Solaris 10 Operating System Essentials

Student Guide

SA-100-S10 Rev C.1

D61752GC11
Edition 1.1
D65079

ORACLE®

This page intentionally left blank.

This page intentionally left blank.

Table of Contents

About This Course

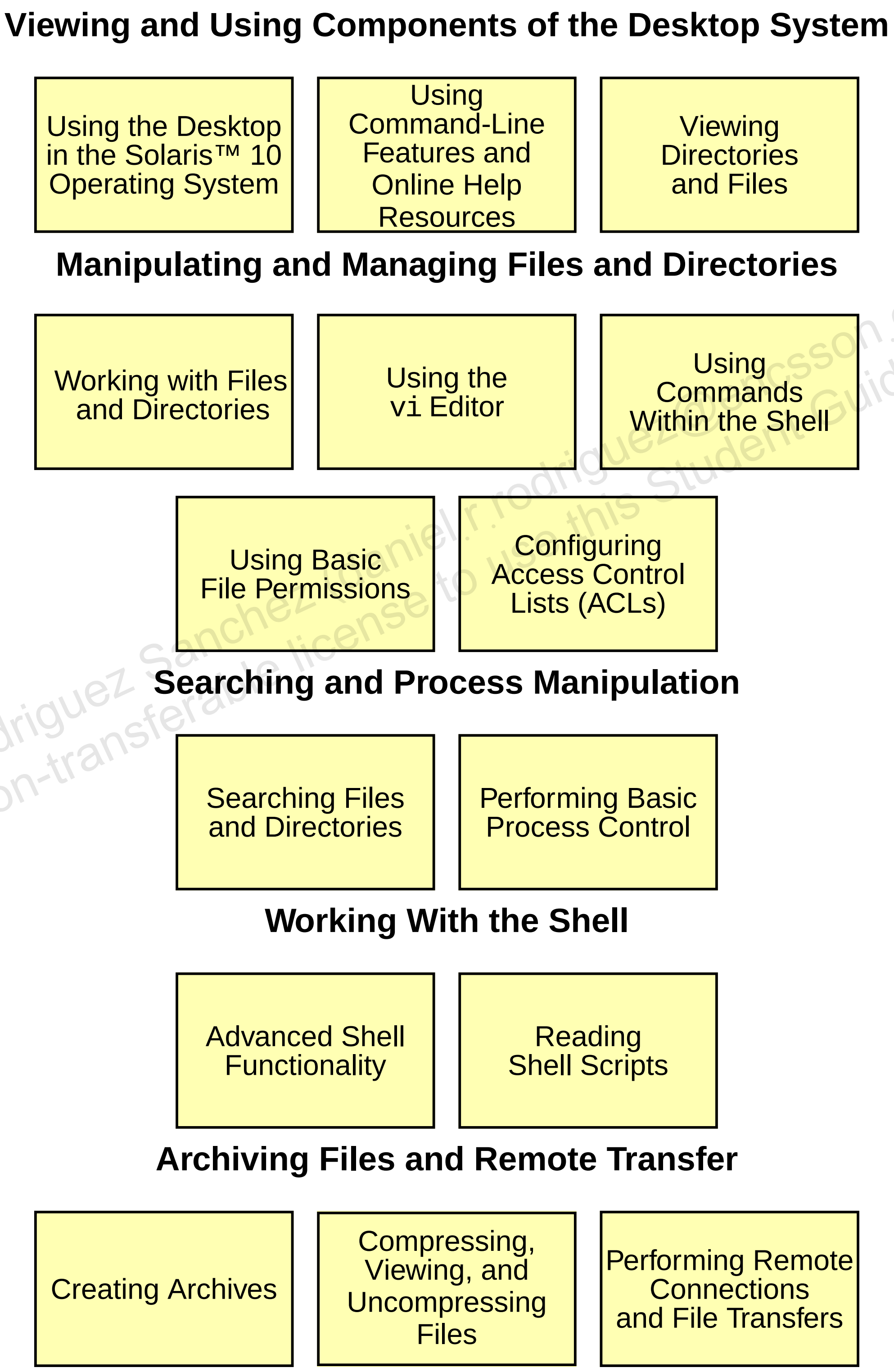
Course Goals

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

- Use components of the Solaris desktop system
- Manage files and directories in Solaris
- Change file permissions and configure ACLs
- Use the `vi` Editor
- Perform basic process control to track and run processes
- Manage a job, aliases, and functions in the shell environment
- Control the user's work environment
- Archive files and perform remote operations

Course Map

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


Viewing and Using Components of the Desktop System
Using the Desktop in the Solaris™ 10 Operating System
Using Command-Line Features and Online Help Resources
Viewing Directories and Files
Manipulating and Managing Files and Directories
Working with Files and Directories
Using the vi Editor
Using Commands Within the Shell
Using Basic File Permissions
Configuring Access Control Lists (ACLs)
Searching and Process Manipulation
Searching Files and Directories
Performing Basic Process Control
Working With the Shell
Advanced Shell Functionality
Reading Shell Scripts
Archiving Files and Remote Transfer
Creating Archives
Compressing, Viewing, and Uncompressing Files
Performing Remote Connections and File Transfers

Topics Not Covered

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

- System administration concepts – Covered in SA-200-S10: *System Administration for the Solaris™ 10 Operating System, Part 1*
- Advanced system administration concepts – Covered in SA-202-S10: *System Administration for the Solaris™ 10 Operating System, Part 2*
- System administration shell programming – Covered in SA-245: *Shell Programming for System Administrators*

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

How Prepared Are You?

To be sure you are prepared to take this course, can you answer yes to the following questions?

- Can you use a keyboard to input commands and control characters?
- Can you use a mouse to point and click in a graphical user interface (GUI)?
- Are you familiar with working in a desktop environment?

Course Environments

Course environments include the following:

- This course uses the Java™ Desktop System (JDS) throughout this course, because it is currently the default graphical user interface (GUI).
- The Korn shell is used throughout this course, as it is the superset of the default Bourne shell in the Solaris™ Operating System (Solaris OS).

Introductions

Now that you have been introduced to the course, introduce yourself to the other students and the instructor, addressing the following items:

- Name
- Company affiliation
- Title, function, and job responsibility
- Experience related to topics presented in this course
- Reasons for enrolling in this course
- Expectations for this course.

How to Use Course Materials

To enable you to succeed in this course, these course materials contain a learning module that is composed of the following components:

- Goals – You should be able to accomplish the goals after finishing this course and meeting all of its objectives.
- Objectives – You should be able to accomplish the objectives after completing a portion of instructional content. Objectives support goals and can support other higher-level objectives.
- Lecture – The instructor presents information specific to the objective of the module. This information helps you learn the knowledge and skills necessary to succeed with the activities.
- Activities – The activities take on various forms, such as an exercise, self-check, discussion, and demonstration. Activities help you facilitate the mastery of an objective.
- Visual aids – The instructor might use several visual aids to convey a concept, such as a process, in a visual form. Visual aids commonly contain graphics, animation, and video.

Note – Many system administration tasks for the Solaris OS can be accomplished in more than one way. The methods presented in the courseware reflect recommended practices used by Sun Educational Services.

Conventions

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

Icons

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

Note – Indicates additional information that can help students but is not crucial to their understanding of the concept being described. Students should be able to understand the concept or complete the task without this information. Examples of notational information include keyword shortcuts and minor system adjustments.

Caution – Indicates that there is a risk of personal injury from a nonelectrical hazard, or risk of irreversible damage to data, software, or the operating system. A caution indicates the possibility (as opposed to the certainty) that a hazard might occur, depending on the action of the user.

Typographical Conventions

`Courier` is used for the names of commands, files, directories, user names, host names, programming code, and on-screen computer output; for example:

> Use the `ls -al` command to list all files.
> `host1# cd /home`

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

> To list the files in this directory, type the following:
> `#` **`ls`**

`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, use the `rm` *`filename`* command.

`Courier italic bold` is used to represent variables whose values are to be entered by the student as part of an activity; for example:

> Type **`chmod a+rwx`** ***`filename`*** to grant read, write, and execute rights for *`filename`*.

Palatino italic is used for book titles, new words or terms, or words that need to be emphasized; for example:

> Read Chapter 6 in the *User's Guide*.
> These are called *class* options.

Additional Conventions

Java programming language examples use the following additional conventions:

- Method names are not followed with parentheses unless a formal or actual parameter list is shown; for example:

  "The `doIt` method..." refers to any method called `doIt`.

  "The `doIt()` method..." refers to a method called `doIt` that takes no arguments.

- Line breaks occur only where there are separations (commas), conjunctions (operators), or white space in the code. Broken code is indented four spaces under the starting code.

- If a command used in the Solaris OS is different from a command used in the Microsoft Windows platform, both commands are shown; for example:

  If working in the Solaris OS

  ```
  $CD SERVER_ROOT/BIN
  ```

  If working in Microsoft Windows

  ```
  C:\>CD SERVER_ROOT\BIN
  ```

Module 1

Using the Desktop in the Solaris™ 10 Operating System

Objectives

This module introduces the desktop in the Solaris 10 OS. This module also describes the desktop hardware and how to add workspaces and manage files.

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

- Describe the Solaris OS
- Describe the hardware components of a computer
- Describe the Solaris OS components
- Describe the SunOS™ software
- Log into the system
- Use the desktop environment

Figure 1-1 is the course map that shows how this module fits into the current instructional goal.

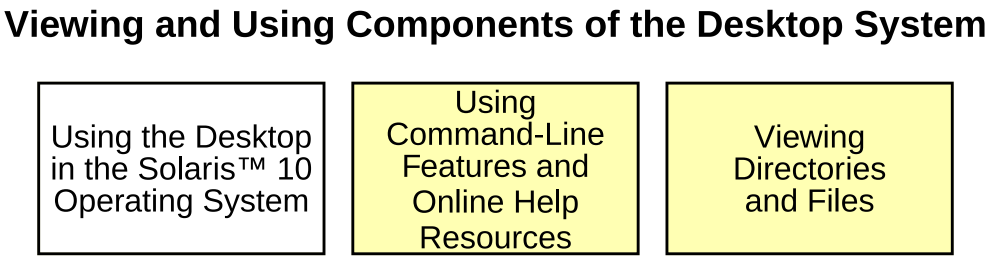

Viewing and Using Components of the Desktop System
Using the Desktop in the Solaris™ 10 Operating System
Using Command-Line Features and Online Help Resources
Viewing Directories and Files

Figure 1-1 Course Map

Introducing the Solaris Operating System

The UNIX® operating system was developed originally 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 the resulting operating system the *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 enabled the operating system to function in a local area network (LAN).

Sun's original version of the UNIX operating system was known as the *SunOS software*, 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.

Computer Hardware Components

A computer consists of hardware and software that work together to perform tasks. The computer hardware is made up of several components, such as the central processing unit (CPU), random access memory (RAM), and disks. Computer software refers to a set of programs or applications that run the computer. The operating system is a set of programs and files that directs and controls both the hardware and the software.

Figure 1-2 shows the four main components that make up the hardware of a computer. These components are the RAM, the CPU, the input/output (I/O) devices, and the hard disk or some other storage device.

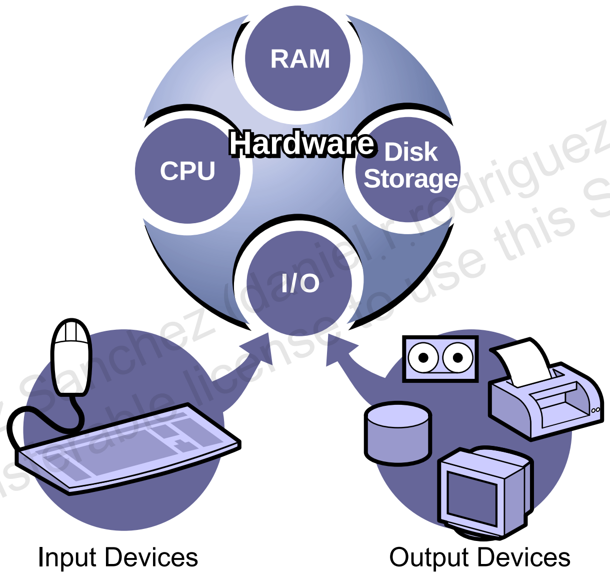


Figure 1-2 Main Hardware Components of a Computer

RAM

Random Access Memory (RAM) is the main computer memory. The statement, "The system has 512 Mbytes of memory," refers to the amount of RAM installed on the computer. Software programs and data must load into RAM before the operating system can process them.

The SunOS 5.*x* operating system is a virtual memory operating system. A virtual memory operating system enables execution of programs as if more memory is available than physically exists in RAM. Disk space is used as temporary memory storage to facilitate this apparent increase in memory.

Programs reside on the hard disk. When you execute a program, a copy of the program loads into virtual memory. Portions of the program are copied into RAM during the actual execution process. The program remains in virtual memory until it finishes. Upon program termination, the operating system can overwrite the virtual memory space with other programs that it needs to execute. If you reboot the system or if there is a power loss, all data in virtual memory is cleared.

CPU

The Central Processing Unit (CPU) is the computer logic chip that executes instructions that it receives from RAM. These instructions are stored in binary form on the hard disk.

I/O Devices

Input/Output (I/O) devices are computer components that communicate with external pieces of equipment. An I/O device reads information from an input device, such as a keyboard or a mouse, into RAM. An I/O device writes this information to output devices, such as monitors, printers, or tape drives.

Hard Disk

The hard disk is a magnetic storage device that stores files, directories, and software programs.

Solaris Operating System Components

The Solaris 10 OS consists of the SunOS 5.*x* operating system, Open Network Computing (ONC+) software, and the Solaris OS desktop environment GUI.

The SunOS operating system is software that manages system resources and schedules system operations. This operating system interprets instructions from the user, or from an application, and instructs the computer what to do. The operating system handles data, keeps track of data stored on disks, and communicates with I/O devices, such as monitors, hard disks, flash drives, printers, and modems.

ONC+ software provides network services, such as Network File System (NFS), which allows file sharing between computers. Another service is Lightweight Directory Access Protocol (LDAP), which is the protocol that clients use to communicate with a directory server. LDAP is a vendor independent protocol and can be used on common TCP/IP networks.

The desktop environment is the windowing environment or GUI that displays the Login screen on your monitor. You can use the desktop environment to access most of the functions of the computer.

SunOS™ Operating System

This section introduces the three components of the SunOS operating system and describes how these components interact with each other. The three main components of the SunOS operating system include the:

- Kernel
- Shell
- Directory Hierarchy

Kernel

The kernel is the core of the SunOS operating system. Figure 1-3 shows the kernel, which manages all the physical resources of the computer, including:

- File systems and structures
- Device management, such as storing data to the hard disk
- Process management or CPU functions
- Memory management

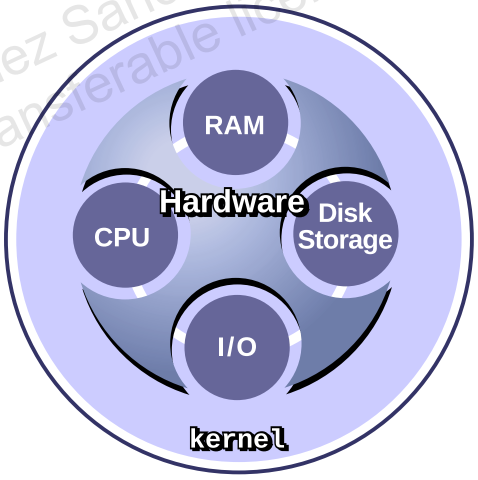


Figure 1-3 Role of the Kernel

Shell

Figure 1-4 shows the shell as an interface between the user and the kernel. The shell is primarily a command interpreter. The shell accepts the commands that a user enters, interprets these commands, and passes them to the kernel. The kernel executes the commands.

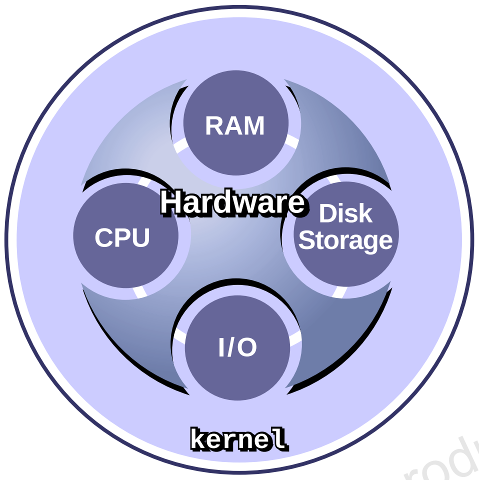


Figure 1-4 Role of the Shell

Default Shells

The Solaris OS supports three primary shells:

- Bourne shell
- C shell
- Korn shell

Note – Examples shown in this course assume the use of the Bourne shell.

The Bourne shell is the original UNIX system shell. The Bourne shell is the default shell for the `root` user. The `root` user is a special system account with unlimited access privileges for system administrators. The default Bourne shell prompt for a regular user is a dollar sign ($). For the `root` user, the default shell prompt is a pound sign (#).

The C shell has several features that the Bourne shell does not, such as command-line history, aliasing, and job control. The default C shell prompt for a regular user is the host name followed by a percent sign (*hostname%*). For the `root` user, the default shell prompt is the host name followed by a pound sign (*hostname#*).

The Korn shell is a superset of the Bourne shell with C shell-like enhancements and additional features, such as command history, command-line editing, aliasing, and job control. The default Korn shell prompt for a regular user is a dollar sign (`$`). For the `root` user, the default shell prompt is the pound sign (#).

Alternative Shells

The Solaris 10 OS contains three additional (alternative) shells:

- Bash – The GNU project's Bourne-Again shell is a Bourne-compatible shell that incorporates useful features from the Korn and C shells, such as command history, command-line editing, and aliasing.
- Z shell – The Z shell most closely resembles the Korn shell, but it includes many other enhancements.
- TC shell – The TC shell is a completely compatible version of the C shell with additional enhancements.

Directory Hierarchy

Figure 1-5 shows a Solaris OS directory hierarchy containing an organized group of directories and files. Figure 1-5 shows the starting point of the `student` home directory within the directory hierarchy.

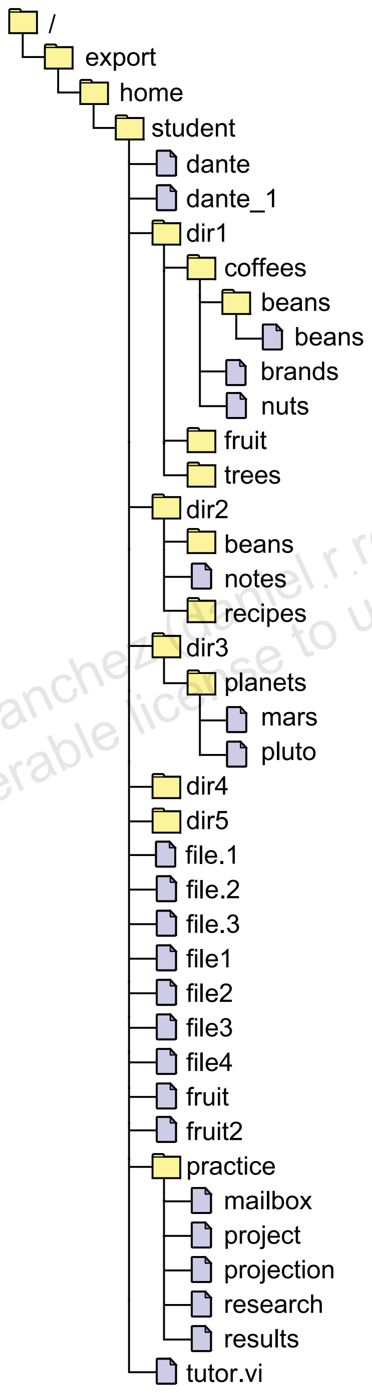


Figure 1-5 Directory Hierarchy

Logging Into the System

All users must follow a login process so that the system can recognize and authenticate the user. The desktop Login screen, which appears on your monitor, enables you to log into the system and use the desktop. Figure 1-6 shows the Login screen.

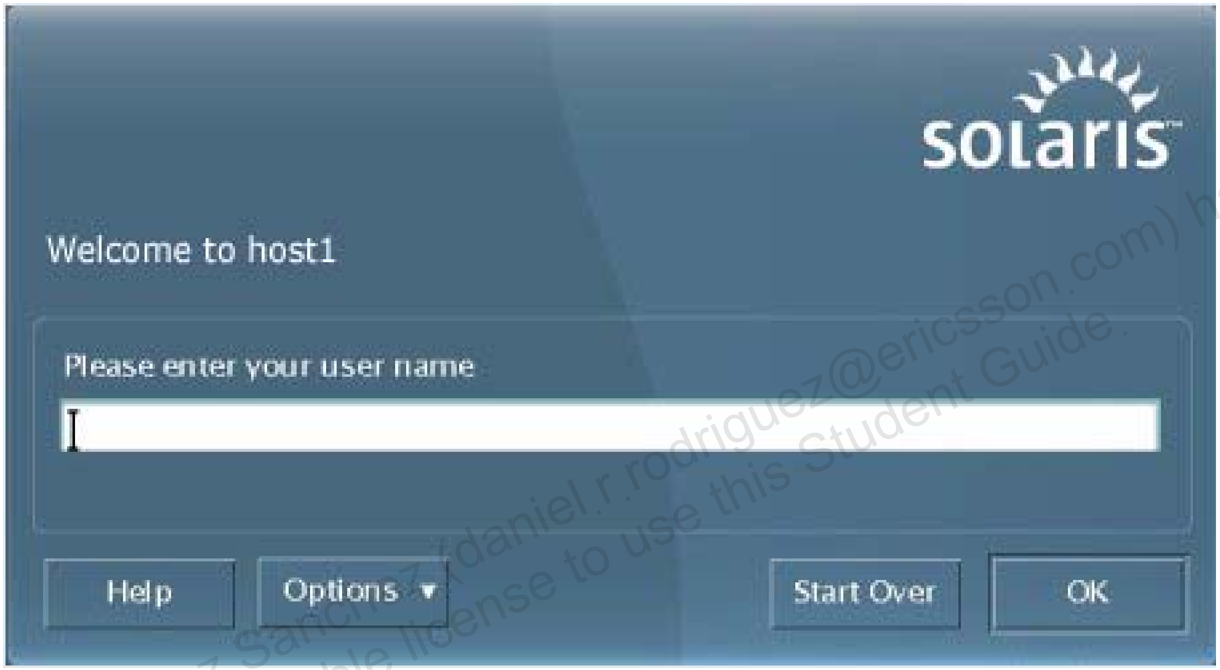


Figure 1-6 Login Screen

There are several ways to log into the system. You can log in directly by entering your user name and password, or you can use the Options button. The Options button enables you to log into a remote host or to log in using the command-line.

Selecting Login Options

If you do not choose to log into a system directly, click Options on the Login screen. The menu lists a hierarchy of login choices. Select the option that you want to use. The following lists the menu items under the Options button of the Login screen:

- Options
  - Language
  - Sessions
    - Common Desktop Environment (CDE)
    - Sun Java™ Desktop System, Release 3
    - User's Last Desktop
    - Failsafe Session
  - Remote Login
    - Enter Host Name...
    - Choose Host From List...
  - Command Line Login
  - Reset Login Screen

Logging in Using the Command Line

To log into a command-line session, complete the following steps:

1. Click Options.

   The Options pull-down menu appears.
2. Click Command Line Login.

   The Login screen closes.
3. Press Return to get a prompt for a user name entry.

   If you do not log in within 30 seconds, the Login screen automatically restarts.
4. Enter your user name at the prompt, and press Return.
5. Enter your password at the `Password:` prompt, and press Return.

   The password does not appear on the screen when it is entered.
6. Log out of your command-line session by typing `exit` and pressing Return.

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

Changing Your Password

Passwords protect user accounts from unauthorized access. Users should change their passwords frequently to prevent unauthorized access to their systems.

In the Solaris OS, a user's password must have the following characteristics:

- Be six to eight characters in length
- Contain at least two alphabetic characters and must contain at least one numeric or special character, such as a semicolon (`;`), an asterisk (`*`), or a dollar sign (`$`)
- Differ from the user's login name
- Differ from the previous password by at least three characters
- Contain spaces (optional)
- Not be the reverse of the user's login name

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

To change your password, complete the following steps:

1. Log into the system.
2. Run the `passwd` command at the shell prompt in a terminal window, and press Return.
3. When the prompt `Enter existing login password:` appears, enter the current password, and press Return.
4. When the prompt `New password:` appears, type the new password, and press Return.

5. When the prompt `Re-enter new Password:` appears, re-enter the new password, and press Return.

   The system uses your second entry to verify the new password as shown in the following example.

```
$ passwd
passwd: Changing password for student
Enter existing login password:
New password:
Re-enter new passwd:
passwd: passwd successfully changed for student
$
```

Using the Desktop Environment

The desktop environment is the standard desktop environment available to Solaris 10 OS users. This section describes how to use the desktop to secure and select sessions, add and delete workspaces, change backgrounds, and manage files.

Selecting Login Options

If you do not choose to log into a system directly, click Options on the Login screen. The menu lists a hierarchy of login choices. Select the option that you want to use. The following lists the menu items under the Options button of the Login screen:

- Options
  - Language
  - Sessions
    - Common Desktop Environment (CDE)
    - Sun Java Desktop System, Release 3
    - User's Last Desktop
    - Failsafe Session
  - Remote Login
    - Enter Host Name...
    - Choose Host From List...
  - Command Line Login
  - Reset Login Screen

Language

You can use the Language option to set a language of your choice for your session. The system administrator sets a default language for your workstation.

Session

You can use the Sessions option to select any of the following sessions:

- Common Desktop Environment
- Java Desktop System
- User's Last Desktop
- Failsafe Session

Remote Login

The Remote Login option enables you to connect to a remote system to start a remote desktop environment login. You can either enter the specific host name of a remote system or select from a list of available remote systems.

Command Line Login

You can use the Command Line Login option to work in a UNIX command environment. This non-GUI environment is not a desktop environment session. The desktop environment is suspended in the command-line login mode.

When you log out from a command-line session, the desktop environment Login screen reappears within 30 seconds.

Reset Login Screen

You can use the Reset Login Screen option to restart the desktop environment Login screen.

Logging in Using the Desktop Login Screen

To log into a desktop session from the desktop environment Login screen, complete the following steps:

1. Type your user name in the text field. Press Return or click OK.
2. Type your password in the password text field. Press Return or click OK.

   If the login attempt fails, a dialog box with the following message appears:

```
Login incorrect; please try again.
```

Securing Your Desktop Environment Session

Securing your desktop environment session prevents unauthorized users from gaining access to the system. There are two ways to secure the system:

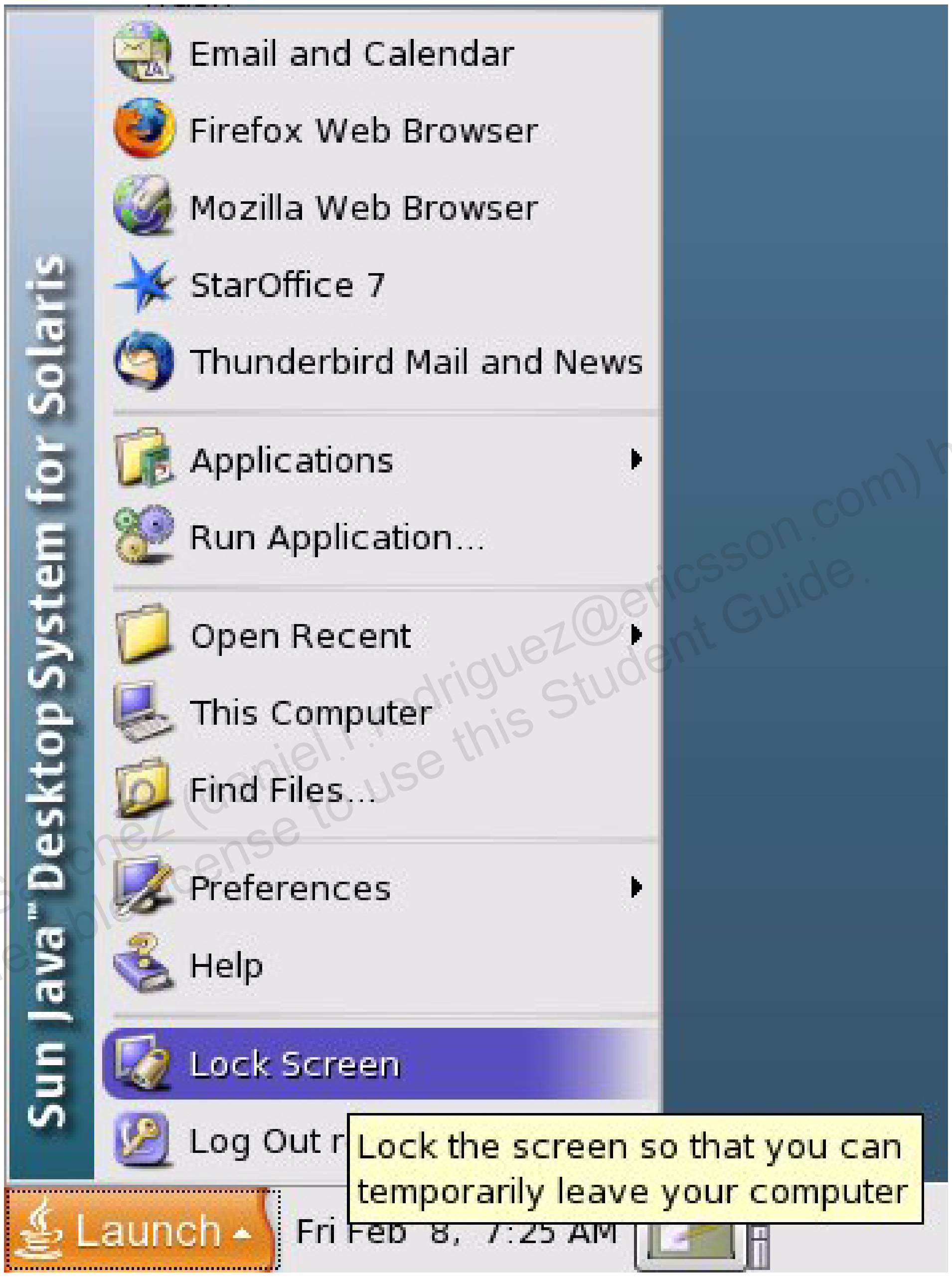
- Locking the screen
- Exiting the session

Locking the Screen

Locking the screen prevents unauthorized users from gaining access to your desktop environment session, while keeping your session intact. To lock the screen, complete the following steps:

1. Click Launch > Lock Screen to lock the current desktop environment session.
2. To regain access to your desktop session you will type your username and password into the dialog box that appears on your workstation when the mouse is moved or keyboard is used.

Figure 1-7 shows the Lock Screen option in the Launch menu:



Lock the screen so that you can temporarily leave your computer

Figure 1-7 Lock Screen

Note – The `root` user has the ability to unlock the desktop login screen without the user's password. The root user will use the root password to accomplish this task.

Exiting the Session

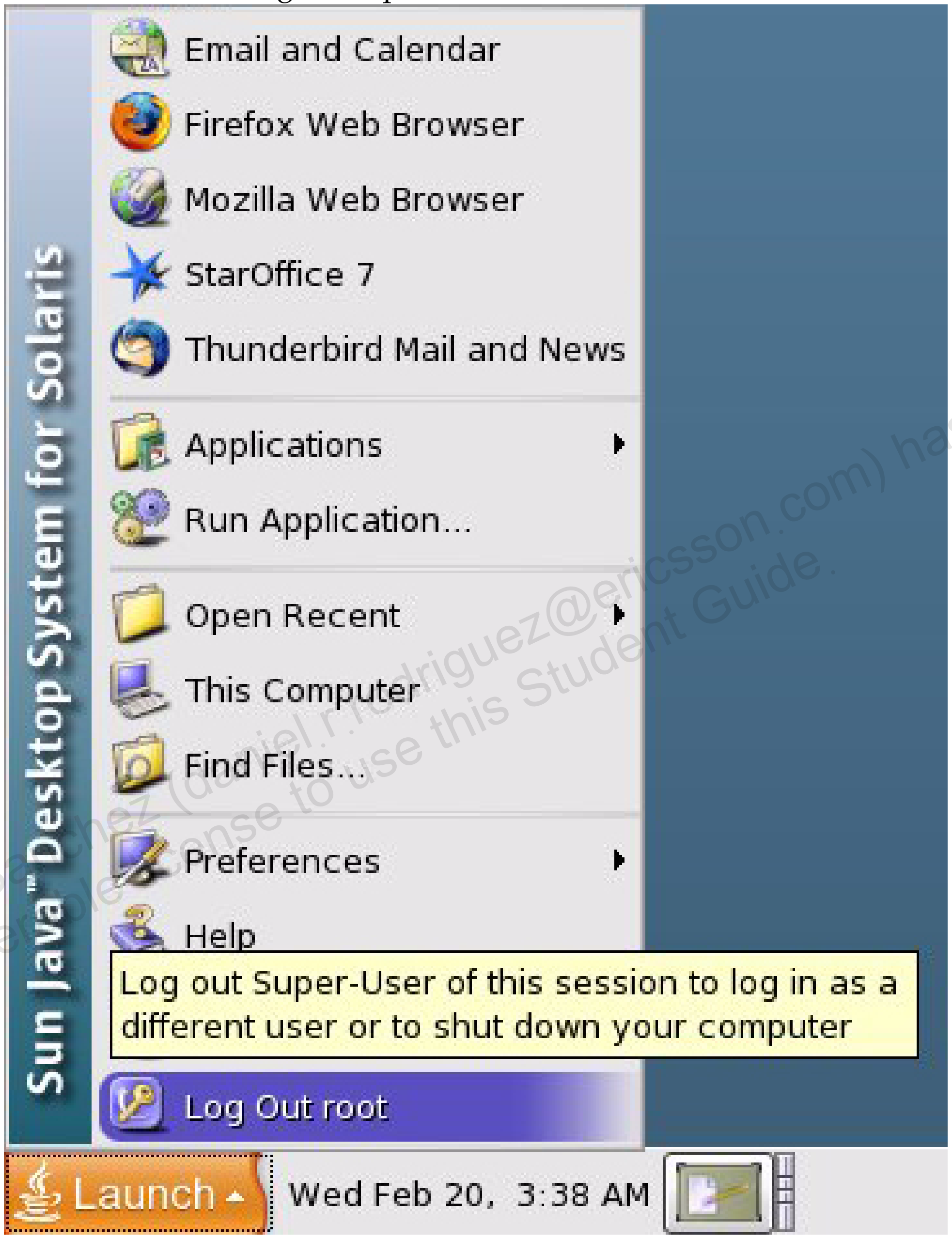
Exiting your session ends that session completely. Any data contained in the current set of open applications is lost when you exit. Save all currently unsaved data before exiting the session. To exit the current session, complete the following steps:

1. Click Launch > Log Out `username`.

   A Logout Confirmation window appears.

2. Click OK or press Return when the OK button is highlighted to confirm that you want to log out.

Figure 1-8 shows the Log Out option in the Launch menu:



Log out Super-User of this session to log in as a different user or to shut down your computer

Figure 1-8 Log Out

Selecting Desktop Login Sessions

The Options button on the Login screen allows you to select three types of sessions: desktop environment, User's Last Desktop, or Failsafe, depending on your desktop requirements.

The User's Last Desktop option displays the desktop that you used at the end of your previous session. This option is the default option if you do not customize your startup session using the Launch > Preferences > Desktop Preferences > Sessions option.

The Failsafe option displays a terminal window instead of starting a desktop window. This login method can be used when you are experiencing problems logging in by starting a single X-terminal window. The Failsafe Session option is always available, even when the Command Line Login option is not available.

Initial Desktop Configuration

When you start a session for the first time, you see a startup screen with a panel at the bottom of the screen and various icons on the desktop background. Figure 1-9 shows the initial desktop configuration of Java Desktop System.

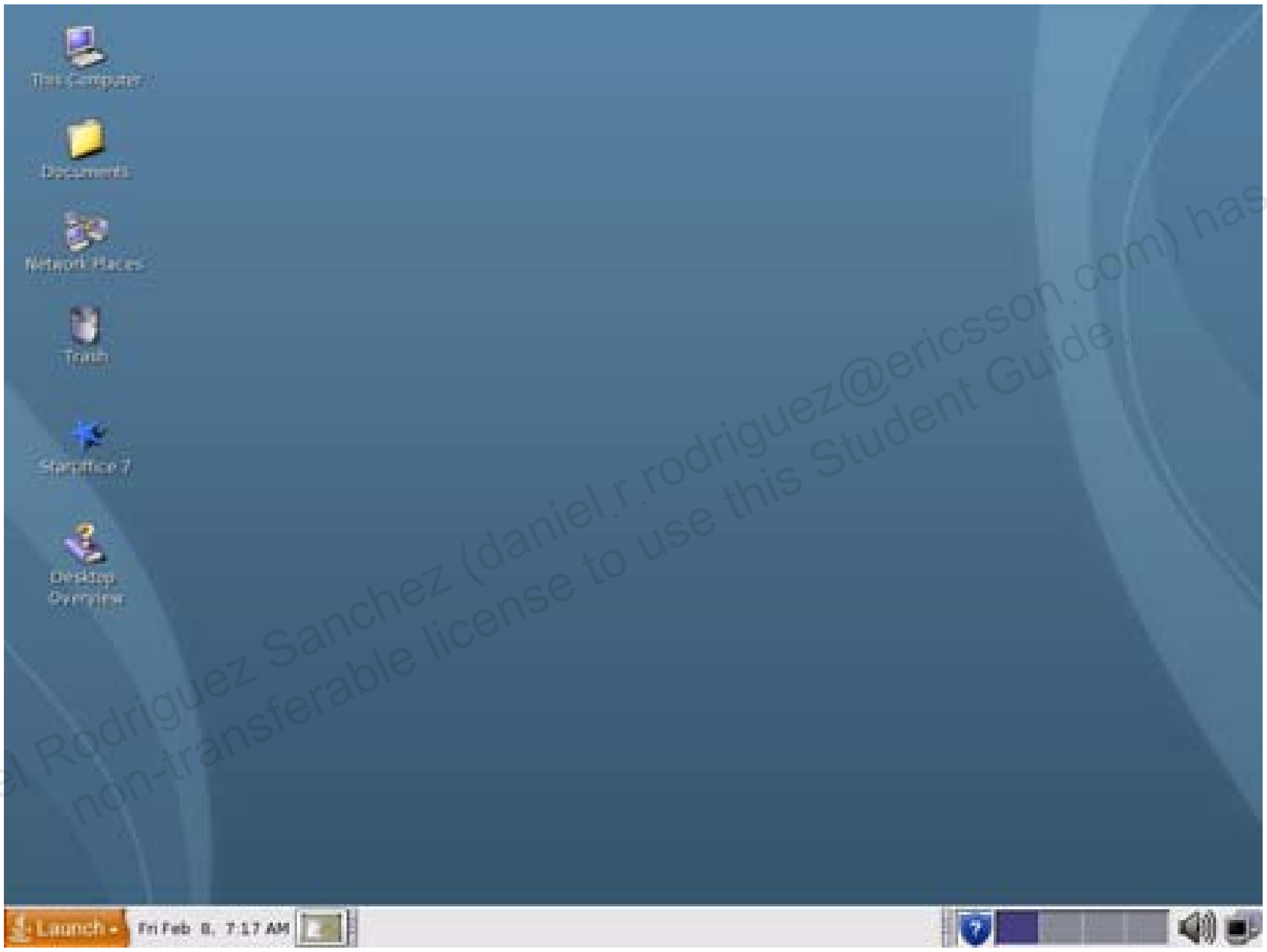


Figure 1-9 Initial Desktop Configuration

The Java Desktop System startup screen typically consists of the following:

- This Computer: Enables you to access media and configuration details about your system.
- Documents: Enables direct access to files stored in your default directory.
- Network Places: Provides access to networked computers from a single directory.
- Trash: Provides a temporary holding area for files, folders or desktop objects that you can later retrieve or permanently delete.
- StarOffice 7: Launches the StarOffice application.
- Desktop Overview: Launches a help browser from where you can access overview information about the Java Desktop System.
- Bottom edge panel: The following panel objects appear in your bottom edge panel by default:
  - Launch button: Provides menus for applications and configuration tools on the Java Desktop System.
  - Clock: Displays the time and date in a panel.
  - Show Desktop: Displays a button that you can click to minimize all open windows. The windows can be restored by clicking the button again.
  - Window List: Displays a button for each window that is open. You can click on a window list button to minimize and restore windows.
  - Notification Area: Displays icons from various applications to indicate activity in the application. For example, when you use the CD Player application to play a CD, a CD icon is displayed in the Notification Area.
  - Workspace Switcher: Displays a visual representation of your workspaces. You can use Workspace Switcher to switch between workspaces. You can also move windows from one workspace to another with the workspace switcher.
  - Volume Control: Displays a button that enables you to control, set, and mute the sound level for your system.
  - Network Monitor: Displays a dialog box that enables you to monitor the host's connection to the network.

- Desktop Background: The desktop background lies behind all of the other components on the Java Desktop System. The desktop background is an active component of the user interface. You can place objects on the desktop background to quickly access your files and directories, or to start applications that you use often. You can also right-click on the desktop background to open a menu to access Help, Open Terminal, Create Folder, Create Document, Change Desktop Background, among other tasks.

Changing the Desktop Background

You can change the background for your desktop with the Desktop Background Preferences window. The color choices you make also affect the backdrop appearance. Figure 1-10 shows the Desktop Background Preferences window.

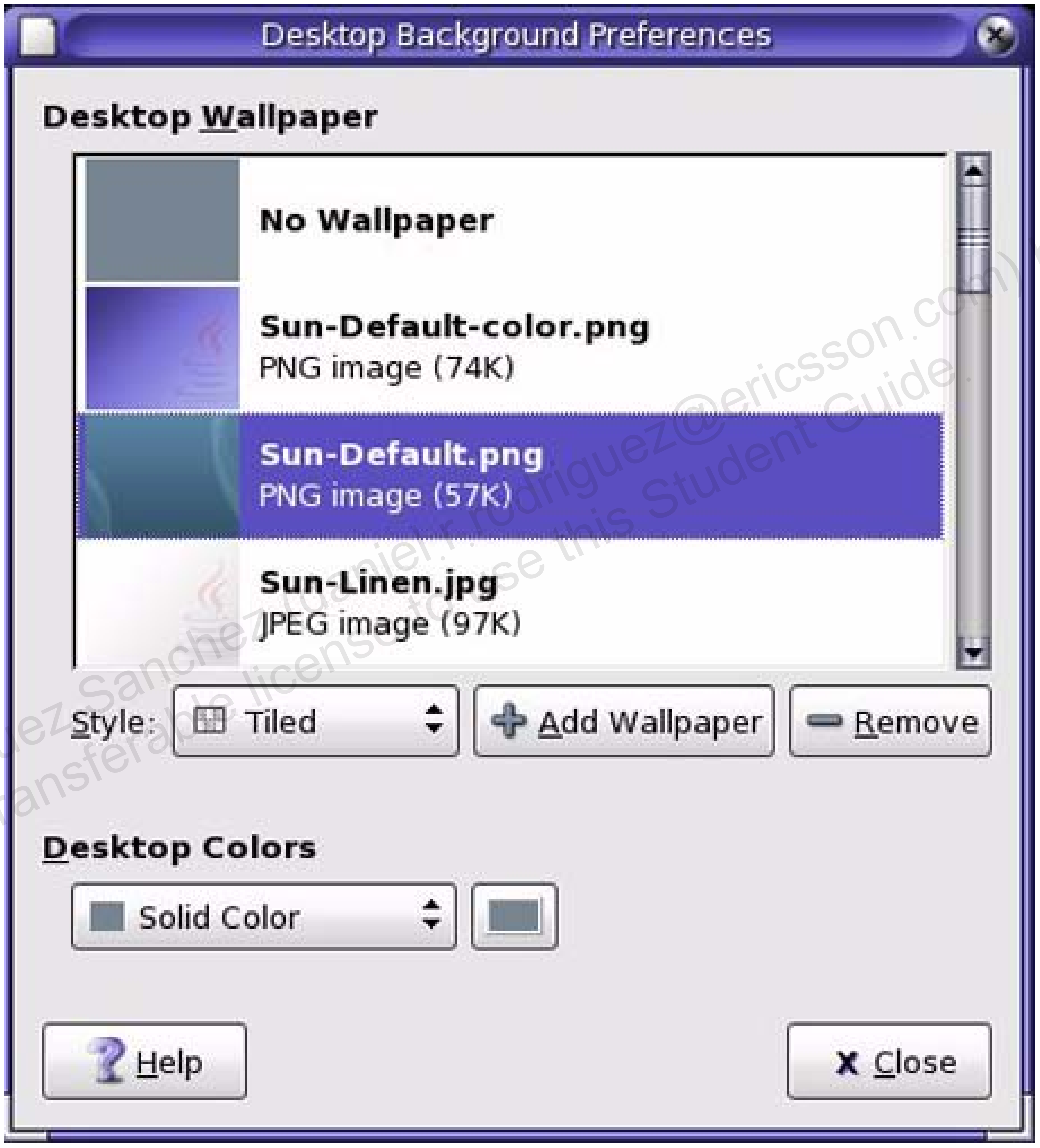


Figure 1-10 Desktop Background Preferences Window

To change the background on your desktop, complete the following steps:

1. Click Launch > Preferences > Desktop Preferences > Display > Desktop Background option to display the Desktop Background Preferences window.
2. Select the backdrop you want to use from the Desktop Background Preferences window.
3. Click Close.

Accessing Desktop Workspaces

There are four default workspace buttons on the front panel. Each workspace is a separate desktop environment. Click the workspace buttons to move from one workspace to another.

You can customize each workspace with the Workspace Switcher Preferences window. You can also add, delete, and rename workspaces. The number of workspaces is reflected by the number of buttons on the front panel. Figure 1-11 shows the Workspace Switcher Preferences window.

