UNIX Administration

Course Designer and Acquisition Editor

Centre for Information Technology and Engineering

Manonmaniam Sundaranar University

Tirunelveli

UNIX Administration

CONTENTS

Lecture 1

Introduction to Unix Administration

Objectives

In this lecture you will learn the following

- Understanding the concept of UNIX Operating system
- Knowing about Unix File System
- Able to understand the features of UNIX Operating System
- Understand the UNIX Architecture

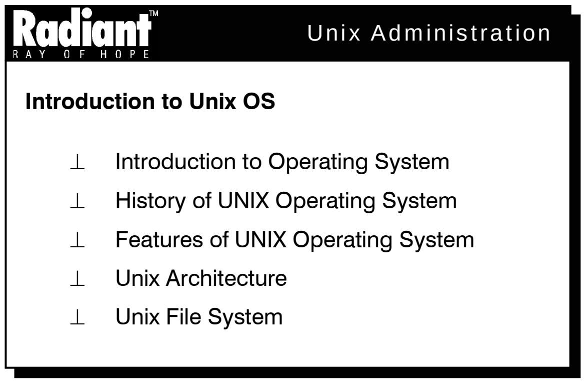
Coverage Plan

Lecture 1

1.1 Snap Shot

System administration encompasses the tasks of planning, installing and maintaining computer systems. The system administrator has a very important role to play.

Introduction to Unix OS



Radiant™ RAY OF HOPE Unix Administration

Introduction to Unix OS

- ⊥ Introduction to Operating System
- ⊥ History of UNIX Operating System
- ⊥ Features of UNIX Operating System
- ⊥ Unix Architecture
- ⊥ Unix File System

1.2 Introduction to Operating System

An operating system is an important part of a computer system. A computer system can be described as a collection of hardware and software components. Hardware components are those components which you can be felt physically, like the Central Processing Unit (CPU), memory, keyboard, hard disk, printer, and so on. The software components are sets of instructions, also referred to as programs, to the hardware (CPU) to perform certain operations to bring out specific results like prepare a letter, send electronic mail etc.. The most important software component of a computer system is the Operating System (OS).

The Operating system is a supervisor program that takes care of coordinating the operations of the machine from the time it is switched on till it is switched off. When the computer is turned on, the OS takes care of all the starting functions that must occur to get the computer to a usable state. Various pieces of hardware are initialized.

The operating system deals with the complexity of manipulating the underlying hardware and allocating logical resource to accomplish the process or task

The main activities of an operating system are:

Running a Program

When a filename is entered on the terminal at the command line, the operating system interprets the command, loads the program into computer's memory and executes the program. When more than one program/process run simultaneously on the system, the operating system takes care of scheduling the CPU for various processes, managing the computer's memory and other related tasks.

Controlling the Input and Output Operations of the Computer

When any file related operations are carried out like deleting a file from the hard disk or saving a file on to the disk, the operating system ensures that the operation is carried out properly without erasing a different file or overwriting an existing file etc. Similarly, it f displays the output on the terminal and accepts input from devices like keyboard or mouse and prints the documents that are specified on the printer connected to the system.

When the computer is shutdown, the operating system ensures that the hardware is shutdown correctly. Without an operating system, no application can run.

1.3 History of Unix Operating System

The Unix operating system has taken thirty years to evolve into today's scenario. Here are some key events that led to its development.

In 1971 the first edition of the Unix server operating system emerged from Bell Labs. Although Linux does not include any Unix code, it is a Unix clone, which means it shares a number of technical features with Unix, which might be considered the forerunner of the open-source operating system. During the 1970s, Unix code was distributed to people at various universities and companies, and they created their own Unix varieties, which ultimately evolved into Sun Microsystems' Solaris, Berkeley's FreeBSD and Silicon Graphics' IRIX.

In 1985 Richard Stallman published his famous "GNU Manifesto", one of the first documents of the open-source revolution. Stallman began working on the GNU operating system in 1983, largely because he wanted to create an open-source version of Unix. (GNU stands for "GNU is Not Unix.") Stallman's Free Software Foundation later created the GNU General Public License, the widely adopted, fully legal "anticopyright" treatise that today allows Linux and other software to remain completely free.

In 1987 Professor Andrew S. Tanenbaum invented Minix, an open-source operating system which is a clone of Unix. Young Linus Torvalds, at the time a computer science student in Finland, was introduced to Minix, and based his plans for Linux on the Minix example.

In **August 1991,** Torvalds announced his plans to create a free operating system on the Minix users newsgroup. He modestly noted in his posting that his OS is "just a hobby. [It] won't be big and professional like GNU." In October, Linux 0.01 was released on the Internet under a GNU public license. In the Minix newsgroup, Torvalds asks his fellow programmers to lend a hand in making the system more workable. He gets enough help to release version 0.1 by December. Over the next several years, the number of Linux developers swelled into the hundreds of thousands and worked to make Linux compatible with GNU programs. Vendors like Red Hat, Caldera and Debian created popular distributions of Linux that bundled the operating system with useful programs and a graphical interface.

In 1997 Torvalds moved to Silicon Valley and went to work at Transmeta.

In **August 1999,** Red Hat completed its initial public offering, making it the first Linux-oriented company to successfully go public. In December, Andover.net, a consortium of Web site resources largely devoted to Linux, and VA Linux, a manufacturer of Linux hardware, had wildly successful IPOs. Linuxcare, a leading Linux service provider, announces alliances with such industry giants as IBM, Dell, Motorola and Informix.

1.4 Features of Unix Operating System

Unix is a comprehensive OS with a number of features and capabilities. Several reasons have been identified for the popularity and success of the Unix System.

Unix operating system is written in "C" language which makes it easier to read, understand, update and port to other machines.

Multi-user OS

Multi-user refers to an OS that allows multiple users to use the system simultaneously. The theory of multi-user system is to approach 100% computer utilization while reducing the cost per user. A single user cannot use the printer, disk, memory or CPU 100% of the time. But multiple users can increase the use of these devices and resources by having an OS that manages the resources for them.

Multi-tasking system

Multitasking refers to an OS that executes multiple tasks simultaneously. Unix refers to a task as a process. A user can run several commands in the background while executing another command in the foreground. When a background task is being executed, the user can continue doing another task e.g., printing a large document can be performed in the background while editing some other document in the foreground.

Portability

Unix is highly portable. Portability is the ability of the software that operates on one machine to operate as efficiently on another, different machine.

Job Control

Job Control on Unix refers to the ability to control which job is to be executed in the foreground, background or suspended. Using Job control to be the productivity of a user can be used by allowing multiple tasks to be juggled back and forth between background, foreground and suspended states.

Hierarchical Structure

Unix uses a hierarchical structure to store and maintain files. This structure allows maximum flexibility for storing information to resemble its real life structure. Multiple users may be grouped by corporate departments. An individual user, you may group the data by project or subject.

Unix Shell

The shell is a very powerful and dynamic Unix utility. It is the primary interface to the OS (Kernel). It can be interactively programmed or used to write scripts that help to solve simple to complex problems.

Pipes and Filters

Pipes and filters contribute to the power of Unix. These enable several commands or utilities to be combined to perform complex functions.

Device Independence

Unix system considers all the devices that are connected to it as files. It hides the machine architecture from the user, makes it easier to write programs that run on different hardware implementations.

System Security:

Being a multi-user OS, Unix offers protection to one user's information from other users. It maintains a list of users who are allowed to access the system and keeps track of the files and resources that each user is authorized to use.

Communication

The Unix system has several built in programs, that enable the user to communicate, transfer files across different Unix systems and between Unix and other OS system.

1.5 Unix Architecture

The Unix system is built up of layers, with each layer representing a building block that can be used to build other building blocks and so on. Most programs and commands supplied with the Unix system can be used in combination with each other to build other tools, and complex mechanisms can be built up from a single line of commands to perform vast assortments of functions.

The hardware at the center of the diagram provides the operating system with the basic hardware services. The operating system interacts directly with the hardware, providing common services to programs and insulating them from hardware specifics.

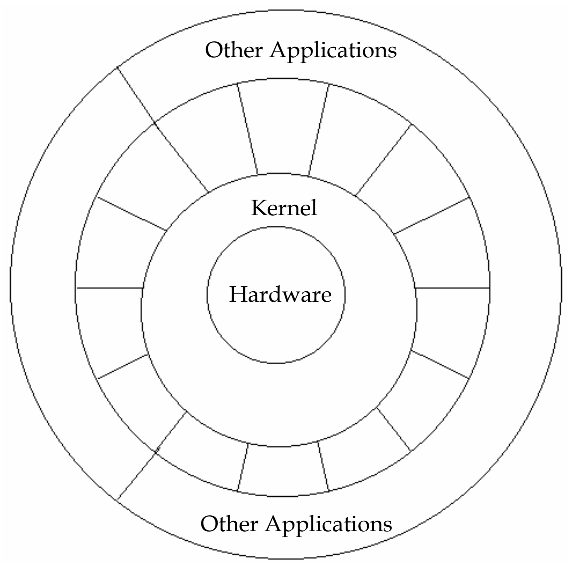


Fig 1.1

The Unix operating system is commonly called as **Kernel**. It provides an interface for all other Unix programs to use the hardware resources. This concept allows the higher level programs to be hardware independent. When the user logs in to Unix communicates with the kernel through the shell program. It does not make itself available for the end user. The Kernel provides the basic services of System initialization, Process management, Memory management, File System management, Communication facilities and Programmatic interface through system calls.

Programs such as the shell and editors (ed and vi) that are shown in the outer layers interact with the Kernel by invoking a well defined set of *system calls*. The system calls instruct the kernel to perform various operations for the calling program and exchange of data between the kernel and the program.

One of the prominent features of the Unix system is its wide variety of powerful utility programs. A utility can be used to locate system information, manage files or the contents of files and manipulate the output of other utilities.

The shell is a utility program that acts as a command interpreter. It is the primary interface to the kernel. The shell is a command language as well as a programming language. As a command language, it can be used to communicate interactively with the Kernel. As a programming language, users can write shell scripts to solve the various problems.

Several programs shown in the figure are in standard system configuration and are known as commands. Over 200 utility programs (commands) are supplied with Unix system. These are also used to solve complex problems.

In addition to utility programs, there are a number of Unix-based application programs like word processors, spreadsheets, database managers and language processors which form the outer most ring in the architecture.

1.6 Unix File System

Information in a Unix system is stored in files, that are similar to ordinary office files.

Each file has a name, contents, a place to keep it and some administrative information such as its size and ownership . A file might contain a letter, or a list of names and addresses, or the source statements of a program, or data to be used by a program.

Files are kept in storage devices, that are usually disks. A number of files can be stored on the disk. To organize these files, Unix operating system divides the disk into various logical units, where each unit can contain a group of related files. These logical units are called ***file systems.***

Unix uses the hierarchical structure to store its files. This structure is referred to as the 'inverted' tree structure from its resemblance to an upside down tree. The file system starts with one main directory called ***root.*** Since the tree is upside down, the root is at the top. The root directory symbolized by '***/***'(forward slash) has multiple directories under it.

The name of a file is given by a path name that describes how to locate the file in the file system hierarchy. A path name is a sequence of component names separated by slash characters; a component is a sequence of characters that designates a file name that is uniquely contained in the previous (directory) component. A full path name starts with a slash character and specifies a file that can be found by starting at the file system root and traversing the file tree, following the branches that lead to successive component names of the path name.

There is no internal structure imposed on the contents of the files. Any structure given by the user can be used.

Features of Unix File System

Hierarchical Structure

Allow grouping of related information and efficient manipulation of these groups.

Dynamic File Growth

Files grow according to the requirements. Disk space is not wasted since only the amount required to store the current contents of the file is only used.

Structureless Files

There is no internal structure imposed on the content of the files. Any structure given by the user can be used.

Security

Unauthorized users can be restricted from using a file.

Device independent

Input and output from a device are processed as if it were in a file. Therefore, programs that process data can also process data to and from a device.

Types of files

A Unix file system may contain six different types of files:

- Regular files
- Directories
- Special files
- Named pipes (FIFO)
- Links
- Symbolic Links

Regular files

A regular file (also known as an ordinary file) contains arbitrary data in zero or more data blocks stored within a file system. These files may simply contain ASCII text , or binary data. Individual applications may store their files in a specific format. There is no structure imposed by the operating system about how a regular file must be formatted.

Data blocks belonging to a regular file may not necessarily reside on the disk in a contiguous order. However, the Unix operating system hides this side effect from the user and presents a file as if it were a contiguous stream of bytes; the user need not be concerned with a file's underlying storage structure.

Unix identifies the files by a unique number called the **index node (inode)** number. A file has only one inode number, although it may have many filenames. The inode numbers are maintained in a directory file along with the related filenames.

Directory files

Directories are a collection of files. For instance, a user may need to group all his project files into one directory. Each directory has a name and each file within the directory has a filename. Directories are special types of files since they provide mapping between the names of files and the files themselves. As a result, the structure of directories defines the structure of the file system as a whole.

The directory consists of a table containing two fields:

inode number filename (to symbolically reference the inode).

Special Files

Special files do not contain data. Instead, they provide a mechanism to map physical devices to file names in a file system. Each device supported by the system, including memory, is associated with at least one special file. Special files have associated software incorporated into the Kernel called device drivers.

There are two types of special files : block-special and character-special. A block-special file is associated with a block structured device such as a disk, which transfers data to the machine's memory in blocks, typically made up of 512, 1024 bytes. A character-special file is associated with any device that is not necessarily block structured. Terminals, system console, serial devices, tape drives are character-special files.

Links

The Unix file system provides a facility for linking files together with different file names. This facility is called linking. The purpose of linking files together is to allow a single program to administer different names. Actually only one copy of data is stored in the file system. The linked files share the same inode number and only a directory entry is made for the file.

Symbolic links

A symbolic link is a data file containing the name of the file to which it is supposed to be linked to. A symbolic link can be created even if the file it is supposed to be linked to does not exist. The advantage of having symbolic link is when the file system has lesser in space but a new software package has to be installed in it, a directory can be made on another file system which is then symbolically linked to the name of the expected installation directory.

With symbolic links, both a directory entry and new inode are created. Additionally, a single data block is reserved for it containing the full pathname of the file it references.

FIFOs (Pipes)

Pipes are used to join two or more Unix processes together allowing the data to flow from one process to another without storing the data on the disk. A pipe file is a special file that buffers up data received in its input so that a process that reads from its output receives the data on a first-in-first-out basis (FIFO). No data is associated with a pipe special file although it uses up a directory entry and inode.

File Naming Conventions

- A filename can contain digits, characters, dot (**.**), hyphen (-), or underscore symbol
- A filename can be in uppercase and lowercase
- A directory can have all characters
- Filename are case sensitive. For instance, a filename **myfile** is different from **MYFILE**
- A filename should not contain white spaces

File System Organization (Directory Hierarchy)

Directories are special files that contain names of other files and sub-directories. Typical directory structure of the Unix system consists of the following directories: -

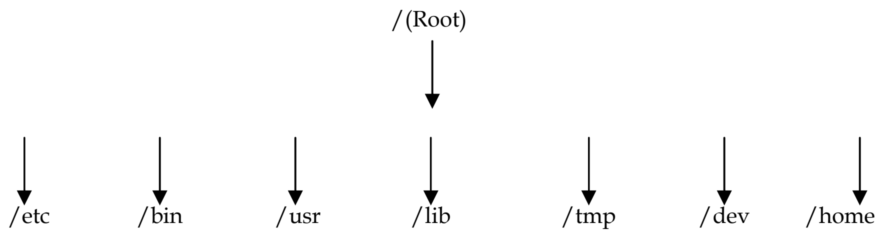


Fig 1.2

The top level directory is called the root directory and is denoted by a single / (forward slash). All the directories and files belongs to the root directory.

Following are the list of standard directory names in the Unix file system.

/ Root directory. This is the parent of all the directories and files in the Unix file system.

/etc System configuration files and executable directory. Most of the administrative, command-related files are stored here.

/bin	Command-line executable directory. This directory contains all the Unix native command executables
/usr	Architecture dependent and architecture independent sharable files
/lib	The library files for various programming languages such as C are stored in this directory
/tmp	This directory is used for the temporary storage of files
/dev	Device directory containing special files for character- and block-oriented devices such a printers and keyboards. A file called null existing in this directory is called the bit bucket and can be used to redirect output to nowhere
/home	This directory contains data that is specific to an individual user

1.7 What is System Administration?

Radiant™ RAY OF HOPE — Unix Administration

- ⊥ Technical Concepts for New Unix System Administrators
- ⊥ Network Centricity
- ⊥ UNIX is Heterogeneous
- ⊥ System Administration Tasks
- ⊥ System Load and Performance
- ⊥ Administration Resources

System Administration involves planning, installing, and maintaining computer systems.
The system administrator takes care of administration.

