ORACLE
UNIVERSITY

Hardware and Software
Engineered to Work Together

UNIX and Linux Essentials

Student Guide
D76989GC10
Edition 1.0 | May 2015 | D77816

Learn more from Oracle University at **oracle.com/education/**

ORACLE

Authors

Uma Sannasi

Pardeep K Sharma

Technical Contributor and Reviewer

Harald van Breederode

Editors

Anwesha Ray

Raj Kumar

Vijayalakshmi Narasimhan

Graphic Designer

Rajiv Chandrabhanu

Publishers

Jobi Varghese

Revathi Ramamoorthy

Contents

1 Introduction

Overview 1-2
Course Goals 1-3
Course Agenda: Day 1 1-4
Course Agenda: Day 2 1-5
Course Agenda: Day 3 1-6
Introductions 1-7
Your Learning Center 1-8
Your Practice Environment 1-9

2 Introduction to the UNIX Environments

Workflow Orientation 2-2
Objectives 2-3
Lesson Agenda 2-4
Multi-User, Multi-tasking, Time-Sharing Operating Systems (1/2) 2-5
Multi-User, Multi-tasking, Time-Sharing Operating Systems (2/2) 2-6
UNIX & Linux OS Structure 2-7
UNIX & Linux OS Structure: The Kernel 2-8
UNIX & Linux OS Structure: The File System 2-9
The File System 2-10
The root Directory “/” 2-11
Files 2-12
UNIX & Linux OS Structure: Processes 2-13
UNIX & Linux OS Structure: The Shell 2-14
Primary Shells 2-15
Additional Shells 2-16
Quiz 2-17
User Accounts 2-18
Components of a User Account 2-19
Roles and Rights Profile 2-21
User’s Home Directory 2-22
Quiz 2-23
UNIX Variants 2-24
Solaris 2-25
Linux 2-26

The Desktop Environment 2-27
Oracle Solaris Desktop 2-28
Oracle Linux Desktop 2-29
Logging In to Oracle Solaris Using the Desktop Login Screen 2-30
Logging In to Oracle Linux Using the Desktop Login Screen 2-31
Logging In Using the Command-line Option 2-32
Logging Out 2-33
Lesson Agenda 2-34
Executing Commands from the Command Line 2-35
Command-Line Syntax 2-36
Using UNIX Commands 2-37
Using Commands with Options 2-38
Using Commands with Arguments 2-41
Using Commands with Options and Arguments 2-42
Using Multiple Commands on the Command Line 2-43
Quiz 2-44
Using the Man Pages 2-45
Displaying the Man Pages 2-46
Scrolling Through the Man Pages 2-47
Searching the Man Pages 2-48
Searching the Man Pages: By Section 2-49
Searching the Man Pages: By Keyword 2-50
Accessing Online Product Documentation 2-51
Quiz 2-52
Summary 2-53
Practice 2 Overview: Introduction to the UNIX Environments 2-54

3 Working with Files and Directories
Workflow Orientation 3-2
Objectives 3-3
Lesson Agenda 3-4
Viewing Directories 3-5
Determining the Current Directory 3-6
Displaying the Directory Content 3-7
Displaying the Directory Content With Options 3-8
Quiz 3-11
Displaying File Types 3-12
Changing Directories 3-14
Changing Directories using Relative or Absolute Pathname 3-15
Home Directory 3-16
Returning to Your Home Directory 3-17

Quiz 3-18
Lesson Agenda 3-19
Viewing Files 3-20
Viewing Files: cat Command 3-21
Viewing Files: more Command 3-22
Viewing File Content: more Command 3-23
Viewing File Content: less Command 3-24
Viewing File Content: head Command 3-25
Viewing File Content: tail Command 3-26
Viewing File Content: wc Command 3-27
Quiz 3-28
Lesson Agenda 3-29
Copying Files and Directories 3-30
Copying Multiple Files 3-31
Copying Files: cp Command Options 3-32
Copying Files Recursively: -r or -R Option 3-33
Preventing Copy Overrides: -i Option 3-34
Moving and Renaming Files and Directories 3-35
Moving a File to Another Directory 3-36
Moving a Directory and Its Content 3-37
Renaming Files and Directories 3-38
Quiz 3-39
Lesson Agenda 3-40
Creating Files 3-41
Creating Directories 3-42
Removing Files 3-44
Removing Directories 3-46
Symbolic Links 3-48
Creating Symbolic Links 3-49
Removing Symbolic Links 3-51
Quiz 3-52
Lesson Agenda 3-54
Regular Expressions 3-55
Regular Expressions Wild Cards 3-56
Creating patterns using Regular Expressions 3-57
Searching Files and Directories 3-58
Searching Files and Directories: find Command 3-59
Searching Files and Directories: Linux also has a locate Command 3-60
Searching Within Files: grep Command 3-61
Searching Within Files: grep Command on Linux 3-62
Quiz 3-63

Summary 3-64
Practice 3 Overview: Working with Files and Directories 3-65

4 Using the vi (vim) Editor
Workflow Orientation 4-2
Objectives 4-3
Agenda 4-4
Introduction to the vi Editor 4-5
Accessing the vi Editor 4-6
The vi Editor 4-7
The vi Editor Modes 4-8
Switching Between Modes 4-9
Agenda 4-10
Viewing Files in the Read-Only Mode 4-11
Moving the Cursor Within the vi Editor 4-12
Inserting and Appending Text 4-14
Text-Deletion Commands 4-15
Edit Commands 4-16
Quiz 4-17
Search and Replace Commands 4-18
Copy and Paste Commands 4-19
Save and Quit Commands 4-20
Session Customization 4-21
Session Customization Commands 4-22
Quiz 4-23
Summary 4-25
Practice 4 Overview: Using the vi (vim) Editor 4-26

5 Using Commands Within the Default Shell
Workflow Orientation 5-2
Objectives 5-3
Lesson Agenda 5-4
Shell Expansions 5-5
Path Name Expansion 5-6
File Name Expansion 5-8
Asterisk (*) Character 5-9
Question Mark (?) Character 5-10
Square Bracket ([]) Characters 5-11
The Brace Expansion 5-12
Quiz 5-13
Lesson Agenda 5-14

Shell Metacharacters 5-15
Redirection Metacharacters 5-16
The File Descriptors 5-17
Command Communication Channels 5-18
Redirecting Standard Input (stdin) 5-19
Redirecting Standard Output (stdout) 5-20
Redirecting Standard Error (stderr) 5-21
The Pipe Character 5-22
Using the Pipe Character 5-23
Quoting Characters 5-24
Quiz 5-25
Lesson Agenda 5-26
Introduction to Variables 5-27
Displaying Shell Variables 5-28
Setting and Unsetting Shell Variables 5-29
Default Bash Shell Variables 5-30
Customizing bash Shell Variables: PS1 5-31
Customizing ksh Shell Variables: PS1 5-32
Customizing Shell Variables: PATH 5-33
Quiz 5-34
Lesson Agenda 5-35
Introducing Command History 5-36
Using the ! Command 5-39
Using the !! Command 5-40
Editing Commands on the Command Line 5-41
Invoking File Name Completion 5-42
Lesson Agenda 5-44
User Initialization Files 5-45
Default User Initialization Files for the Bash Shell 5-46
Configuring the ~/.bashrc File 5-47
Quiz 5-48
Summary 5-49
Practice 5 Overview: Using Commands Within the Default Shell 5-50

6 Using Basic File Permissions
Workflow Orientation 6-2
Objectives 6-3
Lesson Agenda 6-4
Securing Files and Directories 6-5
File and Directory Permissions 6-6
Viewing Permission Categories 6-7

Permission Groups 6-8
Permission Set 6-9
Interpreting File and Directory Permissions 6-11
Determining File or Directory Access 6-12
Interpreting the ls –n Command 6-13
Determining Permissions 6-14
Quiz 6-15
Lesson Agenda 6-16
Changing the Permissions 6-17
Changing Permissions: Symbolic Mode 6-18
Changing Permissions: Octal Mode 6-20
Quiz 6-24
Lesson Agenda 6-26
The umask Command 6-27
Determining umask Value 6-28
Applying the umask Value 6-29
Changing the umask Value 6-30
Summary 6-31
Practice 6 Overview: Using Basic File Permissions 6-32

7 Performing Basic Process Control
Workflow Orientation 7-2
Objectives 7-3
Agenda 7-4
A Process 7-5
Attributes of a Process 7-6
Process States 7-7
Process Subsystems 7-8
Agenda 7-9
Listing System Processes 7-10
Listing All Processes 7-11
Quiz 7-13
Terminating a Process 7-14
Terminating a Process: kill Command 7-15
Terminating a Process: pgrep & pkill Command 7-17
Terminating a Process: pkill Command 7-18
Forcefully Terminating a Process: Signal 9 (SIGKILL) 7-19
Quiz 7-20
Summary 7-21
Practice 7 Overview: Performing Basic Process Control 7-22

8 Using Advanced Shell Features in Shell Scripts

Workflow Orientation 8-2
Objectives 8-3
Agenda 8-4
Jobs in the Bash Shell 8-5
Job Control Commands 8-6
Running a Job in the Background 8-7
Running a Job in the Background: Examples 8-8
Quiz 8-9
The Alias Command 8-10
Command Sequence 8-11
Predefined Aliases 8-12
User-Defined Aliases 8-13
Deactivating an Alias 8-14
Removing an Alias 8-15
Quiz 8-16
Shell Functions 8-17
Defining a Function 8-18
Invoking a Function 8-19
Shell Options 8-20
Example of Shell Options: Activating the noclobber Option 8-21
Example of Shell Options: Deactivating the noclobber Option 8-22
Quiz 8-23
Agenda 8-24
Shell Scripts 8-25
Determining the Shell to Run a Shell Script 8-26
Creating a Shell Script 8-27
Executing a Shell Script 8-28
Comments in a Shell Script 8-29
Positional Parameters in a Shell Script 8-30
Shifting the Positional Parameters 8-31
Quiz 8-32
Checking the Exit Status 8-33
The test Command 8-34
Using the test Command 8-35
Conditional Expressions 8-36
The && Operator 8-37
The || Operator 8-38
The if Statement 8-39
The for Statement 8-40

The while Statement 8-41
The until Statement 8-42
The case Statement 8-43
Quiz 8-44
Summary 8-45
Practice 8 Overview: Using Advanced Shell Features in Shell Scripts 8-46

9 Archiving Files and Performing Remote Transfer
Workflow Orientation 9-2
Objectives 9-3
Agenda 9-4
Introduction to File Archival 9-5
The tar Command 9-6
The tar Command Options 9-7
Creating a tar Archive 9-8
Viewing a tar Archive 9-9
Extracting a tar Archive 9-10
Quiz 9-11
Agenda 9-13
File Compression 9-14
The compress Command 9-15
Compressing a File: compress Command 9-16
Viewing a Compressed File: zcat Command 9-17
Uncompressing a File: uncompress Command 9-18
Compressing a File: gzip Command 9-19
Viewing a Compressed File: gzcat Command 9-20
Uncompressing a File: gunzip Command 9-21
Compressing and Archiving Multiple Files: zip Command 9-22
Viewing and Uncompressing Archive Files: unzip Command 9-23
Compressing a File: bzip2 Command 9-24
Uncompressing a File: bunzip2 Command 9-25
Quiz 9-26
Agenda 9-28
Introduction to Networking 9-29
Layout of a Basic Network 9-30
Remote Login 9-31
SSH for Remote Login 9-32
Copying Files and Directories Between Remote Systems: scp Command 9-33
Copying Files and Directories 9-34
Copying Remote Directories 9-36
Quiz 9-37

Introducing FTP   9-38
FTP and SFTP Commands   9-39
FTP and SFTP Transfer Modes   9-40
Transferring Files Using ASCII Mode   9-41
Transferring Files Using Binary Mode   9-43
Transferring Multiple Files   9-45
SFTP   9-47
Quiz   9-48
Summary   9-50
Practice 9 Overview: Archiving Files and Performing Remote Transfer   9-51

1

Introduction

Overview

- Course goals
- Course agenda
- Introductions
- Your learning center
- Your practice environment

Welcome to the *Unix and Linux Essentials* course. This course is designed for students who have not previously used UNIX and do not know the basic commands for navigating through the OS. To be proficient in Oracle Solaris or Oracle Linux, students need to have basic knowledge of the UNIX operating system structure, such as the file system hierarchy and shell concepts. In addition, students need to know how to build and execute basic UNIX commands from the command line in order to use the operating system. Students can apply the knowledge and skills from this course to both the Oracle Solaris and Oracle Linux operating systems.

To begin, we would like to take about an hour to give you an introduction to the course. We'll start with the course goals, followed by the agenda, and introductions. We'll conclude with some details about the classroom setting. You will then receive an orientation of the practice environment.

Course Goals

The goals of this course are to:

- Provide you with the basic UNIX skills and knowledge to successfully perform simple tasks in an Oracle Solaris 11 or Oracle Linux environment
  - Work with files and directories
  - Use the vi editor to create and modify files
  - Use commands within the default shell
  - View and modify file and directory permissions
  - Manage processes
  - Use advanced shell features in shell scripts
  - Archive files and perform remote file transfer
- Provide you with some meaningful practices around the areas covered in this course

Course Agenda: Day 1

- Lesson 1: Course Introduction
- Lesson 2: Introduction to the UNIX environments
  - Describe the UNIX & Linux operating systems
  - Execute commands from the command line
- Lesson 3: Working with Files and Directories
  - Determine your location in the directory structure
  - View file content
  - Copy and move files and directories
  - Create and remove files and directories
  - Search for files and directories

The *Unix and Linux Essentials* course consists of three days of lecture and practices. Today we will cover lessons 1, 2, and 3.

Note: As part of each lesson, you will have the opportunity to apply what you have learned in a series of hands-on practices.

Course Agenda: Day 2

- Lesson 4: Using the vi (vim) editor
  - Access the vi editor
  - Modify files with the vi editor
- Lesson 5: Using Commands Within the Default Shell
  - Use shell expansion for generating shell tokens
  - Use shell metacharacters for command redirection
  - Use variables in the Bash shell to store values
  - Display the command history
  - Customize the user's work environment
- Lesson 6: Using Basic File Permissions
  - View file and directory permissions
  - Change permissions
  - Modify default permissions

Day 2, we will cover lessons 4, 5, and 6; each followed by hands-on practices.

Course Agenda: Day 3

- Lesson 7: Performing Basic Process Control
  - Describe a process and its attributes
  - Manage processes
- Lesson 8: Using Advanced Shell Features in Shell Scripts
  - Use advanced shell features
  - Write shell scripts
- Lesson 9: Archiving Files and Performing Remote Transfer
  - Archive and retrieve files
  - Compress, view, and uncompress files
  - Perform remote connections and file transfers

Day 3, we will cover lessons 7, 8, and 9; each followed by hands-on practices.

Introductions

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

Now that you have been introduced to the course, introduce yourself to the other students and the instructor, addressing the items listed in the slide.

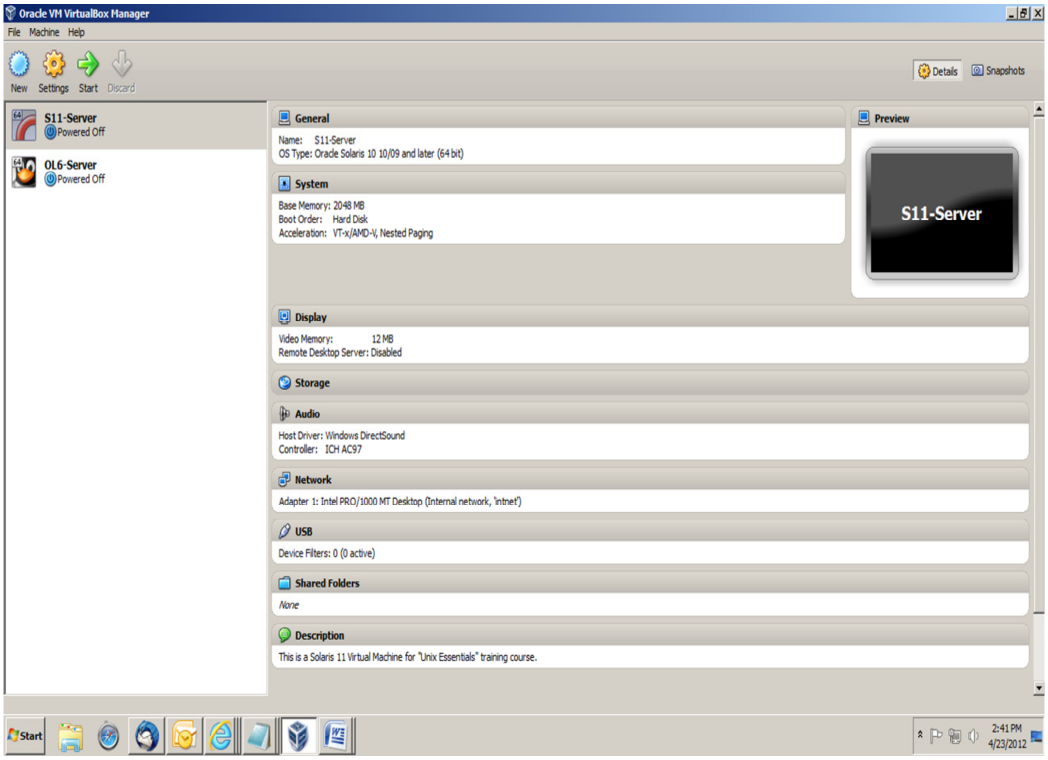
Your Learning Center

- Logistics
  - Restrooms
  - Break rooms and designated smoking areas
- Cafeterias and restaurants in the area
- Emergency evacuation procedures
- Instructor contact information
- Cell phone usage
- Online course attendance confirmation form

The instructor will acquaint you with the layout of the training facility, review the emergency evacuation procedures, provide you with contact information, review the use of cell phones in the classroom, and then walk you through the Oracle University online course attendance confirmation form.

Now that you have an idea of what we will be doing over the next five days, let's get started with an introduction to the practice environment.

Your Practice Environment



As part of each lesson, you will be given an opportunity to perform in a practice environment what you learned during the lecture. The practice environment that we use in this course is based on the Oracle VM VirtualBox virtualization software, an example of which is shown here. Within this VirtualBox environment one will find both the Solaris 11 server (S11-Server) and Oracle Linux 6 server (OL6-Server). When the practice are performed one can choose which environment to use.

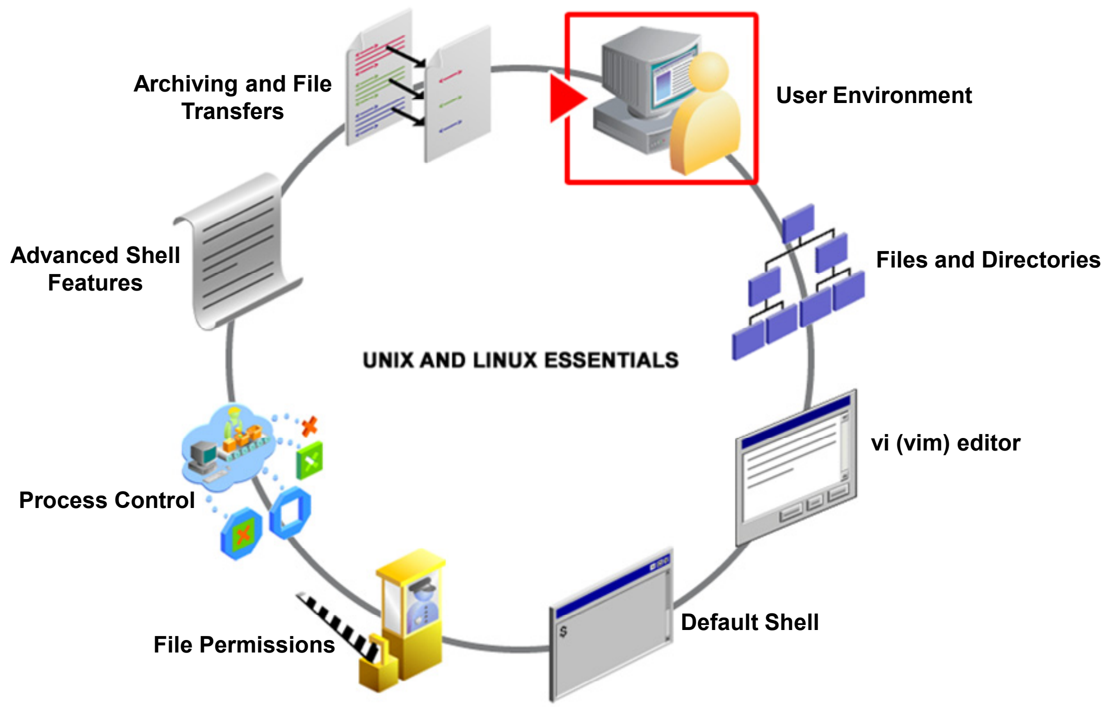
VirtualBox is a cross-platform virtualization application. It extends the capabilities of your existing computer so that it can run multiple operating systems, inside multiple virtual machines, simultaneously.

Open your activity guide to Practices for Lesson 1, Course Introduction. Your instructor will walk you through the material and you will have a chance to familiarize yourself with the practice environment configuration and setup.

2

Introduction to the UNIX Environments

Workflow Orientation



The previous lesson provided an overview of the course and its presentation modality. Lesson 2 introduces you to the UNIX & Linux operating systems (OS) and its primary components, such as kernel, shell, user environment, and directory hierarchy. In addition, there is a brief introduction to the Oracle Solaris 11 and Oracle Linux 6.2 GUIs and command-line interfaces; and the procedures to log in and out of the system. This lesson also explains how commands are executed from the command line.

Objectives

After completing this lesson, you should be able to:

- Describe the UNIX & Linux operating systems
- Execute commands from the command line

Lesson Agenda

- Describing the UNIX & Linux operating systems
- Executing commands from the command line

Multi-User, Multi-tasking, Time-Sharing Operating Systems (1/2)

- In 1965, Multics was being developed by MIT, Bell Labs (AT&T) and GE.
- In 1969, Bell Labs withdrew from the Multics project and began to work on it's own UNIX OS.
- In 1977, programmers at the University of California at Berkeley made significant enhancements, including networking capability resulting in Berkeley Software Distribution (BSD UNIX).
- In 1983, Richard Stallman, formally of MIT's AI Lab, began the GNU & Free Software Foundation to develop a free UNIX like OS called GNU (GNU's Not UNIX).

The first cut at a multi-user, multi-tasking OS was the Multics Project started in 1965 with contributions from MIT, Bell Labs (AT&T) and General Electric.

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

In 1977, the programmers at the University of California at Berkeley made significant modifications to the original source code from Bell Labs and called the resulting OS the Berkeley Software Distribution (BSD) UNIX. This version of the UNIX environment was sent to other programmers around the country, who added tools and code to further enhance BSD UNIX. Possibly the most important enhancement made to the OS by the programmers at Berkeley was adding networking capability. This enabled the OS to operate in a local area network (LAN).

Multi-User, Multi-tasking, Time-Sharing Operating Systems (2/2)

- In 1987, AT&T UNIX, BSD UNIX, and other UNIX OSs were folded into System V Release 4 (SVR4) UNIX, which became an industry standard for the OS.
- In 1991, the GNU OS was still in need of a kernel, Linus Torvalds, a Finnish Computer Science student, offered to create a Linux kernel for the GNU OS. In September, the Linux Kernel and the GNU userspace programs were released as the Linux OS.

In 1987, AT&T UNIX, BSD UNIX, and other UNIX OSs were folded into what became System V release 4 (SVR4) UNIX. This was a new generation OS, which became an industry standard.

The new SVR4 UNIX 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.

In 1991, while the GNU userspace programs were well on their way to being completed. The Hurd Kernel was still not working . A computer science student at the University of Helsinki, Linus Torvalds, offered to create a kernel for the Intel x86 architecture. In September 1991, Linus released the first version of the Linux kernel and in conjunction with the GNU userspace programs became the Linux OS.