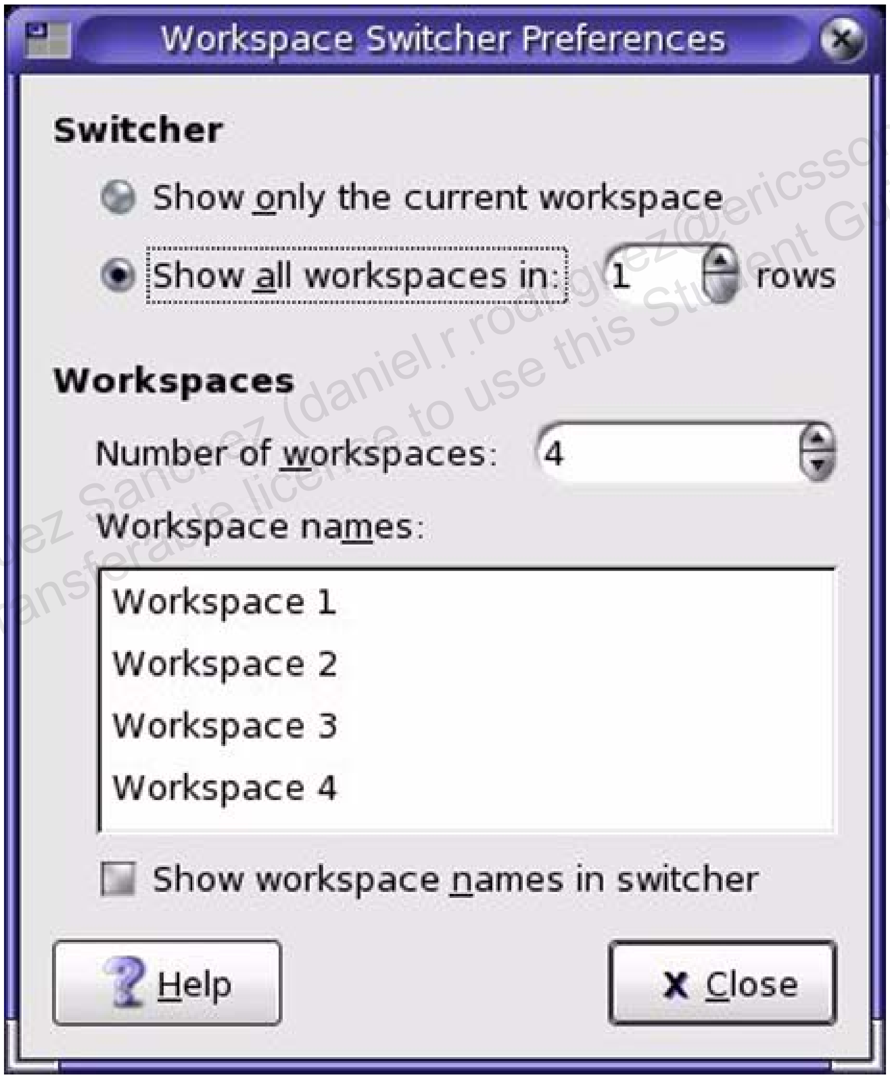

Workspace Switcher Preferences
Switcher
Show only the current workspace
Show all workspaces in: 1 rows
Workspaces
Number of workspaces: 4
Workspace names:
Workspace 1
Workspace 2
Workspace 3
Workspace 4
Show workspace names in switcher
Help
Close

Figure 1-11 Workspace Switcher Preferences

Adding a Desktop Workspace

To add a workspace, complete the following steps:

1. Move the mouse pointer to the Workspace Switcher area on the front panel.
2. Right click on any workspace and select Preferences to open the Workspace Switcher Preferences window.
3. Click the upper button of the Number of workspaces spin box under the Workspaces section. This adds a new workspace at the bottom of the Workspace names list. Additionally, a new workspace appears on the front panel too.

Renaming a Desktop Workspace

To change the name of the workspace that is currently active, complete the following steps:

1. Move the mouse pointer to the Workspace Switcher area on the front panel.
2. Right click any workspace and select Preferences to open the Workspace Switcher Preferences window.
3. Double click on the name of a workspace that you want to rename, then type the new name. The workspace is renamed.

Deleting a Desktop Workspace

To delete a workspace, complete the following steps:

1. Move the mouse pointer to the Workspace Switcher area on the front panel.
2. Right click on any workspace and select Preferences to open the Workspace Switcher Preferences window.
3. Click the lower button of the Number of workspaces spin box under the Workspaces section. This removes a workspace from the bottom of the Workspace names list. Additionally, the workspace is also removed from the front panel.

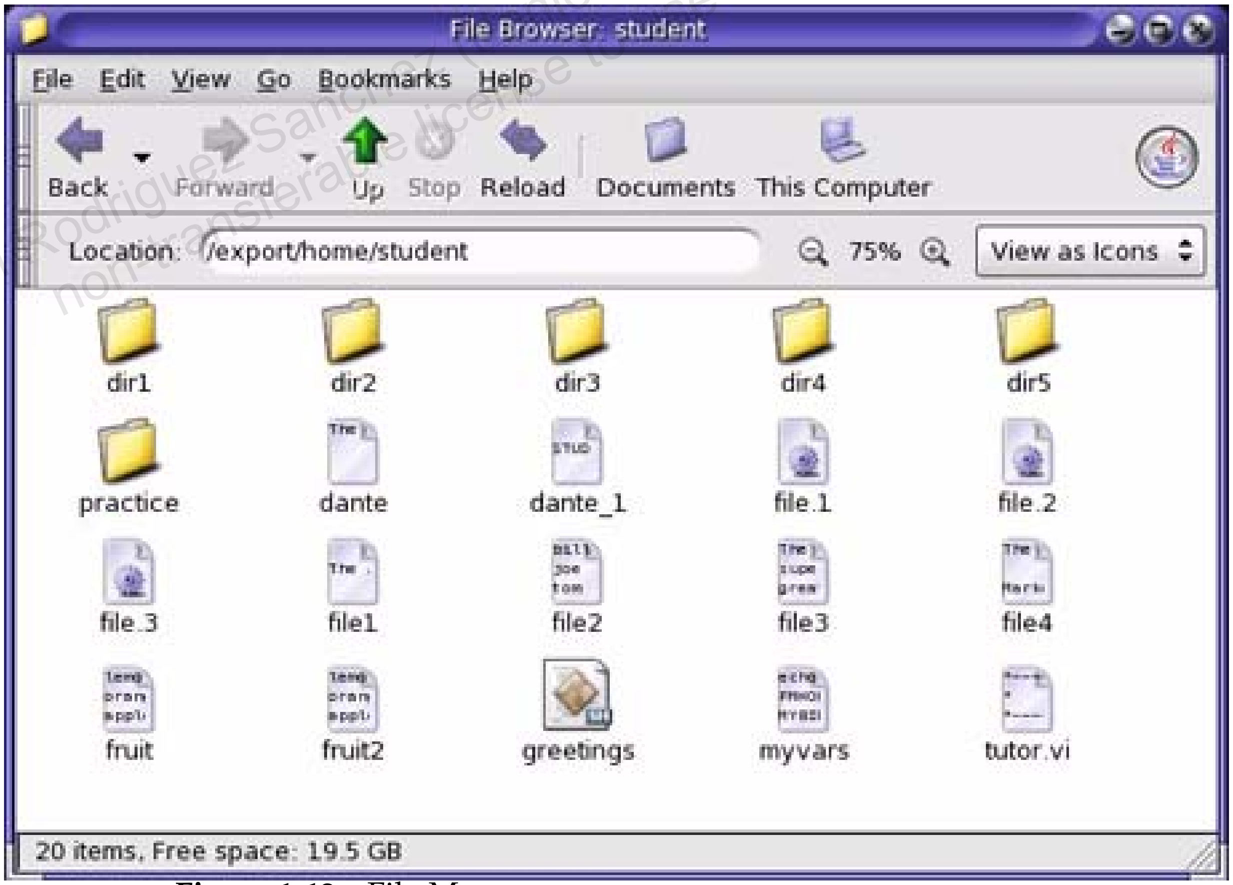
Managing Files Using the File Manager

The File Manager allows you to organize files into a hierarchical structure of directories and subdirectories. When you open the File Manager, the default view lists the contents of your home directory. From that directory, you can move up and down in the hierarchy to view other directories.

To open the File Manager:

- Double click the Documents icon on the desktop.
- Click Launch > Open Recent > Open Documents Folder.

The pathname lists the path through the hierarchy that you must follow to locate a file or directory. The pathname describes the location of a file or directory from the root (`/`) directory or from the parent directory. When you specify the location of a file or directory starting with a slash (`/`) or the root (`/`) directory, it is an absolute pathname. When you specify the location of a file or directory starting with its parent directory, it is a relative pathname. Figure 1-12 shows the pathname and a list of contents in the `student` directory.



/export/home/student

dir1 dir2 dir3 dir4 dir5
practice dante dante_1 file.1 file.2
file.3 file1 file2 file3 file4
fruit fruit2 greetings myvars tutor.vi

20 items, Free space: 19.5 GB

Figure 1-12 File Manager

To move a file from one directory to another, position the mouse pointer over the file icon, hold down the left mouse button, and drag the icon to the appropriate directory icon. When the file icon is positioned over the directory icon, release the left mouse button, and the file moves to that directory.

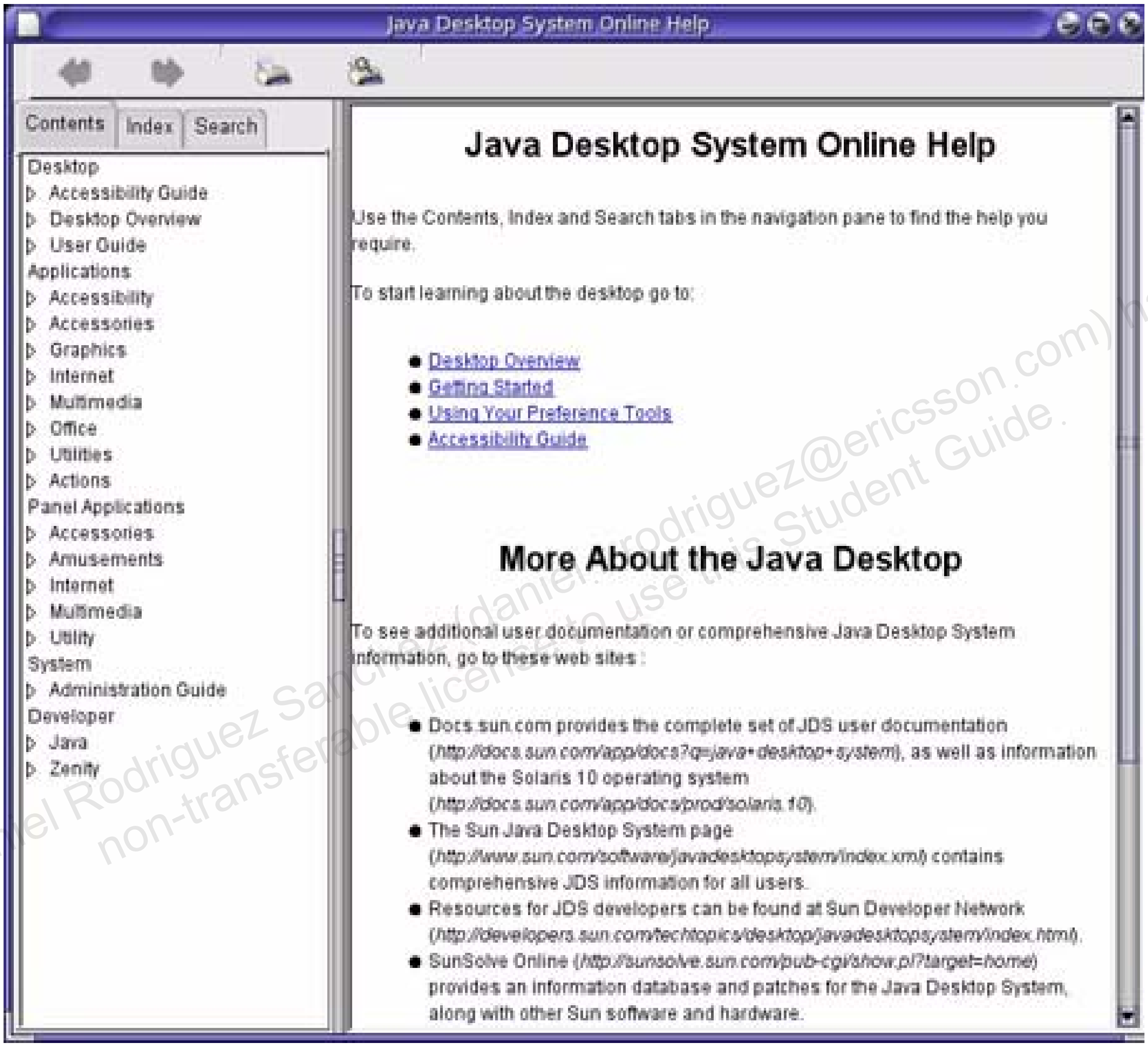
Using Help Resources

The Java Desktop System provides help if you want to find out more about the following:

- Java Desktop System
- Panel applications
- Desktop applications

To access the Java Desktop System help:

1. Click Launch > Help to launch the Help window. Figure 1-13 shows the Java Desktop System Online Help window.



Java Desktop System Online Help

Use the Contents, Index and Search tabs in the navigation pane to find the help you require.

To start learning about the desktop go to:

- Desktop Overview
- Getting Started
- Using Your Preference Tools
- Accessibility Guide

More About the Java Desktop

To see additional user documentation or comprehensive Java Desktop System information, go to these web sites :

- Docs.sun.com provides the complete set of JDS user documentation (*http://docs.sun.com/app/docs?q=java+desktop+system*), as well as information about the Solaris 10 operating system (*http://docs.sun.com/app/docs/prod/solaris.10*).
- The Sun Java Desktop System page (*http://www.sun.com/software/javadesktopsystem/index.xml*) contains comprehensive JDS information for all users.
- Resources for JDS developers can be found at Sun Developer Network (*http://developers.sun.com/techtopics/desktop/javadesktopsystem/index.html*).
- SunSolve Online (*http://sunsolve.sun.com/pub-cgi/show.pl?target=home*) provides an information database and patches for the Java Desktop System, along with other Sun software and hardware.

Figure 1-13 Java Desktop Online Help

2. Use the Contents, Index, and Search tabs in the navigation pane to find the help you require.

To access help for Panel applications:

1. Right click on the panel application.
2. Choose Help.

To access help for Desktop applications:

1. Start the application
2. Perform one of the following:
   - Choose Help > Contents
   - Press F1

Exercise: Using the Solaris 10 OS

In this exercise, you complete the following tasks:

- Identify computer components
- Use password
- Log in using the Command Line Login option
- Invoke a Failsafe session
- Lock the desktop environment screen
- Change the desktop background
- Move files using drag-and-drop

Preparation

Log in or log out of your desktop environment session or system as required.

Remote Lab Data Center (RLDC)

In addition to being able to use local classroom equipment, this lab is designed to also use equipment located in a remote lab data center. Directions for accessing and using this resource can be found at:

`http://remotelabs.sun.com/`

Ask your instructor for the particular SSH (Secure Shell) configuration file that you should use to access the appropriate remote equipment for this exercise.

Task 1 – Identifying Computer Components

To identify components of the computer, complete the following steps:

1. The four main components of the hardware are:
   a. RAM ________________________
   b. CPU ________________________
   c. I/O devices ________________________
   d. Operating system ________________________
   e. Hard disk ________________________
   f. Software ________________________
2. Name the three main components of the SunOS operating system.

   ____
   ____
   ____
3. Define a virtual memory operating system. Name a virtual memory operating system.

   ____
   ____
   ____
4. Distinguish between the Sun OS 5.*x* operating system and the Solaris 10 OS.

   ____
   ____
   ____
5. List the three primary shells supported by the Solaris OS.

   ____
   ____
   ____

6. Name the default shell for the root (/) user in the Solaris 10 OS.

   ____

7. Match the following terms with their descriptions:

__ Shell	a. Core of the Solaris OS
__ Kernel	b. Command interpreter
__ Hard Disk	c. Storage device

Task 2 – Using Password

To log in using your desktop environment login screen and change your password, complete the following steps:

1. Obtain a user name and password from your instructor.
2. Log into the system using the desktop environment Login screen.
   a. Type your user name, and press Return.
   b. Type your password, and press Return.
3. Right click on the desktop environment background.

   The Workspace menu appears.
4. Select Open Terminal from this menu.

   A terminal window appears.
5. Use the `passwd` command to change your password to `mypass1`.

```
$ passwd
passwd: Changing password for username
Enter existing login password:
New password:
Re-enter new passwd:
passwd: passwd successfully changed for username
$
```

6. Close the terminal window.
7. Click Launch > Log Out <*username*> to log out of the desktop environment session.

   A logout confirmation window appears.
8. Click OK or press Return to continue to log out.

9. Enter the following incorrect user name and password on the Login screen:

   User name: `student2`

   Password: `wrong`

   The following message appears:

```
Login incorrect; please try again
```

10. Click and hold down the Options button on the Login screen.
11. Select the Reset Login Screen option from the Options menu.

Note – After successful completion of Step 11, reset the default password of the user before moving to the next task.

Task 3 – Logging in Using the Command Line Login Option

To log in using the command-line, complete the following steps:

1. Click and hold down the Options button on the Login screen.
2. Select Command Line Login.

   The following message appears:

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

3. Press Return to display the following login prompt:

```
hostname console login:
```

4. Type your user name, and press Return.
5. Type your new password, and press Return.
6. At the shell prompt, type `exit`.

   The *`hostname`* `console login:` prompt reappears. After a short time, the Login screen reappears.

Task 4 – Invoking a Failsafe Session

To invoke a Failsafe session, complete the following steps:

1. Click and hold down the Options button on the Login screen.
2. Select Session.

   The Session menu appears.
3. Select Failsafe Session.
4. Type your user name, and press Return.
5. Type your password, and press Return.

   A terminal window appears.
6. Using the mouse, move the cursor into the terminal window.
7. At the shell prompt, type `exit`.

Task 5 – Locking the Desktop Environment Screen

Perform the following steps:

1. What are the two ways in which you can secure your login session?

   ____

   ____

2. To lock the desktop environment screen, complete the following steps:
   a. Log into the desktop using the default desktop environment session.
   b. Click Launch > Lock Screen to lock the current desktop environment session.
   c. Type your password and press Return to regain access to your desktop environment session.

Task 6 – Changing the Desktop Background

To change the desktop background, complete the following steps:

1. Log into the desktop using the default desktop environment session.
2. Click Launch > Preferences > Desktop Preferences > Display > Desktop Background option.

   The Desktop Background Preferences window appears.
3. Select the background you want to use from the Desktop Wallpaper list.
4. Click Close.

Task 7 – Moving Files Using Drag-and-Drop

To drag and drop a file into a directory in the desktop environment, you must create a directory and a file.

1. Double-click the Documents icon on the Desktop to open the File Browser window.
2. In the File Browser window, click File > Create Folder.

   A folder icon with the name, `untitled folder`, appears in edit mode.
3. Name this folder as `Folder1`
4. Click File > Create Document > Empty File in the File Browser window.

   A file icon with the name, `new file`, appears in edit mode.
5. Name this file as `File1`
6. Move your pointer over the `File1` icon.
7. Click and hold the left mouse button.
8. Drag the `File1` icon over the `Folder1` icon, and release the mouse button.

   The `File1` file is moved in to the `Folder1` folder.
9. Move your pointer over the `Folder1` icon.
10. Right click on `Folder1` icon.

    A menu appears.
11. Select Move to Trash.

    The `Folder1` folder is moved to Trash.

Exercise Summary

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

- Experiences
- Interpretations
- Conclusions
- Applications

Exercise Solutions: Using the Solaris 10 OS

This section provides solutions to the following tasks.

Task 1 – Identifying Computer Components

To identify components of the computer, complete the following steps:

1. The four main components of the hardware are:

a.	RAM	___X__________________
b.	CPU	___X__________________
c.	I/O devices	___X__________________
d.	Operating system	______________________
e.	Hard disk	___X__________________
f.	Software	______________________

2. Name the three main components of the SunOS operating system.
   - *The kernel*
   - *The shell*
   - *The directory hierarchy*
3. Define a virtual memory operating system. Name a virtual memory operating system.

   A virtual memory operating system enables the execution of programs, as if more memory is available than physically exists in RAM. SunOS 5.x is a virtual memory operating system.
4. Distinguish between the SunOS 5.*x* operating system and the Solaris 10 OS.

   *The SunOS 5.*x *operating system consists of the kernel, the shell, and the directory hierarchy.*

   *The Solaris 10 OS consists of the SunOS 5.*x *operating system, Open Network Computing (ONC+) software, and the Solaris desktop environment GUI.*
5. List the three primary shells supported by the Solaris OS.
   - *Bourne shell*
   - *C shell*
   - *Korn shell*

6. Name the default shell for the root (/) user in the Solaris 10 OS.

   Bourne shell.

7. Match the following terms with their descriptions:

b	Shell	a. Core of the Solaris OS
a	kernel	b. Command interpreter
c	Hard disk	c. Storage device

Task 5 – Locking the Desktop Environment Screen

Perform the following steps:

1. What are the two ways in which you can secure your login session?

   Locking the screen and exiting the session.

Notes:

Module 2

Using Command-Line Features and Online Help Resources

Objectives

This module describes how to use the command-line syntax and display the online manual pages.

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

- Construct and execute commands from the command-line
- Use online documentation

Figure 2-1 is the course map that shows how this module fits into the current instructional goal.

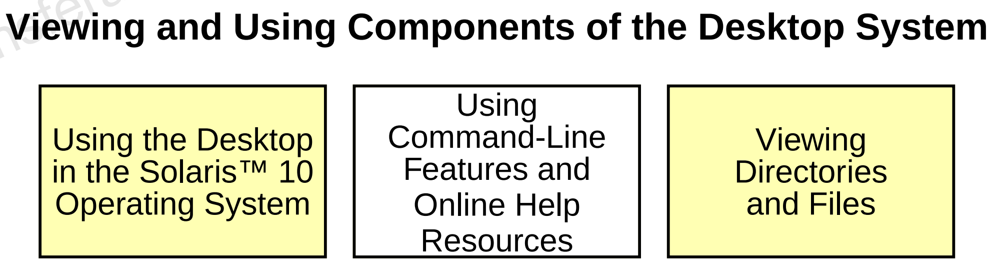

Viewing and Using Components of the Desktop System
Using the Desktop in the Solaris™ 10 Operating System
Using Command-Line Features and Online Help Resources
Viewing Directories and Files

Figure 2-1 Course Map

Constructing and Executing Commands From a Command-Line

You can use Solaris system commands to instruct the system to perform specific tasks. You run command-line commands in a terminal window. Command-line commands can exist with or without options and arguments. A Solaris command *syntax* is the structure and order of the components: command name, options, and arguments.

Note – Commands in the Solaris environment are case sensitive.

To open a terminal window, complete the following steps:

1. Move the cursor to an open space on the desktop.
2. Right click on the desktop.

   A popup menu appears.
3. Click Open Terminal.

   A terminal window appears with a shell prompt. The shell is ready to receive a command.

Command-Line Syntax

You can change the behavior of command functions with options and arguments. Table 2-1 describes these components of a command.

Table 2-1 System Command Components

Item	Description
command	Specifies what the system does (executes).
option	Specifies how the command runs (a modifier). Options start with a dash (-) character.
argument	Specifies what is affected (a file, a directory, or text).

The following example shows the syntax of a command:

```
command options arguments
```

Using Commands

Some examples of basic commands are `uname`, `date`, `cal`, and `clear`.

The `uname` command provides information about the system. By default, when you use the `uname` command, the name of the current operating system appears.

To display the operating system information, enter the following command:

```
$ uname
SunOS
$
```

The `date` command displays the system's current date and time. To display the date and time, enter the following command:

```
$ date
Tue Feb 105 18:22:19 MDT 2009
$
```

The `cal` command displays a calendar for the current month and year. To display the calendar, enter the following command:

```
$ cal
    February 2009
 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

$
```

To clear the terminal window, enter the following command:

```
$ clear
$
```

Using Commands With Options

Adding options to a command alters the type of information displayed. Command options are case sensitive.

You can use more than one option with a command. You can list multiple options separately or combined after one dash (-).

Note – Use of a dash (-) preceding an option is command-specific. Check the appropriate man page for the command for more information. Also, options are command-specific. It is important to check the appropriate man page for proper use of options.

The following examples show the uname command with two options. The -i option shows the name of the hardware platform, and the -n option prints the host name of the local system. The following example shows the use of the uname command with separate options.

```
$ uname -i
SUNW,Sun-Blade-1500
$ uname -n
host1
$
```

The following example shows the uname command with two separate options. The -s option shows the name of the operating system. The -r option shows the operating system release level.

```
$ uname -s -r
SunOS 5.10
$
```

The following example shows the uname command with two combined options.

```
$ uname -rs
SunOS 5.10
$
```

The following example of the uname command with the -a option shows information currently available from the system.

```
$ uname -a
SunOS host1 5.10 Generic_139555-08 sun4u sparc SUNW,Sun-Blade-1500
$
```

Using Commands With Arguments

Arguments enable you to define exactly what you want to do with a command. The following example shows the `cal` command with two arguments. The first argument, `12`, specifies the month to be viewed. The second argument, `2009`, specifies the year to be viewed.

```
$ cal 12 2009
    December 2009
 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

$
```

Using Commands With Options and Arguments

The following examples show the `ls` command, the `ls` command with an option, the `ls` command with an argument, and the `ls` command with an option and an argument. The `ls` command lists the files in a directory. The `-l` option provides additional information about the files. The ***`filename`*** argument specifies the file to be viewed.

```
$ ls
dante      dir3       file.2     file3      greetings
dante_1    dir4       file.3     file4      myvars
dir1       dir5       file1      fruit      practice
dir2       file.1     file2      fruit2     tutor.vi

$ ls -l
total 94
total 94
-rw-r--r--   1 student  class      1319 Feb 6 09:25 dante
-rw-r--r--   1 student  class       368 Feb 6 09:25 dante_1
drwxr-xr-x   5 student  class       512 Feb 6 09:25 dir1
drwxr-xr-x   4 student  class       512 Feb 6 09:25 dir2
drwxr-xr-x   3 student  class       512 Feb 6 09:25 dir3
drwxr-xr-x   2 student  class       512 Feb 6 09:25 dir4
drwxr-xr-x   2 student  class       512 Feb 6 09:25 dir5
-rw-r--r--   1 student  class         0 Feb 6 09:25 file.1
-rw-r--r--   1 student  class         0 Feb 6 09:25 file.2
-rw-r--r--   1 student  class         0 Feb 6 09:26 file.3
-rw-r--r--   1 student  class      1610 Feb 6 09:26 file1
-rw-r--r--   1 student  class       105 Feb 6 09:26 file2
-rw-r--r--   1 student  class       218 Feb 6 09:26 file3
-rw-r--r--   1 student  class       137 Feb 6 09:26 file4
-rw-r--r--   1 student  class        57 Feb 6 09:26 fruit
-rw-r--r--   1 student  class        57 Feb 6 09:26 fruit2
-rw-r--r--   1 student  class        59 Feb 6 09:26 greetings
-rw-r--r--   1 student  class        67 Feb 6 09:26 myvars
drwxr-xr-x   2 student  class       512 Feb 6 09:26 practice
-rw-r--r--   1 student  class     28738 Feb 6 09:26 tutor.vi

$ ls dante
dante

$ ls -l dante
-rw-r--r--   1 student  class      1319 Feb 6 09:25 dante
$
```

Entering Multiple Commands on a Single Command-Line

You can enter multiple commands on a single command-line using a semicolon (`;`) to separate each command. The shell recognizes the semicolon as a command separator. The shell executes each command from left to right when you press Return. The command format for multiple commands is:

```
command option argument;command option argument
```

The following example shows two commands separated by a semicolon.

```
$ date;uname
Tue Feb 10 18:27:48 MDT 2009
SunOS
$
```

The following example shows three commands separated by a semicolon. The `cal` command has two arguments, followed by the `date` command, and the `uname` command with two options.

```
$ cal 12 2009; date; uname -rs
    December 2009
 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

Tue Feb 10 18:28:08 MDT 2009
SunOS 5.10
$
```

Using Online Documentation

The Solaris OS provides online manual pages, which describe commands and their usage. Additionally, online help resources provide general desktop information.

Displaying the Online Manual Pages

The online Reference Manual (man) pages provide detailed descriptions of Solaris commands and how to use them. Use the `man` command to display the man page entry that explains a given command.

The syntax of the `man` command is:

```
man command
man option command
man option filename
```

For example, to display the man pages for the `uname` command, use the command:

```
$ 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 truncated)
```

Scrolling in Man Pages

Table 2-2 shows the keys on the keyboard that you use to control the scrolling capabilities when you are in the man pages.

Table 2-2 Keys to Control Scrolling in Man Pages

Scrolling Keys	**Action**
Space bar	Displays the next screen of a man page
Return	Displays the next line of a man page
`b`	Moves back one full screen
`/`*`pattern`*	Searches forward for a pattern
`n`	Finds the next occurrence of a pattern after you have used `/`*`pattern`*
`h`	Provides a description of all scrolling capabilities
`q`	Quits the `man` command and returns to the shell prompt.

Searching the Man Pages

Two ways to search for information in the man pages are searching by section or by keyword.

Searching Man Pages by Section

You can search within a specific section of the man pages using the `man` command with the `-s` option.

The online man-page entries are organized into sections based on the type or use of the command or file. For example, Section 1 contains user commands, while Section 4 contains information about various file formats.

You can use the `man intro` command to view descriptive information about sections contained in the man pages.

The syntax for looking up a specific section of the man pages is the `man` command with the `-s` option, followed by the section number, and the command or file name.

For example:

```
man -s number command
```

or

```
man -s number filename
```

The bottom portion of a man page, titled `SEE ALSO`, lists other commands or files related to the man page. The number in parentheses reflects the section where the man page is located. You can use the `man` command with the `-l` option to list the man pages that relate to the same command or file name.

The following example shows the SEE ALSO part of the man page for passwd(1). Note that there is also a man page entry for passwd in Section 4.

```
SEE ALSO
     at(1),   batch(1),   finger(1),   login(1),    nistbladm(1),
     orcron(1M),      domainname(1M),     eeprom(1M),     id(1M),
     mkpwdict(1M), passmgmt(1M), pwconv(1M), su(1M), useradd(1M),
     userdel(1M),     usermod(1M),    crypt(3C),    getpwnam(3C),
     getspnam(3C),  getusershell(3C),  nis_local_directory(3NSL),
     pam(3PAM),   loginlog(4),   nsswitch.conf(4),   pam.conf(4),
     passwd(4),  policy.conf(4),  shadow(4),  shells(4),   attri-
     butes(5),          environ(5),          pam_authtok_check(5),
     pam_authtok_get(5),   pam_authtok_store(5),   pam_dhkeys(5),
     pam_ldap(5),      pam_unix_account(5),     pam_unix_auth(5),
     pam_unix_session(5)
```

To view the online man page for the passwd file, use the following commands:

```
$ man -l passwd
passwd (1)     -M /usr/man
passwd (4)     -M /usr/man
$

$ man -s 4 passwd
Reformatting page.  Please Wait... done

File Formats                                        passwd(4)

NAME
     passwd - password file

SYNOPSIS
     /etc/passwd

DESCRIPTION
     The file /etc/passwd is a local source of information about users'
     accounts.  The password file can be used in conjunction with
     other password sources, including the NIS maps passwd.byname and
     passwd.bygid and the NIS+ table  passwd. Programs use the
     getpwnam(3C) routines to access this information.
... (output truncated)
```

Searching Man Pages by Keyword

When you are not sure of the name of a command, you can search for man page entries that are related using the `man` command with the `-k` option and a keyword. The `man` command output provides a list of commands and descriptions that contain the keyword. The syntax for using the command to conduct a keyword search is:

```
man -k keyword
```

To view commands containing the keyword `calendar`, use the command:

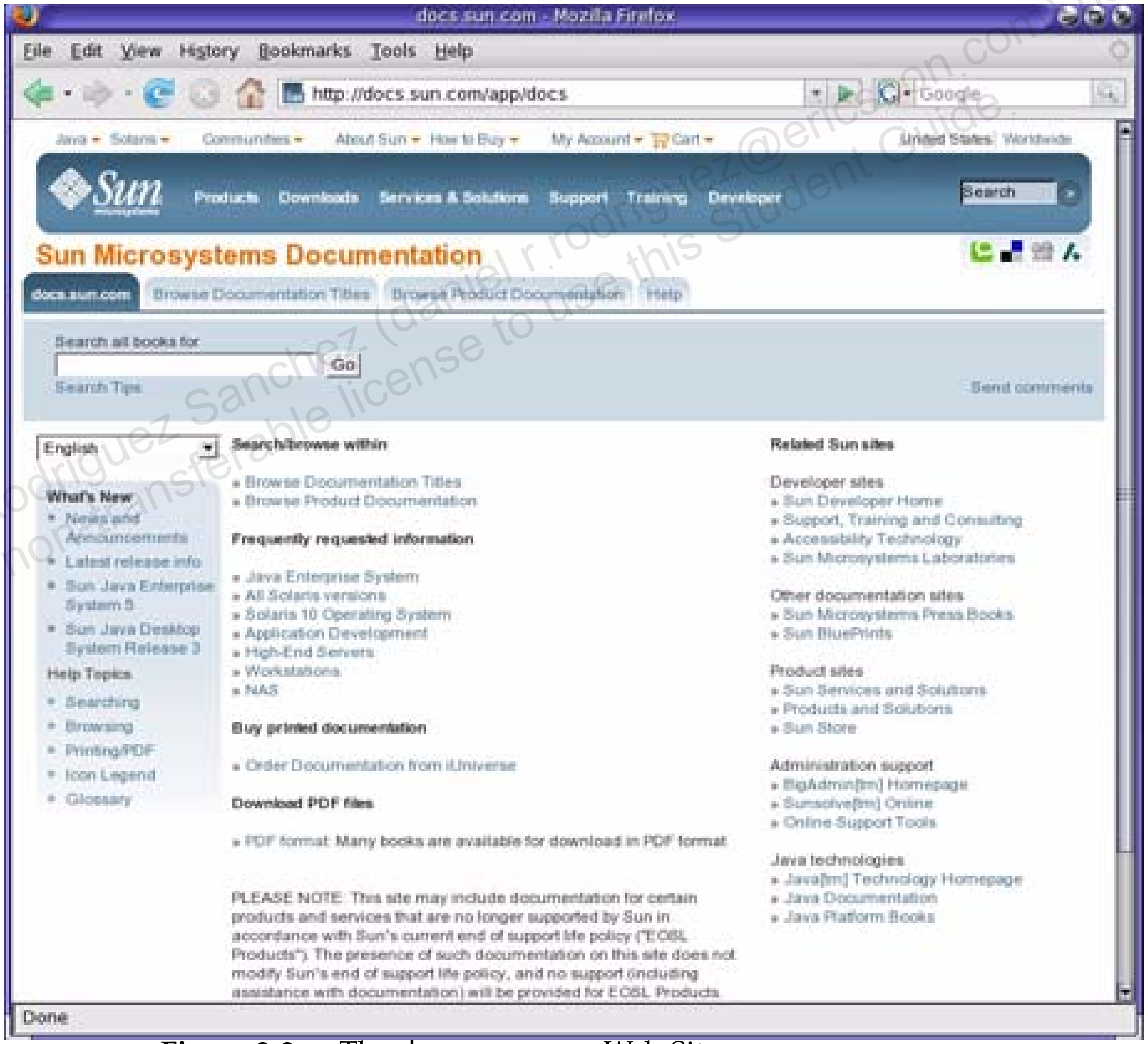
```
$ man -k calendar
cal             cal (1)         - display a calendar
calendar        calendar (1)    - reminder service
difftime        difftime (3c)   - computes the difference between two calendar times
mktime          mktime (3c)     - converts a tm structure to a calendar time
$
```

Using Online Product Documentation

You use a Web browser to access the online product documentation. To start a web browser in the desktop environment:

- Click Launch > Firefox Web Browser
- Click Launch > Applications > Internet > Firefox Web Browser.

To access comprehensive information on Sun products, visit the `http://docs.sun.com` Web site using your web browser. You can search by subject, collection title, product category, and keyword. Figure 2-2 shows a portion of this web page.


docs.sun.com - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://docs.sun.com/app/docs
Google
Java Solaris Communities About Sun How to Buy My Account Cart United States Worldwide
Sun microsystems
Products Downloads Services & Solutions Support Training Developer
Search
docs.sun.com
Browse Documentation Titles
Browse Product Documentation
Help
Go
Search Tips
Send comments
English
Done

Sun Microsystems Documentation

Search all books for

What's New
- News and Announcements
- Latest release info
- Sun Java Enterprise System 5
- Sun Java Desktop System Release 3

Help Topics
- Searching
- Browsing
- Printing/PDF
- Icon Legend
- Glossary

Search/browse within
- Browse Documentation Titles
- Browse Product Documentation

Frequently requested information
- Java Enterprise System
- All Solaris versions
- Solaris 10 Operating System
- Application Development
- High-End Servers
- Workstations
- NAS

Buy printed documentation
- Order Documentation from iUniverse

Download PDF files
- PDF format: Many books are available for download in PDF format

PLEASE NOTE: This site may include documentation for certain products and services that are no longer supported by Sun in accordance with Sun's current end of support life policy ("EOSL Products"). The presence of such documentation on this site does not modify Sun's end of support life policy, and no support (including assistance with documentation) will be provided for EOSL Products.

Related Sun sites

Developer sites
- Sun Developer Home
- Support, Training and Consulting
- Accessibility Technology
- Sun Microsystems Laboratories

Other documentation sites
- Sun Microsystems Press Books
- Sun BluePrints

Product sites
- Sun Services and Solutions
- Products and Solutions
- Sun Store

Administration support
- BigAdmin[tm] Homepage
- Sunsolve[tm] Online
- Online Support Tools

Java technologies
- Java[tm] Technology Homepage
- Java Documentation
- Java Platform Books

Figure 2-2 The `docs.sun.com` Web Site

Exercise: Using Command-Line Features

In this exercise, you perform the following tasks:

- Construct commands
- Use command-line features
- Scroll through online man pages
- Use online documentation

Preparation

No special preparation is required for this exercise.

RLDC

In addition to being able to use local classroom equipment, this lab is designed to use equipment located in a remote lab data center. Directions for accessing and using this resource can be found at:

`http://remotelabs.sun.com/`

Ask your instructor for the particular SSH configuration file that you should use to access the appropriate remote equipment for this exercise.

Task 1 – Constructing Commands

To construct commands, arrange the following terms to show the correct syntax of a command:

options arguments command

____

Task 2 – Using Command-Line Features

To use command-line features, complete the following steps:

1. Open a terminal window, if you do not already have one open.

   Name the two components of command-line syntax that can enhance the capability of a command.

   ___

2. Enter the command to display information about the operating system and the workstation name.

   ___

3. Execute the `man uname` command. List the functions of the `-s` and `-r` options to the `uname` command.

   ___

   ___

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

   ___

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

   ___

4. Enter the command to display the current time and date on your system.

   ___

5. Display this month's calendar. What command did you enter?

   ___

6. What special character is used to separate commands on a single command-line?

   ___

7. Enter the command to clear your terminal window.

   ___

Task 3 – Scrolling Through Online Man Pages

To learn how to search for online documentation, list the keys or the functions of the following keys in the empty cells in Table 2-3 to control scrolling the online man pages:

Table 2-3 Keys to Control Scrolling in the Man Pages

Keys	**Function**
	Displays the next screen of a man page
Return	
`/pattern`	
	Quits the `man` command

Task 4 – Using Online Documentation

To access online documentation, complete the following steps:

1. Using the `-k` option with the `man` command, find the man page that describes how to clear a terminal window (use the keyword `clear`). How do you enter this command on the command-line?

   ____

2. Clear the terminal window.
3. What Web site enables you to browse Sun product documentation?

   ____

Exercise Summary

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

- Experiences
- Interpretations
- Conclusions
- Applications

Exercise Solutions: Using Command-Line Features

This section provides solutions to the following tasks.

Task 1 – Constructing Commands

To construct commands, arrange the following terms to show the correct syntax of a command:

```
options arguments command
```

`command options arguments`

Task 2 – Using Command-Line Features

To use command-line features, complete the following steps:

1. Open a terminal window, if you do not already have one open.

   Name the two components of command-line syntax that can enhance the capability of a command.

   Options and arguments

2. Enter the command to display information about the operating system and the workstation name.

```
$ uname -a
```

3. Execute the `man uname` command. List the functions of the `-s` and `-r` options to the `uname` command.

   The `-r` option prints the operating system release level. The `-s` option prints the name of the operating system.

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

      The operating system name, for example, SunOS.

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

      The operating system release level, for example, 5.10.

4. Enter the command to display the current time and date on your system.

```
$ date
```

5. Display this month's calendar. What command did you enter?

```
$ cal
```

6. What special character is used to separate commands on a single command-line?

   The semicolon (`;`).

7. Enter the command to clear your terminal window.

```
$ clear
```

Task 3 – Scrolling Through Online Man Pages

To learn how to search for online documentation, list the keys or the functions of the following keys in the empty cells in Table 2-4 to control scrolling the online man pages:

Table 2-4 Keys to Control Scrolling in the Man Pages

Key	**Function**
Space bar	*Displays the next screen of a* `man` *page*
Return	*Displays the next line of a* `man` *page*
`/pattern`	*Searches forward for a pattern*
q	*Quits the* `man` *command and returns to the shell prompt.*

Task 4 – Using Online Documentation

To access online documentation, complete the following steps:

1. Using the `-k` option with the `man` command, find the man page that describes how to clear a terminal window (use the keyword `clear`). How do you enter this command on the command-line?

```
$ man -k clear
```

2. Clear the terminal window.

```
$ clear
```

3. What Web site enables you to browse Sun product documentation?

   The Web site `http://docs.sun.com`*.*

Module 3

Viewing Directories and Files

Objectives

This module introduces files and directories and describes how to display directories and directory contents. The module also describes how to view and print files.

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

- Work with directories
- Work with files
- Print files

Figure 3-1 is the course map that shows how this module fits into the current instructional goal.

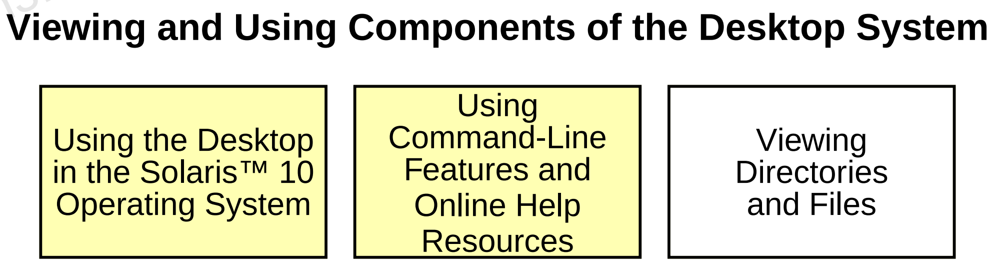

Viewing and Using Components of the Desktop System
Using the Desktop in the Solaris™ 10 Operating System
Using Command-Line Features and Online Help Resources
Viewing Directories and Files

Figure 3-1 Course Map

Viewing Directories

You can use various commands to display the current directory, view the contents of a directory, and change directories.

Directory Terms

The following sections describe basic terms used in conjunction with directories.

Directory

A *directory* is a list of references to objects, which can include files, sub-directories, and symbolic links. Each reference consists of two components: a name and a number. The *name* of the object is used to identify and access the object. The *number* specifies the inode in which information about the object is stored.

Inode

An *inode* is a list of information relating to a particular object, such as a file, directory, or symbolic link. The information held by the inode includes the type of object about which the inode holds information, permissions, ownership information, and the locations in which data is stored.

Determining the Current Directory

The `pwd` command identifies the directory that you are currently accessing.

To display the current working directory, enter the `pwd` command.

```
$ pwd
/export/home/student
$
```

Displaying the Directory Contents

You can use the `ls` command to display the contents of a directory. To list the files and directories within the specified directory, enter the `ls` command without arguments.

The syntax for the `ls` command is:

```
ls -options filename
```

To list the contents of the `student` directory, enter the `ls` command.

```
$ ls
dante      dir3      file.2    file3     greetings
dante_1    dir4      file.3    file4     myvars
dir1       dir5      file1     fruit     practice
dir2       file.1    file2     fruit2    tutor.vi
$
```

You can view a hierarchy of the files and directories in the `/export/home/student` directory, as Figure 3-2 shows.

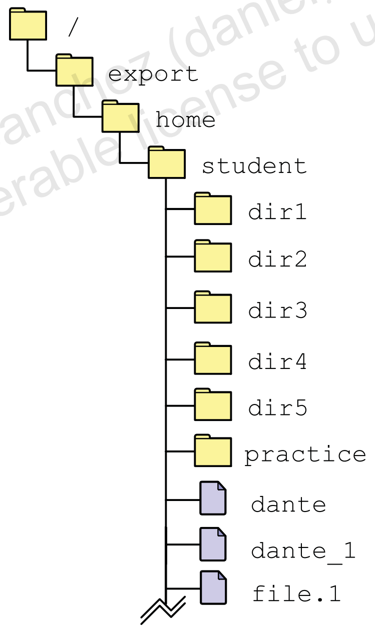


Figure 3-2 Directory List

To display the contents of a specific directory within the current working directory, enter the `ls` command followed by a directory name.

```
$ ls dir1
coffees   fruit   trees
```

To display the contents of a directory that is not in the current working directory, enter the `ls` command with the complete path to that directory.

```
$ ls /export/home/student/dir2
beans    notes    recipes
$
```

Displaying Hidden Files

Some files are hidden from view when you use the `ls` command. Hidden files often contain information that customizes your work environment. You can use the `ls -a` command to list all files in a directory, including hidden files. The file names of hidden files begin with a period (`.`).