System Administrator has to install and configure the operating system, add new users, back up the system(s), keep the systems secure, and ensure their running condition.

Installing, running, and maintaining all the major Unix variants is only part of the art of the system administration. There is a significant nontechnical component to being a system administrator, especially in terms of planning, organizational, and people skills.

As computers become more and more pervasive in business, system administration becomes a critical in many organizations. The administrator has to understand the systems that he is responsible for, the users, and purpose.

Technical Concepts for New Unix System Administrators

Unix differs from Windows and Macintosh at a fundamental level. Unix has been originally intended for multiple users running multiple simultaneously program. The phenomenon of a user having personal workstation at a lower cost, running Linux or any other variant of Unix is actually a recent development in terms of the operating system's history. The fact that Unix has been designed for use by more than a single user is reflected throughout the operating system's file systems, security features, and programming model.

Unlike Windows and Macintosh networking is not an afterthought or a recent development for Unix. Support for sharing files, moving from workstation to workstation, and running applications on remote computers that are controlled and viewed on a local workstation is not only intuitive and natural on Unix, but has been more powerful and stable even in the earlier versions of Unix when compared than the latest versions of Windows and Windows NT.

Network Centricity

Networking has become an integral part of Unix. The ability to share files, support network logins, share information about network configuration, and run applications across a network is included in all the major Unix distributions. It is a natural extension of the base operating system, not an application that is designed by an individual vendor, with idea of networking and administration that has to be purchased separately.

When configured to allow networking, anything that can be done at the *console* (main keyboard and monitor) of a Unix system can also be done at another system through a network connection. (They are referred to as *remote nodes* or *remote systems.*) Actually many server systems, such as Web servers and file servers, have consoles with very limited capabilities (such as a text-only terminal), and are deliberately designed with the idea of doing as much administration as possible from remote nodes outside the data center.

Unix is Heterogeneous

Unix is being frequently criticized for lacking consistency between versions, vendors and even applications. Unix is not the product of any single corporation or group, and this does have a significant impact on its personality. Linux is probably the ultimate expression of UNIX's collective identity. After Linus Torvalds has created the Linux kernel and announced it to the Internet, people from all over the world began to contribute to the Linux Operating System. Obviously, Linux reflects a different views on how computers should work. Unix does too.

Unix has historically been divided into two major variants: AT&T's Unix System V and The University of California's BSD Unix. Most of the major vendors are now moving toward System V system, but many BSD extensions will always remain.

System Administration Tasks

System administration can generally be divided into two broad categories: supporting users and supporting systems.

Supporting Users

Users are the customers. The network would be a single computer, without them probably running a frivolous application like Doom and generating no business or creating no new sales.

Creating User Accounts The most fundamental thing that an administrator can do for a user is to create an account. Unix accounts are contained in the /etc/passwd file with the actual encrypted password being contained in either the passwd file or the /etc/shadow file if the system implements shadow passwords.

Supporting Systems

Unix is supports the systems. Systems have to be built, backed up, upgraded, and fixed.

Adding Nodes A frequent system administration task is adding new nodes to the network. It is also one of the tasks of the system administration that can truly benefit from some planning and insight.

Not all the systems are created equal, and not all of them are used for the same purpose. Spending some time in understanding what the network is really used for and then applying that to the systems is key to network planning. Workstations should have well defined roles and should be configured in accordance with those roles.

When the systems are designed and evaluated, some of the questions that an administrator can ask are:

- Will users be able to access all or some of the systems? Do users need to access more than one system? Are there system that users should never access?
- What network file systems will each workstation need to access? Are there enough that automount would help?
- What network services, such as telnet, remote logins, sharing file systems, and e-mail, do workstations need to provide? Can each service be justified?
- What networks will workstations need to access? Are there networks that should be inaccessible from others?

These questions should help the System Administrators in developing a profile for each workstation. Following a profile makes workstations easier to build, maintain, and troubleshoot. Moreover it makes them more reliable since they tend to be less complex.

Backups Files may get corrupted, lost, accidentally overwritten, or deleted. Backups safeguard the files from such situations. Unix provides several backup tools.

System Load and Performance

The system administrator monitors the system load and performance.This is yet another area where planning and anticipation become inevitable.

The system administrator should carry out daily monitoring of usage statistics, especially on mission – critical systems and systems with which users interact regularly. These statistics can be gathered using automated scripts. Disk usage, CPU utilization, swap, and memory. are to be monitored regularly.

Administration Resources

A system administrator needs minute details and help to carry out the job of system administration successfully.

The Manual Pages

The famous rejoinder RTFM (Read The Fine Manual) refers to the manual pages installed on (hopefully) every Unix system. The man pages, as they are frequently called, contain documentation and instructions on every Unix command, C function call and data file on the system.

The man command searches for documentation based on a command or topic name. So the command

```
man ls
```

provides the documentation on the **ls** command.

Actually, the man pages have a sophisticated structure. The pages are divided into sections, with some of the sections being further divided into subsections.

The section layout resembles the following

- User commands—commands like **ls, tar, and cpio**
- System calls—C programming functions that are considered system calls, like opening and closing files
- C library functions—C programming functions that are not considered system calls, like printing text
- File formats—descriptions of file layouts, such as **hosts.equiv** and **inetd.conf**
- Headers, Tables and Macros—miscellaneous documentation, such as character sets and header files, not already covered
- Games and Demos—games and demo software. (Even Doom for Unix has a man page!)

This is a generalized table of contents. BSD and System V started out with slightly different section schemes, and vendors tend to add their own sections and make their own "improvements."

1.8 Short Summary

⊥ System Administration involves planning, installing, and maintaining computer systems

⊥ The ability to share files, support network logins, share network configuration information, and run applications across a network is included in all of the major Unix distributions

⊥ unix is not the product of any single corporation or group, and this does have a significant impact on its personality

⊥ Files get corrupted, lost, accidentally overwritten, or deleted. The only protection against these situations is backups, since Unix does not have an undelete command

⊥ The Kernel is a collection of software that manages the physical and logical resources of the computer It controls the
  - Allocation of memory
  - Storage and peripheral devices
  - Scheduling and execution of processes or tasks

⊥ The physical resource are controlled by means of software modules , referred to as device drivers

⊥ A device driver is nothing but software modules, that communicates with hardware devices and control their operations. Here each device has a unique driver that is provided with the hardware and identified by the hardware manufacturer model, hardware version and its vendor name

⊥ The logical resource includes process and memory

⊥ A process is a task or program. The Kernel will maintain the following parameters as its internal data structure for every process. It

⊥ Controls the processes

⊥ Controls the scheduling , execution and termination

⊥ One of the most important services of kernel is memory management and it includes the following factors.

⊥ Keep track of resource

⊥ Allocate memory for the resource

⊥ Reallocate memory for other resource either reclaiming or terminating

⊥ Inter-Process Communication (IPC) involves handling the cooperative or interactive communication process.

⊥ The shell is a software module that provides an Interface or Interpreter between the user and the Kernel.

1.9 Brain Storm

1. Brief about the roles of system administrator.
2. Mention how the system load performance is measured.
3. Is Unix operating system heterogeneous? What does it mean ?
4. What is the use of Network Centricity?
5. What are the various tasks of system administration?

Lecture 2

Unix Refresher

Objectives

In this lecture you will learn the following

- About Unix Commands
- Makes the user conversant with UNIX directory commands and their usage.

Coverage Plan

Lecture 2

2.1 Snap Shot

This session introduces UNIX commands that are simple and often used. It session makes the user conversant with the UNIX directory commands and their usage. It also deals with creating directories and manipulating its components. It also shows how to work with files. Various file manipulation commands and file printing commands are also dealt with. The session also discusses the concept of standard input, output and redirecting the input and output to files. It also deals with combining simple UNIX commands to build complex commands using the concept of pipes. Many powerful utilities of UNIX that perform filtering of data are also dealt with. This session also introduces an important concept of UNIX called process. The commands that help in working with various processes are discussed.

Several people can be using a UNIX system at the same time. To enable the system to know who the user is and the resources that can be used, the users must identify themselves. This process of identifying is known as login in. Before the system can be accessed, the System Administrator must configure the computer. Every user needs a unique identification called the user name, which is the name through which the user is identified by the system. To avoid others from using the name, the system protects the user's account by providing a password. The user is responsible for keeping the password secure.

Login

Once the user name and password are provided, the user can access the system. The system prompts for the user name by printing login.

login:

The user should respond to this prompt by typing a valid username (supplied by the system administrator) called the user_id. On pressing the **<Enter>** key, the next line prompts for a password:

Password:

Password is a sequence of letters and digits that is used to verify if the user is allowed to use the user_id. Initially, the password is supplied by the system administrator. Later the user can change it to some secret key. It is password that identifies the authorised users of a system.

After the valid username and password have been entered, the # prompt is displayed on the screen. The # prompt indicates that theUNIX system is ready to accept commands.

Example

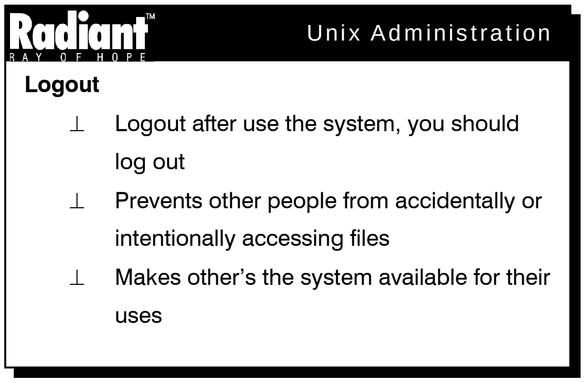
```
UNIX(r) System V Release 4.0 (radiant)

login: root
Password
Last login: Sat May 15 12:00:00
#
```

Root User

There is a user id called "root" or super user. This user has special privileges. The root user has access to all parts of the UNIX operating system. There are no files that cannot be read by the super user, there are no portions of the file system inaccessible to the super user and there are no UNIX commands unavailable to the super user. The super user controls all aspects of UNIX system usage and configuration. The super user is entitled to the special prompt: #.

2.2 Logout


Unix Administration

Logout

- Logout after use the system, you should log out
- Prevents other people from accidentally or intentionally accessing files
- Makes other's the system available for their uses

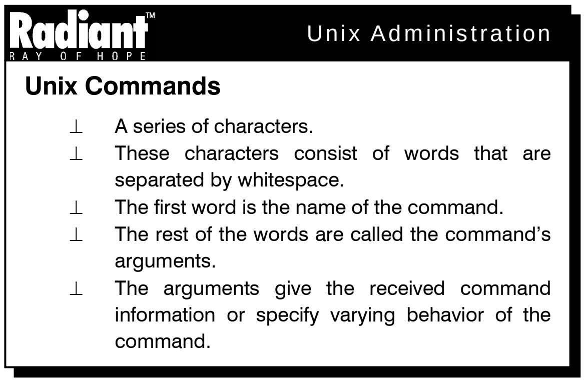
When the user has completed using the system, he/she user should log out. This will prevent other people from accidentally or intentionally getting access to files. It will also make the system available for other users.

The normal way to log out is to type exit. Another way of logging out is to type the end-of-file character (typically Control +D) as shown below:

```
# exit
# ^D
```

Note : It is always safe to logout when a user no longer wants to work on the system.

2.3 Unix Commands


Unix Administration

Unix Commands

- A series of characters.
- These characters consist of words that are separated by whitespace.
- The first word is the name of the command.
- The rest of the words are called the command's arguments.
- The arguments give the received command information or specify varying behavior of the command.

The # prompt is called the command prompt or shell prompt.

A UNIX command is a series of characters that are being typed. These characters consist of words that are separated by whitespaces. Whitespace is the result of typing one or more Space or Tab keys. The first word is the name of the command. The rest of the words are called as the arguments of the command. The arguments give the command information that it might need, or specify varying behavior of the command. To invoke a command, the command name must be typed, followed by the arguments. The shell collects all the characters that are typed until the enter key is pressed and interprets them.

date

This command is used to display the current date and current time of the system.

Syntax

```
date  ["+<string> <options>"]
```

The following options can be used in the date command

%D	Displays the date in MM/DD/YY format
%d	Displays the day of the month(01-31)
%m	Displays the month (01-12)
%y	Displays the year

The following options can be used with the date command to display the weekdays and months in the abbreviated format:

%a	Displays abbreviated weekdays(Sun-Sat)
%A	Displays abbreviated weekdays (Sunday-Saturday)
%b	Displays abbreviated months (Jan-Dec)
%B	Displays abbreviated months (January-December)

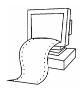
The following options can be used with the date command to display the current time.

%H	Displays the hour
%M	Displays the minutes
%S	Displays the seconds
%I	Displays the IST time
%r	Displays the time with meridian (AM/PM)
%n	Displays the output in the newline

Practice 2.1

The following example shows how to display the current date and time.

```
# date
```



```
Sat May 15 14:09:01 GMT 1999
```

As seen above, the date command gives the day of the week, month, day, time (24 hour clock, Greenwich Meantime) and the year.

Practice 2.2

The following example shows how to display the date with strings.

```
# date “+ Current Date : %D”
```

```
Current Date : 09/22/00
```

In the above example the date is displayed along with the string “Current Date”.

Cal

The cal command generates a simple calendar as the output. By default, the output is the calendar for current month.

Syntax

```
cal  [ [month] year]
```

where **month** specifies the month to be displayed, represented as a decimal integer from 1 (January) to 12 (December). The default is the current month. **Year** specifies the year for which the calendar is to be displayed, represented as a decimal integer from 1 to 9999. The default is the current year.

Practice 2.3

ng example shows the usage of the cal command.

```
# cal
```

```
May 1999

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
```

In this example the calendar for the current month is displayed as no options have been specified.

Note : Year must be entered as a four-digit number. calendar for the year 99 A.D., and not the year 1999. If only one argument is given to the cal command, it is considered as the year and not the month. For example: cal 07 will display the calendar for the year 07 A.D. and not the month July.

Finger

The finger command displays a detailed list of the user information. If a user name is specified, information only for that user is displayed. If no user name is given, information for all users currently logged in to the system is displayed.

Syntax

```
finger [-lmsp]
```

Options are:

-l Produces a multi-line format displaying all of the information described for the -s option as well as the user's home directory, home phone number, login shell, mail status, and the contents of the files ".plan" and ".project" and ".forward" from the user's home directory.

-m Prevents matching of user names. User is usually a login name; however, matching will also be done on the users' real names, unless the -m option is supplied. All name matching performed by finger is case insensitive.

-p Prevents the -l option of finger from displaying the contents of the ".plan" and ".project" files.

-s Finger displays the user's login name, real name, terminal name and write status (as a "*" after the terminal name if write permission is denied), idle time, login time, office location and office phone number. Login time is displayed as month, day, hours and minutes, unless more than six months ago, in which case the year is displayed rather than the hours and minutes.

The finger command displays in multi-column format the following information about each logged-in user:

+ user name + user's full name
+ terminal name (prefixed with a '*' (asterisk) if write-permission is denied)
+ idle time
+ login time
+ host name, if logged in remotely

The following options can be used

-f Suppresses the printing of the header line (short format).

Practice 2.4

The following example shows the usage of the finger command.

```
# finger root
```

```
Login name: root                        In real life: Super-User
Directory: /                            Shell: /sbin/sh
On since May 15 13:37:52 on pts/0 from 80.0.0.98
25 minutes Idle Time
Mail last read Fri May 14 20:22:14 1999
No Plan.
```

The above example shows that the finger command displays information about the user such as login name, directory, idle time, time at which the last mail was received etc.

Id

This command displays the user ID, user name, group ID, group name. The system uses this user ID to identify the files owned by the user. The group ID works the same except it is used for the group level identification.

Syntax

```
id   [username]
```

If no user operand is provided, the id utility will write the user and group IDs and the corresponding user and group names of the invoking process on the terminal.

The following options can be used in id command

-g displays the output of group id

-u displays the output of user id

Practice 2.5

The following example shows the usage of the id command.

```
# id
uid = 102(sunil) gid = 40(radiant)
```

It can be seen that the id command displays the user id and the group id. In brackets are the username and the group name respectively.

