a diskette, invoke the `volcheck` command.

Checking for Media With the `volcheck` *Command*

The `volcheck` command checks for media in a drive; by default, it checks for all diskettes. `volcheck` tells volume management to look at each *`device_pathname`* in sequence and determine if new media has been inserted in the diskette drive.

Command Format

`volcheck [ -v ]` *`device_pathname`*

Samples of Using `volcheck`

For example, to ask volume management to examine the diskette drive for newly inserted media, execute the following:

```
$ volcheck -v /dev/diskette
```

The `volcheck` command reports back with either one of the following messages.

- If a diskette has been inserted in the diskette drive, `volcheck` displays the message:

  ```
  /dev/diskette has media
  ```

- If the diskette drive does not contain a diskette, `volcheck` displays the message:

  ```
  /dev/diskette has no media
  ```

Using the `volcheck` command without a *`device_pathname`* displays one of the following messages:

- `media was found`
- `no media was found`

Accessing Removable Media Devices

The following section describes how to access removable media devices.

CD-ROMs and Volume Management

When the volume manager detects the presence of a CD-ROM, it automatically places the CD-ROM under a standard directory in the directory tree, called `/cdrom`.

You have immediate access to the files on the CD-ROM device by moving to the `/cdrom` directory using the `cd` command.

To access files from the local CD-ROM drive:

1. Insert the CD-ROM (label side up) into the drive.

   The CD-ROM is automatically placed under the `/cdrom` directory.

Note – If the CDE File Manager is running, a new File Manager window displays the contents of the CD-ROM. You can access data from this window and the command line interchangeably.

---

2. In a terminal window, type **`cd /cdrom/cdrom0`** and press the Return key.
3. Type **`ls`** and press the Return key.

   The list of files in the `/cdrom/cdrom0` directory is displayed.

Diskettes and Volume Management

Volume management does not automatically check for the presence of a diskette in the drive. You must use `volcheck` to invoke the volume manager to check the drive.

If a diskette is present, the volume manager places this device under a standard directory in the directory tree called `/floppy`. You can then access files on the diskette by moving to this directory using the `cd` command.

To access files on a diskette in the diskette drive:

1. Insert a formatted diskette (label side up) in the diskette drive.
2. Type **`volcheck`** and press the Return key.

   Volume management places the diskette under the `/floppy` directory.

Note – If a diskette is not in the diskette drive, an error message is not displayed. The `volcheck` command redisplays the prompt.

---

3. In a terminal window, type **`cd /floppy/floppy0`** and press the Return key.
4. Type **`ls`** and press the Return key.

   The names of the files on the diskette are displayed. You can copy files to and from the diskette using the `cp` command.

Note – You can access the files on a diskette either from the command line or from the File Manager window by selecting Open Floppy from the File Menu.

Ejecting Removable Media Devices

After you have finished working with the CD-ROM or a diskette, you can use the `eject` command to remove the device from the drive.

Ejecting a CD-ROM

To eject a CD-ROM from the drive:

1. Click File in the CD-ROM File Manager window, and select the Eject option.

or

1. Close the CD-ROM File Manager window.
2. Type **`cd`** and press the Return key to change out of the `/cdrom` directory.
3. Type **`eject cdrom`** and press the Return key. After a few seconds, the CD-ROM ejects from the drive or a window displays to inform you to manually eject the CD-ROM.

Note – If the specific name of the media device is not known, you can type `eject -q` on the command line to ask the system if a removable media device is on the system.

Ejecting a Diskette

To eject a diskette from the diskette drive:

1. Click File in the File Manager disk window, and select the Eject option.

or

1. Close the File Manager disk window.
2. Type **`cd`** and press the Return key to change out of the `/floppy` directory.
3. Type **`eject floppy`** and press the Return key. After a few seconds, the diskette will eject from the diskette drive or a window is displayed to allow you to manually eject the diskette.

Device Busy Message

If the CD-ROM or diskette does not eject from the drive and the message "`Device busy`" appears, you might still be in the working directory on the CD-ROM or diskette.

You cannot eject removable devices while you are in the device's current working directory.

Check to see if you are in the `/cdrom` or `/floppy` directory by typing `pwd`. If either is the current directory, type `cd` to return home, and then type the `eject` command.

Exercise: Saving and Restoring Files

Exercise objective – In this exercise, you practice compressing and uncompressing files, viewing compressed files, and archiving files to tape or diskette.

Tasks

Note – If you get a "`Permission Denied`" error while performing the following exercises, check the write-protect switch on the tape cartridge or diskette.

Complete the following steps, and write the commands you would use to perform each task in the space provided.

1. Make a `cpio` copy of the files in your home directory in a file called `newfile`. View the contents of `newfile`.

   _____

2. In your home directory, compress the files `dante` and `file1` using the `compress` command.

   _____

3. What are the new names for the compressed versions of these two files?

   _____

4. What two commands can be used to view the contents of a file that was compressed with the `compress` command?

   _____

5. Compress the files `file2` and `dante_1` using the `gzip` command.

   _____

6. What are the new names for the compressed versions of these two files?

   _____

7. Compress the files `file3`, `fruit2`, and `tutor.vi` to a file called `files.zip` using the `zip` command.

   _____

8. What is the new name for the packaged and compressed version of these three files?

   _____

   Do the original versions of these three files still exist?

   _____

9. Uncompress the files `dante` and `file1`. What command should you use?

   _____

   Do these files still have a `.Z` extension on the file names?

   _____

10. Uncompress the files `file2` and `dante_1`. What command should you use?

    _____

    Do these files still have a `.gz` extension on the file names?

    _____

11. Unarchive `file3`, `fruit2`, and `tutor.vi` from the zip file created in Step 7. What command should you use?

    _____

    Does the zipped file `files.zip` still exist in the directory?

    _____

12. Archive your home directory to a file using the `tar` command.

_____

13. Compress the `tar` file using the `compress` command, and archive it to tape.

_____

14. After first ensuring that all files in `~/practice` are readable, use the `jar` command to archive `~/practice.`

_____

15. Use the `tar` command to archive `~/practice`, and compress the file.

_____

16. Compare the `tar` and `jar` file archives of `~/practice` for current size.

_____

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercises.

- Experiences
- Interpretations
- Conclusions
- Applications

Task Solutions

1. Make a `cpio` copy of the files in your home directory in a file called `newfile`. View the contents of `newfile`.

```
$ cd
$ ls | cpio -oc -O newfile.cpio
$ cpio -icv -I newfile.cpio
```

2. In your home directory, compress the files `dante` and `file1` using the `compress` command.

```
$ compress dante
$ compress file1
```

3. What are the new names for the compressed versions of these two files?

```
dante.Z
file1.Z
```

4. What two commands can be used to view the contents of a file that was compressed with the `compress` command?

```
$ uncompress -c filename
```

and

```
$ zcat filename
```

5. Compress the files `file2` and `dante_1` using the `gzip` command.

```
$ gzip file2 dante_1
```

6. What are the new names for the compressed versions of these two files?

```
file2.gz
dante_1.gz
```

7. Compress the files `file3`, `fruit2`, and `tutor.vi` to a file called `files.zip` using the `zip` command.

```
$ zip files.zip file3 fruit2 tutor.vi
```

8. What is the new name for the packaged and compressed version of these three files?

   `files.zip`

   Do the original versions of these three files still exist?

   Yes.

9. Uncompress the files `dante` and `file1`. What command should you use?

   ```
   $ uncompress dante
   $ uncompress file1
   ```

   Do these files still have a `.Z` extension on the file names?

   No.

10. Uncompress the files `file2` and `dante_1`. What command should you use?

    ```
    $ gunzip file2 dante_1
    ```

    Do these files still have a `.gz` extension on the file names?

    No.

11. Unarchive `file3`, `fruit2`, and `tutor.vi` from the zip file created in Step 7. What command should you use?

    ```
    $ unzip files.zip
    ```

    Does the zipped file `files.zip` still exist in the directory?

    Yes.

12. Archive your home directory to a file using the `tar` command.

    ```
    $ cd
    $ cd ..
    $ tar cvf /tmp/homedir.tar login-ID
    $ tar tvf /tmp/homedir.tar
    ```

13. Compress the `tar` file using the `compress` command, and archive it to tape.

```
$ cd /tmp
$ compress homedir.tar
$ tar cv homedir.tar.Z
```

14. After first ensuring that all files in `~/practice` are readable, use the `jar` command to archive `~/practice`.

```
$ cd
$ chmod u+r practice/memo
$ jar cvf practice.jar practice
```

15. Use the `tar` command to archive `~/practice`, and compress the file.

```
$ cd
$ tar cvf practice.tar practice
$ compress -v practice.tar
```

16. Compare the `tar` and `jar` file archives of `~/practice` for current size.

```
$ ls -l *jar *tar.Z
```

Check Your Progress

Before continuing on to the next module, check that you are able to accomplish or answer the following:

- ❑ Determine which commands are suitable for storing, viewing, or retrieving different types of files
- ❑ Demonstrate how to reduce the size of files and directories and store them to tape using the `compress` and `tar` commands
- ❑ Describe the steps for uncompressing or viewing a compressed file with the `uncompress` and `zcat` commands
- ❑ Use the `gzip` and `gunzip` commands to compress and uncompress files
- ❑ Use the `zip` command to package and compress multiple files and use `unzip` to uncompress a zipped archive file
- ❑ Compress and copy multiple files to a single archive file in one step using the `jar` command
- ❑ Copy and extract files from an archive file or tape device with the `cpio` command
- ❑ Understand how to use volume management to access CD-ROMs and diskettes
- ❑ Use the `eject` command to remove CD-ROMs and diskettes from device drives

Remote Connections 9

Objectives

Upon completion of this module, you should be able to:

- Open a session on a remote system using `telnet`
- Log in remotely to another system on the network
- Use `ftp` to get a file from a remote system

Additional Resources

Additional resources – The following references can provide additional details on the topics discussed in this module:

- *Solaris™ Common Desktop Environment: User's Guide,* "Starting a Desktop Session," Part Number 806-1360-10
- *System Administration Guide, Volume 1*, Part Number 805-7228-10

Example Networking Environment

Figure 9-1 shows the relationship between the network and the host.

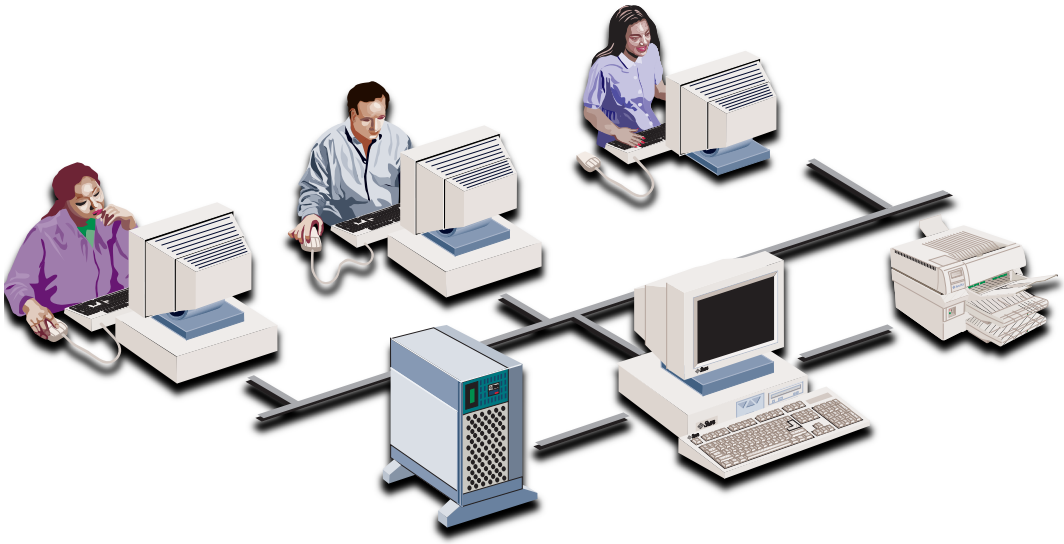


Figure 9-1 Example Networking Environment

Network

A *network* is a connection that enables an exchange of information between systems. Two types of networks are:

- Local area network (LAN) – A network that covers a small area, usually less than a few thousand feet or meters
- Wide area network (WAN) – A network that can span thousands of miles or kilometers

Host

A *host* is a computer system on a network. The *local host* is the system on which you are currently working. A *remote host* is a system that is being accessed by a user on some system that, from that user's point of view, is a *local host*. That is, the terms *local host* and *remote host* are all relative to some particular user's perspective.

Using the `telnet` *Command*

The `telnet` command is an application that is part of the Solaris 8 Operating Environment. It uses Transmission Control Protocol/ Internet Protocol (TCP/IP) to connect to another system.

The `telnet` connection enables you to login to a remote system and work in that environment. When using `telnet`, you can:

- Open a session on a remote system
- Access systems that do not run under the UNIX environment

Command Format

`telnet` *`hostnameXS`*

The following is an example of using `telnet` to connect to a remote system called `host1`:

```
$ telnet someotherhost
Trying host1
Connected to host1
Escape character is '^]'.

SunOS 5.8

login: user2
Password:
Last login: Mon Mar  6 14:13:40 from host1
Sun Microsystems Inc.   SunOS 5.8       Generic February 2000
$
$ uname -n
someotherhost
$ exit
Connection closed by foreign host.
$
```

Using the `rlogin` *Command*

Use the `rlogin` command to establish a remote login session on another workstation.

Command Format

`rlogin` *`hostname`*

Example

To remotely login to another host, execute the following:

```
$ rlogin host2
Last login: Mon Mar  6 16:22:12 from host1
Sun Microsystems Inc.   SunOS 5.8       Generic February 2000
$ id
uid=11001(user1) gid=10(staff)
$ uname -n
host2
$ pwd
/export/home/user1
$ exit
Connection closed.
$
```

Specifying a Different User Name

Use the `-l` option to specify a different login ID (user name) for the remote login session.