UNIX & Linux OS Structure

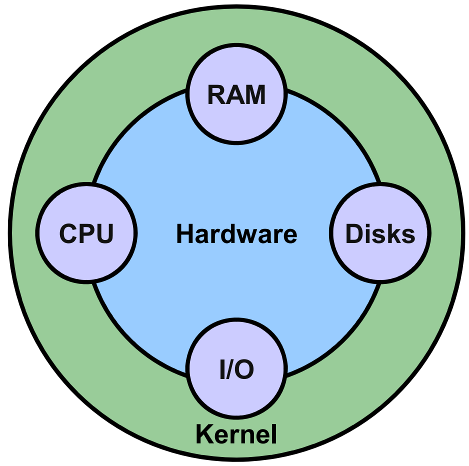
The UNIX & Linux OSs are structured around the following parts:

- Kernel
- File system
- Processes
- Shell

UNIX & Linux OS Structure: The Kernel

The kernel is the core of the OS and 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 Central Processing Unit (CPU) functions
- Memory management



UNIX & Linux OS Structure: The File System

- All data in UNIX & Linux (UNIX) is organized into files.
- All files are organized into directories.
- These directories are organized into a file system.

The File System

- A file system is a logical collection of files on a partition, slice or disk.
- A UNIX file system is a collection of files and directories that has the following properties:
  - It has a root directory (/) that contains other files and directories.
  - Each file or directory is identified by its name, the directory in which it resides, and a unique identifier, called an inode.
  - Each file system is self contained, in that there are no dependencies between one file system and another.

The following file systems are available in Oracle Solaris:

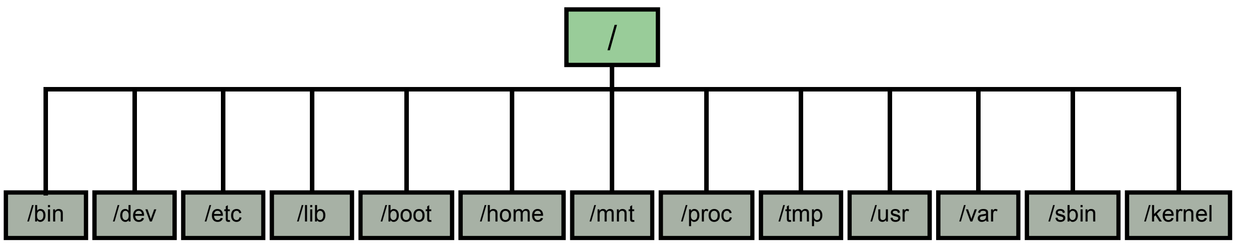
- Disk-based file systems – HSFS, PCFS, UDFS, UFS, and ZFS.
- Network-based file systems – NFS and SMB
- Virtual file systems – MNTFS, NAMEFS, OBJFS, SHAREFS, SPECFS, and SWAPFS
- Temporary file system (TMPFS)
- Loopback file system (LOFS)
- Process file system (PROCFS)

Note: UFS is a supported legacy file system, but it is not supported as a bootable root file system. The following are the default file systems in Oracle Linux:

- UNIX file system having blocks, inodes and directories – ext2 (no longer used)
- ext2 file system enhanced with journaling capabilities – ext3
- ext3 further enhanced – ext4
- CDROM file system – isofs
- Oracle Cluster File System – ocfs, ocfs2
- B-Tree File System – btrfs
- Proc file system acts as an interface to internal data structures in the kernel – procfs

The `root` Directory “/”

- UNIX uses a hierarchical file system structure, much like an upside-down tree, with root (`/`) at the base of the file system and all other directories spreading from there.
- The directories have specific purposes and generally hold the same types of information.
- The following directories exist on major UNIX versions which are POSIX compliant:



Note: On first login, you are taken to your home directory.

The / in Oracle Solaris 11 contains the following directories:

`bin`, `dev`, `export`, `lib`, `net`, `platform`, `rpool`, `tmp`, `boot`, `devices`, `home`, `media`, `nfs4`, `proc`, `sbin`, `usr`, `cdrom`, `etc`, `kernel`, `mnt`, `opt`, `root`, `system`, `var`

The / in Oracle Linux 6.2 contains the following directories as defined by the LSB:

`bin`, `cgroup`, `etc`, `lib`, `lost+found`, `misc`, `net`, `proc`, `sbin`, `srv`, `tmp`, `var`, `boot`, `dev`, `home`, `lib64`, `media`, `mnt`, `opt`, `root`, `selinux`, `sys`, `usr`

Files

- Everything in UNIX is considered to be a file, including physical devices, such as DVD-ROMs, USB devices, and external disks.
- In UNIX, there are three basic types of files:
  - Ordinary files (-)
  - Directories (d)
  - Special files
    - Symbolic links (l)
    - Character device (c)
    - Block device (b)
    - Socket (s)
    - Pipe (p)
    - And in Solaris a Door (D)

In UNIX, there are three basic types of files:

- Ordinary files: An ordinary file is a file that contains data, text, or program instructions.
- Directories: Directories store both special and ordinary files.
- Special files: Some files that provide access to hardware such as hard drives, CD-ROM drives, modems, and Ethernet adapters are called special device files. Other special files are similar to aliases or shortcuts and enable you to access a single file using different names.

UNIX & Linux OS Structure: Processes

- Every program you run in the Oracle Solaris or Oracle Linux 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 provides a graphical prompt for logging in to the OS.

ORACLE

UNIX & Linux OS Structure: The Shell

- The shell is primarily a command interpreter and serves as an interface between the kernel and the user.
- The shell accepts the commands that a user enters, interprets these commands, and passes them to the kernel, which executes the commands.
- In Oracle Solaris 11 and Oracle Linux, the default shell is the Bourne Again Shell (bash) `<bash>`.

Primary Shells

The three primary shells in Oracle Solaris 9 & 10 are:

- Bourne (`sh`) - Stephen Bourne (Bell Labs)
- C (`csh`) - Bill Joy (Berkeley)
- Korn (`ksh`) - David Korn (Bell Labs)

The single shell in Oracle Linux & Oracle Solaris 11 is:

- Bash (`bash`) - GNU Bourne Again Shell

The primary shells are:

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

Note: The Oracle Solaris and Oracle Linux OSs support these primary plus additional shells.

Additional Shells

The additional shells are:

- Z (zsh)
- TC (tcsh)
- and there are others

The additional shells are:

- **ZSH** – The Z shell most closely resembles the Korn shell, but it includes many other enhancements.
- **TCSH** – The TC shell is a completely compatible version of the C shell with additional enhancements.

Note: Examples or codes used in this course assume the use of the Bash shell.

Quiz

Select the primary shells supported by the Oracle Solaris OS:

a. TC shell
b. Z shell
c. Korn shell
d. C shell
e. Bash shell
f. Bourne shell

Answer: c, d, f

User Accounts

- A user account is a login account.
  - Regular users can log in and use the system, but cannot administer the system. (More about users on the next page).
- A group is a collection of users who can share files and other system resources.
  - For example, users working on the same project could be formed into a group.
  - Each group must have a name, a group identification (GID) number, and a list of usernames that belong to the group. A GID number identifies the group internally to the system.

Note

In Oracle Solaris 11, you create and manage users, groups, and roles by using command-line tools only. There is currently no GUI tool for performing these tasks.

In Oracle Linux 6, you create and manage users, and groups, using command-line tools like Oracle Solaris. Or by using `system-config-users` a GUI tool for performing those same tasks. Roles at this time are not supported.

Components of a User Account

Following are the main components of a user account:

- Username
- Password
- User identification (UID) number
- Group identification (GID) number
  - Primary group
  - Secondary group
- Comment (GECOS)
- User's home directory
- User's login shell
  - Prompts for bash shell

Following are the main components of a user account:

- **Username** – A unique name that a user enters to log in to a system. The username is also called the login name.
- **Password** – A combination of up to 256 letters, numbers, or special characters that a user enters with the login name to gain access to a system
- **User identification (UID) number** – A user account's unique numerical identification within the system
- **Group identification (GID) number** – A unique numerical identification of the group to which the user belongs. The two types of groups that a user can belong to are as follows:
  - **Primary group** – Specifies a group that the operating system assigns to files that are created by the user. Each user must belong to a primary group.
  - **Secondary groups** – Specifies one or more groups to which a user also belongs. Users can belong to up to 15 secondary groups.
- **Comment** – (GECOS) Information that identifies the user

- **User’s home directory** – A directory into which the user is placed after login. The home directory is the portion of a file system allocated to a user for storing private files.
- **User’s login shell** – The user’s work environment set up by the initialization files that are defined by the user’s login shell
    - **Prompts for bash shell** – The default bash shell prompt for a regular user is a dollar sign ($). For the root user, the default shell prompt is a pound sign (#).

Roles and Rights Profile

- Unlike the user account, which is a login account, a role is not a login account.
- A role is endowed with a rights profile, which is a collection of administrative capabilities.
  - For example, you want to perform some administrative functions. However, as a user, you cannot directly log in to the root role. The following steps enable you to switch roles:
    1. Open a terminal window.
    2. Log in with your username.
    3. Use the `su - root` command to assume the root role.

Note: A user can assume only the roles that are assigned to the user's login account. Currently roles are only supported on Oracle Solaris.

A rights profile can consist of authorizations, commands with security attributes, and other rights profiles. Rights profiles offer a convenient way to group security attributes. These profiles, listed in `/etc/security/prof_attr`, can be assigned by the root role to any account. The root role is assigned all privileges and all authorizations, so that it can perform all of the tasks, just as root can when root is a user.

User's Home Directory

- When you first log in, you are taken to your home directory.
- The home directory is where you create and organize all your files and subdirectories.
- To go to your home directory use the following commands:

```
$ cd
or
$ cd ~
```

- To go to someone else's home directory use the following command:

```
$ cd ~username
```


The home directory is the portion of a file system that is allocated to a user for storing private files. A home directory can be located either on the user's local system or on a remote file server. In either case, by convention, the home directory should be created as `/export/home/username` in Oracle Solaris and `/home/username` in Oracle Linux

To use a home directory from anywhere on the network, you should always refer to the home directory as `$HOME`, not as `/export/home/username` or `/home/username`. The latter is machine-specific. In addition, any symbolic links that are created in a user's home directory should use relative paths so that the links are valid no matter where the home directory is mounted.

Quiz

A role in Oracle Solaris is a login account just like a user account.

a. True

b. False

Answer: b

UNIX Variants

Several variants evolved out of UNIX. The following are some of the popularly used UNIX variants:

- SunOS (Predecessor of Solaris)
- GNU/Linux
- MacOS
- HP-UX (Hewlett Packard)
- AIX (IBM)
- FreeBSD
- Debian

Note

This course covers only Oracle Solaris and Oracle Linux.

Solaris

- Sun's original version of the UNIX OS was known as SunOS, based on BSD UNIX version 4.2.
- At that time, AT&T's version of the UNIX environment was known as System V.
- AT&T and Sun together created SVR4.
- SVR4 with multiprocessor capability became Oracle Solaris.

Linux

- Linux is a UNIX-like OS assembled and developed by GNU and Linus Torvalds under the open source software development and distribution model.
- The Linux OS kernel was first announced on August 25, 1991.
- Linux is packaged and distributed as a Linux distribution (distro) for desktop and server.
- Some popular Linux distributions include Red Hat, Debian (Ubuntu), OpenSUSE, Slackware and Fedora.

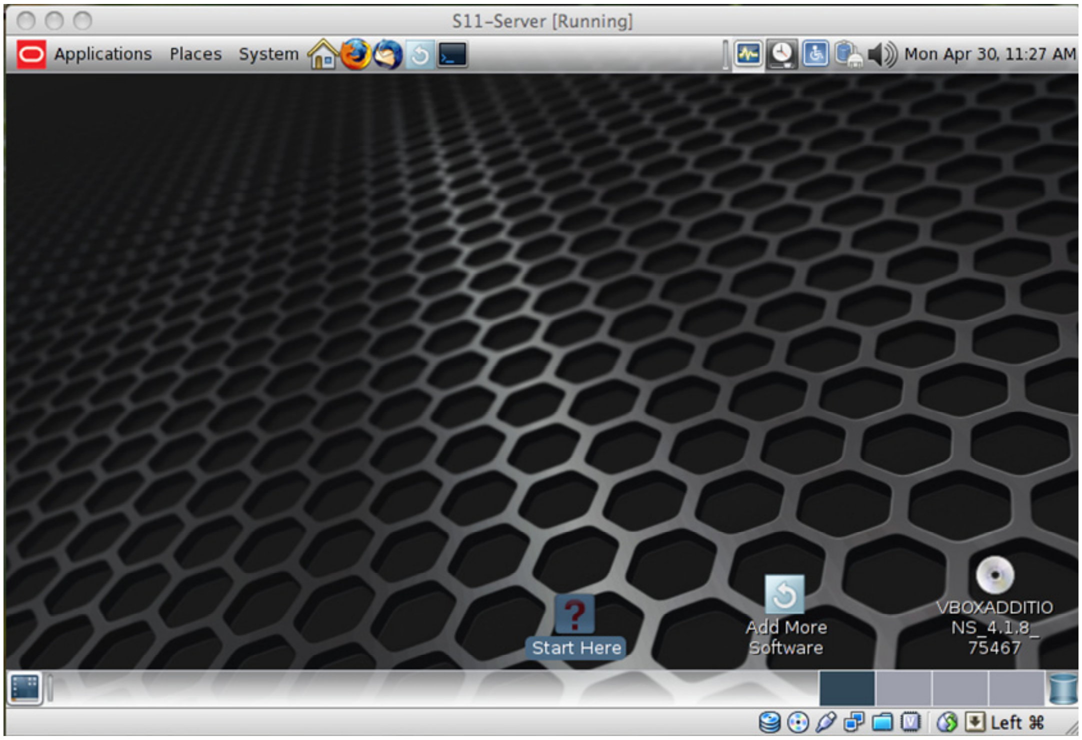
Oracle Linux, formerly known as Oracle Enterprise Linux, is a Linux distribution based on Red Hat Enterprise Linux, repackaged and distributed by Oracle and available under the GNU General Public License (GPL) since late 2010.

Oracle Linux can be freely downloaded through Oracle's E-delivery service, and can be deployed and distributed free of cost.

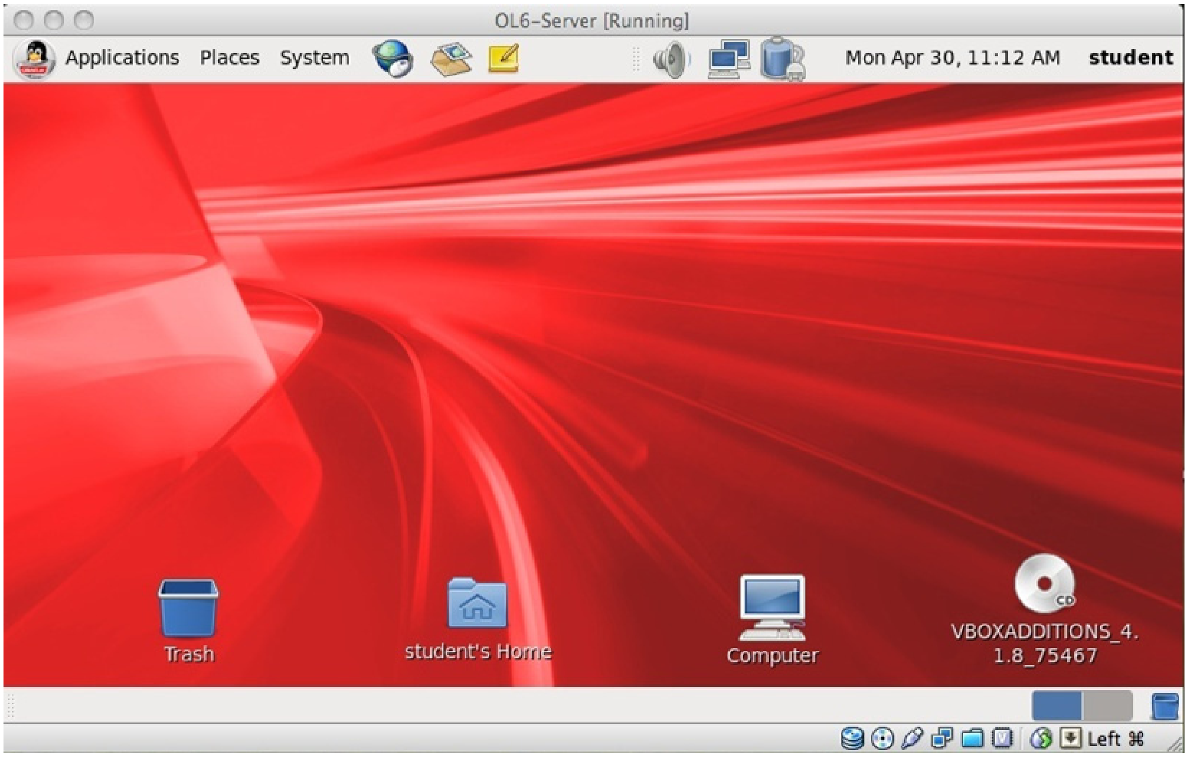
The Desktop Environment

- The desktop environment is a graphic user interface that allows you to perform a range of activities, such as:
  – Securing and selecting sessions
  – Adding and deleting workspaces
  – Changing backgrounds
  – Managing files
- The look and feel of Oracle Solaris and Oracle Linux desktop environments are displayed in the subsequent slides.

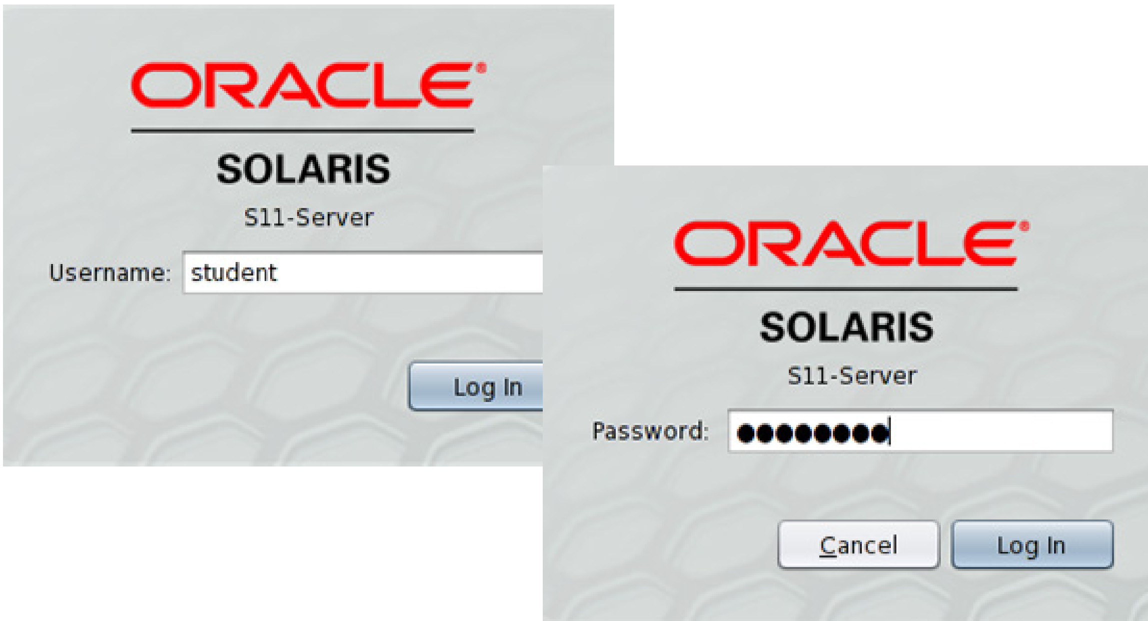
Oracle Solaris Desktop



Oracle Linux Desktop



Logging In to Oracle Solaris Using the Desktop Login Screen

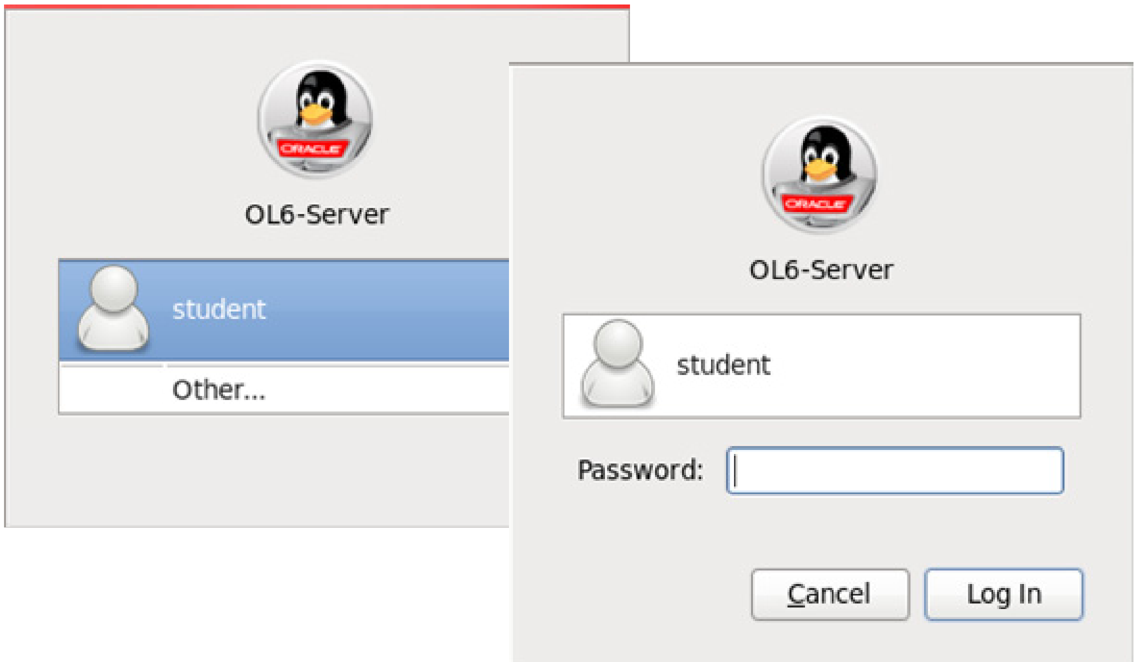


All users must follow a login process so that the system can recognize and authenticate the user. The desktop Login window as displayed on the slide enables you to log in to the system and use the desktop.

To log in to a desktop session, perform the following steps:

1. On the Login screen, enter your username.
2. Press Return or click **Log In**.
3. Next, enter your password.
4. Press Enter/Return or click **Log In**. Retry, if the login attempt fails.