Man

The man command enables information to be found in the online manuals by specifying a keyword. The manual entry is called a man page, even though it is often more than one page long. There are common sections to man pages. Depending on the command, some or all of the sections may be present. At the start of the man page is the Name. This is usually a one-line that give the command's name along with a phrase describing what it does. Next is the Synopsis, which gives the command's syntax including its arguments and options. In the Synopsis, if any argument is enclosed in square brackets ([ ]), then the argument is optional. If two elements of the syntax are separated with a vertical bar (|), then either one or the other (but not both) of the items is allowed. The other sections are Description, Files and See Also.

For example, the following command would display the man pages for the pwd commnad

```
# man pwd
```

More than One Command on a Line

More than one command can be typed on a line by separating the commands with a semicolon. For example, both the current time and the user ID can be found out by typing the date and id commands on the same line as shown below.

```
# date; id
Mon Jan 18 20:39:56 GMT 1999
uid=102(sunil) gid=40(radiant)
```

Who

'who' prints information about users who are currently logged on. If given no non-option arguments, 'who' prints the following information for each user currently logged on: login name, terminal line, login time, and remote hostname or X display. The who command enables you to find out the users on the system.

Syntax

```
       who [OPTION]  [am i]
```

Options

-H	–	heading print line of column headings
-i	–	adds user idle time as HOURS:MINUTES, or old
-q	–	counts and prints only the login names and the number of users logged on.

Who am i

It displays the current username, terminal type, date and login time

Practice 2.6

The following example shows the usage of the who command.

```
# who
```

```
root       pts/0        May 15 13:37
sunil      pts/1        May 15 16:44
ajay       pts/2        May 15 17:12
oracle     pts/3        May 15 17:12
```

The above example displays the list of users' information.

System Commands

The uname command displays the name of the operating system.

The logname command displays the login name of the current user.

The hostname command displays the host name of the unix operating system.

2.4 Short Summary

- Logout prevents other people from accidentally or intentionally accessing files
- Logout makes other's the system available for their users
- Unix command is a series of characters that are being typed
- The argument give the received command information or specify varying behavior of the command
- The shell collects all the characters that are typed until the enter key is pressed and interprets them

2.5 Brain Storm

1. What is the command to logout?
2. What is the output of id command?
3. What is the output of finger command?
4. What is the command to view the help of each command?
5. How will you type more than one command on command prompt?

Lecture 3

Files and Directories

Objectives

In this lecture you will learn the following

- Able to create directories and manipulating components
- Shows how to work with files
- Various File manipulating commands
- Knowing about File printing commands

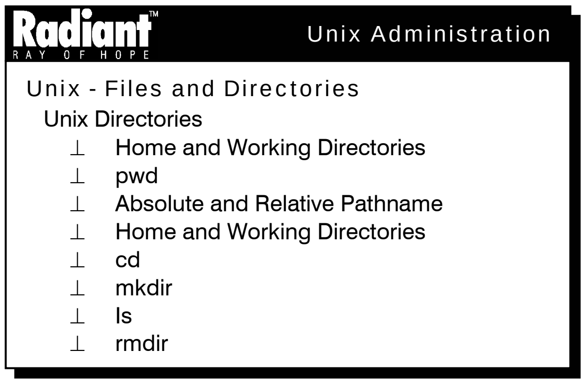
Coverage Plan

Lecture 3

3.1 Snap Shot

This lecture deals with creating directories and manipulating its components. It also shows how to work with files. Various file manipulation commands and file printing commands are also dealt with.

3.2 Unix - Files and Directories



Radiant™ RAY OF HOPE — Unix Administration

Unix - Files and Directories

Unix Directories
- ⊥ Home and Working Directories
- ⊥ pwd
- ⊥ Absolute and Relative Pathname
- ⊥ Home and Working Directories
- ⊥ cd
- ⊥ mkdir
- ⊥ ls
- ⊥ rmdir

Unix Directories

Directories provide a convenient means of organizing files. Since, in the UNIX system, the file system has a hierarchical structure, a directory can further contain sub directories.

Home and Working Directories

The directory in which the user is positioned on logging onto the system is the user's home directory. The administrator creates this directory when the user account is created. The system is notified about this directory and the user is automatically positioned at his/her HOME directory on logging onto the system.

At any point of time, the user will be positioned at a particular directory. This directory is referred as the current working directory.

pwd

pwd command displays the absolute path of the present working directory.

Syntax

```
pwd
```

For example, consider

```
# pwd
/user/sunil
```

The output from this command verifies that the current working directory is /user/sunil.

Absolute and Relative Pathname

In a UNIX file system, a file is identified by its exact location in the directory structure. A path name represents the path to be followed from root through the directory tree to locate a particular file. There are two ways by which a file can be accessed:

- Absolute path name
- Relative path name

Absolute pathname: It is the complete path name from the root that UNIX must follow to reach a particular file. Absolute path name starts with a slash (/). For example, to access the file **myfile** residing under the sub-directory **files** in the home directory, **/usr/sunil,** the notation is as follows

```
/usr/sunil/files/myfile
```

The initial slash (/) refers to the root directory. The following slashes separate the names of subdirectories. The final slash denotes the actual file name.

Relative Pathname: Absolute pathnames could be tedious to type if they are very long. In such cases a relative pathname can be used. This is the pathname that is shortened in relationship to the present directory position. Relative path name is represented by a dot (.).

If the current working directory is /user/sunil, a particular file can be accessed relative to the present directory. For example, to access **myfile**, instead of starting the search from the root, relative referencing starts from the present directory to reach **myfile** as shown below.

```
./files/myfile
```

The dot represents the present directory. The following slashes separate the subdirectories and the final slash denotes the actual file name.

cd

The cd command enables the user to change from the present working directory to a new directory.

Syntax

```
cd  [directory]
```

where **directory** is the name of the directory to be changed to.

Let us assume the present directory is `/user/sunil`

To move to another directory, the cd command has to be used with the path name as follows

```
# cd /usr/ajay
```

Either the absolute pathname or the relative pathname can be used to change directories.

Note:
`cd` cannot perform the requested directory change if it does not exist .
`cd` without any argument will always take the user to his/her HOME directory.
`cd` ../ takes the user to the parent (previous directory) i.e. one level up.
`cd` ../../ takes the user to the parent's parent directory i.e. two levels up.

mkdir

The mkdir command is used to create new directories, thus building a hierarchy of directories to maintain files in an orderly manner. If there are hundreds of files, it is always better to organize files based on the information that they contain. For example, all files containing the personal information of the employees can be kept under one directory, say, personal. This helps in locating files if they are categorized into subdirectories.

Syntax

```
mkdir  [ pathname]
```

where pathname **is the path of the directory to be created.**

Practice 3.1

The following example shows the usage of the mkdir command.

```
# mkdir salary
# cd salary
# pwd
```

```
/usr/sunil/personal/salary
```

The above example shows how to create a directory from the current directory (personal). In this way, directories within directories can be created.

Note: mkdir command only creates a new directory. It does not change the current directory. More than one directory can be created at a time.

ls

ls command can be used to display the names of files and directories. This utility is used to know the files and subdirectories that exist within the directories. Different options can be used with the ls command to list the contents in different formats.

Syntax

```
ls [options]
```

The following options can be used with ls command

-a	Lists all files, including the hidden file
-C,-x	Multi-column output with files sorted down the in column wise
-F	Puts a slash (/) after each filename if the file is a directory, an asterisk (*) if the file is an executable, and an at-sign (@) if the file is a symbolic link.
-i	For each file, prints the i-node number i n the first column of the report.
-l	Lists in long format, giving mode, ACL indication, number of links, owner, group, size in bytes, and time of last modification for each file etc.
-R	Recursively lists subdirectories encountered.
-t	Sorts by time stamp (latest first) instead of by name. The default is the last modification time.

Practice 3.2

The following example shows the usage of the ls command.

```
# pwd
/usr/sunil
# ls
```

```
example1   example4    example7    files    typescript
example2   example5    example8    login
example3   example6    example9    personal
```

In the above example all the files and directories that exist within the /usr/sunil directory have been listed.

Note : ls command lists files in an alphabetical order.

rmdir

The **rmdir** command can be used to remove directories when they are no longer needed. Before removing a directory it should be ensured that it does not contain any files i.e. it should be empty. If there are files in the directory when **rmdir** is executed, the directory will not be removed.

Syntax

```
rmdir [ [options] dir-name ]
```

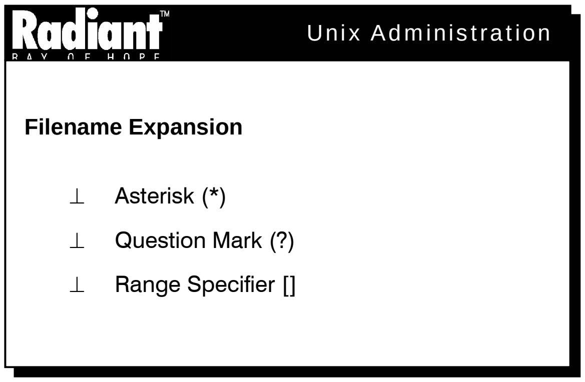
For example,

```
# cd /user/sunil/fruits/
# rmdir apple
```

The directory apple is removed. The user has to change to the corresponding parent directory, only then the files can be deleted.

Note : The directory to be removed must be empty. The empty directory has two entries, the .(dot) representing itself and the ..(dot dot) representing the parent directory. More than one directory can be removed at a time.

3.3 Filename Expansion


Radiant™ RAY OF HOPE Unix Administration

Filename Expansion

⊥ Asterisk (*)

⊥ Question Mark (?)

⊥ Range Specifier []

One powerful feature of the UNIX system is *filename expansion.* It enables to work with files collectively. The group of filename-matching characters is represented by wildcards. Wildcards allow multiple files to be manipulated at the same time using a single command. Wildcards are a kind of shorthand that allows similar files to be specified without having to type multiple names.

There are three types of UNIX wildcards: ***, ?, [].** The shell expands these wildcards, substitutes a group of valid filenames and these filenames are then given to the respective commands.

*	Matches zero or more characters
?	Matches exactly one character
[ ]	Matches any one of the characters in the given range

Asterisk (*)

The * is interpreted as a set of zero or more characters. It provides an easier and quicker way to search directories and access files. The following command lists all files which begin with the letter '**a'.**

```
# ls        a*
acs      ajay.txt
```

Question Mark (?)

The **?** is interpreted as a single character. It can be used to match only one occurrence of the character. The following command lists all files with **'exam'** as the first two characters followed by any single character.

```
# ls exam?
exam1   exam2
```

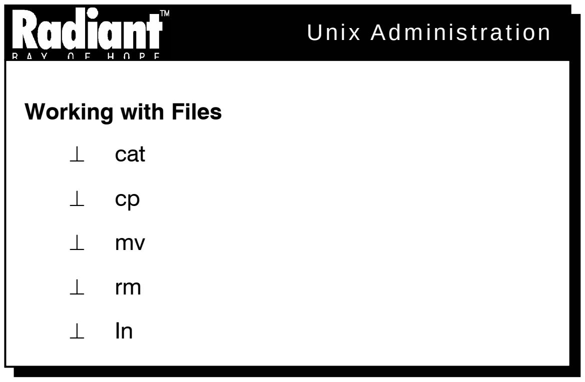
Range specifier []

The [ ] can be used to specify a range of characters which matches either occurrence of the character.

The following command lists all the files starting with any one of the alphabets and ending in **'t'.**

```
# ls [a-z]*t
```

3.4 Working with Files



Radiant™ RAY OF HOPE Unix Administration

Working with Files

- ⊥ cat
- ⊥ cp
- ⊥ mv
- ⊥ rm
- ⊥ ln

The UNIX system provides many tools that enable to work easily with files. Among these tools are the commands that enable to create new files, copy files, remove files, move files between directories, examine the contents of the files, and so on. In this session we will learn to use some of these commands.

cat

The **cat** command can be used for the following purposes:

- To create a new file
- To display the contents of an already existing file

Syntax

```
cat [ filename]
```

where **filename** is the name of the file to be created or displayed.

The following example shows how to create a new file **newfile** using the cat command.

```
# cat > newfile
This is my new file created by cat command
^D
```

Pressing ctrl-d marks the end of file.

The contents of the file **newfile** can be viewed as shown below.

```
# cat newfile
This is my new file created by cat command
```

Note the difference between two Commands. One is used to create a file and the other to display its contents.

Redirection symbols that are used along with the cat command are as follows:

>	Used to create a file
>>	Used to append without overwriting the information
<	Used to input the file to commands

cat command can be used to display the contents of more than one file by giving the filenames separated by a space.

cp

A copy of a file may be required for backup purposes or the user may want to use an existing file as the basis for a new document. If the original file is accidentally removed, it can be restored from the backup. These tasks are accomplished with the cp command.

Syntax

First Form

```
cp  file1  file2
```

where **file1** is the source file, **file2** is the target file.

Second Form

```
cp  file1 file2 file3 …. d1
```

where **file1, file2, file3..** are the source files, **d1** is the target directory.

Third Form

```
cp –R  s1 d1
```

where **s1** is the source directory , **d1** is the target directory, **-R** copies all files and subdirectories.

Practice 3.3

ng example shows how the cp command is used to copy the contents of one file into another.

```
# cp newfile file3
#
```

This will cause the file named **newfile** to be copied into the file named **file3.** The **#** sign after the cp command indicates that the command has been executed successfully.

Practice 3.4

The following example shows how the cp command can be used to copy files into a directory.

```
# cp file1 file2 file3 personal
# ls personal
```

```
file1   file2   file3   salary
```

From the above example it can be seen that the files **file1,file2,file3** have been copied to the directory **personal.**

Note: If the target file is an ordinary file and it already exists, its contents are erased and are overwritten with the contents of the source file. If the target file is a directory, the source file is copied to that directory with the same name as the source. With one cp command, only one file can be copied to another file but one or more files can be copied to a directory at the same time.

mv

There are many instances where a file may have to be moved from one directory to another: The mv command allows for the movement of files in UNIX. The **mv** command can be used for three purposes:

- To rename a file with a new name
- To move one or more files to a different directory
- Rename a directory with a new directory name

Syntax

First Form

```
mv file1  file2
```

where **file1** is the old filename, **file2** is the new name

Second Form

```
mv file1 file2 ….. d1
```

where **file1, file2, ….** are the names of the files to be moved, **d1** is the destination directory

Third Form

```
mv d1 d2
```

where **d1** is the old directory name, **d2** is the new directory name.

In the following example the files **file1, file2** are moved to the directory **newpersonal.**

```
# mkdir newpersonal
# mv file1 file2 newpersonal
```

Note : The new and the old files may have the same names provided they reside in different directories. If the name of the new file is the same as that of an existing file, its contents are lost.

rm

Unwanted files can clog up a hard disk, slowing it down and making file management chores unnecessarily complicated. It is a good practice to regularly go through the subdirectories and remove the files that are not required. **rm** command is used to delete files that are no longer needed.

Syntax

```
rm [option] (filename)
```

where **`filename`** is the name of the file to be removed

Option can be one or any combination of the following:

-f	Forcefully removes a file even if it is write-protected
-i	Interactively asks for confirming the deletion of the files
-r	Removes a directory even if it is not empty

In the following example the file file2 in the newpersonal directory is removed.

```
# rm newpersonal/file2
```

ln

The ln command is used to create one or more links to a file. A link is another name for the same filename, having the same physical storage and same inode number. This means that more than one file can point to the same physical storage.

Syntax

```
# ln filename1 filename2
```

where **`filename1`** is the name of the file on which a link is being created, **`filename2`** is the link created on filename1.

Consider the following

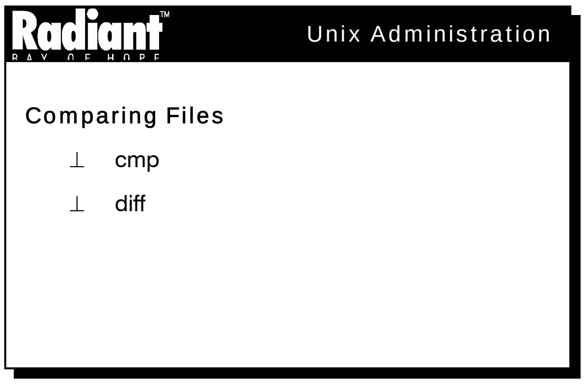
```
# ln newfile file1
```

This will create a link for newfile and the name of the link is file1. The number of links a file has can be seen from the long listing of files.

For creating a symbolic link, the –s option has to be used with the ln command:

Note : ln command gives more than one name to a file but the physical storage of the file is the same. The cp command makes two files with different physical storages. Inode numbers of all hard linked files are the same. A symbolic link can be created even when the file on which the symbolic link is created is not present.