Command Format

`rlogin [ -l` *`username`* `]` *`hostname`*

Before attempting to remotely login to another system as a different user, be sure an account for that user exists on the desired remote system. Check with the system administrator if a user account is required on the remote system. The information you need to know includes:

- Host name
- User name
- Password for the new account

Logging in Remotely as Another User

The following is an example of logging in remotely as another user:

```
$ rlogin -l user2 host1
Last login: Mon Mar  6 16:36:35 from host2
Sun Microsystems Inc.   SunOS 5.8       Generic February 2000
$ id
uid=11002(user2) gid=10(staff)
$ pwd
/export/home/user2
$ uname -n
host1
$ exit
Connection closed.
$
```

Executing a Program on a Remote System

Use the `rsh` command to execute a program on a remote system.

Command Format

```
rsh [ -l username ] hostname command
rsh [ -l username ] IP_Address command
```

Example

For example, to run commands remotely, you can execute one of the following commands:

```
$ rsh host1 showrev
```

```
$ rsh -l user2 host1 ls /var/mail
```

Copying To and From Another System

The `rcp` command enables you to copy files or directories to and from another system.

Command Format

```
rcp source_file hostname:destination_file

rcp hostname:source_file destination_file
```

Copying Files Across the Network

- To copy files from a local directory to a remote host, use the following syntax:

```
$ rcp dante saturn:/tmp
```

- To copy files from a remote host to `/tmp`, use the following syntax:

```
$ rcp saturn:/tmp/dante /tmp
```

- To remotely copy directories with the `-r` option, use the following syntax:

```
$ rcp -r $HOME/perm saturn:/tmp
```

If you are in a directory that contains the file or directory that you want to copy to another system, type the file or directory name. It is not necessary to type the absolute path name.

Caution – The `/tmp` directory is used to store files temporarily. Do not use `/tmp` for long-term storage of important files. The `/tmp` directory is cleared out each time the system is rebooted.

Using the `ftp` *Command*

The `ftp` (File Transfer Protocol) command is an implementation of an industry-standard protocol. Use the `ftp` command to transfer files using ASCII or binary mode between systems using similar or dissimilar operating systems.

After you have successfully used `ftp` to access a remote site, some familiar file and directory access commands, such as `cd` and `ls`, are available.

If permissions are set by the system administrator for you to view the contents of a directory, the `ls` command displays files in that directory.

If permissions are set such that you do not have access to the files, a prompt is returned in response when you enter the `ls` command.

As on your local system, `cd` changes directories on the remote system.

To change directories on your own system in the middle of the `ftp` session, use the `lcd` (local change directory) command.

To end an `ftp` session, type `bye` at the prompt.

Command Format

`ftp` *`hostname`*

Examples

The following examples show how to use `ftp`:

```
$ ftp host1
Connected to host1.
220 host1 FTP server (SunOS 5.8) ready.
Name (host1:user1): user2
331 Password required for user2.
Password:
230 User user2 logged in.
ftp> ls
200 PORT command successful.
150 ASCII data connection for /bin/ls (192.9.200.1,32970) (0 bytes).
dante
dante_1
dat
dir1
dir2
dir3
dir4
file1
file2
file3
file4
fruit
fruit2
practice
tutor.vi
226 ASCII Transfer complete.
113 bytes received in 0.0033 seconds (33.84 Kbytes/s)
ftp> cd dir1
250 CWD command successful.
ftp> ls
200 PORT command successful.
150 ASCII data connection for /bin/ls (192.9.200.1,32971) (0 bytes).
coffees
constellation
feathers
feather_6
fruit
planets
trees
226 ASCII Transfer complete.
23 bytes received in 0.0021 seconds (10.83 Kbytes/s)
ftp> cd coffees
```

```
250 CWD command successful.
ftp> ls
200 PORT command successful.
150 ASCII data connection for /bin/ls (192.9.200.1,32973) (0 bytes).
beans
226 ASCII Transfer complete.
27 bytes received in 0.0022 seconds (12.06 Kbytes/s)
ftp> bin
200 Type st to I.
ftp> get beans
200 PORT command successful.
150 Binary data connection for beans (192.9.200.1,32974) (0 bytes).
226 Binary Transfer complete.
ftp> lcd
Local directory now /export/home/user1
ftp> bye
221 Goodbye.
$
```

Exercise: Performing Network Basics

Exercise objective – In this exercise, you use some of the networking commands introduced in this module.

Tasks

Complete the following steps:

1. Use the `rlogin` command to login to another system in your classroom.

   What directory are you in on the remote system?

   _____

2. Issue the command that shows you the host name of the current system.

   _____

3. Log out of the remote system. Display the host name of your current system to determine if you are back to your own host.

   _____

4. Use the `rlogin` command and option to log in to another system as the user `guest` with a password of `guest` (or as another user as specified by your instructor).

   _____

5. Log out of the remote system.

   _____

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercises.

- Experiences
- Interpretations
- Conclusions
- Applications

Task Solutions

Complete the following steps:

1. Use the `rlogin` command to login to another system in your classroom.

   ```
   $ rlogin hostname
   ```

   What directory are you in on the remote system?

   A home directory on the remote system (either `/home/username` *or* `/export/home/username`*) or the* `root` *(/) directory if no home directory exists.*

2. Issue the command that shows you the host name of the current system.

   ```
   $ uname -n
   ```

3. Log out of the remote system. Display the host name of your current system to determine if you are back to your own host.

   ```
   $ exit
   $ uname -n
   ```

4. Use the `rlogin` command and option to log in to another system as the user `guest` with a password of `guest`.

   ```
   $ rlogin hostname -l guest
   ```

5. Log out of the remote system.

   ```
   $ exit
   ```

Check Your Progress

Before continuing on to the next module, check that you are able to accomplish or answer the following:

- Open a session on a remote system using `telnet`
- Log in remotely to another system on the network
- Use `ftp` to get a file from a remote system

System Processes 10

Objectives

Upon completion of this module, you should be able to:

- Describe how processes are created
- View active processes on the system using the `ps` command
- Find a specific process using the `pgrep` command
- Discuss the purpose of signals for controlling process activity
- End a process using the `kill` and `pkill` commands
- Use job control commands to manage jobs running in the shell

Additional Resources

Additional resources – The following references provide additional details on the topics discussed in this module:

- *Solaris Common Desktop Environment: User's Guide,* "Starting a Desktop Session," Part Number 806-1360-10
- *System Administration Guide, Volume 1*, Part Number 805-7228-10

Process Overview

Every program you run in the Solaris Operating Environment creates a *process*. When you log in and start the shell, it is a process. When you execute a command or you open an application, it starts a process.

A process is any program running on the system.

The system also starts processes called daemons. Daemons are processes that are run to perform a specific task. For instance, the desktop login daemon (`dtlogin`) provides a graphical prompt for a user's login name and password.

Every process is assigned a unique process identification number (PID), which is used by the kernel to track and manage the process. For a user, the PID number is used to identify and control the process.

Process UID and GID

For the kernel to know what a process is allowed to do, it must store information about who owns the process. The kernel stores UIDs and GIDs for this purpose.

The UID and GID of a process are the same as the UID and GID of the user who started the process.

Parent Process

When a process is started, a duplicate of that process is created. This new process is called the child, and the process that created it is called the parent. The child process then replaces the copy of the code the parent process created with the code the child process is supposed to execute.

While the command is executing, the shell waits until the child process has completed. After it completes, the parent process terminates the child process, and a prompt is displayed, ready for a new command.

Viewing Processes and PIDs

The ps (process status) command lists the processes currently running on the system.

For each process, ps displays the process ID (PID), terminal identifier (TTY), cumulative execution time (TIME), and the command (CMD).

Command Format

```
ps -options
```

Options

You can use the following options with the ps command:

-e	Prints information about every process on the system, including PID, TTY, TIME, and CMD
-f	Generates a full (verbose) listing, which adds the fields UID (owner of process), PPID (parent process ID), and STIME (process start time)

Displaying a Full Listing of All Processes

The following example displays a listing of all processes.

```
$ ps -ef | more
UID      PID    PPID  C   STIME       TTY      TIME  CMD
root     0      0     0   16:46:41    ?        0:01  sched
root     1      0     0   16:46:44    ?        0:40  /etc/init -
root     2      0     0   16:46:44    ?        0:00  pageout
root     3      0     0   16:46:44    ?        4:33  fsflush
root     236    1     0   16:48:08    ?        0:01  /usr/lib/saf/sac
root     844    1     0   12:12:10    ?        0:00  /usr/lib/lpsched
--More--
```

Table 10-1 contains descriptions for the column headings.

Table 10-1 Column Headings for `ps -ef` Output

Value	Description
`UID`	The user name of the owner of the process.
`PID`	The unique process identification number of the process.
`PPID`	The parent process identification number for the process.
`C`	The CPU utilization for scheduling; this is obsolete.
`STIME`	The time the process started (*`hh:mm:ss`*).
`TTY`	The controlling terminal where the process started. (The controlling terminal for system daemons appears as a question mark [`?`]).
`TIME`	The cumulative execution time for the process.
`CMD`	The command name.

Searching for a Specific Process

To quickly locate a specific process, you can pipe the output of the `ps` command to `grep` for a specific command name. For example, to find all active processes related to printing, execute the following:

```
$ ps -e | grep lp
217 ?        0:00 lpsched
$
```

Note – `lpsched` is a daemon responsible for controlling print services on the system.

The pgrep *Command*

The pgrep command gives you an even more efficient method for quickly finding a specific process by name.

By default, pgrep displays the PID of any process that matches the specified criteria on the command line; for example:

```
$ pgrep lp
217
$
```

Command Format

```
pgrep -options pattern
```

Options

You can use the following options with the pgrep command.

- `-x` Displays only those PIDs that exactly match *pattern*.
- `-n` Displays only the newest (most recently created) PID that matches *pattern*. (Displays every process ID containing *pattern*.)
- `-U` Displays only those PIDs that belong to the specified owner. (Uses either the user name or a UID number.)
- `-l` Displays the name of the process along with its PID.

Note – The pgrep command has several other useful options. View the man page for this command for a list of all the available options.

In the following examples, using the `-l` option displays both the names of the process and their PIDs.

```
$ pgrep -l lp
217  lpsched

$ pgrep -l mail
   230  sendmail
 13453  dtmail
```

In the next example, combining the `-l` and `-x` options displays only those processes that match the exact names.

```
$ pgrep -lx dtmail
12047  dtmail
```

Sending Signals to Processes

Signals are used to control processes running on the system. Signals are sent to processes to indicate that an event has occurred and the process must respond.

For example, if you type Control-C to terminate a command, this sends an interrupt signal to the process, and the process responds by exiting.

A signal is a simple message containing a signal number as information to a process. There are a number of signals available in the Solaris Operating Environment. Each signal is associated with a unique number, a name, and an action.

You can find a complete list of these signals and their default actions on the following man page:

```
$ man -s3 signal
```

Table 10-2 describes some signals and names.

Table 10-2 Signal Number and Names

Signal Number	Signal Name	Action	Response
1	SIGHUP	Hangup	Exit
2	SIGINT	Interrupt	Exit
9	SIGKILL	Kill	Exit
15	SIGTERM	Terminate	Exit

- 1, SIGHUP – A hangup signal to cause a telephone line or terminal connection to be dropped.
- 2, SIGINT – An interrupt signal generated from your keyboard—usually by Control-C.
- 9, SIGKILL – A signal used to kill a process. You cannot ignore this signal.
- 15, SIGTERM – A signal to terminate a process in an orderly manner. Some processes ignore this signal.

Terminating Processes

This following section will illustrate how to execute the `kill` command.

The `kill` Command

Use the `kill` command to send a signal to one or more running processes. This command is often used to terminate a process.

Note – Regular users can only kill their own processes. The `root` user can kill almost any process.

Command Format

```
kill signal PID ...
```

Terminating a Process

Before you can terminate a process, you must know its PID. Use either the `ps` or `pgrep` command to locate the PID for the process.

To terminate a process, run the command:

```
$ kill PID
```

For example:

```
$ pgrep -l mail
   215  sendmail
 12047  dtmail
$
$ kill 12047
$
```

To terminate more than one process at the same time:

```
$ kill PID PID PID PID
```

Using the `kill` command without specifying a signal on the command line sends the Default Signal `15` to the process. This signal usually causes the process to terminate.

Some processes will ignore Signal `15`. These can be processes waiting for a resource; for example, a process that is waiting for a tape drive to complete an operation so it can continue.

Those processes that do not respond to a Signal `15` can be terminated by force using Signal `9` with the `kill` command.

For example:

```
$ kill -9 PID
```

or

```
$ kill -KILL PID
```

Warning – Use the `kill -9` or `kill -KILL` command as a last resort to terminate a process. Using `kill -9` or `kill -KILL` on a process that controls a database application or a program that updates files can be disastrous! The process is terminated instantly with no opportunity to perform an orderly shutdown.

The `pkill` *Command*

Use the `pkill` command to terminate a process, by default, using Signal `15`.

This command can terminate a process using its process name.

Command Format

`pkill [-`*`options`*`] `*`pattern`*

The options used with `pkill` are the same as those used by the `pgrep` command.

To terminate the mail session process by its name, execute the following:

```
$ pgrep -l mail
 215 sendmail
 470 dtmail

$ pkill dtmail
```

To terminate the `dtmail` session process by its PID, you use the `kill` command.

```
$ kill 470
```

Killing Processes Remotely

When a workstation is not responding to your keyboard or mouse input, the window system might be frozen. In such cases, you can remotely access your workstation using `rlogin` (or `telnet`) from another system.

After you are connected, you can invoke the `pkill` command to terminate the corrupted session on your workstation; for example:

```
$ rlogin host1
Password:
Last login: Fri Feb 04 16:50:30 from host1
Sun Microsystems Inc. SunOS 5.8 Generic February 2000
$ pkill -9 shell
```

or

```
$ pkill -KILL shell
```

Managing Jobs

A job is a process controlled by a terminal and contains a process ID. Every job is assigned a *job ID* by the shell. The handling of multiple jobs within a shell is called *job control*.

The shell gives you the ability to execute a variety of jobs simultaneously. Opening an application, sending a print request, or executing an `ls` command on a directory are all examples of jobs.

When a job is executed in the window environment, it runs in the foreground and ties up that particular window until the job is done.

However, you can execute jobs by the shell in the background, freeing up the window so you can start another job in the foreground.

You can control jobs by their job ID numbers using the commands in Table 10-3.