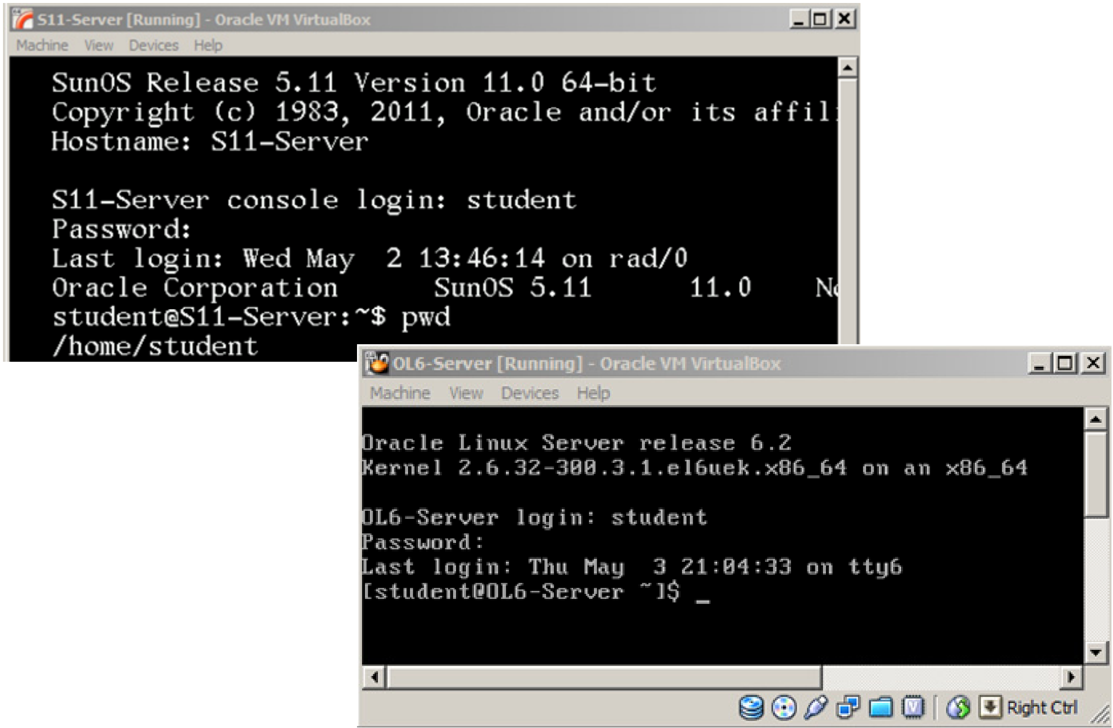
Logging In to Oracle Linux Using the Desktop Login Screen



To log in to an Oracle Linux desktop session, perform the following steps:

1. On the Login screen, enter your username.
2. Press Return.
3. Next, enter your password.
4. Press Enter/Return or click Log In. Retry, if the login attempt fails.

Logging In Using the Command-line Option



You also have the option of logging in using the command line. Pressing the CTRL+ALT+F1 or CTRL+ALT+F6 keys on the login window in Oracle Solaris and Oracle Linux, respectively, switches to the command line mode. On the console prompt, you can log in with your user credentials.

Pressing the CTRL+ALT+F7 or CTRL+ALT+F1 keys in Oracle Solaris and Oracle Linux, respectively, reverts to the desktop window.

Logging Out

- Depending on the interface, you use different commands or steps to log out.
- To log out of the graphical user interface:
  1. On the desktop window, click **System**.
  2. Next, click **Log Out <*username*>**. A logout confirmation window appears.
  3. Click **OK** or press **Enter** to log out.
- To log out of the command line interface:
  Type `exit`. This causes your shell to exit, or stop running.

Note: Some shells, depending on your configuration, also log you out if you type the end-of-file (eof) character, **Ctrl+D**.

When you are done using the system, you should log out. This prevents other people from accidentally or intentionally getting access to your files.

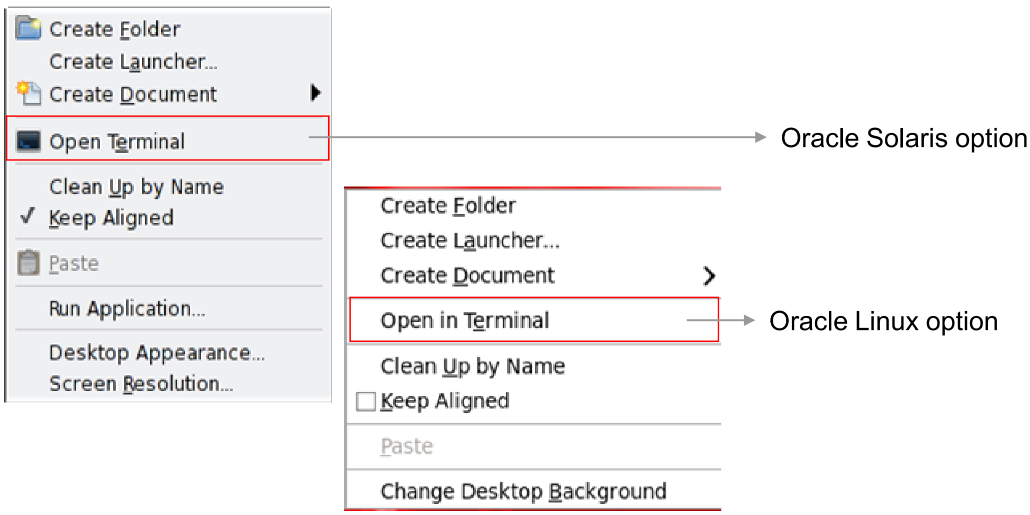
Both Oracle Solaris and Oracle Linux follow the same procedure to log out.

Lesson Agenda

- Describing the UNIX & Linux operating system
- Executing commands from the command line

Executing Commands from the Command Line

- You can use system commands on the command line to instruct the system to perform specific tasks.
- The commands are received into a terminal window.



To open a terminal window, perform the following steps:

1. Move the cursor to an open space on the desktop.
2. Right-click the desktop. A pop-up menu appears.
3. Click Open Terminal or Open in Terminal in Oracle Solaris and Oracle Linux, respectively. A terminal window appears with a shell prompt.

Note: UNIX commands are case-sensitive.

Command-Line Syntax

- The command syntax is the structure and order of the command components: name, options, and arguments.
- Command-line commands can exist with or without options and arguments.
- You can change the behavior of commands by using a combination of options and arguments.

Item	Description
`Command`	Specifies what the system does (an executable)
`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 table in the slide describes the components of a command.

Note: Most Linux commands can sometimes use a double hyphen (--) for their command-line switches.

Using UNIX Commands

- To display the operating system information, enter:

```
$ uname
SunOS
```

(*output will vary by OS*)

- To display the date and time, enter:

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

- To display the calendar, enter:

```
$ cal
```

- To clear the terminal window, enter:

```
$ clear
```


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.

Using Commands with Options

- Adding options to a command alters the information displayed.
- You can use more than one option with a command.
- You can also list multiple options separately or they can be combined after a dash (-).
- Use of a dash (-) preceding an option is command specific. Also, options are command specific.

Note: For additional information and proper usage of options, check the appropriate man page for the command.

Using Commands with Options

- The given example shows the `uname` command on Oracle Solaris with two options:
  - The `-i` option displays the name of the hardware platform.
  - The `-n` option prints the host name of the local system.

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

- The following example shows the `uname` command on Oracle Solaris with two combined options:

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


UNIX and Linux Essentials 2 - 39

Using Commands with Options

- The given example shows the `uname` command on Linux with two options:
  - The `-i` option displays the name of the hardware platform.
  - The `-n` option prints the host name of the local system.

```
$ uname -i
i386
$ uname -n
host01.example.com
```

- The following example shows the `uname` command on Linux with two combined options.

```
$ uname -rs
Linux 3.8.13-26.1.1.el6uek.x86_64
```


UNIX and Linux Essentials 2 - 40

```
[vncuser@edbxr64p0 ~]$ uname --help
Usage: uname [OPTION]...
Print certain system information.  With no OPTION, same as -s.

  -a, --all                print all information, in the following order,
                             except omit -p and -i if unknown:
  -s, --kernel-name        print the kernel name
  -n, --nodename           print the network node hostname
  -r, --kernel-release     print the kernel release
  -v, --kernel-version     print the kernel version
  -m, --machine            print the machine hardware name
  -p, --processor          print the processor type or "unknown"
  -i, --hardware-platform  print the hardware platform or "unknown"
  -o, --operating-system   print the operating system
      --help     display this help and exit
      --version  output version information and exit

Report bugs to <bug-coreutils@gnu.org>.
```

Using Commands with Arguments

- Arguments enable you to additionally define the output from a command.
- The following example shows the `cal` command with two arguments, `12` and `2009`.
  - The first argument, `12`, specifies the month.
  - The second argument, `2009`, specifies the year.

```
$ 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 without an option, with an option, with an argument, and with an option and argument together.

```
$ ls
dante     dir3          file.2          file3      greetings
dante_1  dir4          file.3          file4       myvars
$ 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
(output truncated)
$ ls dante
dante
$ ls -l dante
-rw-r--r-- 1 student class 1319 Feb 6 09:25 dante
```


In the examples, the `ls` command lists the files in a directory. The `-l` option provides additional information about the files. The file name argument specifies the file to be viewed.

Using Multiple Commands on the Command Line

- You can enter multiple commands on a single command line by using a semicolon (`;`) to separate each command.

```
$ command [-option] [argument]; command [-option] [argument]
```

- The shell recognizes the semicolon (`;`) as a command separator.
- The following example shows two commands separated by a semicolon.

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

- The shell executes each command from left to right when you press Enter/Return.

The following example shows three commands separated by semicolons. 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
```

Quiz

Given the `uname -a` command, which type of UNIX command component does `-a` represent?

a. Option
b. Parameter
c. Argument

Answer: a

Using the Man Pages

- The online Reference Manual (man) pages provide detailed descriptions and usage of the commands.
- You can use the `man` command to display the man page entry that explains a given command.

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

Displaying the Man Pages

For example, display the man pages for the `uname` command using the `man` 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 system
on the standard output. When options are specified, symbols
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 Through the Man Pages

The following table lists the keyboard commands for scrolling through the man pages.

Keyboard Command	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 (regular expression)
`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

There are two ways to search for information in the man pages:

- Searching by section
- Searching by keyword

Section	Solaris Description	Section	Linux Description
1	User Commands	1	User Commands
1M	System Administration		
2	System Calls	2	System Calls
3	C Library Functions	3	C Library Functions
4	File Formats	4	Devices and Special files
5	Miscellaneous	5	File Formats
		7	Miscellaneous
		8	System Administration

The section numbers and meaning are different between Oracle Solaris and Oracle Linux.

Searching the Man Pages: By Section

- The online man page entries are organized into sections based on the type or usage of the command or file.
  - For example, Section 1 contains user commands, and Section 4 contains information about various file formats.
- To look up a specific section of the man page, use the `man` command with the `-s` option, followed by the section number, and the command or file name.

```
$ man -s number command
or
$ man -s number filename
```


Note

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.

For example, 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... (output truncated
```

Searching the Man Pages: By Keyword

- When you are unsure of the name of a command, you can use the `man` command with the `-k` option and a keyword to search for matching man page entries.

```
$ man -k keyword
```

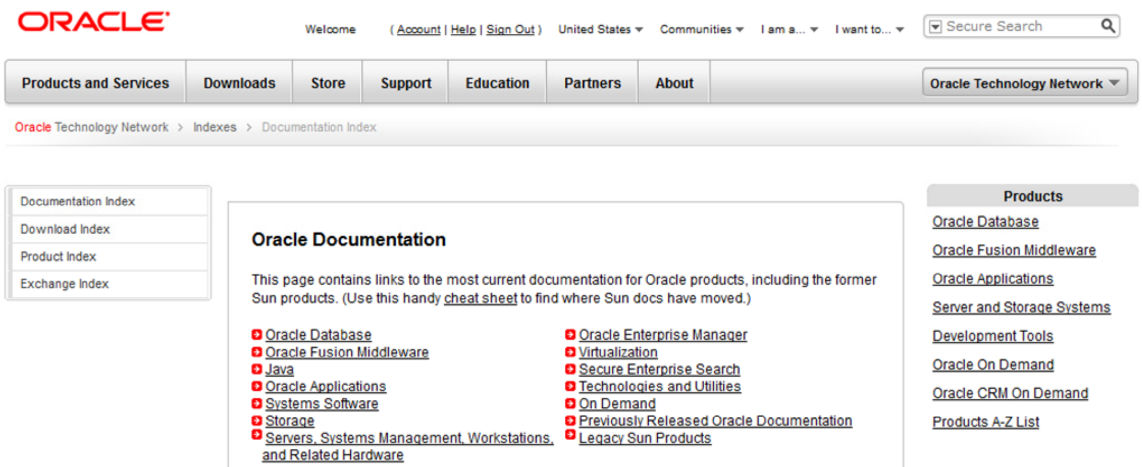
- The `man` command output provides a list of commands and descriptions that contain the specified keyword.

For example, using the `man` command, view commands containing the `calendar` keyword.

```
$ man -k calendar
cal            cal (1)          -Displays 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
```

Accessing Online Product Documentation

For additional information about Oracle products, you can access the Oracle Technical Network (OTN) Documentation website.



ORACLE

You can search the OTN website, http://www.oracle.com/technetwork/indexes/documentation/index.html, by subject, collection title, product category, and keyword.

Quiz

Which of the following `man` command options displays a specific section of the man page?

a. `-h`

b. `-q`

c. `-s`

d. `-n`

Answer: c

Summary

In this lesson, you should have learned how to:

- Describe the UNIX & Linux operating system
- Execute commands from the command line

Practice 2 Overview: Introduction to the UNIX Environments

This practice covers the following topics:

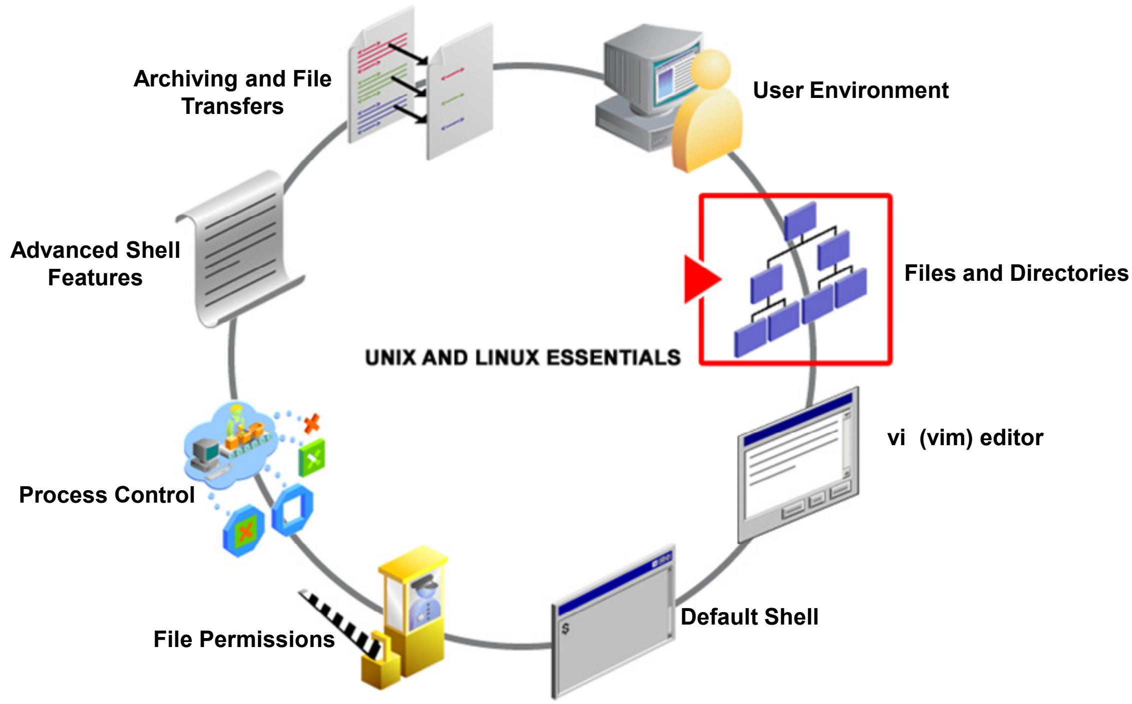
- Logging in to the system
- Changing your user login password
- Displaying system information using the command line
- Using the man pages
- Logging out of the system

You will find the tasks for Practice 2 in your Activity Guide.

3

Working with Files and Directories

Workflow Orientation



The lesson titled "Introduction to the UNIX environments" introduced the operating systems and the dynamics of the user environment. This lesson explains how to work with files and directories. This includes tasks, such as locating your position in the directory structure, viewing file contents, copying and moving files and directories, creating and removing files and directories, and searching for files and directories.

Objectives

After completing this lesson, you should be able to:

- Determine your location in the directory structure
- View file content
- Copy and move files and directories
- Create and remove files and directories
- Search for files and directories

Lesson Agenda

- Determining your location in the directory structure
- Viewing file content
- Copying and moving files and directories
- Creating and removing files and directories
- Searching for files and directories

Viewing Directories

- A directory is a list of references to objects, which can include files, subdirectories, and symbolic links.
- Each reference consists of two components:
  – A name: The name of the object is used to identify and access the object.
  – A number: The number specifies the inode in which metadata about the object is stored.
- You can use various commands to display the current directory, view content of a directory, and change directories.

An inode is a list of information relating to a particular object, such as a file, directory, or symbolic link. The metadata 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 full or absolute path name of the current working directory.

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

Note: The `pwd` command works the same in both Oracle Solaris and Linux, although the directory structures themselves may be different.

Displaying the Directory Content

The `ls` command displays the content of a directory. The syntax for the `ls` command is:

```
$ ls [-option] … [filename]
```


- To list the files and directories in the current directory (`/export/home/student`), enter the `ls` command without arguments.

```
$ ls
dante    dir3    file.2    file3    greetings
```

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

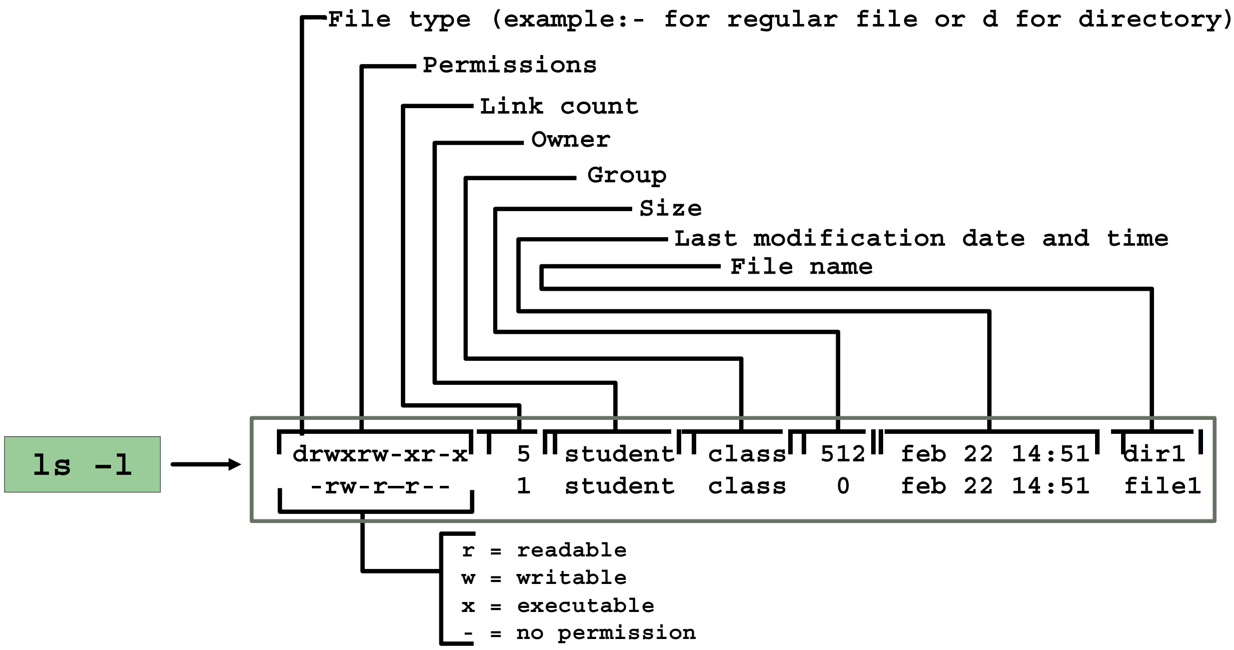
```
$ ls dir1
coffees fruit trees
```

- To display the content 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 the Directory Content With Options

The `ls -l` command displays a long listing of file information.



The following is a brief explanation of the parts of the long list displayed in the slide:

- The first number is the file type.
- The second nine places indicate the file permissions: `r` means readable, `w` means writable, `x` means executable, and the – means no permission.
- The third section (one number) is the link count.
- The fourth section is the owner (student).
- The fifth section is the group (class).
- The sixth section is the file size.
- The seventh section is the date.
- The eighth section is the file name.

Displaying the Directory Content With Options

- The `ls -a` command lists all files in a directory, including hidden files.

```
$ 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
```

- The `ls –i` command list the file's inode number and filename.

```
$ ls –i dante
12783  dante
```


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 filenames that start with a period (`.`) these are hidden files.

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

Displaying the Directory Content With Options

- The `ls -ld` command displays detailed information about a directory without showing its content.

```
$ ls -ld directory_name
```

- The `ls -R` command displays the content of a directory and all its subdirectories. This type of list is known as a recursive list.

```
$ ls -R directory_name
```

- One of the difference between Oracle Solaris and Linux commands is that Solaris uses "`-r`" for recursive operations and Linux uses "`-R`"

- For example, 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
```

- For example, to view a recursive list of the content 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:
```

Quiz

What is the function of the `ls -a` command?

a. Displays the content of a directory and all subdirectories of the directory
b. Displays detailed information for the directory only, not its content
c. Displays detailed information about the content of a directory, including hidden files
d. Displays all the files in a directory, including hidden files

Answer: d

Displaying File Types

- The `ls -F` command or the `file` command displays the file types.

```
$ ls -F
dante   dir3/ file.2 file3 greetings
dante_1 dir4/ file.3 file4 myvars
```

- The following table shows the symbols or indicators used with the `ls -F` command output.

Indicator	File Type
*	Executable
/	Directory
=	Socket
@	Symbolic link
|	First In First Out (FIFO)

Knowing the file type may help you decide the command or program to use for reading the file.

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

Displaying File Types

- The `file` command also helps determine certain file types. The syntax for the `file` command is:

```
$ file filename
```

- 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
```


The output from the `file` command is 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 or 64-bit executable and extensible linking format (ELF) code files and other dynamically linked executable files. Executable files are commands or programs.

Changing Directories

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

```
$ cd directory
```

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

For example, 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
```

Changing Directories using Relative or Absolute Pathname

- You can use either a relative 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.
- The table describes the relative pathname.

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

For example, to change directories using a relative path name, enter the `cd` command with the path name 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
```

For instance, to change directories using an absolute path name, enter the `cd` command with the complete path name from the root (`/`) directory.

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

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.
- 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 in Solaris would be `/export/home/student` or in Linux `/home/student`.

Note:

You can configure systems to use the `/home` directory, instead of the `/export/home` directory, as the default home directory.

Returning to Your Home Directory

- You can return to your home directory by using one of the two methods:
  - Use the `cd` command without arguments.

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

  - Use 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 username. The tilde (~) character is an abbreviation that equates to the absolute path name 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
```

Quiz

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

a. True

b. False

Answer: a

Lesson Agenda

- Determining your location in the directory structure
- Viewing file content
- Copying and moving files and directories
- Creating and removing files and directories
- Searching for files and directories

Viewing Files

- There are several commands that display information about a file in the read-only format.
- The file-viewing commands include the following:
  - `cat`
  - `more`
  - `less`
  - `tail`
  - `head`
  - `wc`

Viewing Files: `cat` Command

- The `cat` command concatenates the content of one or more text files and displays the contents on the screen without pausing.

```
$ cat filename
```

```
$ 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...
```

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 type.

Caution

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: `more` Command

- The `more` command displays the content of a text file one screen at a time.

```
$ more filename
```

- The `--More--(n%)` message appears at the bottom of each screen, where `n%` is the percentage of the file that has been displayed.
- When the entire file has been displayed, the shell prompt appears.

Viewing File Content: `more` Command

When the `--More--(n%)` prompt appears at the bottom of the screen, you can use the keys described in the table to scroll through the file.

Keyboard	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
`/`*`pattern`*	Searches forward for *pattern* (regular expression)
`n`	Finds the next occurrence of *pattern*
`q`	Quits and returns to the shell prompt

For example, 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 and modest means and social standing.
--More--(90%)

Viewing File Content: `less` Command

The GNU project created `less` as a response to `more`, but with more features/actions. The table on the previous page details not only `more` options but some of the `less` options. For a more through list of features a `less -?` will yield greater details.

Here is a snippet from the `less` man pages.