3.5 Comparing Files



Unix Administration

Comparing Files

- ⊥ cmp
- ⊥ diff

Normally, many new files are created during the course of updating older files. Many versions of the same file with slight differences in the content are maintained. At some point of time, it becomes very difficult to keep track of the files and it becomes essential to cleanup the directories by removing the unwanted older versions.

There are utilities in UNIX that can be used to compare the contents of two files to see if they are same. If they are not, the nature of the difference can be determined.

cmp

cmp command is used to compare two files. It compares two files and displays the first instance where the files differ. If there is no difference, cmp returns no output.

Syntax

```
cmp file1 file2
```

where **file1** and **file2** are the files to be compared.

Practice 3.5

The following example shows the usage of the cmp command.

```
# cat > myfile
This is a demonstration of cmp command
^D
# cat > yourfile
This is the demonstrations of cmp command
^D
# cmp myfile yourfile
```

```
myfile yourfile differ: char 9, line 1
```

In the above example the cmp command reports that the two files differ in the ninth character of the document, located in line 1.

Note : cmp indicates only the first character at which the files differ. cmp reports the difference on a character-by-character basis.

diff

Using cmp it can only be known if the files compared are different. It neither shows the extent to which the files are different nor how different they are. The **diff** command compares two files for differences. It determines which lines must be changed to make the two files identical. The diff command scans two files and indicates editing changes that must be made to the first file to make it identical to the second file. Editing may include adding a line, deleting a line, changing a line, and so on.

Syntax

```
diff file1 file2
```

where **file1** and **file2** *are the files to be compared.*

Practice 3.6

The following example shows the usage of the cmp command in comparing two files – file1 and file2

```
# cat file2
diff
login
kill
mv
ln
more
pg
^D
# cat file1
diff
id
sh
mv
ln
^D
# diff file1 file2
```

```
2,3c2,3
< id
< sh
—-
> login
> kill
5a6,7
> more
> pg
```

From the above example it can be seen that lines beginning with < are found only in the first file and lines beginning with > are found only in the second file. The dashed line separates the two lines that appear in the same place in the differing files. The numerals indicate exactly where and how the differences occur.

In the above output, the second and the third lines from file1 have to be changed with second and third lines of file2 to make the two files agree.This is indicated by 2,3c2,3.

Similarly, 5a6,7 indicates that the lines 6 and 7 of file2 needs to be appended to file1 to make them equal.

Likewise sign `'d'` can be used to delete lines from file1 if these lines do not exist in file2.

Note : diff command makes changes in the first file to make it resemble second file. diff command is used to compare files on a line to line basis.

3.6 Printing Files

Radiant™ RAY OF HOPE — Unix Administration

Printing Files

- lp
- cancel
- lpstat

In certain cases, it may be required to take a hard copy of the documents that have been created and that are being stored in the hard disk. This requires the document to be printed on a sheet of paper. For this, a printer is to be attached to the system. The system should request the printer to print the required document. It is the task of the system administrator to setup the printer.

lp

lp spools the output to the printer queue. Spooling is the process of sending the output to a temporary storage area for later processing by the printer.

Syntax

```
lp  filename
```

where filename is the name of the document to be printed.

The print jobs are placed in a print queue as printer requests. The request ids consist of destination printer name and a sequence number. The following example shows that the file myfile is sent to the printer 'hplj' for printing and the request id is hplj-15.

```
# lp myfile
request id is hplj-15 (1 file)
```

cancel

Cancel command removes or cancels print requests made using the lp command.

Syntax

```
cancel request_id
```

where **request_id** is the ID number given by the system when a lp request is placed.

The following example shows how to cancel all queued requests that have been queued up in the printer hplj.

```
# cancel hplj
```

lpstat

The print requests are spooled and they may not be performed immediately. The `lpstat` command to display the current status of all line printers.

Syntax

```
lpstat [options]
```

Practice 3.7

The following example shows the usage of the lpstat command.

```
#lpstat
```

```
hplj-14 root   1524   May 10 17:34
hplj-15 sunil  1024   May 11 11:10
```

Here `lpstat` indicates that there are two requests in the print queue. The first request is with ID hplj-14 initated by root of size 1524 blocks on May 10th 17:34 Hrs and the second request is with ID hplj-15 initiated by sunil of size 1024 block on May 11th at 11:10 Hrs.

3.7 Short Summary

- Unix Directories provides a convenient means of organizing files
- The directory in which the user is positioned on logging onto the system is the user's home directory
- Absolute pathname is the complete path name form the root that UNIX must follow to reach a particular file
- Relative pathname is shortened in relationship to the present directory position.
- A group of filename-matching characters is represented by wildcards
- Wildcards allow multiple files to be manipulated at the same time using a single command
- *cmp* command is used to compare two files and displays the first instance where the files differ.
- Spooling is the process of sending and output to a temporary storage area for later processing by the printer.

3.8 Brain Storm

1. Give a notes on UNIX files and directories?
2. What are the two ways by which a file can accessed?
3. Which command is used to display the files and directories?
4. What is the use of mv command and purpose of use it?
5. By which command you can compare two files?
6. Using date command list the options to display the current time

Lecture 4

Working with Pipes & Filters

Objectives

In this lecture you will learn the following

- Able to combine simple UNIX commands to build complex commands using the concept of pipes
- About Filters and its command
- Knowing about Unix process and it types
- Able to switch between processes

Coverage Plan

Lecture 4

4.1 Snap Shot

This lecture discusses the concept of standard input, output and redirecting the input and output to files. It also deals with combining simple UNIX commands to build complex commands using the concept of pipes. Many powerful utilities of UNIX that perform filtering of data are also dealt with. This lecture also introduces an important concept of Unix called process. The commands that help in working with various processes are discussed.

4.2 I/O Redirection, Pipes and Filters

Standard Input

Whenever programs read or write files, they do not use filenames, but file descriptors which are integer values like 0, 1, 2, etc. The files are used to read in information (Standard Input), write output (Standard output) and write error messages that are pertinent to the utility (standard error). Their respective file descriptors are: 0, 1 and 2.

Normally when a command is run, the standard input is connected to the keyboard, standard output is connected to the display and standard error is also connected to the display screen. Thus, a command reads from the keyboard (standard input), and writes to the screen (standard output or error).

I/O Redirection

The shell allows the redirection of the standard input, output, and error of a command. Redirection of input/output is the capability to change the source of the input and the destination of the output. This is done by using greater than (>) and less than (<) signs.

Output Redirection

Under the UNIX system, the output from a command usually intended for standard output can be easily diverted to a file. This capability is known as output redirection.

Syntax

```
command > filename
```

The above format indicates that the command is being diverted to a file **filename** using a > sign.

Suppose the names of the logged_in_users have to be stored inside a file **names.**

```
# who > names
```

In the above example, the who command gets executed and instead of writing the output to the standard output (terminal), the output is being written to the file **names**.

If a command redirects its output to a file and the file already contains some data, then that data is lost and the new data is overwritten.

The existing contents can be retained as shown below:

```
# echo Hi, Good Morning > salute
# cat salute
Hi, Good Morning
# echo Hi, Good Evening >> salute
# cat salute
```

Hi, Good Morning

Hi, Good Evening

The second `echo` command uses a different type of redirection symbol >> (double greater than) sign. This sign causes the standard output from the command to be appended to the specified file. Therefore, the previous contents of the file are not lost and the new output is simply added onto the end of the first.

Note : In the output redirection, the stdout can be made explicit by preceding the `>` by the number 1, which is the file descriptor for stdout.(`# echo Hi, Good Morning 1> salute)`

Input Redirection

Just as the output of a command can be redirected to a file, so can the input of a command be redirected from a file i.e., the input for a command can be taken from a file instead of taking from the standard input (keyboard).

Syntax

```
command < filename
```

In this case, less than (<) sign is used to redirect the input of command from the filename.

Consider the following:

```
# sort < mydata
```

When the sort command opens the standard input and begins to read, it will read from the file mydata and not from the keyboard.

Error Redirection

As mentioned earlier, along with the standard output, the error messages, if any, generated by executing the command is also displayed on the screen. At times it may be difficult to catch the error message as it might get mixed up with the standard output. It would be better, if the error messages are made available separately. Just as the output is redirected to a file, the error can also be redirected. In the command line, by preceding the '>' output redirection symbol by a '2' the standard error message can be redirected.

Practice 4.1

The following example shows how to redirect the error message.

```
# ls *.txt 2> /tmp/err.out
# cat /tmp/err.out
ls: *.txt not found
```

In the above example the error message is redirected to the file /tmp

Input and Output Redirection

The input and the output of a command can be simultaneously redirected.

Practice 4.2

The following example illustrates the simultaneous redirection of both the input and the output.

```
# sort < names > sort_names
# cat sort_names
```

```
ajay       pts/3        May 17 11:27    (80.0.0.98)
oracle     pts/2        May 17 11:27    (80.0.0.98)
root       pts/0        May 17 11:27    (80.0.0.98)
sunil      pts/1        May 17 11:27    (80.0.0.98)
```

In this example, the sort command takes its input from the file names and its output is redirected to the file sort_names.

4.3 PIPES

The unix system enables the user to effectively connect two or more commands together. This connection is called a *pipe*. A pipe enables to take the output from one command and feed it directly into the input of the another command. A pipe is effected by the character | (called pipe), which is placed between two commands. All the commands in a pipeline are executed sequentially. Unix handles the flow of data from one command to the next, producing the effect as if one command is being executed.

Radiant™ RAY OF HOPE   Unix Administration

Pipes

⊥ Enables the user to effectively connect two or more commands together
⊥ Enables to take the output from one command and feed it directly into the input of the another command
⊥ Pipelines are executed sequentially
⊥ Handles the flow of data from one command to the next

As an example of a pipe, suppose you wanted to count the number of files in your directory. Now we know that ls command displays the names of all files, and the command wc –l is used to count the number of lines in a file. So both these commands can be effectively piped to get the desired result.

Consider the following

```
#ls | wc -l
     10
```

The output indicates that the directory contains 10 files. First, the ls command is executed to list the files, this output is then sent through the pipe as an input to the wc –l command, which is executed to give the number of files. The final output is the output of the last command.

4.4 Filters

Radiant™ RAY OF HOPE Unix Administration

Filters

- sort
- wc
- tr
- grep
- find
- cut
- head
- tail
- more
- tee

Filter is a UNIX utility that takes input data, processes it, and sends the result to the output. A filter selectively alters the data that passes through it. For example, the grep command (discussed below) filters out unwanted data and passes on the selected data to an output file.

sort

sort is a filter utility that is used to order the contents of the indicated file, alphabetically and display the result of the sort at the terminal. The original contents of the file remain unchanged.

Syntax

```
sort [options] filename
```

Options can be

-n	Sorts the input numerically
-r	Sorts the input in reverse order
-f	Ignores the significance of uppercase and lowercase letters
-o	Sends the output to a file rather than the standard output
-k	Sorts the records fieldwise
-t SEP	Uses SEParator instead of non- to whitespace transition

Practice 4.3

The following shows the usage of the sort command.

```
# cat students
Rama
Anju
Anita
Sanjana
Hema
Jeevan
Bobby
Neha
Priya
# sort students
```

```
Anita
Anju
Bobby
Hema
Jeevan
Neha
Priya
Rama
Sanjana
```

In the above example there is a file **students** that contains the names of the students of a particular batch. In order to display the names in an alphabetical order, the sort command is used. The sort command orders the names alphabetically.

Wc

Using the UNIX utility `wc`, the user can count all the lines, words and characters in a file.

Syntax

```
wc  [options] filename
```

The following options can be used with wc command.

-l	Indicates the number of lines
-w	Indicates the number of words
-c	Indicates the number of characters

Consider the following

```
 # wc num
   12       12       36 num
```

From the output it can be seen that there are four fields:

Number of lines	12
Number of words	12
Total number of characters	36
Name of the file	num

tr

`tr` command can be used to translate a set of characters to another. It reads from the standard input, searches for all the special characters, and translates each into another specified character and writes to the standard output. It cannot read/write from/to the files. Therefore, the redirection symbols or pipe must be used to input to the tr command.

Syntax

```
tr string1 string2
```

where **`string1, string2`** are the translation control strings. Each string represents a set of characters to be converted into an array of characters used for the translation. Consider the following

```
# tr “123” “ABC” <  mydata
```

As a result of executing `tr,` the following translation takes place in the file mydata which is output on the terminal.

1 -> A
2-> B
3 -> C

grep

The `grep` command searches files for a pattern and prints all lines that contain that pattern.

Syntax

```
grep [options] pattern  filename
```

If no files are specified, grep assumes standard input. Normally, each line found is copied to standard output. The file name is printed before each line found if there is more than one input file.

The file is searched line by line for the pattern. Every line that contains the pattern is displayed on the terminal. The pattern that is searched for in the file is called a regular expression.

Regular Expressions

Character	Description	Example
[*class*]	A character class. Matches any one character in the *class*.	“[xyz]” specifies the pattern either x or y or z.
[*c1-c2*]	Matches any one of the character specified in the range.	“[a-d]” specifies the pattern either a, b, c or d.
^	Pattern following it must occur at the beginning of each line.	“^my” specifies the pattern “my” should appear at the beginning of each line.
[^ *class*]	Does NOT match any of the characters specified.	“[^abc] specifies that the pattern should not contain a,b or c.
#	Matches the end of line.	“ball#” specifies the pattern “ball” at the end of each line.
\	Escapes the special meaning of the character.	“\#900” specifies the pattern “#900”, the sign # loses its special meaning.
. (*dot*)	Matches any single character.	“[abc].” Specifies the pattern a, b, or c followed by any one character.

The options can be :

-i	Ignore upper/lower case distinction during comparisons.
-n	Precede each line by its line number in the file (first line is 1).
-c	Print only a count of the lines that contain the pattern.
-l	Print only the names of files with matching lines, separated by NEWLINE characters. Do not repeat the names of files when the pattern is found more than once.

-v Print all lines except those that contain the pattern.

Consider the file **stud_rec** with the following data

```
# cat stud_rec
Anjana          23      UC++ Annanagar
Divya           19      UC++ Padi
Farida          26      DBA  T.Nagar
Prashant        22      UNIX Admin Kilpauk
Rahul           21      ORVB Shenoy Nagar
Fardeen         27      DBA Adayar
```

The students who are taking UCC++ course can be displayed using the grep as follows:

```
# grep UC++ stud_rec
Anjana          23      UC++ Annanagar
Divya           19      UC++ Padi
```

Using grep as a Filter

The `grep` utility can be used with other utilities as an information filter.

```
# ls | grep samples
```

The output of the `ls` command is directly given to the `grep` command as input. The pattern "samples" is searched in the list of filenames, with the `grep` command.

Note : One must be careful while using the characters #, *, [, ^, |, (,), and \ in the pattern_ list because they are also meaningful to the shell. It is safe to enclose the entire pattern_ list in single quotes '...'.

find

The find command locates the files that match specified expression. It starts searching for files in the given directory and continues its search through all subdirectories. It also provides a mechanism for performing actions on the files that meet the search criteria.

Syntax

```
find  path expression
```

where **path** is the list of directories to be searched. The names are separated by space or tab. To search the current directory, the dot (.) notation can be used.

Here **expression** can be :

-name <file>	Indicates the names of files to be found in the specified directory. For example: find /usr/ajay –name *.c will search all files with extension .c in the /usr/ajay directory.
-print	Causes the current path name to be printed.
-type c	Indicates the type of the file is c, where c is b, c, d, l, p, or f for block special file, character special file, directory, symbolic link, fifo (named pipe), or plain file, respectively.
-size n[c]	Finds the files containing n blocks. If n is followed by c, then n is counted in characters instead of blocks.
-exec cmd	Executes the command cmd. A pair of braces {} can be set to signify the presence of current pathname. The cmd must end with an escaped semicolon (\;). The expressions can also be used in combination.
-ok command	Like –exec, except that the generated command line is printed with a question mark first, and is executed only if the user responds by typing y.

The following example finds all the files under the current directory and prints them

```
# pwd
/usr/sunil
# find .
.
./salary
./file1
./file2
./file3
```

cut

cut is a filter utility that is used to cut out columns or fields from each line of a file. A field is normally separated by a tab character (default delimiter), but it can also be delimited by any other character.