Table 10-3 Job Control Commands

Command	Value
`jobs`	Displays which jobs are currently running
`bg %`*n*	Places the specified job in the background (*n* is the job ID)
`fg %`*n*	Places the specified job in the foreground (*n* is the job ID)
`^Z`	Stops the foreground job
`stop %`*n*	Stops the specified background job (*n* is the job ID)

Note – You can control a job only by using these commands in the window where the job started.

To have a job run in the background, type the command to be executed, followed by an ampersand (&) symbol.

For example, this command line executes the `find` command in the background. It locates all files named `core` in the current working directory. It then prints the full path name of each `core` file into a new file called `list`.

```
$ find . -name core > list &
[1]     3028
$
```

The shell returns a job ID number contained in brackets and a PID number for the command. The job ID number gives you the ability to control the job. The PID number is used by the kernel to manage the job.

When you press the Return key in the window, a message is displayed indicating that the background job has completed.

```
[1] +  Done find . -name core > list &
$
```

- Use the `jobs` command to list your current jobs.

```
$ jobs
[1] + Running find . -name core > list &
```

- Use the `fg` command to bring a background job to the foreground.

```
$ fg %1
find . -name core > list
```

Note – This occupies the window until the job is completed, stopped, or stopped and placed back into the background.

- To return this job to the background, suspend it first using the Control-Z keys, and then use the `bg` command.

```
$ find . -name core > list
^Z
[1] + Stopped(SIGTSTP) find . -name core > list &

$ jobs
[1] + Stopped(SIGTSTP) find . -name core > list &

$ bg %1
[1] find . -name core > list &
```

Note – Placing a stopped job into either the foreground or the background restarts the job.

- To stop a background job, use the specific job number as the argument to the `stop` command.

```
$ stop %1
[1] + Stopped (SIGSTOP)        script1 &
$
```

Exercise: Manipulating System Processes

Exercise objective – In this exercise, you use the commands learned in this module to determine PID numbers, kill processes, and control jobs.

Tasks

Complete the following steps, and write the commands you would use to perform each task in the space provided.

1. Issue the following command in the background:

   ```
   $ sleep 500 &
   ```

2. Using the `jobs` command, find the job number of the `sleep` command in Step 1.

   Job number _____

3. Bring the job to the foreground, and then put it back in the background.

   _____

4. Kill the job running the `sleep` command.

   _____

5. Use the following `ps` commands to list the processes currently running on your system. What information does each command provide?

   ```
   $ ps
   ```

   _____

   ```
   $ ps -f
   ```

   _____

```
$ ps -e
```

_____

```
$ ps -ef
```

_____

6. In a terminal window, run the `ps -ef` command. Identify the PID of this command.

   ```
   $ ps -ef
   ```

   PID number: _____

7. In a separate terminal window, issue the following command:

   ```
   $ cat -v /dev/zero
   ```

Note – This command is being used to produce a continuously running process for demonstration purposes only. For information on the `/dev/zero` file, refer to `man zero`.

---

8. Open another terminal window, and use the `ps` command to identify the PID of the `cat` command.

   PID number: _____

9. From this terminal window, kill the `cat` command using the PID number.

   _____

10. From the same window, enter the `tty` command to identify the name of this terminal window. The name displays as `/dev/pts/#`, where # is an actual number; for example, `/dev/pts/4`.

    ```
    $ tty
    ```

    /dev/pts/_____

11. Move back to the original terminal window, and use `pgrep` to find the PID associated with the name of the second terminal window.

    ```
    $ pgrep -t pts/#
    ```

    PID number: _____

12. In the current window, use the `kill` command to kill the second terminal window.

    ```
    $ kill PID#
    ```

    Did it work? _____

13. Use the `kill` command with the `-9` option to kill the second terminal window.

    ```
    $ kill -9 PID#
    ```

    Did it work? _____

14. Start a Korn shell in the remaining window.

    ```
    $ ksh
    ```

15. Run the following `kill` commands to identify the signals it sends when you use the specified options.

    ```
    $ kill -l 9
    ```
    (- l is the letter l)

    Signal: _____

    ```
    $ kill -l 15
    ```
    (- l is the letter l)

    Signal: _____

16. Exit the Korn shell.

    ```
    $ exit
    ```

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercises.

- Experiences
- Interpretations
- Conclusions
- Applications

Task Solutions

Complete the following steps, and write the commands you would use to perform each task in the space provided.

1. Issue the following command in the background:

   ```
   $ sleep 500 &
   ```

2. Using the `jobs` command, find the job number of the `sleep` command started in Step 1.

   ```
   $ jobs
   ```

3. Bring the job to the foreground, and then put it back in the background.

   ```
   $ fg %1
   ^Z
   $ bg %1
   ```

4. Kill the job running the `sleep` command.

   ```
   $ kill %1
   ```

5. Use the following `ps` commands to list the processes currently running on your system. What information does each command provide?

   ```
   $ ps
   ```

   Prints information for the current user and terminal.

   ```
   $ ps -f
   ```

   Prints a full listing of the above.

   ```
   $ ps -e
   ```

   Prints information about every process running.

   ```
   $ ps -ef
   ```

   Prints a full listing of the above.

6. In a terminal window, run the `ps -ef` command. Identify the PID of this command.

   The PID number differs from system to system.

7. In a separate terminal window, issue the following command:

   ```
   $ cat -v /dev/zero
   ```

8. Open another terminal window, and use the `ps` command to identify the PID of the `cat` command.

   ```
   $ ps -ef | grep cat
   ```

9. From this terminal window, kill the `cat` command using the PID number.

   ```
   $ kill PID
   ```

   where `PID` is the PID of the `cat` command.

10. From the same window, enter the `tty` command to identify the name of this terminal window. The name displays as `/dev/pts/#`, where # is an actual number; for example, `/dev/pts/4`.

    ```
    $ tty
    ```

    `/dev/pts/#` – *This name differs from system to system.*

11. Move back to the original terminal window, and use `pgrep` to find the PID associated with the name of the second terminal window.

    ```
    $ pgrep -t pts/#
    ```

    The PID number differs from system to system.

12. In the current window, use the `kill` command to kill the second terminal window.

    ```
    $ kill PID#
    ```

    Did it work?

    No.

13. Use the `kill` command with the `-9` option to kill the second terminal window.

    ```
    $ kill -9 PID#
    ```

    Did it work?

    Yes.

14. Start a Korn shell in the remaining window.

    ```
    $ ksh
    ```

15. Run the following `kill` commands to identify the signals it sends when you use the specified options.

    ```
    $ kill -l 9
    ```

    Signal `KILL`.

    ```
    $ kill -l 15
    ```

    Signal `TERM`.

16. Exit the Korn shell.

    ```
    $ exit
    ```

Check Your Progress

Before continuing on to the next module, check that you are able to accomplish or answer the following:

- ❑ Describe how processes are created
- ❑ View active processes on the system using the `ps` command
- ❑ Find a specific process using the `pgrep` command
- ❑ Discuss the purpose of signals for controlling process activity
- ❑ End a process using the `kill` and `pkill` commands
- ❑ Use job control commands to manage jobs running in the shell

The Korn Shell 11

Objectives

Upon completion of this module, you should be able to:

- Describe the functions of the Korn shell as a command interpreter
- Demonstrate the use of quoting to mask the special meaning of metacharacters by the Korn shell
- Define the terms standard input, standard output, and standard error
- Use metacharacters to redirect standard input, standard output, and standard error
- Connect two or more commands together using the pipe feature
- Implement the file name completion mechanism in the Korn shell
- Use commands to view, set, and unset shell variables
- Invoke the `history` mechanism to repeat or edit previously executed commands
- Use the `alias` utility to customize and abbreviate UNIX commands
- Create Korn shell functions to construct customized commands
- Define the Korn shell initialization files used to customize a user's environment

The Shell as a Command Interpreter

In any UNIX environment, the shell is commonly referred to as a *command interpreter*. It allows you to interact with the kernel by interpreting commands that are either typed on the command line or placed within a shell script.

The shell as a command interpreter accepts, analyzes, and processes the input from users or shell scripts. The shell also generates appropriate error messages.

Shells are interchangeable, so you can chose which command interpreter to use at any time—changing easily between the Bourne shell, the C shell, the Korn shell, the TC shell, the Z shell, the BASH shell, or any other available shell.

Although the shell acts as an intermediary between the user and the kernel, it has other important functions, too.

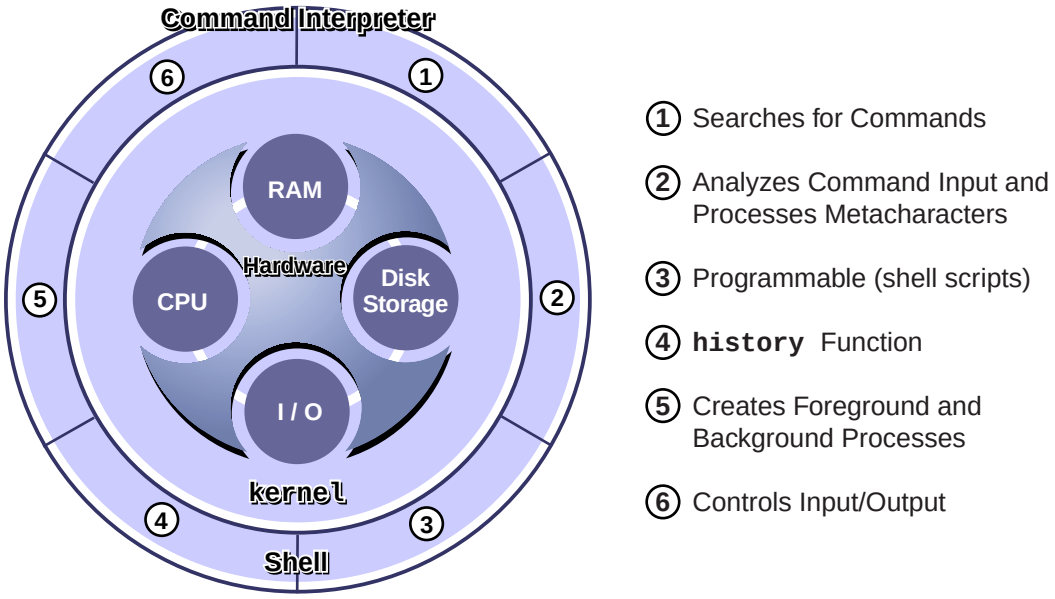


Figure 11-1 Tasks Accomplished by the Shell

Responsibilities of the Shell as a Command Interpreter

The following section describes the responsibilities of the shell:

- The shell searches for commands in all directory locations defined in the PATH shell variable. This is a colon-separated list of directories. The shell searches these directories from left to right to locate the command to be executed.
- The shell sets up pipes, I/O redirection, and background processing.
- The shell can be custom-tailored by each individual user by creating aliases (abbreviated names) for a command or series of commands. When an alias is not sufficient, users can create shell functions.
- The shell can be adapted to different terminal environments by setting the shell variable TERM.
- The shell saves typing with the command-line completion mechanism.
- The shell stores previously executed commands in a history list that can be rerun and edited.
- The shell can be customized for each user on the system through the use of shell initialization files.

Input and Output Redirection and Piping

This section discusses the redirection and piping of input and output.

Redirecting Input/Output

The shell typically receives or reads command input from the keyboard and displays or writes command output to the terminal screen.

You can, however, instruct the shell to redirect command input and command output by using the redirect (< and >) symbols.

I/O redirection is commonly used for redirecting input and output to files on the command line or within shell scripts.

Input redirection is the ability to force a command to read input from a file instead of from the keyboard.

Output redirection is the ability to send the output from a command into a file or to another command (using a pipe) instead of sending the output to the terminal screen.

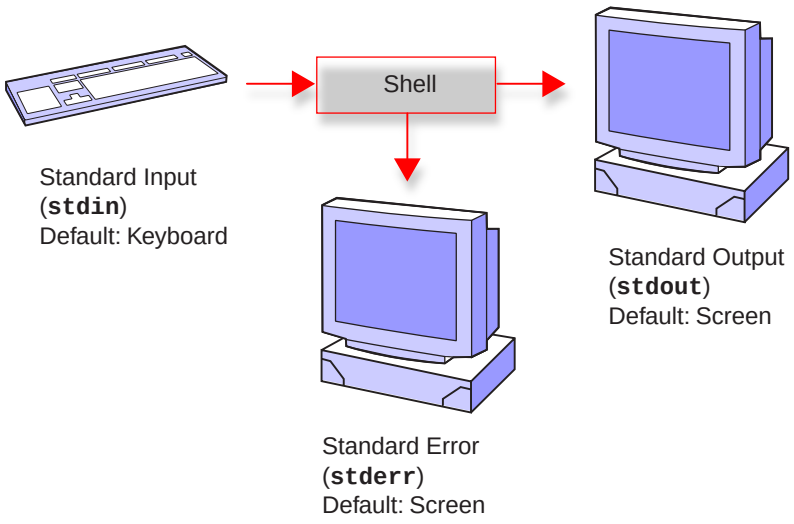


Figure 11-2 Standard Command I/O in the Shell

File Descriptors

Each process created by the shell is associated with the file descriptors listed in Table 11-1.

Table 11-1 File Descriptors

File Descriptor	Meaning
`0 stdin`	Standard input
`1 stdout`	Standard output
`2 stderr`	Standard error

These file descriptors are used by the shell to determine where input to the command comes from, and where the output and error messages are sent.

The following describes the standard command input and output:

- Standard input (`stdin`) is always File Descriptor `0` *(zero)*.
- Standard output (`stdout`) is always File Descriptor `1`.
- Standard error (`stderr`) is always File Descriptor `2`.

All commands processing file content are implemented to read from standard input and write to standard output.

This is demonstrated by invoking the `cat` command without arguments and typing the two lines of text. To exit, press Control-D.

```
$ cat                (read from stdin)
First line           (read from stdin)
First line           (write to stdout)
What's going on?     (read from stdin)
What's going on?     (write to stdout)
^d                   (read from stdin)
$
```

The `cat` command takes its standard input from the keyboard and displays the standard output to the terminal window.

Redirecting stdin, stdout, *and* stderr

You can change the default behavior of standard input, standard output, and standard error within the shell.