```
SYNOPSIS
     less -?
     less --help
     less -V
     less --version
     less [-[+]aABcCdeEfFgGiIJKLmMnNqQrRsSuUVwWX~]
          [-b space] [-h lines] [-j line] [-k keyfile]
          [-{oO} logfile] [-p pattern] [-P prompt] [-t tag]
          [-T tagsfile] [-x tab,...] [-y lines] [-[z] lines]
          [-# shift] [+[+]cmd] [--] [filename]...
```

Viewing File Content: `head` Command

- The `head` command displays the first 10 lines of a file.

```
$ head [-n] filename
```

- You can change the number of lines displayed by using the `-n` option.
- For example, to display the first **five** lines of the `/usr/dict/words` file, enter the `head` command with the `-n` option set to `5`.

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

ORACLE

Viewing File Content: `tail` Command

- The `tail` command displays the last 10 lines of a file.

```
$ tail [-option] filename
```

- You can change the number of lines displayed by 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.
- Possibly one of the more popular options is `–f` which allows one to *follow* the data that is being appended to the end of the file.

For example, to display the last four lines of the `/usr/dict/words` file, enter the `tail` command with the `-n` option set to `4`.

```
$ tail -4 /usr/dict/words
zounds
z's
zucchini
Zurich
```

For example, 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
```

Viewing File Content: `wc` Command

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

```
$ wc [-option] filename
```

- You can use the following options with the `wc` command.

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

ORACLE

Note:

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

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

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

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

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

Quiz

The default number of lines displayed by the `head` command is:

a. 5
b. 10
c. 15
d. 20

Answer: b

Lesson Agenda

- Determining your location in the directory structure
- Viewing file content
- Copying and moving files and directories
- Creating and removing files and directories
- Searching for files and directories

Copying Files and Directories

- The `cp` command copies single or multiple files and directories.

`$ cp [-option] source(s) target,` where `source(s)` is a file and `target` can be a file or directory.

  – For example, 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.

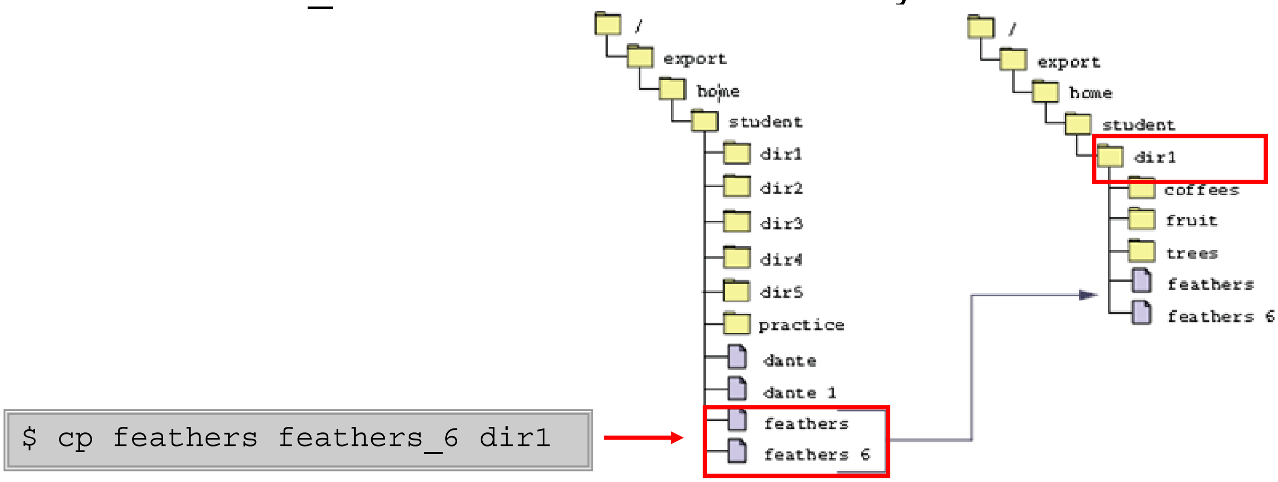
```
$ cp filename newfilename
```


For example, 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.

```
$ 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
```

Copying Multiple Files

- To copy multiple files to a different directory, use the `cp` command with multiple file names for the source and a single directory name for the target.
- This figure represents copying the `feathers` and `feathers_6` files to the `dir1` subdirectory.



For example, to copy the `feathers` and `feathers_6` files 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
```

Copying Files: cp Command Options

You can use the cp command with options and modify the functions of the command.

Option	Description
`-i`	Prevents you from accidentally overwriting existing files or directories
`-r (Oracle Solaris)` `-R (Linux)`	Recursive, includes the contents of a directory, and the contents of all subdirectories, when you copy a directory

Copying Files Recursively: `-r` or `-R` Option

- The `-r` option recursively copies a directory.
- If the target directory does not exist, the `cp -r` command creates a new directory with that name.
- If the target directory exists, the `cp -r` command creates a new subdirectory with that name, below the destination directory.
- The source option is one or more directory names. The target option is a single directory name.

For example, 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
```

Preventing Copy Overrides: `-i` Option

- The `-i` option prevents existing files from being overwritten with new files.
- When using the `-i` option, the system prompts for a `yes/no` response from you before overwriting existing files.
- For example, while copying the `feathers` file to `feathers_6` file, the overwrite prompt appears.

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

Moving and Renaming Files and Directories

- The `mv` command helps move and rename files and directories within the directory hierarchy.

`$ mv [-option] source target,` where `source(s)` is the old file or directory name and `target` is the new file or directory name.

- The `mv` command does not affect the content of the files or directories being moved or renamed.
- The moved/renamed file maintains its original inode number.

Caution: `mv` is a destructive command if not used with the correct option(s).

Note:

Unlike the `cp` command, the `mv` command moves the respective file or directory, and the original no longer exists.

Moving a File to Another Directory

For example, use the `mv` command to move the `brands` file from the `coffees` directory into the `student` directory.

```
$ 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
```


The `-i` option prompts you for confirmation to prevent you from overwriting existing files by the new files.

`$ mv -i` *`source target`*

- A `yes` response permits the `mv` command to overwrite the existing files.
- A `no` response prevents the `mv` command from overwriting the existing files.

Moving a Directory and Its Content

- You can also use the `mv` command to move a directory and its content to a different directory.
- For example, use the `mv` command to move the `practice` directory and its content into a new directory named `letters`.

```
$ 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
$ 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
```


Note:

When you move a single directory to a target directory that does not exist, you actually rename 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 Files and Directories

- The `mv` command is also used for renaming existing files and directories.
- For example, use the `mv` command to rename the `dante` file to `dantenew` in the current directory.

```
$ pwd
/export/home/student
$ mv dante dantenew
$ ls
dante_1         dir2    feathers        file.3  file4   myvars
dantenew        dir3    feathers_6      file1   fruit   practice
```

Quiz

In your home directory, you created a directory called `newdir` as a placeholder for a variety of files. You notice that the directory now contains monthly reports. What would be the proper syntax to rename the `newdir` directory to `monthly_reports`? (Note: You are not in your home directory.)

a. `mv ~/newdir ~/monthly_reports`

b. `mv newdir monthly_reports`

c. `mkdir ~/monthly_reports`

Answer: a

Lesson Agenda

- Determining your location in the directory structure
- Viewing file content
- Copying and moving files and directories
- Creating and removing files and directories
- Searching for files and directories

Creating Files

- The `touch` command creates a new empty file.

```
$ touch filename
```

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

ORACLE

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
```

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

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

Creating Directories

- The `mkdir` command creates new directories.

```
$ mkdir directory_name
and
$ mkdir -p directory_names
```

- Include the `-p` option if the directory name includes a path name and the intermediate directories will also be created.
- You can use absolute or relative pathnames on the command line when creating new directories.

Note:

The command used with the `-p` option creates all of the nonexisting parent directories that do not yet exist in the path to the new directory.

Creating Directories

- For example, create a new directory, named `Reports`, within the `student` directory.

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

- For example, create a `Weekly` directory in the `Reports` directory.

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


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
$ mkdir -p newdir/empty_directory
$ ls newdir
empty_directory
```

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

- You can permanently remove files from the directory hierarchy with the `rm` command.

```
$ rm [-option] filename
```

- The `rm` command is a destructive command.
- The table describes the some of the options that you can use with the `rm` command when removing files and directories.

Option	Description
`-r or -R`	Recursive, 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 `-r` or `-R` option allows you to remove directories that contain files and subdirectories.
- The `-i` option prompts you for confirmation before removing any file.
  - A `yes` response completes the removal of the file.
  - A `no` response aborts the removal of the file.

Removing Files

- For example, remove the file named `projection` from the `letters` directory.

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

- For example, using `-i` remove the contents of a directory.

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


Note:

The asterisk (*) symbol used in the second example is a wildcard character, sometimes called a "glob".

Removing Directories

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

```
$ rm [-option] directory
```

- For example, remove the `letters` directory and its content by using the `rm -r` command.

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


Note:

If you do not use the `-r` or `-R` option with the `rm` command while removing directories, the following error message appears: `rm: directoryname: is a directory.`

To interactively remove a directory and its contents, use the `-i` option along with the `rm -r` command. For example, create a new directory called `rmtest` in your `/export/home/student` directory by using the `rm -ir` command.

```
$ 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
```

Removing Directories

- The `rmdir` command removes empty directories.

```
$ rmdir directory
```

- 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 in which you are currently working in, you must first change to its parent directory (`cd ..`).

Note:

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

For example, remove `empty_directory` using the `rmdir` command.

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

Symbolic Links

- A symbolic link (sometimes called a symlink or a soft 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 path name.
- Symbolic links can link files and directories located across different file systems.
- 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.

```
$ ln -s source_file target_file
```

- The file name for the symbolic link appears in the directory in which it was created.
- You can use either relative or absolute pathnames when creating a symbolic link for a file.

In the syntax displayed in the slide, 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, if the `source_file` does not exist, a symbolic link that points to a non-existing file is created.

Creating Symbolic Links

- For example, use the `ln -s` command to create a symbolic link file named `dante_link` to the `dante` file.

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

- When using the `ls -F` command to display a list of files and directories, the symbolic links are identified with an @ symbol.

```
$ 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/
```


Note:

The `@` symbol that follows the file name indicates that the file is a symbolic link. The output of the `ls -F` command in the slide lists the `dante_link` file as a symbolic link.

To further view the content of the `dante_link` file, use the `cat` command.

```
$ 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 path name to which a symbolic link is pointing to, enter the `ls -l` command with the symbolic link file name.

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

Removing Symbolic Links

- You can use the `rm` command to remove a symbolic link file, just as you would remove a standard file.
- For example, remove the `dante_link` symbolic link file using the `rm` command.

```
$ ls -l dante_link
lrwxrwxrwx 1 student class ... 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
```

Quiz

Which of the following command creates two new directories with the second directory as a subdirectory of the first?

a. `mkdir -i dir dir2`

b. `mkdir -p dir1/dir2`

c. `mkdir -r dir1/dir2`

Answer: b

Quiz

The `rm -r` command removes non-empty directories.

a. True
b. False

Answer: a

Lesson Agenda

- Determining your location in the directory structure
- Viewing file content
- Copying and moving files and directories
- Creating and removing files and directories
- Searching for files and directories

Regular Expressions

- Sometimes when searching in UINX & Linux a simple text string is all you need. However, at other times when the pattern is more complex one needs *wild cards*.
- Regular Expressions (sometimes called regex or regexp) is way to build those *wild cards* patterns.
- Virtually, any userspace program that performs searches uses regex patterns,
  - Example: `grep`, `more`, `less`, `vi`, `vim`, `emacs`, `sed`, `awk`, `tcl`, `ls`, `find`, and many more.

Regular Expressions Wild Cards

- Wild cards (metacharacters)
  - `*` asterisk (glob) matches zero or more characters
  - `?` question mark matches zero or a single character
  - `.` period matches a single character
  - `^` caret at the beginning of the line
  - `$` dollar at the end of the line
  - `[ ]` square brackets--character class the symbols inside the brackets match one character position
  - `'` single quote (apostrophe) tells the shell to ignore any enclosed metacharacters
  - `"` double quotes enclose a space
  - \ back slash escape’s the character

Creating `patterns` using Regular Expressions

- The `ls` command will list all of the files in the current directory.

```
$ ls
file1    file11    file23    file3    file32    file4    file45
```

- But what if one only wanted to list `file2` and `file4` they could build a regex character class `file[24]`

```
$ ls file[24]
file2    file4
```

- Second example: what if one wanted all files that end in 2 numerals, but the last numeral has to be a 3. An option could be `file[0-9][3]` or `file?[3]` and there are more regex solutions.

Searching Files and Directories

- The `find` command searches for files that match the 'filename pattern', starting in the location specified by path.

```
$ find [path] -name 'filename pattern'
```

- The `find` command also searches for subdirectories recursively.
- When a file matches the expression specified in the `find` command, its path is printed.

Searching Files and Directories: `find` Command

- For example, search for a file named `myscript.htm` in the current directory and any subdirectory.

```
$ find -name myscript.htm
```

- For example, search for any file named `mypage` on the root directory (`/`) and all subdirectories from the root.

```
$ find / -name mypage
```

- To find all the files that start with `mess` located in the /var directory, add an asterisk (*) or glob.

```
$ find /var -name mess*
```

Searching Files and Directories: Linux also has a `locate` Command

- To search using `locate`.

```
$ locate [-option] pattern
```

- For example, search for any file named `myscript.htm`.

```
$ locate myscript.htm
```

- The `locate` command works much like the `grep` command (covered in the following pages), to `locate` any files that contain the pattern “message”.

```
$ locate message
```

Searching Within Files: `grep` Command

- The `grep` command allows you to search for a specified pattern in one or more files, printing any lines that contain the specified pattern.

```
$ grep [-option] pattern [file(s)]
```

- For example, to search for the occurrence of "first" in a file called "Hello", enter the following command:

```
$ grep first Hello
This is my first file in vi Editor
```


In addition, two variant programs `egrep` and `fgrep` are available. While `egrep` is the same as `grep -E`, `fgrep` is the same as `grep -F`.

- `grep -E`: Interprets `PATTERN` as an extended regular expression
- `grep -F`: Interprets `PATTERN` as a list of fixed strings, separated by new lines, any of which is to be matched

Searching Within Files: `grep` Command on Linux

- Sometimes it is useful to be able to highlight what was being searched for. The `grep` command on Linux has an option `--color` which hightlights using "**red**" what you were searching for in the output.
- To use the previous example, to search for the occurrence of "first" in a file called "Hello" highlighting what was found:

```
$ grep first --color Hello
This is my first file in vi Editor
```

Quiz

Which of the following copy commands results in an error message?

a. `cp directory1 directory2`

b. `cp -r directory1 directory2`

c. `cp -ri directory1 directory2`

Answer: a

Summary

In this lesson, you should have learned how to:

- Determine your location in the directory structure
- View file content
- Copy and move files and directories
- Create and remove files and directories
- Search for files and directories

Practice 3 Overview: Working with Files and Directories

This practice covers the following topics:

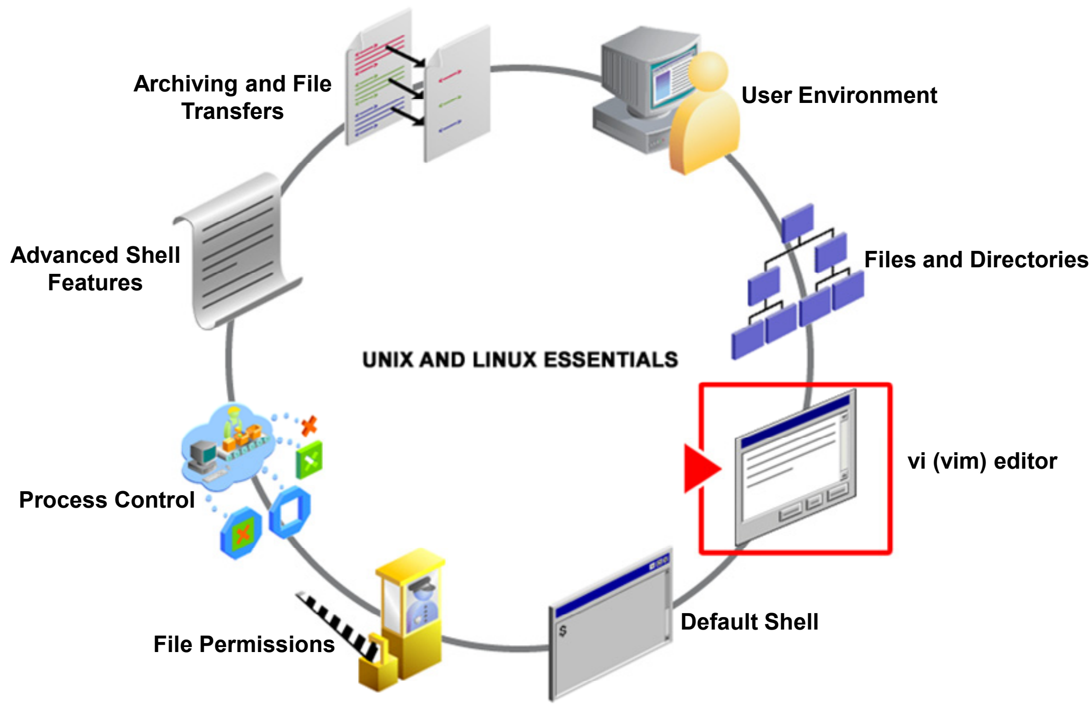
- Displaying user information
- Displaying directory contents
- Displaying file types
- Changing directories
- Accessing files
- Copying files and directories
- Moving files and directories
- Creating files and directories
- Removing files and directories
- Using symbolic links
- Searching files and directories

You will find the tasks for Practice 3 in your Activity Guide.

4

Using the vi (vim) Editor

Workflow Orientation



The lesson titled "Working With Files and Directories" explained how to work with files and directories, in terms of copying, moving, renaming, deleting, and creating files and directories. This lesson introduces the vi editor and explains how to perform basic editing functions on files using the `vi` commands.

Objectives

After completing this lesson, you should be able to:

- Access the vi editor
- Modify files with the vi editor

Agenda

- Accessing the vi editor
- Modifying files with the vi editor

Introduction to the `vi` Editor

- The vi editor is a command-line, interactive editor that you can use to create and modify text files.
  - 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.
- In Oracle Solaris 11 and Oracle Linux 6.2, vi improved (vim) is the default editor.
  - The vim editor is an enhanced version of the vi editor and is accessed via an `alias “vi”`.

Note

This lesson primarily dwells on the vi editor. For additional information about the Vim editor, use the `vimtutor` command on the command prompt to go through the built-in tutorial for beginners. You can also access the Vim Users' Manual that details the features of Vim. This too is available from within Vim, or can be found online.

Accessing the `vi` Editor

- To create, edit, and view files in the vi editor, use the `vi` command.
- The `vi` command includes the following three syntaxes:

```
$ vi
$ vi filename
$ vi [-options] filename
```


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

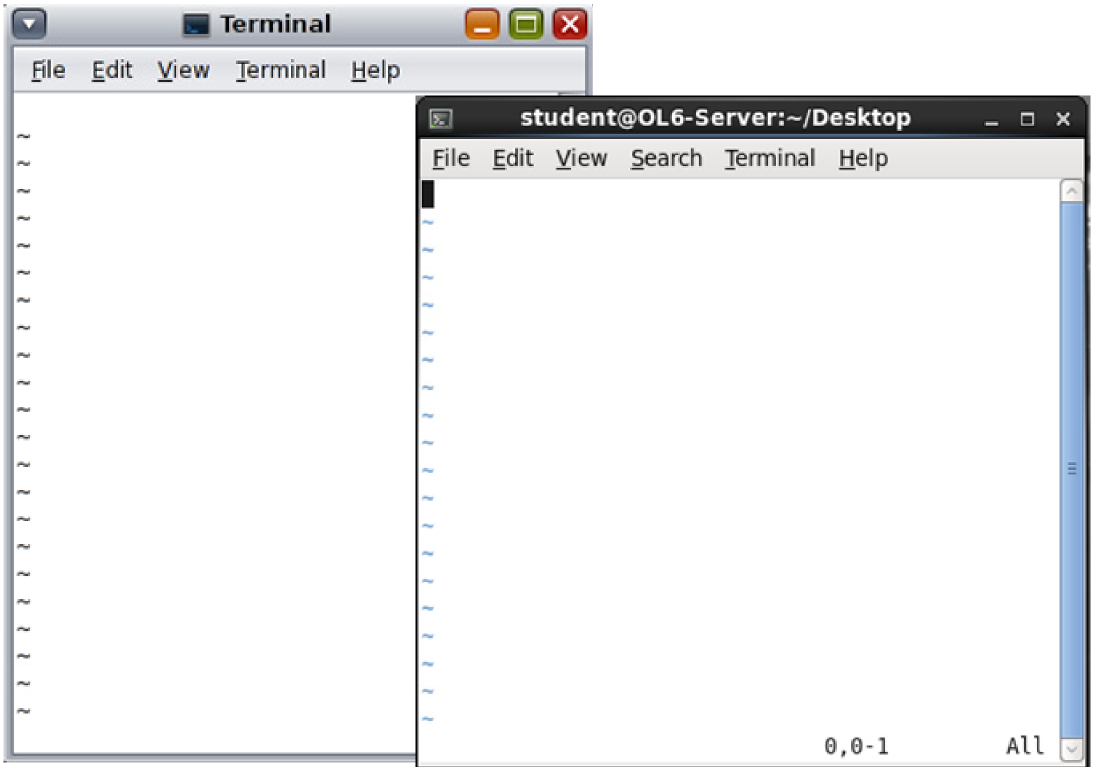
```
$ vi -r filename
```

The file opens so that you can edit it. You can then save the file and exit the vi editor, by using the following command:

```
$ vi -R filename
```

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

The vi Editor



The initial display of the editor in a terminal window is a blank window filled with tildes and a blinking cursor in the top left corner. The screenshots displayed in the slide are that of the vi editor in Oracle Solaris and the Vim editor in Oracle Linux.

The vi Editor Modes

The vi editor provides two modes of operation:

- Command
- Insert (and replace)

Note: The vi editor is essentially the visual interface (vi) to the ex editor, which in turn is an extended version of the ed editor.

The vi editor provides two modes of operation:

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.

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

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

You can also invoke the input mode to insert text into a file by pressing one of the following uppercase 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

Switching Between Modes

- The default mode for the vi editor is the command mode.
- To switch to the insert mode, press `i`, `o`, `a` or `R`.
- To return to the command mode, press the **Esc** key.
- In the command mode, you can save (write) the file and quit the vi editor, and return to the shell prompt.

Agenda

- Accessing the vi editor
- Modifying files with the vi editor

Viewing Files in the Read-Only Mode

- The `view` command enables you to view files in the read-only mode.