Syntax

```
cut -c columnlist [ file ... ]
cut -f fieldlist [ -d delim ] [ -s ] [ file ... ]
```

where	**list** is a comma-separated or blank-character-separated list of integer field numbers (in increasing order), with optional - to indicate ranges (for instance, 1,4,7; 1-3,8; -5,10 (short for 1-5,10); or 3- (short for third through last field)).
-c list,	the list following -c specifies character positions (for instance, -c1-72 would pass the first 72 characters of each line).
-f list,	the list following -f is a list of fields assumed to be separated in the file by a delimiter character.

-d delimiter, the character following -d is the field delimiter (-f option only). Default is tab. Space or other characters with special meaning to the shell must be quoted. delimiter can be a multi-byte character.

Practice 4.4

Consider the file **names** containing the following information

```
Pat      Gaja       22       C++
Tedd     Boycott    34       ORVB
Anju     Ahuja      25       C++
Hema     Malini     27       DBA
Raju     Naidu      31       ORVB
Sonu     Misra      28       ORVB
```

The following example shows the usage of the cut command.

```
# cut –f 2,4 names
```

```
Gaja      C++
Boycott   ORVB
Ahuja     C++
Malini    DBA
Naidu     ORVB
Misra     ORVB
```

In the above example the second, fourth fields are cut from the file using the cut command.

paste

The `paste` command joins two files horizontally (column wise). For instance, if a file contains the list of students and the other file contains their ages, the two files can easily be joined together to get two columns: one containing names and the other containing ages.

Practice 4.5

Consider two files **students** and **stu_age** whose contents are as follows:

```
# cat students
Rama
Anju
Anita
Sanjana
Hema
Jeevan
Bobby
Neha
Priya
```

```
# cat stu_age
29
```

```
19
22
32
35
24
19
23
31
```

The following example shows the usage of the paste command

```
# paste students stu_age
```

```
Rama    29
Anju    19
Anita   22
Sanjana 32
Hema    35
Jeevan  24
Bobby   19
Neha    23
Priya   31
```

It can be seen that when the paste command is issued, the two files merge and the output appears in two columns.

head

The `head` utility copies the specified number of lines from the beginning of the file to the standard output. If no filename is given, head copies lines from the standard input. The default value is 10 lines.

Syntax

```
head [- number] file
```

Practice 4.6

Consider the file mybook that contains 16 lines. The following example shows the use of the head command.

```
# head mybook
```

```
This is my first book.
I am trying to a write.  Here, I test my skill in writing.
I leave 2 lines blank.

You may be wondering Why?
Because, I am a writer and I have to think.
I want to write but nothing seems to be coming out.
I have severe constipation of ideas.
```

Here since the numberof lines have not been specified, the first 10 lines (default value) of mybook are displayed.

tail

`tail` command is the reverse of the head command. It is used to display the last few lines of a file. If no file is named, the standard input is used. By default, it displays the last 10 lines.

Syntax

```
tail [ -/+ number ] file
```

Practice 4.7

Consider the file **mybook** discussed in the previous example. The following example shows how to display the lines from the end of the file **mybook.**

```
# tail -5 mybook
```

```
Anyway, I am not going to give up.
Definitely, one day I am going to be a greater writer.
But, today I think, I need a break.
Good Bye.
Good Bye.
```

In this example the last 5 lines of **mybook** are displayed.

more

more is a filter that displays the contents of a text file on the terminal, one screen at a time. It normally pauses after each screen.

Syntax

```
more filename
```

more scrolls up to display one more line in response to a RETURN character; it displays another screen in response to a SPACE character.

tee

Any output from a command that gets piped into another command is not seen at the terminal. Sometimes, the user might want to save the output that is produced in the middle of a pipe. The tee command enables to do this easily.

Syntax

```
tee file
```

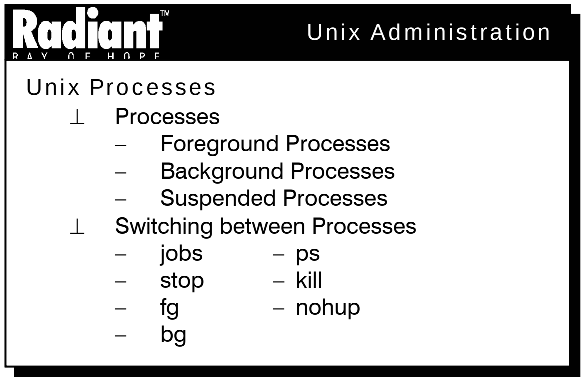
The **tee** command simply copies the data coming in to the standard output, while saving a copy in the specified file.

Consider the following example

```
# sort students | tee stud | more
Anita
Anju
Bobby
Hema
Jeevan
Neha
Priya
Rama
Sanjana
(EOF):
```

This will sort the student file, display it page wise and save it in file stud. Now the sorted file stud can be displayed on the terminal using cat command.

4. 5 Unix Processes


Radiant™ RAY OF HOPE — Unix Administration

Unix Processes

- ⊥ Processes
  - – Foreground Processes
  - – Background Processes
  - – Suspended Processes
- ⊥ Switching between Processes
  - – jobs – ps
  - – stop – kill
  - – fg – nohup
  - – bg

Processes

Unix is a multi-user as well as multitasking system. This means that more than one user can use the system and each user can perform a number of tasks at the same time. In UNIX system, a task is referred to as a process. A process is the existence of an executing program in the computer.

Since the CPU can carry out one task at a time, multitasking is accomplished using the concept of time-sharing. The UNIX Kernel maintains a list of processes started by a user and each process is allocated a small time quantum, called time slice, during which it can carry out its execution. When its time elapses, the first process is suspended and other is given a chance to run. When all processes in the list have had their chance to run, the kernel switches back to the first process, starting the execution from where it was suspended. In this way, every process is allowed to work its way to completion, a little bit at a time.

The time slice for each process is so small (a few hundredths of a second), it gives the impression that each user is being served simultaneously, even though, they are being served one at a time.

There are various types of processes that can be started simultaneously by a user. These processes fall into the following categories:

Foreground processes- processes with which the user directly interacts.

Background processes- processes that are dissociated from the terminal.

Suspended processes- processes whose execution has been suspended for a while.

Foreground Processes

A foreground process is the one in which the user interfaces with, from the keyboard. The user writes a command at the # prompt and waits till the execution is over.

When the command is issued, it gets executed and the result is echoed to the screen.

Background Processes

A background process can be started along with a foreground process. When a command is appended with an ampersand (&) sign, it gets dissociated from the terminal and carries its execution in the background.

Consider the following example

```
# sort data > /tmp/out &
[1] 1074
#
[1] + Done(0)                    cat data 1>/tmp/out
#
```

The above command sorts the file data and redirects the output to another file out. This whole process is appended by an &, implying that it has to be carried out in the background. When a process is sent to the background, UNIX automatically displays a unique number identifying that process, called process id (PID). In the above example, it is 1074. After displaying the PID number, the terminal is immediately available to the user (indicated by the # sign).

It is useful to carry out those processes in the background which do not require user input so that, in the meantime, another process can be executed in the foreground. For example, printing a large document in the background while editing another document in the foreground.

Suspended Processes

A user can suspend a job running in the background or one running in the foreground. This is useful when the user is running a command and some other function needs to be performed. A running job can be suspended by pressing Ctrl-Z.

Switching between Processes

Sometimes a process may have to be brought from its suspended state to its running state, either in the background or foreground. This can be accomplished by using `fg` and `bg` commands. To know which processes are in background or in a suspended state, the jobs command can be used.

jobs

The jobs command enables to know about the processes that are in background or in the suspended state.

Syntax

```
jobs [+/-]
```

+ Indicates the current background process

Indicates the previous process

Note : Using the option –l, the process IDs of the processes can also be displayed.

stop

This command is used to stop the background process.

Syntax

```
# stop [% job id]
```

The following example shows how to stop the sixth process.

```
# stop %6

#jobs

[6] + Stopped(signal)  find / -name sample&
[5] +  Running    find / -name sample&
[4] +  Running    find / -name sample&
[3] +  Running    find / -name sample&
[2] +  Running    find / -name sample&
[1] +  Running    find / -name sample&
```

fg

The `fg` command allows the suspended jobs to be resumed by placing them in the foreground. The suspended jobs, when brought back to the foreground, become interactive again.

Syntax

```
fg [ % job]
```

where % **job** can be :

% num	Indicates a job number for an associated process
% +	Indicates the current job; the last job suspended
% -	Indicates the previous job; process before the current job

The following example shows how to bring the stopped job 6 to foreground.

```
# fg %6
```

bg

The `bg` command allows the suspended jobs to be resumed by placing them in background.

Syntax

```
bg [ % job]
```

where % **job** can be

% **num**	Indicates a job number for an associated process
% **+**	Indicates the currernt job; the last job suspended
% **-**	Indicates the previous job; process before the current job

The following example shows how to restart the suspended job 6.

```
# bg %6
[1] ls -Rl / 1>/tmp/out1 &

# jobs
[6] + Running                     ls -Rl / 1>/tmp/out1
```

ps

The `ps` command is used to display the status of the running processes. The ps command reports the process ID, the associated terminal type, the amount of time a process has used and the command being executed.

Syntax

```
ps [options]
```

Options:

-u <username>	Displays the process status for a particular user
-e	Displays the information about every process
-f	Displays the full listing of header information
-t <terminal>	Displays the process status for a particular terminal

The column headers are:

UID	user's ID no. of the process's owner
PID	process ID of the process
PPID	process ID of the parent process
C	the processor utilization used for scheduling purposes
STIME	time the command started
TTY	terminal from which the process was started
TIME	cumulative execution time for the process
COMMAND	name of the process started

The following example shows how to get the process status of the processes executed from the current terminal

```
# ps
   PID TTY      TIME CMD
  1077 pts/2    0:11 ls
  1070 pts/2    0:00 sh
  1072 pts/2    0:00 jsh
```

Note : A ? mark in the TTY column represents that the process is started automatically and there is no controlling terminal.

kill

If a process is running in background and for some reasons that process has to be terminated, then `kill` command can be used to accomplish the task.

Syntax

```
kill  pid
```

where **pid** is the procees id number

The following example shows how to kill a process.

```
# ls -R / > /tmp/out &
[1] 1098
#
#
# ps
   PID TTY      TIME CMD
  1098 pts/2    0:00 ls
  1070 pts/2    0:00 sh
  1072 pts/2    0:00 jsh
# kill 1098
# ps
   PID TTY      TIME CMD
  1070 pts/2    0:00 sh
  1072 pts/2    0:00 jsh
[1] + Terminated                  ls -R / 1>/tmp/out
```

When a process is placed in background, its PID number is immediately displayed. To know the PID number of the process to be terminated, the ps or jobs –l command can be used.

Note : Only the owner of the process or the superuser can kill a process. Killing the shell will logout the user from the system.

nohup

A background process can continue its execution even after the user has logged out. Normally, all processes terminate at the time of logout. But if the background process is started with a nohup command, the background process can still be continued.

The nohup utility invokes the named command with the arguments supplied. When the command is invoked, nohup arranges for the Hangup signal which intimates the process to hangup, to be ignored by the process. nohup can be used when it is known that command will take a long time to run and the user wants to logout of the terminal; when a shell exits, the system sends its children hangup signals, which by default cause them to be killed.

Syntax

```
nohup command [arguments]
```

All stopped, running, and background jobs will ignore hangup signal and continue running, if their invocation is preceded by the nohup command.

The output of the nohup command is captured in a file called nohup.out if standard output is a terminal and if the current directory is writable.

The following example shows the usage of the nohup command.

```
# nohup ls -Rl / > /tmp/out &
[1] 1101
# exit
```

On logging in again and it can be seen that the command that has been issued is still running.

4.6 Short Summary

- Files are used to read in information (Standard Input), write output (Standard output) and write messages that are pertinent to the utility.
- Redirection of the input/output is the capability to change the source of the input and the destination of the output.
- Pipes enables the user to effectively connect two or more commands together
- Filter is the UNIX utility that takes input data, processes it and sends the result to the output.
- A process is the existence of an executing program in the computer

4.7 Brain Storm

1. Explain the difference between multi-user and multi-tasking environment?
2. What is the background process? How is it suspended?
3. How to switch between processes?
4. What is the use of the kill command?
5. Explain the differences between suspend and kill.

Lecture 5

Introduction to Shell Programming

Objectives

In this lecture you will learn the following

- Knowing about Shell and its types
- Processing commands by shells
- Understand about the variable and their types

Coverage Plan

Lecture 5

5.1 Snap Shot

The shell is a program that interacts with the UNIX Operating System and the User. It provides a user interface to the Operating System that listens for user commands, does the necessary translations and tells the system to execute them. The Bourne shell is the orginal UNIX Command interpreter. Over the years, new interpreters based on the Bourne Shell have been developed. The most popular ones are the Korne shell(ksh) and bach(Bourne Again Shell). The Bourne shell is also the preferred choice to develop shell scripts.

5.2 What is a Shell?

Radiant™ RAY OF HOPE — Unix Administration

Filters

- ⊥ Types of Shells
- ⊥ Processing Commands by shells
- ⊥ Variables
- ⊥ Types of Variables
- ⊥ Command Substitution
- ⊥ Positional Parameters
- ⊥ Export Command

The Shell is both a command-line interpreter and a high-level programming language. When it is acting as a command-line interpreter, it processes commands as entered at the command prompt. When it is used as a programming language, it processes commands that are stored in files known as Shell scripts.

Shell scripts allow to group command lines together and execute them by entering a single command at the command line. This allows complex functions to be completed by any user, and repetitive functions can be completed easily. Input and output can also be redirected from a Shell script.

Types of Shell

There are three types of shells available in most Unix Systems.They are as follows

Shell Name	Prompt	Developed By	Description
Bourne(sh)	$	Steven Bourne at AT&T Bell Labs	The default shell for all Unix operating system.
C(csh)	%	Bill Joy at	The C shell was separately
		University of	developed and it is similar
	Calfornia	to C programming language.	
Korne(ksh)	$	David G.Korn at AT & T Bell labs features of C shell.	This is derived from bourne shell and with some enhanced

Table 5.1

Bourne shell is the accepted standard for SystemV UNIX.

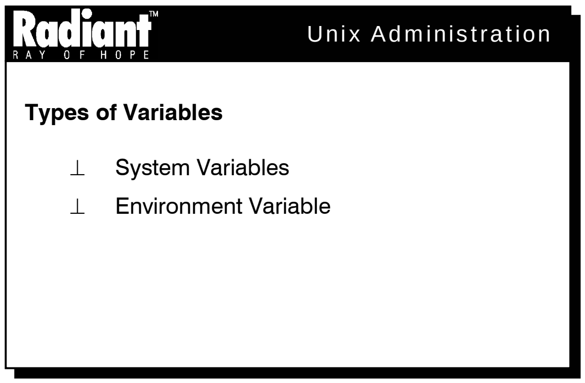
Processing Commands by shells

Each shell creates *subshells* and *child processes*—subordinate shells and processes that are executed within the originating, or *parent,* shell—to interpret and execute commands. For example, the following list shows a simplified version of the order in which the Korne shell processes commands:

1. Parses (divides up) the command into units separated by the fixed set of meta characters: Space Tab Newline ;( ) < > | &. Types of units include words, keywords, I/O redirectors, semicolons, and others.
2. Checks the first part of each unit for shell keywords, such as function or if statements, with no quotes or backslashes. When it finds a keyword, the shell processes the compound command.
3. Searches the list of aliases.
4. Expands any tilde (~) expressions.
5. Substitutes variables.
6. Substitutes commands.
7. Substitutes arithmetic expressions.
8. Splits the items that result from parameter, command, and arithmetic substitution and again splits them into words
9. Expands wildcards.
10. Looks up built-in commands, functions, and executable files.
11. Sets up I/O redirection.
12. Runs the command.

Note : The Bourne shell interprets commands similarly, but does not check for aliases, tildes, or arithmetic. The C shell interprets commands in a different order.

5.3 What is a Variable?



Unix Administration

Types of Variables

⊥ System Variables

⊥ Environment Variable

Variable is nothing but a memory storage, which is used to store the value. It helps to refer the value during processing.

Syntax

```
variable-name=value
```

Only letters, digits, underscore and $ should be used in variable names.

Note: No blanks are allowed before and after the = sign

Shell variables are string variables

```
# name="radiant"
# a=10
```

Here "radiant" and the value 10 will be treated as strings. It is not possible to perform arithmetic operations on the variable a.

The value stored in the variable *name* can be viewed given by using echo $variablename. The echo command is used to print the value on the screen.

Example

```
# name=radiant
# a=10
# echo $name
radiant
# echo $a
10
```

To perform the arithmetic operations, the expr command has to be used as shown below

```
# echo $a + 10
which  will produce the output as 10 + 10
```

The user can store a null string, which consists of no characters in a variable in the following formats

```
#  name=''
#  name=
#  name=""
```

The formats equate the variable *name* to the null string.