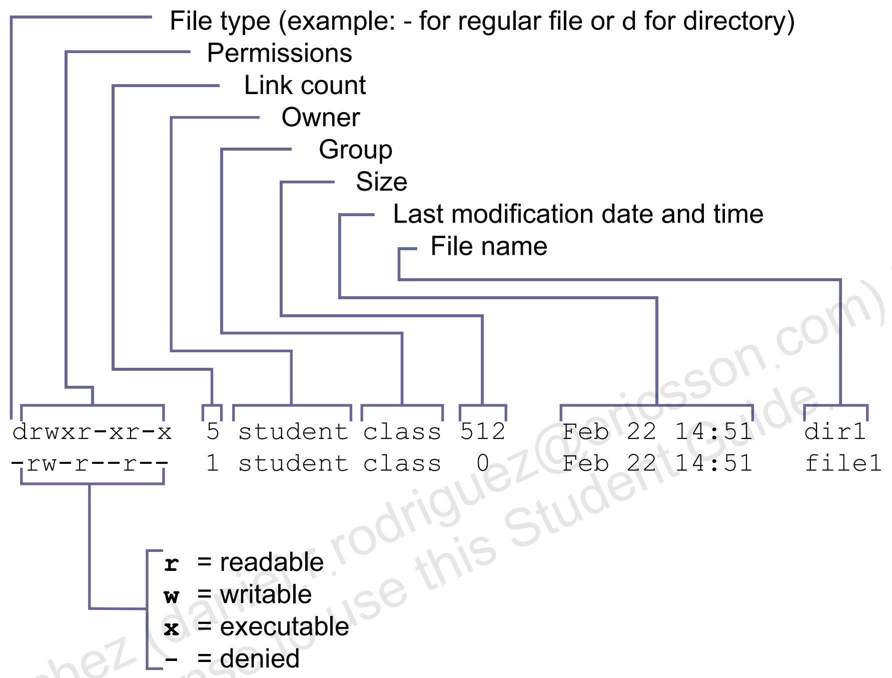
To display hidden files, enter the `ls -a` command.

```
$ ls -a
.                  .gnome2_private    dante             file2
..                 .gtkrc-1.2-gnome2  dante_1           file3
.ICEauthority      .metacity          dir1              file4
.Xauthority        .mozilla           dir2              fruit
.bash_history      .nautilus          dir3              fruit2
.dt                .netbeans          dir4              greetings
.dtprofile         .recently-used     dir5              myvars
.gconf             .rhosts            file.1            practice
.gconfd            .sh_history        file.2            tutor.vi
.gnome             .softwareupdate    file.3
.gnome2            .sunw              file1
$
```

A single period (`.`) represents the current working directory. The double period (`..`) represents the parent directory, which contains the current working directory.

Displaying a Long List

You can use the `ls -l` command to view detailed information about the contents of a directory. The output of the `ls -l` command displays a long listing of file information as Figure 3-3 shows.



File type (example: - for regular file or d for directory)
Permissions
Link count
Owner
Group
Size
Last modification date and time
File name
drwxr-xr-x 5 student class 512 Feb 22 14:51 dir1
-rw-r--r-- 1 student class   0 Feb 22 14:51 file1
r = readable
w = writable
x = executable
- = denied

Figure 3-3 Long List File Information

For a long listing of the contents of the student directory, enter the ls -l command, from the student directory.

```
$ ls -l
-rw-r--r--   1 student  class       1319 Feb 6 09:25 dante
-rw-r--r--   1 student  class        368 Feb 6 09:25 dante_1
drwxr-xr-x   5 student  class        512 Feb 6 09:25 dir1
drwxr-xr-x   4 student  class        512 Feb 6 09:32 dir2
drwxr-xr-x   3 student  class        512 Feb 6 09:25 dir3
drwxr-xr-x   2 student  class        512 Feb 6 09:25 dir4
drwxr-xr-x   2 student  class        512 Feb 6 09:25 dir5
-rw-r--r--   1 student  class          0 Feb 6 09:25 file.1
-rw-r--r--   1 student  class          0 Feb 6 09:25 file.2
-rw-r--r--   1 student  class          0 Feb 6 09:26 file.3
-rw-r--r--   1 student  class       1610 Feb 6 09:26 file1
-rw-r--r--   1 student  class        105 Feb 6 09:26 file2
-rw-r--r--   1 student  class        218 Feb 6 09:26 file3
-rw-r--r--   1 student  class        137 Feb 6 09:26 file4
-rw-r--r--   1 student  class         57 Feb 6 09:26 fruit
-rw-r--r--   1 student  class         57 Feb 6 09:26 fruit2
-rw-r--r--   1 student  class         59 Feb 6 09:26 greetings
-rw-r--r--   1 student  class         67 Feb 6 09:26 myvars
drwxr-xr-x   2 student  class        512 Feb 6 09:25 practice
-rw-r--r--   1 student  class      28738 Feb 6 09:26 tutor.vi
$
```

To view detailed information on the contents of the dir1 directory, enter the ls -l dir1 command from the student directory.

```
$ ls -l dir1
total 6
drwxr-xr-x   3 student  class        512 Feb 6 09:30 coffees
drwxr-xr-x   2 student  class        512 Feb 6 09:30 fruit
drwxr-xr-x   2 student  class        512 Feb 6 09:30 trees
$
```

Figure 3-4 shows the list of directories and files in the `dir1` directory.

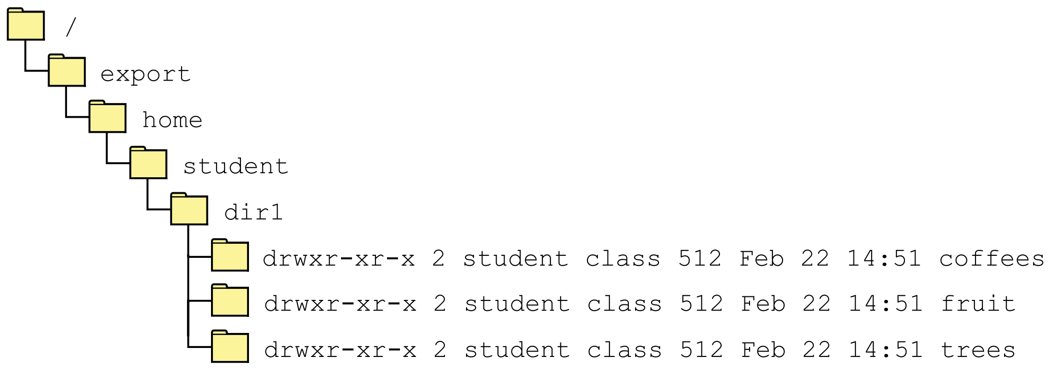

/
export
home
student
dir1
drwxr-xr-x 2 student class 512 Feb 22 14:51 coffees
drwxr-xr-x 2 student class 512 Feb 22 14:51 fruit
drwxr-xr-x 2 student class 512 Feb 22 14:51 trees

Figure 3-4 Long List of Files and Directories

Displaying Individual Directories

You can use the `ls -ld` command to view detailed information about a directory without viewing its contents.

To obtain detailed directory information for the `dir1` directory, enter the `ls -ld` command.

```
$ ls -ld dir1
drwxr-xr-x   5 student  class       512 Feb 6 09:30 dir1
$
```

Displaying a Recursive List

You can use the `ls -R` command to display the contents of a directory and the contents of all of the directory's subdirectories. This type of list is known as a recursive list.

To view a recursive list of the contents of the `dir1` directory, enter the `ls -R dir1` command.

```
$ ls -R dir1
dir1:
coffees  fruit    trees

dir1/coffees:
beans    brands    nuts

dir1/coffees/beans:
beans

dir1/fruit:

dir1/trees:
$
```

Displaying File Types

You can use either the `ls -F` command or the `file` command to display file types.

Using the `ls -F` Command

Table 3-1 shows the symbols used in the `ls -F` command output.

Table 3-1 File Type Symbols

Symbol	File Type
/	Directory
*	Executable
(None)	Plain text file or American Standard Code for Information Interchange (ASCII)
@	Symbolic link

The following example shows the output of the `ls -F` command.

```
$ ls -F
dante       dir3/       file.2      file3       greetings
dante_1     dir4/       file.3      file4       myvars
dir1/       dir5/       file1       fruit       practice/
dir2/       file.1      file2       fruit2      tutor.vi
$
```

Note – A *symbolic link* is a special type of file that points to another file or directory.

Using the `file` Command

You can use the `file` command to determine certain file types. If you know the file type, you can decide which command or program to use to read the file.

Note – Normally, in the Solaris environment file suffixes do not indicate the file type unless it was generated by an application, such as StarOffice (`.sxw`) or Framemaker (`.fm`).

The output from the `file` command can be one of the following:

- Text – Text files include American Standard Code for Information Interchange (ASCII) text, English text, command text, and executable shell scripts.
- Data – Data files are created by programs. The `file` command indicates the type of data file, such as a FrameMaker document, if the type is known. The `file` command indicates that the file is a data file if the type is unknown.
- Executable or binary – Executable files include 32-bit executable and extensible linking format (ELF) code files and other dynamically linked executable files. Executable files are commands or programs.

The syntax for the `file` command is:

```
file filenames
```

To view the file type for the `dante` file, enter the `file` command and specify the name of the file.

```
$ file dante
dante:          English text
$
```

Changing Directories

When working within the directory hierarchy, you always have a current working directory. When you initially log into the system, the current directory is set to your home directory. You can change your current working directory at any time using the `cd` command.

The syntax for the `cd` command is:

```
cd directory
```

When you use the `cd` command without options or arguments, the current working directory changes to your home directory.

To change directories from the `student` directory to the `dir1` directory, use the `cd` command:

```
$ pwd
/export/home/student
$ cd dir1
$ pwd
/export/home/student/dir1
$
```

Using Pathname Abbreviations

You can use pathname abbreviations to easily navigate or refer to directories on the command-line. Table 3-2 shows the pathname abbreviations.

Table 3-2 Pathname Abbreviations

Symbol	Pathname
`.`	Current or working directory
`..`	Parent directory, the directory directly above the current working directory

To move to the parent directory for `dir1`, enter the `cd ..` command.

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

Confirm the current working directory using the `pwd` command.

```
$ pwd
/export/home/student
$
```

Figure 3-5 shows the `dir1` directory and its parent directory, the `student` directory.

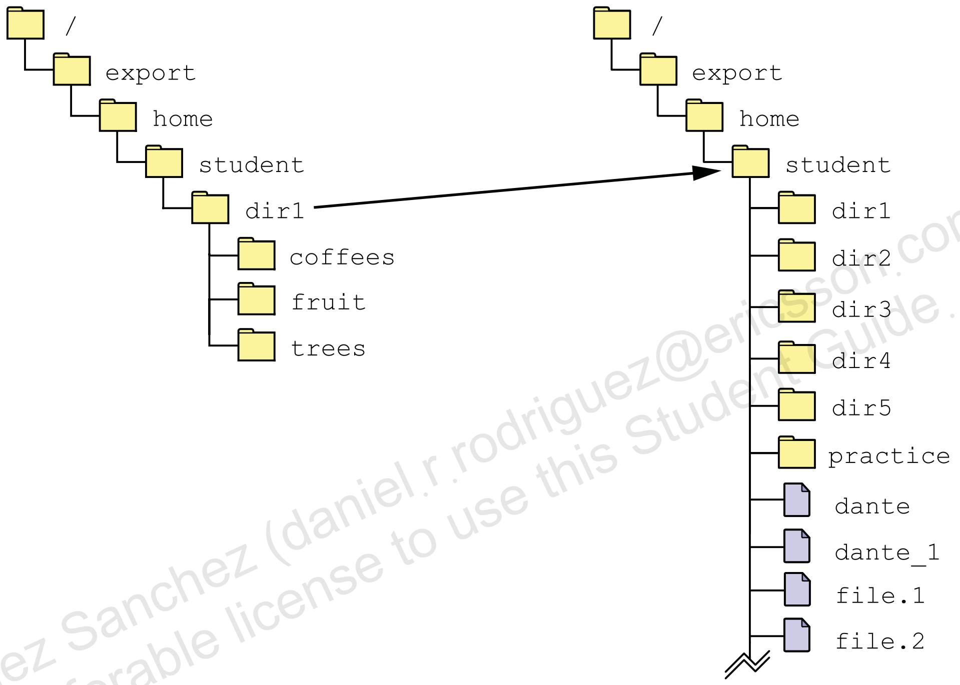


Figure 3-5 Navigating the Directory Tree

You can move up multiple levels of the directory hierarchy using the `cd ..` command followed by a slash (`/`).

```
$ pwd
/export/home/student
$ cd ../../..
$ pwd
/
$
```

Moving Around the Directory Hierarchy

You can either use a relative pathname or an absolute pathname to move around the directory hierarchy.

A relative pathname lists the directories in the path relative to the current working directory. An absolute pathname lists all the directories in the path, starting with the root (/) directory.

To change directories using a relative pathname, enter the cd command with the pathname that starts from the current working directory, student.

```
$ cd
$ cd dir1
$ pwd
/export/home/student/dir1
$ cd ../dir2
$ pwd
/export/home/student/dir2
$ cd
$ cd dir1/coffees
$ pwd
/export/home/student/dir1/coffees
$
```

To change directories using an absolute pathname, enter the cd command with the complete pathname from the root (/) directory.

```
$ cd
$ cd /export/home/student/dir1/coffees
$ pwd
/export/home/student/dir1/coffees
$
```

Returning to Your Home Directory

The home directory of a regular user is where the user is placed after logging in. The user can create and store files in the home directory.

Note – The directory named `/export/home` is the default directory that contains the home directories of regular users. However, you can configure systems to use the `/home` directory, instead of the `/export/home` directory, as the default directory that holds the home directories of regular users.

Often the name of a user's home directory is the same as the user's login name. For example, if your user login name is `student`, your home directory would be `export/home/student` or `/home/student`.

You can return to your home directory using two methods:

- Perform the `cd` command without arguments.

```
$ cd
$ pwd
/export/home/student
$
```

- Perform the `cd` command with the absolute pathname to your home directory.

```
$ cd /export/home/student
$
```

To navigate to a user's home directory, enter the `cd` command with a tilde (~) character in front of the user name. The tilde (~) character is an abbreviation that equates to the absolute pathname of the user.

Note – The tilde (~) character is a shell facility and is not available in all shells.

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

You can also use the tilde (~) character to represent your home directory in a relative path. The tilde (~) in the following example represents the `student` home directory.

```
$ cd ~/dir1/fruit
$
```

You can also use the tilde (~) character to navigate to another user's home directory.

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

Viewing Files

There are different commands available that enable you to view file content in a read-only format or to display information about a file. These commands include the `cat` command, the `more` command, the `tail` command, the `head` command and the `wc` command.

Viewing Files Using the cat Command

The `cat` command displays the contents of one or more text files on the screen. The `cat` command displays the contents of the entire file without pausing.

Note – Before you attempt to open a file with the `cat` command, it is recommended that you first run the `file` command to determine the file's type.

```
$ 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.
... (output truncated)
```

Note – Do not use the `cat` command to read binary files. Using the `cat` command to read binary files can cause a terminal window to freeze. If your terminal window freezes, close the terminal window, and open a new terminal window.

Viewing Files Using the `more` Command

To view or page through the contents of a long text file, use the `more` command. The `more` command displays the contents of a text file one screen at a time. The following message appears at the bottom of each screen.

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

where *n%* is the percentage of the file that has been displayed.

When the `--More--(n%)` prompt appears, you can use the keys described in Table 3-3 to scroll through the file.

Table 3-3 Scrolling Keys for the `more` Command

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

When the entire file has been displayed, the shell prompt appears.

The syntax of the `more` command is:

```
more filename
```

To display the first screen of the dante file, use the more command.

```
$ more 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.  There is only one Dante, as we recognize
one Raphael, one Michelangelo, and one Galileo.

Who is this Dante, whom T.S. Eliot calls "the most universal of poets
in the modern languages?"


YOUTH

Exact details about his youth are few indeed.  He was born in the city
of Florence, Italy, in May of 1265.  His family was of noble origin
by modest means and modest social standing.  His great grandfather died
in the Second Crusade in the Holy Land.  Dante was a thoughtful, quiet,
rather sad young man.  His mother, whose name was Bella (beautiful) died
while he was still a child.  His father remarried a certain Lapa who
didn’t shower too much love and affection on her stepson.

About Dante’s personal appearance, his friend Boccaccio wrote, "Our poet
was of medium height.  He had a long face, an aquiline nose, large jaws,
and his lower lip was more prominent than the upper.  He was slightly
--More--(90%)
```

Viewing File Content Using the head Command

The head command displays the first 10 lines of a file. You can change the number of lines displayed using the *-n* option. The *-n* option displays the number of lines (*n*) starting at the beginning of the file.

The syntax for the head command is:

```
head -n filename
```

To display the first six lines of the /usr/dict/words file, enter the head command with the *-n* option set to 6.

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

Viewing File Content Using the tail Command

The tail command displays the last 10 lines of a file. You can change the number of lines displayed using the *-n* or *+n* options. The *-n* option displays *n* lines from the end of the file. The *+n* option displays the file from line *n* to the end of the file.

The syntax for the tail command is:

```
tail -n filename
```

or

```
tail +n filename
```

To display the last five lines of the `/usr/dict/words` file, enter the `tail` command with the `-n` option set to 5.

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

To display line 25136 through the end of the `/usr/dict/words` file, enter the `tail` command with the `+n` option set to 25136.

```
$ tail +25136 /usr/dict/words
Zorn
Zoroaster
Zoroastrian
zounds
z’s
zucchini
Zurich
zygote
$
```

Displaying Line, Word, and Character Counts

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

The syntax for the `wc` command is:

```
wc -options filenames
```

Table 3-4 shows the options that you can use with the `wc` command.

Table 3-4 Options for the `wc` Command

Option	Description
`-l`	Line count
`-w`	Word count
`-c`	Byte count
`-m`	Character count

When you use the `wc` command without options, the output displays the number of lines, words, and characters contained in the file.

To display the number of lines, words, and characters in the `dante` file, use the `wc` command.

```
$ wc dante
    32     223    1319 dante
$
```

To display the number of lines in the `dante` file, enter the `wc` command with the `-l` option.

```
$ wc -l dante
    32 dante
$
```

Printing Files

You can use the `lp` command, the `lpstat` command, and the `cancel` command to submit print requests to a destination, to display the status of all user print requests, and to cancel print requests.

Using the `lp` Command

The `lp` command is located in the `/usr/bin` directory. The `lp` command submits a print job to the default printer or to another printer by specifying the printer name. Perform one of the following commands:

```
$ /usr/bin/lp filename
```

or

```
$ /usr/bin/lp -d printername filename
```

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

Table 3-5 describes some of the options that you can use with the `lp` command.

Table 3-5 Options for the `lp` Command

Option	Description
`-d` *`destination`*	Specifies the desired printer. The `-d` *`destination`* option is not necessary if printing to the default printer.
`-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.

To print the `dante` file located in your home directory on the default printer, use the `lp` command without options.

```
$ lp /export/home/student/dante
request id is printerA-4 (1 file(s))
$
```

Note – The example above assumes that you have a default printer set up that is called `printerA`. To check the name of your default printer, use the `lpstat -d` command.

To print a file on the `printerB` printer, which is not the default printer, enter the `lp` command with the `-d` *`destination`* option set to `printerB`.

```
$ lp -d printerB /export/home/student/dante
request id is printerB-2 (1 file(s))
$
```

Using the `lpstat` Command

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

The syntax for the `lpstat` command is:

```
lpstat -options printer
```

Table 3-6 describes some of the options for the `lpstat` command.

Table 3-6 Options for the `lpstat` Command

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

To display the status of all print requests, enter the `lpstat` command with the `-o` option.

```
$ lpstat -o
printerA-5    student  391        Feb 6  16:30     on printerA
printerB-3    user2    551        Feb 6  16:45      filtered
$
```

Table 3-7 describes the information in the status response of the `lpstat` command.

Table 3-7 Status Information for the `lpstat` Command

Status	**Definition**
`Request-ID`	Name of the printer and job number
`User-ID`	Name of the user accessing the printer
`File Size`	Output size in bytes
`Date/Time`	Current date and time
`Status`	Status of the print request

To display requests in the queue for `printerB`, enter the `lpstat` command followed by the printer name.

```
$ lpstat printerB
printerB-3     user2     551         Feb 6  16:45    on printerB
$
```

To determine the status of all configured printers, enter the `lpstat` command with the `-t` option.

```
$ 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 Feb 7 09:43 2009
_default accepting requests since Feb 8 08:20 2009
printerA accepting requests since Feb 8 08:20 2009
printer printerB is idle. enabled since Feb 7 09:43 2009. available.
printer _default is idle. enabled since Feb 8 08:20 2009. available.
printer printerA is idle. enabled since Feb 8 08:20 2009. available.
$
```

To determine which printers are configured on the system, enter the `lpstat` command with the `-s` option.

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

To display which printers are accepting requests, enter the `lpstat` command with the `-a` option.

```
$ lpstat -a
printerB accepting requests since Feb 7 09:43 2009
_default accepting requests since Feb 8 08:20 2009
printerA accepting requests since Feb 8 08:20 2009
$
```

Using the `cancel` Command

The `cancel` command enables you to cancel previously sent print requests, using the `lp` command. You can use the `lpstat` command to identify the *`printer`* associated with your print request.

The syntax for the `lp cancel` command is:

```
cancel Request-ID
```

or

```
cancel -u username
```

Note – When you use the desktop environment Printer Manager, it appears that you can cancel another user's print job, but the job is re-displayed in the Print Manager the next time the Print Manager is refreshed. To identify the *`Request-ID`* for your print request, use the `lpstat` *`printername`* command.

```
$ lpstat printerB
printerB-3 user3 630          Feb 6 16:45
printerB-4 user3 631          Feb 6 16:45
printerB-5 user3 632          Feb 6 16:47
$
```

To cancel the print request, enter the `cancel` command followed by the *`Request-ID`* argument.

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

To remove all requests made by `user3`, enter the command:

```
$ cancel -u user3
printerB-3: cancelled
printerB-4: cancelled
$
```

As the `root` user, you can cancel all print requests owned by all users. If you are not a `root` user, you can only remove your own print jobs.

Exercise: Accessing Files and Directories

In this exercise, you use the commands described in this module to list and change directories. All the directories are situated within your home directory.

Preparation

Ensure that a default printer is configured for your system. You can use the `lpstat -d` command to view the default printer information. Ask your instructor for any help required.

RLDC

In addition to being able to use local classroom equipment, this lab was designed to also use equipment located in a remote lab data center. Directions for accessing and using this resource can be found at:

`http://remotelabs.sun.com/`

Ask your instructor for the particular SSH configuration file that you should use to access the appropriate remote equipment for this exercise.

Tasks

To access files and directories, complete the following steps. Write the commands that you use to perform each task in the space provided.

1. Display your current working directory.

   ____

2. Change to your home directory.

   ____

3. Display the contents of your current working directory.

   ____

4. Display all files, including any hidden files.

   ____

5. Display a long list of the contents of the current working directory.

   ____

6. Display the file types in your current working directory.

7. Change to the `dir1` directory.

8. Display a long list of the contents of the current working directory.

9. Change to the `fruit` directory.

10. Change to the `planets` directory available under `$HOME/dir3` directory using the relative pathname.

11. Return to your home directory.

12. Change directories to the `dir1` directory using an absolute pathname.

13. Return to your home directory.

14. Change to the `/etc` directory using a relative pathname.

15. Return to your home directory.

16. Display a long list of the contents of the current working directory.

17. Display the contents of the `fruit` file using the `cat` command.

18. Under what circumstances must you refrain from using the `cat` command to view the contents of a file?

19. Display the contents of the `fruit` file and the `fruit2` file using a single command.

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

____

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

____

22. Distinguish between the `head` and `tail` commands.

____

____

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

____

24. Print the `dante_1` file to the default printer.

____

25. What action occurs when you enter the `lp -m` command for the default printer?

____

26. What does the ~ symbol represent?

____

Exercise Summary

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

- Experiences
- Interpretations
- Conclusions
- Applications

Exercise Solutions: Accessing Files and Directories

To access files and directories, complete the following steps. Write the commands that you use to perform each task in the space provided.

1. Display your current working directory.

```
$ pwd
```

2. Change to your home directory.

```
$ cd
$ pwd
```

3. Display the contents of your current working directory.

```
$ ls
```

4. Display all files, including any hidden files.

```
$ ls -a
```

5. Display a long list of the contents of the current working directory.

```
$ ls -l
```

6. Display the file types in your current working directory.

```
$ ls -F
```

7. Change to the `dir1` directory.

```
$ cd dir1
$ pwd
```

8. Display a long list of the contents of the current working directory.

```
$ ls -l
```

9. Change to the `fruit` directory.

```
$ cd fruit
$ pwd
```

10. Change to the `planets` directory available under `$HOME/dir3` directory using the relative pathname.

```
$ cd ../../dir3/planets
$ pwd
```

11. Return to your home directory.

```
$ cd
$ pwd
```

12. Change to the `dir1` directory available under `$HOME` directory using an absolute pathname.

```
$ cd /export/home/student/dir1
$ pwd
```

or

```
$ cd ~/dir1
```

13. Return to your home directory.

```
$ cd;pwd
```

14. Change to the `/etc` directory using a relative pathname.

```
$ pwd
/export/home/student
$ cd ../../../etc
$ pwd
/etc
```

15. Return to your home directory.

```
$ cd;pwd
```

16. Display a long list of the contents of the current working directory.
17. Display the contents of the `fruit` file using the `cat` command.

```
$ cat fruit
```

18. Under what circumstances must you refrain from using the cat command to view the contents of a file?

    Do not use the `cat` *command to view binary files.*

19. Display the contents of the `fruit` file and the `fruit2` file using a single command.

```
$ cat fruit fruit2
```

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

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

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

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

22. Distinguish between the head and `tail` commands.

    The head *command displays the first 10 lines of a file, whereas the* `tail` *command displays the last 10 lines of a file.*

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

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

24. Print the `dante_1` file to the default printer.

```
$ lp dante_1
```

25. What action occurs when you enter the `lp -m` command for the default printer?

    The `lp -m` *command sends a mail message to you after the print job is complete.*

26. What does the ~ symbol represent?

    The ~ symbol represents the user's home directory.

Notes:

Working with Files and Directories

Objectives

This module introduces commands that enable you to change the contents of directories in the Solaris OS, by using commands to create, copy, rename, and remove files and directories.

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

- Copy files and directories
- Move and rename files and directories
- Create files and directories
- Remove files and directories
- Use symbolic links

Figure 4-1 is the course map that shows how this module fits into the current instructional goal.

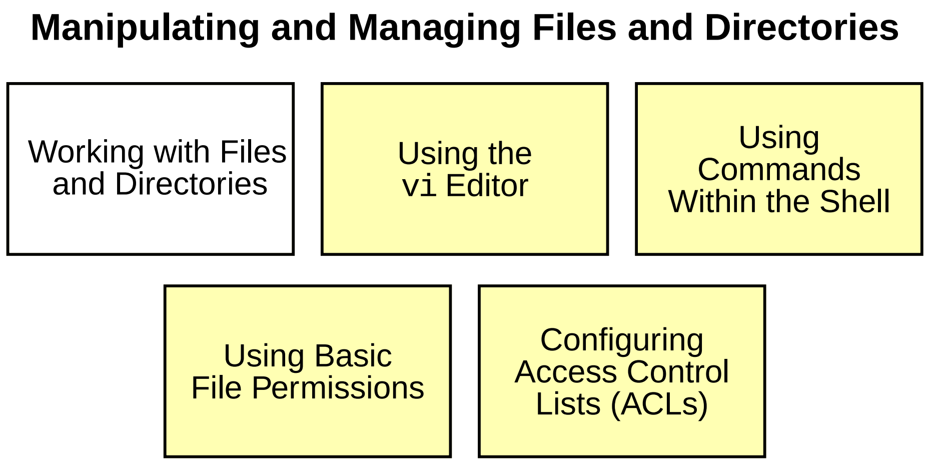

Manipulating and Managing Files and Directories
Working with Files and Directories
Using the vi Editor
Using Commands Within the Shell
Using Basic File Permissions
Configuring Access Control Lists (ACLs)

Figure 4-1 Course Map

Copying Files and Directories

You can copy a file or a directory from one place to another using the `cp` command. The `cp` command copies files to a specified target file or directory. The target file or directory is the last argument in the command.

Caution – The `cp` command is a destructive command if not used with the correct option.

Copying Files

You can use the `cp` command to copy the contents of a file to another file. You can also use the `cp` command to copy multiple files. You can use the `cp` command with options and modify the functions of the command. For example, using the `-i` (interactive) option prevents overwriting existing files when copying files. When you use the `cp` command with the `-i` option, it prompts you for confirmation before the copy overwrites an existing target.

The syntax for the `cp` command when copying files is:

```
cp -option(s) source(s) target
```

The *`source`* option is a file. The *`target`* option can be a file or a directory.

Table 4-1 describes some options you can use with the `cp` command when you are copying files and directories.

Table 4-1 Options for the `cp` Command

Option	Description
`-i`	Prevents you from accidentally overwriting existing files or directories
`-r`	Includes the contents of a directory, including the contents of all subdirectories, when you copy a directory

Copying a File Within a Directory

To copy a file to a new file name in the same directory, use the `cp` command with the name of the source file and the target file.

To copy the file named `file3` to a new file named `feathers`, within the `student` directory, enter the following commands:

```
$ cd
$ pwd
/export/home/student
$ ls
dante      dir3       file.2     file3      greetings
dante_1    dir4       file.3     file4      myvars
dir1       dir5       file1      fruit      practice
dir2       file.1     file2      fruit2     tutor.vi
$ cp file3 feathers
$ ls
dante      dir3       file.1     file2      fruit2     tutor.vi
dante_1    dir4       file.2     file3      greetings
dir1       dir5       file.3     file4      myvars
dir2       feathers   file1      fruit      practice
$
```

To copy the `feathers` file to a new file named `feathers_6`, within the `student` directory, enter the `cp` command.

```
$ cp feathers feathers_6
$ ls
dante       dir3        feathers_6  file1       fruit       practice
dante_1     dir4        file.1      file2       fruit2      tutor.vi
dir1        dir5        file.2      file3       greetings
dir2        feathers    file.3      file4       myvars
$
```

Copying Multiple Files

To copy multiple files to a different directory, use the `cp` command with multiple file names for the source and use a single directory name for the target.

To copy the `feathers` file and the `feathers_6` file from the `student` directory into the `dir1` subdirectory, enter the following commands:

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

Figure 4-2 shows how you can copy the `feathers` file and the `feathers_6` file to the `dir1` directory.

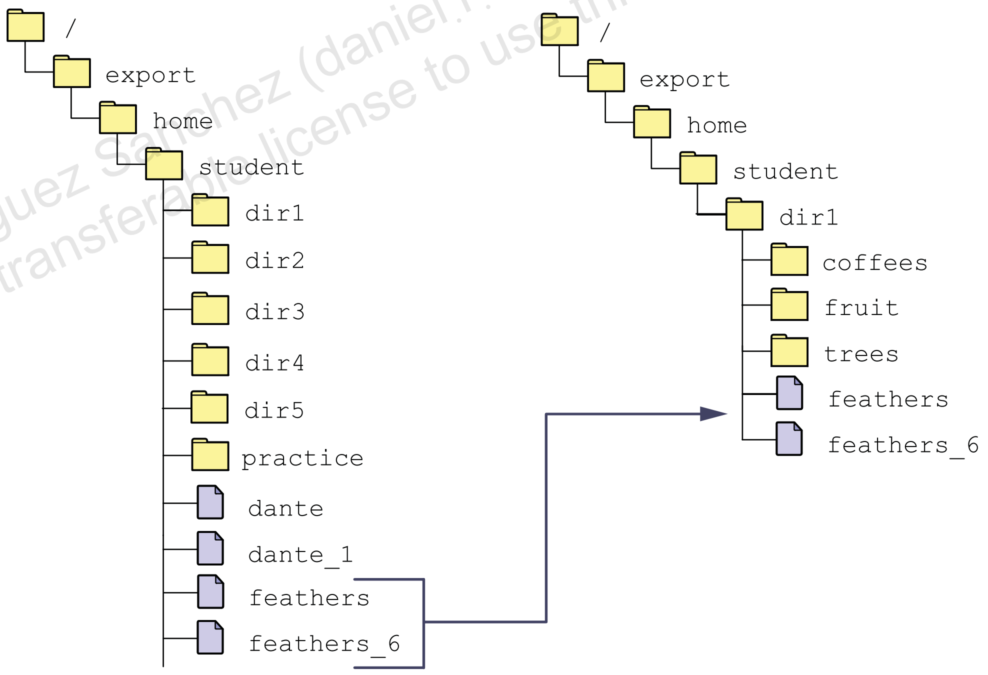


Figure 4-2 Copying Multiple Files

Preventing Overwrites to Existing Files While Copying

To prevent overwriting existing files when copying new files, you can enter the `cp` command with the `-i` option. When you use the `-i` option, the system prompts you for a confirmation before overwriting existing files with new ones.

- A `yes` response permits the overwrite.
- A `no` response prevents the `cp` command from overwriting the target file.

To copy the `feathers` file to the `feathers_6` file, enter the `cp -i` command. Because the `feathers_6` file already exists, the overwrite prompt appears.

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

Copying Directories

You can use the `cp` command with the `-r` option to copy a directory recursively. If the target directory does not exist, the `cp -r` command creates a new directory with that name. If the target directory exists already, the `cp -r` command creates a new sub-directory with that name, below the destination directory.

The syntax for the `cp` command when copying directories is:

```
cp -option sources target
```

The *source* option is one or more directory names. The *target* option is a single directory name. To copy the contents of the `dir3` directory to a new directory named `dir10`, use the `cp -r` command. Both directories are in the `student` directory.

```
$ cd
$ pwd
/export/home/student
$ ls dir3
planets
$ cp dir3 dir10
cp:  dir3: is a directory
$ cp -r dir3 dir10
$ ls dir10
planets
$ ls dir3
planets
$
```

To copy the `planets` directory from the `dir3` directory to a new directory called `constellation`, use the `cp -r` command. The `constellation` directory is not in the current working directory.

```
$ cd
$ pwd
/export/home/student
$ cd dir3
$ cp -r planets ../dir4/constellation
$ ls ../dir4/constellation
mars   pluto
$ cd
$
```

Moving and Renaming Files and Directories

You can use the mv command to:

- Move files and directories within the directory hierarchy
- Rename existing files and directories

The mv command does not affect the contents of the files or directories being moved or renamed.

Caution – The mv command is a destructive command if not used with the correct option.

Moving and Renaming a File

You can use the mv command to rename a single file within the same directory.

For example, use the mv command to rename the dante file to dantenew and then back again.

```
$ pwd
/export/home/student
$ mv dante dantenew
$ ls
dante_1     dir2        feathers    file.3      file4       myvars
dantenew    dir3        feathers_6  file1       fruit       practice
dir1        dir4        file.1      file2       fruit2      tutor.vi
dir10       dir5        file.2      file3       greetings
$ mv dantenew dante
$ ls
dante       dir2        feathers    file.3      file4       myvars
dante_1     dir3        feathers_6  file1       fruit       practice
dir1        dir4        file.1      file2       fruit2      tutor.vi
dir10       dir5        file.2      file3       greetings
$
```

Moving a File to Another Directory

You can use the mv command to move a file to a different directory. The syntax for the mv command is:

```
mv source target
```

The *source* option is the old file or directory name. The *target* option is the new file or directory name.

To move the brands file from the coffees directory into the student directory, enter the following commands.

```
$ cd ~/dir1/coffees
$ pwd
/export/home/student/dir1/coffees
$ ls
beans   brands  nuts
$ mv brands ~
$ ls
beans   nuts
$ cd
$ pwd
/export/home/student
$ ls -l brands
-rw-r--r--   1 student  class         0 Feb 6  2009 brands
$
```

Note – Unlike the copy command, the mv command moves your file or directory, and the original will no longer exist.

Prevent a File Overwrite With the -i Option

The -i option prompts you for confirmation to prevent you from overwriting existing file(s) by the new file(s).

```
mv -i source target
```

- A yes response permits the mv command to overwrite an existing file(s).
- A no response prevents the mv command from overwriting the existing file(s).

Moving a Directory and its Contents

You can use the `mv` command to move a directory and its contents to a different directory.

To move the `practice` directory and its contents into a brand new empty directory named `letters`, enter the following commands:

```
$ cd
$ pwd
/export/home/student
$ ls -l practice
-rw-r--r--   1 student  class         0 Feb 6   2009  mailbox
-rw-r--r--   1 student  class         0 Feb 6   2009  project
-rw-r--r--   1 student  class         0 Feb 6   2009  projection
-rw-r--r--   1 student  class         0 Feb 6   2009  research
-rw-r--r--   1 student  class         0 Feb 6   2009  results
$ mkdir letters
$ ls -l letters
total 0
$ mv practice letters
$ ls -l letters
drwxr-xr-x   2 student class        512 Feb 6 14:11 practice
$ ls -lR letters
letters:
total 2
drwxr-xr-x   2 student  class       512 Feb 6 14:12 practice

letters/practice:
total 0
-rw-r--r--   1 student  class         0 Feb 6  2009 mailbox
-rw-r--r--   1 student  class         0 Feb 6  2009 project
-rw-r--r--   1 student  class         0 Feb 6  2009 projection
-rw-r--r--   1 student  class         0 Feb 6  2009 research
-rw-r--r--   1 student  class         0 Feb 6  2009 results
$
```

When you move a single directory to a target directory that *does not exist*, you are actually renaming the current directory and its path.

When you move multiple directories to a target directory that does not exist, the following error message appears:

```
mv: target_directory not found.
```

Renaming a Directory

You can use the `mv` command to rename directories within the current directory.

Enter the following commands to rename the `dir5` directory to `dir5new`

```
$ cd
$ pwd
/export/home/student
$ mv dir5 dir5new
$ ls
brands      dir10       dir5new     file.2      file3       greetings
dante       dir2        feathers    file.3      file4       letters
dante_1     dir3        feathers_6  file1       fruit       myvars
dir1        dir4        file.1      file2       fruit2      tutor.vi
$ mv dir5new dir5
$ ls
brands      dir10       dir5        file.2      file3       greetings
dante       dir2        feathers    file.3      file4       letters
dante_1     dir3        feathers_6  file1       fruit       myvars
dir1        dir4        file.1      file2       fruit2      tutor.vi
$
```

Creating Files and Directories

You can create new files and directories within the directory hierarchy using the `touch` and `mkdir` commands. The `touch` command creates a new empty file, and the `mkdir` command creates a new directory.

Creating Empty Files

You can use the `touch` command to create an empty file. You can create multiple files on the same command-line. If the file name or directory name already exists, the `touch` command updates the modification time and access time to the current date and time.

The syntax for the `touch` command is:

```
touch filename
```

You can use absolute or relative pathnames on the command-line when creating new files.

To create an empty file named `space` in the `dir3` directory, enter the following commands:

```
$ pwd
/export/home/student
$ cd dir3
$ ls
planets
$ touch space
$ ls
planets  space
$
```

To create three empty files named `moon`, `sun`, and `cosmos` in the `dir3` directory, use the `touch` command.

```
$ touch moon sun cosmos
$ ls
cosmos   moon     planets  space    sun
$
```

Creating Directories

You can use the `mkdir` command with a *`directory_name`* to create directories. If the *`directory_name`* includes a pathname, use the `mkdir` command with the `-p` option. The command used with the `-p` option creates all of the non-existing parent directories that do not yet exist in the path to the new directory. The syntax for the `mkdir` command includes:

```
mkdir directory_name
```

and

```
mkdir -p directory_names
```

You can use absolute or relative pathnames on the command-line when creating new directories.

To create a new directory, named `Reports`, within the `student` directory, use the `mkdir` command.

```
$ cd
$ pwd
/export/home/student
$ mkdir Reports
$ ls -ld Reports
drwxr-xr-x   2 student  class        512 Feb 6 19:02 Reports
$
```

To create a new directory named `empty_directory` located inside a directory named `newdir`, use the `mkdir` command with the `-p` option. The `newdir` directory does not yet exist.

```
$ cd
$ pwd
/export/home/student
$ ls
Reports     dir10       feathers    file1       fruit2
brands      dir2        feathers_6  file2       greetings
dante       dir3        file.1      file3       letters
dante_1     dir4        file.2      file4       myvars
dir1        dir5        file.3      fruit       tutor.vi
$ mkdir -p newdir/empty_directory
$ ls newdir
empty_directory
$
```

Figure 4-3 shows two new directories named `Reports` and `newdir` in the `student` directory.

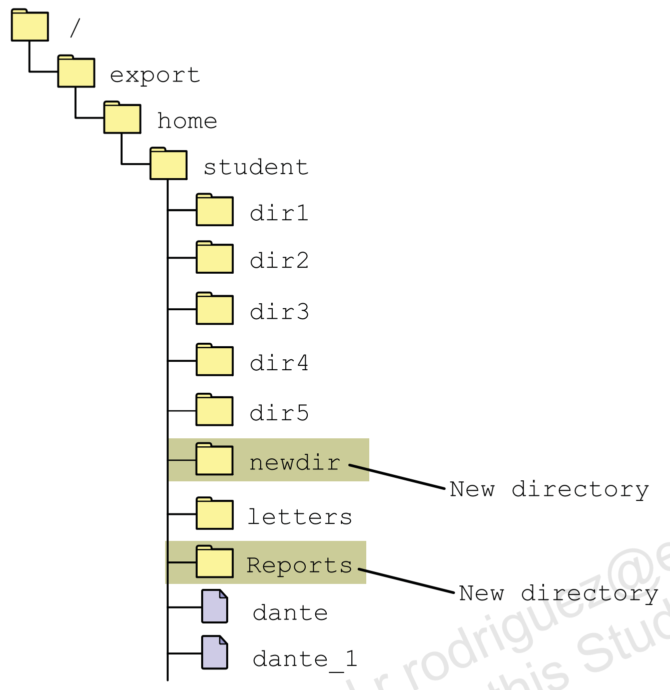


Figure 4-3 Creating a Directory

To create a `Weekly` directory in the `Reports` directory, enter the `mkdir` command.

```
$ mkdir Reports/Weekly
$ ls Reports
Weekly
$
```

To create the `dir1`, `dir2`, and `dir3` directories in the `Weekly` directory, enter the `mkdir` command.

```
$ cd Reports/Weekly
$ mkdir dir1 dir2 dir3
$ ls -F
dir1/ dir2/ dir3/
$
```

Removing Files and Directories

You can permanently remove files and directories from the directory hierarchy with the `rm` command. You can use the `rmdir` command to remove empty directories. Files and directories are removed without prompting you for confirmation.

Caution – The `rm` command is a destructive command if not used with the correct option.

Removing Files

You can use the `rm` command to remove a file or multiple files on the same command-line.

The syntax for the `rm` command is:

```
rm -option filename
```

To remove the file named `projection` from the `letters` directory, enter the following commands:

```
$ cd ~/letters
$ ls
mailbox    project    projection  research    results
$ rm projection
$ ls
mailbox   project   research  results
$
```

To remove the `research` file and the `project` file from the `letters` directory, enter the following commands:

```
$ pwd
/export/home/student/letters
$ ls
mailbox   project  research  results
$ rm research project
$ ls
mailbox  results
$
```

The `-i` option prompts you for confirmation before removing any file:

- A `yes` response completes the removal of the file.
- A `no` response prevents the `rm` command from removing the file.

To remove the contents of a directory, enter the `rm` command with the `-i` option.

```
$ cd
$ rm -i file*
rm: remove file1: (yes/no) ? Y
rm: remove file2: (yes/no) ?  Y
rm: remove file3: (yes/no) ? Y
rm: remove file4: (yes/no) ? Y
$ ls
$
```

Note – The asterisk (`*`) symbol used in the previous example is a wild card character, and it is described in more detail in Module 6, "Using Commands Within the Shell."

Removing Directories

There are two ways to remove directories. You can use the `rmdir` command to remove empty directories. You can use the `rm` command with the `-r` option to remove directories that contain files and subdirectories.

To remove a directory in which you are currently working, you must change to its parent directory.

Removing an Empty Directory

You can use the `rmdir` command to remove empty directories.

The syntax for the `rmdir` command is:

```
rmdir directories
```

If a directory is not empty, the `rmdir` command displays the following error message:

```
rmdir: directory "directory_name": Directory not empty
```

To remove a directory that is empty (contains no files or subdirectories), enter the following commands:

```
$ cd
$ pwd
/export/home/student
$ cd newdir
$ pwd
$ ls -F
empty_directory/
$ rmdir empty_directory
$ ls
$
```

Removing a Directory With Contents

You can use the `rm` command with the `-r` option to remove directories that contain files and subdirectories.

The syntax for the `rm` command is:

```
rm -options directories
```

If you do not use the `-r` option, the following error message appears:

```
rm: directoryname: is a directory.
```

Table 4-2 describes the options that you can use with the `rm` command when you are removing directories.

Table 4-2 Options for the `rm` Command

Option	Description
`-r`	Includes the contents of a directory and the contents of all subdirectories when you remove a directory
`-i`	Prevents the accidental removal of existing files or directories

The `-i` option prompts you for confirmation before removing a file or directory.

- A `yes` response completes the removal.
- A `no` response prevents the removal.

To remove the `letters` directory and its contents, use the `rm -r` command.