Redirecting stdin

The following shows what a stdin command will do:

- *command* < *filename*
- *command* 0< *filename*

The *command* reads its input from the file called *filename* instead of reading input from the keyboard; for example:

```
$ mailx user1 < ~/dante
```

Redirecting stdout

The following example shows the redirecting of stdout:

- *command* > *filename*
- *command* 1> *filename*

The *command* output is directed to a file called *filename*. If *filename* does not exist, it is created. If *filename* exists, the redirection overwrites the contents of the file; for example:

```
$ ps -ef > process_list
```

Redirecting `stdout` *Using Append Mode*

The following example shows the redirecting of `stdout` using the append symbols:

- *`command >> filename`*

  The command output is directed to a file called *`filename`* and is appended to the end of the existing file content. If *`filename`* does not exist, it is created.

  For example:

```
$ cat /etc/passwd > my_file;cat my_file
$ echo "That's my passwd file" >> my_file;cat my_file
That's my passwd file
$
```

Redirecting `stderr`

The following example shows the redirecting of `stderr`:

- *`command`* `2> /dev/null`

  Any command error messages are redirected to the file `/dev/null`.

  This is useful to suppress error messages considered insignificant, so that no error messages display on the terminal screen.

  For example:

```
$ find /etc -type f -exec grep PASSREQ {} \; -print 2> /dev/null
# PASSREQ determines if login requires a password.
PASSREQ=YES
/etc/default/login
$
```

Redirecting `stderr` *to* `stdout`

The following example shows the redirecting of `stdout` and `stderr`:

- *`command`* `1>` *`filename`* `2>&1`

  The syntax `2>&1` instructs the shell to redirect `stderr` (2) to the same file to which `stdout` (1) is directed to write its results; for example:

```
$ ls  /var  /no  1>  dat  2>&1
$ more dat
/no: No such file or directory              (stderr)
/var:                                           (stdout)
adm                                             (stdout)
audit                                           (stdout)
cron                                            (stdout)
<output omitted>
```

The Pipe Feature

The shell enables you to effectively connect two commands together. This connection is known as a pipe (|). Building a pipe on the command line takes the output from one command and passes it directly into the input of another command.

A pipe is indicated by the character | and is placed between two commands.

Command Format

command | *command*

Some Basic Examples Using a Pipe

You can insert a pipe between any two commands, provided the first command writes its output to standard output, and the second command reads its input from standard input.

For example:

```
$ who | wc -l
       5
$
```

The standard output of the first command is sent directly as the standard input for the second command.

The output of the who command never appears on the terminal screen because it is *piped* directly into the wc command.

To obtain the total number of active processes on the system:

```
$ ps -ef | wc -l
      62
$
```

To get a listing of all the subdirectories located in `/etc`:

```
$ ls -F /etc | grep "/"
acct/
cron.d/
default/
...
```

Building a Pipeline

You can create a pipeline that consists of more than two commands.

You can connect an unlimited number of commands by pipes.

Example 1

```
$ head -10 dante | tail -3 | lp
request id is printerA-177        (standard input)
```

Example 2

```
$ ps -ef | tail +2 | wc -l
      74
```

Korn Shell Option Settings

Options are switches that control the behavior of the Korn shell. They are boolean, in that each can be either `on` or `off`.

To switch an option `on`, type:

```
$ set -o option_name
```

To switch an option `off`, type:

```
$ set +o option_name
```

To show all set options, type:

```
$ set -o
```

Note – The `set -o` and `set +o` options can only change a single option setting at a time.

Protecting File Content During I/O Redirection

Redirecting standard output to an existing file overwrites the previous file content, which results in data loss. This process of overwriting existing data is known as "*clobbering*." To prevent an overwrite from occurring, the shell supports a `noclobber` option.

When the `noclobber` option is set, the shell refuses to redirect standard output to the existing file and displays an error message to the screen.

The `noclobber` option is activated in the shell with the `set` command; for example:

```
$ set -o noclobber
$ set -o | grep noclobber
noclobber       on
$ ps -ef > file_new
$ cat /etc/passwd > file_new
ksh: file_new: file already exists
$
```

Deactivating the noclobber *Option*

To temporarily deactivate the noclobber option, use the syntax >| on the command line. The noclobber option is ignored for this command line only, and the contents of the file are overwritten.

```
$ ls -l >| file_new
```

Note – There is no space between the “>” and “|” on the command line.

- To deactivate the noclobber option, execute the following:

```
$ set +o noclobber
$ set -o | grep noclobber
noclobber       off
$ ls -l > file_new
$
```

File Name Completion in the Korn Shell

File name completion, often referred to as file name expansion, is a mechanism that allows you to type the first few characters of a file name. Then, by pressing a specific sequence of keys, you instruct the Korn shell to complete the remainder of the file name.

Note – The Korn shell has two modes for command-line editing (including file name completion). These are "`vi`-mode" and "`emacs`-mode." The following examples demonstrate the "`vi`-mode" commands. See the `ksh(1)` man page for information on the "`emacs`-mode" commands.

Using File Name Completion

To turn on the `vi` built-in editor to activate file name completion, you can run one of the following commands:

```
$ set -o vi
or
$ export EDITOR=/bin/vi
or
$ export VISUAL=/bin/vi
```

To invoke file name completion, type the `ls` command followed by one or more characters of a file name, and then press the following keys in sequential order: Escape (`Esc`) and backslash (\).

Note – The key sequence presented previously applies only to `vi`-mode command-line editing.

If the shell finds a file name beginning with the specified characters, it prints the complete file name or file names to the command line.

For example, to expand a file name beginning with the characters "`de`" in the `/usr` directory:

```
$ cd /usr
$ ls de      Press Esc and \
```

The shell completes the remainder of the file name, displaying:

```
$ ls demo/
```

You can request the shell to present all the possible alternatives of a partial file name from which you could then select. This action is invoked by pressing the following keys sequentially: Escape (Esc) and the equal (=) key.

To request that the shell present all file names beginning with the letter "g" in the /etc directory, type:

```
$ cd /etc
$ cat g          Press Esc, press the = key
1) getty
2) group
3) grpck
4) gss/
$ cat g
```

The cursor is positioned on top of the letter "g" at this point.

Korn Shell Variables

A variable is a name that refers to a temporary storage area in memory. Variables contain information used for customizing the shell and information required by other processes so they function properly.

The shell allows you to store values into variables.

Korn shell programming uses two types of variables: variables that are exported to subprocesses and variables that are not.

Table 11-2 summarizes the Korn shell commands to `set`, `unset`, or view variables:

Table 11-2 Korn Shell Commands for Variables

Action	Command
To set a variable	`VAR=`*`value`* `export VAR=`*`value`*
To unset a variable	`unset` *`VAR`*
To display all variables	`set`, `env`, or `export`

To display values stored in variables	`echo $VAR`
	or
	`print $var`

If a valid shell variable name follows the $ sign, the shell takes this as an indication that the value stored inside that variable is to be substituted at that point.

Referencing Values in Variables

You can use the `echo` command to display the value that is stored inside a shell variable; for example:

```
$ echo $SHELL
/bin/ksh
```

Displaying Variables

The `set` command lists all shell variables with their current values; for example:

```
$ set
DISPLAY=:0.0
EDITOR=/usr/bin/vi
ERRNO=13
FCEDIT=/bin/vi
HELPPATH=/usr/openwin/lib/locale:/usr/openwin/lib/help
HOME=/export/home/user1
HZ=100
IFS=
LANG=C
LINENO=1
LOGNAME=user1
MAIL=/var/mail/user1
MAILCHECK=600
MANPATH=/usr/man:/usr/openwin/share/man
OLDPWD=/export/home/user1
OPENWINHOME=/usr/openwin
PATH=/usr/openwin/bin:/bin:/usr/bin:/usr/ucb:/usr/sbin
PPID=596
PS1='$ '
PS2='> '
PS3='#? '
PS4='+ '
PWD=/tmp
SHELL=/bin/ksh
TERM=dtterm
TERMINAL_EMULATOR=dtterm
TMOUT=0
TZ=MET
USER=user1
_=set
office=/export/home/user1/office
private=/export/home/user1/private
```

To make the value of a variable known to a subshell, it must be exported with the `export` command.

You can view the list of all these variables and their current values with the `export` command.

```
$ export
DISPLAY=:0.0
EDITOR=/usr/bin/vi
HELPPATH=/usr/openwin/lib/locale:/usr/openwin/lib/help
HOME=/export/home/user1
LANG=C
LOGNAME=user1
MAIL=/var/mail/user1
MANPATH=/usr/openwin/share/man:/usr/man
OPENWINHOME=/usr/openwin
PATH=/usr/openwin/bin:/bin:/usr/bin:/usr/ucb:/usr/sbin
PWD=/etc
SHELL=/bin/ksh
TERM=dtterm
TERMINAL_EMULATOR=dtterm
TZ=MET
USER=user1
_=export
office=/export/home/user1/office
$
```

Setting Shell Variables

A variable is set and a value is assigned with the following syntax:

```
var=value

VAR=value
```

There is no space on either side of the equal sign (=); for example:

```
$ private=/export/home/user1/private
$ set | grep private
private=/export/home/user1/private
$ cd $private; pwd
/export/home/user1/private
```

To make the value of a variable known to a subshell, use the following command syntax:

```
export VAR=value
```

For example:

```
$ export office=/export/home/user1/office
$ echo $office
/export/home/user1/office
```

Unsetting Shell Variables

The values stored in shell variables can be deleted with the following command syntax:

```
unset VAR
```

or

```
unset var
```

For example, to `unset` the variable `private`, type the following commands:

```
$ unset private
$ echo $private

$
```

The `echo` command outputs a blank line, as shown previously.

Variables Set by the Shell on Login

Table 11-3 shows a list of variables that are assigned default values by the shell on login.

Table 11-3 Variables Set by the Shell on Login

Variable	Meaning
`EDITOR`	Defines the default editor for the shell.
`FCEDIT`	Defines the editor for the `fc` command. Used with the history mechanism for editing previously executed commands.
`HOME`	Sets the directory that `cd` changes to when no argument is supplied on the command line.
`LOGNAME`	Sets the login name of the user.
`PATH`	Specifies colon-delimited list of directories to be searched when the shell needs to find a command to be executed.
`PS1`	Specifies the primary Korn shell prompt: `$.`
`PS2`	Specifies the secondary command prompt, normally: `>.`
`SHELL`	Specifies the name of the shell (that is, `/bin/ksh`).

Note – The values presented for the `PS1` and `PS2` prompts are defaults and can be changed or customized by the user.

Customizing Korn Shell Variables

The following sections describe how to customize Korn shell variables.

The PS1 Prompt Variable

The shell prompt string is stored in the shell variable PS1, and you can customize it according to your preferences.

```
$ PS1="$LOGNAME@`uname -n`$ "
user1@host1
$
```

In this example, the prompt displays the login name of the user and the system's *hostname*.

The *username* is read from the variable LOGNAME, and the *hostname* comes from the output of the uname -n command.

This prompt displays the correct information even when the user logs on to different hosts.

The second example shows how the value of another variable is used for prompt definition. *(Be aware that your shell prompt will change from "$" to "I Like UNIX >").*

```
$ ILU="I Like UNIX"
$ PS1="$ILU > "
I Like UNIX > echo $ILU
I Like UNIX
I Like UNIX >
```

Note – To have this new shell prompt appear in every shell, it must be included in the user's Korn shell initialization file, usually named .kshrc.

The PATH *Variable*

The PATH variable contains a list of directory path names, separated by colons. When you invoke a command from the command line, the shell searches these directories from left to right to locate the command to be executed.

The first command found is the one that will be executed by the shell.

If the shell does not find the command in any of the listed directories, it displays the following error message:

```
ksh: command_name:  not found
```

The shell restricts its search to only those directories specified in PATH. So the command the shell could not find might reside in a directory that has not been specified in the PATH variable.

When the shell is unable to execute a command because it is not found, the user can type the absolute path name of the command; for example:

```
$ /usr/bin/id
```

If the command is then successfully executed by the shell, the user should check the PATH variable to ensure the directory exists in the search path and, if so, check that it has been entered correctly; for example:

```
$ echo $PATH
/usr/dt/bin:/usr/openwin/bin:/usr/bin:/usr/ucb
$
```

Extending the PATH *Variable*

In the following example, the PATH variable is extended to include the home directory of the user.

```
$ echo $PATH
/usr/dt/bin:/usr/openwin/bin:/usr/bin:/usr/ucb
$
$ PATH=$PATH:~
$
$ echo $PATH
/usr/dt/bin:/usr/openwin/bin:/usr/bin:/usr/ucb:/export/home/user1
$
```

The PATH variable automatically passes the value to subshells.

Korn Shell Metacharacters

Metacharacters, which have special meaning to the shell, are a powerful feature of any shell. However, there are times when you need to instruct the shell to mask, or ignore the special meaning of metacharacters.

This is accomplished through a process called *quoting*.

The special characters used by the shell for quoting include the backslash (\),single quote (’), and the double quote (").

Quoting With the Backslash

Placing a backslash (\) in front of a metacharacter prevents the shell from interpreting it as a metacharacter.

For example, a file was created with an asterisk (*) as its name. To delete this file, the asterisk (*) must be masked by a backslash (\) character.

```
$ ls
dat1  dat2  dat3  dir1  dir2  *
$ rm \*
```

Quoting With Single Quotes and Double Quotes

Placing special characters between quote characters prevents the shell from interpreting them as metacharacters.

Unlike the backslash, which only changes the behavior of the character immediately following it, the use of single quotes and double quotes allows you to change the behavior of all special characters in the given string.

- Single (forward) quotes (’ ’) – Tell the shell to ignore all enclosed metacharacters.
- Double quotes (" ") – Tell the shell to ignore most of the enclosed metacharacters.

These three metacharacters retain their special meaning to the shell when placed inside double quotes.

- Backslashes (\)
- Dollar signs ($)
- Back quotes (‘ ‘)

Basic Examples of Quoting

```
$ echo ’$SHELL’
$SHELL
$ echo "$SHELL"
/bin/ksh
$ echo "\$SHELL"
$SHELL
```

Command Substitution

Any UNIX command that is placed between back quotes is executed by the shell and its output displayed; for example:

```
$ echo date
date
$ echo ‘date‘
Tue May 2 14:10:05 MDT 2000
$
```

Using back quotes for command substitution is common among all the UNIX shells; however, the Korn shell provides an alternative method for command substitution.

```
$ echo pwd
pwd
$ echo $(pwd)
/export/home/user1
```

The History Mechanism

The Korn shell keeps a history of recently entered commands. This history mechanism enables users to view, repeat, or modify their previously executed commands.