```
$ view filename
```

- The `view` command invokes the vi editor in the read-only option, which means you cannot save changes to the file.
- For example, to view the `dante` file in the read-only mode, enter the following command:

```
$ view dante
```

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

Moving the Cursor Within the vi Editor

The table shows the key sequences that move the cursor.

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

The table in the slide shows the key sequences that move the cursor in the vi editor.

Moving the Cursor Within the vi Editor

Key Sequence	Cursor Movement
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*
nG	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

The table in the slide shows the key sequences that move the cursor in the vi editor.

Inserting and Appending Text

The table describes the commands to insert and append text to a new or existing file by using 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 insert commands. Also, most of the insert commands and cursor movements can be preceded by a number to repeat the command that many times.

Text-Deletion Commands

The table shows commands that delete text in 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 Escape.
`C`	Changes or overwrites characters from the 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,nd*	Deletes lines *n* –*n* (For example, `:5,10d` deletes lines 5–10.)

ORACLE

You can use numerous commands to edit files by 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.

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

Edit Commands

The table describes the commands to change text, undo a change, and repeat an edit function in 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 letter casing to uppercase or lowercase, at the cursor
`u`	Undoes the previous command
`U`	Undoes all changes to the current line
`.`	Repeats the previous command

Note

Many of these commands change the vi editor into the insert mode. To return to the command mode, press the Esc key.

Quiz

In which vi mode are commands normally initiated?

a. Ed mode
b. Ex mode
c. Command mode
d. Input mode

Answer: c

Search and Replace Commands

The table shows the commands that search for and replace text in the vi editor.

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

Copy and Paste Commands

The table shows the commands that cut, copy, and paste text in the vi editor.

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
:*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.

Save and Quit Commands

The table describes the commands that save the text file, quit the vi editor, and return to the shell prompt.

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
`ZQ`	Quits without saving changes

Session Customization

- You can customize a vi session by setting options for the session.
- When you set a option, you enable a feature that is not activated by default.
- You can use the `set` command to enable and disable options.
- Two of the many `set` command options include displaying line numbers and invisible characters, such as the Tab and the end-of-line characters.

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

1. Create a file named `.exrc` in your home directory.
2. Enter any of the set variables into the `.exrc` file.
3. Enter each set variable without the preceding colon.
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.

Note: The same steps apply for customizing a session in the Vim editor. Except that, instead of creating an `.exrc` file, you need to create a `.vimrc` file.

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

The table in the slide describes some of the variables of the `set` command.

Quiz

Which three commands help save changes in your file and quit the vi editor?

a. `:wq`

b. `:wq!`

c. `ZZ`

d. `:q!`

e. `:w`

Answer: a, b, c

Quiz

Which of the following commands searches backward for the pattern?

a. `?pattern`

b. `/pattern`

c. `!pattern`

d. `~pattern`

Answer: a

Summary

In this lesson, you learned to:

- Access the vi editor
- Modify files with the vi editor

Practice 4 Overview: Using the vi (vim) Editor

This practice covers the vi editor commands explained in the `tutor.vi` tutorial.

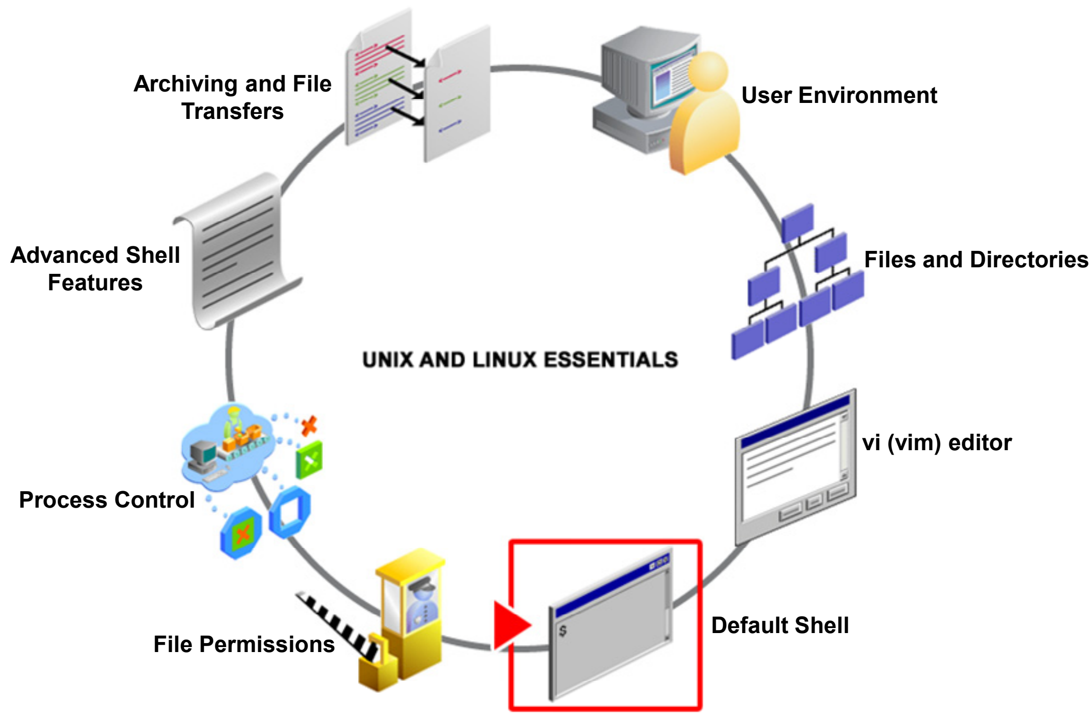
You will find the tasks for Practice 4 in your Activity Guide.

5

Using Commands Within the Default Shell

ORACLE

Workflow Orientation



The lesson titled “Using the vi (vim)Editor” explained the vi editor and the basic editing functions. This lesson introduces you to the bash shell, the default shell for both Oracle Solaris 11 and Oracle Linux. It also describes the various functions you can perform on a bash shell, such as using expansion characters, applying shell metacharacters, using command redirection, and working with the user initialization files.

Objectives

After completing this lesson, you should be able to:

- Use shell expansion for generating shell tokens
- Use shell metacharacters for command redirection
- Use variables in the Bash shell to store values
- Display the command history
- Customize the user's work environment

Lesson Agenda

- Using shell expansion for generating shell tokens
- Using shell metacharacters for command redirection
- Using variables in the Bash shell to store values
- Displaying the command history
- Customizing the user's work environment

ORACLE

Shell Expansions

- While working in a shell, sets or ranges of information are often repeated.
- Shell expansion helps generate a large number of shell tokens using compact syntaxes.
- Expansion is performed on the command line after the command is split into tokens.
- Of the many expansions available, the pathname, file name, and brace expansions are explained ahead.

Path Name Expansion

- The pathname expansion simplifies location changes within the directory hierarchy.
- The pathname expansion include:
  - The tilde (~) character, which represents the home directory of the current user
  - The tilde (~) character with a username, which represents the home directory of the specified user
  - The dash (-) character, which represents the previous working directory

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

The tilde (~) character is a substitution that equates to the absolute path name of the user's home directory.

Consider the following examples for each of the expansion character listed in the slide:

- Change directories to `dir1` by using the tilde (~) character.

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

- Change directories to the `user2` home directory using the tilde (~) character followed by a `username`.

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

Switch between the `user1` and `tmp` directories using the - expansion character.

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

File Name Expansion

The file name expansions are:

- The asterisk (*) character
- The question mark (?) character
- The square bracket ([ ]) characters

Asterisk (*) Character

- The asterisk (*) expansion character is also called the wildcard character or a glob and represents zero or more characters, except the leading period (.) of a hidden file.
- For example, list all files and directories that start with the letter f followed by zero or more other characters.

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


For example, list all files and directories that end with the number 3, preceded by zero or more characters.

```
$ ls *3
file.3 file3
dir3:
cosmos moon planets space sun vegetables
```

Question Mark (`?`) Character

- The question mark (`?`) character is also called a wildcard character and represents zero or any single character except the leading period (`.`) of a hidden file.
- For example, list all files and directories that start with the string `dir` and followed by one other character.

```
$ ls dir?
dir1:
coffees fruit trees
dir2:
beans notes recipes
dir3:
cosmos moon planets space sun vegetables
dir4:
constellation memo roses
Dir5:
```


Note

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 necessarily have 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 followed by a hyphen (-) and then the last character, for example, [a-z] or [0-9].
    - When specifying a range, arrange the characters in the order that you want them to appear in the output.
      - For example, use [A-Z] or [a-z] to search for any uppercase or lowercase alphabetical character, respectively.

For example, list all files and directories that start with the letters a through f.

```
$ 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
```

For example, list all files and directories that start with the letters f or p.

```
$ 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
```

The Brace Expansion

- The brace { } expansion is a mechanism by which arbitrary strings may be generated.
- Patterns to be brace expanded take the form of an optional preamble, followed by either a series of comma-separated strings or a sequence expression between a pair of braces, followed by an optional postscript.
- The preamble “a” is prefixed to each string contained within the braces, and the postscript “e” is then appended to each resulting string, expanding left to right.

```
bash$ echo a{d,c,b}e
ade ace abe
```

Quiz

Which of the following expansion character equates to the absolute path name of the user's home directory?

a. #

b. []

c. *

d. ~

Answer: d

Lesson Agenda

- Using shell expansion for generating shell tokens
- Using shell metacharacters for command redirection
- Using variables in the Bash shell to store values
- Displaying the command history
- Customizing the user's work environment

Shell Metacharacters

- Shell metacharacters are specific characters, generally symbols, that have special meaning within the shell.
- The metacharacters supported in bash are listed as follows:
  - `|`
  - `&`
  - `;`
  - `( )`
  - `<`
  - `>`
  - `space tab ^M (end-of-line)`

Note: The subsequent slides in this topic cover only the redirection characters.

The shell metacharacters are listed as follows:

- | : Sends the output of the command to the left as the input to the command on the right of the symbol
- & : Runs the process in the background, allowing you to continue working on the command line
- ; : Allows you to list multiple commands on a single line, separated by this character
- ( ) : Groups commands and sends their output to the same place
- < : Gets input for the command to the left from the file listed to the right of this symbol
- > : Sends the output of the command on the left into the file named on the right of this symbol
- `space tab ^M (eol)`

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

Redirection Metacharacters

Command redirection is enabled by the following shell metacharacters:

- Redirection of standard input (<)
- Redirection of standard output (>)
- Redirection of standard output (>>) append
- Redirection of standard error (2>)
- The pipe character (|)

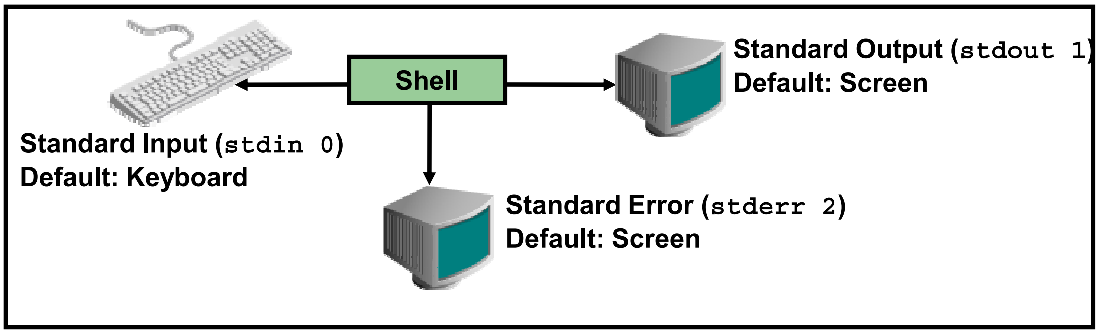
The File Descriptors

- Each process works with file descriptors.
- File descriptors determine where the input to the command originates and where the output and error messages are directed to.
- The table explains the file descriptors.

File Descriptor Number	File Description Abbreviation	Definition
0	`stdin`	Standard command input
1	`stdout`	Standard command output
2	`stderr`	Standard command error

Command Communication Channels

- By default, the shell receives or reads input from the standard input, the keyboard and displays the output and error messages to the standard output, the screen.



- 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.

Redirecting Standard Input (`stdin`)

- The less than (`<`) metacharacter processes a file as the standard input instead of reading the input from the keyboard.

```
command < filename
or
command 0< filename
```

- For example, use the `dante` file as the input for the `mailx` command.

```
$ mailx student < ~/dante
$
```

Redirecting Standard Output (`stdout`)

- The greater-than (>) metacharacter directs the standard output to a file instead of printing the output to the screen.

```
command > filename
or
command 1> filename
```

- If the file does not exist, the system creates it. If the file exists, the redirection overwrites the content of the file.
- For example, redirect the list of files and subdirectories of your current home directory into `directory_list` file.

```
$ cd
$ pwd
/export/home/student
$ ls -l > directory_list
```


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 content of the file.

```
$ command >> filename
```

For example, append the “That’s my directory_list file” string to the end of the `my_file` file.

```
$ ls -l > my_file; cat my_file
-rw-r--r--    1 student class 1319 Jun 28 2009 dante
drwxr-xr-x    5 student class 512 Jun 28 2009 dir1
... (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
drwxr-xr-x    5 student class 512 Jun 28 2009 dir1
... (output truncated)
That’s my directory list file
```

Note: The semicolon (;) is a shell metacharacter that allows you to use multiple commands on a single command line.

Redirecting Standard Error (`stderr`)

- A command using the file descriptor number (`2`) and the greater-than (`>`) sign redirects any standard error messages to the `/dev/null` file.

```
$ command 2>/dev/null
```

- The following example shows the standard output and the standard error redirected to the `dat` file.

```
$ ls /var /test 1> dat 2>&1
$ more dat
/test: No such file or directory (stderr)
/var: (stdout)
adm (stdout)
... (output truncated)
```

Note: The syntax `2>&1` instructs the shell to redirect `stderr (2)` to the same file that receives `stdout (1)`.

The Pipe Character

- The pipe character redirects the standard output from one command to the standard input of another command.
- The first command writes the output to standard output and the second command reads standard output from the previous command as standard input.

```
$ command | command
```

- For example, use the standard output from the `who` command as the standard input for the `wc -l` command.

```
$ who | wc -l
35
```

Note: You can use pipes to connect numerous commands.

Note

The output of the `who` command never appears on the terminal screen because it is piped directly into the `wc -l` command.

Using the Pipe Character

- To view a list of all the subdirectories located in the `/etc` directory, enter the following command.

```
$ ls -F /etc | grep "/"
X11/
acct/
apache/
apache2/
apoc/
<output truncated>
```

- For example, use the output of the `head` command as the input for the `tail` command and print the results.

```
$ head -10 dante | tail -3 | lp
request id is printerA-177 (Standard input)
```

Quoting Characters

- Quoting is a process that instructs the shell to mask or ignore the special meaning of shell metacharacters.
- The quoting characters are:
  - 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:
    - Backslash (`\`): Prevents the shell from interpreting the next character after the (`\`) as a metacharacter
    - Single backward quotation marks (`` ` ``): Instruct the shell to execute and display the output for a UNIX system command
    - Parentheses (`$`(command)): Instruct the shell to execute and display the output of the command enclosed within parentheses

For example, to ignore the special meaning of the dollar ($) metacharacter, enter the following command:

```
$ echo '$SHELL'
$SHELL
$ echo $SHELL
/bin/bash
```

Observe that the `echo` utility writes arguments to the standard output.

Quiz

The `ls -l 2> directory_list` command lists the content of your current directory and redirects that list into a file called `directory_list`.

a. True

b. False

Answer: b

Lesson Agenda

- Using shell expansion for generating shell tokens
- Using shell metacharacters for command redirection
- Using variables in the Bash shell to store values
- Displaying the command history
- Customizing the user's work environment

Introduction to Variables

- A variable is a temporary storage area in memory that is either set by the user, shell, system, or any program that loads another program.
- There are two categories of variables:
  - The shell variables apply only to the current instance of the shell and are used to set short-term working conditions.
  - The environment variables are a subset of the total shell variables and are valid for the duration of the session and any spawn sessions.

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.

Displaying Shell Variables

- The `echo` command displays the value stored inside a shell variable.

```
$ echo $SHELL
/bin/bash
```

- The `set` command lists all shell variables and their values.

```
$ 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
(Output truncated)
```

Setting and Unsetting Shell Variables

- To create a `bash` or `ksh` shell variable.

```
$ history=50
$ echo $history
```

- To create a `bash` or `ksh` environment variable.

```
$ export history=50
$ env | grep history
history=50
$ echo $history
```

- C shell variables are set using the `set` command and unset using the `unset` command.

```
$ set history = 50
$ echo $history
```


Note

The `set` command is also used to display the shell variables and their values.

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 equals (`=`) sign. For example:

```
$ private=/export/home/student/private
$ set | grep private
private=/export/home/student/private
$ cd $private; pwd
/export/home/student/private
$
```

Default Bash 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 on 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 Bash shell prompt: `$`
`PS2`	Specifies the secondary command prompt, normally: `>`
`SHELL`	Specifies the name of the shell (that is, `/bin/bash`)

The table describes variables that are assigned default values by the bash shell on login.

Note: Bash is a Bourne shell compatible.

Customizing `bash` Shell Variables: PS1

- The shell prompt string is stored in the shell variable `PS1`, and you can customize it according to your preference.

```
$ PS1='[\u@\h \W]\$'
[student@host1 ~]$
```

  – In this example, the prompt displays the login name of the user, the system's host name, and the current working directory.
  – The username is "`\u`" and the host name is "`\h`" and the current working directory is "`\W`".
  – This shell prompt displays the correct information even when the user logs in on different hosts.

Customizing `ksh` Shell Variables: PS1

- The shell prompt string is stored in the shell variable `PS1`, and you can customize it according to your preference.

```
$ PS1="$LOGNAME@`uname -n` \$PWD $ "
student@host1: $
```

  – In this example, the prompt displays the login name of the user, the system's host name, and the current working directory.
  – The username is read from the variable `LOGNAME`, and the host name comes from the output of the `uname -n` command, and the `$PWD` is the current working directory.
  – This shell prompt displays the correct information even when the user logs in on different hosts.
  – The back quotation (`` ` ``) mark delimits an embedded command string.

Customizing Shell Variables: `PATH`

- The `PATH` variable contains a list of directory pathnames, separated by colons.
- When executing a command on the command line, the shell searches for these directories from left to right, in sequence to locate that command.
- If the shell does not find the command in the list of directories, it displays a "not found" error message.
- To ensure that commands operate smoothly, you should include the respective directory in the `PATH` variable.
- The example illustrates the inclusion of the home directory into the `PATH` variable.

For example, to include the home directory in the `PATH` variable, perform 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/user1
```

The `PATH` variable automatically passes the value to the subshells.

Quiz

The `set` command lists all shell variables and their values.

a. True
b. False

Answer: a

Lesson Agenda

- Using shell expansion for generating shell tokens
- Using shell metacharacters for command redirection
- Using variables in the Bash shell to store values
- Displaying the command history
- Customizing the user's work environment

Introducing Command History

- The shell keeps a history of previously entered commands.
- This history mechanism enables you to view, repeat, or modify previously executed commands.
- By default, the `history` command displays all history entries to the standard output.

```
$ history
...
109 date
110 cd /etc
111 touch dat1 dat2
112 ps -ef
113 history
```

Note: The output may vary based on the commands recorded in the `.bash_history` file when you `exit` a session.

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.

Displaying Previously Executed Commands

- For example, display the list of command history without line numbers.

```
$ history -n
date
cd /etc
touch dat1 dat2
ps -ef
```

- For example, display the current command and the three commands preceding it.

```
$ history -3
111 touch dat1 dat2
112 ps -ef
113 history
114 history -n
$
```


Note

Command history is shared among all shells for a given user.

Displaying History List in Reverse Order

- You can display the history list in the reverse order too.

```
$ 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
...
$
```

- For example, display the most recent `cd` command to the most recent `date` command.

```
$ history -r cd date
110 cd /etc/
109 date
$
```

Using the `!` Command

- The `!` command is an alias built into the shell that enables you to repeat last command.
- In the output from the `history` command shows a line number in front of the command, use `!###` to re-execute any command.

```
$ history
...
109 date
110 cd /etc
111 touch dat1 dat2
112 ps -ef
113 history
```

- To re-execute the `112 ps -ef:`

```
$ !122
```


For example, to repeat the comand `cat dante`:

```
$ history
...
122 cat dante
123 ls
124 cd ~/dir1
$ !122
```

Using the `!!` Command

- The `!!` command is an alias built into the shell that enables you to repeat last command.
- For example, repeat the `cal` command by using `!!`.

```
$ 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
$ !!
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
```


For example, to repeat the comand `cat dante`:

```
$ history
...
122 cat dante
123 ls
124 cd ~/dir1
$ !122
```

Editing Commands on the Command Line

- You can edit commands using a shell inline editor.
- The default command-line editing mode in bash is emacs.
- You can, however, switch to the vi mode as well.
- The `set` command switches between the two modes.

```
$ set -o vi
$ set -o emacs
```

- You can also set the editing mode by using the `EDITOR` or `VISUAL` environment variables.

```
$ export EDITOR=/bin/vi
or
$ export VISUAL=/bin/vi
```


UNIX and Linux Essentials 5 - 41

Invoking File Name Completion

- File name completion is a feature that allows you to type part of a file name or directory name and press a key to fill out the rest.
- To invoke file name completion, enter the respective command followed by one or more characters of a file name and then press the `Esc/Tab` key.
- For example, expand a file name beginning with the characters `de` in the `/usr` directory:

```
$ cd /usr
$ ls de "Press the Tab key"
```

- The shell completes the remainder of the file name by displaying, `ls demo/`.

ORACLE

Invoking File Name Completion

- You can request the shell to present all possible alternatives of a partial file name from which you can select from.
- This request can be invoked by pressing the Escape (`Esc`) and the equal (`=`) sign keys in sequence.
- Or by pressing the Tab twice.

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 followed by the = key" or press the Tab twice
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.

Lesson Agenda

- Using shell expansion for generating shell tokens
- Using shell metacharacters for command redirection
- Using variables in the Bash shell to store values
- Displaying the command history
- Customizing the user's work environment

User Initialization Files

- Apart from having a home directory to create and store files, users need an environment that gives them access to the tools and resources.
- When a user logs in to a system, the user's work environment is determined by the initialization files.
- These initialization files are defined by the user's startup shell, which can vary depending on the release.
- The default initialization files in your home directory enable you to customize your working environment.

Default User Initialization Files for the Bash Shell

- When Bash is invoked, it first reads and executes commands from the `/etc/profile` file, if the file exists.
- Bash then reads and executes commands from the `~/.bash_profile`, `~/.bash_login`, `~/.bashrc` which reads `/etc/bashrc`, if it exists, and in the absence of the aforementioned files `~/.profile` file.
- When a login shell exits, Bash reads and executes commands from the `~/.bash_logout` file, if it exists.

- The `/etc/profile` file is a systemwide file that the system administrator maintains. This file defines tasks that the shell executes for every user who logs in. The instructions in the file usually set the shell variables, such as `PATH`, `USER`, and `HOSTNAME`.
- The ~/`.bash_profile` file is a configuration file for configuring user environments. The users can modify the default settings and add any extra configurations in it.
- The ~/`.bash_login` file contains specific settings that are executed when a user logs in to the system.
- The ~/`.bashrc` file contains specific settings that are executed when a user logs in to the system.
- The `/etc/bashrc` file contains specific settings that are executed when a user logs in to the system.
- The ~/`.profile` file is yet another configuration file that is read in the absence of the ~/`.bash_profile` and ~/`.bash_login` files.
- The ~/`.bash_logout` file contains instructions for the logout procedure.