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

To interactively remove a directory and its contents, use the `rm -r` command with the `-i` option. Create a new directory called `rmtest` in your `/export/home/student` directory.

```
$ pwd
/export/home/student
$ mkdir rmtest
$ ls
Reports     dir10       feathers    file1       fruit2      tutor.vi
brands      dir2        feathers_6  file2       greetings
dante       dir3        file.1      file3       myvars
dante_1     dir4        file.2      file4       newdir
dir1        dir5        file.3      fruit       rmtest
$

$ cd rmtest
$ touch testfile
$ cd
$ rm -ir rmtest
rm: examine files in directory rmtest (yes/no)? y
rm: remove rmtest/testfile (yes/no)? y
rm: remove rmtest: (yes/no)? y
$ ls
Reports     dir10       feathers    file1       fruit2
brands      dir2        feathers_6  file2       greetings
dante       dir3        file.1      file3       myvars
dante_1     dir4        file.2      file4       newdir
dir1        dir5        file.3      fruit       tutor.vi
$
```

Using Symbolic Links

Files (and directories) might be located on several different file systems. You can use symbolic links to link files that are in different file systems.

There are two main reasons you might choose to use symbolic links:

- To move files to a new location – This includes moving a directory on a different disk (partition) but leaving a link so that other users do not need to know about the move.
- To provide a convenient name for a file rather than the original name, which might be complicated or unknown – When accessing a USB flash drive, a user can type `ls /rmdisk/rmdisk0` without having to find out what the USB flash drive is called.

A file system is a collection of certain types of files, organized for administrative convenience. The organization of these files enables you to store files that need to be shared on one machine. These shared files can be accessed by many machines by using a remote file access mechanism.

A symbolic link is a pointer that contains the pathname to another file or directory. The link makes the file or directory easier to access if it has a long pathname. A symbolic link file is identified by the letter `l` in the file-type field. To view symbolic link files, use the `ls -l` command.

Creating Symbolic Links

You can use the `ln -s` command to create a symbolic link file. You can use either relative or absolute pathnames to create a symbolic link file. The file name for the symbolic link appears in the directory in which it was created.

The syntax for the `ln -s` command is:

```
ln -s source_file target_file
```

The *`source_file`* variable refers to the file to which you create the link. The *`target_file`* variable refers to the name of the symbolic link. When creating a symbolic link, the *`source_file`* might not already exist. If the *`source_file`* does not exist, a symbolic link that points to a non-existing file is created.

To create a symbolic link file, named `dante_link`, to the `dante` file, enter the `ln -s` command.

```
$ cd
$ pwd
/export/home/student
$ mv dante /var/tmp
$ ln -s /var/tmp/dante dante_link
$
```

You can display a list of files and directories, by entering the `ls -F` command.

```
$ ls -F
Reports/     dir10/       feathers     file1*       fruit2
brands       dir2/        feathers_6   file2*       greetings
dante_1      dir3/        file.1*      file3*       myvars
dante_link@  dir4/        file.2*      file4*       newdir/
dir1/        dir5/        file.3*      fruit        tutor.vi
$
```

The output of the `ls -F` command shows the file `dante_link` with the `@` symbol following it to indicate that `dante_link` is a symbolic link.

```
$ cat dante_link
                  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.
... (output truncated)
```

To see the pathname to which a symbolic link is pointing, enter the `ls -l` command with the symbolic link file name.

```
$ ls -l dante_link
lrwxrwxrwx   1 student  class        14 Feb 6 14:17 dante_link ->
/var/tmp/dante
$
```

Removing Symbolic Links

You can use the `rm` command to remove a symbolic link file in the same manner as you would remove a standard file.

To remove the `dante_link` symbolic link file, enter the following commands:

```
$ cd
$ pwd
/export/home/student
$ ls -l dante_link
lrwxrwxrwx   1 student  class        14 Feb 6 14:17 dante_link ->
/var/tmp/dante
$ rm dante_link
$ cat dante
No such file or directory
$ mv /var/tmp/dante ~/dante
$ ls -l dante dante_link
dante_link: No such file or directory
-rw-r--r--   1 student  class     1319 Feb 6 14:18 dante
$
```

Exercise: Using Directory and File Commands

In this exercise, you use the commands described in this module to copy, move, rename, create, and remove files and directories.

Preparation

No special preparation is required for this exercise.

RLDC

In addition to being able to use local classroom equipment, this lab was designed to also use equipment located in a remote lab data center. Directions for accessing and using this resource can be found at:

```
http://remotelabs.sun.com/
```

Ask your instructor for the particular SSH configuration file that you should use to access the appropriate remote equipment for this exercise.

Tasks

To use directory and file commands, complete the following steps. Write the commands that you would use to perform each task in the space provided.

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

   ____

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

   ____

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

   ____

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

   ____

5. Complete the missing options and their descriptions in Table 4-3:

Table 4-3 Directory Options

Option	**Description**
`cp -i`	
	Includes the contents of a directory, including the contents of all subdirectories, when you copy a directory

6. From your home directory, create a directory called `practice1`.

7. Using a single command, copy the files `file.1` and `file.2` to the `practice1` directory.

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

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

10. Using a single command with options, copy the `play` directory in the `practice1` directory to a new directory in the `practice1` directory called `appointments`.

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

12. In your home directory, create a directory called `house` with a subdirectory of `furniture` using a single command.

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

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

15. Create a new file called `carrot`, and then rename it `celery`.

16. Using a single command, remove the directories called `memos` and `misc` from your home directory.

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

18. Identify the command to remove a directory that is not empty. Remove the `house/furniture` directory. List the contents of the `house` directory to verify that the `furniture` directory has been removed.

19. Create a new directory named `newname`, and then rename it `veggies`.

20. Create a file named `mycontents` that is a symbolic link to the `/var/sadm/install/contents` file.

21. Verify that the symbolic link works.

22. Type `q` to quit the `mycontents` file view.

23. Remove the symbolic link that you created in Step 20.

Exercise Summary

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

- Experiences
- Interpretations
- Conclusions
- Applications

Exercise Solutions: Using Directory and File Commands

To use directory and file commands, complete the following steps:

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

```
$ cd; ls
```

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

```
$ cp dir1/coffees/beans/beans dir4/roses
```

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

```
$ mkdir dir3/vegetables
```

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

```
$ mv dir1/coffees/beans/beans dir2/recipes
```

5. Complete the missing options and their descriptions in Table 4-4:

Table 4-4 Directory Options

Option	*Description*
`cp -i`	*Prevents you from accidentally overwriting existing files or directories*
`-r`	Includes the contents of a directory, including the contents of all subdirectories, when you copy a directory

6. From your home directory, create a directory called `practice1`.

```
$ mkdir practice1
```

7. Using a single command, copy the files `file.1` and `file.2` to the `practice1` directory.

```
$ cp file.1 file.2 practice1
```

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

```
$ cp dir3/planets/mars practice1/addresses
```

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

```
$ mkdir practice1/play
$ mv practice1/addresses practice1/play
```

10. Using a single command, copy the `play` directory in the `practice1` directory to a new directory in the `practice1` directory called `appointments`.

```
$ cp -r practice1/play practice1/appointments
```

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

```
$ ls -R practice1
```

12. In your home directory, create a directory called `house` with a subdirectory of `furniture` using a single command.

```
$ cd; mkdir -p house/furniture
```

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

```
$ touch house/furniture/chairs
```

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

```
$ mkdir records memos misc
```

15. Create a new file called `carrot`, and then rename it `celery`.

```
$ touch carrot
$ mv carrot celery
```

16. Using a single command, remove the directories called `memos` and `misc` from your home directory.

```
$ rmdir memos misc
```

or

```
$ rm -r memos misc
```

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

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

18. Identify the command to remove a directory that is not empty. Remove the `house/furniture` directory. List the contents of the `house` directory to verify that the `furniture` directory has been removed.

```
$ rm -r house/furniture
$ ls house
$
```

19. Create a new directory named `newname`, and then rename it `veggies`.

```
$ mkdir newname
$ mv newname veggies
```

20. Create a file named `mycontents` that is a symbolic link to the file `/var/sadm/install/contents`.

```
$ ln -s /var/sadm/install/contents mycontents
```

21. Verify that the symbolic link works.

```
$ more mycontents
```

22. Type `q` to quit the `mycontents` file view.

23. Remove the symbolic link that you created in Step 20.

```
$ rm mycontents
$ ls mycontents
mycontents: No such file or directory
```

Notes:

Using the `vi` Editor

Objectives

This module introduces the `vi` editor and describes the `vi` commands. These commands include input commands, positioning commands, and editing commands.

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

- Describe the fundamentals of the `vi` editor
- Modify files using the `vi` editor

Figure 5-1 is the course map that shows how this module fits into the current instructional goal.

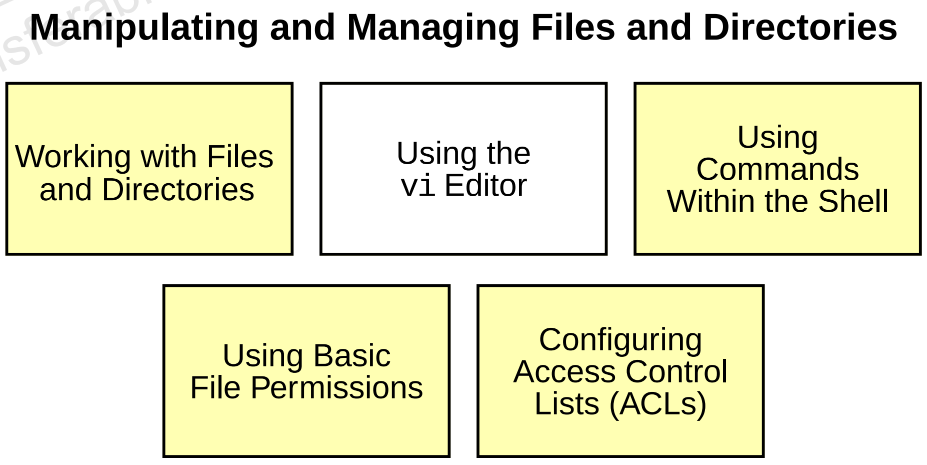

Manipulating and Managing Files and Directories
Working with Files and Directories
Using the vi Editor
Using Commands Within the Shell
Using Basic File Permissions
Configuring Access Control Lists (ACLs)

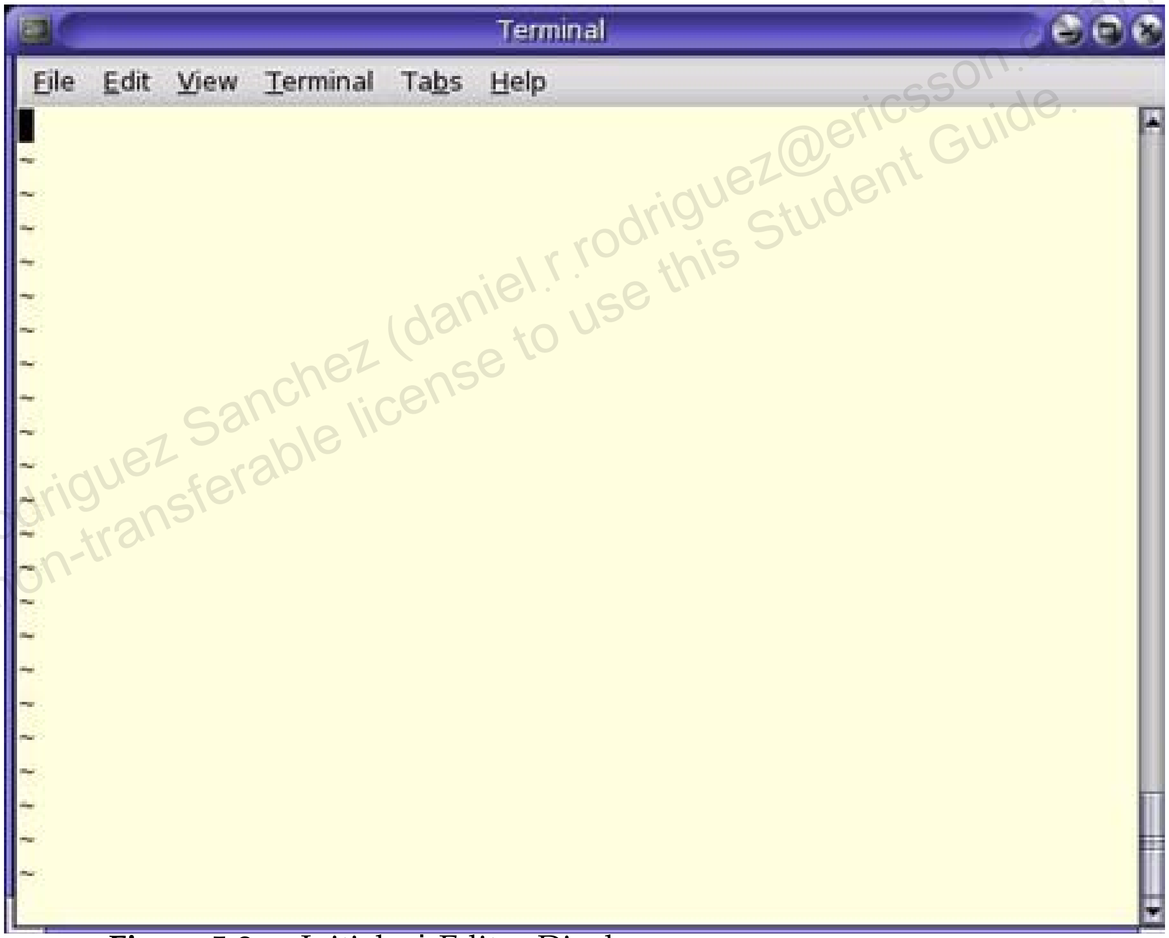
Figure 5-1 Course Map

Fundamentals of the vi Editor

The visual display or vi editor is an interactive editor that you can use to create and modify text files. You can use the vi editor when the desktop environment window system is not available. The vi editor is also the only text editor that you can use to edit certain system files without changing the permissions of the files.

All text editing with the vi editor takes place in a buffer. You can either write the changes to the disk, or discard them.

Figure 5-2 shows the initial vi display in a terminal window.



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

Figure 5-2 Initial vi Editor Display

The vi Editor Modes of Operation

The vi editor is a command-line editor that has three basic modes of operation:

- Command mode
- Input mode
- Last line mode

Command Mode

The command mode is the default mode for the vi editor. In this mode, you can run commands to delete, change, copy, and move text. You can also position the cursor, search for text strings, and exit the vi editor.

Input Mode

You can insert text into a file in the input mode. The vi editor interprets everything you type in the input mode as text. To invoke input mode, press one of the following lower-case keys:

- `i` – Inserts text before the cursor
- `o` – Opens a new blank line below the cursor
- `a` – Appends text after the cursor

You can also invoke the input mode to insert text into a file by pressing one of the following upper-case keys:

- `I` – Inserts text at the beginning of the line
- `O` – Opens a new blank line above the cursor
- `A` – Appends text at the end of the line

Last Line Mode

You can use advanced editing commands in the last line mode. To access the last line mode, enter a colon (`:`) while in the command mode. Entering the colon (:) character places the cursor at the bottom line of the screen.

Switching Between the Command and Input Modes

The default mode for the `vi` editor is the command mode. When you enter an `i`, `o`, or `a` command, the `vi` editor switches to the input mode. After editing a file, press Esc key to return the `vi` editor to the command mode. When in the command mode, you can save the file and quit the `vi` editor.

The following example shows how to switch modes in the `vi` editor:

1. Enter the `vi` *`filename`* command to create a file. You are automatically in the command mode.
2. Type the `i` command to insert text. The `i` command switches the `vi` editor to the input mode.
3. Press Esc key to return to the command mode.
4. Enter the `:wq` command to save the file, exit the `vi` editor and return to the shell prompt.

Using the vi Command

The `vi` command enables you to create, edit, and view files in the `vi` editor.

The syntax for the `vi` command includes the following three choices:

```
vi
```

or

```
vi filename
```

or

```
vi options filename
```

If the system crashes while you are editing a file, you can use the `-r` option to recover the file.

To recover a file, enter the command:

```
$ vi -r filename
```

The file opens so that you can edit it. You can then save the file and exit the `vi` editor.

```
$ vi -R filename
```

The file opens in read-only mode to prevent accidental overwriting of the contents of the file.

Modifying Files With the vi Editor

You can use the vi editor to view files in the read-only mode, or you can edit files in the vi editor using the vi editing commands. When using the vi editor, you can move the cursor using certain key sequences.

Viewing Files in the Read-Only Mode

The view command enables you to view files in the read-only mode. It invokes the vi editor with the read-only option. Although most of the vi commands are available, you cannot save changes to the file.

The syntax for the view command is:

```
view filename
```

To view the dante file in the read-only mode, enter the command:

```
$ cd
$ view dante
```

The dante file appears. Enter the :q command to exit the file, the vi editor, and return to the shell prompt.

Inserting and Appending Text

Table 5-1 describes the commands that you can use to insert and append text to a new or existing file using the `vi` editor. Each of these commands switch the vi editor to the input mode. To return to the command mode, press the Esc key.

Table 5-1 Input Commands for the `vi` Editor

Command	**Function**
`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
`:r` *`filename`*	Inserts text from another file into the current file

Note – The `vi` editor is case sensitive. Use the appropriate case for the input commands. Also, most of the input commands and cursor movements can be preceded by a number to repeat the command that number of times.

Moving the Cursor Within the vi Editor

Table 5-2 shows the key sequences that move the cursor in the vi editor.

Table 5-2 Key Sequences for the vi Editor

Key Sequence	**Cursor Movement**
h, left arrow, or Backspace	Left one character
j or down arrow	Down one line
k or up arrow	Up one line
l, right arrow, or space bar	Right (forward) one character
w	Forward one word
b	Back one word
e	To the end of the current word
$	To the end of the line
0 (zero)	To the beginning of the line
^	To the first non-white space character on the line
Return	Down to the beginning of the next line
G	Goes to the last line of the file
1G	Goes to the first line of the file
:*n*	Goes to Line *n*
*n*G	Goes to Line *n*
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
Control-G	Displays current buffer information

Editing Files Using the vi Editing Commands

You can use numerous commands to edit files using the `vi` editor. The following sections describe basic operations for deleting, changing, replacing, copying, and pasting. Remember that the `vi` editor is case sensitive.

Using the Text-Deletion Commands

Table 5-3 shows the commands that delete text in the `vi` editor.

Table 5-3 Text-Deletion Commands for the `vi` Editor

Command	**Function**
`R`	Overwrites or replaces characters on the line at and to the right of the cursor. To terminate this operation, press Esc key.
`C`	Changes or overwrites characters from cursor to the end of the line.
`s`	Substitutes a *`string`* for a character at the cursor.
`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 from the cursor to the right end of the line.
`:`*`n`*`,`*`n`*`d`	Deletes Lines *`n`* –*`n`* (for example, `:5,10d` deletes Lines `5`–`10`).

Note – Output from the `delete` command writes to a buffer from which text can be retrieved.

Using the Text-Changing Commands

Table 5-4 describes the commands that you can use to change text, undo a change, and repeat an edit function in the vi editor. Many of these commands change the vi editor to input mode. To return to command mode, press the Esc key.

Table 5-4 Edit Commands for the vi Editor

Command	Function
`cw`	Changes or overwrites characters at the cursor location to the end of that word
`r`	Replaces the character at the cursor with one other character
`J`	Joins the current line and the line below
`xp`	Transposes the character at the cursor and the character to the right of the cursor
`~`	Changes the case of the letter, either uppercase or lowercase, at the cursor
`u`	Undo the previous command
`U`	Undo all changes to the current line
`.`	Repeats the previous command

Using the Text-Replacing Commands

Table 5-5 shows the commands that search for and replace text in the vi editor.

Table 5-5 Search and Replace Commands

Command	**Function**
/*string*	Searches forward for the *string*.
?*string*	Searches backward for the *string*.
n	Searches for the next occurrence of the *string*. Use this command after searching for a *string*.
N	Searches for the previous occurrence of the *string*. Use this command after searching for a *string*.
:%s/*old*/*new*/g	Searches for the *old* string and replaces it with the *new* string globally.

Using the Text-Copying and Text-Pasting Commands

The yy command *yanks* lines of text and holds a copy of the lines in a temporary buffer. The put commands (p, P) inserts those lines of text, held in the temporary buffer, into the current document at the specified location.

The copy (co) and move (m) commands copy or move specified lines to the selected location within the file.

Table 5-6 shows the commands that yank (yy) and put (p, P) text in the vi editor.

Table 5-6 Copy and Paste Commands

Command	**Function**
yy	Yanks a copy of a line
p	Puts yanked or deleted text under the line containing the cursor
P	Puts yanked or deleted text before the line containing the cursor

Table 5-7 shows the commands that copy (co) and move (m) text in the vi editor.

Table 5-7 Additional Copy and Paste Commands

Command	**Function**
`:n,n co n`	Copies lines *n* –*n* and puts them after line *n*. For example, :`1,3 co 5` copies lines 1–3 and puts them after line 5.
`:n,n m n`	Moves lines *n* –*n* to line *n*. For example, `:4,6 m 8` moves lines 4–6 to line 8, line 6 becomes line 8, line 5 becomes line 7, and line 4 becomes line 6.

Using the File Save and Quit Commands

Table 5-8 describes the commands that save the text file, quits the vi editor and returns to the shell prompt.

Table 5-8 Save and Quit Commands

Command	**Function**
`:w`	Saves the file with changes by writing to the disk
`:w new_filename`	Writes the contents of the buffer to *new_filename*
`:wq`	Saves the file with changes and quits the vi editor
`:x`	Saves the file with changes and quits the vi editor
`ZZ`	Saves the file with changes and quits the vi editor
`:q!`	Quits without saving changes

Customizing a vi Session

You can customize a vi session by setting variables for the session. When you set a vi variable, you enable a feature that is not activated by default. You can use the set command to enable and disable variables.

Table 5-9 describes some of the variables of the set command, including displaying line numbers and invisible characters, such as the Tab character and end-of-line characters.

Table 5-9 Edit Session Customization Commands

Command	Function
`: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 ^I for a Tab and $ for end-of-line characters
`:set nolist`	Turns off the display of invisible characters
`:set showmode`	Displays the current mode of operation
`:set noshowmode`	Turns off the mode of operation display
`:set`	Displays all the vi variables that are set
`:set all`	Displays all vi variables and their current values

To create an automatic customization for all of your vi sessions, complete the following steps:

1. Create a file in your home directory named .exrc
2. Enter any of the set variables into the .exrc file.
3. Enter each set *variable* without the preceding colon, (as shown in Table 5-9).
4. Enter each command on one line.

The vi editor reads the .exrc file, located in your home directory each time you open a vi session, regardless of your current working directory.

Exercise: Using the vi Editor

In this exercise, you practice performing vi editor commands in the tutor.vi tutorial. Use Figure 5-3 on page 5-15 as a reference to complete the exercise.

Preparation

No special preparation is required for this exercise.

RLDC

In addition to being able to use local classroom equipment, this lab was designed to also use equipment located in a remote lab data center. Directions for accessing and using this resource can be found at:

http://remotelabs.sun.com/

Ask your instructor for the particular SSH configuration file that you should use to access the appropriate remote equipment for this exercise.

Tasks

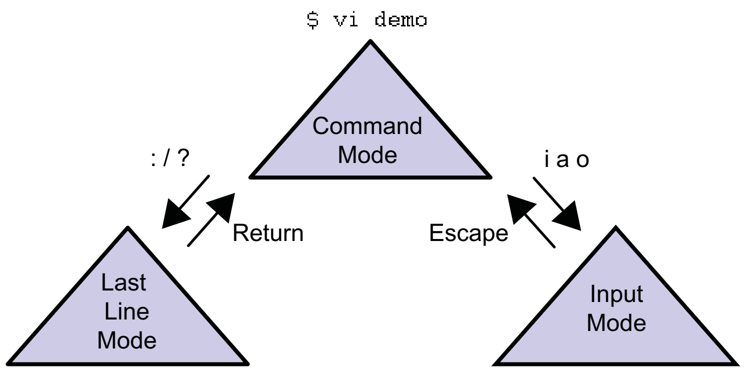
To learn how to use the vi editor, complete the following steps.

1. Make sure that you are in your home directory. To open the tutor.vi tutorial file, enter the following command:

```
$ vi tutor.vi
```

2. Complete the lessons outlined in this tutorial.

Figure 5-3 shows a quick reference chart for the vi editor.



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 line number
`: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 5-3 Quick Reference Chart for the vi Editor

Exercise Summary

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

- Experiences
- Interpretations
- Conclusions
- Applications

Module 6

Using Commands Within the Shell

Objectives

This module introduces commands within the Korn shell. The module describes command-line processing, shell metacharacters, command redirection, command history, and initialization files.

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

- Use shell metacharacters
- Describe the Korn shell variables
- Display the command history
- Redirect commands
- Describe the command-line interpreter
- Work with user initialization files

Figure 6-1 is the course map that shows how this module fits into the current instructional goal.

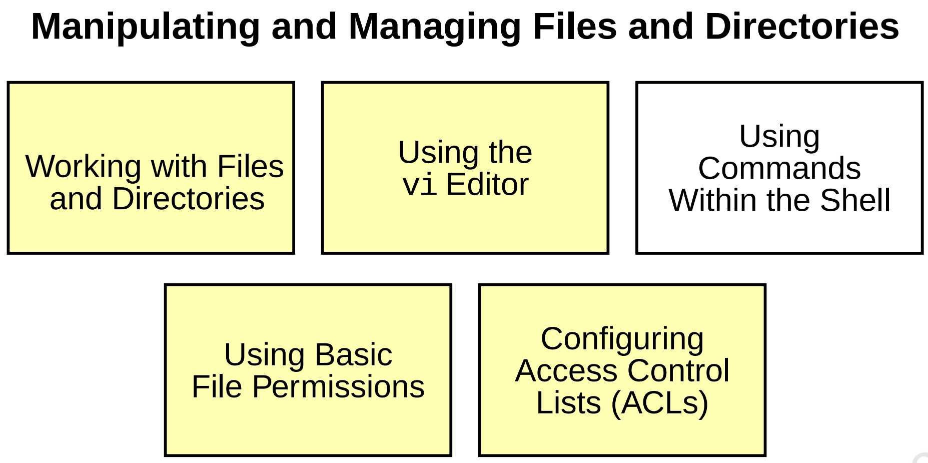

Manipulating and Managing Files and Directories
Working with Files and Directories
Using the vi Editor
Using Commands Within the Shell
Using Basic File Permissions
Configuring Access Control Lists (ACLs)

Figure 6-1 Course Map

Using the Shell Metacharacters

Korn shell metacharacters are specific characters, generally symbols, that have special meaning to the shell. Three types of metacharacters are pathname metacharacters, file name substitution metacharacters, and redirection metacharacters.

Caution – Do not use these metacharacters when creating file and directory names. These characters hold special meaning to the shell.

Using the Pathname Metacharacters

Some of the shell metacharacters have specific pathname functions. These metacharacters simplify location changes within the directory hierarchy. Some examples of pathname metacharacters are:

```
~ ~username
```

Tilde (~) Character

The tilde (~) character represents the home directory of the current user. It is a substitution that equates to the absolute pathname of the user's home directory.

To change directories to `dir1` using the tilde (`~`) character, enter the following command:

```
$ cd ~/dir1
$ pwd
/export/home/student/dir1/
$
```

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

Tilde (~) Character With a User Name

The tilde (~) character followed by a user name represents the home directory of the specified user.

To change directories to the `user2` home directory, enter the following commands:

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

Dash (-) Character

The dash (`-`) character in the shell represents the previous working directory. You can use the dash character to switch between two specific directories. The shell automatically displays the current directory path.

To switch between the `student` and `tmp` directories, enter the following commands:

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

Using the File Name Substitution Metacharacters

You can substitute some shell metacharacters for other characters. These metacharacters simplify commands. Some examples of file name substitution metacharacters are:

```
* ? []
```

Asterisk (*) Character

The asterisk (*) character is also called the *wild card character* and represents zero or more characters, except the leading period (.) of a hidden file.

To list all files and directories that start with the letter f followed by zero or more other characters, enter the following commands:

```
$ cd
$ ls f*
feathers    file.1      file.2      file.3      file4       fruit2
feathers_6  file1       file2       file3       fruit
$
```

To list all files and directories that start with the letter d followed by zero or more other characters, enter the following command:

```
$ ls d*
dante    dante_1

dir1:
coffees  fruit    trees

dir2:
beans  notes  recipes

dir3:
cosmos   moon   planets   space   sun    vegetables

dir4:
constellation  memo  roses

dir5:

dir10:
planets
$
```

To list all files and directories that end with the number 3, preceded by zero or more characters, enter the following command:

```
$ ls *3
file.3  file3

dir3:
cosmos   moon   planets   space   sun    vegetables
$
```

Question Mark (?) Character

The question mark (?) character represents any single character except the leading period (.) of a hidden file. The question mark (?) character is also called a *wild card* character.

To list all files and directories that start with the string dir and followed by one other character, enter the following command:

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

dir2:
beans  notes  recipes

dir3:
cosmos   moon   planets   space   sun    vegetables

dir4:
constellation  memo  roses

dir5:
$
```

If no files match an entry using the question mark (?) character, an error message appears.

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

Square Bracket ([]) Characters

The square bracket ([]) characters represent a set or range of characters for a single character position.

A set of characters is any number of specific characters; for example, [acb]. The characters in a set do not generally need to be in any order. For example, [abc] is the same as [cab].

A range of characters is a series of ordered characters. A range lists the first character, a hyphen (-), and the last character, for example, [a–z] or [0–9]. When you specify a range, arrange the characters in the order that you want them to appear in the output. Use [A–Z] or [a–z] to search for any uppercase or lowercase alphabetical character, respectively.

To list all files and directories that start with the letters a through f, enter the following command:

```
$ ls [a-f]*
brands      dante_1     file.1      file2       file4
celery      feathers    file1       file.3      fruit
dante       feathers_6  file.2      file3       fruit2

dir1:
coffees  fruit    trees

dir10:
planets

dir2:
beans    notes    recipes

dir3:
cosmos      moon        planets     space       sun         vegetables

dir4:
constellation  memo          roses

dir5:
$
```

To list all files and directories that start with the letters `f` or `p`, enter the following command:

```
$ ls [fp]*
feathers    file.1      file.2      file.3      file4       fruit2
feathers_6  file1       file2       file3       fruit

perm:
group   motd    skel    vfstab

practice1:
appointments  file.1        file.2        play
$
```

Introducing Korn Shell Variables

A shell variable refers to a temporary storage area in memory. Shell variables contain:

- Information needed for customizing the shell
- Information needed by other processes to function properly

The shell enables you to store values in variables.

Korn shell programming uses two types of variables:

- Variables that are exported to subprocesses
- Variables that are not exported to subprocesses

Table 6-1 describes the Korn shell commands used to set, unset, and view variables.

Table 6-1 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`

Note – When a shell variable follows the dollar `$` sign character, the shell interprets 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

To list all shell variables with their current values, enter the `set` command. 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/student
HZ=100
IFS=
LANG=C
LINENO=1
LOGNAME=student
MAIL=/var/mail/student
MAILCHECK=600
MANPATH=/usr/man:/usr/openwin/share/man
OLDPWD=/export/home/student
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=student
_=set
office=/export/home/student/office
private=/export/home/student/private
$
```

To make the value of a variable known to a sub-shell, export it using the `export` command. To view a list of all these shell variables and their current values, enter the `export` command:

```
$ export
DISPLAY=:0.0
EDITOR=/usr/bin/vi
HELPPATH=/usr/openwin/lib/locale:/usr/openwin/lib/help
HOME=/export/home/student
LANG=C
LINENO=1
LOGNAME=student
MAIL=/var/mail/student
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=student
_=set
office=/export/home/student/office
$
```

Setting and Unsetting Shell Variables

A variable is set and a value is assigned with the following syntax:

```
var=value
```

or

```
VAR=value
```

There is no space on either side of the equal (=) sign. For example:

```
$ private=/export/home/student/private
$ set| grep private
private=/export/home/student/private
$ cd $private; pwd
/export/home/student/private
$
```

To make the value of a variable known to a sub-shell, use the following command syntax:

```
export VAR
```

or

```
export var
```

For example:

```
$ export office=/export/home/student/office
$ echo $office
/export/home/student/office
$
```

You can delete the values stored in shell variables with the following command syntax:

```
unset VAR
```

or

```
unset var
```

For example, to `unset` the variable `private`, enter the following commands:

```
$ unset private
$ echo $private

$
```

The output of the `echo` command is a blank line.

Note – Usually, uppercase variables are used for environment settings, and lowercase variables are used for settings local to the shell.

Displaying Default Variables

Table 6-2 describes variables that are assigned default values by the shell on login.

Table 6-2 Default Korn Shell Variables

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 to which the `cd` command changes when no argument is supplied in the command-line.
`LOGNAME`	Sets the login name of the user.
`PATH`	Specifies a 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 (>).
`SHELL`	Specifies the name of the shell (that is, `/bin/ksh`).

Customizing Korn Shell Variables

This section describes 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' \$PWD $ "
student@host1: $
```

In this example, the prompt displays the login name of the user the system's *`hostname`* and the current working directory.

The *`username`* is read from the variable `LOGNAME`, and the *`hostname`* comes from the output of the `uname -n` command.

This shell prompt displays the correct information even when the user logs in on different hosts.

The back quotation (‘) marks delimit an imbedded command string

The following example shows how the value of another shell variable is used for prompt definition.

Note – Please be aware that your shell prompt changes from “$” to “I like Solaris $”

```
$ ILU=”I like Solaris”
$ PS1=”$ILU $ ”
I Like Solaris $ echo $ILU
I like Solaris
I like Solaris $
```

Note – To make the new shell prompt appear in every shell, it must be included in the user’s Korn shell initialization file (usually the user’s `.kshrc` file).

The PATH Variable

The PATH variable contains a list of directory pathnames, separated by colons. When you run a command on the command-line, the shell searches these directories from left to right to locate that command.

The shell executes the first command that it finds.

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 the directories specified in the PATH variable. Sometimes when the shell is unable to find a particular command, the command may reside in a directory that has not been specified in the PATH variable.

When the shell is unable to find a particular command, the user can type the absolute pathname of the directory in which the command resides, on the command line. For example:

```
$ /usr/bin/id
```

If the shell is able to execute the command using the absolute pathname, check the PATH variable to ensure that the directory path already exists in the search path. If the directory exists in the search path, check that it has been entered correctly, by entering the following command:

```
$ echo $PATH
/usr/dt/bin:/usr/openwin/bin:/usr/bin:/usr/ucb
$
```

Extending the PATH Variable

To extend the PATH variable to include the home directory of the user, enter the following commands.

```
$ 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/student
```

The PATH variable passes the value automatically to the sub-shells.

Using the Quoting Characters

Quoting is a process that instructs the shell to mask, or ignore, the special meaning of metacharacters. The quoting characters are single forward quotation marks (`' '`), double quotation marks (`" "`), backslash (\), and parentheses (`$ (command)`).

Quotation marks enclosing a string of metacharacters prevent the shell from interpreting the special meaning of these metacharacters.

There are two types of quotation marks that mask the special meaning of metacharacters, these include the single forward quotation marks (`' '`) and double quotation marks (`" "`).

Single forward quotation marks instruct the shell to ignore all enclosed metacharacters. Double quotation marks instruct the shell to ignore all enclosed metacharacters, except for the following three characters listed below:

- The single backward quotation marks (`‘`) instruct the shell to execute and display the output of a Solaris system command.
- The backslash (\) character in front of a metacharacter will prevent the shell from interpreting the next character as a metacharacter.
- The dollar ($) sign instructs the shell to execute and display the output for the command enclosed within parentheses. Parentheses (`$ (command)`) instruct the shell to execute and display the output for the command enclosed within.

  You can also use parentheses () to perform command substitution. For example, the output of the nested command, *`command2`*, is substituted when the *`command1`* command is executed. The syntax for this is:

```
command1 $(command2)
```

To ignore the special meaning of the dollar ($) sign metacharacter, enter the following command:

```
$ echo '$SHELL'
$SHELL
$
```

Note – The `echo` utility writes arguments to the standard output.

To interpret (or view) the special meaning of the dollar ($) sign metacharacter, enter the following command:

```
$ echo "$SHELL"
/bin/ksh
$
```

To ignore the special meaning of the dollar ($) sign metacharacter when you use double quotes, enter the following command:

```
$ echo "\$SHELL"
$SHELL
$
```

To display the output for the `date` command using quoting characters, enter the following command:

```
$ echo “Today’s date is ‘date‘”
Today’s date is Wed Jan 28 21:26:33 MST 2009
$
```

To execute the `pwd` command using the dollar ($) sign and parentheses, enter the following command:

```
$ echo “The user is currently in the $(pwd) directory.”
The user is currently in the /export/home/student directory.
$
```

Displaying the Command History

The shell keeps a history of recently entered commands. This history mechanism enables you to view, repeat, or modify previously executed commands.

Note – Command history is shared among all Korn shells for a given user.

Using the `history` Command

The `history` command displays previously executed commands. By default, the `history` command displays the last 16 commands to the standard output.

The syntax for the `history` command is:

```
history option
```

To display previously executed commands, enter the following command:

```
$ history
...
109    date
110    cd /etc
111    touch dat1 dat2
112    ps -ef
113    history
```

The `history` command is an alias built into the Korn shell that enables you to display previously executed commands.

Note – Your output varies based on the commands recorded in your `.sh_history` file. The output can include commands from multiple windows.

The numbers displayed to the left of the command are command numbers. You can use a command number with the history `command` to instruct the shell to re-execute a particular command or command-line.

To display the list of command history without line numbers, enter the following command:

```
$ history -n
    date
    cd /etc
    touch dat1 dat2
    ps -ef
    history
    history -n
```

To display the current command and the four commands preceding it, enter the following command:

```
$ history -4
111    touch dat1 dat2
112    ps -ef
113    history
114    history -n
115    history -4
$
```

To display the history list in reverse order, enter the following command:

```
$ history -r
116    history -r
115    history -4
114    history -n
113    history
112    ps -ef
111    touch dat1 dat2
110    cd /etc
109    date
$
```

To display the most recent `cd` command to the most recent `date` command, enter the following command:

```
$ history cd date
110    cd /etc/
109    date
$
```

Note – The Korn shell stores command history in a file specified by the `HISTFILE` variable. The default file is the `~/.sh_history` file. You can use the `HISTSIZE` variable to specify the number of commands to store in this buffer. If this variable is not set, then the shell stores the most recent 128 commands.

Using the `r` Command

The `r` command is an alias built into the Korn shell that enables you to repeat a command.

To repeat the `cal` command using the `r` command, enter the following command:

```
$ cal
    February 2009
 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

$ r
cal
    February 2009
 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

$
```

To re-execute a command by line number, use the `r` command followed by the respective line number. For example:

```
$ history
...
109   date
110   cd /etc
111   touch dat1 dat2
112   ps -ef
113   history
$ r 112
(output of the ps -ef command omitted)
```

You can also use the `r` command to re-execute a command beginning with a particular character, or string of characters. To re-run the most recent occurrence of a command that begins with the letter "`c`" enter the following command:

```
$ r c
cd /etc
$
```

To re-run the most recent occurrence of the `ps` command, enter the following command:

```
$ r ps
ps -ef
(output of the ps -ef command omitted)
```

You can use the `r` command to repeat a previous command, perform a simple edit of a command-line and run the newly modified command.

For example, to repeat the most recent occurrence of a command beginning with the letter "`c`" and replace `dir1` with `dir2`, enter the following sequence of command:

```
$ history
...
121   cd
122   cat dante
123   ls
124   cd ~/dir1
125   cal
$ r c
cd ~/dir1
$ r dir1=dir2
cd ~/dir2
```

Editing and Re-running Previously Executed Commands

You can edit previously executed commands and re-run these commands using a shell in-line editor.

Set uses the `vi` editor to both turn on and enable the shell history editing feature with one of the following commands:

```
$ set -o vi
```

or

```
$ export EDITOR=/bin/vi
```

or

```
$ export VISUAL=/bin/vi
```

Note – All three commands listed above will turn on the `vi` command-line editing in the shell. You can use the `set -o` command to verify this feature is turned on.

You can access a command in the history buffer, edit the command with the `vi` editor, and execute the modified command by following these steps:

1. Verify that the built-in `vi` editor is enabled.

```
$ set -o | grep -w vi
vi              on
```

2. Type the `history` command to view the command history list.

```
$ history
```

3. Press the Esc key to access the command history list.

   Use the following keyboard keys to move the cursor through the command history list.

   - `k` – Moves the cursor up one line at a time
   - `j` – Moves the cursor down one line at a time
   - `l` – Moves the cursor to the right
   - `h` – Moves the cursor to the left

Note – You cannot use the arrow keys to move the cursor within the command history list.

4. Use the `vi` commands to edit any previously executed command.
5. To run a modified command, press the Return key.

File Name Completion

To invoke file name completion type the `ls` command followed by one or more characters of a file name. Then press the following keys in sequential order: Escape (`Esc`) and backslash (\)

Note – The key sequence presented above applies only to the `vi` mode of command-line editing.

If the shell finds a file name beginning with the specified characters, it automatically 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 \ (backslash) keys
```

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 Escape (Esc) and equal (=) sign keys sequentially.

To request that the shell present all file names beginning with the letter "g" in the /etc directory, enter the following commands:

```
$ cd /etc
$ cat g            Press Esc, press the = key
1) gconf/
2) getty
3) gimp/
4) gnome-vfs-2.0/
5) gnome-vfs-mime-magic
6) gnopernicus-1.0/
7) group
8) grpck
9) gss/
10) gtk-2.0/
11) gtk/
$ cat g
The cursor is positioned on top of the letter "g" at this point.
```

Using Command Redirection

You can redirect input to and output from commands using redirection. There are two redirection commands: the greater than (>) sign and the less than (<) sign metacharacters. Both of these commands are implied with the pipe (|) character.

Typically, the shell receives or reads command input from the keyboard, and displays or writes the command output to the screen. Figure 6-2 shows standard input and output.

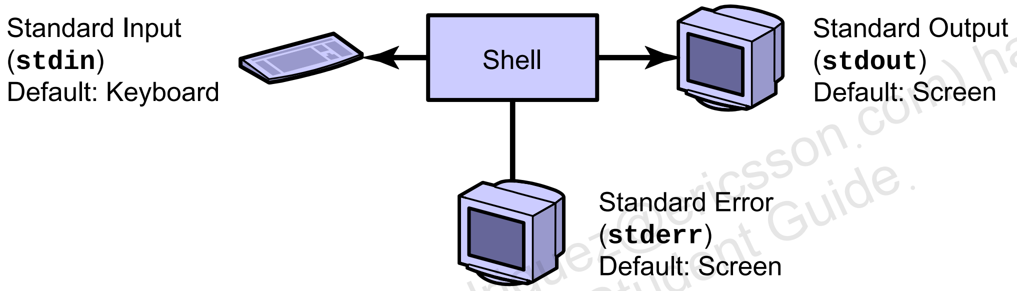


Figure 6-2 Standard Command I/O in the Shell

You can instruct the shell to redirect command input from and command output to files, on the command-line, or within shell scripts. Input redirection forces a command to read the input from a file instead of from the keyboard. Output redirection sends the output from a command into a file instead of sending the output to the screen.

As the command generates error messages, these messages are sent to the standard error. Usually error messages are sent to the terminal screen.

The File Descriptors

Each process that the shell creates works with file descriptors. File descriptors determine where the input to the command originates and where the output and error messages are sent. Table 6-3 shows the file descriptors.

Table 6-3 File Descriptors

File Descriptor Number	**File Description Abbreviation**	**Definition**
`0`	`stdin`	Standard command input
`1`	`stdout`	Standard command output
`2`	`stderr`	Standard command error

All commands that process file content read from the standard input and write to the standard output.

The following example shows the standard input (bold text) and the standard output (plain text) for the `cat` command. Press Control-D to exit the `cat` command.

```
$ 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)
Control-D  (Read from stdin)
$
```

The `cat` command takes the standard input from the keyboard and sends the standard output to the terminal window.

You can modify the default action of the standard input, standard output, and standard error within the shell by redirecting `stdin`, `stdout`, and `stderr`.

Redirecting Standard Input

The following syntax example shows a command using the less than (<) metacharacter to process a file as the standard input instead of reading the input from the keyboard:

```
command < filename
```

or

```
command 0< filename
```

To use the `dante` file as the input for the `mailx` command, enter the command:

```
$ mailx student < ~/dante
$
```

Redirecting Standard Output

The following syntax example shows a command using the greater than (>) metacharacter to direct the standard output to a file instead of printing the output to the screen. If the file does not exist, the system creates it. If the file exists, the redirection overwrites the contents of the file.

```
command > filename
```

or

```
command 1> filename
```

To list the current contents of your home directory and redirect that list of files and subdirectories into a file called `directory_list`, enter the commands:

```
$ cd
$ pwd
/export/home/student
$ ls -l > directory_list
$
```

The following example shows a command using two greater than (>>) characters to direct the standard output to the end of existing content in a file. The standard output gets appended to the existing content. If the file does not exist, the system creates it.

```
command >> filename
```

Note – When you use a single greater than (>) metacharacter, the command overwrites the original contents of the file, if the file already exists. When you use two greater than (>>) characters, the command appends the output to the original contents of the file.

To append the string `That’s my directory_list file` to the end of the `my_file` file, enter the command:

```
$ ls -l > my_file; cat my_file
-rw-r--r--   1 student class      1319 Jun 28  2009 dante
-rw-r--r--   1 student class       368 Jun 28  2009 dante_1
drwxr-xr-x   5 student class       512 Jun 28  2009 dir1
drwxr-xr-x   3 student class       512 Jun 28  2009 dir2
drwxr-xr-x   3 student class       512 Jun 28  2009 dir3
drwxr-xr-x   2 student class       512 Jun 28  2009 dir4
drwxr-xr-x   2 student class       512 Jun 28  2009 dir5
... (output truncated)
$ echo "That’s my directory_list file" >> my_file; cat my_file
-rw-r--r--   1 student class      1319 Jun 28  2009 dante
-rw-r--r--   1 student class       368 Jun 28  2009 dante_1
drwxr-xr-x   5 student class       512 Jun 28  2009 dir1
drwxr-xr-x   3 student class       512 Jun 28  2009 dir2
drwxr-xr-x   3 student class       512 Jun 28  2009 dir3
drwxr-xr-x   2 student class       512 Jun 28  2009 dir4
drwxr-xr-x   2 student class       512 Jun 28  2009 dir5
... (output truncated)
That’s my directory list file
$
```

Note – The semicolon (`;`) is a shell metacharacter that enables use of multiple commands on a single command-line.