Note – Command history is shared between all Korn shells.

The `history` Command

The Korn shell stores command history in a file specified by the `HISTFILE` variable. The default file is `$HOME/.sh_history`.

You can specify the number of commands to be stored using the `HISTSIZE` variable. If the variable is not set, the most recent 128 commands are stored.

You can edit previously executed commands and rerun these commands using a Korn shell in-line editor.

To set the `EDITOR` variable to an in-line editor, such as `vi` for the `history` command, use one of the following commands:

```
$ set -o vi
or
$ export EDITOR=/bin/vi
or
$ export VISUAL=/bin/vi
```

By default, the `history` command displays the last 16 commands to standard output; for example:

```
$ history
...
157    date
158    cd /etc
159    touch dat1 dat2
160    ps -ef
161    history
```

The numbers on the left are command numbers, which you can use to instruct the shell to re-execute a particular command line.

To display the command history without line numbers, execute the following:

```
$ history -n
```

To display this command and the four commands preceding it, execute the following:

```
$ history -4
```

To display the history list in reverse order, execute the following:

```
$ history -r
```

To display the most recent `cd` command to the most recent `ls` command, execute the following:

```
$ history cd ls
```

The `r` *Command*

The `r` command is a simple Korn shell command that allows you to repeat a command; for example:

```
$ cal
May 2000
 S  M Tu  W Th  F  S
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31


$ r
May 2000
 S  M Tu  W Th  F  S
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
```

You can use the r command to re-execute a command by its number. For example, the shell reruns 160 in the command history.

```
$ r 160
```

You can use the r command to re-execute a command beginning with a particular character, or string of characters.

The following command instructs the shell to rerun the most recent occurrence of a command that beings with the letter "c."

```
$ r c
cd /etc
```

This example has the shell rerun the most recent occurrence of the ps command.

```
$ r ps
ps -ef
```

You can use the r command to repeat a previous command, perform a simple edit, and execute the modified command; for example:

```
$ history
...
157    date
158    cd /etc
159    ps -ef
$
$ r c
cd /etc
$ r etc=tmp
cd /tmp
```

In this example, the r command repeated the most recent occurrence of a command beginning with the letter "c." It then replaced etc with tmp and executed the modified command.

Using `vi` *Commands to Edit a Previously Executed Command*

You can access a command in the history, edit the command with the `vi` editor, and execute the modified command following these steps:

1. Verify that the built-in `vi` editor has been enabled.

```
$ set -o | grep '^vi'
vi                  on
```

2. Type the `history` command to view the command history list.

```
$ history
```

3. Press the Escape key to access the command history list.

   Use the following cursor movement keys to scroll up and down the command history.

   - `k` – Move the cursor up one line at a time
   - `j` – Move the cursor down one line at a time

   Use the next set of cursor movement keys to scroll to the left and to the right on the command line.

   - `l` – Move the cursor to the right
   - `h` – Move the cursor to the left

Note – You cannot use the arrows keys to move the cursor. The `h`, `j`, `k`, and `l` keys are used for cursor movement within the command history.

---

4. Use `vi` commands to edit any previously executed command.
5. To execute a modified command, press the Return key.

The Korn Shell Alias Utility

An alias is a shorthand notation in the Korn shell to allow you to customize and abbreviate UNIX commands. An alias is defined by using the `alias` command.

Command Format

`alias` *`name=command_string`*

For example:

```
$ alias dir='ls -lF'
```

The shell keeps a list of aliases that it searches when a command is entered. If the first word on the command line is an alias, the shell replaces that word with the text of the alias. When an alias is created, the following rules apply:

- There is no space on either side of the equal sign.
- The command string must be quoted if it includes any options, metacharacters, or spaces.
- Each command in a single alias is separated with a semicolon.

Predefined Korn Shell Aliases

The Korn shell contains several predefined aliases, which you view using the `alias` command. Any new user-defined aliases are also displayed.

```
$ alias
autoload='typeset -fu'
command='command '
functions='typeset -f'
history='fc -l'
integer='typeset -i'
local=typeset
nohup='nohup '
r='fc -e -'
stop='kill -STOP'
suspend='kill -STOP $$'
```

Table 11-4 lists of alias definitions.

Table 11-4 Alias Definitions

Alias	Value	Definition
`autoload`	`typeset -fu`	Defines how to automatically load a function
`functions`	`typeset -f`	Displays a list of functions
`history`	`fc -l`	Lists commands from the history file
`integer`	`typeset -i`	Displays any variable with an integer value
`local`	`typeset`	Sets a local attribute for shell variables and functions
`nohup`	`nohup`	Keeps jobs running even if you log out
`r`	`fc -e -`	Executes the previous command again
`stop`	`kill -STOP`	Suspends a job
`suspend`	`kill -STOP $$`	Suspends the current shell

User-defined Aliases

Aliases are commonly used to abbreviate or customize frequently used commands; for example:

```
$ alias h=history
$
$ h
278    cat /etc/passwd
279    pwd
280    cp /etc/passwd /tmp
281    ls ~
282    alias h=history
283    h
```

Using the commands `rm`, `cp`, and `mv` can inadvertently result in loss of data. As a precaution, you can alias these commands with the interactive option.

For example:

```
$ alias rm='rm -i'
$ rm dat1
rm: remove  dat1: (yes/no)? no
$
```

By creating a `cp -i` and `mv -i` alias, the shell asks you before overwriting any existing file.

You can deactivate an alias temporarily by placing a backslash (\) in front of the alias on the command line.

The backslash prevents the shell from looking in the alias list, causing it to execute the original `rm` command; for example:

```
$ rm file1
rm: remove file1 (yes/no)? no
$
$ rm \file1
$ ls file1
file1: No such file or directory
```

Command Sequences

You can group several commands together under one alias name. Individual commands are separated by semicolons; for example:

```
$ alias info='uname -a; id; date'
$ info
SunOS host1 5.8 Generic sun4u sparc SUNW,Ultra-5_10
uid=102(user2) gid=10(staff)
Fri Jun 30 15:22:47 MST 2000
$
```

In the next example, an alias is created using a pipe (|) to direct the output of the `ls -l` command to the `more` command. When the new alias is invoked, you get a long directory listing, one screenful at a time; for example:

```
$ alias ll='ls -l | more'
$ cd /usr
$ ll
total 136
drwxrwxr-x   2 root     bin         1024 May 13 18:33 4lib
drwx------   8 root     bin          512 May 13 18:14 aset
drwxrwxr-x   2 root     bin         7168 May 13 18:23 bin
drwxr-xr-x   4 bin      bin          512 May 13 18:13 ccs
drwxrwxr-x   5 root     bin          512 May 13 18:28 demo
--More--
```

Removing Aliases

Use the `unalias` command to remove aliases from the alias list.

Command Format

`unalias` *`name`*

For example:

```
$ unalias h
$ h
ksh: h:  not found
```

Note – To pass the new aliases to every shell invoked, place it in your Korn shell initialization file, usually called `.kshrc`.

Korn Shell Functions

Functions are a powerful feature of shell programming used to construct customized commands.

A function is a group of UNIX commands organized as separate routines. Using a function involves two steps:

- Defining the function
- Invoking the function

Defining a Function

A function is defined using the general format:

```
function name { command; . . . command; }
```

Note – A space must appear after the first brace and before the closing brace.

Some Function Examples

The following example creates a function called `num` to execute the `who` command, directing the output to the `wc` command to display the total number of users currently logged on the system.

```
$ function num { who | wc -l; }
$ num
     9
```

The following example creates a function called `list` to execute the `ls` command, directing the output to the `wc` command to display the total number of subdirectories and files in the current directory.

```
$ function list { ls -al | wc -l; }
$ list
     34
```

Note – If a command name is defined as a function and an alias, the alias takes precedence.

To display a list of all functions, use the following command:

```
$typeset -f
function list
{
ls -al | wc -l; }
function num
{
who | wc -l; }
```

To display just the function names, use the following command:

```
$ typeset +f
list
num
```

Configuring the Korn Shell Environment

You can customize your shell environments by placing two initialization files within the home directory.

- `~/.profile`
- `~/.kshrc`

The Korn shell looks in your home directory for an initialization file called `.profile` and an environment file conventionally called `.kshrc`. After executing commands found in these files, the shell prompt ($) appears on the screen and the Korn shell waits for commands.

The `~/.profile` File

The `.profile` file is a user-defined initialization file that is executed once at login by the login shell and is found in your home directory. It gives you the ability to customize and modify your working environment.

Shell variables and terminal settings are normally set in this file, and if an application is to be initiated, it can be started here.

If the `.profile` file contains a special variable called `ENV`, the file name assigned to that variable is executed next.

Note – The `/etc/profile` is a system-wide file maintained by the system administrator to set up tasks executed by the Korn shell for every user who logs in.

The `~/.kshrc` File

By convention, this `ENV` file is often called `.kshrc`. It contains Korn shell variables and aliases. The `ENV` file is executed every time a `ksh` subshell is started.

Note – You can choose and set any name for the `ENV` file.

The following items are normally set in the `~/.kshrc` file:

- Shell prompt definitions (`PS1`, `PS2`)
- Alias definitions
- Korn shell functions
- History variables
- Korn shell options (`set -o` *`option`*)

Rereading Initialization Files

When you make changes to your individual initialization files, those changes take effect when you log in the next time.

However, if changes have been made and you want those changes to take effect immediately, the `.profile` and `.kshrc` files can be sourced using the `.` (dot) command.

```
$ .  ~/.profile
$
$ .  ~/.kshrc
```

Note – Running the two commands shown previously only updates the current shell.

Configuring the CDE Environment

There is a configuration file that CDE reads upon execution. Modifications to this configuration file allow users to customize their desktop environment to some extent.

The ~/.dtprofile File

If you work in the CDE environment, there is another initialization file called `.dtprofile`. It resides in your home directory and determines generic and customized settings for the CDE. The `.dtprofile` file means your preferences for variable settings can overwrite default settings.

There is a standard, system-wide `.dtprofile` file contained in the `/usr/dt/config` directory in the file called `.dtprofile`. The `dtsourceprofile=true` variable is located at the bottom of this file. This standard file is used by CDE to generate a `.dtprofile` file for placement in a user's home directory the first time the user logs in to CDE.

Whenever you log in to CDE, the `.dtprofile` is read, and then the Korn shell reads the user's `.profile` file and the `.kshrc` file.

The `.profile` and `.kshrc` files are read again whenever you open a console window. The `.kshrc` file is always read whenever the user opens a terminal window in CDE.

Exercise: Modifying the Korn Shell

Exercise objective – In this exercise, you use the commands learned in this module to redirect standard output and standard error, create command aliases and functions, and use the `history` command for specific tasks.

Tasks

Complete the following steps, and write the commands you would use to perform each task in the space provided.

1. Change to your home directory and list the contents of the current directory on the same command line. What commands are used?

   _____

2. List the contents of the current directory in long format and direct standard output (`stdout`) to a file named `dirlist`. Use the `cat` command to display the contents of `dirlist`.

   _____

3. Use `cat` to read the file `fruit2`, and append standard output to a file called `itemlist`.

   _____

4. Turn `noclobber` on, and verify that you have turned it on with the `set` command.

   _____

5. Now list the contents of `dir1`, and redirect standard output to the file called `itemlist`. What message was displayed?

   _____

6. This time, overwrite the file `itemlist` with the contents of the `dir1` directory. Use the `cat` command to display the contents of `itemlist`.

   _____

7. Execute the `date` command, and append standard output to the file `itemlist`.

_____

8. Execute the `who` command, and redirect standard output to a file named `login.list`.

_____

9. Use the `ps -ef` command and the `wc -l` command to display the total number of processes running on your system, and append that information to the file `login.list`.

_____

10. Execute the command `ls -l fruit fruit1 fruit2`, and redirect standard output and standard error to the file named `check`. Use the `cat` command to display the contents of `check`.

_____

11. Now execute that command in Step 10 again, but this time redirect standard output to a file named `file.list`, and redirect standard error to the file named `error.list`. Use the `cat` command to display the contents of both new files.

_____

12. Display all predefined aliases.

_____

13. Create an alias named `cls` that clears the terminal screen.

_____

14. Create an alias named `dir` that displays a long listing of all the files and directories in the current directory.

_____

15. Create an alias named `h` that lists your command history.

_____

16. Execute the following commands:

```
$ date
$ who
$ ls
```

17. Display the history list in reverse order.

_____

18. Re-execute the `who` command.

_____

19. Unalias the `history` command and the `clear` command.

_____

20. Display all defined functions.

_____

21. Create a function called `data` that clears the terminal screen; displays the date and time, who is logged into your system, and the path of the current working directory; and lists the current working directory in long format.

_____

22. Use the `vi` editor to create and edit the `.profile` in your home directory, and add the following line entries:

```
$ vi ~/.profile
```

Add:

```
PATH=/bin:/usr/bin:/usr/ucb
ENV=$HOME/.kshrc
PS1="$LOGNAME@$(uname -n) $ "
export PATH ENV PS1
```

23. Use the `vi` editor to create and edit the `.kshrc` file in your home directory, and add the following line entries:

```
$ vi ~/.kshrc
```

Add:

```
set -o vi
alias h='history'
alias cls='clear'
alias lf='pwd ; ls -lF'
```

24. Execute the following commands:

```
$ . ~/.profile
```

```
$ . ~/.kshrc
```

25. Test your new aliases and functions.

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercises.

- Experiences
- Interpretations
- Conclusions
- Applications

Task Solutions

Complete the following steps and write the commands you would use to perform each task in the space provided.

1. Change to your home directory and list the contents of the current directory on the same command line. What commands are used?

   ```
   $ cd ; ls
   ```

2. List the contents of the current directory in long format and direct the standard output (`stdout`) to a file named `dirlist`. Use the `cat` command to display the contents of `dirlist`.

   ```
   $ ls -l > dirlist
   $ cat dirlist
   ```

3. Use `cat` to read the file `fruit2`, and append standard output to a file called `itemlist`.

   ```
   $ cat fruit2 >> itemlist
   ```

4. Turn `noclobber` on, and verify that you have turned it on with the `set` command.

   ```
   $ set -o noclobber
   $ set -o | more
   ```

5. Now list the contents of `dir1`, and redirect standard output to the file called `itemlist`. What message was displayed?