Types of variables

System Variables

It is a predefined variable developed when the UNIX Operating System was developed. It is used to store the system status output which is used for feature or current reference purpose.

```
$?
```

Returns the exit status (return code) of the last command executed as a decimal string. Most commands return a zero exit status if they complete successfully, otherwise a non-zero exit status is returned. Testing the value of return codes is dealt with later under **if** and **while** commands.

```
# pwd
/usr/nellaiprakash
# echo $?
0
```

```
$#
```

Returns the number of positional parameters (in decimal). Positional parameters are elaborately discussed later in this session.

```
$$
```

Returns the process number of the shell (in decimal). Since process numbers are unique among all existing processes, this string is frequently used to generate unique temporary file names.

```
  # echo $$
  786
$!
```

Returns the process number of the last process that runs in the background (in decimal).

```
# echo $!
790
```

```
$-
```

Returns the current shell flags, such as **-x** and **-v.**

Some variables have a special meaning to the shell and should be avoided for general use.

Environment Variables

It is a predefined variable where the user can store values and make it as the current environment settings.

```
$MAIL
```

When used interactively the shell looks at the file specified by this variable before it issues a prompt. If the specified file has been modified since it was last opened, the shell prints the message *you have mail* before prompting for the next command. This variable is typically set in the file **.profile,** in the user's login directory.

```
  MAIL=/usr/mail/fred
$HOME
```

This is the default argument for the *cd* command. The current directory is used to resolve file name references that do not begin with a **/,** and is changed using the *cd* command.

The following command

```
#cd /usr/fred/bin changes the  current directory to /usr/fred/bin.
```

The command *cd* with no argument is equivalent to

```
#cd $HOME
```

This variable is also set in the user's login profile.

```
$PATH
```

Stores the list of directories that contain commands (the *search path*). Each time a command is executed by the shell, a list of directories is searched for an executable file. If **$PATH** is not set then the current directory, **/bin**, and **/usr/bin** are searched by default. Otherwise **$PATH** consists of the directory names separated by **:**(semicolon).

The following command

```
PATH=:/usr/fred/bin:/bin:/usr/bin
```

specifies that the directories, the current directory (the null string before the first **:**), **/usr/fred/bin, /bin** and **/usr/bin** are to be searched in that order. This enables individual users to have their own 'private' commands that are accessible independent of the current directory. If the command name contains a **/** then this directory search is not used; a single attempt is made to execute the command.

```
$PS1
```

Stores the primary shell prompt string, by default, '**$** '. The user can change his prompt.

For example, consider the following

```
# PS1=radiant
Now the prompt will be
radiant
```

```
$PS2
```

Stores the shell prompt string, by default, '> '

Example

```
#PS2=radiant
now the prompt will be
radiant>
```

```
$TERM
```

Stores the terminal specification. A Unix system can have different types of terminals like vt100,vt220 and so on

```
$LOGNAME
```

This variable contains the user's login name.

```
# echo $LOGNAME
root
```

```
$SHELL
```

Stores the name of the shell.

```
# echo $SHELL
/bin/sh
```

Note : Using env command, all the environment variables can be viewed.

5.4 Command Substitution

Command substitution allows to capture the output of any command as an argument for another command. When a command line is placed within " (backquotes or tilde characters), the shell first runs the command or commands, and then replaces the entire expression, including the backquotes, with the output. This feature is often used to give values to shell variables.

Consider the following

```
# today='date'
```

Now this assigns the string representing the current date to the variable today.

```
# echo $today
```

will display the output of date command.

The following assignment saves, in the files variable, the number of files in the current directory:

```
# files=‘ls | wc -l‘
```

Now the following command

```
# echo $files
```

will display the total number of files in the current directory.

Command substitution can be performed on any command that writes to the standard output. To nest command substitutions, each of the nested backquotes have to be preceded by a \ (backslash), as in the following

```
# logmsg=‘echo Your login directory is \‘pwd\“
# echo $logmsg
Your login directory is /home/manu
```

Positional Parameters

Positional parameters are defined as numbered variables representing the position of the arguments passed to the shell script. The name of the shell script is passed as the positional parameter $0, the first argument is passed as positional parameter $1 and so on. Unix shell creates a number of arguments that are specified in the command and the shell assigns values to these positional parameters.

```
$0 – Command or Source file name
$1-$9– Passed as an argument using the user defined file or commands
$# - list the total number of parameter
$* - list the individual details about each and every arguments.
```

Practice 5.1

The following shell program shows how to implement the positional parameters.

```
# cat > pscript

echo “The command is $0”
echo “The First argument is $1”
echo “The Second argument is $1”
echo “The Third argument is $1”
echo “The Fourth argument is $1”
```

```
# chmod u + x pscript
#./ pscript a b c d

The command is pscript
The First argument is a
The Second argument is b
The Third argument is c
The Fourth argument is d
```

In the above program, the pscript (filename) is $0 and the parameters are a, b, c and d are represented as $0,$1,$2 and $3 respectively.

The export command

When a variable is created, it is by definition a shell or local variable, which means that it is available for use by the shell and not for any UNIX applications. If these variables are used inside UNIX applications, they should be made global. This is done by explicitly telling the shell to export the variables to all the programs. The variables are exported using the export command.

Syntax

```
export  variablename
```

where variablename is the name of the variable to be exported.

Consider the following

Here a local variable Z is assigned a value of 100. Thereafter,another process starts. The new process does not know about the variable Z, which is assigned in its parent shell.

```
# Z=100
# echo $Z
100
# sh
# echo $Z
# exit
```

The variable Z can be made global so that it is available to all the processes as shown in the following example

```
# export  Z

# sh
# echo $Z
100
# Z=200
# echo $Z
200
# exit
# echo $Z
100
```

Here the value of Z is available to all the subprocesses.If another value is assigned to Z within the subprocess,the local variable Z precedes the global variable Z.

Note : Local variables are available only during the lifetime of the process. They get destroyed with the process. Global variables are available throughout the shell session. When the local and the global variable have the same name, the local variable gets the precedence during execution. If an export command is used without any argument, it lists out all the exported variables, if you are in a sub shell, only the variables that have been exported.

5.5 Short Summary

- Shell is command line interpreter and high level programming language Bourne, Korne and C are three types of shell.
- Variable is nothing but a memory storage, which is used to store the value, and to refer the value for processing.
- The two types of variables that are used are: System Variables and Environment Variables are the two types of variables.
- System variable is a predefined developed when the UNIX Operating System was developed. It is used to store the system status output which is used for feature or current reference purpose.
- Environment variable is a predefined variable where the user can store values and make it as the current environment settings.
- Command substitution allows you to capture the output of any command as an argument to another command.
- Positional parameters are numbered variables that represent the position of the arguments that are passed to the shell script.
- The export command makes the variable available for global usage

5. 6 Brain Storm

1. What is shell? Mention the types of shell?
2. What is the difference between system variables and environment variables?
3. What is the purpose of command substitution?
4. What is the use of the export command?
5. What are the different positional parameters used in shell scripts?
6. What is the use of the shift operator?
7. What is the use of the trap command?

Lecture 6

Advanced Shell Scripts

Objectives

In this lecture you will learn the following

- About Shell Scripts
- Knowing the command constructs in the Shell programming

Coverage Plan

Lecture 6

6.1 Snap Shot

Shell Commands can be placed within a file and then the shell can be made to read and execute the commands in the file. In this sense, the file functions as a shell program executing shell commands as if they were statements in a program. A file that contains shell commands is called a shellscript. Advanced shell scripting enables the administrator to design and control certain system events and procedures.

6.2 Advanced Shell Scripts

Radiant™ RAY OF HOPE — Unix Administration

Advanced Shell Scripts

- The echo command
- The expr command
- The if statement
- The while statement
- The case statement
- The continue statement
- Read Command
- The test command
- The for statement
- The Until Statement
- The break command
- The trap command

This command writes each given STRING to the standard output, with a space between each STRING and a new line after the last one.

Syntax

```
echo [OPTION]... [STRING]...
```

This command accepts the following options. This command is exclusivly used in shell scripts.

```
-n  Disables output of  trailing new line.
-e  Enable interpretation of the following backslash-escaped characters in
    each STRING:
    only in the case of LINUX
\a  alert (bell)
\b  backspace
\c  suppress trailing newline
\f  form feed
\n  new line
\r  carriage return
\t  horizontal tab
\v  vertical tab
\\  back slash
```

Consider the following example

# echo hello welcome to shell programming

hello welcome to shell programming

```
# a=10
# name=lambent
```

```
#echo the value of a is $a
the value of a is 10
```

```
# echo The Name is $name
 The Name is lambent
```

```
# echo -e “hello \a”
```

```
The output will be hello with an alarm sound.
```

The read Command

When writing shell programs, the user might need to enter some input/data from the keyboard and assign it to the shell variables. The read command enables this

Syntax

```
read variablename
```

The read command waits for the user to enter a value. The shell assigns that value to the shell variable, which is specified by variable name.

Practice 6.1

The following program illustrates the usage of the read command.

```
# cat > read
```

```
echo “enter the name:”
read name
echo “$name Welcome to Unix environment”
```

```
# sh read
enter the name
raj
raj welcome to Unix environment
```

The above program will interact the user and get the name **raj** from the user through the keyboard and substitute the name and display the message **raj Welcome to Unix Environment.**

The expr command

expr - Evaluate expressions

This command evaluates an expression and writes the result on the standard output. Each token of the expression must be a separate argument.

Numeric expressions

The ‘expr’ command supports the numeric operators.The numerical operators that are supported in the order increasing precedence are +,-,*,/,%.The arguments are coerced to number (an error can occur if this cannot be done).

Consider the following example

```
# a=10
# b=20
# echo “Addition `expr $a + $b`
Addition 30
```

```
# echo “Subtraction `expr $b - $a`
Subtraction 10

# a=10
# b=20

# echo “Multiplication `expr $a \* $b`
Multiplication 200

# echo “Division `expr $b / $a `
Division 2
which gives quotient part

# echo “Modulus `expr $b / $a`
Modulus 0
which gives remainder part
```

The test command

The *test* command is used to evaluate conditional expressions.

Syntax

```
test expression
```

or

```
[ expression ]
```

Applying Integer Operator using test command

Operator	**Description**
int1 -eq int2	Returns true if int1 is equal to int2
int1 -ge int2	Returns true if int1 is greater than or equal to int2
int1 -gt int2	Returns true if int1 is greater than int2
int1 -le int2	Returns true if int1 is less than or equal to int2
int1 -lt int2	Returns true if int1 is less than int2
int1 -ne int2	Returns true if int1 is not equal to int2

Table 6.1

Applying String Operator using test command

Operator	**Description**
str1 = str2	Returns true if str1 is identical to str2
str1 != str2	Returns true if str1 is not identical to str2
str	Return true if str is not null
-n str	Return true if the length of str is greater than zero
-z str	Return true if the length of str is equal to zero

Table 6.2

Applying File Operator using test command

Operator	**Description**
-d filename	Returns true if file,filename is directory
-f filename	Returns true if file,filename is an ordinary file
-r filename	Returns true if file,filename can be read by the process
-s filename	Returns true if file,filename has a nonzero length

-w filename	Returns true if file,filename canbe written by the process
-x filename	Returns true if file,filename is executable

Table 6.3

Applying Logical Operator using test command

Operator	Description
! expr	Returns true if expr is not true.
expr1 -a expr2	Returns true if expr1 and expr2 are true.
expr1 -o expr2	Returns true if expr1 or expr2 is true.

Table 6.4

The if statement

All the three shells support nested if..then..else statements.These statements provide a way of performing complicated conditional tests in shell programs.

Syntax

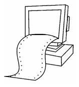
```
if [ expression ]
then
  commands
elif [ expression ]
then
  commands
else
  commands
fi
```

Practice 6.2

The following shell program will find if the '.profile' file is present in the current directory

```
# vi checkprofile
```

```
echo -e "Enter the filename to search :\\c"
read fname
if [ -f $fname ]
then
  echo "There is $fname file in the current directory"
else
  echo "Could not find the .profile file"
fi
```



```
# sh checkprofile
```

The above program will check for the file. Profile in the current directory using the test option –f. If the file is found, then it will display the message there is. Profile file in the current directory else it will display the message Could not find the .profile file.

Practice 6.3

The following shell program will find the of 3 numbers.

```
# vi greatnum
echo -e “Enter the value for a, b and c:”
read a
read b
read c
if [ $a -gt $b -a $a -gt $c ]
then
  echo $a is greater than $b and $c
elif [ $b -gt $c ]
then
  echo $b is greater than $a and $c
else
  echo $c is greater than $a and $b
fi
```

```
# sh greatnum
Enter the value for a, b and c:
12
34
56
56 is greater than 12 and 34
```

The above program accepts three numbers for a, b and c respectively. Using the if statement and logical AND operator (-a) checking will be performed .It will check the value of a with b and c and confirm whether a is greater than b and c else it will check b value with c and confirm whether b is greater than a and c else c is greater than a and b.

The for statement

The for statement executes the commands that are contained within do and done, a specified number of times.

Syntax

```
for variable in list-of-commands
do
  commands
done
```

Practice 6.4

The following shell program will convert all the file content to uppercase started with a as first name and by creating with same file name and extension as caps.

```
# vi upperconv
for fname in ls a*
do
  tr “[a-z]” “[A-Z]” < $fname > $fname.caps
done
```

```
# sh upperconv
```

The above program will find the filename which starts with a as the first letter and open the file ID will then convert the entire file content into uppercase and store it in a file with the same filename but with extension in caps.

The while statement

The while statement is another iteration statement that is offered by the shell programming language. This statement causes a block of code to be executed while a provided conditional expression is true.

Syntax

```
while [ command-list1 ]
  do
      command-list2
done
```

The value tested by the **while** command is the exit status of the last simple command following **while.** After each iteration c*ommand-list1* is executed; if a zero exit status is returned then *command-list2* is executed; otherwise, the loop terminates. For example,

```
while test $1
do ...
   shift
done
is equivalent to
for i
do ...
done
```

shift is a shell command that renames the positional parameters **$2, $3, ...** as **$1, $2, ...** and loses the parameter **$1.** Here the shift command moves the command-line parameters to the left by one parameter.

Practice 6.5

ing shell program lists the parameters that are passed to the program , along with the number of the parameter.

```
# vi whileparam

count=1
while [ -n “$*” ]
do
  echo “This is parameter number $count $1”
  shift
  count=‘expr $count + 1‘
done
```

```
# sh while 1 2 3 4 5 6

This is parameter number 1 1
This is parameter number 2 2
This is parameter number 3 3
This is parameter number 4 4
This is parameter number 5 5
This is parameter number 6 6
```

The above program will accept the arguments and display the respective position of each argument or parameter. Here the shift command has been used to shift the parameter to next position.

Practice 6.6

The following shell program will generate the fibonacci series 0,1,1,2,3,5,8,13,......

```
# vi whilefib

n1=0
n2=1
echo $n1
while [ $n2 -lt 200 ]
do
  echo $n2
  n2=‘expr $n2  + $n2 ‘
  n1=‘expr $n2 - $n1‘
done
```

```
# sh whilefib
0
1
1
2
3
5
8
13
21
```

The above program uses the while statement to generate the fibonacci series.

The Until Statement

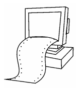
The until statement is very similar in syntax and function to the while statement. The difference between the two statements is that, the until statement executes its code block while its conditional expression is false, and the while statement executes its code block while its conditional expression is true.

```
until [ condition ]
do
  commands
done
```

Practice 6.7

The following shell program lists the parameters that were passed to the program , along with parameter number

```
count=1
until [ -n “$*” ]
do
 echo “This is parameter number $count  $1”
 shift
 count=‘expr $count + 1‘
done
```



```
# sh while 1 2 3 4 5 6

This is parameter number 1 1
This is parameter number 2 2
This is parameter number 3 3
This is parameter number 4 4
This is parameter number 5 5
This is parameter number 6 6
```

The above program will accept the arguments and display the respective position of each argument or parameter. Here the shift command has been used to shift the parameter to the next position.

The case statement

The case statement enables to compare a pattern with several other patterns and execute a block of code if a match is found. The shell case statement is relatively powerful than the case statement in Pascal or the switch statement in C. The reason for such a condition is that in the shell case statement it is possible to compare strings using wildcard characters.