Redirecting Standard Error

The following example shows a command using the file descriptor number (2) and the greater than (>) sign to redirect any standard error messages to the `/dev/null` file. This redirection is useful to suppress error messages so that no error messages appear on the screen.

```
command 2> /dev/null
```

The following example shows you how to re-direct error messages to the `/dev/null` file, which will then list the errors you would otherwise see displayed on the screen, if not for the redirection of `stderr`.

```
$ find /etc -type f -exec grep PASSREQ {} \; -print
# PASSREQ determines if login requires a password.
PASSREQ=YES
/etc/default/login
grep: can’t open /etc/inet/mipagent.conf-sample
grep: can’t open /etc/inet/mipagent.conf.fa-sample
grep: can’t open /etc/inet/mipagent.conf.ha-sample
grep: can’t open /etc/security/dev/audio
grep: can’t open /etc/security/dev/fd0
... (output truncated)

$ find /etc -type f -exec grep PASSREQ {} \; -print 2> /dev/null
# PASSREQ determines if login requires a password.
PASSREQ=YES
/etc/default/login
$
```

The following example shows a command redirecting the standard output to a file and the standard error to the same file. The syntax `2>&1` instructs the shell to redirect `stderr` (2) to the same file that receives `stdout` (1).

```
command 1> filename 2>&1
```

To print the standard output and the standard error to the dat file, enter the command:

```
$ ls /var /test 1> dat 2>&1
$ more dat
/test: No such file or directory        (stderr)
/var:                                   (stdout)
adm                                     (stdout)
audit                                   (stdout)
cron                                    (stdout)
... (output truncated)
```

The Pipe Character

The following example shows a command using the pipe (|) character to redirect the standard output to the standard input of another command:

command | *command*

You can insert a pipe character between any two commands the first command writes the output to standard output and the second command reads standard output from the previous command as standard input.

To use the standard output from the who command as the standard input for the wc -l command, enter the command:

```
$ who | wc -l
       35
$
```

The output of the who command never appears on the terminal screen because it is piped directly into the wc -l command.

To view a list of all the subdirectories located in the /etc directory, enter the command:

```
$ ls -F /etc | grep "/"
X11/
acct/
apache/
apache2/
apoc/
<outout truncated>
```

You can use pipes to connect numerous commands on the command-line.

To use the output of the head command as the input for the `tail` command and print the results, enter the command:

```
$ head -10 dante | tail -3 | lp
request id is printerA-177       (Standard input)
```

Command-Line Interpreter

When you type a command on the command-line, the shell interprets the command by parsing the line, handling metacharacters and redirection, and controlling the execution of commands. It then searches for the command and executes it.

The shell analyzes each typed line and initiates execution of the requested program.

For example, the shell runs the following command, and performs the following steps:

```
ps -ef | sort -k 1b,1 | more
```

1. The shell breaks up the command-line into its components, called *tokens*, as follows: `ps`, `-ef`, `|`, `sort`, `-k 1b,1`, `|`, and `more`.
2. The shell determines that `ps`, `sort`, and `more` are commands, `-ef` and `-k 1b,1` are arguments, and the pipes (`|`) are I/O operations.
3. Based on the shell parse order, the shell sets up `stdout` from `ps` to be `stdin` to `sort` and `stdout` from `sort` to be `stdin` to `more`.
4. The shell locates the `ps`, `sort`, and `more` commands and then executes them in order, applying the arguments as specified.

Working With User Initialization Files

This section describes the Solaris 10 OS default initialization files of the Bourne shell, the Korn shell, and the C shell.

Based on the shell, there might be either one or two default initialization files in your home directory that enable you to customize your working environment.

Table 6-4 defines the initialization files for the three primary shells in the Solaris 10 OS.

Table 6-4 Initialization Files at Login

Shell	**System-wide Initialization Files**	**Primary User Initialization Files Read at Login**	**User Initialization Files Read When a New Shell Is Started**	**Shell Pathname**
Bourne	`/etc/profile`	`$HOME/.profile`		`/bin/sh`
Korn	`/etc/profile`	`$HOME/.profile` `$HOME/.kshrc`	`$HOME/.kshrc`	`/bin/ksh`
C	`/etc/.login`	`$HOME/.cshrc` `$HOME/.login`	`$HOME/.cshrc`	`/bin/csh`

Bourne Shell Initialization File

The `.profile` file is an initialization file that you define in your home directory. The login shell executes the `.profile` file when you log in. You can customize environment variables and terminal settings in the `.profile` file to modify your working environment. You can also instruct the system to initiate applications in the `.profile` file.

Korn Shell Initialization Files

The Korn shell employs two initialization files, which include the `.profile` file and the `.kshrc` file.

The `~/.profile` File

The `.profile` file is an initialization file that resides in your home directory. The login process runs the `.profile` file when you log in. You can customize environment variables and terminal settings in this file to modify your working environment. You can also instruct the system to initiate applications in the `.profile` file.

The `~/.kshrc` File

The `.kshrc` file in your home directory contains shell variables and aliases. The system executes the `.kshrc` file every time you log in and when a `ksh` sub-shell is started.

This file is used to define Korn shell specific settings. To use this file the `ENV` variable must be defined in the `.profile` file.

Note – The `ENV` variable is used to define the Korn shell specific initialization script, which by convention is named `.kshrc`

You typically configure the following settings in a `.kshrc` file:

- Shell prompt definitions (`PS1` and `PS2`)
- Alias definitions
- Shell functions
- History variables
- Shell options (`set -o` *`option`*)

When you make changes to your individual initialization files, the changes take effect the next time you log in. However, if you want the changes to take effect immediately, you can source the `.profile` file and the `.kshrc` file using the dot (`.`) command.

```
$ .  ~/.profile
$
$ .  ~/.kshrc
```

Note – The `/etc/profile` file is a separate, system-wide file that the system administrator maintains. This file sets up tasks that the Korn shell executes for every user who logs in.

C Shell Initialization Files

The C shell employs two initialization files, which include the `.cshrc` file and the `.login` file.

The `~/.cshrc` File

The `.cshrc` file is an initialization file that you define in your home directory. The login shell executes the `.cshrc` file when you log in. You can customize environment variables and terminal settings in the `.cshrc` file to modify your working environment. You can also instruct the shell to initiate applications in the `.cshrc` file.

You typically configure the following settings in the `.cshrc` file:

- Shell prompt definitions (PS1 and PS2)
- Alias definitions
- Shell functions
- History variables
- Shell options (`set option`)

The `~/.login` File

The `.login` initialization file in your home directory has the same functionality as the .cshrc file and has been retained for legacy reasons.

Note – The `/etc/.login` file is a separate, system-wide file that the system administrator maintains. This file sets up tasks that the C shell executes for every user who logs in.

When you make changes to your individual initialization files, the changes take effect the next time you log in. However, if you want the changes to take effect immediately, you can source the `.cshrc` file and the `.login` file using the `source` command.

```
$ source ~/.cshrc
$ source ~/.login
```

Note – Sourcing or reading the `.profile`, `.kshrc`, `.cshrc`, or `.login` files will only update the current working shell.

The `~/.dtprofile` File

Another initialization file, called `.dtprofile`, resides in your home directory. The `.dtprofile` file determines generic and customized settings for the desktop environment. Your variable settings in the `.dtprofile` file can overwrite any default desktop environment settings.

The desktop environment generates a `.dtprofile` file for your home directory the first time you log into the desktop environment.

Note – If the `DTSOURCEPROFILE` variable is set to `true`, then the `dtlogin` causes the shell to read the `.profile` file. If this variable does not exist or it is not set to `true`, then the `.profile` file is not read by the shell.

Each time you log into the desktop environment, the shell reads the `.dtprofile` file first, your `.profile` file next, and then your `.kshrc` file. The shell reads the `.profile` and `.kshrc` files again when you open a console window. The shell also reads the `.kshrc` file when you open a terminal window in the desktop environment.

Configuring the `$HOME/.profile` File

You can configure the `ENV` variable in the `$HOME/.profile` file to instruct the login process to execute the initialization file referenced by the `ENV` variable.

To configure this variable, edit the `$HOME/.profile` file and insert the following lines of text:

```
ENV=$HOME/.kshrc
export ENV
```

To verify that the changes were applied, you can either re-run the file or log out and log back in again.

Configuring the `$HOME/.kshrc` File

The `$HOME/.kshrc` file contains Korn shell specific settings.

To configure the `PS1` variable, edit the `$HOME/.kshrc` file and insert the following lines:

```
PS1="‘hostname‘ $"
export PS1
```

To verify that the changes were applied, you can either re-run the file or log out and log back in again.

Exercise: Working in the Korn Shell

In this exercise, you will answer questions on the material presented in this module.

Preparation

No special preparation is required for this exercise.

RLDC

In addition to being able to use local classroom equipment, this lab was designed to also use equipment located in a remote lab data center. Directions for accessing and using this resource can be found at:

```
http://remotelabs.sun.com/
```

Ask your instructor for the particular SSH configuration file that you should use to access the appropriate remote equipment for this exercise.

Tasks

Complete the following steps. Write your answer in the space provided.

1. Which specific shell characters have special meaning to the shell?

   ____

2. Name some common shell metacharacters.

   ____

3. Which metacharacter is a shell substitute for the home directory of a user?

   ____

4. Navigate to your home directory from your current working directory using the appropriate special metacharacter.
5. Which two metacharacters are often referred to as wildcard characters?

   ____

6. Make sure you are in your home directory. List the contents of all the files and directories in your home directory starting with `d` using a wild card entry.
7. Which character do you use to match a single character, excluding a leading period?

   ____

8. Which character or characters would you use to match a set or range of characters?

   ____

9. Which character or characters would you use to have the shell ignore the special meaning of metacharacters?

   ____

10. What are file descriptors?

    ____

    ____

11. Which symbol or symbols do you use to redirect output and append the output to a file?

    ____

12. Which command redirects standard error messages?

    ____

13. Which symbol or character do you use to connect two or more commands on a single command-line?

    ____

14. Define a variable. Name the kinds of variables used in Korn shell programming.

    ____

    ____

15. Which command do you use to display shell variables and their current values?

    ____

16. Which command do you use to display a list of previously executed commands in the shell?

    ____

17. What is an initialization file?

____

____

18. List the four user initialization files described in this module.

____

____

____

19. Edit your `~/.profile` file to set the `ENV` variable to `$HOME/.kshrc`. Also add `/etc` to your path.
20. Edit your `~/.kshrc` file to set the prompt to be `hostname` and the current directory.
21. Log out and log in again to check that your settings work.

Exercise Summary

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

- Experiences
- Interpretations
- Conclusions
- Applications

Exercise Solutions: Working in the Korn Shell

Complete the following steps:

1. Which specific shell characters have special meaning to the shell?

   Metacharacters.

2. Name some common shell metacharacters.

   Metacharacters include the following: `~`, `-`, `+`, `*`, `?`, *and* `[]`.

3. Which metacharacter is a shell substitute for the home directory of a user?

   The ~ character.

4. Navigate to your home directory from your current working directory using the appropriate special metacharacter.

```
$ cd
$ pwd
$ /export/home/student
```

5. Which two metacharacters are often referred to as wild card characters?

   The `*` *and* `?` *characters.*

6. Make sure you are in your home directory. List the contents of all the directories in your home directory starting with `d` using a wildcard entry.

```
$ pwd
/export/home/student
$ ls d*
dante    dante_1

dir1:
coffees  fruit    trees

dir2:
beans  notes   recipes

dir3:
planets    vegetables

dir4:
roses

dir5:
```

7. Which character do you use to match a single character, excluding a leading period?

   The `?` *character.*

8. Which character or characters would you use to match a set or range of characters?

   The `[]` *character.*

9. Which character or characters would you use to have the shell ignore the special meaning of metacharacters?

   The `\`*,* `'`*, and* `"` *characters.*

10. What are file descriptors?

    File descriptors determine where the input to the command originates and where the output or error messages are sent.

11. Which symbol or symbols do you use to redirect output and append the output to a file?

    The `>>` *character.*

12. Which command redirects standard error messages?

```
$ command 2> filename
```

13. Which symbol or character do you use to connect two or more commands on a single command-line?

    The `|` *character.*

14. Define a variable. Name the kinds of variables used in Korn shell programming.

    A variable refers to a temporary storage area in memory.

    Korn shell programming uses two types of variables:

    - *Variables that are exported to subprocesses*
    - *Variables that are not exported to subprocesses*

15. Which command do you use to display shell variables and their current values?

    The `set` *command.*

16. Which command do you use to display a list of previously executed commands in the shell?

    The `history` *command.*

17. What is an initialization file?

    An initialization file is the file that you use to control the features of your desktop environment.

18. List the four user initialization files described in this module.

    The user initialization files are:

    ```
    ~/.profile
    ~/.kshrc
    ~/.cshrc
    ~/.dtprofile
    ```

19. Edit your `~/.profile` file to set the `ENV` variable to `$HOME/.kshrc`. Also add `/etc` to your path.

    Use the following commands:

    ```
    ENV=$HOME/.kshrc
    PATH=$PATH:/etc
    export ENV PATH
    ```

20. Edit your `~/.kshrc` file to set the prompt to be `hostname` and the current directory.

    Use the following commands:

    ```
    PS1="'hostname': \$PWD $ "
    export PS1
    ```

21. Log out and log in again to check that your settings work.

    Use the following commands:

    ```
    sys-03:/export/home/user $
    sys-03:/export/home/user $ echo $PATH
    /usr/bin:/usr/dt/bin:/usr/openwin/bin:/bin:/etc
    ```

Module 7

Using Basic File Permissions

Objectives

This module introduces file permissions and describes how to view and change permissions.

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

- View file and directory permissions
- Determine file or directory access
- Change the permissions
- Modify the default permissions

Figure 7-1 is the course map that shows how this module fits into the current instructional goal.

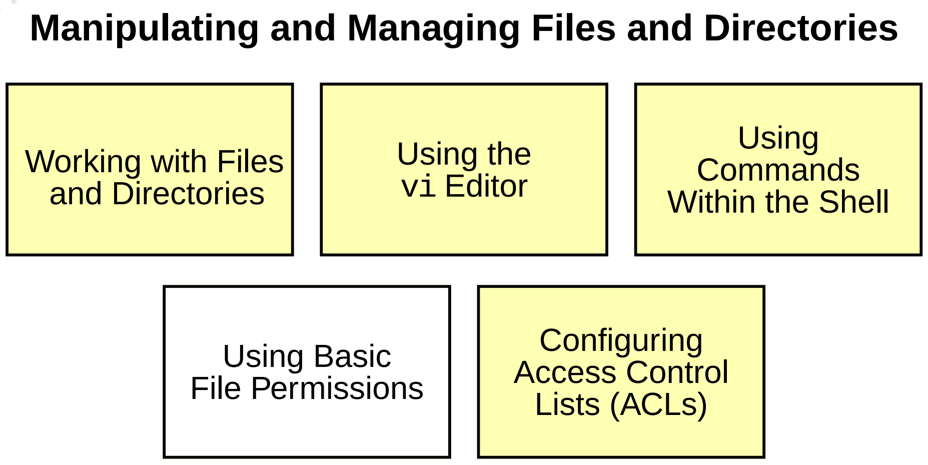

Manipulating and Managing Files and Directories
Working with Files and Directories
Using the vi Editor
Using Commands Within the Shell
Using Basic File Permissions
Configuring Access Control Lists (ACLs)

Figure 7-1 Course Map

Viewing File and Directory Permissions

All files and directories in the Solaris OS have a standard set of access permissions. These access permissions control which files can be accessed by whom, and provides a fundamental level of security for the system. You can use the `ls -l` command and the `ls -n` command to view the permissions for a given file or directory. You can also change the permissions for certain files.

Security Fundamentals

One of the important functions of a secure system is to limit access to authorized users and prevent unauthorized users from accessing the files. Although the system administrator maintains the primary security of a system, users also play a role in keeping the system secure.

The Solaris OS uses two basic measures to prevent unauthorized access to a system and to protect data:

- The first measure is to authenticate a user's login by verifying that the user name and password exist in the `/etc/passwd` and `/etc/shadow` files.
- The second measure is to protect file and directory access automatically. The Solaris OS assigns a standard set of access permissions at the time of creation of files and directories.

Note – The Solaris OS also provides a special user account on every system, called the `root` user. The `root` user, often referred to as the superuser, has complete access to every user account and all files and directories. The `root` user can override the permissions placed on all files and directories.

Viewing Permission Categories

To view the permissions for files and directories, use the `ls -l` command. Figure 7-2 shows the information displayed for the `dante` file.

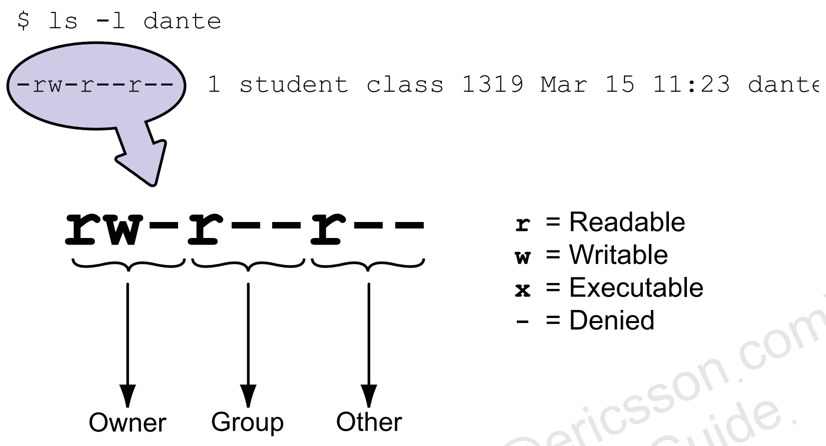


Figure 7-2 Permissions Example

The first field of information displayed by the `ls -l` command is the file type. The file type typically specifies whether it is a file or a directory. A file is represented by a hyphen (`-`). A directory is represented by the letter `d`.

The remaining fields represent three types of users: owner, group, and other.

Table 7-1 describes each type of user with a brief description of each.

Table 7-1 Types of Users

Field	**Description**
Owner	Permissions used by the assigned owner of the file or directory.
Group	Permissions used by members of the group that owns the file or directory.
Other	Permissions used by all users other than the file owner, and members of the group that owns the file or the directory.

Each type of user has three permissions, called a permission set. Each permission set consists of read, write, and execute permissions. Each file or directory has three permission sets for the three types of users. The first permission set represents the owner permissions. The second permission set represents the group permissions. The last permission set represents the other users' permissions.

Permission Types

The read, write, and execute permissions in the owner, group, and other permission sets are represented by the characters `r`, `w`, and `x` respectively. The presence of any of these characters, such as `r`, indicates that the particular permission is granted. The dash (`-`) symbol in place of a character in a permission set indicates that a particular permission is denied.

Owner Permissions

The owner permission set determines the type of access the owner has for the file or directory.

The three characters in this set of permissions represent read, write, and execute permissions for the owner.

Group Permissions

The group permission set determines the type of shared file access for each user belonging to the file owner's group. A group is a collection of users who can access files owned by the group, based on the permission set.

The three characters in this set of permissions represent read, write, and execute permissions.

Note – The system administrator creates and maintains groups in the `/etc/group` file. The system administrator assigns users to groups according to the need for shared file access.

Other Permissions

The other permission set determines the type of access for all other users who have access to the system, but who do not own the file or directory, and are not a member of the file's or directory's group.

Permission Characters and Sets

The read, write, and execute permissions are interpreted differently when assigned to a file than when assigned to a directory.

Table 7-2 shows the permission definitions.

Table 7-2 Permission Characters

Permission	**Character**	**Access for a File**	**Access for a Directory**
Read	`r`	You can display file contents and copy the file.	You can list the directory contents with the `ls` command.
Write	`w`	You can modify the file contents.	You can modify the contents of a directory, such as deleting a file. You must also have the execute permission for this to happen.
Execute	`x`	You can execute the file if it is an executable. You can execute a shell script if you also have read and execute permissions.	You can use the `cd` command to access the directory. If you also have read access, you can run the `ls -l` command on the directory to list contents. If you do not have read access you can run the `ls` command as long as you know the file name.

Note – For a directory to be of general use, it must at least have read and execute permissions.

Table 7-3 shows examples of different permission sets for files and directories.

Table 7-3 Permission Sets

Permissions	**Description**
`-rwx------`	This file has read, write, and execute permissions set for the file owner only. Permissions for group and other are denied.
`dr-xr-x---`	This directory has read and execute permissions set for the directory owner and the group only.
`-rwxr-xr-x`	This file has read, write, and execute permissions set for the file owner. Read and execute permissions are set for the group and other.

When you create a new file or directory, the Solaris OS assigns initial permissions automatically. The initial default permissions for a file and directory are then modified based on the default `umask` value set on the host.

Note – You can assign execute permissions on files with the `chmod` command. The `chmod` command is described later in this module. Execute permissions are not assigned by default when you create a file.

Determining File or Directory Access

The following sections describe how to use the `ls -n` command in the Solaris OS to determine ownership of files and directories.

Using the `ls -n` Command

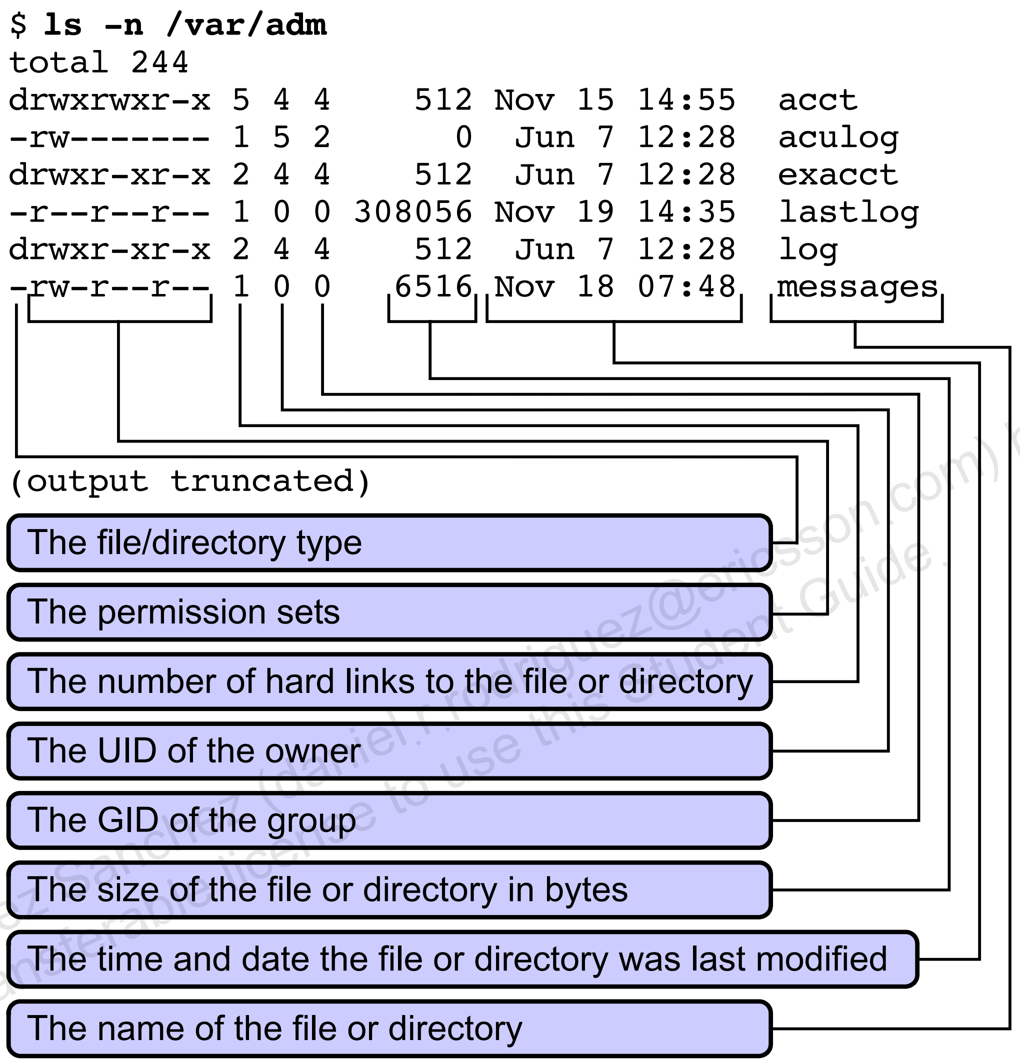
All files and directories have an associated user identification number (UID) and a group identification number (GID). The UID identifies the user who owns the file or directory. The GID identifies the group of users who own the file or directory. A file or directory can belong to only one group at a time. The Solaris OS uses these numbers to track ownership and group membership of files and directories.

To view the UIDs and GIDs, run the `ls -n` command on the `/var/adm` directory.

```
$ ls -n /var/adm
total 244
drwxrwxr-x   5 4        4            512 Nov 15 14:55 acct
-rw-------   1 5        2              0 Jun  7 12:28 aculog
drwxr-xr-x   2 4        4            512 Jun  7 12:28 exacct
-r--r--r--   1 0        0         308056 Nov 19 14:35 lastlog
drwxr-xr-x   2 4        4            512 Jun  7 12:28 log
-rw-r--r--   1 0        0           6516 Nov 18 07:48 messages

... (output truncated)
```

Figure 7-3 describes the fields in the output of the `ls -n` command.



```
$ ls -n /var/adm
total 244
drwxrwxr-x 5 4 4    512 Nov 15 14:55  acct
-rw------- 1 5 2      0  Jun 7 12:28  aculog
drwxr-xr-x 2 4 4    512  Jun 7 12:28  exacct
-r--r--r-- 1 0 0 308056 Nov 19 14:35  lastlog
drwxr-xr-x 2 4 4    512  Jun 7 12:28  log
-rw-r--r-- 1 0 0   6516 Nov 18 07:48  messages
(output truncated)
```

The file/directory type
The permission sets
The number of hard links to the file or directory
The UID of the owner
The GID of the group
The size of the file or directory in bytes
The time and date the file or directory was last modified
The name of the file or directory

Figure 7-3 Description of the Output of the `ls -n` Command

Note – A hard link is a pointer that shows the number of files or directories a particular file is linked to within the same file system.

Determining Permissions

When a user attempts to access a file or directory, the Solaris OS compares the UID of the user to the UID of the file or directory. If the UIDs match, the permission set for the owner determines whether the owner has access to the file or directory.

If the UIDs do not match, the Solaris OS compares the user’s GID and the GID of the file or directory. If these numbers match, the group permissions apply.

If the GIDs do not match, the Solaris OS uses the permission set for other to determine file and directory access.

Figure 7-4 shows the decision tree for determining file and directory permissions.

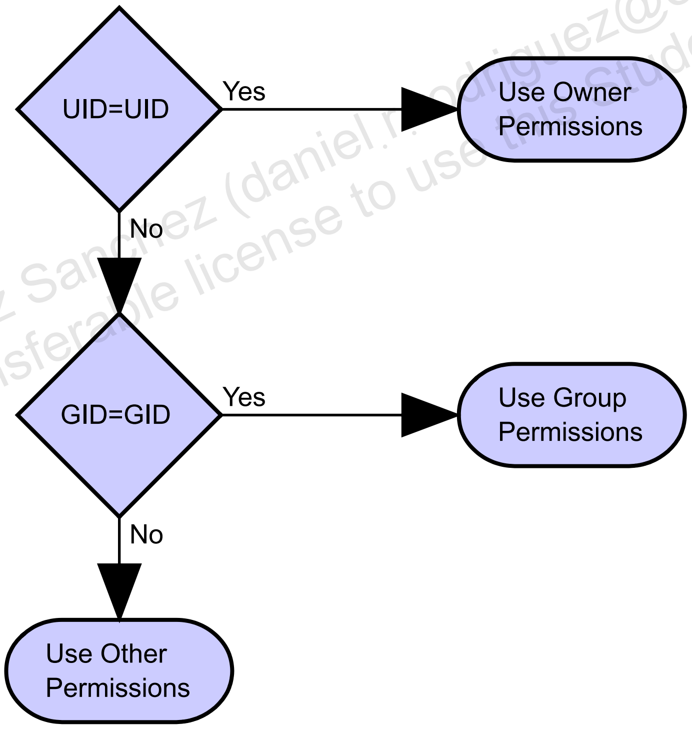


Figure 7-4 Process for Determining Permissions

Changing the Permissions

You can change the permissions set on files or directories using the `chmod` command. Either the owner of the file or directory or the `root` user can use the `chmod` command to change permissions.

Permission Modes

The `chmod` command can change permissions specified in either symbolic mode or octal mode.

- Symbolic mode — uses combinations of letters and symbols to add or remove permissions for each type of user.
- Octal mode — uses octal numbers to represent each permission. Octal mode is also referred to as absolute mode.

Changing Permissions in Symbolic Mode

The syntax for the `chmod` command in the symbolic mode is:

```
chmod symbolic_mode filename
```

The *`symbolic_mode`* option consists of three parts: the user category (owner, group, or other) affected, the function performed, and the permissions affected. For example, if the option is `g+x`, the executable permission is added for the group.

Figure 7-5 shows the `symbolic_mode` options.

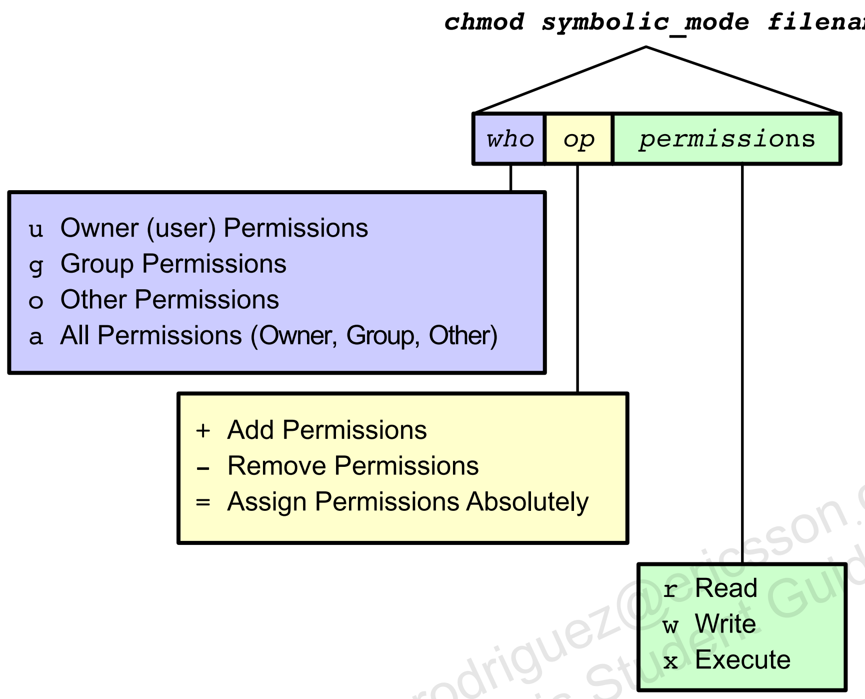

chmod symbolic_mode filename
who
op
permissions
u Owner (user) Permissions
g Group Permissions
o Other Permissions
a All Permissions (Owner, Group, Other)
+ Add Permissions
- Remove Permissions
= Assign Permissions Absolutely
r Read
w Write
x Execute

Figure 7-5 Symbolic Mode Command Syntax

The following examples show you how to modify permissions on files and directories using symbolic mode.

To remove the read permission for other users, run the following commands:

```
$ ls -l dante
-rw-r--r--   1 student    class       1319 Jan 22 14:51 dante
$ chmod o-r dante
$ ls -l dante
-rw-r-----   1 student    class       1319 Jan 22 14:51 dante
$
```

To remove the read permission for the group, run the following commands:

```
$ chmod g-r dante
$ ls -l dante
-rw-------   1 student    class       1319 Jan 22 14:51 dante
$
```

To add an execute permission for the owner and a read permission for the group and other, run the following commands:

```
$ chmod u+x,go+r dante
$ ls -l dante
-rwxr--r--   1 student    class       1319 Jan 22 14:51 dante
$
```

To assign read and write permissions for owner, group, and other, run the following commands:

```
$ chmod a=rw dante
$ ls -l dante
-rw-rw-rw-   1 student    class       1319 Jan 22 14:51 dante
$
```

Changing Permissions in Octal Mode

The `chmod` command syntax in octal mode is:

```
chmod octal_mode filename
```

The *`octal_mode`* option consists of three octal numbers, from `0`–`7`, which represent a combination of permissions for the file or directory.

Table 7-4 shows the octal numbers for each individual permission.

Table 7-4 Assigned Octal Values for Permissions

Octal Value	Permission
4	Read
2	Write
1	Execute

These numbers are combined into one number for each permission set.

Table 7-5 shows the octal numbers that represent a combined set of permissions.

Table 7-5 Octal Digits for Permission Sets

Octal Value	**Permission Sets**	**Binary**
7	`rwx`	111 (4+2+1)
6	`rw-`	110 (4+2+0)
5	`r-x`	101 (4+0+1)
4	`r--`	100 (4+0+0)
3	`-wx`	011 (0+2+1)
2	`-w-`	010 (0+2+0)
1	`--x`	001 (0+0+1)
0	`---`	000 (0+0+0)

You can modify the permissions for each category of users by combining octal numbers. The first octal number defines owner permissions, the second octal number defines group permissions, and the third octal number defines other permissions.

Table 7-6 shows the permission sets for the three-digit octal numbers.

Table 7-6 Combined Values and Permissions

Octal Mode	**Permissions**
644	`rw-r--r--`
751	`rwxr-x--x`
775	`rwxrwxr-x`
777	`rwxrwxrwx`

The chmod command fills in any missing octal digits to the left with zeros. For example:

```
$ chmod 44 dante
$ ls -l dante
----r--r-- 1 student    class       1319 Jan 22 14:51 dante
$
```

The previous example shows that the chmod 44 dante command is interpreted as chmod 044 dante

Caution – Not using the correct octal values or leaving off one or more of the values can lead to unwanted access to files or directories.

The following examples show how to modify permissions on files and directories using the octal mode.

Note – Each example builds on the resulting permissions from the previous example.

To set permissions so that the owner, group, and other have read and execute access only, enter the following commands:

```
$ chmod 555 dante
$ ls -l dante
-r-xr-xr-x   1 student    class       1319 Jan 22 14:51 dante
$
```

To change owner and group permissions to include write access, enter the following commands:

```
$ chmod 775 dante
$ ls -l dante
-rwxrwxr-x   1 student    class       1319 Jan 22 14:51 dante
$
```

To change the group permissions to read and execute only, enter the following commands:

```
$ chmod 755 dante
$ ls -l dante
-rwxr-xr-x   1 student    class       1319 Jan 22 14:51 dante
$
```

Modifying Default Permissions

Every new file or directory has a set of default permissions assigned to it at the time of creation. The user mask affects the default file permissions assigned to the file or directory. You can set the user mask using the `umask` command in a user initialization file. You can modify the default permissions set at the time of creation, using the `umask` utility.

The `umask` Utility

The `umask` utility affects the initial permissions for files and directories when the files and directories are created. The `umask` utility is a three-digit octal value that is associated with the read, write, and execute permissions. 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.

In the Solaris OS, the default `umask` value is `022`.

To view the `umask` value, run the `umask` command:

```
$ umask
022
$
```

The Solaris OS assigns initial permission values automatically when files and directories are created.

The initial permission value specified by the system at the time of file creation is `666` (`rw-rw-rw-`).

The initial permission value specified by the system for a directory at the time of its creation is `777` (`rwxrwxrwx`).

To determine the `umask` value you want to set, remove the value of the permissions you want from `666` (for a file) or `777` (for a directory). The remainder is the value to use with the `umask` command. For example, suppose you want to change the default mode for files to `644` (`rw-r--r--`). The difference between `666` and `644` is `022`, which is the value you would use as an argument to the `umask` command.

Table 7-7 shows the file and directory permissions that are created for each of the umask octal values. You can also determine the umask value you want to set using the values listed in the table below.

Table 7-7 File and Directory Permissions for umask Values

umask Octal Value	File Permissions	Directory Permissions
0	rw-	rwx
1	rw-	rw-
2	r--	r-x
3	r--	r--
4	-w-	-wx
5	-w-	-w-
6	---	--x
7	---	--- (none)

To set the default file permissions in a user initialization file to rw-rw-rw-, run the following command:

```
umask 000
```

Applying the umask Utility

You can calculate the default permissions for new files and directories by applying the umask value to the initial value specified by the system in the octal mode.

For example, the initial permissions for a new file in the symbolic mode are as follows:

```
rw-rw-rw-
```

This set of permissions corresponds to read-write access for the owner, group, and other. This value is represented in the octal mode:

```
420420420 or 666
```

Use the default `umask` value of `022` to mask out the write permission for the group and other.

The result in the octal mode is:

```
420400400 or 644
```

The result in the symbolic mode is derived, as shown in Table 7-8.

Table 7-8 Symbolic Mode Permission Fields

Permission Field	**Description**
`rw-rw-rw-`	Initial value specified by the system for a new file
`----w--w-`	Default `umask` utility value to be removed
`rw-r--r--`	Default permissions assigned to newly created files

When you mask out certain permissions from the initial value, the default permissions assigned to the new files remain.

All newly created files are assigned read and write access for the owner, and read access for the group and other.

You can apply this same process to determine the default permissions when you create new directories.

For directories, the initial value specified by the system is:

```
rwxrwxrwx
```

This corresponds to read, write, and execute access for the owner, group, and other. This value is represented in octal mode as:

```
421421421 or 777
```

Use the default `umask` value of `022` to mask out the write permission for the group and other.

The result in the octal mode is:

```
421401401 or 755
```

The result in the symbolic mode is derived as shown in Table 7-9.

Table 7-9 Symbolic Mode Permission Fields

Permission Field	**Description**
`rwxrwxrwx`	Initial value specified by the system for a new directory
`----w--w-`	Default `umask` utility value to be removed
`rwxr-xr-x`	Default permissions set for newly created directories

When you mask out certain permissions from the initial value, the default permissions assigned to the new directories remain.

All newly created directories are assigned read, write, and execute access for the owner, and read and execute access for the group and other.

Changing the umask Value

Some users require a more secure `umask` value of `027`, which assigns the following access permissions to newly-created files and directories.

- Files have read and write permissions for the owner, read permission for the group, and no permissions for other.

  `rw-r-----`

- Directories have read, write, and execute permissions for the owner, read and execute permissions for the group, and no permissions for other.

  `rwxr-x---`

You can change the `umask` value to a new value on the command-line. For example, to change the `umask` value to `027` and verify the new value, run the command:

```
$ umask 027
$ umask
027
$
```

The new `umask` value affects only those files and directories that are created from this point forward. However, if the user logs out of the system, the new value (`027`) is replaced by the old value (`022`) on subsequent logins because the `umask` value was changed using the command-line.

Exercise: Changing File Permissions

In this exercise, you practice reading permissions on files and changing permissions using the symbolic mode and the octal mode.

Preparation

Ensure that the umask value is set to 022 on your system. If not, set the umask value to 022 by running the following command:

```
$ umask 022
```

RLDC

In addition to being able to use local classroom equipment, this lab was designed to also use equipment located in a remote lab data center. Directions for accessing and using this resource can be found at:

`http://remotelabs.sun.com/`

Ask your instructor for the particular SSH configuration file that you should use to access the appropriate remote equipment for this exercise.

Tasks

To view permissions on files and change permissions, complete the following steps:

1. Perform the following commands in your home directory:

```
$ mkdir perm
$ cd /etc
$ ls -l group motd shadow vfstab
$ cp group motd shadow vfstab ~/perm
```

When trying to copy the `shadow` file, the error message `cp: cannot open shadow: Permission denied` appears. Why?

____

____

```
$ ls -l ~/perm
$ cd
$ cp -r /etc/skel perm
```

2. Change to the perm directory, and list the contents of the directory.

```
$ cd perm
$ ls -l
```

Fill in the permission sets for each file in Table 7-10, and write the three-digit octal number that represents the combined set of permissions.

Table 7-10 Permission Sets

File or Directory	Permissions			Octal Value
	Owner	Group	Other	
group				
motd				
skel				
vfstab				

3. Create a new file and a new directory.
   a. What are the default permissions given to the new file?

      ____

   b. What are the default permissions given to the new directory?

      ____

4. Distinguish between the symbolic mode and the octal mode.

   ____

   ____

5. Using the symbolic mode, add write permission for the group to the motd file.

   ____

6. Using the octal mode, change the permissions on the motd file to -rwxrw----.

   ____

7. Using the octal mode, add write permission for other on the file named group.

   ____

8. Identify the GID and UID for the `motd` file. Which command did you use?

    ____

    ____

9. Create a new file called `memo` in your `dir4` directory.

    ____

10. Remove the read permission for the owner from the `memo` file in the `dir4` directory. Use either the symbolic mode or the octal mode.

    ____

    a. What happens when you try to use the `cat` command to view the `memo` file?

        ____

    b. What happens when you try to copy the `memo` file?

        ____

        ____

11. What is the function of the `umask` utility? What is the default `umask` that exists on your system?

    ____

12. Change the `umask` to `027`. Which command did you run?

    ____

13. Create a new file and a new directory. Record the access permissions. Which command did you run?

    ____

    ____

14. Change the `umask` back to `022`.

15. Create a new file and a new directory.

```
$ touch test2file
$ mkdir test2dir
```

16. Record the access permissions.

```
$ ls -l test2file
```

```
$ ls -ld test2dir
```

Exercise Summary

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

- Experiences
- Interpretations
- Conclusions
- Applications

Exercise Solutions: Changing File Permissions

To view permissions on files and change permissions, complete the following steps:

1. Perform the following commands in your home directory:

```
$ mkdir perm
$ cd /etc
$ ls -l group motd shadow vfstab
-rw-r--r--   1 root     sys          290 Jun  5 09:40 group
-rw-r--r--   1 root     sys           49 Apr  6  2002 motd
-r--------   1 root     sys          795 Jun 28 15:25 shadow
-rw-r--r--   1 root     sys          257 Jun  5 10:01 vfstab
$ cp group motd shadow vfstab ~/perm
```

When trying to copy the `shadow` file, the error message `cp: cannot open shadow: Permission denied` appears. Why?

Only the owner of this file, who is the `root` *user, has read permission.*

```
$ ls -l ~/perm
-rw-r--r--   1 root     sys          290 Jun  5 09:40 group
-rw-r--r--   1 root     sys           49 Apr  6  2002 motd
-rw-r--r--   1 root     sys          257 Jun  5 10:01 vfstab
$ cd
$ cp -r /etc/skel perm
```

2. Change to the `perm` directory, and list the contents of the directory.

```
$ cd perm
$ ls -l
-rw-r--r--   1 student    class        290 Jul 29 14:34 group
-rw-r--r--   1 student    class         49 Jul 29 14:34 motd
drwxr-xr-x   2 student    class        512 Jul 29 14:34 skel
-rw-r--r--   1 student    class        420 Jul 29 14:34 vfstab
```

Fill in the permission sets for each file in Table 7-11 on page 7-25, and write the three-digit octal number that represents the combined set of permissions.

Table 7-11 Permission Sets

File or Directory	Permissions			Octal Value
	Owner	Group	Other	
`group`	`rw-`	`r--`	`r--`	644
`motd`	`rw-`	`r--`	`r--`	644
`skel`	`rwx`	`r-x`	`r-x`	755
`vfstab`	`rw-`	`r--`	`r--`	644

3. Create a new file and a new directory.

   a. What are the default permissions given to the new file?

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

   b. What are the default permissions given to the new directory?

```
rwxr-xr-x
```

4. Distinguish between the symbolic mode and the octal mode.

   The symbolic mode uses combinations of letters and symbols to add or remove permissions for each type of user.

   The octal mode uses octal numbers to represent each permission. The octal mode is also referred to as the absolute mode.

5. Using the symbolic mode, add write permission for the group to the `motd` file.

```
$ chmod g+w motd
$ ls -l
```

6. Using the octal mode, change the permissions on the `motd` file to `-rwxrw----`.

```
$ chmod 760 motd
$ ls -l
```

7. Using the octal mode, add write permission for other on the file named `group`.

```
$ chmod 646 group
$ ls -l
```

8. Identify the GID and UID for the `motd` file. Which command did you use?

```
$ ls -n motd
-rwxrw----   1  11001    10         49 Nov 19 15:16 motd
```

9. Create a new file called `memo` in your `dir4` directory.

```
$ touch ~/dir4/memo
```

10. Remove the read permission for the owner from the `memo` file in the `dir4` directory. Use either the symbolic mode or the octal mode.

```
$ chmod u-r ~/dir4/memo
```

or

```
$ chmod 244 ~/dir4/memo
```

a. What happens when you try to use the `cat` command to view the `memo` file?

You cannot use the `cat` *command, because read permission has been removed for the user. Even though you are part of the group, the permissions are viewed in the order in which they appear. The following message appears:*

```
cat: cannot open /export/home/student/dir4/memo
```

b. What happens when you try to copy the `memo` file?