   ```
   $ ls dir1 > itemlist
   ksh: itemlist: file already exists
   ```

6. This time, overwrite the file `itemlist` with the contents of the `dir1` directory. Use the `cat` command to display the contents of `itemlist`.

   ```
   $ ls dir1 >| itemlist
   ```

7. Execute the `date` command, and append the standard output to the file `itemlist`.

   ```
   $ date >> itemlist
   ```

8. Execute the `who` command, and redirect the standard output to a file called `login.list`.

   ```
   $ who > login.list
   ```

9. Use the `ps -ef` command and the `wc -l` command to display the total number of processes running on your system, and append that information to the file `login.list`.

```
$ ps -ef | wc -l >> login.list
```

10. Execute the command `ls -l fruit fruit1 fruit2`, and redirect standard output and standard error to the file named `check`. Use the `cat` command to display the contents of `check`.

```
$ ls -l fruit fruit1 fruit2 > check 2>&1
$ cat check
```

11. Now execute that command in Step 10 again, but this time redirect standard output to a file named `file.list`, and redirect standard error to the file named `error.list`. Use the `cat` command to display the contents of both new files.

```
$ ls -l fruit fruit1 fruit2 >| file.list 2>| error.list
```

12. Display all predefined aliases.

```
$ alias
```

13. Create an alias named `cls` that clears the terminal screen.

```
$ alias cls=clear
```

14. Create an alias named `dir` that displays a long listing of all the files and directories in the current directory.

```
$ alias dir='ls -l'
```

15. Create an alias named `h` that lists your command history.

```
$ alias h=history
```

16. Execute the following commands:

```
$ date
$ who
$ ls
```

17. Display the history list in reverse order.

```
$ history -r
```
or
```
$ h -r
```

18. Re-execute the `who` command.

```
$ history
$ r number_of_the_who_command
or
$ r w
or
$ r wh
```

19. Unalias the `history` command and the `clear` command.

```
$ unalias h

$ unalias cls
```

20. Display all defined functions.

```
$ typeset -f
```

21. Create a function called `data` that clears the terminal screen; displays the date and time, who is logged into your system, and the path of the current working directory; and lists the current working directory in a long format.

```
$ function data { clear; date; who; pwd; ls -l; }
```

22. Use the `vi` editor to create and edit the `.profile` in your home directory, and add the following line entries:

```
$ vi ~/.profile
```

Add:

```
PATH=/bin:/usr/bin:/usr/ucb
ENV=$HOME/.kshrc
PS1="$LOGNAME@$(uname -n) $ "
export PATH ENV PS1
```

23. Use the `vi` editor to create and edit the `.kshrc` file in your home directory, and add the following line entries:

```
$ vi ~/.kshrc
```

Add:

```
set -o vi
alias h='history'
alias cls='clear'
alias lf='pwd ; ls -lF'
```

24. Execute the following commands:

```
$ .  ~/.profile
```

```
$ .  ~/.kshrc
```

25. Test your new aliases and functions.

Check Your Progress

Before continuing on to the next module, check that you are able to accomplish or answer the following:

- ❑ Describe the functions of the Korn shell as a command interpreter
- ❑ Demonstrate the use of quoting to mask the special meaning of metacharacters by the Korn shell
- ❑ Define the terms standard input, standard output, and standard error
- ❑ Use metacharacters to redirect standard input, standard output, and standard error
- ❑ Connect two or more commands together using the pipe feature
- ❑ Implement the file name completion mechanism in the Korn shell
- ❑ Use commands to view, set, and unset shell variables
- ❑ Invoke the `history` mechanism to repeat or edit previously executed commands
- ❑ Use the `alias` utility to customize and abbreviate UNIX commands
- ❑ Create Korn shell functions to construct customized commands
- ❑ Define the Korn shell initialization files used to customize a user's environment

Introducing sed *and* awk 12

Objectives

Upon completion of this module, you should be able to:

- Use the stream editor (sed) to edit the contents of a text file from the command line and send the results to standard output
- Issue sed commands to delete lines, print lines containing a pattern, add text to lines, or change characters using regular expression metacharacters
- Use awk to scan text files or standard input to display specific data, change data format, and add text to existing data

The Stream Editor

Use the sed program (or stream editor) to edit data in files without opening them in an interactive editor, such as vi. It allows you to specify edits, or modifications, to a file from the command line and send the output to the screen by default. This enables you to do repetitive tasks quickly.

The sed editor does not change the content of the source file. To save the output, you need to redirect it to a new file.

This editor is best used to make the same changes across multiple files quickly.

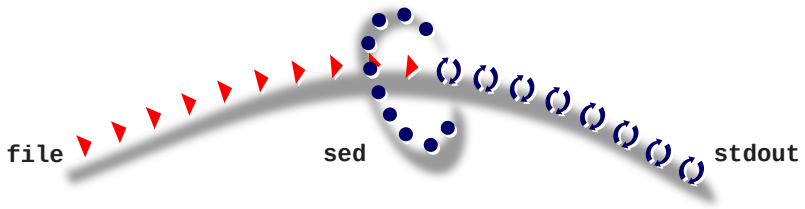


Figure 12-1 The sed Command

Command Format

```
sed [-options] [address] command file... [>newfile]
```

Note – The sed command can also be used in a pipe; for example:
`ls -l | sed '/2/d'`

Options

Options are used to control the behavior of sed. The most commonly used options are:

- The -e option allows multiple edits on the same command line
- The -n option suppress the default output

Review of Regular Expressions

Similar to the `grep` command, `sed` uses a number of special metacharacters to control pattern searching.

Table 12-1 describes some useful regular expression metacharacters with `sed`.

Table 12-1 Regular Expression Metacharacters

Metacharacter	Purpose	Sample	Result
`^`	Beginning of line anchor	`'^pattern'`	Matches all lines beginning with “pattern”
`$`	End of line anchor	`'pattern$'`	Matches all lines ending with “pattern”
`.`	Matches one character	`'p.....n'`	Matches lines containing a “p,” followed by five characters, followed by an “n”
`[ ]`	Matches one character in the pattern	`'[Pp]attern'`	Matches lines containing “Pattern” or “pattern”
`*`	Matches the preceding item zero or more times	`'[a-z]*'`	Matches lowercase alphanumeric characters
`[^]`	Matches one character not in the pattern	`'[^a-m]attern'`	Matches lines not containing “a” through “m,” followed by “attern”

Using the Stream Editor

The sed command uses regular expressions as parameters for the functions as recipes for the edits that sed makes.

Deleting Lines With the d Command

The following examples show how sed searches for lines containing a *pattern* and removes those lines.

- Search for a pattern within a file, and remove all lines containing that pattern. The results display to the screen, and the file is untouched.

  ```
  sed  '/pattern/d' filename
  ```

- Delete all lines containing the pattern root from the file /etc/group. The results display to the screen, and the file is untouched.

  ```
  $ sed '/root/d' /etc/group
  ```

- Delete all lines containing the pattern "3" in the output of the ls command.

  ```
  $ ls -l | sed  '/3/d'
  ```

- Search for specific lines within a file, and remove those lines. The results display to the screen, and the file is untouched.

  ```
  sed  '#,#d' filename    (removes line # to line #)

  sed  '#d' filename      (removes only line #)

  sed  '#,$d' filename    (removes line # to the last line)

  sed  '$d' filename      (removes only the last line)
  ```

  For example, direct the output of the ls command to sed and remove Line 5 to the last line in the output, placing the results in a new file.

  ```
  $ ls -l | sed  '5,$d' > newfile
  $ cat newfile
  [output not shown]
  ```

Printing Lines With the p *Command*

The following example shows how `sed` prints all lines to standard output by default, duplicating lines containing *`pattern`* in addition to all the other lines in a file.

```
$ sed '/Dante/p' dante
[output not shown]
```

By default, `sed` prints all lines to standard output.

If the pattern `Dante` is found, `sed` prints duplicate lines of output, in addition to all the other lines in the file.

To suppress this default action use the `-n` option with the `p` command; for example:

```
$ sed -n '/Dante/p' dante
[output not shown]
```

In this example, only the lines containing the pattern `Dante` are printed.

Placing Characters at the End of Each Line

The following illustrates how to append characters to the end of lines.

To append the character string `EOL` at the end of every line and display the results to standard output, execute the following:

```
$ ls -l | sed 's/$/ EOL/'
[output not shown]
```

Changing Spaces to Colons in Data

To search for at least one or more spaces and replace all spaces that are found with a single colon character, execute the following:

```
$ ls -l | sed 's/  */:/g'
[output not shown]
```

There are two spaces following the **`'s/`** in the preceding command line.

Multiple Edits With sed

The following shows how sed can perform multiple edits on the same command line.

```
$ sed -e 's/Dante/DANTE/g' -e 's/poet/POET/g' dante
[output not shown]
```

In the previous example, a new file is created, sed is then given two edit commands using the -e option.

The first edit replaces all occurrences of Dante to DANTE.

The second edit replaces the word poet to POET.

The results are displayed to standard output.

Text Processing Using the awk *Command*

The awk command is a flexible text processor used for manipulating columns of data and generating reports. It scans a file (or input) line-by-line, from first to last, searching for lines that match a specified pattern and performing selected actions on those lines.

Note – The awk command derived its name from the first initial of the last names of those who authored this command; Alfred Aho, Peter Weinberger, and Brian Kernighan.

Some basic uses of awk include making format changes to data, reordering columns, and adding to existing text.

Command Format

```
awk '{ action }' filename
```

Basic awk *Command Format*

The basic format of this command consists of the awk command, the instructions enclosed in quotes and curly braces, and the name of the input file.

If an input file is not specified, then standard input is used, for example, the keyboard.

The following is a basic awk command. The output of the `ls -l` command is piped to awk. For each line received by awk, the `print` action is executed, which prints the output to the screen.

```
$ ls -l | awk  '{print $0}'
[output not shown]
```

The results of this command are exactly the same output as the `ls -l` command. See Figure 12-2.

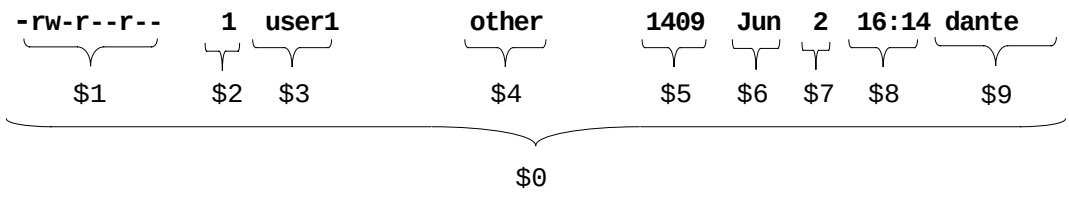


Figure 12-2 Results of the Basic awk Command

When awk reads in a line it automatically breaks the line into fields. Each field is assigned a variable name. Spaces or tabs are used as the default delimiter between fields.

The variable names assigned to fields are a dollar sign ($) followed by the number of the field, counting from left to right.

The variable name $1 represents the contents of Field 1. The variable name $2 represents the contents of Field 2, and so on. The entire line is represented by the variable name $0.

Using awk to Display Specific Data

To instruct awk to display specific data (for example, the file owner, file size, and file name), the fields' variable names are used with the action.

```
$ ls -l | awk '{print $3 $5 $9}'
user154120dante
user1368dante_1
user1176dat
user1512dir1
user1512dir2
user1512dir3
user1512dir4
user1235file1
user1105file2
user1218file3
user1137file4
user156fruit
user157fruit2
```

The standard output does not include a space between the three fields.

The following shows how to instruct awk to place spaces between the fields in the output by the use of a comma.

```
$ ls -l | awk '{print $3, $5, $9}'
user1 54120 dante
user1 368 dante_1
user1 176 dat
user1 512 dir1
user1 512 dir2
user1 512 dir3
user1 512 dir4
user1 235 file1
user1 105 file2
user1 218 file3
user1 137 file4
user1 35 file5
user1 56 fruit
user1 57 fruit2
user1 512 practice
user1 28738 tutor.vi
$
```

This example places a space between each field; however, the fields are not aligned.

To provide for exact alignment between fields, press the Tab key to embed a tabbed space between the double quotes.

```
$ ls -l | awk '{print $3 "\t" $5 "\t" $9}'
user1    54120      dante
user1      368      dante_1
user1      176      dat
user1      512      dir1
user1      512      dir2
user1      512      dir3
user1      512      dir4
user1      235      file1
user1      105      file2
user1      218      file3
user1      137      file4
user1       57      fruit2
user1      512      practice
user1    28738      tutor.vi
$
```

Using awk *to Change the Format of Data*

You can instruct the awk command to rearrange the fields for the purpose of changing the format of the data.

The following shows how to format the data to display the file name first, then file size, and then the file owner:

```
$ ls -l | awk '{print $9,$5,$3}'
dante 54120 user1
dante_1 368 user1
dat 176 user1
dir1 512 user1
dir2 512 user1
dir3 512 user1
dir4 512 user1
file1 235 user1
file2 105 user1
file3 218 user1
file4 137 user1
fruit 56 user1
fruit2 57 user1
practice 512 user1
tutor.vi 28738 user1
$
```

The following shows how to format the data to display the file owner, file name, and file creation or modification date. (To provide exact alignment between these fields, press the Tab key between the double quotes.)

```
$ ls -l | awk '{print $3 "    " $9 "    " $6,$7}'
user1    dante       Apr 16
user1    dante_1     Mar 22
user1    dat          May 2
user1    dir1         May 1
user1    dir2        Mar 22
user1    dir3        Mar 22
user1    dir4        Mar 22
user1    file1        May 1
user1    file2       Mar 22
user1    file3       Mar 22
user1    file4       Mar 22
user1    fruit       Mar 22
user1    fruit2      Mar 22
user1    practice    Mar 22
user1    tutor.vi    Mar 22
$
```

Using awk *to Add Text to Data*

You can instruct the awk command to rearrange the fields, as well as to add new text between fields.

```
$ ls -l | awk '{print $9,"is using",$5,"bytes"}'
file1 is using 405 bytes
file2 is using 66 bytes
file3 is using 66 bytes
file4 is using 66 bytes
$
```

When adding text on an awk command line, each text insert must be enclosed in double quotes, and all but the last text insert must be followed by a comma, as shown above.