Syntax

```
case string1 in
str1)
 commands;;
str2)
 commands;;
*)
 commands;;
esac
```

Practice 6.8

The following shell program will create,display and delete a file the using case statement

```
# vi casefile

echo -e “Enter the filename :\\c”
read fname

echo -e “Enter the choice (1-create;2-display;3-delete):\\c”
read choice

case $choice in
```

```
1) cat  >  $fname;;
2) cat $fname;;
3) rm $fname;;
*) echo “please choose option between [1-3] “
   break;
esac
```

```
# sh casefile
Enter the filename: sivam
Enter the choice (1-create;2-display;3-delete):1
This is a demo to create a file using case statement
To display the file
^d
# sh casefile
Enter the filename: sivam
Enter the choice (1-create;2-display;3-delete):2
This is a demo to create a file using case statement
To display the file
# sh casefile
Enter the filename: sivam
Enter the choice (1-create;2-display;3-delete):4
please choose option between [1-3]
```

The above program gives 3 choices from 1 to 3. Depending on the value that is chosen the corresponding command is executed. If any other value is chosen, it will display “ please choose option between [1-3]”.

The break command

Break statement can be used to come out of a loop. The further execution of the loop stops and the next statement after the done statement is executed. This may be useful when the user does not want the while loop to execute indefinite number of times. If the mechanism is not provided the execution will remain in the loop forever and probably, the system may hang. Break command helps to overcome such situations.

Practice 6.9

The following shell program will find whether the file is available in the directory or not.

```
# cat > break
while true
do
 echo “Enter the file name:”
 read fname
 if [ -f $fname ]
 then
    echo “File not found”
    break
 else
    echo “File found”
 fi
 echo “Bye”
done
```

```
# sh break
Enter the filename:
casefile
file found
bye
Enter the filename
Case
File not found
#
```

The above program will accept the filename and find whether file exists in the directory or not. If it is found, it prompts "file found and get another filename to search. If the file is not found, it will automatically break from the while loop.

The Continue Statement

The continue command is like the break command. It causes the execution to come out of the loop. But unlike break which causes the execution to continue after the loop, the continue statement causes the loop to execute from the next iteration.

Practice 6.10

The following shell program if a given number is greater than 1000 or not.

```
$ cat > until

until false
 do
     echo "enter a number less than 1000:"
     read nm
     if [ $nm –ge 1000 ]
     then
        continue
     else
        echo "The number is $num"
     fi
done
```

```
Explanation

Here if a number less than 1000 is entered that numbere is displayed and
program exits. But if the number is greater than 1000 then the program
encounters the continue statement and the program repeats prompting the user
to enter a number less than 1000. This process continues till the users enters
a number less than 1000.
```

The trap command

During execution there are certain things that are not under the control of the program.

The user of the program may press the interrupt key or send a kill command to the process or the controlling terminal may become disconnected from the system. In UNIX, any of these events can cause a signal to be sent to the process. The default action, when a process receives a signal is to terminate.

Sometimes, the user might want to take some special action when a signal is received. If a program is creating temporary data files, and it is terminated by a signal, the temporary data files remain. The user can change the default action of the program by using the trap command.

Syntax

trap command-string signals

where signals are one or more signals which are to be trapped

command-string is one or more commands separated by semicolon which are to be executed when the signals are received.

Example

```
$ trap “rm /tmp/*$$ ; exit “ 1 2 15
```

The command informs the shell to remove the files that end with PID at the temporary directory and exit the program when signals 1,2 and 15 are received.

Signal	Description
1.	Hangup
2.	Operator Input
15	Software Termination (kill signal

6.3 Short Summary

- Shell is a command-line interpreter and high-level programming language Bourne, Korne and C are three types of shells.
- Variable is nothing but a memory storage, which is used to store the value, and to refer the value for processing
- The two types of variables that are used are: System Variables and Environment Variables
- System variables is a predefined variable developed when the UNIX Operating System was developed.It is used to store the system status output which is used for feature or current reference purpose
- Environment variable is predefined variable where the user can store values and make it as the current environment settings
- Command substitution allows to capture the output of any command as an argument to another command
- Positional Parameters are numbered variables that represent the position of the arguments that are passed to the shell script.
- The export command makes the variable available for global usage.

6.4 Brain Storm

1. What is a shell? Mention the types of shell.
2. What is the difference between system variables and environment variables?
3. What is the purpose of command substitution?
4. What is the use of the export command?
5. What are the different positional parameters used in shell scripts?
6. What is the use of the shift operator?
7. What is the use of trap the command?

Lecture 7

Booting Shutting

Objectives

In this lecture you will learn the following

- About Booting and shutting
- Knowing the types of Booting
- About Boot Process
- Knowing about Run levels
- Overview of different run levels

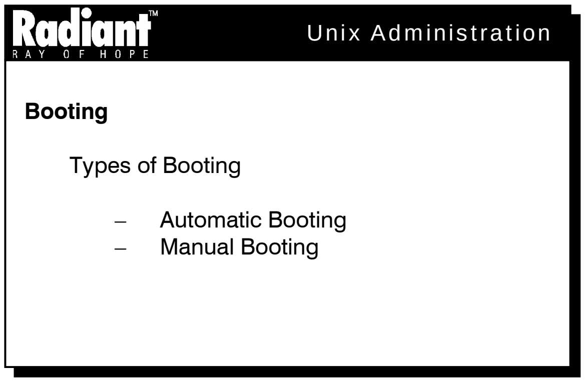
Coverage Plan

Lecture 7

7.1 Snap Shot

The Operating system can be accessed by performing the system startup process which involves a series of events collectively called Boot Process. Similarly, once the system is not in use it has to be shutdown and this also involves a series of steps starting with closing down in order of priority all the software applications followed by the hardware.

7.2 Booting



Radiant™ RAY OF HOPE — Unix Administration

Booting

Types of Booting

- Automatic Booting
- Manual Booting

Loading the OS into memory and getting it running is called "bootstrapping". In simpler terms, it is referred to as "booting". The boot process can be complex and understanding it requires an in-depth knowledge of the working of Unix. Understanding the boot process will help in understanding Unix in general. In the boot process the philosophy of Unix having many small programs can be seen. These programs perform a specific jobs, rather than having one large program which attempts to do everything.

Types of Booting

There are two types of booting which are as follows:

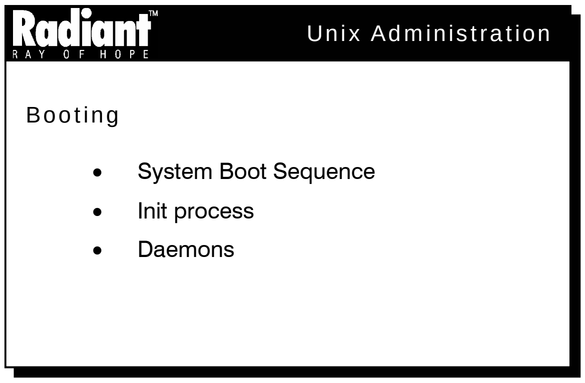
Automatic Booting

Here the system bootstraps itself, and comes up running normally (often called "multi-user mode"). This is the default.

Manual Booting

The user intervenes, and tells the system to load a different kernel, or come up in "single-user mode," which may be necessary to fix certain problems.

7.3 Boot Process


Radiant™ RAY OF HOPE — Unix Administration

Booting

- System Boot Sequence
- Init process
- Daemons

All computer systems start the boot process by executing the code in ROM (specifically, the BIOS) to load the sector from sector 0 of the boot drive. The boot drive is usually the first hard disk. The BIOS then tries to execute this sector. On all hard disks, sector 0, contains the start of an operating system kernel, such as Unix.

Once the Kernel is completely loaded, it goes through some basic device initializations. The kernel has to look for the root file system, if it does not find a loadable image there, it halts.

At this point the system finds the init program on the root file system (in /bin or /sbin) and executes it. The init program reads its configuration file /etc/inittab, looks for a line-designated sysinit, and executes the named script. The sysinit script is usually something like /etc/rc.d/rc.sysinit. This script is a set of shell commands that set up the basic system services, such as

- Loading necessary Kernel modules
- Starts swapping,
- Running fsck on all disks,
- Initializing the network
- Mounting disks mentioned in /etc/fstab

This script often invokes various other scripts to do modular initialization. For example, in the common SysV init structure, the directory /etc/rc.d contains a complex structure of subdirectories whose files specify how to startup and shutdown most system services. However, on a boot disk the sysinit script is often very simple.

System Boot Sequence

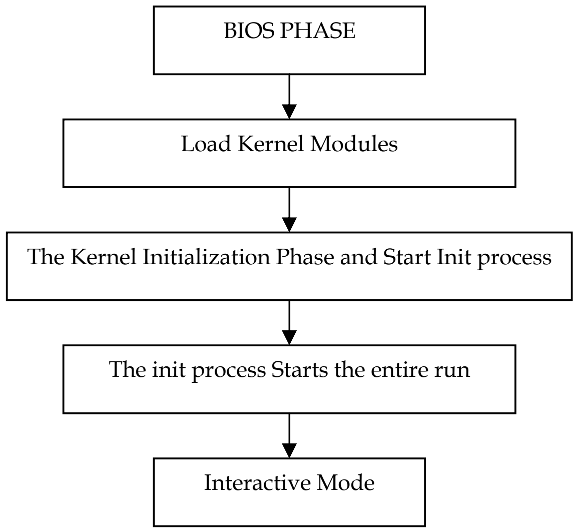

BIOS PHASE
Load Kernel Modules
The Kernel Initialization Phase and Start Init process
The init process Starts the entire run
Interactive Mode

Figure 7.1

init Process

Init is the program that is run by the kernel at boot time. It is in charge of starting all the normal processes that need to run at boot time. These include the getty processes that allow to login, NFS daemons, FTP daemons.

init is quickly becoming the standard in the world of Unix to control the startup of software at boot time. This is due to the fact that it is easier to use and more powerful and flexible. It resides in /etc/rc.d.

```
/etc/rc.d/init.d
/etc/rc.d/rc0.d
/etc/rc.d/rc1.d
/etc/rc.d/rc2.d
/etc/rc.d/rc3.d
/etc/rc.d/rc4.d
/etc/rc.d/rc5.d
/etc/rc.d/rc6.d
```

The above directories contain scripts, which are controlled by rc.sysinit process.

Note : All the mentioned directories reside directly in /etc in SUN UNIX and SCO UNIX.

Example

/etc/init.d, /etc/rc0.d

init.d contains a variety of scripts. Basically, one script for each service is required to start at boot time or when entering another run level. Services include things like nfs, sendmail, httpd, etc. Services do not include items such as set serial that must only be run once and then exited. Such services are found in rc.local or rc.serial.

Example

```
# /etc/rc.d/init.d/nfs start
# /etc/rc.d/init.d/nfs stop
```

init is the parent process of all the processes. The primary role of this process is to create child processes from the script stored in the file /etc/inittab. This script has entries related to init level through which all corresponding scripts get executed at startup. The init process is located in the /sbin/init.

Init is centrally configured through the /etc/inittab file. Here, the so-called "run levels" are defined. Depending on the entries in */etc/inittab* several scripts are started by init. These script are located in the /sbin/init.d folder.init maintains the process of starting and shutting down.

`/etc/inittab`

The inittab file describes the processes that are started at bootup and during normal operation. The format of the inittab file is as follows.

id:runlevel:action:process

Id	–	Id is unique identifier within the *inittab* file which identifies each entry with a limit of 1-4 characters.
Run level	–	Lists the run level for which the specified action should be taken.
Action	–	Describes the action to be taken.
Process	–	Specifies the process to be executed.

Action fields are as follows:

Respawn	–	The process will be restored whenever it terminates.
Wait	–	The process will be started once when the specified run level is entered and init will wait for its termination.
Boot	–	The process will be executed during the booting of the system . The run level field is ignored.
Initdefault	–	An initdefault entry specifies the run level, which should be entered after system boot. If none exists, init will ask for a run level on the console. The process fields are ignored.
Sysinit	–	The process will be executed during the booting of the system. It will be executed before any boot or bootwait entries. The run level field is ignored.
Powerfail	–	The process will be executed when the power goes off. Init is usually informed about this by a process that takes to a UPS which is connected to the system. Init does not wait for the process to complete.

init process and the /etc/inittab file work in the following manner

`/etc/inittab`

If init default is set to level 3 the init process will run entries with sysinit in the action field and entries with 3 in rstate field

Steps involved in the init process are as follows:

1. The init process reads the inittab file.
2. The init process scans for other default run level by reading the initdefault entry.
3. The init process executes the commands or scripts for entries that have sysinit in the action field.

4. The init process executes the scripts for any entry that has a 3 in the rstate field.

Inittab resembles

```
Id:5:initdefault:
Si:sysinit:/etc/rc.d/rc 0
10:0:wait:/etc/rc.d/rc 0
11:1:wait:/etc/rc.d/rc 1
12:2:wait:/etc/rc.d/rc 2
13:3:wait:/etc/rc.d/rc 3
14:4:wait:/etc/rc.d/rc 4
15:5:wait:/etc/rc.d/rc 1
16:6:wait:/etc/rc.d/rc 1
```

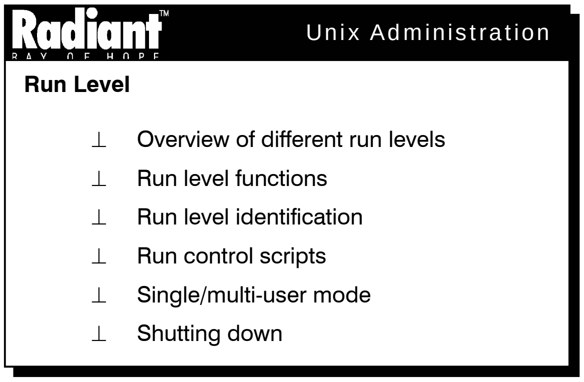
Daemons

These are a group of programs running in the back ground which provides services to the user. All daemons are stored in /bin directory and executed by the services stored in /etc/rc.d/init.d directory. All current running processes are stored in /proc file system.

Example

Services	–	Nfs,Volmgt etc.,
Daemons	–	Vold,nfsd etc.,

7.4 Run Levels



Radiant™ RAY OF HOPE — Unix Administration

Run Level

- ⊥ Overview of different run levels
- ⊥ Run level functions
- ⊥ Run level identification
- ⊥ Run control scripts
- ⊥ Single/multi-user mode
- ⊥ Shutting down

The Unix system environment provides several run levels that determine the various modes of system operation. The Run Levels under Unix, define how the system gets started up. After booting, the system starts as defined in ./etc/inittab. An alternative to this method is assigning a special run level at boot time. The kernel passes any parameters, which it does not need, directly to init. Run levels can be changed while the system is running , as shown below.

```
# init S
```

which brings in a single user mode, which is for maintenance and administration of the system. One can go to runlevel 2 again by the following

```
# init 2
```

Now all the essential programs are started and users can log in and work with the system.

Overview of Different Run Levels

A run level is a software configuration of the system, which allows only a selected group of processes to execute. Init has eight run levels from 0-6 and S or s. Here telinit is the privileged user, which sends a signal to init, telling it which run level to change. For change of each run level in inittab the init process executes a script.

After init is invoked as the last step of the kernel boot sequence it looks for the script /etc/inittab to see if there is any entry of the type initdefault.This initdefault entry determines the initial run levels of the system startup.Run level S or s brings the system to single user mode and does not check for the script /etc/inittab.In the single user mode it is opened on /dev/console. If the system is entering into the multiuser mode for the first time, init performs the boot and bootwait entries to allow file systems to be mounted and all entries that match the run level are processed before getting in to login prompt.

When starting a new process, init first checks whether the file /etc/initscripts exists and uses this script to start the process. Each time a child terminates, init records the facts and reasons for its termination in.

/var/run/utmp and /var/log/wtmp, provided these files exist.

Runlevel Functions

Run Level	**Description**
0	Halt, graceful or proper shutdown
S	Single user mode (To repair corrupted file system)
1	Single user mode (administrative purpose) without network
2	Multi-user mode with network (standard)
3	Multi-user mode with network
4	Unused
5	Takes to Windows environment
6	Reboot or restart the system

Note : In Sun UNIX, the default run level is “3” and run level “5” is to switch off the power of the system. In SCO UNIX the default run level is “S” and run level “2” is for full multi-user mode and run level “3” is the alternative multi-user mode.

Runlevel Identification

To identify the current runlevel and previous run level the command is

```
# runlevel
N 3
```

Here N indicates the previous run level and the second column indicates the current run level .