You cannot copy the file, because the user has no read permission. The following message appears:

```
cp: cannot open /export/home/student/dir4/memo: Permission denied
```

11. What is the function of the `umask` utility? What is the default `umask` that exists on your system?

The `umask` *utility modifies the default permissions set for files and directories at the time of creation. To view the default* `umask` *value on your system, run the* `umask` *command.*

```
$ umask
022
$
```

12. Change the `umask` to `027`. Which command did you run?

```
$ umask 027
```

13. Create a new file and a new directory. Record the access permissions. Which command did you run?

```
$ touch testfile
$ mkdir testdir
$ ls -l testfile
-rw-r-----
$ ls -ld testdir
drwxr-x---
```

14. Change the `umask` back to `022`.

```
$ umask 022
```

15. Create a new file and a new directory.

```
$ touch test2file
$ mkdir test2dir
```

16. Record the access permissions.

```
$ ls -l test2file
-rw-r--r--
$ ls -ld test2dir
drwxr-xr-x
```

Notes:

Module 8

Configuring Access Control Lists (ACLs)

Objectives

This module describes how to create and configure unique access permissions on files and directories using access control lists (ACLs). Upon completion of this module, you should be able to:

- Describe ACLs
- Configure ACLs using the command-line
- Configure ACLs using the File Manager GUI

Figure 8-1 is the course map that shows how this module fits into the current instructional goal.

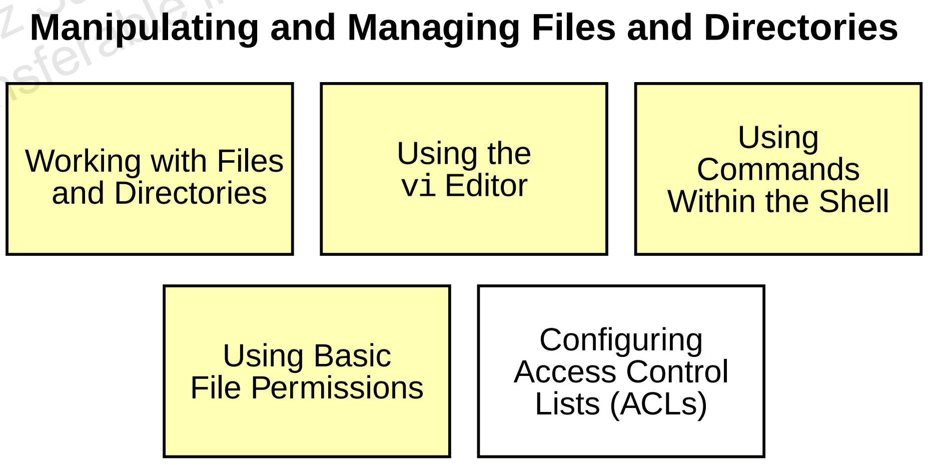

Manipulating and Managing Files and Directories
Working with Files and Directories
Using the vi Editor
Using Commands Within the Shell
Using Basic File Permissions
Configuring Access Control Lists (ACLs)

Figure 8-1 Course Map

Access Control Lists

The Solaris OS file protection provides read, write, and execute permissions for the three user classes: file owner, file group, and others. ACLs provide greater data access control for each file or directory. ACLs enable you to control file permissions more finely.

Limitations of the Basic Solaris OS Permissions

The Solaris OS file permissions do not allow the owner of the file or directory to grant or deny access to other users.

ACL Benefits

An ACL is used by the owner of a file or directory to grant or deny specific user access, using the three user classes of user, group, and other.

ACL Commands

Table 8-1 shows you which command and options to enter when you want to view or set ACLs for a file or directory.

Table 8-1 ACL Commands and Descriptions

Command and Option	Description
`getfacl` *`filename(s)`*	Displays ACL entries for files
`setfacl` *`acl_entries filename`*	Configures ACL entries on files

Viewing ACL Entries

Each ACL entry has the following syntax:

```
entry-type:[UID or GID]:perm
```

where:

- `entry-type` – Specifies the scope of the file permissions to the owner, owner's group, specific users, additional groups, or the ACL mask.
- `UID or GID` – Specifies the user's name or user's identification number (UID), or the group's name or group's identification number (GID).
- `perm` – Symbolically specifies permissions for `entry-type` using `r`, `w`, `x`, and `-`, or using octal values from `0` to `7`.

Note – ACL entries are labeled as `acl_entry` in all the command-line examples.

You use the `getfacl` command to display the contents of ACL entries for a file or directory. The syntax of the command is:

```
getfacl [-a] filename1 [filename2 ...]
```

where:

`-a`	Displays the file name, file owner, file group, and ACL entries for the specified file or directory
`filename`	Specifies one or more files or directories

```
$ getfacl file1

$ file: file1
$ owner: student
$ group: class
user::rw-
group::r--            #effective:r--
mask:r--
other:r--
$
```

ACL Entry Types

Table 8-2 shows ACL syntax types.

Table 8-2 ACL Entry Types

Entry Type	**Description**
`u[ser]::`*`perm`*	The permissions for the file owner.
`g[roup]::`*`perm`*	The permissions for the owner's group.
`o[ther]:`*`perm`*	The permissions for users other than the owner or members of the owner's group.
`u[ser]:`*`UID`*`:`*`perm`* `or` `u[ser]:`*`username`*`:`*`perm`*	The permissions for a specific user. The *`username`* must exist in the `/etc/passwd` file.
`g[roup]:`*`GID`*`:`*`perm`* `or` `g[roup]:`*`groupname`*`:`*`perm`*	The permissions for a specific group. The *`groupname`* must exist in the `/etc/group` file.
`m[ask]:`*`perm`*	The ACL mask indicates the maximum effective permissions allowed for all specified users and groups. The mask does not set the permissions for the owner or others. You can use the mask as a quick way to change effective permissions for all the specific users and groups.

Determining if a File Have an ACL

There are two types of ACLs: trivial and non-trivial. You can use the `ls -l` command to see which files or directories have a non-trivial ACL entry. The `ls` command does not display the actual list of ACL entries. To display the list of ACL entries, use the `getfacl` command.

When viewing the output of the `ls -l` command, if a file has an non-trivial ACL entry, a plus (+) sign appears at the end of the permission field.

```
$ pwd
/export/home/student
$ ls -l
total 0
-rw-r--r--   1 student    class          0 Jan 22 13:40 file1
-rw-r--r--+  1 student    class          0 Jan 22 13:40 file2
```

In this example, the lack of a `+` sign for the `file1` file shows that it possesses a trivial ACL entry. The presence of a `+` sign for the file named `file2` indicates that this file has a non-trivial ACL entry. This indicates that permissions were customized.

The following examples show the output of the `getfacl` command:

```
$ getfacl file1
# file: file1
# owner: student
# group: class
user::rw-
group::r--            #effective:r--
mask:r--
other:r--
```

No custom ACL entries are configured, hence the ACL is trivial. The permissions listed in the output from the `getfacl` command are the same as the current permissions for the file or directory.

```
$ getfacl file2
# file: file2
# owner: student
# group: class
user::rw-
user:acluser:rwx          #effective:r--
group::r--              #effective:r--
mask:r--
other:r--
```

If a custom ACL entry is configured, the ACL is non-trivial. The `file2` file has a custom ACL entry for the `acluser` user.

The `acluser` user is given a custom ACL entry that permits read, write, and execute permissions (`rwx`) for `file2`. However, the ACL mask on `file2` allows only read permission (`r--`). Therefore, `acluser` has an effective permission of only `r--`.

The `setfacl` Command

Table 8-3 shows you which command and options to enter when you want to administer ACLs for a file or directory.

Table 8-3 ACL Command Options and Descriptions

Command/Option	**Description**
`setfacl -m` *`acl_entries filename`*	Creates or modifies ACL entries on files
`setfacl -s` *`acl_entries filename`*	Substitutes new ACL entries for old ACL entries
`setfacl -d` *`acl_entries filename`*	Deletes one or more ACL entries on files
`setfacl -r` *`filename`*	Recalculates the ACL mask based on the ACL entries, when used with the `-m` or `-s` option

Configuring ACLs Using the Command-Line

You can set ACLs using the command-line or the File Manager GUI. These tools enable you to do the following:

- Modify an ACL
- Substitute an ACL
- Delete an ACL
- Recalculate an ACL mask

The ACL permission bits define specific user or specific group permissions that are available, subject to the ACL mask. The ACL mask defines the maximum set of effective permissions that are available for an ACL entry. An ACL mask setting of `rw–` (or octal number `6`) on a file permits read and write permission on the file but does not permit execute permission on this file.

Note – In the previous context, the ACL mask is not directly related to the shell's `umask` value in any way. The `umask` value globally controls the initial permissions that are set for files or directories for each shell. The ACL mask controls the effective permissions granted for that file or directory. Each file or directory has its own ACL mask.

Configuring or Modifying an ACL

The most common method used to configure an ACL is to modify the ACL. To modify ACL entries on a file, use the `setfacl` command. The syntax of the command is:

```
setfacl -m acl_entry filename
```

where:

`-m`	Modifies the existing ACL entry.
`acl_entry`	Specifies a list of modifications to apply to ACLs for one or more files, directories, or both. See Table 8-2 on page 8-4 for a description of available ACL entries.
`filename`	Specifies one or more files or directories.

Note – To verify the new ACL entries, use the `getfacl` command.

The following example shows you how to add a new ACL entry to a file with existing ACL entries.

```
$ getfacl file2

# file: file2
# owner: student
# group: class
user::rw-
user:acluser:rwx          #effective:r--
group::r--              #effective:r--
mask:r--
other:r--
$ setfacl -m u:acluser2:7 file2
$ getfacl file2

# file: file2
# owner: student
# group: class
user::rw-
user:acluser:rwx          #effective:r--
user:acluser2:rwx          #effective:r--
group::r--              #effective:r--
mask:r--
other:r--
```

Even though the ACL entries for `acluser` and `acluser2` request read, write, and execute permissions, the ACL mask does not allow write and execute permissions for both entries.

From the previous example you can see that although `acluser` and `acluser2` have `rwx` permission on `file2` they have effectively only read permission. The following example shows you how to modify the mask.

```
$ setfacl -m m:rwx file2
$ getfacl file2

# file: file2
# owner: student
# group: class
user::rw-
user:acluser:rwx          #effective:rwx
user:acluser2:rwx          #effective:rwx
group::r--              #effective:rwx
mask:rwx
other:r--
```

Recalculating an ACL Mask

You can control globally the effective permissions of a custom ACL entry using the ACL mask. Examples in this module show that the ACL mask sometimes does not provide the requested permissions that you set to be effective.

You recalculate the ACL mask to provide all of the requested permissions in the list of ACL entries to be effective. After recalculating the ACL mask, the effective permission of each ACL entry provides the full set of requested permissions for that entry.

The following example shows how to recalculate the ACL mask entry:

```
setfacl -r -m acl_entry filename...
```

where:

`-r`	Recalculates the ACL mask entry to allow maximum effective permissions for every ACL entry.
`-m`	Modifies the existing ACL entry.
`acl_entry`	Specifies a list of modifications to apply to the ACLs for one or more files, directories, or both. ACL entries can also be added, modified, or deleted in addition to the recalculation of the mask. See Table 8-2 on page 8-4 for a description of available ACL entries.
`filename`	Specifies one or more files or directories.

Caution – Using the `chmod` command on a file that already has an ACL on it will recalculate the mask. Take care that access to a resource is not denied or incorrect access is not granted.

```
$ getfacl file1

# file: file1
# owner: student
# group: class
user::rwx
user:acluser:rwx          #effective:rw-
group::rw-              #effective:rw-
mask:rw-
other:r--
```

```
$ setfacl -r -m u:acluser:7 file1
$ getfacl file1

# file: file1
# owner: student
# group: class
user::rwx
user:acluser:rwx          #effective:rwx
group::rw-              #effective:rw-
mask:rwx
other:r--
```

Note – The file owner and other permissions are not considered when recalculating the ACL mask.

An example of the effect of using the `chmod` command on a file that already has an ACL set.

```
$ getfacl testfile
# file: testfile
# owner: student
# group: class
user::rw-
user:acluser:rw-           #effective:r--
group::r--              #effective:r--
mask:r--
other:r--
$ chmod 664 testfile
$ getfacl testfile
# file: testfile
# owner: student
# group: class
user::rw-
user:acluser:rw-          #effective:rw-
group::rw-              #effective:rw-
mask:rw-
other:r--
```

By changing the group permission to read-write, this has recalculated the mask and given `acluser` read and write permission for the `testfile`.

Substituting an ACL

To replace the entire ACL on a file from the command-line, you must specify at least the basic set of ACL entries: user, group, other, and ACL mask permissions. Use the `setfacl` command with the following options to substitute an ACL on a file:

```
setfacl -s u::perm,g::perm,o:perm,m:perm,[u:UID:perm],[g:GID:perm] filename
```

where:

`-s`	Specifies a substitution is being made for the entire ACL contents.
`acl_entry`	Specifies which ACL entry (from a list of one or more ACL entries) to modify on the file or directory. See Table 8-2 on page 8-4 for a description of available ACL entries.
`filename`	Specifies one or more files or directories.

The following example shows that no non-trivial ACL entries currently exist on `file1` and `file2`:

```
$ ls -l
total 0
-rw-r--r--   1 student    class       0 Jan 18 15:44 file1
-rw-r--r--   1 student    class       0 Jan 18 15:44 file2
```

To display the trivial ACL permissions for `file1`, enter the `getfacl` command:

```
$ getfacl file1

# file: file1
# owner: student
# group: class
user::rw-
group::r--              #effective:r--
mask:r--
other:r--
```

The following example shows you how to substitute an ACL on the file named `file1`. The ACL permissions are configured as follows:

- The file owner has read, write, and execute permissions
- The group has read and write permissions
- The other users have read-only permissions
- The user named `acluser` has read, write, and execute permissions on the file
- The ACL mask has read and write permissions

Enter the `setfacl` command with the following options to substitute an ACL on `file1`:

```
$ setfacl -s u::rwx,g::rw-,o:r--,m:rw-,u:acluser:rwx file1
```

The `ls -l` command shows that an ACL exists on the file named `file1`.

```
$ ls -l
total 0
-rwxrw-r--+  1 student    class      0 Jan 18 15:44 file1
-rw-r--r--   1 student    class      0 Jan 18 15:44 file2
$ getfacl file1

# file: file1
# owner: student
# group: class
user::rwx
user:acluser:rwx          #effective:rw-
group::rw-              #effective:rw-
mask:rw-
other:r--
```

The following example shows how to substitute an ACL on the file2 file, using octal notations to establish the ACL entries. Before you replace the entire ACL, use the getfacl command to display the ACL for file2.

```
$ getfacl file2

# file: file2
# owner: student
# group: class
user::rw-
group::r--             #effective:r--
mask:r--
other:r--

$ setfacl -s u::7,g::6,o:4,m:6,u:acluser:7 file2
```

After you substitute the ACL permissions using the setfacl command, use the ls -l command to verify if an ACL exists on the file named file2.

```
$ ls -l
total 0
-rwxrw-r--+  1 student    class      0 Jan 18 15:44 file1
-rwxrw-r--+  1 student    class      0 Jan 18 15:44 file2
```

The output of the getfacl command on file1 and file2 shows the ACLs are identical, regardless of which method is used to create the ACL.

```
$ getfacl file1 file2

# file: file1
# owner: student
# group: class
user::rwx
user:acluser:rwx          #effective:rw-
group::rw-             #effective:rw-
mask:rw-
other:r--

# file: file2
# owner: student
# group: class
user::rwx
user:acluser:rwx          #effective:rw-
group::rw-             #effective:rw-
mask:rw-
other:r--
```

Deleting an ACL

You use the `setfacl` command to delete an ACL. When deleting an ACL, you specify the entry-type and the UID or GID that you want to delete. You cannot delete the ACL entries for the file owner, file group owner, or the ACL mask. The syntax of the command is:

```
setfacl -d acl_entry filename
```

where:

`-d`	Deletes one or more *`acl_entry`* arguments.
`acl_entry`	Specifies which ACL entry to delete in the file or directory. See Table 8-2 on page 8-4 for a description of available ACL entries.
`filename`	Specifies the file or directory from which to delete the *`acl_entry`* argument.

The outputs of the `pwd` and `ls -l` commands show the current working directory and its contents.

```
$ pwd
/export/home/student
$ ls -l
total 0
-rw-r--r--   1 student    class          0 Jan 22 13:40 file1
-rw-r--r--+  1 student    class          0 Jan 22 13:40 file2
$
```

The output of the `getfacl` command shows the current ACL configuration for `file2`.

```
$ getfacl file2

# file: file2
# owner: student
# group: class
user::rw-
user:acluser:rwx          #effective:r--
user:acluser2:rwx          #effective:r--
group::r--              #effective:r--
mask:r--
other:r--
$
```

The following example shows how to delete an ACL entry from the file named `file2`:

```
$ setfacl -d u:acluser2 file2
$
```

The output from the `ls -l` command shows that the deleted ACL entry was not the only ACL entry on the `file2` file. To verify this output, enter the `getfacl` command.

```
$ ls -l
total 0
-rw-r--r--   1 student    class           0 Jan 22 13:40 file1
-rw-r--r--+  1 student    class           0 Jan 22 13:40 file2
$
```

The ACL entry for `acluser2` is deleted, but the ACL entry for `acluser` remains.

```
$ getfacl file2

# file: file2
# owner: student
# group: class
user::rw-
user:acluser:rwx          #effective:r--
group::r--               #effective:r--
mask:r--
other:r--
$
```

When you remove the last ACL entry on a file, the output of the `ls -l` command reports that the file has a trivial ACL (shown by the absence of a + symbol in the output).

```
$ setfacl -d u:acluser file2
$ ls -l
total 0
-rw-r--r--   1 student    class           0 Jan 22 13:40 file1
-rw-r--r--   1 student    class           0 Jan 22 13:40 file2
$
```

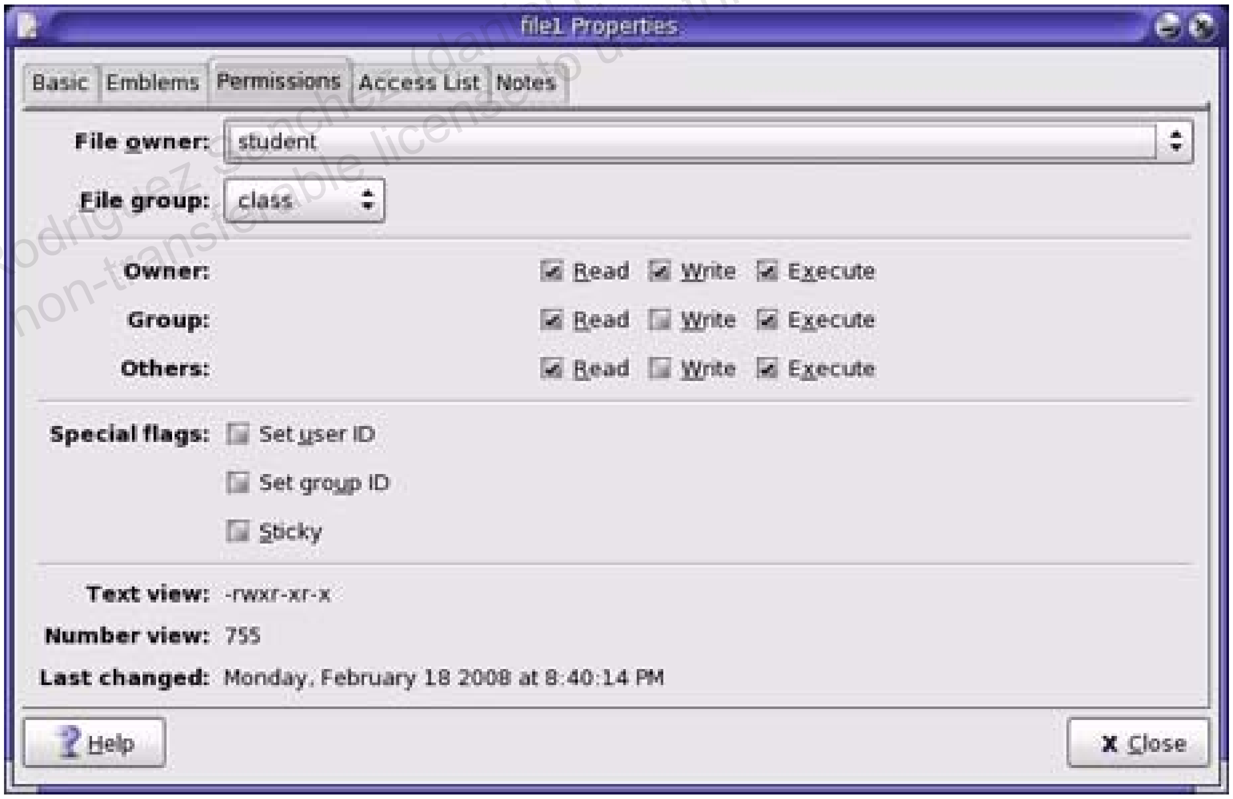
Configuring ACLs Using the File Manager GUI

The File Manager GUI, contains mechanisms to perform the following tasks:

- Display ACLs
- Add ACLs
- Change ACLs
- Delete ACLs

Displaying ACLs

In the File Manager GUI, select a file or directory to view its ACL. To display the file permissions, right click on the file and choose Properties from the popup menu. The Properties window appears. Click the Permissions tab to view the various permissions associated with the file, as shown in Figure 8-2:



File owner: student
File group: class

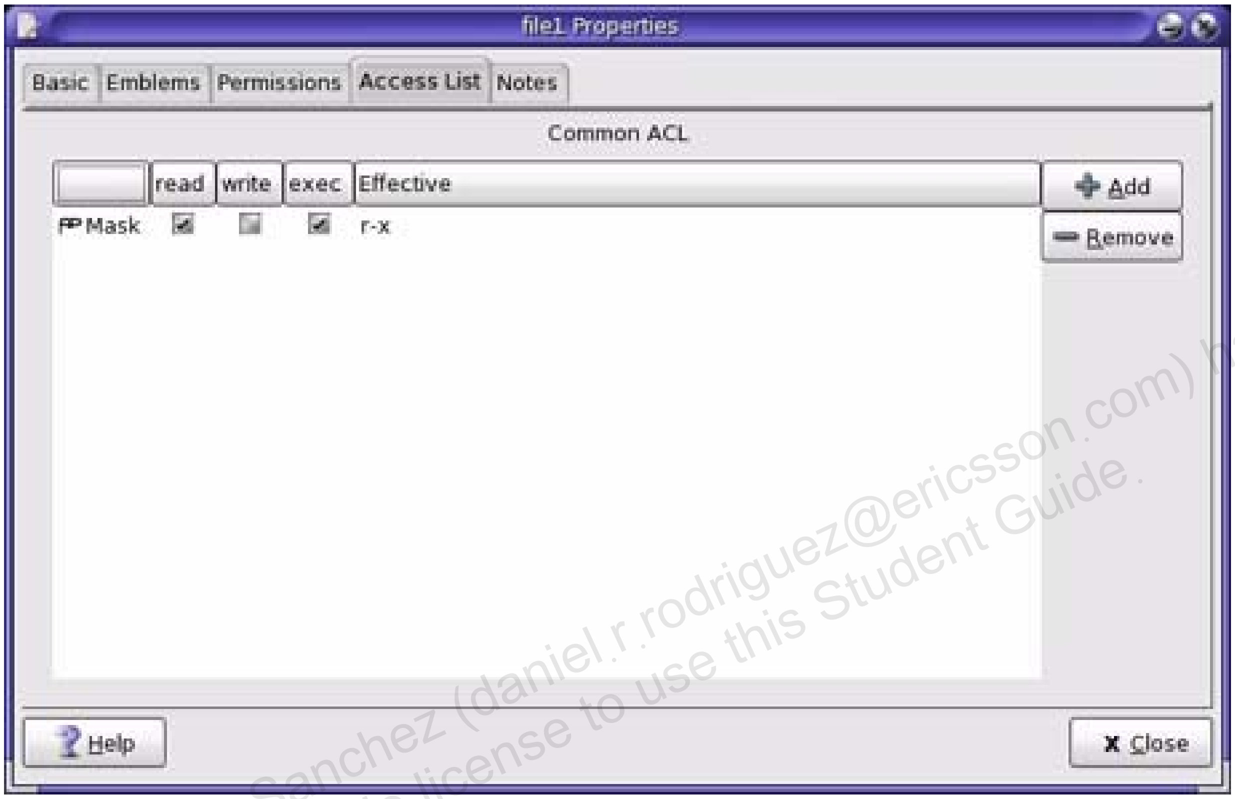
Owner: [✓ Read] [✓ Write] [✓ Execute]
Group: [✓ Read] Write [✓ Execute]
Others: [✓ Read] Write [✓ Execute]

Special flags: Set user ID
Set group ID
Sticky

Text view: -rwxr-xr-x
Number view: 755
Last changed: Monday, February 18 2008 at 8:40:14 PM

Figure 8-2 File Permissions

To display the ACL associated with the file, click Access List tab. The Access List tab displays a list of any existing ACL entries for the file, as shown in Figure 8-3:



	read	write	exec	Effective
Mask	✓		✓	r-x

Figure 8-3 File Access List

From the Access List tab, you can perform any of the following:

- Click Add to add ACL entry for a specific user or a group. Figure 8-4 shows the Add User/Group dialog box:

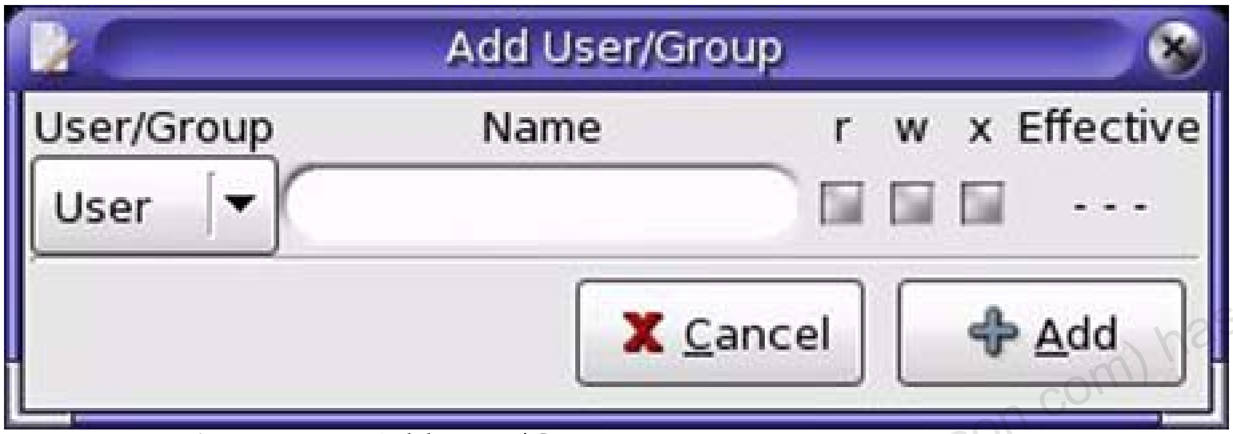


Figure 8-4 Add User/Group

- Change the ACL entries.
- Select an entry in the ACL list, and click Remove to remove an ACL entry.

Switching Users on a System

You can use the `su` command to switch to another user without logging out and back in to the system as that user.

The following command-line provides the format you will use to run the `su` command:

```
su - username
```

To switch to another user, and have that user's environment, you enter the `su` command with the dash (`-`) option and the login name of the user to whom you want to switch. You will be prompted to enter the password for that user. For example:

```
$ su - user1
Password: <enter user1's password>
```

To verify that you are now logged in as the actual user, `user1`, and are located in this user's home directory, enter the following commands:

```
$ whoami;pwd
user1
/export/home/user1
$
```

Note – The `whoami` command resides in the `/usr/ucb` directory.

The `su -` (dash) option specifies a complete login by reading all of the user's shell initialization files. The `-` (dash) option changes your work environment to what would be expected if you had logged in directly as that specified user. It also changes the user's home directory.

When you run the `su` command, the Effective User ID (EUID) and the Effective Group ID (EGID) are changed to the new user to whom you have switched. Access to files and directories are now determined by the value of the EUID and EGID for the effective user, rather than by the UID and GID numbers of the original user who logged into the system.

To return to your original user login and home directory, enter the `exit` command:

```
$ exit
```

Exercise: Using Access Control Lists

In this exercise, you create three files, and modify the ACLs associated with them.

Preparation

A user called `acluser` with a password of `acluser1` and a group called `aclgroup` is required for this exercise. This user and group were created for you.

RLDC

In addition to being able to use local classroom equipment, this lab was designed to also use equipment located in a remote lab data center. Directions for accessing and using this resource can be found at:

`http://remotelabs.sun.com/`

Ask your instructor for the particular SSH configuration file that you should use to access the appropriate remote equipment for this exercise.

Tasks

Complete the following steps:

1. Log in and open a terminal window.

   ___

2. Create the directory named `$HOME/acl_test`, and change directories to that location.

   ___

3. Use the `echo` command to create `file1`, and add the text string `Success for life` to the file.

   ___

4. Display the ACL for the file named `file1`.

   ____

   Do the permissions in the ACL match the permissions indicated by the `ls` command?

   ____

5. Change permissions on the file named `file1` so that only the owner and group have read access.

   ____

6. Change your user identity to `acluser`.

   ____

7. Display the contents of the file named `file1`.

   What is the output?

   ____

8. Exit your `su` session.

   ____

9. Use the `setfacl` command to add an ACL entry that allows read access for `acluser` to the ACL for the `file1` file.

   ____

10. Verify that the new ACL entry exists.

    ____

11. Change your user identity back to `acluser`.

    ____

12. Use the `ls` command to display the permissions applied to the file named `file1`.

    According to these permissions, does `acluser` have read access?

    ____

    What indicates that an additional ACL entry exists for the `file1` file?

    ____

13. Change to the `acl_test` directory and attempt to display the contents of the `file1` file.

    What is the result?

14. Exit your `su` session.

15. Create and display the ACL for the `file2` file.

    Do the group permissions match the permissions associated with the mask entry?

16. Change the permission mode to grant read, write, and execute permissions to the group that owns the `file2` file.

17. Display the ACL and a long listing for the `file2` file.

    Do the mask permissions match the group permissions?

18. Set the mask permissions for the file named `file2` to read-only.

19. Display the ACL and a long listing for `file2`.

    Do the mask permissions match the group permissions?

    In the long listing output, do you find an indication that `file2` has additional ACL entries?

20. Add an ACL entry for the group named `aclgroup` to the `file2` file. Grant only read and write permissions for this group.

21. Change your user identity to `acluser`, thereby inheriting the users environment.

22. Use the echo command to insert the text string `Success for life.` into the `file2` file.

    What is the output?

23. Exit your `su` session.
24. Display the ACL for the `file2` file.

25. Create a file called `file3` and add an ACL entry to the file that recalculates the mask allowing a user from the `aclgroup` group to insert a line of text to the file.

26. Change your user identity to `acluser`, inheriting the users environment.

27. Use the `echo` command to insert the text string `Success for life.` in the `file3` file.

28. Display the contents of the `file3` file.

29. Exit your `su` session.

Exercise Summary

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

- Experiences
- Interpretations
- Conclusions
- Applications

Exercise Solutions: Using Access Control Lists

Complete the following steps:

1. Log in and open a terminal window.
2. Create the directory named `$HOME/acl_test`, and change directories to that location.

```
$ mkdir $HOME/acl_test
$ cd $HOME/acl_test
```

3. Use the `echo` command to create the `file1` file, and add the text string `Success for life.` to the file.

```
$ echo “Success for Life” > file1
$ ls -l
-rw-r--r--  1  student    class      17 Feb 9 09:40  file1
```

4. Display the ACL for the `file1` file.

```
$ getfacl file1
```

Do the permissions in the ACL match the permissions indicated by the `ls` command?

Yes, they should.

5. Change permissions on the `file1` file so that only the owner and the group have read access.

```
$ chmod 440 file1
$ ls -l
-r--r-----   1 student  class         17 Sep 25 13:06 file1
$ getfacl file1
```

6. Change your user identity to `acluser`.

```
$ su - acluser
passwd:acluser1
```

7. Display the contents of the `file1` file.

What is the output?

```
$ cat ~student/acl_test/file1
```

The following error message is displayed:

```
cat: cannot open /export/home/student/acl_test/file1
```

8. Exit your `su` session.

```
$ exit
```

9. Use the `setfacl` command to add an ACL entry that allows read access for `acluser` to the ACL for the `file1` file.

```
$ setfacl -m user:acluser:4 file1
```

10. Verify that the new ACL entry exists.

```
$ getfacl file1
```

11. Change your user identity back to `acluser`.

```
$ su - acluser
passwd:acluser1
```

12. Use the `ls` command to display the permissions applied to the `file1` file.

```
$ ls -l ~student/acl_test/file1
```

According to these permissions, does `acluser` have read access?

No.

What indicates that an additional ACL entry exists for the `file1` file?

The + symbol at the end of the permissions string.

13. Change to the `acl_test` directory and attempt to display the contents of the `file1` file.

```
$ cd ~student/acl_test
$ cat file1
```

What is the result?

The file content appears.

14. Exit your `su` session.

```
$ exit
$
```

15. Create and display the ACL for the `file2` file.

```
$ touch file2
$ getfacl file2
```

Do the group permissions match the permissions associated with the mask entry?

Yes.

16. Change the permission mode to grant read, write, and execute permissions to the group that owns the `file2` file.

```
$ chmod g=rwx file2
```

17. Display the ACL and a long listing for the `file2` file.

```
$ getfacl file2
$ ls -l file2
```

Do the mask permissions match the group permissions?

Yes.

18. Set the mask permissions for the `file2` file to read-only.

```
$ setfacl -m mask:r-- file2
```

19. Display the ACL and a long listing for the `file2` file.

```
$ getfacl file2
$ ls -l file2
```

Do the mask permissions match the group permissions?

Yes.

In the long listing output, do you find an indication that the `file2` file has additional ACL entries?

No.

20. Add an ACL entry for the group named `aclgroup` to the `file2` file. Grant only read and write permissions for this group.

```
$ setfacl -m group:aclgroup:6 file2
```

21. Change your user identity to `acluser`, thereby inheriting the users environment.

```
$ su - acluser
password: acluser1
```

22. Use the `echo` command to insert the text string `Success for life.` into the `file2` file.

```
$ echo “Success for Life” > ~student/acl_test/file2
```

What is the output?

```
ksh: /export/home/student/acl_test/file2 cannot create
```

23. Exit your `su` session.

```
$ exit
```

24. Display the ACL for the `file2` file.

```
$ getfacl file2
```

25. Create a file called `file3` and add an ACL entry to the file that recalculates the mask allowing a user from the `aclgroup` group to insert a line of text to the file.

```
$ touch file3
$ setfacl -r -m g:aclgroup:6 file3
```

26. Change your user identity to `acluser`, inheriting the users environment.

```
$ su - acluser
password: acluser1
```

27. Use the `echo` command to insert into the text string `Success for life.` in the `file3` file.

```
$ echo “Success for Life” > ~student/acl_test/file3
```

28. Display the contents of the `file3` file.

```
$ cat ~student/acl_test/file3
```

29. Exit your `su` session.

```
$ exit
```

30. Display the ACL for the `file3` file.

```
$ getfacl file3
```

Notes:

Module 9

Searching Files and Directories

Objectives

The module describes how to search the contents of files using the `grep`, `egrep`, and `fgrep` commands. This module also describes how to locate a file or directory using the `find` command.

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

- Search for content in files
- Search for files and directories

Figure 9-1 is the course map that shows how this module fits into the current instructional goal.

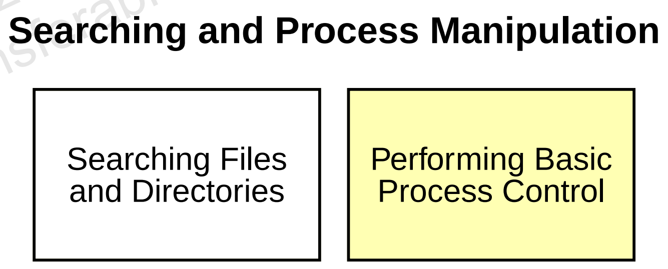

Searching and Process Manipulation
Searching Files and Directories
Performing Basic Process Control

Figure 9-1 Course Map

Searching for Content in Files

You can search the content of files for either patterns or strings of characters using the `grep`, `egrep`, and `fgrep` commands.

The `grep` and `egrep` commands enable you 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. A regular expression (RE) is either some plain text or special characters used for pattern-making.

The `fgrep` command enables you to search the contents of one or more files for a specific string. A string is a literal group of characters. The `fgrep` command does not use regular expressions.

Using the grep Command

The term `grep` means *to globally search for a regular expression and print all lines containing it*. When you use the `grep` command every line containing a specified character pattern prints to the screen. Using the `grep` command does not change file content.

The syntax for the `grep` command is:

```
grep options pattern filenames
```

The options that you use with the `grep` command can modify your search. Each option except the `-w` option can be used with the `egrep` and `fgrep` commands. Table 9-1 describes the options for the `grep` command.

Table 9-1 Options for the `grep` Command

Option	Definition
`-i`	Searches for both uppercase and lowercase characters
`-l`	Lists the names of files with matching lines
`-n`	Precedes each line with the relative line number in the file
`-v`	Inverts the search to display lines that do not match *`pattern`*
`-c`	Counts the lines that contain *`pattern`*
`-w`	Searches for the expression as a complete word, ignoring those matches that are substrings of larger words

To search for all lines that contain the pattern `root` in the `/etc/group` file and view their line numbers, enter the command:

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

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

To search for all lines that do not contain the pattern `root` in the `/etc/group` file, enter the command:

```
$ grep -v root /etc/group
student::10:
sysadmin::14:
smmsp::25:smmsp
gdm::50:
webservd::80:
nobody::60001:
noaccess::60002:
nogroup::65534:
$
```

To search for the names of the files that contain the pattern `root` in the `/etc` directory, enter the commands:

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

To count the number of lines containing the pattern `root` in the `/etc/group` file, enter the command:

```
$ grep -c root group
11
$
```

The `grep` command supports several regular expression metacharacters to further define a search pattern. Table 9-2 describes some of the regular expression metacharacters.

Table 9-2 Regular Expression Metacharacters

Metacharacter	Purpose	Example	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, and followed by an "`n`"
`*`	Matches the preceding item zero or more times	`'[a-z]*'`	Matches lowercase alphanumeric characters or nothing at all
`[ ]`	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 that do not contain "`a`" through "`m`" and followed by `attern`

To print all lines that begin with the letters no in the /etc/passwd file, enter the command:

```
$ grep '^no' /etc/passwd
nobody:x:60001:60001:NFS Anonymous Access User:/:
noaccess:x:60002:60002:No Access User:/:
nobody4:x:65534:65534:SunOS 4.x NFS Anonymous Access User:/:
$
```

To print all lines containing an "A," followed by three characters, followed by an "n" in the /etc/passwd file, enter the command:

```
$ grep 'A...n' /etc/passwd
adm:x:4:4:Admin:/var/adm
lp:x:71:8:Line Printer Admin:/usr/spool/lp
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
```

To print all lines that end with the word adm in the /etc/group file, enter the command:

```
$ grep 'adm$' /etc/group
sys::3:root,bin,adm
tty::7:root,adm
lp::8:root,adm
```

Using the egrep Command

The `egrep` command searches the contents of one or more files for a pattern using extended regular expression metacharacters. Extended regular expression metacharacters include the regular expression metacharacters that the `grep` command uses along with some additional metacharacters. Table 9-3 describes the additional metacharacters.

Table 9-3 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`, or `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`)

The syntax for the `egrep` command is:

```
egrep -options pattern filenames
```

To search for all lines containing one or more lowercase letters followed by the pattern '`body`' one or more times, enter the command:

```
$ egrep '[a-z]+body' /etc/passwd
nobody:x:60001:60001:NFS Anonymous Access User:/:
nobody4:x:65534:65534:SunOS 4.x NFS Anonymous Access User:/:
```

To search for lines containing the pattern `Network Admin` or `uucp Admin`, enter the command:

```
$ 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

You can use the `fgrep` command to search a file for a literal string or a group of characters. The `fgrep` command reads all regular expression characters literally. Regular expression metacharacters have no special meaning to the `fgrep` command. For example, the `fgrep` command interprets a `?` character as a question mark, and a `$` character as a dollar sign.

The syntax for the `fgrep` command is:

```
fgrep options string filenames
```

To search for all lines in the file containing a literal asterisk (*) character, enter the command:

```
$ fgrep '*' /etc/system
*ident  "@(#)system    1.18    97/06/27 SMI" /* SVR4 1.5 */
*
* SYSTEM SPECIFICATION FILE
*
* moddir:
*
*    Set the search path for modules.  This has a format similar to the
*    csh path variable. If the module isn't found in the first directory
*    it tries the second and so on. The default is /kernel /usr/kernel
... (output truncated)
```

Searching for Files and Directories

You can use the `find` command to locate files or directories in the directory hierarchy. The `find` command searches using criteria, such as file name, size, owner, modification time, and type.

Using the `find` Command

The `find` command recursively descends the directory tree in the pathname list, looking for the files that match the search criteria. As the `find` command locates the files that match those criteria, the path to each file is displayed on the screen.

The syntax for the `find` command is:

```
find pathnames expressions actions
```

Table 9-4 shows the *pathname*, *expression*, and *action* arguments for the `find` command.

Table 9-4 Arguments for the `find` Command

Argument	**Definition**
pathname	The absolute or relative path where the search originates.
expression	The search criteria specified by one or more options. Specifying multiple options causes the `find` command to use the boolean operator `and`, so all listed expressions must be verified as true.
action	The action required after the files have been located. The default action is to print all pathnames matching the criteria to the screen.

Table 9-5 describes some of the expressions that you can use with the `find` command.

Table 9-5 Expressions for the `find` Command

Expression	**Definition**
`-name` *`filename`*	Finds files matching the specified *`filename`*. Metacharacters are acceptable if placed inside `" "`.
`-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`* days, less than -*`n`* days, or exactly *`n`* days.
`-mtime [+|-]`*`n`*	Finds files that have been modified more than +*`n`* days ago, less than -*`n`* days ago, or exactly *`n`* days ago.
`-user` *`loginID`*	Finds all files that are owned by the *`loginID`* name.
`-type`	Finds a file type, for example, `f` (file) or `d` (directory).
`-perm`	Finds files that have certain access permission bits.

Table 9-6 describes the action arguments for the `find` command.

Table 9-6 Actions for the `find` Command

Action	**Definition**
`-exec` *`command`* `{} \;`	Runs the specified *`command`* on each file located. 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`* `{} \;`	Requires confirmation before the `find` command applies the *`command`* to each file located. This is the interactive form of the `-exec` command.
`-print`	Instructs the `find` command to print the current pathname to the terminal screen. This is the default.
`-ls`	Displays the current pathname and associated statistics, such as the inode number, the size in kilobytes, protection mode, the number of hard links, and the user.

To search for a file called `dante` starting in your home directory, enter the command:

```
$ find ~ -name dante
/export/home/student/dante
$
```

To search from your home directory, looking for files or directories called `removefile` and asking before deleting any matches to the pattern, enter the commands:

```
$ touch removefile
$ find ~ -name removefile -ok rm {} \;
< rm ... /export/home/student/removefile >? y
$ ls removefile
removefile: No such file or directory
$
```

To search from your home directory, looking for files or directories called `removefile` and deleting any matches to the pattern, enter the commands:

```
$ touch removefile
$ find ~ -name removefile -exec rm {} \;
$ ls removefile
removefile: No such file or directory
$
```

To look for all files that have not been modified in the last two days starting in the current directory, enter the command:

```
$ find . -mtime +2
<output will vary on each system>
```

To find files larger than 10 blocks (512-byte blocks) starting in your home directory, enter the command:

```
$ find ~ -size +10
/export/home/student/.sh_history
/export/home/student/dir1/coffees/beans
/export/home/student/tutor.vi
<example output from command, output will vary on each system>
```

Note – Your output will vary depending on the activity performed in your home directory.

Exercise: Locating Files and Text

In this exercise, you practice searching for files and directories using the `find` command. You also display and manipulate text in files.

Preparation

No special preparation is required for this exercise.

RLDC

In addition to being able to use local classroom equipment, this lab was designed to also use equipment located in a remote lab data center. Directions for accessing and using this resource can be found at:

```
http://remotelabs.sun.com/
```

Ask your instructor for the particular SSH configuration file that you should use to access the appropriate remote equipment for this exercise.

Tasks

To locate files and text, complete the following steps. Write the commands that you would use to perform each task in the space provided.

1. Distinguish between the `grep`, `egrep`, and `fgrep` commands.

   ___

   ___

   ___

2. Search for the text string `other` in the `/etc/group` file. Display it to the screen.

   ___

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

   ___

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

   ___

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

   ____

6. Which option is peculiar to the `grep` command but does not apply to the `egrep` and `fgrep` commands?

   ____

7. Display all the lines that have the pattern `load` in the `/etc/system` file.

   ____

8. Use the `grep` command to display how many lines contain at least one instance of the word `module` (uppercase and lowercase) in the `/etc/system` file.

   ____

9. Use the `grep` command to record how many instances of the word `Module` (uppercase `M` only) appear in the `/etc/system` file.

   ____

10. Starting in your home directory, find all files that were modified in the last one day.

    ____

11. Use the `find` command to search the `/etc` directory for all the files owned by the user `lp`.

    ____

12. Starting in your home directory, find all files of type `f` for file.

    ____

13. In your home directory, find all files of type `d` for directory.

    ____

14. From your home directory, use the `find` command with an option that prompts you before removing any files, to search for ordinary files of size 0 (zero) in the `/tmp` directory.

    ____

Note – Make sure that you answer No when prompted to remove any files.

Exercise Summary

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

- Experiences
- Interpretations
- Conclusions
- Applications

Exercise Solutions: Locating Files and Text

To locate files and text, complete the following steps:

1. Distinguish between the `grep`, `egrep`, and `fgrep` commands.

   The `grep` *command searches the contents of one or more files for a character pattern.*

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

   The `fgrep` *command searches a file for a literal string or a group of characters.*

2. Search for the text string `other` in the `/etc/group` file. Display it to the screen.

```
$ grep other /etc/group
```

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

```
$ cd
$ grep -v M file4
```

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

```
$ grep he dante file1 dante_1
```

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

```
$ egrep '(Sales|Finance)' file4
```

6. Which option is peculiar to the `grep` command but does not apply to the `egrep` and `fgrep` commands?

   The `-w` *option is peculiar to the* `grep` *command alone.*

7. Display all the lines that have the pattern `load` in the `/etc/system` file.