Exercise: Using sed *and* awk

Exercise objective – In this exercise, you use `sed` to edit a file and the `awk` command to match patterns in files using regular expressions.

Tasks

Complete the following steps and write the commands you would use to perform each task in the space provided.

1. Use `sed` on the `file1` file, and remove all lines containing the word "Achievers." Let the output of `sed` display to the terminal screen.

   _____

2. Use the output of the `ls` command with `sed` to remove all lines containing the number 0. Redirect the output of `sed` to a new file called `new.file`.

   _____

3. Using `sed`, remove Line 2 to Line 7 from the `fruit` file.

   _____

4. Using `sed`, delete the last line of the `fruit` file.

   _____

5. Direct the output of the `ls` command to `sed`, and remove Line 5 to the last line in the output. Place the results in a new file called `results.file`.

   _____

6. Using `sed`, substitute all occurrences of the string "the" with the word "COOL" in the `dante` file.

   _____

7. Using `sed`, search for and delete the first eight characters of each line in the `file3` file.

_____

8. Using `sed`, append an asterisk (`*`) character to the end of each line in the `fruit2` file.

_____

9. Using `sed`, print the lines in `file1` to standard output, including any line containing the pattern "Achievers," and examine the command output.

_____

10. Using `sed`, print only the lines in `file1` that contain the pattern "Achievers."

_____

11. Direct the output of the `ls -l` command to `awk` to display the file owner, file size, date, time, and file name for each line.

_____

12. Reenter the same command from Step 11, and instruct `awk` to provide spaces between the five fields.

_____

13. Direct the output of the `ls -l` command to `awk` to rearrange fields to display the file size, file owner, and file name. Instruct `awk` to provide spaces between the fields.

_____

14. Direct the output of the `ls -l` command to `awk` to display the following three fields: file name, file owner, and file size. At the same time, instruct `awk` to add the following text between the fields:

    Add the text “belongs to” between the file name and the file owner fields.

    Add the text “and it is” between the file owner and the file size fields.

    Add the text “bytes in size” after the file size field.

    _____

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercises.

- Experiences
- Interpretations
- Conclusions
- Applications

Task Solutions

Complete the following steps and write the commands you would use to perform each task in the space provided.

1. Use `sed` on the `file1` file, and remove all lines containing the word "Achievers." Let the output of `sed` display to the terminal screen.

   ```
   $ sed '/Achievers/d' file1
   ```

2. Use the output of the `ls` command with `sed` to remove all lines containing the number 0. Redirect the output of `sed` to a new file called `new.file`.

   ```
   $ ls -l | sed '/0/d' > new.file
   ```

3. Using `sed`, remove Line 2 to Line 7 from the `fruit` file.

   ```
   $ sed '2,7d' fruit
   ```

4. Using `sed`, delete only the last line of the `fruit` file.

   ```
   $ sed  '$d' fruit
   ```

5. Direct the output of the `ls` command to `sed`, and remove Line 5 to the last line in the output. Place the results in a new file called `results.file`.

   ```
   $ ls | sed  '5,$d' > results.file
   ```

6. Using `sed`, substitute all occurrences of the string "the" with the word "COOL" in the `dante` file.

   ```
   $ sed 's/the/COOL/g' dante
   ```

7. Using `sed`, search for and delete the first eight characters of each line in the `file3` file.

   ```
   $ sed 's/^......//' file3
   ```

8. Using `sed`, append an asterisk (*) character to the end of each line in the `fruit2` file.

   ```
   $ sed 's/$/*/' fruit2
   ```

9. Using `sed`, print the lines in `file1` to standard output, including any line containing the pattern "Achievers," and examine the command output.

   ```
   $ sed '/Achievers/p' file1
   ```

10. Using `sed`, print only the lines in `file1` that contain the pattern "Achievers."

    ```
    $ sed -n '/Achievers/p' file1
    ```

11. Direct the output of the `ls -l` command to `awk` to display the file owner, file size, date, time, and file name for each line.

    ```
    $ ls -l | awk  '{print $3 $5 $6 $7 $8 $9}'
    ```

12. Reenter the same command from Step 11, and instruct `awk` to provide spaces between the five fields.

```
$ ls -l | awk '{print $3 "    " $5 "    " $6 "    " $7 "    " $9}'
```

13. Direct the output of the `ls -l` command to `awk` to rearrange fields to display the file size, file owner, and file name. Instruct `awk` to provide spaces between the fields.

```
$ ls -l | awk  '{print $5 "     " $3 "     " $9}'
```

14. Direct the output of the `ls -l` command to `awk` to display the following three fields: file name, file owner, and file size. At the same time instruct `awk` to add the following text between the fields.

    Add the text "belongs to" between the file name and the file owner fields.

    Add the text "and it is" between the file owner and the file size fields.

    Add the text "bytes in size" after the file size field.

```
$ ls -l | awk '{print $9,"belongs to",$3,"and it is",$5,"bytes in
size."}'
```

Check Your Progress

Before continuing on to the next module, check that you are able to accomplish or answer the following:

- Use the stream editor (`sed`) to edit the contents of a text file from the command line and send the results to standard output
- Issue `sed` commands to delete lines, print lines containing a pattern, add text to lines, or change characters using regular expression metacharacters
- Use `awk` to scan text files or standard input to display specific data, change data format, and add text to existing data

Reading Shell Scripts 13

Objectives

Upon completion of this module, you should be able to:

- Identify which shell program interprets the lines of a shell script
- Explain how command-line arguments are passed to a shell script with special variables called positional parameters
- Demonstrate the use of two conditional commands: `if` and `case`, as well as the `test` command
- Interpret the contents of a simple Bourne shell administration script

Additional Resources

Additional resources – The following reference provides additional details on the topics discussed in this module:

- *Common Desktop Environment: Desktop Korn Shell User's Guide*, Part Number 806-2912

The Basics of Shell Scripts

A shell script is an ASCII file that contains a sequence of commands and comments.

Comments are text used to document what the script does and describe what the lines within the script are supposed to do when it is executed. Comments are preceded by a pound (#) symbol and can be on a line by themselves or on the same line following a command.

Determining the Type of Shell Script

The top line of a script identifies the shell program that executes the lines in the script. For example, for a Bourne shell script, the first line would read:

```
#!/bin/sh
```

The `#!` is used by the parent shell to identify the program that interprets the lines in the script.

Note – The first line in a Korn shell script would read `#!/bin/ksh`. The first line in a C shell script would read `#!/bin/csh`.

However, not all shell scripts use the top line to identify the shell program.

If a shell script does not begin with the characters "#!", then the parent shell is used to execute it.

Other shell scripts might have no comment at all on the top line. If this is the case, the default shell is used to execute the script.

Caution – When naming shell scripts, do not use the word `script` or `test`. These are names of actual commands in the Solaris Operating Environment.

Creating a Basic Shell Script

The following steps demonstrate how to build a basic program.

1. With the `vi` editor, create a file called `myfile1`, and enter each command followed by a tab and pound (#) symbol to define it as a comment.

2. Make the file executable, and run the new script by typing its name on the command line.

```
$ vi myfile1
who    # To view who is logged on the system
date   # To view the current date and time
ls -l  # To view files in current directory
```

Save the file, and quit the editor.

Apply the `chmod` command to make the script executable.

```
$ chmod 755 myfile1
```

Execute the new script.

```
$ myfile1
user1   console      Aug 23 09:40    (:0)
user1   pts/3        Aug 24 14:50    (:0.0)
user1   pts/5        Aug 23 09:41    (:0.0)
user1   pts/4        Aug 23 09:41    (:0.0)
Fri Aug 25 13:33:28 MDT 2000
total 256
drwx--x--x   3 user1 staff         96 Aug 25 10:38 Reports
-rwx--x--x   1 user1 staff          0 May 31 16:45 brands
-rwx--x--x   1 user1 staff       1320 May 31 16:44 dante
-rwx--x--x   1 user1 staff        368 May 31 16:45 dante_1
drwx--x--x   6 user1 staff       8192 Aug 25 11:48 dir1
drwx--x--x   4 user1 staff         96 May 31 16:45 dir2
drwx--x--x   3 user1 staff         96 May 31 16:45 dir3
drwx--x--x   3 user1 staff         96 May 31 16:45 dir4
-rwx--x--x   1 user1 staff        218 Aug 25 11:40 feathers
-rwx--x--x   1 user1 staff        218 Aug 25 11:41 feathers_6
-rwx--x--x   1 user1 staff          0 May 31 16:45 file.1
<output omitted>
$
```

Bourne Shell Programming

The standard administration scripts used to manage the Solaris Operating Environment are Bourne shell scripts.

To successfully control or modify the behavior of the operating environment, system administrators must be able to read, modify, and customize the contents of these shell scripts.

Note – Currently there are two comprehensive Sun Education courses on shell programming available: SL-120: *Shell Programming for System Programmers* and SA-245: *Shell Programming for System Administrators*.

In general, all Bourne shell scripts consist of a combination of UNIX commands, Bourne shell built-in commands, programming constructs, and comments.

Bourne Shell Scripts

To be able to read and, more importantly, comprehend the contents of a basic shell script, you must be able to:

- Understand how arguments are passed to a script through the use of special built-in variables called positional parameters
- Identify and analyze simple conditional constructs and flow control

Special Built-in Shell Variables

Shells have special built-in variables, which are described in the next section.

Positional Parameters

Special built-in shell variables, called *positional parameters*, are used to pass arguments from the command line into a shell script.

On the command line, each word, separated by a space, that follows the shell script file name is called an argument. These arguments are referenced in the shell script with positional parameters.

Command Format

```
scriptname argument1 argument2 argument3 ...
```

When the script is executed, the shell automatically stores the first argument on the command line in the positional parameter `$1`, the second argument in positional parameter `$2`, the third argument in `$3`, and so on.

Resetting Positional Parameters

Use the `set` command to reset the positional parameters based on the arguments.

The following example illustrates how the `set` command and positional parameters work within a shell script.

First, execute the `who` command with the `-m` option, and view the output.

```
$ who -m
user1      pts/5        Mar 13 11:43    (host1)
$
```

Now, create a shell script with the `vi` command, and name it `myfile2`.

```
$ vi myfile2
#!/bin/sh
set 'who -m'
echo Here are the positional variables that have been set:
echo The first is: $1
echo The second is: $2
echo The third is: $3
echo The fourth is: $4
echo The fifth is: $5
echo The sixth is: $6
echo This script is: $0
```

Save the file, and quit the editor.

Apply the `chmod` command to make the script executable.

```
$ chmod 755 myfile2
$
```

Execute the new script.

```
$ ./myfile2
Here are the positional variables that have been set:
The first is: user1
The second is: pts/5
The third is: Mar
The fourth is: 13
The fifth is: 11:43
The sixth is: (host1)
This script is: ./myfile2
$
```

The Bourne shell allows more than nine positional parameters. However, only the first nine, $1 through $9 can be referenced directly. The shell script file name is explicitly referenced in the $0 variable.

Table 13-1 Positional Parameters

Positional Parameter	Meaning
$0	References the name of the current shell script
$1 - $9	References up to nine positional parameters

Two other variables associated with positional parameters are $# and $*.

- $# – Returns the total number of command line arguments
- $@ – Displays a list of the values of all the positional parameters

The following example demonstrates how these two built-in shell variables function within a shell script.

Create the shell script named myfile3.

```
$ vi myfile3
#!/bin/sh
echo The script name is: $0
echo
echo The first argument passed is: $1
echo The second argument passed is: $2
echo
echo The highest numbered parameter is: $#
echo The parameters passed to the script are: $*
```

Save the file, and quit the editor.

Apply the chmod command to make the script executable.

```
$ chmod 755 myfile3
$
```

Execute the new script with the two arguments: `stop` and `start`.

```
$ ./myfile3 start stop
The script name is: myfile3

The first argument passed is: start
The second argument passed is: stop

The highest numbered parameter is: 2
The parameters passed to the script are: start stop
$
```

Conditional Commands and Flow Control

Conditional commands allow you to perform some tasks based on whether a condition succeeds or fails.

The simplest form of a conditional command is the `if` command.

The `if` command enables you to test a condition and then change the flow of a shell script's execution based on the results of the test.

Command Format

```
if expression
then
        command
fi
```

For example, create the shell script named `my-script1` to determine if someone is logged on the system.

```
$ vi my-script1
#!/bin/sh

user="$1"

if who | grep "$user"
then
        echo "$user is logged on"
fi
```

Save the file, and quit the editor.

Apply the `chmod` command to make this script executable.

```
$ chmod 755 my-script1
```

Run the `who` command to determine who is currently logged on.

```
$ who
user1    console       Aug 17 15:37    (:0)
user1    pts/3         Aug 17 15:37    (:0.0)
```

Select a user name from the who output and use the user name as an argument on the command line; for example:

```
$ ./my-script1 user1
user1    console      Aug 17 15:37    (:0)
user1    pts/3        Aug 17 15:37    (:0.0)
user1 is logged on
$
```

Exit Status

Whenever a program completes execution, within a script or on the command line, it returns an exit status back to the shell. An exit status is an integer between 0 and 255.

Following the generic command format above, the command following the if construct is executed and its *exit status* is returned.

If the exit status is 0 (zero) the command succeeded, and the command(s) that follow between the then and fi constructs are executed. The fi terminates the if block.

If, however, the first command is executed and the exit status is nonzero, it means the command failed. The statements after the then keyword are ignored and control goes to the line directly after the fi statement. Failures can be caused by invalid arguments passed to the program or by an error condition that is detected.

The shell variable $? is automatically set by the shell to the exit status of the last command executed. The echo command displays its value.

This example demonstrates how to view the exit status of a successful and unsuccessful command execution from the command line.

```
$ mkdir newdir
$ echo $?
0
$

$ mkdir newdir
mkdir: Failed to make directory "newdir"; File exists
$ echo $?
2
$
```

The `test` *Command*

The built-in `test` command is often used as the command following the `if` command.

The `test` command evaluates an expression, and, if the result is true, it returns an exit status of zero. Otherwise, the result is false, and it returns a nonzero exit status.