Configuring the `~/.bashrc` File

- The `~/.bashrc` file is a personal initialization file for configuring the user environment.
- The file is defined in your home directory and can be used for the following:
  - Modifying your working environment by setting custom environment variables and terminal settings
  - Instructing the system to initiate applications

Quiz

Which of the following is the default command-line editing mode in bash?

a. vi
b. ed
c. emacs
d. Vim

Answer: c

Summary

In this lesson, you should have learned how to:

- Use shell expansion for generating shell tokens
- Use shell metacharacters for command redirection
- Use variables in the Bash shell to store values
- Display the command history
- Customize the user's work environment

Practice 5 Overview: Using Commands Within the Default Shell

This practice covers the following topics:

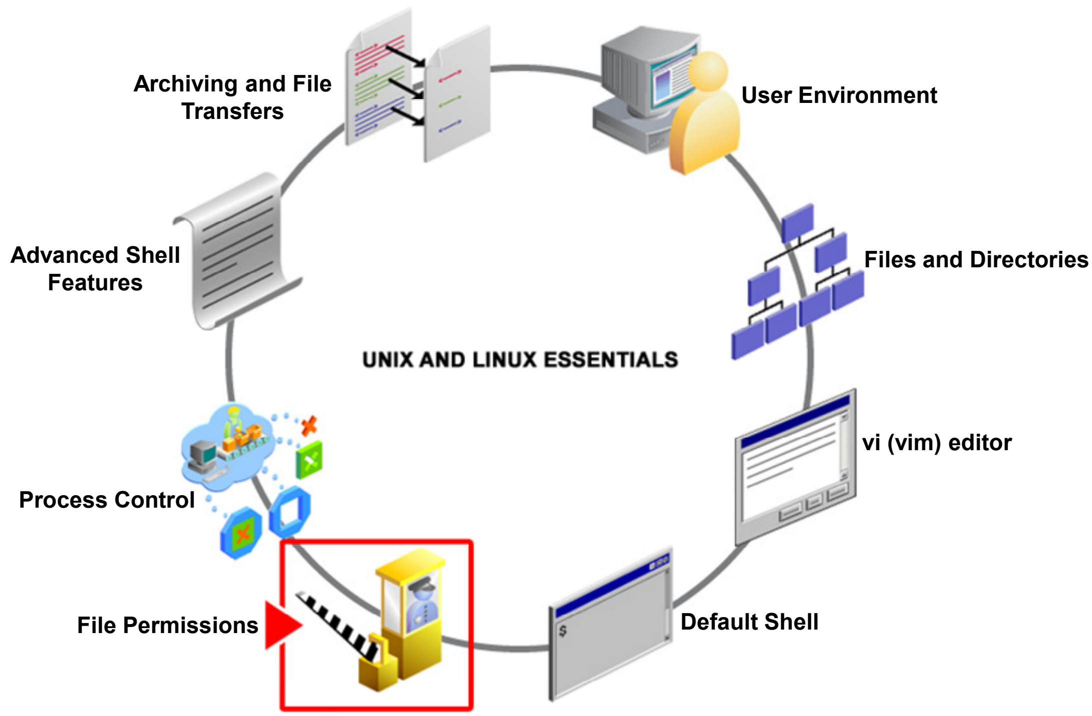
- Using the shell metacharacters
- Using variables in the Bash shell
- Displaying the command history
- Using the redirecting commands
- Customizing the user's work environment

You will find the tasks for Practice 5 in your Activity Guide.

6

Using Basic File Permissions

Workflow Orientation



ORACLE

The lesson titled "Using Commands Within the Default Shell" explained the workings of the bash shell. This lesson explains the basic file and directory permissions and the procedures to modify them when required.

Objectives

After completing this lesson, you should be able to:

- View file and directory permissions
- Change permissions
- Modify default permissions

Lesson Agenda

- Viewing file and directory permissions
- Changing permissions
- Modifying default permissions

Securing Files and Directories

- One of the important functions of a secure system is to limit access to authorized users and prevent unauthorized users from accessing the files.
- Oracle Solaris and Oracle Linux use two basic measures to prevent unauthorized access to a system:
  - The first measure is to authenticate a user's login by verifying that the username and password exist.
  - The second measure is to protect file and directory access automatically.
    - The Oracle Solaris and Oracle Linux OSs assign a standard set of access permissions at the time of file and directory creation.

Note

The Oracle Solaris OS and Oracle Linux OS also provide 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.

File and Directory Permissions

- All files and directories in Oracle Solaris and Oracle Linux have a standard set of access permissions.
- These access permissions control who can access what files, and provides a fundamental level of security to the files and directories in a system.

Viewing Permission Categories

- To view the permissions for files and directories, use the `ls -l` or `ls -n` commands.

```
$ ls -l dante
-rw-r--r-- 1 student class 1319 Mar 15 11:23 dante
```

rw- r-- r--

User/Owner Group Other

File type

`r` = Readable
`w` = Writeable
`x` = Executable
`-` = No permission

ORACLE

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 the permission groups: owner, group, and other.

Permission Groups

- There are three permissions groups:
  - User who owns the file (Owner)
  - Group
  - Other
- The table describes the permission groups and their scope:

Permission	Description
User/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

Permission Set

- Each permission group has three permissions, called a permission set.
- Each set consists of read, write, and execute permissions.
- Each file or directory has three permission sets for the three types of permission groups.
- The first permission set represents the user/owner permissions, the second set represents the group permissions, and the last set represents the other (world) permissions.

Permission Set

- The read, write, and execute permissions 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.
- A dash (`-`) symbol in place of a character in a permission set indicates that a particular permission is denied.
- Oracle Solaris and Oracle Linux assign initial permissions automatically when a new file or directory is created.

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.

Interpreting File and Directory Permissions

Permission	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 by 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.

The read, write, and execute permissions are interpreted differently when assigned to a file than when assigned to a directory. The table in the slide shows the permission definitions for a file and directory.

Note: For a directory to be of general use, it must at least have read and execute permissions.

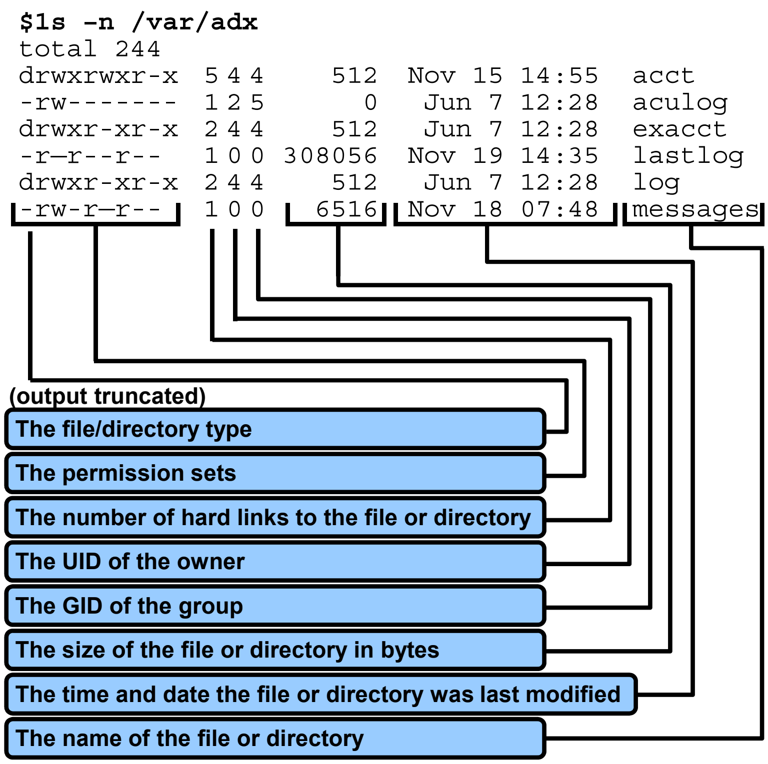
Determining File or Directory Access

- The `ls -n` command helps determine the ownership of files and directories.
- All files and directories have an associated user identification number (UID) and a group identification number (GID).
- 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
... (output truncated)
```


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 Oracle Solaris and Oracle Linux OSs use these numbers to track ownership and group membership of files and directories.

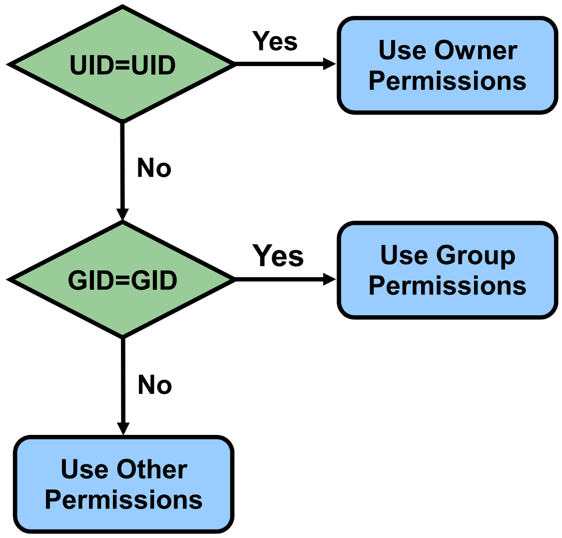
Interpreting the `ls -n` Command



The image in the slide illustrates the parts of the output of the `ls-n` command.

- The first character is the file/directory type.
- The next nine characters are the permission set.
- The next character represents the number of hard links to the file or directory. 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
- The next character represents the UID of the owner.
- The next character represents the GID of the group.
- The next set of characters represents the size of the file or directory in bytes.
- The next set of characters represents the time and date the file or directory was last modified.
- The last set of characters represents the name of the file or directory.

Determining Permissions



When a user attempts to access a file or directory, the UID of the user is compared with 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 user's GID is compared with the GID of the file or directory. If these numbers match, the group permissions apply.

If the GIDs do not match, the permission set for other is used to determine file and directory access.

The image in the slide shows the decision tree for determining file and directory permissions.

- If the UID equals the UID, then use the owner permissions.
- If not, does the GID equal the GID? If yes, use group permissions. If not, use other permissions.

Quiz

Which of the following directories have read and execute permissions set for the owner and group only?

a. `dr-xr-x---`

b. `dr-x---r-x`

c. `d---r-xr-x`

Answer: a

Lesson Agenda

- Viewing file and directory permissions
- Changing permissions
- Modifying default permissions

Changing the Permissions

- You can change the permissions on files and directories by using the `chmod` command.
- Either the owner of the file or directory or the root user can use the `chmod` command to change permissions.
- The `chmod` command can be used in either symbolic or octal mode.
  - Symbolic mode uses a combination of letters and symbols to add or remove permissions for each permission group.
  - Octal mode, also called the absolute mode, uses octal numbers to represent each permission group.

Note

You can assign execute permissions on files with the `chmod` command. The `chmod` command is described later in this lesson. Execute permissions are not assigned by default when you create a file.

Changing Permissions: 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 (user/owner, group, other, or all) affected
  - The function performed
  - The permissions affected
- For example, if the option is `g+x`, the executable permission is added to the group.

ORACLE

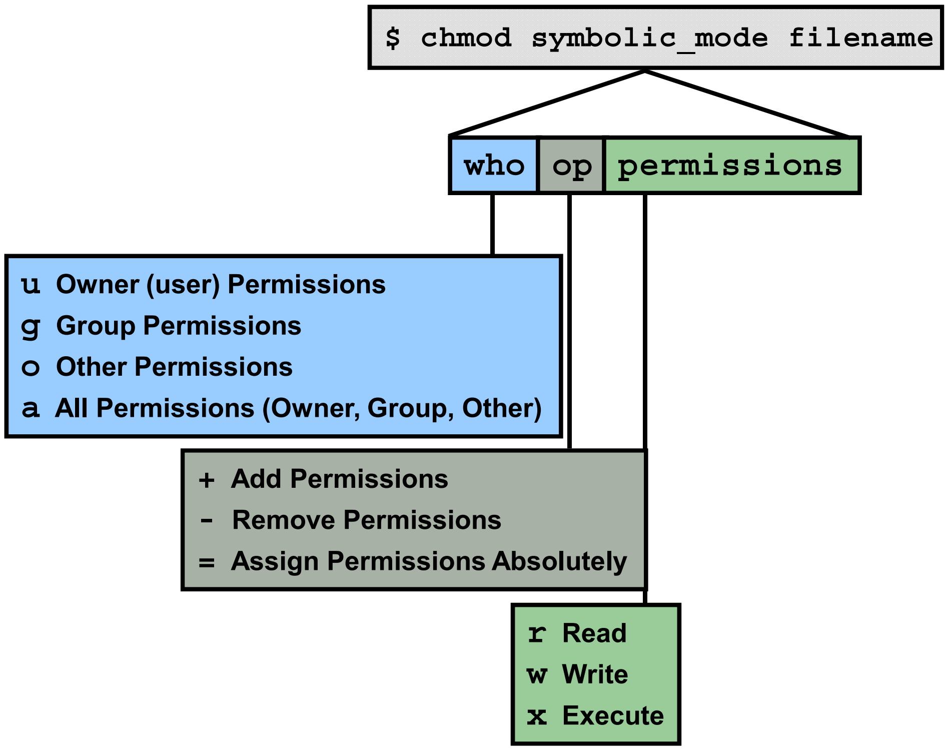
The following examples illustrate how to modify permissions on files and directories by using the symbolic mode. To remove the read permission for other users, run the following commands:

```
$ 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
```

Changing Permissions: Symbolic Mode



The image in the slide shows the components of the symbolic mode command syntax. The first three letters represent "who" and consist of the following codes:

- `u`: Owner (user) permission
- `g`: Group permissions
- `o`: Other permissions
- `a`: All permissions (owner, group, other)

The next section is the "op" section and consists of the following:

- `+`: Add permissions
- `-`: Remove permissions
- `=`: Assign permissions

The last section is the "permissions" section and consists of the following:

- `r`: Read
- `w`: Write
- `x`: Execute

Changing Permissions: Octal Mode

- The `chmod` command syntax in the octal mode is:

```
$ chmod octal_mode filename
```

- The `octal_mode` option consists of three octal numbers, 4, 2, and 1, which represent a combination of permissions, from 0–7, for the file or directory.

Octal Value	Permission
4	Read
2	Write
1	Execute

The table in the slide shows the octal numbers for each individual permission. These numbers are combined into one number for each permission set.

Changing Permissions: Octal Mode

Octal Value	Permission	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)`

The table in the slide shows the octal numbers that represent a combined set of permissions.

Changing Permissions: Octal Mode

- You can modify the permissions for each category of users by combining the octal numbers.
- The first set of octal number defines owner permissions, the second set defines group permissions, and the third set defines other permissions.

Octal Mode	Permissions
`644`	`rw-r--r--`
`751`	`rwxr-x--x`
`775`	`rwxrwxr-x`
`777`	`rwxrwxrwx`

The table shows the permission sets in the octal mode.

Changing Permissions: Octal Mode

- For example, set permissions so that the owner, group, and other have read and execute access only.

```
$ chmod 555 dante
$ ls -l dante
-r-xr-xr-x 1 student class 1319 Jan 22 14:51 dante
```

- The `chmod` command fills in any missing octal digits to the left with zeros.

```
$ chmod 44 dante
$ ls -l dante
----r--r-- 1   student        class    1319 Jan 22 14:51 dante
```

Note: `chmod 44 dante` becomes `chmod 044 dante`.

Caution

Not using the correct octal values or leaving one or more of the values can lead to unwanted access to files or directories. Some additional examples show how to modify permissions on files and directories by using the octal mode.

- Change owner and group permissions to include write access.

```
$ chmod 775 dante
$ ls -l dante
-rwxrwxr-x 1 student class 1319 Jan 22 14:51 dante
```

- Change the group permissions to read and execute only.

```
$ chmod 755 dante
$ ls -l dante
-rwxr-xr-x 1 student class 1319 Jan 22 14:51 dante
```

Quiz

What is the correct octal value for the “write and execute” file permission?

a. 3
b. 5
c. 6
d. 7

Answer: a

Quiz

What is the correct permission set for the `rwxr-xr-x` octal mode?

a. 775

b. 644

c. 755

d. 674

Answer: c

Lesson Agenda

- Viewing file and directory permissions
- Changing permissions
- Modifying default permissions

The `umask` Command

- When files and directories are created, initial permission values are automatically assigned.
- The initial permission value for a file is `666` (`rw-rw-rw-`) and `777` (`rwxrwxrwx`) for a directory.
- The user mask affects and modifies the default file permissions assigned to the file or directory.
- You can set the user mask by using the `umask` command in a user initialization file.
- To view the `umask` value, run the `umask` command.

```
$ umask
022
```

Note: The default `umask` value for Oracle Solaris is `022` while the default value on Linux is `002`.

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.

For example, to set the default file permissions in a user initialization file to `rw-rw-rw-`, run the following command:

```
$ umask 000
```

Determining `umask` Value

`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)

The table in the slide shows the file and directory permissions for each of the `umask` octal value. This table can also help you determine the `umask` value that you want to set on files and directories. To determine the `umask` value, subtract the value of the permissions that you want from `666` for a file or `777` from a directory.

For example, 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.

Applying the `umask` Value

- When you mask out certain permissions from the initial value, the default permissions assigned to the new files and directories remain.
- The table displays the results in the symbolic mode.

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
`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

ORACLE

For example, the initial permissions for a new file in the symbolic mode is `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 as, `420420420` or `666`.

- To mask out the write permission for the group and other, use `022`, the default `umask` value.
- The result in the octal mode is `420400400` or `644`, and `rw-r—r–` in the symbolic mode.

You can apply this same process to determine the default permissions for 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 the 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`, and `rwxr-xr-x` in the symbolic mode.

Changing the `umask` Value

- You can change the `umask` value to a new value on the command line.
- For instance, you might require a more secure `umask` value of say `027`, which assigns the following access permissions to newly created files and directories:
  - Files with read and write permissions for the owner, read permission for the group, and no permissions for other (`rw-r-----`)
  - Directories with read, write, and execute permissions for the owner, read and execute permissions for the group, and no permissions for other (`rwxr-x---`)

```
$ umask 027
$ umask
027
```

Note: The default `umask` is set in `/etc/profile` file.

Note

The new `umask` value affects only those files and directories that are created from this point onward. 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.

Summary

In this lesson, you should have learned how to:

- View file and directory permissions
- Change permissions
- Modify default permissions

Practice 6 Overview: Using Basic File Permissions

This practice covers the following topics:

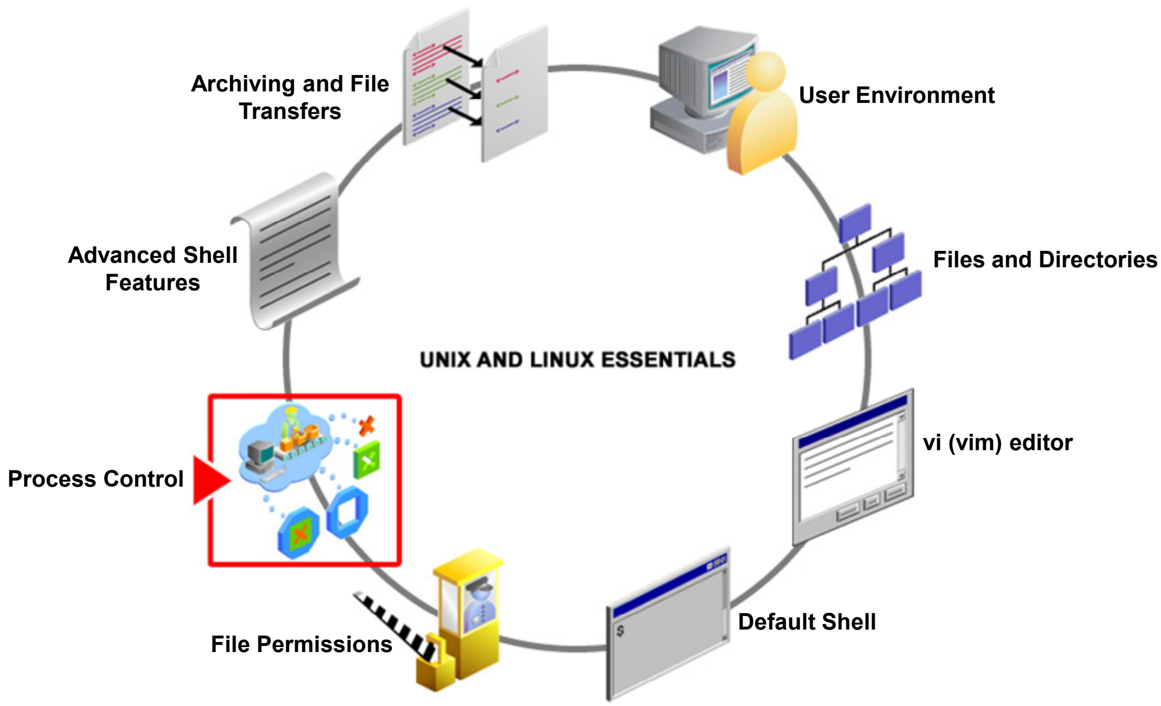
- Changing file ownership
- Changing file permissions
- Using the symbolic mode to change permissions
- Using octal mode to change permissions
- Modifying default permissions
- Viewing the default `umask`
- Changing the `umask` setting

You will find the tasks for Practice 6 in your Activity Guide.

7

Performing Basic Process Control

Workflow Orientation



The lesson titled “Using Basic File Permissions” explained the basic files permissions, such as viewing file and directory permissions, changing permissions, and modifying default permissions. This lesson describes a process, its attributes, the process states, and process subsystems. You also learn about the various commands that help you in managing and controlling system processes.

Objectives

After completing this lesson, you should be able to:

- Describe a process and its attributes
- Manage processes

ORACLE

Agenda

- Describing a process and its attributes
- Managing processes

A Process

- A process, also known as a task, is the running form of a program.
- Programs are stored on disk and processes run in memory.
- Processes have a parent/child relationship.
- A process can spawn one or more children.
- Multiple processes can run in parallel.

Attributes of a Process

- The kernel assigns a unique identification number to each process called a process ID or PID.
  - The kernel uses this PID to track, control, and manage the process.
- Each process is further associated with a UID and a GID.
  - UIDs and GIDs indicate the process owner.
  - Generally, the UID and GID associated with a process are the same as the UID and GID of the user who started the process.

A process consists of an address space and a metadata object. The process space pertains to all the memory and swap space a process consumes. The process metadata is just an entry in the kernel's process table and stores all other information about a process.

Process States

- The s, stat, and state output specifiers describe the state of a process.
- A process may be in any one of the following states:
  - D: Uninterruptible sleep (usually IO)
  - R: Running or runnable (on run queue)
  - S: Interruptible sleep (waiting for an event to complete)
  - T: Stopped, either by a job control signal or because it is being traced
  - Z: Defunct ("zombie") process, terminated but not reaped by its parent
- See the man pages for ps for more information.

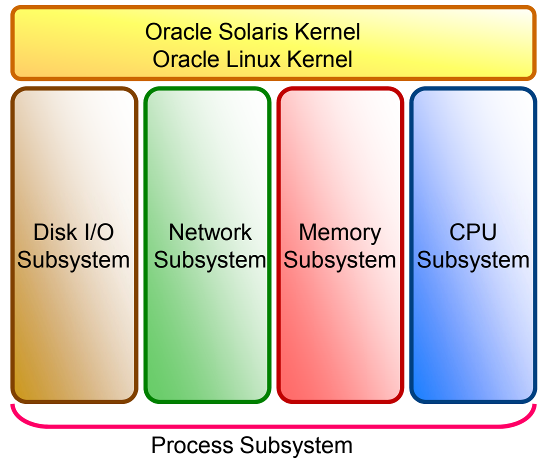
The process state can be displayed using the ps command. For BSD formats and when the stat keyword is used, additional state information is displayed such as the following:

- <: High-priority (not nice to other users)
- N: Low-priority (nice to other users)
- L: Has pages locked into memory (for real time and custom IO)
- s: Is a session leader
- l: Is multithreaded
- +: Is in the foreground process group

Note: nice is a useful program that is used to lower or increase the scheduling priority of a process or batch processes. Users can assign nice values between 0 (no effect) and 19 (greatest effect). The higher the nice value, the lower the scheduling priority.

Process Subsystems

- Each time you boot a system, execute a command, or start an application, the system activates one or more processes.
- A process as it runs, uses the resources of the various subsystems:
  - Disk I/O
  - Network
  - Memory
  - CPU



A process, as it runs, uses the resources of the various subsystems:

- The disk I/O subsystem: Controls disk utilization and resourcing as well as file system performance
- The network subsystem: Controls the throughput and directional flow of data between systems over a network connection
- The memory subsystem: Controls the utilization and allocation of physical, virtual, and shared memory
- The CPU subsystem: Controls CPU resources, loading, and scheduling

If not monitored and controlled, processes can consume your system resources, causing the system to run slowly and in some cases even halt. The Oracle Solaris 11 and Oracle Linux kernel collects performance-relevant statistics on each of these subsystems, to include process information. You can view and use this information to assess the impact that the processes have on the subsystem resources.

Agenda

- Describing a process and its attributes
- Managing processes

Listing System Processes

- The process status (`ps`) command lists the processes that are associated with your shell.