```
$ grep load /etc/system
```

8. Use the `grep` command to display how many lines contain at least one instance of the word `module` (uppercase and lowercase) in the `/etc/system` file.

```
$ grep -ic module /etc/system
```

9. Use the `grep` command to record how many lines contain at least one instance of the word `Module` (uppercase `M` only) in the `/etc/system` file.

```
$ grep -c Module /etc/system
```

10. Starting in your home directory, find all files that were modified in the last one day.

```
$ find /export/home/student -mtime -1
```

11. Use the `find` command to search the `/etc` directory for all the files owned by the user `lp`.

```
$ find /etc -user lp
```

12. Starting in your home directory, find all files of type `f` for file.

```
$ find ~ -type f
```

13. In your home directory, find all files of type `d` for directory.

```
$ find ~ -type d
```

14. From your home directory, using the `find` command search the `/tmp` directory for ordinary files of size 0 (zero) with an option that prompts you before removing any files.

```
$ find /tmp -type f -size 0 -ok rm {} \;
```

Note – Make sure that you answer No when prompted to remove any files.

Module 10

Performing Basic Process Control

Objectives

This module describes how the kernel uses identification numbers to track and run processes.

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

- Describe Solaris OS processes
- View a process
- Search for a specific process
- Send a signal to a process

Figure 10-1 is the course map that shows how this module fits into the current instructional goal.

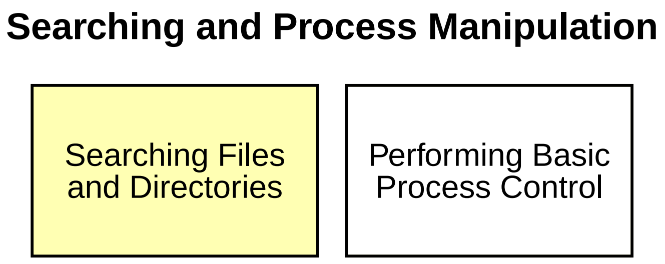

Searching and Process Manipulation
Searching Files and Directories
Performing Basic Process Control

Figure 10-1 Course Map

Solaris OS Processes

Every program you run in the Solaris OS creates a process. When you log in and start the shell, you start a process. When you run a command or when you open an application, you start a process.

The system starts processes called *daemons*. Daemons are processes that run in the background and provide services. For instance, the desktop login daemon (`dtlogin`) provides a graphical prompt that you use to log in to the operating system.

Using a PID

Every process has a unique process identification number (PID), which the kernel uses to track, control, and manage the process.

Using Process UID and GID Numbers

Each process is associated with a UID and a GID. These numbers indicate who owns a process and determine the functions of a process. Generally the UID and GID associated with a process are the same as the UID and GID of the user who started the process.

Understanding the Parent Process

When one process creates another, the first process is considered to be the parent of the new process. The new process is called the child process.

While the child process runs, the parent process waits. When the child finishes its task, it informs the parent process. The parent process then terminates the child process. If the parent process is an interactive shell, a prompt appears, indicating that it is ready for a new command.

Viewing a Process

You can use the process status (ps) command to list the processes that are associated with your shell. The ps command has several options you can use to determine which processes to display and how to format the output.

Using the ps Command

The syntax for the ps command is as follows:

```
ps options
```

For each process, the ps command displays the PID, the terminal identifier (TTY), the cumulative execution time (TIME), and the command name (CMD).

Identifying the ps Options

Table 10-1 describes some of the options you can use with the ps command.

Table 10-1 Options for the ps Command

Option	Description
-e	Prints information about every process on the system, including the PID, TTY, TIME, and CMD
-f	Generates a full (verbose) listing, which adds fields including the UID, parent process identification number (PPID), and process start time (STIME)

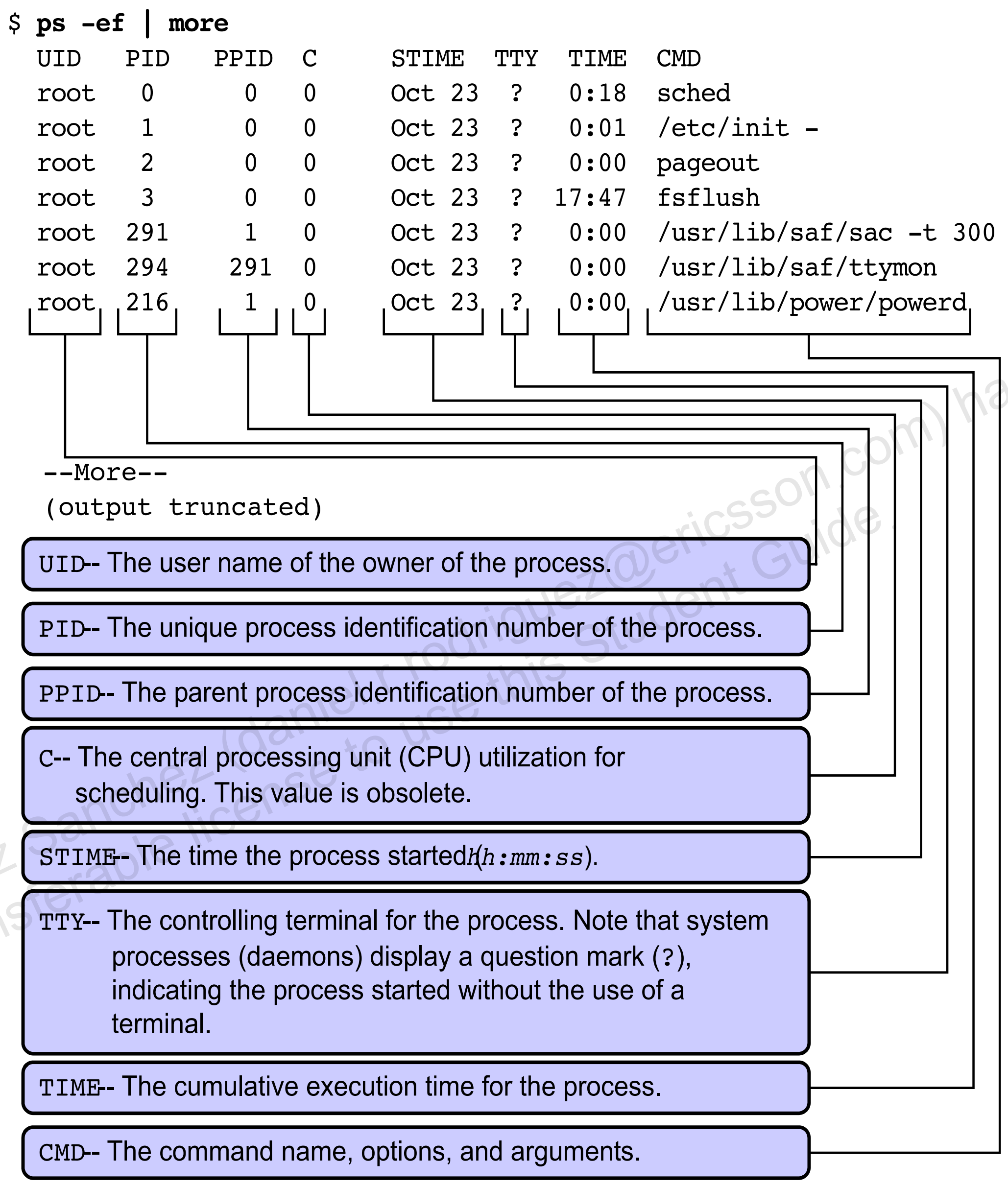
Note – Refer to the online man pages for a complete list of options for the ps command.

Displaying a Listing of All Processes

You can use the `ps -ef` command to view a listing of all the processes currently scheduled to run on the system.

```
$ ps -ef | more
    UID     PID PPID C  STIME TTY       TIME  CMD
    root     0     0  0   Feb 13 ?        0:18 sched
    root     1     0  0   Feb 13 ?        0:01 /etc/init -
    root     2     0  0   Feb 13 ?        0:00 pageout
    root     3     0  0   Feb 13 ?       17:47 fsflush
    root    7      1  0   Feb 13 ?        0:00 /lib/svc/bin/svc.startd
    root     9     1  0   Feb 13 ?        0:00 svc.configd
--More--
... (output truncated)
```

Figure 10-2 shows a description of the fields in the output of the `ps -ef` command.



```
$ ps -ef | more
  UID   PID   PPID  C     STIME  TTY  TIME   CMD
  root    0      0  0    Oct 23  ?    0:18  sched
  root    1      0  0    Oct 23  ?    0:01  /etc/init -
  root    2      0  0    Oct 23  ?    0:00  pageout
  root    3      0  0    Oct 23  ?   17:47  fsflush
  root  291      1  0    Oct 23  ?    0:00  /usr/lib/saf/sac -t 300
  root  294    291  0    Oct 23  ?    0:00  /usr/lib/saf/ttymon
  root  216      1  0    Oct 23  ?    0:00  /usr/lib/power/powerd
   --More--
   (output truncated)
```

`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 of the process.

`C`-- The central processing unit (CPU) utilization for scheduling. This value is obsolete.

`STIME`-- The time the process started (*hh:mm:ss*).

`TTY`-- The controlling terminal for the process. Note that system processes (daemons) display a question mark (`?`), indicating the process started without the use of a terminal.

`TIME`-- The cumulative execution time for the process.

`CMD`-- The command name, options, and arguments.

Figure 10-2 The `ps -ef` Command Output Description

Searching for a Specific Process

In the Solaris OS, you need to be able to search for processes that are running on the system. You can use the `ps` and `grep` commands together, or the `pgrep` command alone, to search for specific processes.

Using the ps and grep Commands

To search for a specific process, you can combine the `ps` and `grep` commands using the pipe (`|`) character. The pipe character causes the `ps` command to send the list of processes to the `grep` command. The `grep` command then searches the list of processes for the text that you specify.

The following example shows how to find active processes with names that contain the string `lp`:

```
$ ps -e | grep lp
217 ?        0:00 lpsched
$
```

Note – The `lpsched` daemon is responsible for controlling print services on the system.

Using the pgrep Command

You can use the `pgrep` command to search for specific processes by name. By default, the `pgrep` command displays the PID of every process that matches the criteria you specify on the command-line.

The syntax for the `pgrep` command is:

```
pgrep options pattern
```

Table 10-2 describes some of the options you can use with the `pgrep` command.

Table 10-2 Options for the `pgrep` Command

Option	Description
`-x`	Displays the PIDs that match the pattern exactly.
`-n`	Displays only the most recently created PID that contains the pattern.
`-U` *`uid`*	Displays only the PIDs that belong to the specified user. This option uses either a user name or a UID.
`-l`	Displays the name of the process along with the PID.
`-t` *`term`*	Displays only those processes that are associated with a terminal in the `term` list.

Note – Refer to the online man pages to view all the options for the `pgrep` command.

The following example shows how to use the `pgrep` command to display the PID of any process with a name that contains the string `lp`.

```
$ pgrep lp
217
$
```

The following example shows how to use the `pgrep` command with the `-l` option to display the PID and name of any process with a name that contains the string `vold`.

```
$ pgrep -l vold
    217   vold
```

The following example shows how to use the pgrep command with the -l option to display the PID and name of any process with a name that contains the string dt.

```
$ pgrep -l dt
29930 dtlogin
15454 dtfile
56091 dtfile
49987 dtcm
21855 dtterm
56525 sdt_shell
63484 dtgreet
14048 sdt_shell
3266 dtmail
56525 dtpad
30136 dtlogin
```

Note – The dtmail process only appears if you have the desktop environment mail process running; it does not run by default.

The following example shows how to use the pgrep command with the -lt option to display the PID and the name of any process associated with a terminal window.

```
$ pgrep -lt pts/2
12222 sdt_shell
12309 gnome-session
12343 gconfd-2
12351 xscreensaver
12224 bash
12349 gnome-keyring-d
12307 Xsession2.jds
```

The following example shows how to use the pgrep command with the -fl options to display the full process argument.

```
$ pgrep -l mountd
  155 automountd
$ pgrep -fl mountd
   2110 /usr/lib/autofs/automountd
```

Using the `ptree` Command

The `ptree` command displays a process tree based on the process ID that was passed as an argument to the command. The output has the specified PIDs or users, with child processes indented from their respective parent processes. An argument of all digits is taken to be a PID, otherwise it is assumed to be a user login name. The default behavior is all processes will be displayed.

```
$ ptree 56525
4650  /usr/dt/bin/dtlogin -daemon
  23101 /usr/dt/bin/dtlogin -daemon
    23566 /usr/dt/bin/dtlogin -daemon
      23567 /bin/ksh /usr/dt/bin/Xsession
        24058 /usr/dt/bin/sdt_shell -c unset DT; DISPLAY=:52;
          24064 -ksh -c unset DT; DISPLAY=:52; /usr/dt/bin/dt
            24209 /bin/ksh /usr/dt/config/Xsession2.jds
              24216 /usr/bin/gnome-session
```

Sending a Signal to a Process

A signal is a message that you can send to a process. Processes respond to signals by performing the action that the signal requests. Signals are identified by a signal number and by a signal name, and each signal has an associated action.

For example, signal 2 is known as the `SIGINT` signal. If you press Control-C, you send the `SIGINT` signal to the process running in your current terminal window. The process responds by exiting. Generally, you use either the `kill` command, or the `pkill` command to send signals to stop processes.

Table 10-3 describes some of the available signals.

Table 10-3 Signal Numbers and Names

Signal Number	Signal Name	Event	Definition	Default Response
1	`SIGHUP`	Hang up	A hang-up signal that drops a telephone line or terminal connection. This signal also causes some programs to re-initialize themselves without terminating.	Exit
2	`SIGINT`	Interrupt	An interrupt signal you generate from your keyboard, usually using Control-C.	Exit
9	`SIGKILL`	Kill	A signal that kills a process. A process cannot ignore this signal.	Exit
15	`SIGTERM`	Terminate	A signal that terminates a process in an orderly manner. Some processes ignore this signal. This is the default signal that `kill` and `pkill` send.	Exit

Note – Refer to the `-s3head signal` man page to view a complete list of signals and their default actions. You can run the `kill -l` (lowercase letter l) command in the bash shell to display the signal name associated with a particular signal number. Running this command in the `csh` or `ksh` shell will only display the signal names.

Terminating Processes With the `kill` Command

You can use the `kill` command to send a signal to one or more processes, The `kill` command terminates only those processes that you own. The `root` user can use the `kill` command on any process. The `kill` command sends signal 15, the terminate signal, by default. This signal causes the process to terminate in an orderly manner.

The syntax for the `kill` command is:

```
kill [-signal] PIDs
```

Before you can terminate a process using the `kill` command, you must know the PID to specify. Use either the `ps` command or the `pgrep` command to locate the PID for the process you want to terminate. You can terminate several processes at the same time by entering multiple PIDs on one command-line.

The following example shows how to use the `kill` command to terminate the `dtmail` process:

```
$ pgrep -l mail
   215  sendmail
 12047  dtmail
$ kill 12047
$ pgrep -l mail
   215  sendmail
```

Note – The `dtmail` process is only present when the desktop environment's mail tool is running. This process is not present by default.

Some processes ignore the default signal that the `kill` command sends. For example, a process waiting for a tape drive to complete an operation might ignore signal 15.

If a process does not respond to signal 15, you can force it to terminate using signal 9 with the `kill` command. To terminate a process using signal 9, enter the command:

```
$ kill -9 PID
```

Caution – Use the `kill -9` command only when necessary. When you use the `kill -9` command on an active process, the process terminates instantly instead of performing an orderly shutdown. Using signal 9 on processes that control databases or programs that update files could cause data corruption.

Terminating Processes With the pkill Command

You use the `pkill` command to send signals to processes. The `pkill` command sends signal 15, the terminate signal, by default. The `pkill` command allows you to use process names to identify the process you want to terminate.

The syntax for the `pkill` command is:

```
pkill [-options] pattern
```

The options for the `pkill` command are similar to those for the `pgrep` command.

The following example shows how to terminate the mail session process using the `dtmail` command name:

```
$ pgrep -l mail
 215 sendmail
 470 dtmail
$ pkill dtmail
$ pgrep -l mail
 215 sendmail
$
```

You can force processes that do not respond to signal 15 to terminate using signal 9 with the `pkill` command.

The following example shows how to terminate a process using signal 9.

```
$ pkill -9 -x process_name
```

Exercise: Manipulating System Processes

In this exercise, you use the commands described in this module to determine PIDs, view a process tree, and kill processes.

This exercise introduces the `tty` command, which displays the name of the current terminal window. The name displayed by the `tty` command includes a unique identification number assigned by the Solaris OS to each open terminal window; for example: `/dev/pts/2`. In the tasks illustrating the `tty` command, the unique identification number is displayed as `/dev/pts/`*`n`*, where *`n`* is a numeral.

Preparation

No special preparation is required for this exercise.

RLDC

In addition to being able to use local classroom equipment, this lab was designed to also use equipment located in a remote lab data center. Directions for accessing and using this resource can be found at:

```
http://remotelabs.sun.com/
```

Ask your instructor for the particular SSH configuration file that you should use to access the appropriate remote equipment for this exercise.

Tasks

Perform the following:

1. 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
```

____

2. Perform the `ps -ef` command in a terminal window. Identify the process ID related to the `ps -ef` command.

```
$ ps -ef
```

PID: ______________________________

3. Open another terminal window, referred to as terminal window 1, and enter the following command:

```
$ gcalctool
```

Note – This command launches the calculator, which is terminated using the `kill` command from another terminal window in a subsequent task.

4. Open another terminal window, called terminal window 2. Use the `ps` or `pgrep` command to identify the PID of the `gcalctool` command.

   PID: ______________________________

5. From terminal window 2, terminate the `gcalctool` command using the PID.

____

6. From terminal window 2, use the `tty` command to identify the name of this terminal window. The name appears as `/dev/pts/n`, where *n* is a numeral; for example, `/dev/pts/4`.

```
$ tty
```

`/dev/pts/`____

Move back to terminal window 1. Use the `pgrep` command to find the PID associated with the name of terminal window 2.

```
$ pgrep -t pts/n
```

PID: ___________________________________

7. In terminal window 1, use the `kill` command to terminate the terminal window 2.

```
$ kill PID
```

Did it work? ____________________________________

8. Use the `kill` command with the `-9` option to terminate terminal window 2.

```
$ kill -9 PID
```

Did it work? ____________________________________

9. Name the commands used to search for a specific process.

____

____

10. Perform the following `kill` commands to identify the signal names associated with signal numbers.

`$ kill -l 9` (`-l` is the letter l)

Signal: ____

`$ kill -l 15` (`-l` is the letter l)

`Signal:` ____

11. Distinguish between a process and a job.

____

____

12. Distinguish between the `pkill` command and the `kill` command.

____

____

13. In a terminal window type the command **`sleep 500 &`**.
14. In the same terminal window use the `ps` command to identify the shell process running in that window.

    ___

15. In another terminal window use the `ptree` command using the shell PID from Step 14 as the argument.

    ___

16. In this terminal window use the kill command with the PID as the argument that was used in Step 15. Does the window close down?

    ___

Exercise Summary

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

- Experiences
- Interpretations
- Conclusions
- Applications

Exercise Solutions: Manipulating System Processes

Perform the following steps:

1. Use the following `ps` commands to list the processes currently running on your system. What information does each command provide?

```
$ ps
```

This command prints information for the current user and terminal.

```
$ ps -f
```

This command prints a full listing of the `ps` *command.*

```
$ ps -e
```

This command prints information about every process running.

```
$ ps -ef
```

This command prints a full listing of the `ps -e` *command.*

2. Perform the `ps -ef` command in a terminal window. Identify the process ID related to the `ps -ef` command.

   The PID differs from system to system.

3. Open another terminal window, referred to as terminal window 1, and enter the following command:

```
$ gcalctool
```

Note – This command launches the calculator, which is terminated using the `kill` command from another terminal window in a subsequent task.

4. Open another terminal window, called terminal window 2. Use the `ps` command to identify the PID of the `gcalctool` command.

```
$ ps -ef | grep gcalctool
```

or

```
$ pgrep gcalctool
```

5. From terminal window 2, terminate the `gcalctool` command using the PID.

```
$ kill PID
```

The value `PID` *is the PID of the* `gcalctool` *command.*

6. From terminal window 2, enter the `tty` command to identify the name of this terminal window. The name appears as `/dev/pts/n`, where *n* is a numeral; for example, `/dev/pts/4`.

```
$ tty
```

This name differs from system to system.

7. Move back to terminal window 1. Use the `pgrep` command to find the PID associated with the name of terminal window 2.

```
$ pgrep -t pts/n
```

The PID differs from system to system.

8. In the current window, use the `kill` command to terminate terminal window 2.

```
$ kill PID
```

Did it work?

No. Use the `kill` *command with the* `-9` *option to terminate terminal window 2.*

```
$ kill -9 PID
```

Did it work?

Yes.

9. Name the commands used to search for a specific process.

   The `pgrep` *command and the* `ps` *command with the* `grep` *command.*

10. Run the following `kill` commands to identify the signal names associated with signal numbers.

```
$ kill -l 9
```

The signal name is `KILL`.

```
$ kill -l 15
```

The signal name is `TERM`.

11. Distinguish between a process and a job.

    Every program that runs on the Solaris 10 OS creates a process. A process that a shell can manage is called a job.

12. Distinguish between the `pkill` command and the `kill` command.

    You can terminate a job using the `kill` *command. You can terminate a specific process using the* `pkill` *command.*

13. In a terminal window type the command **sleep 500 &**.

```
$ sleep 500 &
[1] 15866
$
```

14. In the same terminal window use the `ps` command to identify the shell process running in that window.

```
$ ps
 PID    TTY       TIME CMD
3989    pts/3     0:00 ksh
4029    pts/3     0:00 ps
4028    pts/3     0:00 sleep
```

15. In another terminal window use the `ptree` command using the shell PID from Step 14 as the argument.

```
$ ptree 3989
3985 gnome-terminal
   3989 ksh
      4028 sleep 500
```

16. In this terminal window use the `kill` command with the PID as the argument that was used in Step 15. Does the window close down?

```
$ kill -9 3989
```

Advanced Shell Functionality

Objectives

This module describes managing a job, aliases, and functions. It also covers configuring the shell environment.

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

- Manage jobs in the Korn shell
- Describe the Korn shell alias utility
- Use Korn shell functions
- Set Korn shell options

Figure 11-1 is the course map that shows how this module fits into the current instructional goal.

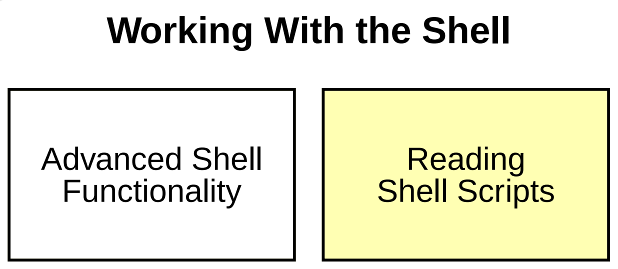


Figure 11-1 Course Map

Managing Jobs in the Korn Shell

This section describes how to manage jobs in the Korn shell. The Korn shell enables you to run jobs in the background, which frees your terminal to perform other commands. You can also run jobs in the foreground. You can use Korn shell commands to list current jobs as well as to stop a job.

Introducing Jobs

A job is a process that the shell can manage. Shells start and control jobs. Because jobs are processes, each job has an associated PID. The shell also assigns each job a sequential job ID number.

The shell enables you to run multiple jobs at the same time. Job control commands enable you to manage multiple jobs within a shell. There are three types of jobs that shells manage: foreground jobs, background jobs, and stopped jobs.

When you enter a command in a terminal window, the command occupies that terminal window until it completes. This type of job is called a foreground job.

When you enter an ampersand (&) symbol at the end of a command-line, the command runs without occupying the terminal window. The shell prompt is displayed immediately after you press Return. This type of job is called a background job.

If you press Control-Z for a foreground job, or enter the `stop` command for a background job, the job stops. This job is called a stopped job.

Job control commands enable you to place jobs in the foreground or background, and to start or stop jobs. Table 11-1 describes the commands you can use for job control.

Table 11-1 Job Control Commands

Command	**Value**
`jobs`	Lists all jobs that are currently running or are stopped in the background
`bg %`*`n`*	Runs the current or specified job in the background (*n* is the job ID)
`fg %`*`n`*	Brings the current or specified job into the foreground (*n* is the job ID)
Control-Z	Stops the foreground job and places it in the background as a stopped job
`stop %`*`n`*	Stops a job that is running in the background (*n* is the job ID)

Note – You can control a job using these commands only in the shell in which the job started.

Running a Job in the Background

To run a job in the background, enter the command you want to run along with an ampersand (&) symbol at the end of the command-line. For example, you can run the `sleep` command in the background as follows:

```
$ sleep 500 &
[1]     3028
$
```

Note – The `sleep` command suspends execution of a program for *n* seconds.

The shell returns the job ID number it assigned to the command, contained in brackets, and the PID associated with the command. You use the job ID number to manage the job with job control commands. The kernel can use the PID number to manage the job.

When a background job has finished and you press Return, the shell displays a message indicating that the job is done.

```
[1] +  Done                    sleep 500 &
$
```

Listing Current Jobs

You can use the `jobs` command to display the list of jobs that are running or stopped in the background. For example:

```
$ jobs
[1] +  Running                 sleep 500 &
$
```

Bringing a Background Job Into the Foreground

You can use the `fg` command to bring a background job to the foreground. For example:

```
$ fg %1
sleep 500
```

Note – The foreground job occupies the shell until the job is completed, stopped, or stopped and placed into the background.

Sending a Foreground Job to the Background

You can use the Control-Z keys and `bg` command to return a job to the background. The Control-Z keys suspend the job, and place it in the background as a stopped job. The `bg` command runs the job in the background. For example:

```
$ sleep 500
^Z[1] + Stopped (SIGTSTP)          sleep 500
$ jobs
[1] + Stopped (SIGTSTP)          sleep 500
$ bg %1
[1]    sleep 500&
$ jobs
[1] +  Running                   sleep 500
$
```

Note – When you place a stopped job in either the foreground or the background, the job restarts.

Korn Shell Alias Utility

An *alias* is a shorthand notation in the Korn shell to enable you to customize and abbreviate Solaris commands. An alias is defined using the `alias` command.

Command Format

The `alias` command syntax is as follows:

```
alias name=command_string
```

For example:

```
$ alias dir='ls -lF'
```

The shell maintains 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 can be 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 must be separated with a semicolon.

Predefined Korn Shell Aliases

The Korn shell contains several predefined aliases, which you can display using the `alias` command. 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 $$'
```

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 `rm`, `cp`, and `mv` commands 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 prompts you before overwriting existing files.

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, thereby causing it to run the original `rm` command to remove the file `file1`.

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 a single alias name. Individual commands are separated by semicolons. For example:

```
$ alias info='uname -a; id; date'
$ info
SunOS host1 5.10 Generic_120011-14 sun4u sparc SUNW,Sun-Blade-1500
uid=1002(user2) gid=1000(class)
Fri Feb 13 15:22:47 MST 2009
$
```

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, a directory listing appears. If the listing is longer than the available space on the screen, the `--More--` line item appears at the end of the list, indicating that additional directory contents are displayed on the subsequent screen.

For example:

```
$ alias ll='ls -l | more'
$ cd /usr
$ ll
total 136
drwxrwxr-x   2 root     bin         1024 Feb 13 18:33 4lib
drwx------   8 root     bin          512 Feb 13 18:14 aset
drwxrwxr-x   2 root     bin         7168 Feb 13 18:23 bin
drwxr-xr-x   4 bin      bin          512 Feb 13 18:13 ccs
drwxrwxr-x   5 root     bin          512 Feb 13 18:28 demo
--More--
```

Removing Aliases

Use the `unalias` command to remove aliases from the alias list. The `unalias` command syntax is as follows:

```
$ unalias alias_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`).

Using Korn Shell Functions

Functions are a powerful feature of shell programming used to construct customized commands. A function is a group of commands organized as separate routines. Using a function involves two steps:

1. Define the function
2. Invoke 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.

Function Examples

The following example creates a function called `num` to run the `who` command, and directs the output to the `wc` command to display the total number of users currently logged on the system:

```
$ function num { who | wc -l; }
$ num
```

Note – Functions are only used in shell scripts; however, for purposes of demonstrations, shell functions are run on the command-line simply to illustrate how the functions will perform.

The following example creates a function called `list` to run 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 Shell Environment

The shell secondary prompt string is stored in the PS2 shell variable, and you can customize it as needed.

```
$ PS2="somethings missing >"
$ echo $PS2
somethings missing >
$
```

In this example, the secondary prompt displays the message that the command-line is incomplete.

Note – To have this secondary shell prompt appear in every shell, it must be included in the user's Korn shell initialization file (usually named `.kshrc`).

Setting Korn Shell Options

Options are switches that control the behavior of the Korn shell. Options are boolean, meaning that they can be either on or off.

To turn an option on, type:

```
$ set -o option_name
```

To turn an option off, type:

```
$ set +o option_name
```

To show current option settings, 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 using 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 >| deactivation 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, enter the following commands:

```
$ set +o noclobber
$ set -o | grep noclobber
noclobber        off
$ ls -l > file_new
$
```

Exercise: Managing Jobs in the Korn Shell

In this exercise, you use the commands described in this module to control jobs.

Preparation

No special preparation is required for this exercise.

RLDC

In addition to being able to use local classroom equipment, this lab was designed to also use equipment located in a remote lab data center. Directions for accessing and using this resource can be found at:

`http://remotelabs.sun.com/`

Ask your instructor for the particular SSH configuration file that you should use to access the appropriate remote equipment for this exercise.

Tasks

Complete the following steps. Write the commands that you would use to perform each task in the space provided.

1. Perform the following command in the background:

```
$ sleep 500 &
```

____

2. 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. What are job control commands? Name any two commands that are used for job control and mention the functions of each of them.

   ____

   ____

5. Terminate the job running the `sleep` command.

____

6. Enable the `noclobber` option, and verify that you enabled it using the `set` command.

____

7. Display all predefined aliases.

____

8. Create an alias named `cls` that clears the terminal screen.

____

9. Create an alias named `dir` that displays a long listing of all the files and directories in the current directory.

____

10. Create an alias named `h` that lists your command history.

____

11. Unalias the `history` command and the `clear` command.

____

12. Display all defined functions.

____

13. Create a function called `data` that:
    - Clears the terminal screen
    - Displays the date and time
    - Displays who is logged in to your system
    - Displays the path of the current working directory
    - Lists the current working directory in long format

____

14. Use `vi` to edit the `.kshrc` file in your home directory.

```
$ vi ~/.kshrc
```

Add the following line entries:

```
set -o vi
alias h='history'
alias cls='clear'
alias lf='pwd ; ls -lF'
```

15. Enter the following commands:

```
$ . ~/.profile
$ . ~/.kshrc
```

16. 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

Exercise Solutions: Managing Jobs in the Korn Shell

Complete the following steps.

1. Perform the following command in the background:

```
$ sleep 500 &
```

2. Find the job number of the `sleep` command started in Step 1.

```
$ jobs
```

The job number differs from system to system.

3. Bring the job to the foreground, and then put it back in the background.

```
$ fg %1
sleep 500
^Z[1] + Stopped (SIGTSTP)        sleep 500 &
$ jobs
[1] + Stopped (SIGTSTP)        sleep 500 &
$ bg %1
[1]     sleep 500 &
```

4. What are job control commands? Name any two commands that are used for job control and mention the functions of each of them.

Job control commands enable you to place jobs in the foreground or background, and to start or stop jobs.

Two commands used for job control are the `jobs` *command and the* `bg %n` *command. The* `jobs` *command lists all jobs that are currently running or are stopped in the background and the* `bg %n` *command runs the current or specified job in the background (*`n` *is the job ID).*

5. Terminate the job running the `sleep` command.

```
$ kill %1
[1] + Terminated               sleep 500 &
```

6. Enable the `noclobber` option, and verify that you enabled it using the `set` command.

```
$ set -o noclobber
$ set -o | more
```

7. Display all predefined aliases.

```
$ alias
```

8. Create an alias named `cls` that clears the terminal screen.

```
$ alias cls=clear
```

9. Create an alias named `dir` that displays a long listing of all the files and directories in the current directory.

```
$ alias dir='ls -l'
```

10. Create an alias named `h` that lists your command history.

```
$ alias h=history
```

11. Unalias the `history` command and the `clear` command.

```
$ unalias h
$ unalias cls
```

12. Display all defined functions.

```
$ typeset -f
```

13. Create and test a function called `data` that:

- Clears the terminal screen
- Displays the date and time
- Displays who is logged in to your system
- Displays the path of the current working directory
- Lists the current working directory in long format.

```
$ function data { clear; date; who; pwd; ls -l; }
```

14. Use `vi` to edit the `.kshrc` file in your home directory:

```
$ vi ~/.kshrc
```

Add the following line entries:

```
set -o vi
alias h='history'
alias cls='clear'
alias lf='pwd ; ls -lF'
```

15. Perform the following commands:

```
$ .  ~/.profile
$ .  ~/.kshrc
```

16. Test your new aliases and functions.

Notes:

Module 12

Reading Shell Scripts

Objectives

Upon completion of this module you should be able to:

- Describe shell scripts
- Run shell scripts
- Pass values to shell scripts
- Use the `test` command
- Perform conditional commands

Figure 12-1 is the course map that shows how this module fits into the current instructional goal.

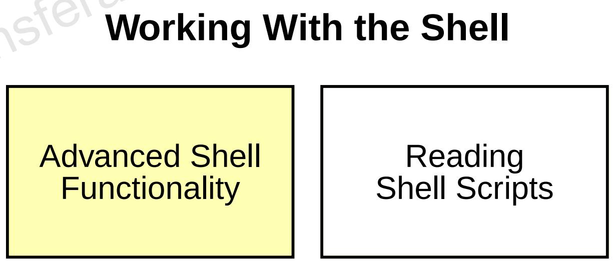


Figure 12-1 Course Map

Shell Scripts

Shell scripts are text files that automate a series of commands that otherwise must be run one at a time. Shell scripts are often used to automate command sequences that repeat, such as services that start or stop on system start up or shut down.

Any command that can be run from the command-line, such as `ls`, can be included in a shell script. Similarly, any command that can be included in a shell script can be run on the command-line.

Users with little or no programming experience can create and run shell scripts. You initiate the sequence of commands in the shell script by simply entering the name of the shell script on the command-line.

Determining the Type of Shell to Run a Shell Script

There are several different shells available in the Solaris OS. Two of the most commonly used shells are the Bourne shell and the Korn shell.

To ensure that the correct shell is used to run a shell script, the first line of the script should always begin with the characters `#!` followed immediately by the absolute pathname of the shell required to run the script. These characters must be on the first line of the file.

```
#!/full-pathname-of-shell
```

For example:

```
#!/bin/sh
```

or

```
#!/bin/ksh
```

Comments

Comments are text entries in shell scripts that provide information about the script files. If you were to view the contents of script files, it would be helpful if the authors of those shell scripts had included some explanations in comments. The comments should explain the purpose of the script and should explain any specific lines that might be especially confusing.

Note – It is good practice to put comments into programs and shell scripts.

Comments are always preceded by a hash (#) character. Whenever the shell encounters a # character in a script file the line of text following it is ignored by the shell.

The addition of comments in a shell script file does not affect the execution of the script unless a syntactical error is introduced when the comments are added. Remember that the comments are there for documentation purposes, so someone reading the script will have an idea of what will occur when the script is executed.

For example:

```
# This is a comment inside a shell script
ls -l  # lists the files in a directory
```

Running Shell Scripts

The shell interprets shell scripts line by line. Shell scripts are not compiled into binary form. Because shell scripts have to be read line by line when they are run, the user must have read permissions to be able to run a shell script. For example, you will grant the read and execute permissions on the `mycmd` script file, so that you can execute this shell script as a command:

```
$ chmod u+x mycmd
```

Note – When a shell script is running, any applied changes occur in the sub-shell or child process. A sub-shell cannot change the values of a variable in the parent shell, or its working directory.

```
$ cat myvars
echo running myvars
FMHOME=/usr/frame
MYBIN=/export/home/student/bin
$ ls -l myvars
-rw-r--r--   1 student   class 65 Feb 15 16:14 myvars
$ chmod u+x myvars
$ ls -l myvars
-rwxr--r--   1 student   class 65 Feb 15 16:14 myvars
$ ./myvars
running myvars
```

After running the script, the `FMHOME` and `MYBIN` variables are not available because the script is run in a sub-shell.

```
$ echo $FMHOME
```

One of the most frequently used shell scripts is a users initialization file (`~/.profile`). This script is specifically designed to set the working environment of the user in the current shell.

If you made changes to your `.profile` file, you need to implement these changes in the current shell without logging out and logging back in.

The dot (`.`) command runs the commands in the specified script in the current shell, as if the commands were entered on the command-line.

```
$ . ./myvars
$ echo $FMHOME
/usr/frame
```

Passing Values to a Shell Script

Shell scripts become more useful when you pass values to them when they are run. When you run a shell script and pass values to it on the command-line, the shell stores the first word after the script name in the variable $1, the second in the variable $2, and so on. These special variables are called *positional parameters*, and they are very useful to verify that the user passed the correct number of values when the script was run.

For example:

```
$ cat greetings
#!/bin/sh
echo $1 $2  #echo the first two parameters passed
```

Add execute permissions to `greetings`.

```
$ chmod u+x greetings
```

Run `greetings` while passing the `hello` and `world` values.

```
$ greetings hello world
hello world
```

The `shift` Command

In the Bourne and Korn shells you can pass as many values as necessary on the command-line. However, the Bourne shell accepts only a single number after the `$` sign. In the Bourne shell, an attempt to access the value in the tenth argument using the notation `$10` results in the value of `$1` followed by a zero (`0`).

The `shift` command enables you to shift your positional parameter values back by one position. For example, the value of the `$2` parameter becomes assigned to the `$1` parameter. Similarly, the value of the `$3` parameter becomes assigned to the `$2` parameter, and so on.

Note – In the Korn shell, you can access the tenth parameter directly with the value of the tenth argument `${10}` to the script. For the Bourne shell, however, you have to use the `shift` command.

Checking the Exit Status

All commands in the Solaris environment return an exit status. This numeric value is used to indicate the success or failure of a command. A value of zero indicates success. A non-zero value indicates failure. This non-zero value can be any integer in the range of 1–255.

The developer can use the exit status values to indicate different error situations. The exit status of the last command run in the foreground is held in the `?` special shell variable, and can be tested using the `echo` command.

For example:

```
$ grep other /etc/group
other::1:
$
$ echo $?
0
$
$ grep others /etc/group
$ echo $?
1
$
```

Using the `test` Command

The `test` command, within a shell script, is used for testing conditions.

The `test` command can be used to verify many conditions, including:

- Variable contents
- File access permissions
- File types

The `test` command can be written as *`test expression`* or using the `[` *`expression`* `]` special notation.

The `test` command does not return any output. If the condition being tested is true, the exit status of the `test` command is set to `0`. If the condition being tested is false, the exit status is set to `1`.

For the purpose of demonstration only, the following examples of the `test` command are run on the command-line simply to illustrate how the `test` command will perform.

- Test if the value of the `LOGNAME` variable is `student`.

```
$ echo $LOGNAME
student

$ test "$LOGNAME" = "student"
$ echo $?
0
```

- Test if the value of the `LOGNAME` variable is `student` using the `[` *`expression`* `]` notation.

```
$ echo $LOGNAME
student

$ [ "$LOGNAME" = "student" ]
$ echo $?
0
```

- Test if the user has read permissions on the `/etc/group` file.

```
$ ls -l /etc/group
-rw-r--r--   1 root sys  290 Feb 13 15:14 /etc/group

$ test -r /etc/group
$ echo $?
0
```

- Test if the user has read permissions on the `/etc/group` file using the `[ expression ]` notation.

```
$ ls -l /etc/group
-rw-r--r--   1 root sys  290 Feb 13 15:14 /etc/group

$ [ -r /etc/group ]
$ echo $?
0
```

- Determine if `/etc` is a directory.

```
$ ls -ld /etc
drwxr-xr-x  53 root     sys         3584 Feb 18 11:48 /etc

$ test -d /etc
$ echo $?
0
```

- Determine if `/etc` is a directory using the `[ expression ]` notation.

```
$ [ -d /etc ]
$ echo $?
0
```

- Compare the result against a known file.

```
$ test -d /etc/group
$ echo $?
1
```

- Compare against a known file using the `[ expression ]` notation.

```
$ [ -d /etc/group ]
$ echo $?
1
```

Executing Conditional Commands

The shell provides two special constructs that enable you to run a command based on whether a preceding command succeeds or fails.

The && construct ensures that a command is run only if the preceding command succeeds.

For example:

```
$ mkdir $HOME/newdir && cd $HOME/newdir
```

The `||` construct ensures that a command is run only if the preceding command fails.

For example:

```
$ mkdir /usr/tmp/newdir || mkdir $HOME/newdir
```

Using the `if` Command

The `if` command evaluates the exit status of a command and initiates additional actions based on the returned value. The `if` command syntax is as follows:

```
$ if command1
> then
> execute command2
> else
> execute command3
> fi
```

If the exit status is zero (0), any commands that follow the `then` statement are run. If the exit status is non-zero, any commands that follow the `else` statement are run.

The `if` command is always closed with the `fi` statement. The `if` command is often used in conjunction with the `test` command.

Examples of the `if` command includes the display of a greetings message:

```
$ id
uid=101(frame) gid=1(other)
$

$ if test "$LOGNAME" = root
> then echo Hello System Administrator
> else
> echo Hello "$LOGNAME"
> fi
Hello frame

$ if [ "$LOGNAME" = "root" ]
> then echo hello System Administrator
> else
> echo hello "$LOGNAME"
> fi
hello frame
```

Confirm that the user has read permissions for the `/etc/group` file.

```
$ if test -r /etc/group
> then
> echo "You have read permission on /etc/group"
> else
> echo "Sorry unable to read /etc/group file"
> fi
You have read permission on the /etc/group file

$ if [ -r /etc/group ]
> then
> echo "You have read permission on /etc/group"
> else
> echo "Sorry unable to read /etc/group file"
> fi
You have read permission on the /etc/group file
```

Determine if a file is a directory.

```
$ ls -ld /etc
drwxr-xr-x  53 root     sys         3584 Feb 18 11:48 /etc

$ if test -d /etc
> then
> echo /etc is a directory
> else
> echo /etc is not a directory
> fi
/etc is a directory

$ if [ -d /etc ]
> then
> echo /etc is a directory
> else
> echo /etc is not a directory
> fi
/etc is a directory

$ if test -d /etc/group
> then
> echo /etc is a directory
> else
> echo /etc is not a directory
> fi
/etc is not a directory
```

Using the `while` Command

The `while` command enables you to repeat a command or group of commands. The `while` command syntax is as follows:

```
$ while command1
> do
> command2
> done
```

In this example, the `while` command evaluates the exit status of the `command1` command that follows it.

If the value is zero, any commands that follow the `do` statement are run, `command1` is run again, and the exit status checked again.

If the exit status of command1 is non-zero, the loop terminates.

For example, use the set command to assign values to the positional parameters as follows:

```
$ set this is a while loop
$ echo $*
this is a while loop
$ while [ $# -gt 0 ]
> do
> echo  $1
> shift
> done
this
is
a
while
```

Note – The special shell variable * is the value of all command-line arguments. The special shell variable # is the number of arguments passed to the script.

Using the case Command

The case command compares a single value against other values, and runs a command or group of commands when a match is found.

The case command syntax is as follows:

```
$ case value in
> pat1)command
> command
> ...
> command
> ;;
> pat2)command
> command
> ...
> command
> ;;
> ...
> patn)command
> command
> ...
> command
> ;;
> esac
```

When a match is found and the respective commands are run, no other patterns are checked. For example:

```
#!/sbin/sh
#
# Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# ident"@(#)volmgt1.703/12/09 SMI"

$ case "$1" in
> 'start')
>   if [ -f /etc/vold.conf -a -f /usr/sbin/vold -a \
>   "${_INIT_ZONENAME:='/sbin/zonename'}" = "global" ]; then
>      echo 'volume management starting.'
>      /usr/sbin/vold >/dev/msglog 2>&1 &
>   fi
>   ;;
>
> 'stop')
>   /usr/bin/pkill -x -u 0 vold
>   ;;
>
>   *)
>   echo "Usage: $0 { start | stop }"
>   exit 1
>   ;;
>
> esac
```

Note – The positional parameter `$0` is the script name.

Notes:

Creating Archives

Objectives

This module introduces archives and describes how to archive and retrieve files. The module also describes the `tar` and `jar` commands.

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

- Archive files
- Compress and archive files with the `jar` command

Figure 13-1 is the course map that shows how this module fits into the current instructional goal.

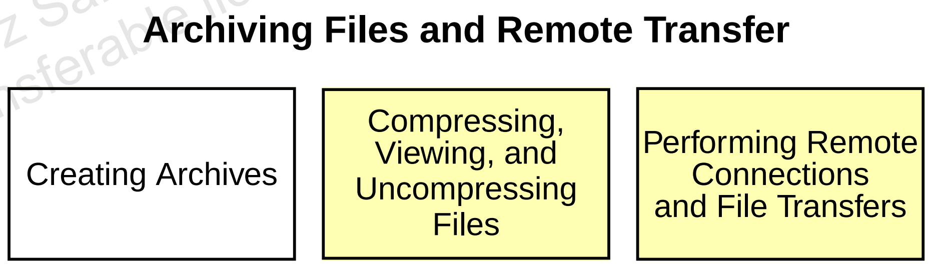

Archiving Files and Remote Transfer
Creating Archives
Compressing, Viewing, and Uncompressing Files
Performing Remote Connections and File Transfers

Figure 13-1 Course Map

Archiving Files

To safeguard your files and directories, you can create a copy, or archive of the files and directories on a removable medium, such as a USB flash drive or a cartridge tape. You can use the archived copies to retrieve lost, deleted, or damaged files.