Command Format

```
if  test expression
then
     command
fi
```

In the following example, `test` evaluates the variable `$name` to determine if it has been preset to the value `user2`.

```
if test "$name" = "user2"
then
echo "matches"
fi
```

The equal (=) operator is used to test if two values are identical. It tests to see if the contents of the shell variable `$name` is identical to the characters `user2`. If it is, test returns an exit status of zero; otherwise, it returns a nonzero value.

An Alternate Format for `test`

The `test` command is used so often in shell scripts that an alternative format of the command is recognized by the shell. The bracket is actually the name of the `test` command.

The previous statement could be written as follows:

```
if [ "$name" = "user2" ]
. . .
```

The shell still executes the `test` command; only in this format, `test` expects to see a closing bracket at the end of the expression. Spaces must appear after the first bracket and before the closing bracket.

Command Format

```
if [ expression ]
then
     command
     command
fi
```

The following example shows `test` evaluating an expression. If the result is true, an exit status of zero is returned. If the result is false, it returns a nonzero exit status.

The `test` *Command Operators*

Almost every shell script deals with one or more files, so the `test` command has a wide assortment of file operators. These provide you with the tools necessary to ask questions about files when building a script.

The most commonly used file operators are listed in Table 13-2.

Table 13-2 The `test` File Operators

Operator	Returns TRUE (exit status of 0) if
`-d` *`file`*	*`file`* is a directory
`-f` *`file`*	*`file`* is an ordinary file
`-r` *`file`*	*`file`* is readable by the process
`-s` *`file`*	*`file`* has nonzero length
`-w` *`file`*	*`file`* is writable by the process
`-x` *`file`*	*`file`* is executable

The following example shows the use of a file operator to test if a file exists.

Create the shell script called `myfile5`, make it executable, and run the program. As a last step, check the results of the exit status.

```
$ vi myfile5
#!/bin/sh
if [ -d $HOME ]
then
      echo "EUREKA!"
fi
```

Save the file, and quit the editor.

Apply the `chmod` command to make this shell script executable.

```
$ chmod 755 myfile5
```

Execute the new shell script, and display the exit status.

```
$ ./myfile5
EUREKA!
$ echo $?
0
$
```

The case *Command*

Use the case command when there are many conditions to test.

Command Format

```
case value in
value1 )
            command
            command
            ;;

value2 )
            command
            command
            ;;

* )
            command
            ;;

esac
```

The Value of case

The value of a case variable is matched against *value1, value2,* and so on, until a match is found.

When a value matches the case variable, the commands following that value are executed until double semicolons (;;) are reached. Then control goes to the line directly after the esac statement.

If the value of a case variable is not matched, the program executes the commands after the default value *) until double semicolons or esac is reached.

Note – The case values allow the use of shell wildcards and the pipe symbol for OR-ing two values.

The following example illustrates the use of the `case` command.

```
$ vi myfile6
#!/bin/sh

answer=$1

case “$answer” in

y)
      echo “You selected Yes!”
      ;;
n)
      echo “You selected No!”
      ;;
*)
      echo “Invalid selection”
      ;;
esac
```

Save the file, and quit the editor.

Apply the `chmod` command to make the shell script executable.

```
$ chmod 755 myfile6
```

Execute the new shell script with one of the three arguments: `y`, `n`, or `x`; for example:

```
$ ./myfile6 y
You selected Yes!

$ ./myfile6 n
You selected No!

$ ./myfile6 x
Invalid selection
```

The `case` command evaluates the variable `answer`.

The value of the `case` variable `answer` is matched against each value in the shell script until a match is found. Then the result is echoed to the screen.

The first value tested is to determine if the value of the `case` variable matches `y`.

The second value tested is to determine if the value of the `case` variable matches `n`.

The third value tested is to determine if the value of the `case` variable matches `x`.

The `exit` *Command*

The built-in shell command called `exit` gives the user the ability to immediately terminate execution of a shell script.

The `exit` command is frequently used as a convenient method to end, or breakout of, a shell script and return to the command line.

Command Format

`exit` *n*

The argument given to the `exit` command is an integer ranging from 0 to 255.

You can place one or more `exit` commands in a shell script as a way to instruct the shell to exit the script if a particular condition occurs, prior to the program's normal completion.

In large shell scripts, it is useful to have a wide range of numbers as an argument to each `exit` command to pinpoint where in the script the particular condition occurred.

If the shell script exits with a zero as an argument, the program exited with success. A nonzero argument (1–255) would indicate some type of failure.

You can quickly identify the exit result, which is generated by the `exit` command, using the command:

```
$ echo $?
```

Reading a Sample Solaris Administration Shell Script

The following Bourne shell script is one of many administration scripts maintained in the `/etc/init.d` directory.

These shell scripts are used in system startup and shutdown.

A system administrator can manually start and stop system daemons and services by executing an administration script from the command line using the argument `start` or `stop`.

Note – An additional system administration Bourne shell is located in `/sbin/sh`.

For example, to manually stop or start the audit program, the `root` user would invoke one of the following commands:

```
# /etc/init.d/audit stop
```

or

```
# /etc/init.d/audit start
```

If one of these arguments (that is, `start` or `stop`) is not included on the command line, the following error message is displayed on the screen:

```
Usage: /etc/init.d/audit { start | stop }
```

Using these newly acquired shell scripting skills, analyze each line in the following script to determine what is happening as the program is executed.

The `/etc/init.d/audit` *Shell Script*

```
1    #!/sbin/sh
2    #
3    # Copyright (c) 1997 by Sun Microsystems, Inc.
4    # All rights reserved.
5    #
6    #ident  "@(#)audit      1.5     97/12/08 SMI"
7
8    case "$1" in
9    'start')
10       if [ -f /etc/security/audit_startup ]; then
11                echo 'starting audit daemon'
12                /etc/security/audit_startup
13                /usr/sbin/auditd &
14       fi
15       ;;
16
17   'stop')
18       if [ -f /etc/security/audit_startup ]; then
19                /usr/sbin/audit -T
20       fi
21       ;;
22
23   *)
24       echo "Usage: $0 { start | stop }"
25       exit 1
26       ;;
27   esac
28   exit 0
```

Interpreting the `audit` *Administration Shell Script*

The following is a line-by-line description of what happens when the audit script is executed.

Line 1: Identifies the shell that is used to execute this program (in this case, the Bourne shell).

Lines 2 - 6: Comments, because each line is preceded by a pound symbol (#).

Line 7: Blank line.

Line 8: Begins a `case` statement. The variable given to `case` is the positional parameter `$1`. This variable passes the first argument on the command line into the shell script.

Line 9: Identifies the first value to be matched with the variable given to `case` (in this case, `start`).

If the two values match, the result is true, returning an exit status of 0 (zero) and control goes down to the next line, Line 10.

If the two values do not match, the result is false, returning an exit status of nonzero, and control goes to the line directly after the `;;` statement (in this case, Line 16).

Line 10: Begins a conditional `if` statement with the `test` command evaluating an expression.

If the `/etc/security/audit_startup` file exists, the result is true and control goes to the next line, Line 11.

If the `/etc/security/audit_startup` file does not exist, the statements after the `then` keyword are ignored and control goes to the line directly after the `fi` statement (in this case, Line 15).

Line 11: Executes the `echo` command to display the following text to the screen.

```
starting audit daemon
```

Line 12: Executes the `audit_startup` script to initialize the security audit subsystem prior to the audit daemon actually being started.

Line 13: Starts the audit daemon. This program controls the generation and location of audit trail files.

Line 14: Terminates the `if` block statement.

Line 15: Sends control to the last line directly after the `esac` statement. Lines 16 through 27 are skipped.

Line 16: Blank line.

Line 17: Identifies the second value to be matched with the case variable (in this case, `stop`).

If the two values match, the result is true, which returns an exit status of 0 (zero), and control goes down to the next line, Line 18.

If the two values do not match, the result is false, which returns an exit status of nonzero, and control goes to the line directly after the `;;` statement (Line 22).

Line 18: Begins a conditional `if` statement with the `test` command evaluating an expression.

If the `/etc/security/audit_startup` file exists, the result is true and control goes to the next line, Line 19.

If the `/etc/security/audit_startup` file does not exist, the statements after the `then` keyword are ignored and the control goes to the line directly after the `;;` statement (Line 21).

Line 19: Executes the `/usr/sbin/audit -T` function to signal the audit daemon to close the current audit trail file, to disable auditing, and to terminate.

Line 20: Terminates the `if` block statement.

Line 21: Sends control to the last line directly after the `esac` statement. Lines 22 through 27 are skipped.

Line 22: Blank line.

Line 23: If the `case` variable did not match the values in either Line 9 or Line 17, the shell executes the commands on Line 24 and Line 25.

Line 24: Executes the `echo` command to display the following text string to the screen:

```
Usage: /etc/init.d/audit { start | stop }
```

Line 25: Executes the `exit` command to immediately terminate execution of the shell script and to return control to the command line. Indicates that some type of failure occurred in the program.

Line 26: Sends control to the last line directly after the `esac` statement.

Line 27: Terminates the `case` block statement.

Line 28: Executes the `exit` command to terminate execution of the shell script and return control to the command line. Indicates the program exited with success.

Exercise: Introduction to Reading Shell Scripts

Exercise objective – In this exercise, you practice reading the contents of several different shell scripts using the key points described in this module.

Tasks

Reading Shell Scripts

Examine this particular shell script, and then answer the questions that follow.

```
$ cat /etc/init.d/spc
PATH=/usr/bin:/usr/sbin; export PATH

case "$1" in
'start')
        [ -f /usr/lib/print/printd ] || exit 0

        if [ -z "$_INIT_PREV_LEVEL" ]; then
                set -- '/usr/bin/who -r'
                _INIT_PREV_LEVEL="$9"
        fi

        if [ $_INIT_PREV_LEVEL != 2 -a $_INIT_PREV_LEVEL
!= 3 ]; then
                rm -f /var/spool/print/tf* >/dev/null
2>&1
                /usr/lib/print/printd &
        fi
        ;;

'stop')
        /usr/bin/pkill -x -u 0 printd
        ;;
```

```
*)
        echo "Usage: $0 { start | stop }"
        exit 1
        ;;
esac
exit 0
```

1. What must be the value of the first positional parameter (`$1`) for the first branch of the `case` statement to be executed?

   _____

2. After entering the `start` branch of the `case` statement, under what conditions will this shell script exit?

   _____

   _____

   _____

   _____

3. Commands typically report their execution errors using `stderr` (standard error). If the `rm` command in this shell script experienced an error during its execution, where would the error be displayed?

   _____

   _____

   _____

   The following is a segment of a shell script called `/etc/init.d/MOUNTFSYS`. Examine this shell script, and then answer the questions that follow.

```
if [ ! -d /usr/sbin ]; then
        echo "WARNING: /usr subtree is missing: changing to single user
mode"
        /sbin/init S
fi
```

4. Under what condition will the `then` branch of the first `if` construct be executed?

   _____

Here is a segment of a shell script called /etc/init.d/rpc. Examine this shell script, and then answer the questions that follow.

```
# Start the key service but only if the domain has been set
        if [ -x /usr/sbin/keyserv -a \
            -n "`/usr/bin/domainname 2>/dev/null`" ]; then
                /usr/sbin/keyserv >/dev/msglog 2>&1
                echo " keyserv\c"
        fi
```

5. Under what conditions will the then branch of the if construct be executed?

   _____

6. When /usr/sbin/keyserv is executed, where will standard output be displayed? Where will the standard error output be displayed?

   _____

Below is a segment of a shell script called /etc/init.d/sysid.sys. Examine this shell script, and then answer the questions that follow.

```
pkginfo -q SUNWsibi
status=$?

if [ "$status" = "1" -a "$NeedReboot" = "yes" ]
then
        echo "\nrebooting system due to change(s) in \
/etc/default/init \n"
        Net=`need_net_reboot`
        /usr/sbin/reboot -l $Net
fi
```

7. Assume that the pkginfo command executes without error. What value will be assigned to the variable called status?

   _____

8. What is the purpose of the special reserved variable $? ?

   _____

9. Under what conditions will the commands inside the if construct be executed?

   _____

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the exercises.

- Experiences
- Interpretations
- Conclusions
- Applications

Task Solutions

Reading Shell Scripts

1. What must be the value of the first positional parameter (`$1`) for the first branch of the `case` statement to be executed?

   `$1` *must have a value of start for the first branch of the* `case` *statement to execute.*

2. After entering the `start` branch of the `case` statement, under what conditions will this shell script exit?

   The shell script will exit if the ninth positional parameter (`$9`*) in the output of the* `who -r` *command is either a 2 or a 3.*

3. Commands typically report their execution errors using `stderr` (standard error). If the `rm` command in this shell script experienced an error during its execution, where would the error be displayed?

   Errors would not be displayed at all because they are being redirected to `/dev/null`.

4. Under what condition will the `then` branch of the first `if` construct be executed?

   The `then` *branch will execute if the* `/usr/sbin` *directory does not exist.*

5. Under what conditions will the `then` branch of the `if` construct be executed?

   The `then` *branch will execute if the* `/usr/sbin/keyserv` *file is executable.*

6. When `/usr/sbin/keyserv` is executed, where will standard output be displayed? Where will the standard error output be displayed?

   Standard output and standard error output will be displayed on the console (`/dev/console`*).*

7. Assume that the `pkginfo` command executes without error. What value will be assigned to the variable called `status`?

   The value zero is assigned to the variable `status`.

8. What is the purpose of the special reserved variable, `$?` ?

   `$?` *is the variable that always contains the exit status (success or failure) of the most recently executed command.*

9. Under what conditions will the commands inside the `if` construct be executed?

   The commands inside the `if` *construct will be executed if the* `pkginfo` *command does not execute correctly and the value of the variable called* `NeedReboot` *is yes.*

Check Your Progress

Before continuing on to the next module, check that you are able to accomplish or answer the following:

- ❑ Identify which shell program interprets the lines of a shell script
- ❑ Explain how command-line arguments are passed to a shell script with special variables called positional parameters
- ❑ Demonstrate the use of two conditional commands: `if` and `case`, as well as the `test` command.
- ❑ Interpret the contents of a simple Bourne shell administration script

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

W

Z

Please
Recycle

Adobe PostScript