```
$ 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`).
- For example, list the currently running processes on the system using the `ps` command.

```
$ ps
 PID    TTY     TIME CMD
 1001   pts/1   0:00 bash
 1004   pts/1   0:00 ps
```


The `ps` command has several options that you can use to display additional process information.

- `-a`: Prints information about all processes most frequently requested, except process group leaders and processes not associated with a terminal
- `-e`: Prints information about every process currently running
- `-f`: Generates a full listing
- `-l`: Generates a long listing
- `-o format`: Writes information according to the format specification given in a format. Multiple `-o` options can be specified. The format specification is interpreted as the space-character-separated concatenation of all the format option arguments.

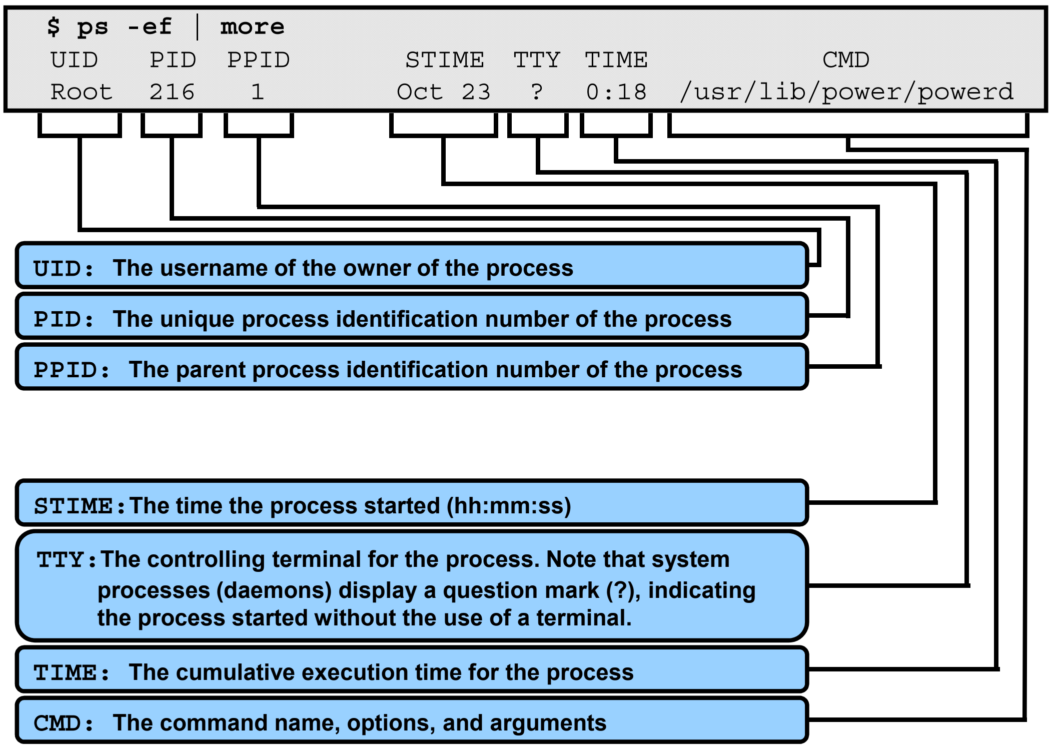
Note: Refer to the online man pages for a complete list of options for the `ps` command.

Listing All Processes

For example, use the `ps -ef` command to list 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      9        1       0        Feb 13 ?        0:00 svc.configd
--More--
... (output truncated)
```

Listing All Processes



The illustration in the slide interprets the output of the `ps -ef` command.

- The first column is the `UID`, the username of the owner of the process.
- The second column is the `PID`, the unique process identification number of the process.
- The third column is the `PPID`, the parent process identification number of the process.
- The fourth column is the `STIME`, the time the process started.
- The fifth column is the `TTY`, the controlling terminal for the process. Note that system processes (daemons) display a question mark (`?`).
- The sixth column is the `TIME`, the cumulative execution time for the process.
- The seventh column is the `CMD`, the command name, options, and arguments.

Quiz

Which of the following is not a process attribute?

a. UID
b. GID
c. PS
d. PID

Answer: c

Terminating a Process

- There might be times when you need to terminate an unwanted process.
- A process might have gotten into an endless loop, or it might have hung.
- You can kill or stop any process that you own.
- You can use the following two commands to terminate one or more processes:
  - `kill` *`PID [,PID ...]`*
  - `pkill` *`ProcessName`*
- The `kill` and `pkill` commands send signals to processes directing them to terminate.
- Each signal has a number, name, and an associated event.

However, there are processes that should not be terminated, such as the `init` process. Killing such processes can result in a system crash.

Note: A superuser can kill any process in the system.

Terminating a Process: `kill` Command

- You can terminate any process by issuing the appropriate signal to the process concerned.
- The `kill` command sends a termination signal to one or more processes.

```
$ kill [-signal] PIDs
```

- For more details on signal values review `man 7 signal`.

Note: The `kill` command terminates only those processes that you own.

The `kill` command sends signal `15`, the terminate signal, by default. This signal causes the process to terminate in an orderly manner.

You need to know the PID of the process before you can terminate it. You can use either the `ps` or `pgrep` command to locate the PID of the process. Also, you can terminate several processes at the same time by entering multiple PIDs on a single command line.

Note: The root user can use the `kill` command on any process.

Terminating a Process: `kill` Command

Use the `kill` command to terminate the `dtmail` process.

```
$ pgrep -l mail
215 sendmail
12047 dtmail
$ kill 12047
$ pgrep -l mail
215 sendmail
```

Terminating a Process: `pgrep & pkill` Command

- Often times one can use the `pgrep -l` command to identify the processes to be killed, and then use the `pkill` command to kill them.

```
$ sleep 500 &
[1] 4378
$ pgrep -l sleep
4378 sleep
$ pkill sleep
[1]+   Terminated            sleep 500
```

- The `pkill` command requires you to specify the name instead of the PID of the process.

Terminating a Process: `pkill` Command

Use the `pkill` command to terminate the `dtmail` process.

```
$ pkill dtmail
$ pgrep -l mail
215 sendmail
```

Forcefully Terminating a Process: Signal `9 (SIGKILL)`

- Some processes ignore the default signal `15 (SIGTERM)` that the `kill` command sends.
- If a process does not respond to signal `15`, you can force it to terminate by using signal `9` with the `kill` or `pkill` command.

```
$ kill -9 PID
or
$ pkill -9 -x process_name
```

Note: Sending signal `15` does not necessarily kill a process gracefully. Only if the signal is caught by the process, it cleans itself up in order and dies. If not, it just dies.

Caution

Use the `kill -9` command only when necessary. When you use the `kill -9` command on an active process, the process terminates instantly. Using signal `9` on processes that control databases or programs that update files could cause data corruption.

Quiz

Ordinary users can only `kill` processes they own.

a. True
b. False

Answer: a

Summary

In this lesson, you should have learned how to:

- Describe a process and its attributes
- Manage system process

Practice 7 Overview: Performing Basic Process Control

This practice covers the following topics:

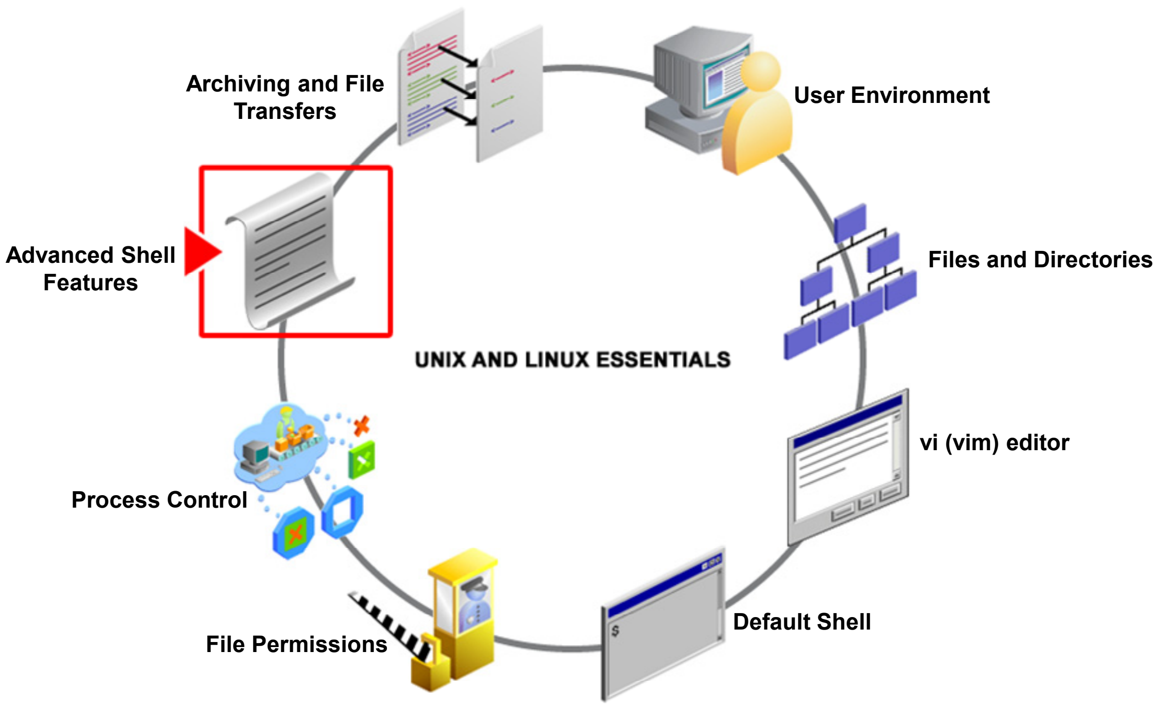
- Listing system processes
- Controlling system processes
- Terminating a process

You will find the tasks for Practice 7 in your Activity Guide.

8

Using Advanced Shell Features in Shell Scripts

Workflow Orientation



The lesson titled "Performing Basic Process Control" described the various aspects of a process and ways to manage and control processes. This lesson introduces some advanced shell features in shell scripts, such as managing jobs, creating aliases, using shell functions, and options. The lesson also explains how to create shell programs using scripting constructs and execute the programs.

Objectives

After completing this lesson, you should be able to:

- Use advanced shell features
- Write shell scripts

Agenda

- Using advanced shell features
- Writing shell scripts

Jobs in the Bash Shell

- A job is a process that the shell manages.
- Each job is assigned a sequential job ID.
- Because a job is a process, each job has an associated PID.
- There are three types of job statuses:
  - Foreground
  - Background
  - Stopped

Note: Except the Bourne shell, the other shells support job control.

There are three types of job statuses:

- **Foreground:** When you enter a command in a terminal window, the command occupies that terminal window until it completes. This is a foreground job.
- **Background:** 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 is an example of a background job.
- **Stopped:** 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

- Job control commands enable you to place jobs in the foreground or background, and to start or stop jobs.
- The table describes the job control commands.

Option	Description
`jobs`	Lists all jobs
`bg %`*n*	Places the current or specified job in the background, where *n* is the job ID
`fg %`*n*	Brings the current or specified job into the foreground, where *n* is the job ID
`Control-Z`	Stops the foreground job and places it in the background as a stopped job

Note

The job control commands enable you to run and manage multiple jobs within a shell. However, you can use the job control commands only in the shell where the job was initiated.

Running a Job in the Background

- To run a job in the background, you need to enter the command that you want to run, followed by an ampersand (`&`) symbol at the end of the command line.
- For example, run the `sleep` command in the background.

```
$ sleep 500 &
[1] 3028
```

- The shell returns the job ID, in brackets, that it assigns to the command and the associated PID.

Note: The `sleep` command suspends execution of a program for `n` seconds.

Note

With the job ID, you can use the job control commands to manage the job whereas the kernel uses PIDs to manage jobs.

When a background job is complete and you press Return, the shell displays a message indicating the job is done.

```
[1] + Done          sleep 500 &
```

Running a Job in the Background: Examples

- You can use the `jobs` command to list the jobs that are currently running or suspended in the background.

```
$ jobs
[1] + Running             sleep 500 &
```

- You can use the `fg` command to bring a background job to the foreground.

```
$ fg %1
sleep 500
```

Note: The foreground job occupies the shell until the job is completed, suspended, or stopped and placed into the background.

ORACLE

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 either in the foreground or background, the job restarts.

Quiz

To run a job in the background, you need to enter the command that you want to run, followed by a pipe (|) symbol at the end of the command line.

a. True

b. False

Answer: b

The Alias Command

- An alias is a shorthand shell notation that allows you to customize and abbreviate commands.

```
$ alias name=command_string
```

- If the first word on the command line is an alias, the shell replaces that word with the text of the alias.
- The shell maintains a list of aliases that it searches when a command is entered.
- The following rules apply while creating an alias:
  - 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.

Note

Aliases are available in all other shells, as well.

Command Sequence

- You can group several commands under a single alias name.
- Individual commands are separated by semicolons.

```
$ alias info='uname -a; id; date'
$ info
SunOS host1 5.10 Generic_120011-14 Oracle4u sparc OracleW,
Oracle-Blade-1500 uid=1002(user2) gid=1000(class)
Fri Feb 13 15:22:47 MST 2009
```


In the following 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 list appears.

```
$ 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--
```

Predefined Aliases

- The Linux `bash` shell contains several predefined aliases.
- You can display these predefined aliases by using the `alias` command.

```
$ alias
alias grep='grep --color'
alias l.='ls -d .* --color=auto'
alias ll='ls -l --color=auto'
alias ls='ls --color=auto'
alias vi='vim'
alias which='alias | /usr/bin/which --tty-only --read-alias
--show-dot --show-tilde'
```

Note: The `alias` command also displays user-defined aliases.

Oracle Solaris 11 does not contain any predefined aliases, but Oracle Linux 6.2 does provide them.

User-Defined Aliases

- User-defined aliases are defined by a user, usually to abbreviate or customize frequently used commands.
- For example, the `history` command is aliased as `h` using the `alias` command in the following code:

```
$ 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, the `rm` command is aliased with the `-i` option as coded as follows:

```
$ alias rm='rm -i'
$ rm dat1
rm: remove dat1: (yes/no)? no
```

Similarly, creating a `cp -i` and `mv -i` alias ensures that the shell prompts you for confirmation before overwriting existing files.

Deactivating an Alias

- You can temporarily deactivate an alias by placing a backslash (`\`) in front of the alias on the command line.
- For example, in the following code, the backslash prevents the shell from looking in the alias list. This allows the shell to run the original `rm` command to remove the `file1` file.

```
$ rm file1
rm: remove file1 (yes/no)? no
$
$ \rm file1
$ ls file1
file1: No such file or directory
```

Removing an Alias

- The `unalias` command removes aliases from the alias list.

```
$ unalias alias_name
```

- For example, the `h` alias that was created earlier is removed using the `unalias` command.

```
$ unalias h
$ h
bash: h: not found
```


Note

To pass the new aliases to every shell invoked, place it in your Bash shell initialization file.

Quiz

Which of the following rules do not apply while creating an alias?

a. There can be no space on either side of the equal sign.
b. The backslash (\) is always placed in front of the alias.
c. The command string in an alias must be quoted if it includes any options, metacharacters, or spaces.
d. Each command in a single alias must be separated with a semicolon.

Answer: b

Shell Functions

- Functions, a powerful feature of shell programming, is a group of commands organized by common functionality.
- These easy-to-manage units, when called return a single value, and do not output anything.
- Using a function involves two steps:
  1. Defining the function
  2. Invoking the function

Note: Shell functions and aliases are different on two counts. First, aliases do not take arguments as functions do. Second, if a command name is defined as a function and an alias, the alias takes precedence.

ORACLE

- 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
```

Defining a Function

- A function is defined by using the following general format:

```
function name { command; . . . command; }
```

Note: A space must appear after the opening brace and before the closing brace.

- The following example defines a function called `num` that displays the total number of users currently logged in to the system. The `num` function runs the `who` command, whose output is further directed to the `wc` command.

```
$ function num { who | wc -l; }
```


Functions are not only useful in shell scripts but are also used in command-line situations where an alias is unusable. For demonstration, shell functions are run on the command line to illustrate how the functions perform.

The following example creates a function called `list` that displays the total number of subdirectories and files in the current directory. The `list` function calls the `ls` command, whose output is directed to the `wc` command:

```
$ function list { ls -al | wc -l; }
$ list
34
```

Invoking a Function

You can invoke a function by merely entering the function name on the command line or within the shell script.

```
$ num
```

Shell Options

- Options are switches that control the behavior of the shell.
- Options are of the `Boolean` data type, which means they can be either `on` or `off`.
- To show current option settings, enter:

```
$ set -o
```

- To turn on an option, enter:

```
$ set -o option_name
```

- To turn off an option, enter:

```
$ set +o option_name
```


Note

The `set -o` and `set +o` options can change only a single option setting at a time.

Example of Shell Options: Activating the `noclobber` Option

- 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 on the screen.
- The `noclobber` option is activated in the shell by using the `set` command.

The `noclobber` option is activated in the shell by using the `set` command.

```
$ set -o noclobber
$ set -o | grep noclobber
noclobber on
$ ps -ef > file_new
$ cat /etc/passwd > file_new
bash: file_new: file already exists
```

Example of Shell Options: Deactivating the `noclobber` Option

- To deactivate the `noclobber` option, enter the following commands:

```
$ set +o noclobber
$ set -o | grep noclobber
noclobber off
```

- To temporarily deactivate the `noclobber` option, use the `>|` deactivation syntax on the command line.

```
$ ls -l >| file_new
```

Note: There is no space between the `>` and `|` on the command line. The `noclobber` option is ignored for this command line only, and the contents of the file are overwritten.

Quiz

Which of the following syntaxes can be used to turn off an option?

a. `$ set +o option_name`

b. `$ set -o e`

c. `$ set -o option_name`

Answer: a

Agenda

- Using advanced shell features
- Writing shell scripts

Shell Scripts

- A shell script is a text file that contains a sequence of commands and comments.
- Shell scripts are often used to automate repeating command sequences, such as services that start or stop on system startup or shutdown.
- Users with little or no programming experience can create and run shell scripts.
- You can run the shell script by simply entering the name of the shell script on the command line.

Note

A shell script can only be activated on the command line if it is made executable. Nonexecutable scripts can be executed by using the bash script.

Determining the Shell to Run a Shell Script

- Oracle Solaris and Oracle Linux support various shells, such as Bourne, Korn, C, and their derivatives.
- The first line of a script identifies the shell program that interprets and executes the lines in the script.
- The first line should always begin with the characters `#!` followed immediately by the absolute path name of the shell required to run the script.

```
#!/full-pathname-of-shell
```

- For example, the first line for a Bash shell script is as follows:

```
#!/bin/bash
```


Note

Shell scripts are not compiled into binary form.

Creating a Shell Script

- To create a shell script, you need a text editor.
- A text editor is a program that reads and writes text files.
- The following code is a simple shell script.

```
#!/bin/bash
# This is my first shell script.
echo "Hello World!"
```

  - The first line of the script indicates the program that interprets the script. In this case, it is `/bin/bash`.
  - The second line is a comment. Everything that appears after a (#) symbol is ignored by bash.
  - The last line is the `echo` command, which prints what is displayed.

Executing a Shell Script

- After the shell script is created, you can run it.
- To run a shell script, the user must have execute permissions.
  - For example, to grant read and execute permissions to the user so that you can execute the `mycmd` shell script, use the `chmod` command.

```
$ chmod u+rx mycmd
```

- A shell script is executed by just calling out the script name on the command line.
  - For example, to run the `mycmd` script in the current directory, enter `./mycmd` on the command line.

ORACLE

Note

When a shell script is running, any applied changes occur in the subshell or child process. A subshell 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
```

Comments in a Shell Script

- A comment is a textual description of the script and the lines within the script file.
- Comments are always preceded by a hash (#) character.

```
# This is a comment inside a shell script
ls -l # lists the files in a directory
```

- Whenever a shell encounters a (#) character in a script file, the line 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.

Positional Parameters in a Shell Script

- You can pass command-line arguments to a shell script while it is running.
- As you pass these arguments on the command line, the shell stores the first parameter after the script name into variable `$1`, the second into variable `$2`, and so on.
- These variables are called positional parameters.

The following set of commands illustrates how arguments are passed to the `greetings` script.

1. View the `greetings` script.
```
$ cat greetings
#!/bin/sh
echo $1 $2 #echo the first two parameters passed
```
2. Add `execute` permissions to `greetings`.
```
$ chmod u+x greetings
```
3. Run `greetings` while passing the `hello` and `world` values.
```
$ greetings hello world
hello world
```

Shifting the Positional Parameters

- While passing command-line arguments, the Bourne shell accepts only a single number after the `$` sign (`$0-$9`).
- 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 when processing the positional parameters in a loop.
  - For example, the value of the `$2` parameter becomes assigned to the `$1` parameter.
- Both the Korn shell and Bash shell can access the 10th parameter directly with the value of the 10th argument `${10}`. However that could become very cumbersome in a loop.

Quiz

To run a shell script, the user must necessarily have write permissions.

a. True

b. False

Answer: b

Checking the Exit Status

- Exit status is a numeric value that indicates the success or failure of a command.
  - A value of zero indicates success.
  - A nonzero value indicates failure.
    - This nonzero value can be any integer in the range of `1–255`.
- A developer can use the exit status values to indicate different error situations.
- All commands in the Oracle Solaris and Oracle Linux environment return an exit status, which is held in the read-only shell variable `$?`.

The exit status of the last command run in the foreground is held in the `?` special shell variable, and can be tested by using the `echo` command.

```
$ grep other /etc/group
other::1:
$ echo $?
0
$ grep others /etc/group
$ echo $?
1
```

The `test` Command

- The built-in `test` command, within a shell script, is used for testing conditions.
- The `test` command is also used for evaluating expressions, such as the following:
  - Variable values
  - File access permissions
  - File types
- The `test` command can be written as a test expression or written using the `[ expression ]` special notation.

Using the `test` Command

- The `test` command often follows the `if` statement.
- The `test` command evaluates an expression, and, if the result is true, it returns an exit status of zero.
- If the result is false, the `test` command returns a nonzero exit status.

```
if test_expression
        then
        command
fi
```


For demonstration, the following examples of the `test` command are run on the command line.

- Test whether the value of the `LOGNAME` variable is `student`.

```
$ echo $LOGNAME
student
$ test "$LOGNAME" = "student"
$ echo $?
0
```

- Test whether the value of the `LOGNAME` variable is `student` using the `[ expression ]` notation.

```
$ echo $LOGNAME
student
$ [ "$LOGNAME" = "student" ]
$ echo $?
0
```

Conditional Expressions

The shell provides the following special expressions that enable you to run a command based on the success or failure of the preceding command.

- The `&&` operator
- The || operator
- The `if` statement
- The `for` statement
- The `while (true)` statement
- The `until (true)` statement
- The `case` statement

The `&&` Operator

The `&&` operator ensures that a command is run only if the preceding command succeeds.

```
$ mkdir $HOME/newdir && cd $HOME/newdir
```

The || Operator

The || operator ensures that a command is run only if the preceding command fails.

```
$ mkdir /usr/tmp/newdir || mkdir $HOME/newdir
```

The `if` Statement

- The `if` statement evaluates the exit status of a command and initiates additional actions based on the return value.

```
$ if command1
 > then
 > execute command2
 > else
 > execute command3
 > fi
```

- If the exit status is zero, any commands that follow the `then` statement are run.
- If the exit status is nonzero, any commands that follow the `else` statement are run.