Archiving Techniques

You can use several commands to store, locate, and retrieve files on a USB flash drive, a tape device, or from an archive file. Two of the commands you can use are:

- The `tar` command, to create and extract files from a file archive or any removable media, such as a USB flash drive or a tape device.
- The `jar` command, to combine multiple files into a single archive file.

The `tar` Command

The `tar` command archives files to and extracts files from a single file called a `tar` file. The default device for a `tar` file is a magnetic tape device.

The syntax for the `tar` command is:

```
tar functions archivefile filenames
```

Note – You should use relative pathnames to archive files.

Table 13-1 shows the functions associated with the `tar` command.

Table 13-1 Functions for the `tar` Command

Function	**Definition**
`c`	Creates a new `tar` file.
`t`	Lists the table of contents of the `tar` file.
`x`	Extracts files from the `tar` file.
`f`	Specifies the archive file or tape device. The default tape device is `/dev/rmt/0`. If the name of the archive file is "`-`", the `tar` command reads from the standard input when reading from a `tar` archive or writes to the standard output if creating a `tar` archive.
`v`	Executes in verbose mode, writes to the standard output.
`h`	Follows symbolic links as standard files or directories.

Creating an Archive

You can use the `tar` command to create an archive file that contains multiple files or directories. You can then place the file on a USB flash drive or a tape so that other users can share the file or attach it to email messages.

Creating an Archive on a Tape

To create an archive on a tape, first verify that the system has a tape drive available. You can use the `mt` utility with the `status` option to print status information about the tape unit. You use the `mt` utility to send commands to a magnetic tape drive.

Note – For more information on the use of the `mt` command to control a magnetic tape drive, refer to the `mt` man page.

The following example shows you how to use the default tape device to archive your home directory. In this example, `student` creates a tape archive of the `student` home directory.

```
$ cd
$ mt -f /dev/rmt/0 status
<output will be your local tape device info>
$ tar cvf /dev/rmt/0 .
a ./ 0 tape blocks
a ./.rhosts 1 tape blocks
a ./dante 3 tape blocks
a ./fruit 1 tape blocks
a ./dante_1 1 tape blocks
a ./dir1/ 0 tape blocks
a ./dir1/coffees/ 0 tape blocks
a ./dir1/coffees/beans/ 0 tape blocks
a ./dir1/coffees/beans/beans 24 tape blocks
... (output truncated)
```

You can also use the `tar` command to create an archive file containing multiple files or directories.

The following example shows you how to archive the `file1`, `file2`, and `file3` files in an archive file called `files.tar`.

```
$ cd
$ tar cvf files.tar file1 file2 file3
a file1 2K
a file2 1K
a file3 1K
```

Creating an Archive on a USB Flash Drive

The Volume Management daemon, `vold` provides automatic detection of removable media i.e., `vold` daemon is now hot-plug aware. This means that if you insert removable media, the media is automatically detected and mounted by `vold`. You do not need to restart `vold` manually to recognize and mount a file system from any removable media device.

Note – You can use the `rmformat` command to check for all removable media managed by `vold`.

If you are using a legacy or non-USB diskette device, then you might need to issue the `volcheck` command before `vold` can recognize the media. If the media is detected, but for some reason, is unmounted, then you'll need to run the following command:

```
$ volrmmount -i rmdisk0
```

Entering the `volrmmount` command creates the `/rmdisk` directory and its content when a flash drive is present. You can use the `cd` command to access files on the flash drive by changing to `/rmdisk/rmdisk0` directory. You can also use the `cp` command to copy an archive into the `/rmdisk/rmdisk0` directory.

The following example shows you how to use the `cp` command to copy an archive file from your home directory to a flash drive.

```
$ volrmmount -i rmdisk0
$ cd /rmdisk/rmdisk0
$ ls
$
$ cd
$ cp files.tar /rmdisk/rmdisk0
$ ls /rmdisk/rmdisk0
files.tar
```

To eject a flash drive:

1. Enter the `cd` command to change from the `/rmdisk/rmdisk0` directory to your home directory.

```
$ cd
```

2. Enter the `eject rmdisk0` command.

```
$ eject rmdisk0
```

You cannot eject removable devices while you are in the current working directory for the device. If you see an error message, such as `Device busy`, when trying to eject the flash drive, you might still be in the working directory on the flash drive. Enter the `pwd` command to see if you are in the `/rmdisk/rmdisk0` directory. If you are in the `/rmdisk/rmdisk0` directory, enter the `cd` command to return to your home directory. Then enter the `eject` command.

Viewing an Archive

You can view the names of all the files that have been written directly to a tape archive or a file archive.

Viewing an Archive From a Tape

To view the contents of the `student` home directory on a tape, enter the command:

```
$ tar tf /dev/rmt/0
/.rhosts
./dante
./fruit
./dante_1
./dir1/
./dir1/coffees/
./dir1/coffees/beans/
./dir1/coffees/beans/beans
./dir1/coffees/nuts
./dir1/coffees/brands
./dir1/fruit/
./dir1/trees/
<directory list truncated>
```

Viewing Files in an Archive File

To view the contents of the `files.tar` archive file, enter the command:

```
$ tar tf files.tar
file1
file2
file3
```

Retrieving tar Archive Data

You can retrieve or extract the contents of an archive that was written directly to a tape device or to a file.

Retrieving a Directory From a Tape

If the contents of your home directory are deleted, you can extract the directory contents from an archive tape.

To retrieve all the files from the tape archive, enter the commands:

```
$ cd
$ tar xvf /dev/rmt/0
x ., 0 bytes, 0 tape blocks
x ./.rhosts, 2 bytes, 1 tape blocks
x ./dante, 1319 bytes, 3 tape blocks
x ./fruit, 57 bytes, 1 tape blocks
x ./dante_1, 368 bytes, 1 tape blocks
x ./dir1, 0 bytes, 0 tape blocks
x ./dir1/coffees, 0 bytes, 0 tape blocks
x ./dir1/coffees/beans, 0 bytes, 0 tape blocks
x ./dir1/coffees/beans/beans, 12288 bytes, 24 tape blocks
x ./dir1/coffees/nuts, 0 bytes, 0 tape blocks
x ./dir1/coffees/brands, 0 bytes, 0 tape blocks
x ./dir1/fruit, 0 bytes, 0 tape blocks
x ./dir1/trees, 0 bytes, 0 tape blocks
... (output truncated)
```

You can extract files from an archive file using the `tar` command. The following example shows you how to extract files from the `files.tar` archive file for placement into the current directory.

```
$ tar xvf files.tar
tar: blocksize = 11
x file1, 1610 bytes, 4 tape blocks
x file2, 105 bytes, 1 tape blocks
x file3, 218 bytes, 1 tape blocks
```

Retrieving Files From a Flash Drive

To retrieve files archived to a flash drive, you can follow the same process as you did for archiving to a flash drive except that you copy files from the flash drive back to your home directory.

The following example shows you how to retrieve the archived file `files.tar` from a flash drive.

```
$ volrmmount -i rmdisk0
$ cd /rmdisk/rmdisk0
$ ls
files.tar
$ cp files.tar /export/home/student
$ cd
$ ls files.tar
files.tar
$ tar xvf files.tar
```

Compressing and Archiving Files With the `jar` Command

This section describes how to use the `jar` command to compress and archive files. The `jar` command combines multiple files into a single archive file.

The `jar` Command

The `jar` command has syntax similar to the `tar` command, but the `jar` command compresses the named files in addition to archiving the files. The `jar` command copies and compresses multiple files into a single archive file. The `jar` command is a standard feature of the Solaris™ 10 Operating System (Solaris 10 OS). The `jar` command is available on any system that uses Java virtual machine (JVM™) software.

Note – The `jar` command was created to enable programmers working with Java™ technology to create a single archive that they could download instead of downloading multiple individual files. Also it allows java programs to run files from within a jar file, without having to extract the files. Also correct use of the manifest allows jar files to be run as a self contained executable program without having to extract the files.

The syntax for the `jar` command is:

```
jar options destination filenames
```

Options for the `jar` Command

Table 13-2 shows the options you can use with the `jar` command.

Table 13-2 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.

Table 13-2 Options for the `jar` Command (Continued)

`f`	Specifies the `jar` file to process (for example, `/tmp/`*`file`*`.jar`). The `jar` command sends the data to the screen (`stdout`) 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 shows how to use the `jar` command to copy and compress multiple files into a single archive file called `bundle.jar`.

```
$ ls
Reports     dir10       feathers    file1       fruit2
brands      dir2        feathers_6  file2       greetings
dante       dir3        file.1      file3       myvars
dante_1     dir4        file.2      file4       newdir
dir1        dir5        file.3      fruit       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: brands(in = 0) (out= 0)(stored 0%)
adding: dante(in = 1319) (out= 744)(deflated 43%)
adding: dante_1(in = 368) (out= 242)(deflated 34%)
adding: dat(in = 275) (out= 186)(deflated 32%)
adding: dir1/(in = 0) (out= 0)(stored 0%)
adding: dir1/coffees/(in = 0) (out= 0)(stored 0%)
adding: dir1/coffees/beans/(in = 0) (out= 0)(stored 0%)
adding: dir1/coffees/beans/beans(in = 12288) (out= 3161)(deflated 74%)
adding: dir1/coffees/nuts(in = 0) (out= 0)(stored 0%)
adding: dir1/fruit/(in = 0) (out= 0)(stored 0%)
adding: dir1/trees/(in = 0) (out= 0)(stored 0%)

... (output truncated)
```

Note – The `jar` command does not back up symbolic links as links. The `jar` command resolves the symbolic link and copies the file's contents.

Exercise: Archiving and Retrieving Files

In this exercise, you practice archiving, viewing, and retrieving files on a tape or a flash drive.

Preparation

Ensure that a tape device is connected to your system. Ask your instructor for any help required.

RLDC

In addition to being able to use local classroom equipment, this lab was designed to also use equipment located in a remote lab data center. Directions for accessing and using this resource can be found at:

`http://remotelabs.sun.com/`

Ask your instructor for the particular SSH configuration file that you should use to access the appropriate remote equipment for this exercise.

Tasks

To create a file archive, view the archive and retrieve files from the archive, complete the following steps:

Note – If you get a `Permission Denied` error message while performing the following exercises, check the write-protect switch on the tape cartridge. Archive your home directory to a file using the `tar` command.

1. What type of file does the `tar` command create?

   ____

2. Archive your home directory on a tape.

   ____

3. Which command do you use to mount a flash drive?

   ____

4. Which command do you use to copy files to a flash drive?

    ____

5. Archive your home directory to a file called `username.tar` in the `/tmp` directory using the `tar` command.

    ____

6. Use the `tar` command to view the contents of your home directory archive.

    ____

7. Create a new directory in your home directory called `Retrieve`. Use the `cd` command to move to the new directory. You can use the new directory to practice retrieving files from archives. Retrieve the contents of the archive `tar` file that you created on tape.

    ____

8. Ensure that all files in the `~/dir1` directory are readable, then use the `jar` command to archive the `~/dir1` directory.

    ____

9. Use the `tar` command to archive the `~/dir1` directory and the contents of the `dir1` directory.

    ____

10. Distinguish between the `tar` command and the `jar` command.

    ____

11. Compare the current size of the `tar` file and the `jar` file archives.

    ____

    Is there a difference in the size of the files?

    ____

Exercise Summary

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

- Experiences
- Interpretations
- Conclusions
- Applications

Exercise Solutions: Archiving and Retrieving Files

To create a file archive, view the archive and retrieve files from the archive, complete the following steps:

1. What type of file does the `tar` command create?

   A `tar` *file*

2. Archive your home directory on a tape.

```
$ cd
$ tar cvf /dev/rmt/0 .
```

3. Which command do you use to mount a flash drive?

   The `volrmmount -i rmdisk0` *command.*

4. Which command do you use to copy files to a flash drive?

   The `cp` *command.*

5. Archive your home directory to a file called *`username`*`.tar` in the `/tmp` directory using the `tar` command.

```
$ tar cvf /tmp/username.tar .
```

6. Use the `tar` command to view the contents of your home directory archive.

```
$ tar tf /tmp/username.tar
```

7. Create a new directory in your home directory called `Retrieve`. Use the `cd` command to move to the new directory. You can use the new directory to practice retrieving files from archives. Retrieve the contents of the archive `tar` file that you created on tape.

```
$ mkdir Retrieve
$ cd ~/Retrieve
$ tar xvf /dev/rmt/0
... (output truncated)
```

8. Ensure that all files in the `~/dir1` directory are readable, then use the `jar` command to archive the `~/dir1` directory.

```
$ cd
$ jar cvf dir1.jar dir1
```

9. Use the `tar` command to archive the `~/dir1` directory and the contents of the `~/dir1` directory.

```
$ cd
$ tar cvf dir1.tar dir1
```

10. Distinguish between the `tar` command and the `jar` command.

The `tar` *command archives files to a single file called a* `tar` *file. The* `jar` *command compresses the named files in addition to archiving the files.*

11. Compare the current size of the `tar` file and the `jar` file archives.

```
$ ls -l dir1.tar dir1.jar
```

Is there a difference in the size of the files?

Yes.

Notes:

Module 14

Compressing, Viewing, and Uncompressing Files

Objectives

This module introduces compressing files using the `compress` command. The module describes how to view compressed files using the `zcat` command. This module also describes the `uncompress` command.

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

- Compress files using the `compress` command
- View compressed files using the `zcat` command
- Uncompress files with the `uncompress` command
- Compress a file with the `gzip` command
- View files using the `gzcat` command
- Compress and archive multiple files with the `zip` command

Figure 14-1 is the course map that shows how this module fits into the current instructional goal.

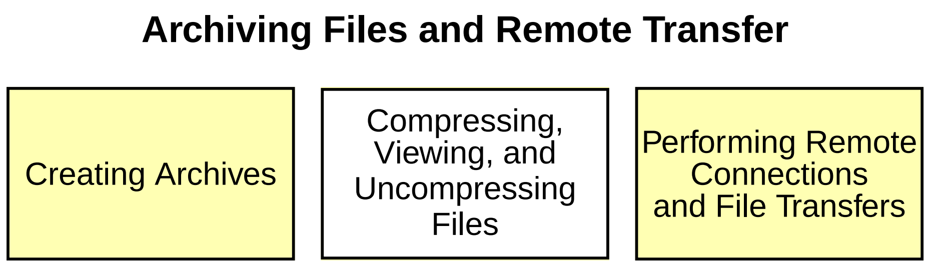

Archiving Files and Remote Transfer
Creating Archives
Compressing, Viewing, and Uncompressing Files
Performing Remote Connections and File Transfers

Figure 14-1 Course Map

Compressing Files Using the compress Command

Compressing files helps you to conserve disk space. You can use the compress command to reduce the size of a file. You can compress large files to reduce the use of disk space as well as the time required for file transfers over a network. The amount of compression depends on the type of file you compress. Typically, compression reduces a text file by 50 percent to 60 percent.

The syntax for the compress command is:

```
compress [ -v ] filename
```

For example:

```
$ compress dante
```

When you compress a file, the compress command replaces the original file with a new file that has a .Z extension. The ownership and modification time of the original file remain the same, but the contents of the file change.

Compressing a File

The following example shows you how to compress a file called files.tar, using the -v (verbose) option. Using this option with the compress command displays a message providing information about the percentage of reduction or expansion of each file.

```
$ compress -v files.tar
files.tar: Compression: 70.20% -- replaced with files.tar.Z
```

The compressed file, which replaces the files.tar file, is called files.tar.Z. When a file has a .Z extension it is a compressed file, and you should not view or print such a file without first uncompressing it.

Caution – When you compress a file that has already been compressed, the file size increases, instead of becoming smaller. When you rename a file that has already been compressed and you run the `compress` command on it once again, the file size increases, instead of becoming smaller.

If you do not use the `-v` (verbose) option with the `compress` command, no output appears when you run the command. For example:

```
$ compress files.tar
$
```

Viewing Compressed Files With the zcat Command

You can print the uncompressed form of a compressed file to standard output.

You can use the `zcat` command to view files that were compressed by the `compress` command. The `zcat` command interprets the compressed data and displays the contents of a file as if it were not compressed. The `zcat` command does not change the contents of a compressed file.

The syntax for the `zcat` command is:

```
zcat filename
```

Note – The `zcat filename` command is functionally identical to the `uncompress -c filename` command.

Viewing the Contents of a Compressed File

The following example shows you how to view the contents of the compressed file called `dante.Z`.

```
$ zcat dante.Z | more
                        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.  There is only one Dante, as we recognize
one Raphael, one Michelangelo, and one Galileo.
... (output truncated)
$
```

You can use the pipe (|) character with the zcat command to extract the contents of a compressed tar file without uncompressing the tar file first. For example:

```
$ zcat files.tar.Z | tar xvf -
```

The dash (-) at the end of the tar command indicates that the tar command should read the tar file from standard input instead of reading the tar file from a file or tape. In this example the standard input comes from the output of the zcat command because of the pipe (|).

Uncompressing Files Using the uncompress Command

You can restore a compressed file to its original state after it has been compressed. This section describes how you can uncompress files using the uncompress command.

The uncompress command restores a compressed file back to its original state.

The syntax for the uncompress command is:

```
uncompress options filename
```

Uncompressing a File

The following example shows you how to uncompress the files.tar.Z file and replace it with the original file named files.tar. The -v option causes the uncompress command to display messages about the action being performed.

```
$ uncompress -v files.tar.Z
files.tar.Z:  -- replaced with files.tar
$
```

Viewing the Contents of a Compressed File

You can use the uncompress command with the -c option to send the contents of a compressed file to the screen (stdout), without changing the compressed (.Z) file. You can use the pipe (|) character to send the output of the uncompress command to another program. You can use the tar command to list the contents of the file that the uncompress command is reading.

```
$ uncompress -c files.tar.Z | tar tvf -
tar: blocksize = 11
-rw-rw---- 1233/10    1610 Feb 7 14:12 2009 file1
-rw-rw---- 1233/10     105 Feb 7 14:12 2009 file2
-rw-rw---- 1233/10     218 Feb 7 14:12 2009 file3
```

The dash (-) at the end of the command-line indicates that the tar command reads the data from the piped output of the uncompress command rather than a tar file or a tape.

Compressing a File With the gzip Command

You can also use the gzip command to reduce the size of files. When the gzip command successfully compresses a file, the original file is replaced by a file with the same name and a .gz extension. The files keep the same ownership modes, access, and modification times. The gzip command is used to compress regular files.

The syntax for the gzip command is:

```
gzip [ -v ] filenames
```

The following example shows you how to compress the file1, file2, file3, and file4 files with the gzip command.

```
$ gzip file1 file2 file3 file4
$ ls *.gz
file1.gz file2.gz file3.gz file4.gz
```

Note – The gzip command performs the same general function as the compress command, but the gzip command generally produces smaller files.

Restoring a gzip File Using the gunzip Command

You can use the gunzip command to restore a file that has been compressed with the gzip command.

The following example shows you how to uncompress the file1.gz file.

```
$ gunzip file1.gz
```

Viewing Files With the `gzcat` Command

You can use the `gzcat` command to look at a compressed file.

The `gzcat` command can be used to view files that were compressed with either the `gzip` command or the `compress` command.

The `gzcat` command interprets the compressed data and displays the contents of the file as if it were not compressed. The `gzcat` command does not change the contents of the compressed file. The compressed file remains on the disk in compressed form.

The syntax for the `gzcat` command is:

```
gzcat filename
```

Viewing the Contents of a Compressed File

You can use the `gzcat` command to view the contents of a compressed `tar` file without having to first uncompress the file.

The following example shows you how to view the `file1.gz` file.

```
$ gzip file1
$ ls file1*
file1.gz
$ gzcat file1.gz
                    The Achievers

Unconsciously or not, they divide their work totally differently than the
sustainers do. Certainly Achievers work longer hours. New York magazine
has
published several surveys on work needs which reveal that well-known
typically work from to a million hours a week.
... (output truncated)
$
```

Compressing and Archiving Multiple Files Using the `zip` Command

You can compress and archive files with a single command. The following section describes how to perform this combined task using the `zip` command.

The `zip` command compresses multiple files into a single archive file. The `zip` command adds the `.zip` extension to the file name of the compressed archive file if you do not assign a new file name with an extension.

Note – You can run the `zip` command or the `unzip` command on the command-line to view a list of options used with each command.

The syntax for the `zip` command is:

```
zip target_filename source_filenames
```

The following example shows you how to use the `zip` command to compress the `file2` and `file3` files into the `file.zip` archive file.

```
$ zip file.zip file2 file3
adding: file2 (deflated 16%
adding: file3 (deflated 26%)
$ ls
file.zip
file2
file3
```

To list the files in a `zip` archive, enter the `unzip -l` command.

```
$ unzip -l zipfile
```

Restoring a zip File With the unzip Command

You can uncompress the contents of a zip file using the unzip command.

The following example shows you how to uncompress the file.zip archive file.

```
$ unzip file.zip
```

Note – The jar command and the zip command create files that are compatible with each other. The unzip command can uncompress a jar file, and the jar command can uncompress a zip file.

Exercise: Compressing and Restoring Files

In this exercise, you practice compressing and uncompressing files, and viewing compressed files.

Preparation

Note – Before you begin this lab, uncompress any file that you compressed in the examples in the module.

RLDC

In addition to being able to use local classroom equipment, this lab was designed to also use equipment located in a remote lab data center. Directions for accessing and using this resource can be found at:

```
http://remotelabs.sun.com/
```

Ask your instructor for the particular SSH configuration file that you should use to access the appropriate remote equipment for this exercise.

Tasks

To compress and uncompress files, complete the following steps. Write the commands that you would use to perform each task in the space provided.

1. In your home directory, compress the `dante` file and the `file1` file using the `compress` command.

   ____

2. What are the new names for the compressed versions of the `dante` file and the `file1` file?

   ____

3. Which three commands can you use to view the contents of a file that was compressed with the `compress` command?

   ___

4. Use the `gzip` command to compress the `file2` file and the `dante_1` file.

   ___

5. What are the new names for the compressed versions of the `file2` file and the `dante_1` file?

   ___

6. Distinguish between the `gzip` command and the `zip` command.

   ___

7. Use the `zip` command to compress the `file3` file, the `fruit2` file, and the `tutor.vi` file to a file called `files.zip`.

   ___

8. Which command do you use to view the compressed archive file called `files.zip`?

   ___

9. Do the original versions of the `file3` file, the `fruit2` file, and the `tutor.vi` file still exist?

   ___

10. Uncompress the `dante.Z` file and the `file1.Z` file.

    ___

    Do the `dante` file and the `file1` file still have a `.Z` extension on the file names?

    ___

11. Which command do you use to uncompress the `file2.gz` file and the `dante_1.gz` file?

    ___

    Do the `file2` file and the `dante_1` file still have a `.gz` extension on the file names?

    ___

12. Which command do you use to unarchive the `file3` file, the `fruit2` file, and the `tutor.vi` file from the `zip` file created in Step 6?

    ____

    Does the `files.zip` file still exist in the directory?

    ____

13. What happens when you compress a file that is already compressed?

    ____

Exercise Summary

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

- Experiences
- Interpretations
- Conclusions
- Applications

Exercise Solutions: Compressing and Restoring Files

To compress and uncompress files, complete the following steps:

1. In your home directory, compress the `dante` file and the `file1` file using the `compress` command.

```
$ compress dante
$ compress file1
```

2. What are the new names for the compressed versions of the `dante` file and the `file1` file?

   ```
   dante.Z
   file1.Z
   ```

3. Which three commands can you use to view the contents of a file that was compressed with the `compress` command?

   The `uncompress -c filename` *command, the* `zcat filename` *command, or the* `gzcat filename` *command.*

4. Use the `gzip` command to compress the `file2` file and the `dante_1` file.

```
$ gzip file2 dante_1
```

5. What are the new names for the compressed versions of the `file2` file and the `dante_1` file?

   The names are:

   ```
   file2.gz
   dante_1.gz
   ```

6. Distinguish between the `gzip` command and the `zip` command.

   The `gzip` *command reduces the size of files, whereas the* `zip` *command compresses one or more files into a single* `zip` *archive.*

7. Use the `zip` command to compress the `file3` file, the `fruit2` file, and the `tutor.vi` file to a file called `files.zip`.

```
$ zip files.zip file3 fruit2 tutor.vi
  adding: file3 (deflated 25%)
  adding: fruit2 (deflated 17%)
  adding: tutor.vi (deflated 75%)
```

8. Which command do you use to view the compressed archive file called `files.zip`?

   The `unzip -l files.zip` *command.*

9. Do the original versions of the `file3` file, the `fruit2` file, and the `tutor.vi` file still exist?

   Yes.

10. Uncompress the `dante.Z` file and the `file1.Z` file.

```
$ uncompress dante.Z
$ uncompress file1.Z
```

Do the `dante` file and the `file1` file still have a `.Z` extension on the file names?

No.

11. Which command do you use to uncompress the `file2` file and the `dante_1` file?

    The `gunzip file2.gz dante_1.gz` *command.*

    Do the `file2` file and the `dante_1` file still have a `.gz` extension on the file names?

    No.

12. Which command do you use to unarchive the `file3` file, the `fruit2` file, and the `tutor.vi` file from the `zip` file created in Step 6?

```
$ unzip files.zip
Archive: files.zip
replace file3? [y]es, [n]o, [A]ll, [N]one, [r]ename: A
  inflating: file3
  inflating: fruit2
  inflating: tutor.vi
```

Does the `files.zip` file still exist in the directory?

Yes

13. What happens when you compress a file that is already compressed?

    Compressing a file that has already been compressed results in a file size that is larger and not smaller.

Module 15

Performing Remote Connections and File Transfers

Objectives

This module introduces remote connections and file transfers. The module also describes the `rlogin` command, the `rsh` command, the `rcp` command, the `telnet` command, and the `ftp` command.

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

- Establish a remote login session
- Copy files or directories to and from another system
- Transfer files between systems

Figure 15-1 is the course map that shows how this module fits into the current instructional goal.

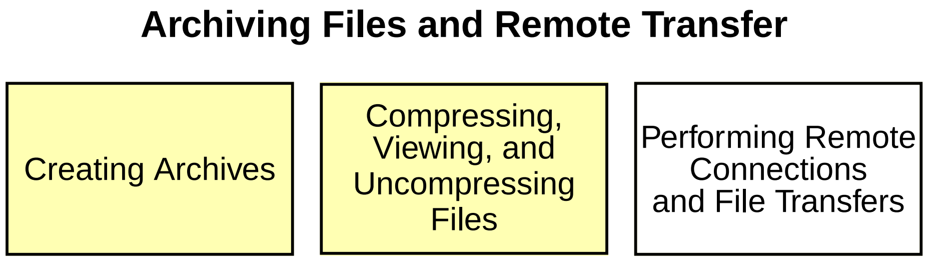

Archiving Files and Remote Transfer
Creating Archives
Compressing, Viewing, and Uncompressing Files
Performing Remote Connections and File Transfers

Figure 15-1 Course Map

Establishing a Remote Login Session

A remote login session enables you to access a remote host from your current machine and to use the resources of the remote host. This section describes how to log into your account remotely from another system on a network.

Introducing an Example Network

A network is a connection of computer systems that enables an exchange of information among the systems. A network also enables you to access your user account from another system. 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

A host is a computer system on a network. The local host is your current working system. A remote host is a different system that you access from your local host.

Note – The client is a host or a process that uses services from another program, known as a server.

Figure 15-2 shows the relationship between the network and the host.

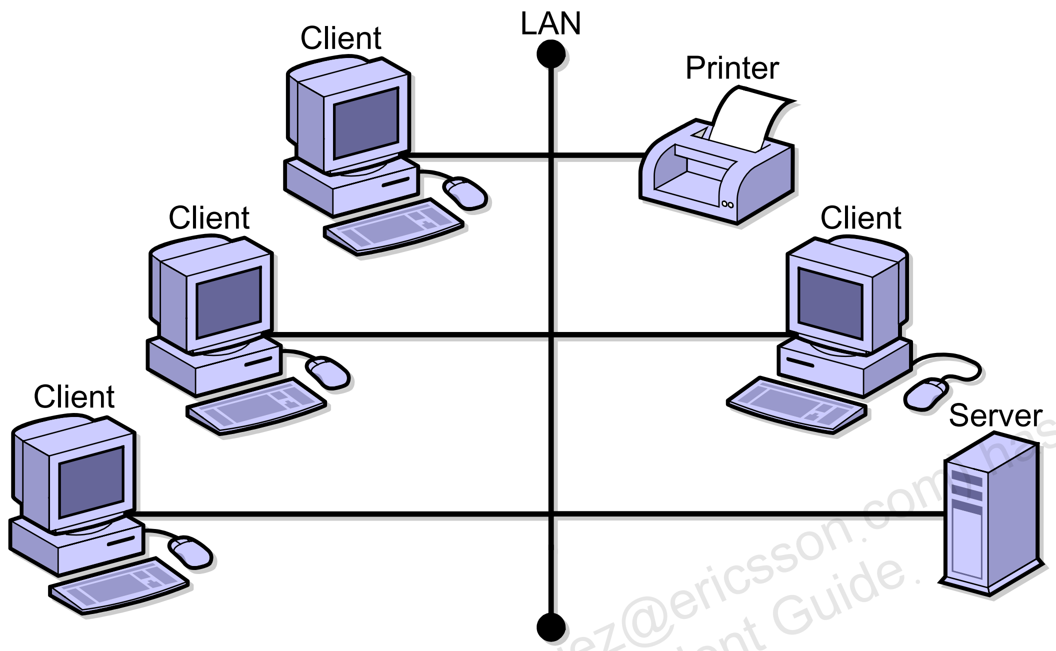


Figure 15-2 Example Network

The `~/.rhosts` File

When a remote user requests a local host for login access, the local host searches the local `/etc/passwd` file for an entry for that user. If an entry for the user exists, the user can log into the local host from a remote system. If a password is associated with the account, the remote user supplies the password to gain system access. If no entry exists in the local `/etc/passwd` file for the remote user, the remote user cannot access the system.

The `~/.rhosts` file provides another authentication procedure to determine if a remote user can access the local host with the identity of a local user. This procedure bypasses the password authentication. The system checks the `~/.rhosts` file in the home directory of the local host user to determine if the remote user can access the system.

Note – If the user's `.rhosts` file contains a plus (+) character, then the user is able to log in from any known system on the network without supplying a password.

Using the `rlogin` Command

You can use the `rlogin` command to establish a remote login session on another system.

The syntax for the `rlogin` command is:

```
rlogin hostname
```

The following example shows you how to use the `rlogin` command to log into another system remotely. You can use the `uname -n` command to verify the name of the remote system.

```
$ rlogin host2
Last login: Wed Feb 20 19:52:32 from host1
Sun Microsystems Inc.   SunOS 5.10      Generic January 2005
$ uname -n
host2
$ pwd
/export/home/student
$ exit
Connection to host2 closed.
$
```

You can use the `rlogin` command with the `-l` option to specify a different user name for a remote login session.

The syntax for the `rlogin -l` command is:

```
rlogin -l username hostname
```

To log in as a different user, you can use the following information to identify and log into the account:

- The host name
- The user name
- The password for the user on the remote host

The following example shows you how to log in remotely to the `host2` system as another user named `user2`.

```
$ rlogin -l user2 host2
Password:
Last login: Wed Feb 20 20:00:20 from host2
Sun Microsystems Inc.   SunOS 5.10      Generic January 2005
... (output truncated)
$ pwd
/export/home/user2
$ uname -n
host2
$ exit
Connection to host2 closed.
$
```

Running a Program on a Remote System

You use the `rsh` command to run a program on a remote system without having to log into the remote system.

The syntax for the `rsh` command is:

```
rsh hostname command
```

Note – The syntax of the `rsh` command also allows you to use the system's *`IP-Address`* on the command-line instead of *`hostname`*.

The following example shows you how to use the `rsh` command as `student` to run the `ls` command remotely on the `host2` system.

```
$ rsh host2 ls
```

The following example shows how `student` runs the `rsh` command to run the `ls /var/mail` command on the remote system named `host2`.

```
$ rsh host2 ls /var/mail
```

The output from these commands shows information from the `host2` system.

Note – The `rsh` command works only if a `.rhosts` file exists for the user because the `rsh` command does not prompt for a password to authenticate new users.

Terminating a Process Remotely by Logging on to Another System

If your system is not responding to your keyboard or to mouse input, the window system might be frozen. You can use the `rlogin` command to access your system remotely from another system. Then you can run the `pkill` command to terminate the corrupted session.

The following example shows you how to use the `rlogin` command to terminate a process remotely on the `host2` system.

```
$ rlogin host2
Password:
Last login: Wed Feb 20 19:52:32 from host1
Sun Microsystems Inc.   SunOS 5.10      Generic January 2005
$ pkill shell
```

or

```
$ pkill -9 shell
```

Using the ssh (Secure Shell) Remote Login Command

You can use the `ssh` program to log into a remote system and execute commands on a remote system. It is intended to replace `rlogin` and `rsh`, and to provide secure encrypted communications between two untrusted hosts over an insecure network.

Note – X11 connections and arbitrary TCP/IP ports can also be forwarded over the secure channel.

The syntax for the `ssh` command is:

```
ssh [-l login_name] hostname | user@hostname [ command]
```

The `ssh` program connects and logs into the specified hostname. The user must prove his or her identity to the remote system using one of several methods depending on the protocol version used:

- First, if the system that the user logs in from is listed in `/etc/hosts.equiv` or `/etc/shosts.equiv` on the remote system, and the user names are the same on both sides, the user is immediately permitted to log in.
- Second, if `.rhosts` or `.shosts` exists in the user's home directory on the remote system and contains a line with the name of the client system and the name of the user on that system, the user is permitted to log in.

Note – This form of authentication alone is normally not allowed by the server because it is not secure.

Using the `telnet` Command

You can use the `telnet` command to log into a remote system and work in that environment.

The syntax for the `telnet` command is:

```
telnet hostname
```

The following example shows how to use the `telnet` command to connect to a remote system called `host2`.

```
$ telnet host2
Trying host2
Connected to host2
Escape character is '^]'.

SunOS 5.10

login: user2
Password:
Last login: Wed Feb 20 20:00:20 from host1
Sun Microsystems Inc.   SunOS 5.10      Generic January 2005
$ uname -n
host2
$ exit
Connection to hostname closed by foreign host.
$
```

Note – The `telnet` command always prompts for the user's password, and does not use the `.rhosts` file.

Copying Files or Directories to and From Another System

You can use the `rcp` command to copy files or directories from one host to another. To copy subdirectories and their contents, you can use the `rcp -r` command.

The syntax for the `rcp` command is:

```
rcp source_file hostname:destination_file
```

or

```
rcp hostname:source_file destination_file
```

or

```
rcp hostname:source_file hostname:destination_file
```

Note – The *`source_file`* file is your original file and the *`destination_file`* file is the copy of the original file.

Copying Files From a Local Directory to a Remote Host

You can use the `rcp` command to copy files from a local directory to a remote host. The `rcp` command checks the `~/.rhosts` file to determine if you have the necessary access permissions.

The following example shows you how to copy the `dante` file from the local directory to the `/tmp` directory on a system called `host2`.

```
$ rcp dante host2:/tmp
```

Caution – Do not use the `/tmp` directory for long-term storage of important files. The `/tmp` directory is emptied each time you reboot the system.

Copying Files From a Remote Host to a Local Directory

You can use the rcp command to copy files from a remote host to the local /tmp directory. The /tmp directory is used because it has read, write, and execute permissions for each category of user.

The following example shows you how to copy the dante file from a remote host called host2 to the local /tmp directory.

```
$ rcp host2:/tmp/dante /tmp
```

Copying Remote Directories

You can use the rcp command with the -r option to copy directories to and from another system. If your current working directory contains the file or directory that you want to copy, use the file or directory name. You do not need to use the absolute pathname.

The following example shows you how to copy the perm directory in the local home directory to the /tmp directory on the host system called host2.

```
$ rcp -r ~/perm host2:/tmp
```

Transferring Files Between Systems

This section explains how you can transfer files between systems using File Transfer Protocol (FTP). The `ftp` command allows you to copy files from one computer to another over a network.

Using the `ftp` Command

The `ftp` command uses standard FTP to transfer files between systems with similar or dissimilar operating systems. You use FTP to transfer files in the American Standard Code for Information Interchange (ASCII) mode or the binary mode.

FTP does not use the `.rhosts` file for authentication purposes. Instead it prompts you for a password that is linked to the user as whom you sign in. If you are using anonymous FTP, your password is usually a valid email address.

The syntax for the `ftp` command is:

```
ftp hostname
```

When you access a remote system using the `ftp` command, some file and directory access commands, such as the `ls` and `cd` commands are available at the `ftp>` prompt.

If the permissions allow you to view the contents of a directory, the `ls` command lists files in a directory. If you do not have access to a directory or file, FTP generates a "Permission denied" error.

The `cd` command changes the current working directory on the remote system.

Note – You can use the `lcd` command at the `ftp>` prompt to change the current working directory on your local system.

To end an `ftp` session, enter the `bye` command or the `quit` command at the `ftp>` prompt.

Introducing FTP Transfer Modes

In Solaris 9 OS and later, the binary mode for file transfer is the default mode using an `ftp` connection. The binary mode enables you to transfer binary, image, or any non-text files. Because the default mode for file transfer is the binary mode for an `ftp` connection in the Solaris 9 OS and later, you do not have to type the `bin` command to ensure complete data transfer.

Note – In the Solaris 8 OS or earlier versions, the default mode for an `ftp` connection is the ASCII mode. This default mode transfers plain files such as text files. Therefore, to transfer binary, image, or any non-text files you have to type the `bin` command to ensure complete data transfer in Solaris 8 OS, and earlier versions.

Transferring Files Using ASCII Mode

The following example establishes an FTP connection from the `host1` system to the `host2` system, gets the `feathers` file from the `user2` directory on `host2`, stores the `feathers` file in the `student` home directory on `host1`, and quits the `ftp` session.

```
$ ftp host2
Connected to host2.
220 host2 FP server ready.
Name (host2:student): user2
331 Password required for user2.
Password: password
230 User user2 logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ascii
200 Type set to A.
ftp> lcd ~student
Local directory now /export/home/student
ftp> ls
200 PORT command successful.
150 Opening ASCII mode data connection for file list.
dante
dante_1
dir1
dir2
dir3
```

```
dir4
dir5
file.1
file.2
file.3
file1
file2
file3
file4
fruit
fruit2
practice
tutor.vi
(directory list truncated)
226 Transfer complete.
133 bytes received in 0.081 seconds (1.61 Kbytes/s)
ftp> get fruit
200 PORT command successful.
150 Opening ASCII mode data connection for fruit (57 bytes).
226 Transfer complete.
local: fruit remote: fruit
66 bytes received in 0.042 seconds (1.54 Kbytes/s)
ftp> bye
221-You have transferred 66 bytes in 1 files.
221-Total traffic for this session was 1326 bytes in 4 transfers.
221-Thank you for using the FTP service on host2.
221 Goodbye.
$
```

Transferring Multiple Files

You can use the mget command to transfer multiple files from the remote system to the current working directory on the local system.

You can use the mput command to transfer multiple files from the local system to a directory on the remote system.

You can use the prompt command to switch interactive prompting on and off. The system is set up to prompt you by default. When prompting is on during multiple file transfers, the system prompts you to confirm the transfer of each file before completing the transfer.

The following example shows how you can establish an ftp connection from the host1 system to the host2 system. The ftp connection enables you to transfer multiple files using the prompt command, the mget command, and the mput command.

```
$ ftp host2
Connected to host2.
220 host2 FTP server ready.
Name (host2:user2): user2
331 Password required for user2.
Password:
230 User user2 logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
200 PORT command successful.
150 Opening ASCII data connection for file list.
file.1
file1
file.2
file2
file.3
file3
file4
fruit
fruit2
(file list truncated)
226 Transfer complete
52 bytes received in 0.028 seconds (1.79 Kbytes/s)
ftp> prompt
Interactive mode off
ftp> mget file.1 file.2
```

```
200 PORT command successful.
150 Opening BINARY mode data connection for file.1 (0 bytes).
226 Transfer complete.
200 PORT command successful.
150 Opening BINARY mode data connection for file.2 (0 bytes).
226 Transfer complete.
ftp> mput file3 file4
200 PORT command successful.
150 Opening BINARY mode data connection for file3.
226 Transfer complete.
200 PORT command successful.
150 Opening BINARY mode data connection for file4.
226 Transfer complete.
ftp> prompt
Interactive mode on.
ftp> mget file.1 file.2
mget file.1? y
200 PORT command successful.
150 Opening BINARY mode data connection for file.1 (0 bytes).
226 Transfer complete.
mget file.2? y
200 PORT command successful.
150 Opening BINARY mode data connection for file.2 (0 bytes).
226 Transfer complete.
ftp> bye
221-You have transferred 0 bytes in 8 files.
221-Total traffic for this session was 2654 bytes in 13 transfers.
221-Thank you for using the FTP service on host2.
221 Goodbye.
$
```

Transferring Files Using Binary Mode

The following example shows how to transfer a binary file in the Solaris 9 OS or later versions.

Note – The `binary.file` file is an example file for demonstration purposes only. The file is not located on your system.

```
$ cd /tmp
$ ftp host2
Connected to host2.
220 host2 FTP server ready.
Name (host2:user2): user2
331 Password required for user2.
Password:
230 User user2 logged in.
Remote system type is UNIX.
ftp> get binary.file
200 PORT command successful.
150 Opening BINARY mode data connection for binary.file (19084 bytes).
226 Transfer complete.
local: binary.file remote: binary.file
19084 bytes received in 0.0044 seconds (4212064 Kbytes/s)
ftp> bye
221-You have transferred 19084 bytes in 1 files.
221-Total traffic for this session was 19507 bytes in 1 transfers.
221-Thank you for using the FTP service on host2.
221 Goodbye.
$
```

Exercise: Performing Remote Connection Commands

In this exercise, you will use some of the remote connection commands introduced in this module.

Preparation

Launch the calculator on each system prior to beginning the exercise, by performing the `gcalctool` command.

RLDC

In addition to being able to use local classroom equipment, this lab was designed to also use equipment located in a remote lab data center. Directions for accessing and using this resource can be found at:

`http://remotelabs.sun.com/`

Ask your instructor for the particular SSH configuration file that you should use to access the appropriate remote equipment for this exercise.

In this particular exercise you will be required to access another system from the one you are using. To do this you will need to use the system of another student. Coordinate this activity with a lab partner.

Tasks

To use remote connection commands, complete the following steps. Write the commands that you would use to perform each task in the space provided.

1. Perform the `rlogin` command to log into another system in your classroom.

   ____

   Which directory are you accessing on the remote system?

   ____

2. Perform the command that shows you the host name of the current system.

   ____

3. Perform the `ps` command to identify the PID of the `gcalctool` command on the remote system.

   ___

4. Terminate the `gcalctool` command using the PID.

   ___

   Are you able to terminate the process? Why or why not?

   ___

5. Log out of the remote system.

   ___

6. Display the host name of your current system to determine if you are back on your host system.

   ___

7. Which remote connection command do you use to run a program remotely?

   ___

8. Which command do you use to terminate a process by remotely logging into another system?

   ___

9. Which command do you use to copy files remotely?

   ___

10. Copy the `dante` file from your local directory to the `/export/home/student/dir1` directory on the remote host.

    ___

11. Copy the `dante` file from the remote system back to the home directory on your local system.

    ___

12. Which command do you use to copy remote directories?

    ___

13. Which remote connection command do you use to transfer files from system to system, including binary files?

    ___

14. How do you end an `ftp` session?

    ___

15. How do you simultaneously transfer multiple files from a remote system to your current working directory on your local system?

    ____

16. How do you transfer multiple files from the local system to a directory on 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

Exercise Solutions: Performing Remote Connection Commands

To use remote connection commands, complete the following steps:

1. Perform the `rlogin` command to log into another system in your classroom.

```
$ rlogin hostname
```

Which directory are you accessing 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. Perform the command that shows you the host name of the current system.

```
$ uname -n
```

3. Perform the `ps` command to identify the PID of the `gcalctool` command on the remote system.

```
$ ps -ef | grep gcalctool
```

4. Terminate the `gcalctool` command using the PID.

```
$ kill PID
```

Are you able to terminate the process? Why or why not?

If you log into the remote system as root, you are able to terminate the process. If you log into the remote system as the same user (same UID) as the user that started the process on the remote system, then you are also able to terminate the process. If you log into the remote system as some other user, then you are not able to terminate the process because you do not own it and do not have permission.

5. Log out of the remote system.

```
$ exit
```

6. Display the host name of your current system to determine if you are back on your host system.

```
$ uname -n
```

7. Which remote connection command do you use to run a program remotely?

The `rsh` *command.*

8. Which command do you use to terminate a process by remotely logging into another system?

   The `rlogin` *command to remotely log into your system from another system, followed by the* `pkill` *command.*

9. Which command do you use to copy files remotely?

   The `rcp` *command.*

10. Copy the `dante` file from your local directory to the `/export/home/student/dir1` directory on the remote host.

```
$ rcp dante hostname:/export/home/student/dir1
```

11. Copy the `dante` file from the remote system back to the home directory on your local system.

```
$ rcp hostname:/export/home/student/dir1/dante $HOME
```

12. Which command do you use to copy remote directories?

```
$ rcp -r directory_name hostname:/export/home/student
```

13. Which remote connection command do you use to transfer files from system to system, including binary files?

   The `ftp` *command.*

14. How do you end an `ftp` session?

   Type `bye` *or* `quit` *at the* `ftp>` *prompt.*

15. How do you simultaneously transfer multiple files from a remote system to your current working directory on your local system?

   Perform the `mget` *command.*

16. How do you transfer multiple files from the local system to a directory on the remote system?

   Perform the `mput` *command.*