Note: The `if` statement is often used with the `test` command.

For example, display the greetings message using the `if` statement.

```
$ id
uid=101(frame) gid=1(other)
$
$ if test "$LOGNAME" = root
> then echo Hello System Administrator
> else
> echo Hello "$LOGNAME"
> fi
Hello frame
```

The `for` Statement

- The `for` command enables you to repeat a command or group of commands in a loop.

```
$ for arg in command1;
> do
> command2
> done
```

- The `for` command evaluates the exit status of the `in` operation that follows it.
  - If the exit status is zero, any instructions that follow the `do` statement are run, `command1` is rerun, and the exit status rechecked.
  - If the exit status is nonzero, the loop terminates.

The `while` Statement

- The `while` command enables you to repeat a command or group of commands in a loop.

```
$ while command1
> do
> command2
> done
```

- The `while` command evaluates the exit status of the `command1` command that follows it.
  - If the exit status is zero, any instructions that follow the `do` statement are run, `command1` is rerun, and the exit status rechecked.
  - If the exit status is nonzero, 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
loop
```

The `until` Statement

- The `until` command enables you to repeat a command or group of commands in a loop.

```
$ until command1
> do
> command2
> done
```

- The `until` command evaluates the exit status of the `command1` command that follows it.
  – If the exit status is nonzero, any instructions that follow the `do` statement are run, `command1` is rerun, and the exit status rechecked.
  – If the exit status is zero, the loop terminates.

For example, use the `set` command to assign values to the positional parameters as follows:

```
$ set this is a until loop
$ echo $*
this is a while loop
$ until  [ $# -le 0 ]
> do
> echo $1
> shift
> done
this
is
a
while
loop
```

The `case` Statement

The `case` command compares a single value against other values, and runs a command or group of commands when a match is found.

```
$ case value in
> pat1)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 Oracle Corporation 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
```

Quiz

Which of the following evaluate the exit status of a command and initiate additional actions based on the return values?

a. The `case` statement
b. The `test` command
c. The `if` statement
d. The `while` statement

Answer: c, d

Summary

In this lesson, you should have learned how to:

- Use advanced shell features
- Write shell scripts

Practice 8 Overview: Using Advanced Shell Features in Shell Scripts

This practice covers the following topics:

- Managing jobs in the Bash shell
- Creating an `alias`
- Using Bash shell functions
- Setting Bash shell options
- Creating and running shell scripts
- Passing values to a shell script
- Using the `test` command
- Executing conditional statements

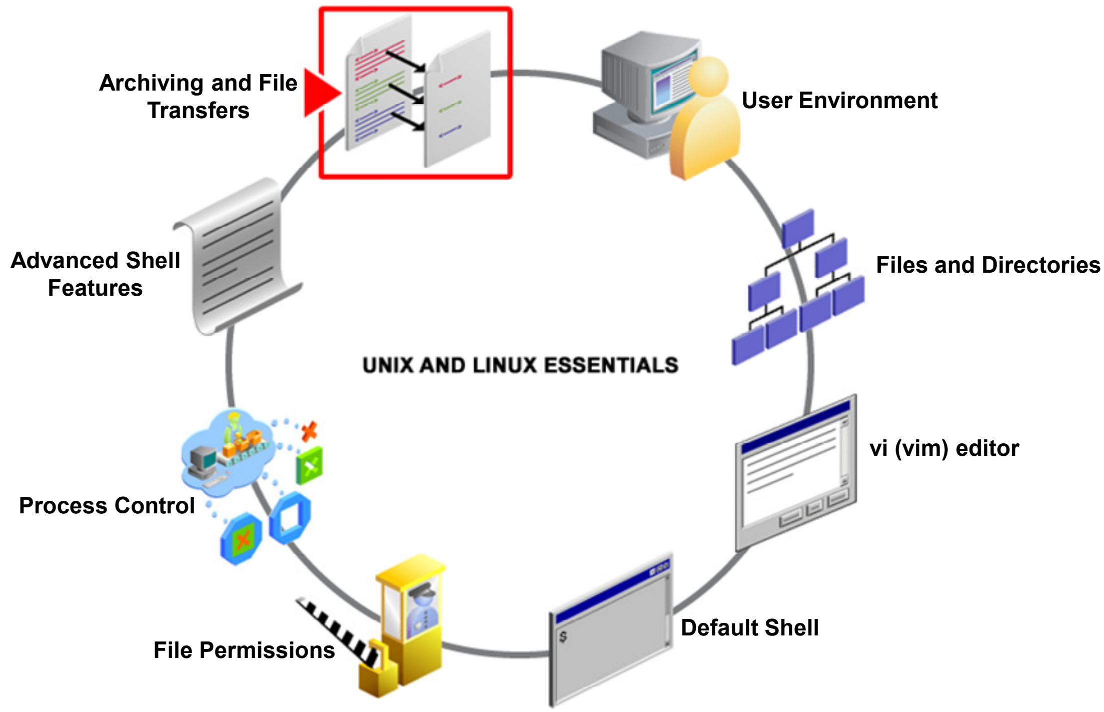
You will find the tasks for Practice 8 in your Activity Guide.

9

Archiving Files and Performing Remote Transfer

ORACLE

Workflow Orientation



The lesson titled "Using Advanced Shell Features in Shell Scripts" introduced some advanced shell features in shell scripts and explained how to create and execute shell programs. This lesson addresses the need for and the means of file archival and retrieval. Also, the lesson covers the various file compression and uncompression tools. Finally, you learn about remote file transfers.

Objectives

After completing this lesson, you should be able to:

- Archive and retrieve files
- Compress, view, and uncompress files
- Perform remote connections and file transfers

Agenda

- Archiving and retrieving files
- Compressing, viewing, and uncompressing files
- Performing remote connections and file transfers

Introduction to File Archival

- To safeguard your files and directories, you can create a copy of all the files and directories in your system.
- This copy is a repository of files and directories and is called an archive.
- The archive serves as backup in the event of data loss.
- You can create an archive on a storage device, such as a disk or a tape.
- Of the many commands, the following are most commonly used for creating and retrieving archived files:
  - The `tar` command

Note: It is a good practice to use relative path names to archive files.

Note

`cpio` is yet another preferred archival program. Unlike `tar`, which automatically recurses subdirectories, `cpio` reads a list of files and directories from `stdin`, creates the archive, and writes the archive to `stdout`.

The `tar` Command

- The `tar` command stores, lists, or extracts files in a tape archive file.

```
$ tar [option] archivefile filenames
```

- The output of using a `tar` command is a `tar` file.
- The default output location for a `tar` file in Oracle Solaris and Oracle Linux is the `stdout`.

Note

The `tar` command in Oracle Solaris 11 and Oracle Linux 6.2 strips the leading "`/`" character automatically, unlike in Oracle Solaris 10. This means that the files are now extracted to the current directory and not to `root`.

The `tar` Command Options

Option	Description
`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.
`v`	Executes in verbose mode, writes to the standard output
`h`	Follows symbolic links as standard files or directories
`z`	Reads or writes archives through gzip
j	Compresses and extracts files and directories using bzip

The table describes some of the commonly used `tar` command options. For a detailed explanation of the `tar` command and its options, read the `tar` man page.

Creating a `tar` Archive

- You can use the `tar` command to create an archive file containing multiple files or directories onto a disk or file.
- The following example shows you how to archive your home directory onto a disk.

```
$ tar cvf /dev/rmt/0 .
a ./ 0 tape blocks
a ./.rhosts 1 tape blocks (output truncated…)
```

- The following example shows you how to archive multiple files into an archive file called `files.tar`.

```
$ tar cvf files.tar file1 file2 file3
a file1 2K
a file2 1K
a file3 1K
```

Viewing a `tar` Archive

- You can view the names of all the files that have been written directly to a disk or file archive.
- To view the content of the student home directory on the disk, enter the following command:

```
$ tar tf /dev/rmt/0
/.rhosts
./dante
./fruit (output truncated…)
```

- To view the content of the `files.tar` archive file, enter the following command:

```
$ tar tf files.tar
file1
File2 (output truncated…)
```

Extracting a `tar` Archive

- You can retrieve or extract the contents of an archive that was written directly to a disk device or to a file.
- To retrieve the files from the disk archive, enter the following command:

```
$ tar xvf /dev/rmt/0
x ., 0 bytes, 0 tape blocks
x ./.rhosts, 2 bytes, 1 tape blocks (output truncated…)
```

- To extract files from the `files.tar` archive file, enter the following command:

```
$ tar xvf files.tar
tar: blocksize = 11
x file1, 1610 bytes, 4 tape blocks (output truncated…)
```

Quiz

Which command would you use to view the content of the archive file named `file8.tar`?

a. `tar xvf file8.tar`

b. `tar cvf file8.tar`

c. `tar tf file8.tar`

Answer: c

Quiz

Which command has packaging and compression capabilities, in addition to archiving features?

a. The `tar` command

b. The `mt` command

Answer: a, b

Agenda

- Archiving and retrieving files
- Compressing, viewing, and uncompressing files
- Performing remote connections and file transfers

File Compression

- With the enormous amount of enterprise data that is created and stored, there is a pressing need to conserve disk space and optimize data transfer time.
- There are various tools, utilities, and commands that are used for file compression. Some of the commonly used commands are:
  - The `compress` command
  - The `gzip` command
  - The `zip` command
  - The `bzip2` command

The `compress` Command

- When compressing a file, the `compress` command replaces the original file with a new file that has a `.Z` extension.

```
$ compress [option] filename
```

- The ownership and modification time of the original file remain intact, but the content of the file changes.

The amount of compression depends on the type of file you compress. Typically, compression reduces a text file by 50 to 60 percent.

Compressing a File: `compress` Command

- The following example shows you how to compress a file called `files.tar`.

```
$ compress -v files.tar
files.tar: Compression: 70.20% -- replaced with files.tar.Z
```

  – The `-v` (verbose) option provides information about the percentage of reduction or expansion of each file.
  – The compressed file, `files.tar.Z`, replaces the `files.tar` file.

Note: 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.

When you compress a file that has already been compressed, the file size increases, instead of becoming smaller. Also, 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.

Viewing a Compressed File: `zcat` Command

- The `zcat` command prints the uncompressed form of a compressed file to the standard output.

```
$ zcat [option] filename
```

- To view the content of the `dante.Z` compressed file, enter the following command:

```
$ 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 (Output Truncated…)
```

Note: The `zcat` command interprets the compressed data and displays the content of the file as if it were not compressed.

Note

The `zcat filename` command is functionally identical to the `uncompress -c filename` command.

Uncompressing a File: `uncompress` Command

- The `uncompress` command restores a compressed file to an uncompressed state.

```
$ uncompress [options] filename
```

- To uncompress the `files.tar.Z` file and restore it to the `files.tar` file, enter the following command:

```
$ uncompress -v files.tar.Z
files.tar.Z: -- replaced with files.tar
$
```

  – The `-v` option displays additional messages about the action being performed.

You can use the `uncompress` command with the `-c` option to send the content of a compressed file to the `stdout`, the screen, without changing the compressed `.Z` file. Otherwise, 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 content 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 disk.

Compressing a File: `gzip` Command

- Alternatively, you can use the `gzip` command to compress files.

```
$ gzip [option] filenames
```

- The `gzip` command performs the same function as the `compress` command, but the `gzip` command generally produces smaller files.
- For example, to compress a set of files, `file1`, `file2`, `file3`, and `file4`, enter the following command:

```
$ gzip file1 file2 file3 file4
$ ls *.gz
file1.gz file2.gz file3.gz file4.gz
```


Note

The compressed files have a `.gz` extension.

Viewing a Compressed File: `gzcat` Command

- Unique to Solaris the `gzcat` command displays files that were compressed with either the `gzip` or `compress` commands.

```
$ gzcat [option] filename
```

- To view the `file1.gz` file, use the following command:

```
$ 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)
```


Note

The `gzcat` command does not change the content of the compressed file. The compressed file remains on the disk in the compressed form.

Uncompressing a File: `gunzip` Command

- The `gunzip` command uncompresses a file that has been compressed with the `gzip` command.

```
$ gunzip [option] filename
```

- To uncompress the `file1.gz` file, use the following command:

```
$ gunzip file1.gz
```

ORACLE

Compressing and Archiving Multiple Files: `zip` Command

- The `zip` command compresses and archives multiple files into a single file in one go.

```
$ zip [option] target_filename source_filenames
```

- To compress `file2` and `file3` into the `file.zip` archive file, enter the following command:

```
$ zip file.zip file2 file3
adding: file2 (deflated 16%
adding: file3 (deflated 26%)
$ ls
file.zip
file2
File3
$
```


By default, the `zip` command adds the `.zip` extension to the compressed archive file if you do not assign a new file name with an extension.

Note: You can run the `zip` or `unzip` command on the command line to view a list of options used with each command.

Viewing and Uncompressing Archive Files: `unzip` Command

- The `unzip` command is used for listing the files and also for extracting the content of a compressed `.zip` file.

```
$ unzip [option] zipfile
```

- To uncompress the `file.zip` archive file, use the following command:

```
$ unzip file.zip
```

Compressing a File: `bzip2` Command

- Alternatively, you can use the `bzip2` command to compress files.

```
$ bzip2 [option] filenames
```

- The `bzip2` command performs the same function as the `compress` command, but the `bzip2` command generally produces smaller files.
- For example, to compress a set of files, `file1`, `file2`, `file3`, and `file4`, enter the following command:

```
$ bzip2 file1 file2 file3 file4
$ ls *.bz2
file1.gz file2.gz file3.gz file4.gz
```

ORACLE

Note

The compressed files have a `.gz` extension.

Uncompressing a File: `bunzip2` Command

- The `bunzip2` command uncompresses a file that has been compressed with the `bzip2` command.

```
$ bunzip2 [option] filename
```

- To uncompress the `file1.bz2` file, use the following command:

```
$ bunzip2 file1.bz2
```

Quiz

The Oracle Solaris `gzcat` command is used for viewing files that have been compressed by using the `compress` command.

a. True

b. False

Answer: a

Quiz

What is the output of the `zip file7.zip file4 file12` command?

a. An error message: The files must be compressed separately, one per `zip` command.

b. `file7.zip`, `file4.zip`, and `file12.zip`: The compressed versions of each file

c. `file7.zip`: The packaged and compressed zip file that contains two compressed files, `file4` and `file12`

Answer: c

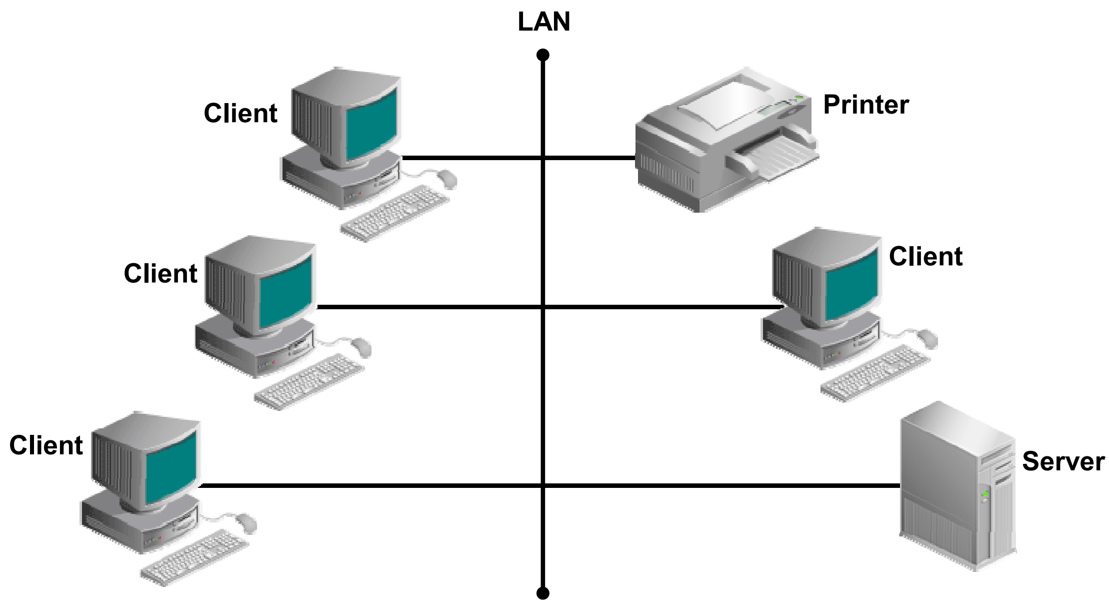
Agenda

- Archiving and retrieving files
- Compressing, viewing, and uncompressing files
- Performing remote connections and file transfers

Introduction to Networking

- A network is a group of computer components connected with each other by communication channels that allow sharing of resources and information.
- A computer system on a network is called a host.
  - The local host is your current working system.
  - A remote host is a different system that you access from your local host.

Layout of a Basic Network



The illustration depicts the relationship between the network and the host. The client is a host or a process that uses services from another program, known as a server.

Remote Login

- A remote login session enables you to access another computer and its resources from your current machine.
- The two parties involved in a remote login are the local host and the remote host.
- For remote login to work, both parties should have the remote login software running, and the remote host should be powered on and connected to the Internet.
- Each time you run the remote login client software on the local host, a new session is initiated.
- Each session is authenticated with an ID and password.
- After the session is authenticated and established, both the local and remote hosts communicate with each other over the network.

SSH for Remote Login

- Secure Shell (SSH) is a network protocol that provides secure encrypted communication between two untrusted hosts over an insecure network.
- SSH allows you to connect and log in to a specified host.

```
$ ssh [-l login_name] hostname | user@hostname [command]
```

- SSH uses public-key encryption to authenticate a remote login session.
- In public-key encryption, the public key is dispatched to all hosts that intend to communicate with the holder of the matching private key.

Note: In Oracle Solaris and Oracle Linux, SSH is installed by default and usable.

Note

The best practice is to set SSH to `not permitted` for root user because any user would be able to `su` into root. This provides a layer of protection for the root user.

Copying Files and Directories Between Remote Systems: `scp` Command

- The Secure Copy (`scp`) command securely copies files and directories between local and remote hosts.
- `scp` is an updated version of an older and less secure utility named Remote Copy (`rcp`).
- `scp` is more secure than `rcp` because `scp` uses SSH, which encrypts both the data and the passwords in transit.

Copying Files and Directories

- To copy files from a local directory to a remote host, use the following command syntax:

```
$ scp [option] SourceFile user@host:directory/TargetFile
```

- For example, to copy the `dante` file from the local directory to the `/tmp` directory on a remote system called `host2`, enter the following command.

```
$ scp dante username@host2:/tmp
```


Caution

Do not use the `/tmp` directory for long-term storage. The `/tmp` directory is emptied each time you reboot the system.

Copying Files and Directories

- To copy files from a remote host to a local directory, use the following command syntax:

```
$ scp user@host:/directory/SourceFile TargetFile
```

- For example, to copy the `dante` file from a remote host called `host2` to the local `/tmp` directory, enter the following command.

```
$ scp username@host2:/tmp/dante /tmp
```


Note

The `SourceFile` file is your original file and the `TargetFile` file is the copy of the original file.

Copying Remote Directories

- The `scp` command with the `-r` option copies directories to and from another system.
- For example, to copy the `perm` directory in the local home directory to the `/tmp` directory on the remote system called `host2`, enter the following command.

```
$ scp -r ~/perm username@host2:/tmp
```


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 path name.

Quiz

Identify the correct command to copy the `/opt/dante` file from a remote host to the `/tmp` directory in your system.

a. `scp username@host2:/dante/tmp`
b. `dante scp username@host2:/tmp`
c. `username@host2/scp dante:/tmp`
d. `scp username@host2:/opt/dante /tmp`

Answer: d

Introducing FTP

- FTP is a network protocol used for exchanging files over a TCP/IP network.
- FTP implements user-based password authentication.
- FTP also allows anonymous user access, where the password is usually a valid email address.
- You can access a remote system for exchanging files using the `ftp` command.

```
$ ftp [option] hostname
```

Note: In Oracle Solaris 11, the FTP client and server software comes installed by default, whereas Oracle Linux uses the more secure `sftp` and `vsftpd` server.

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. Refer to the `ftp` man page for additional information about `ftp` commands.

FTP and SFTP Commands

Following are some of the frequently used `ftp` and `sfpt` commands:

- The `open` command opens a connection with another computer on the network.
- The `get` command transfers a file from the remote system to the local system's current directory.
- The `put` command transfers a file from the local system to a directory on the remote system.
- The `mget` command transfers multiple files from the remote system to the local system's current directory.
- The `mput` command transfers multiple files from the local system to a directory on the remote system.
- The `bye` and `quit` commands enable exiting the FTP environment.

The following are the common `ftp` commands:

- `ascii`: Sets the mode of file transfer to ASCII
- `binary`: Sets the mode of file transfer to binary
- `cd`: Changes directory on the remote machine
- `close`: Terminates a connection with another computer
- `delete`: Deletes or removes a file in the current remote directory
- `help`: Requests a list of all available FTP commands
- `lcd`: Changes directory on your local machine
- `ls`: Lists the names of the files in the current remote directory
- `mkdir`: Makes a new directory within the current remote directory
- `pwd`: Finds out the path name of the current directory on the remote machine
- `rmdir`: Removes or deletes a directory in the current remote directory
- `prompt`: Prompts you to confirm the transfer of each file before completing the transfer. By default, prompting is set to `on`.

Note: You can use `?` to request for help or additional information about the `ftp` commands.

FTP and SFTP Transfer Modes

FTP supports two types of transfer modes:

- American Standard Code for Information Interchange (ASCII) mode
  - ASCII mode transfers plain files such as text files.
- Binary mode
  - Binary mode enables you to transfer binary, image, or any nontext files.

Note

In Solaris 8 OS and 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 nontext files you have to type the `bin` command to ensure complete data transfer.

Transferring Files Using ASCII Mode

The example coded on the notes page establishes an FTP connection from the `host1` system to the `host2` system. After the connection is established, the user changes to ASCII mode. The user then gets the `fruit` file from the `user2` directory on `host2`, stores the `fruit` file in the student home directory on `host1`, and quits the FTP session.

ORACLE


```
$ 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
file.1
file.2
file1
file2
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 Files Using Binary Mode

The example coded on the notes page shows how to transfer a binary file.

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.
$
```

Transferring Multiple Files

The example coded on the notes page establishes an FTP connection from the `host1` system to the `host2` system and transfers multiple files by using the `prompt`, `mget`, and `mput` commands.


```
$ 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
(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 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.
$
```

SFTP

- Secure FTP (SFTP) is an interactive file transfer program, similar to FTP.
- However, unlike FTP, SFTP performs all operations, such as file access, transfer, and management over an encrypted SSH transport.
- Being an extension of the SSH protocol, SFTP uses many features of SSH, such as public key authentication and compression to enforce security.
- Like `ftp`, you can access a remote system for exchanging files using the `sftp` command.

```
$ sftp [option] hostname
```

Note: For information about `sftp`, refer to the man pages.

The practices in this lesson cover file transfers using `sftp`.

Quiz

Which is the most secure command for remotely logging in to another system within the network?

a. `rsh`

b. `ssh`

c. `ftp`

Answer: b

Quiz

Select the two correct `ftp` command syntaxes to end an ftp session.

a. `ftp> exit`

b. `ftp> quit`

c. `ftp> close`

d. `ftp> bye`

Answer: b, d

Summary

In this lesson, you should have learned how to:

- Archive and retrieve files
- Compress, view, and uncompress files
- Perform remote connections and file transfers

Practice 9 Overview: Archiving Files and Performing Remote Transfer

This practice covers the following topics:

- Creating an archive file on a disk
- Viewing an archive file on a disk
- Retrieving archive data from a disk
- Compressing files
- Viewing compressed files
- Uncompressing files
- Establishing a remote login session
- Copying files or directories to and from another system
- Transferring files between systems

You will find the tasks for Practice 9 in your Activity Guide.