Note:

In the case of SUN SOLARIS and SCO UNIX to identify the runlevel the command is

```
#  who –r
run-level 2 Feb 29 12:27   2  0  S
```

The first column indicates the current run level, date & time at which this run level has been entered. The next column indicates the current level and the subsequent column gives the number of times the run level has been run and S indicates the previous run level.

Run Control Scripts

The UNIX system provides runcontrol (rc) scripts that help to change the run level. These changes include

1. Checking and mounting file systems.
2. Starting or stopping processes such as print daemons , the NFS client server daemons.
3. Performing housekeeping tasks.

The runcontrol scripts located in the /etc/rc.d directory direct the system to read script files in the related run level directories located in /etc/rc.d/rc#.d. For example , the /etc/rc.d/rc script would read the files located in the /etc/rc.d/rc2.d directory . The /etc/rc.d/rc#.d directories contain scripts that start and stop system services. Files that start with S are used to start processes. Files that start with K are used to kill processes. These files are run in alphanumeric orders. The sequential order is determined by

Uppercase K(0-9)(A-Z,a-z,0-9)

Uppercase S(0-9) (A-Z,a-z,0-9)

File name starting with lowercase letters will not run

Note : To run a startup file, the filename has to be add in sequential order with "S" as a prefix to the filename.

Practice 7.1

The following program shows how to write a run control script

```
# cd /etc/rc.d/rc3.d
# cat > S100sample

echo
echo
echo "##########################################"
echo " DEMO FOR RUNCONTROL SCRIPTS                  "
echo " DEFAULT RUN LEVEL                                          "
echo "##########################################"
echo
echo
echo

# init 6
```

##

```
 DEMO FOR RUNCONTROL SCRIPTS
 DEFAULT RUN LEVEL
##########################################
```

The above procedure will echo the message to indicate the DEFAULT RUN LEVEL" which is kept in /etc/rc.d/rc3.d. Here the default run level is 3 and when init 6(reboot) is given the updation will be done to the system and the appropriate process will be invoked.

Single User Mode

Single user mode refers to the virtual console terminal is that assigned to the system console.It is often referred to as the superuser or maintenance level. In this mode, only minimal processes can be run and the user cannot log in.

Multi-User Mode

Multi user mode means all the defined terminal and daemon processes are running. By default, the system is brought to run level 3(full multi-user mode)

Shutting Down

There are many ways of shutting down the Unix system but that cannot include switching off the power of the computer. In fact, like many other operating systems, Unix intensely uses cache memories to accelerate the input/output processes, mostly with the disks (hard disk, floppy disk...). It can never be ensured that all the data are saved recently are actually on the disk or not. Hence the user has to quite the system properly. Under the root command line, type "init 0","shutdown", "halt", and wait for the computer to print "System Halted" before cutting off the power.

A faster version of the previous solution consists of pressing the ctrl-alt-del keys simultaneously. Then, the system unmounts all peripherals properly. The user has to wait for the computer to reinitialize and then cut off the power.

The shutdown command, which resides in a directory named /sbin can be used as follows:

```
# /sbin/shutdown -h now
h - To halt the system
```

Only the root user can issue the shutdown command. To restart a Unix system, an alternative form of the shutdown command has to be used as shown below:

```
# /sbin/shutdown -r now
r -  To restart the system
```

```
# init 0
# init 6
# halt
# reboot
```

While shutting down a system, Unix automatically logs off all users, terminates all running programs, and closes all open files. Before shutting down a system, each virtual console has to be checked to determine if an important operation is in progress. If so, the shutting of the system should be delayed until the operation is completed.

7.5 Short Summary

- The process of loading the OS into memory and getting it running is call “bootstrapping” often called “booting”

- Runlevel allows only a selected group of processes to execute.

- While shutting down a system, Unix automatically logs off all users, terminates all running programs, and closes all open files.

7.6 Brain Storm

1. What is booting and what are its types?
2. What is the default init level booting?
3. Can the default boot level be changed? If so, how?
4. Explain the sequence of Booting.
5. What is the difference between init 1 and init S?
6. How is the system rebooted and the hardware updated when there is a hardware upgradation?
7. How is the run level identified?
8. What is the difference between single user mode and multi-user mode?
9. What are the various ways of shutting down the system?

Lecture 8

User and Group Management

Objectives

In this lecture you will learn the following

- Able to manage the groups
- Knowing to manage the users

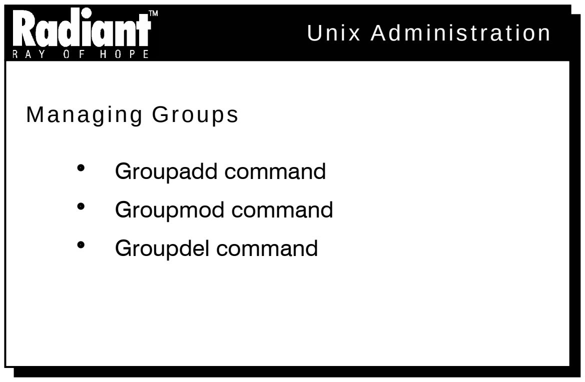
Coverage Plan

Lecture 8

8.1 Snap Shot

All the users acessing the Unix System belong to a particular group. A group is therefore a collection of users. One of the most important tasks of an Administrator is to create and maintain an account of users and groups, assigning passwords to the users and keeping a back of their password expiry and hence controlling the users' access to the systems.

8.2 Managing Groups



Radiant™ RAY OF HOPE

Unix Administration

Managing Groups

- Groupadd command
- Groupmod command
- Groupdel command

Every user on a UNIX system belongs to a group. A group is a collection of individuals. The users in a group may belong to the same department, need access to a particular utility etc. Users can belong to any number of groups. However, at any point of time the user should belong to only one group. This is due to the fact that UNIX allows one group ID per user at any point in time. Groups can have their permissions set to enable their members to gain access to devices, directories, files, filesystems etc., Group information is maintained in the file /etc/group, /etc/gshadow.

The common system administrative tasks to be performed to manage groups are:

- Adding a group
- Adding users to new groups
- Deleting a group

Groupadd command

Syntax

```
groupadd [ Options ]
```

The **groupadd** command creates a new group account using the values specified on the command line and the default values from the system. The new group will be entered into the system files as needed. The options which apply to the **groupadd** command are

-g *gid*

This is the numerical value of the group's ID. The value must be unique, unless the **-o** option is used. The value must be non-negative. The default is to use the smallest value which is greater than 500. Moreover, this value must be greater than the group ID value of the already existing groups. Values between 0 and 499 are typically reserved for *system accounts*.

-r

This flag instructs **groupadd** to add a *system account*. The first available *gid* which is lower than 499 will be automatically selected unless **-g** option is givenalso on the command line. Related files to be updated when the groupadd command is used are

```
/etc/group - group account information
/etc/gshadow - secure group account information
```

Practice 8.1

The following shows how to create groups

```
# groupadd –g 515 marketing
# groupadd –g 516 sales
# groupadd –g 517 technical
# groupadd –g 518 student
# groupadd –g 519 manager
# groupadd –g 520 dummy1
# groupadd –g 521 dummy2
# groupadd –g 522 dummy3
```

The above command will create groups for marketing, sales, technical, student, manager, dummy1, dummy2, and dummy3 with unique ids.

The Groupmod command

The groupmod command modifies the system account files to reflect the changes that are specified on the command line.

Syntax

```
groupmod [ options ]
```

The options which apply to the groupmod command are

`-g gid`

This is used to specify the numerical value of the group's ID. This value must be unique. The value must be non-negative. The values between 0 and 499 are typically reserved for system groups.

`-n group_name`

This is used to change the name of the group from group to group_name.

Related files to be updated when the groupmod command is used are

```
/etc/group - group information
/etc/gshadow - secure group information
```

Practice 8.2

The following shows how to modify group names and group ids.

```
# groupmod –g 515 –n mark marketing
# groupmod –g 516 –n sale sales
# groupmod –g 517 –n tech technical

# groupmod –g 528 student
# groupmod –g 529 manager
```

The above command will modify group ids for student and manager and modify the group names for marketing, sales and technical as mark, sale and tech respectively

The Groupdel command

The groupdel command modifies the system account files, deleting all entries that refer to the group. The named group must exist. All the filesystems should be manualy checked to ensure that no one of the files remain with the named group as the file group ID.

Syntax

```
groupdel [ group ]
```

group - name of the group

Note : The primary group of any existing user may not be removed. But the user must be removed before removing the group.

Related files to be updated when the groupdel command is used are

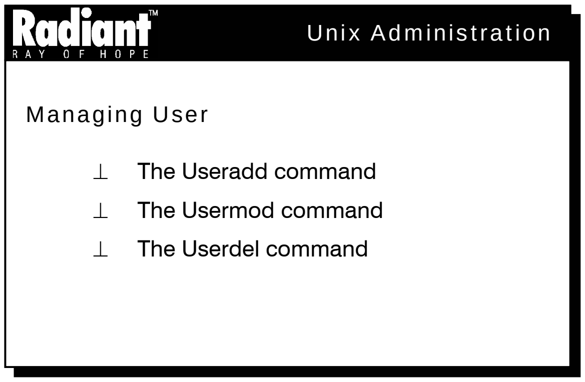
```
/etc/group - group information
/etc/gshadow - secure group information
```

Practice 8.3

```
# groupdel  dummy1
# groupdel  dummy2
# groupdel  dummy3
```

The above command will delete groups dummy1, dummy2 and dummy3.

8.3 Managing User



Radiant™ RAY OF HOPE

Unix Administration

Managing User

⊥ The Useradd command

⊥ The Usermod command

⊥ The Userdel command

All access to a UNIX system is through a user account. Every user except the root account must be set up by the system administrator. Every person using the UNIX system should have his / her unique user name and password.

The common system administrative tasks associated with the management of users are

- Adding users
- Modifying users
- Deleting users

The useradd command

This utility is used to create a user account.

Syntax

```
Useradd  - [ Options ] hostname: username
```

Note : Other flavors of UNIX use something similar; often, it is adduser. The individual must login as root to add users to UNIX workstations.

Useradd has a specific syntax and many options, which may or may not be useful. Frequently, the tool to add new user account works with default files that are normally stored in the /etc directory. In some flavors, these default files may be stored elsewhere.

The default files are called /etc/login.defs. This file determines how the computer adds user accounts. It defines

- The range of acceptable user ID and group ID numbers
- Whether or not a user's password or account expires and if so, when and where the user's mail will be kept
- Whether and where the computer will create a home directory for new users; the minimum acceptable length for user passwords.
- Whether or not the system will remove all processes that are owned by a deleted user

The default files contain information about disk space quotas, standard home directory structures, password aging requirements, and so on. If values for these are not specified when creating a new user, the system will use the default values stored in these files.

The options that apply to this command are

```
-u  user-id
```

It is a numerical id for the user and should be unique. The range of values that are utilized for creating user-ids will start from 500. The upper limit of the range is based on the respective flavors.

Note : The values between 0 to 499 are reserved for system accounts in linux.

```
-g group-id
```

It specifies a numerical id for the group which already exists. The range of values that are utilized for the creation of group-id start from 500 The upper limit based on their respective flavors. By default the group id will be 1 when no value is specified for the group.

```
-G secondary group-id
```

It is used to add a user in secondary groups belong to provided, the same user belong to the primary group.

Note : The user may be allowed to belong to 16 secondary groups.

```
-c  comment
```

This specifies a text string of not more than 512 characters. The string must not contain colons (:) or new lines.

```
-d  home directory
```

This specifies the new home directory of the user. If the home directory is being changed, the contents of the previous home directory are modified only if -m is specified. Directory names must not contain colons (:).

```
-m  -k skeleton directory
```

The user's home directory will be created if it does not exist. The files contained in skeleton_dir will be copied to the home directory if the -k option is used, otherwise the files contained in /etc/skel will be used. Any directories contained in skeleton_dir or /etc/skel will be created in the user's home directory as well. The -k option is only valid in conjunction with the –m option. The default is not to create the directory and not to copy any files.

```
- s default shell
```

This specifies the name of the user's login shell. The default is to leave this field blank, which causes the system to select the default login shell. It is possible to change the shell while creating the user by providing this option with either /bin/ksh or /bin/csh.

Note : The default shell is Bourne shell (/etc/sh) and in the case of LINUX. The default shell is Bourne again shell (/bin/bash) which is derived from Bourne shell.

```
-e [expire-date]
```

This indicates the date on which the user account will be disabled. The date is specified in the format YYYY-MM-DD.

Note : It is only available in LINUX.

```
-f  [Inactive time period]
```

This indicates the number of days for which the account will remain inactive after a password expires until the account is permanently disabled. A value of 0 disables the account as soon as the password, and a value of -1 disables this feature. The default value is -1.

Note : It is only available in LINUX.

```
-r
```

This flag is used to create a system account. if the user with an UID lower than the value of UID_MIN defined in /etc/login.defs. Useradd will not create a home directory for such a user, regardless of the default setting in /etc/login.defs. -m option must be specified to have a home directory to create a system account.

Note : It is only available in LINUX.

username

A user name has a limit of 8 lowercase letters or numbers, but must not begin with a number. In addition, user names cannot include colons (:) (besides the hostname: user syntax used to create a remote account) or new lines.

For distributed accounts, only the user name, comment, password, login shell, home directory, login group, group membership, password, and lock status are valid across the network. For example, the maximum number of failed login attempts cannot be set for a distributed user on a remote system (it only takes effect on the master server).

Related Files to be updated when the useradd command is used are

- /etc/passwd - user account information
- /etc/shadow - secure user account information
- /etc/group - group information
- /etc/default/useradd - default information
- /etc/login.defs - system-wide settings
- /etc/skel - directory containing default files

To Add a User

```
# useradd –u 501 –g 501 –G 502,503,504,505 –c “demo for adding new  user”
         -d /home/demo1 –m –s /bin/sh  -e 2001-03-01 –f 5 –r demo1
```

Practice 8.4

```
# useradd – u 525 –g 515 –d /home/sudha –m –s / bin/sh sudha
# useradd – u 526 –g 516 –d /home/raja –m –s / bin/sh raja
# useradd – u 527 –g 517 –d /home/ravi –m –s / bin/sh ravi
# useradd –  u 528 –g 528 –d /home/ragu –m –s / bin/sh –e 25-09-2001 ragu
# useradd – u 529 –g 529 –d /home/ramesh –m –s / bin/sh ramesh
# useradd – u 530 –g 520 –d /home/dummy1 –m –s / bin/sh dummy1
# useradd – u 531 –g 521 –d /home/dummy2 –m –s / bin/sh dummy2
# useradd – u 532 –g 522 –d /home/dummy2 –m –s / bin/sh dummy2
```

```
set password for sudha, raja, ravi, ragu, ramesh, dummy1, dummy2, dummy3
```

The above command will create users sudha, raja, ravi, ragu, ramesh, dummy1, dummy2 and dummy3 with respective users ids and group ids. Finally you have to create password for all the above users. For user ragu we are assinging expiry date.

The usermod command

The usermod command modifies the system account files to reflect the changes that are specified on the command line.

Syntax

```
Usermod [options ] login
```

The options which apply to the usermod command are as follows:

`-c comment`

This specifies new information about the user's password file comment field.

`-d home_dir`

This indicates the user's new login directory. If the -m option is given, the contents of the current home directory will be moved to the new home directory, which is created if it does not exist already.

`-e expire_date`

This specifies the date on which the user's account will be disabled.   The date is specified in the format YYYY- MM-DD.

`-f inactive_days`

This specifies the number of days for which the account will remain inactive after the password expires until the account is permanently disabled. A value of 0 disables the account as soon as the password expires, and a value of -1 disables this feature.The default value is -1.

`-g initial_group`

This specifies the group name or number of the user's new initial login group. The group name must exist. A group number must refer to an already existing group. The default group number is 1.

`-G group,[...]`

This indicates a list of supplementary groups to which the user belongs. Each group is separated from the next by a comma, with no intervening whitespace. The groups are subject to the same restrictions as the group given with the -g option. If the user is currently a member of a group which is not listed, the user will be removed from the group.

`-l login_name`

This is used to change the name of the user from login to login_name. In particular, the user's home directory name should probably be changed to reflect the new login name.

`-s shell`

This is used to specify the name of the user's new login shell.   Setting this field to blank causes the system to select the default login shell.

`-u uid`

This specifies the numerical value of the user's ID.This value must be unique, unless the -o option is used. The value must be non-negative. Values between 0 and 499 are typically reserved for system accounts. Any files which the user owns that are located in the directory tree rooted at the user's home directory will have the file user ID changed automatically.   Files that are outside the user's home directory must be altered manually.

`-L`

This option locks a user's password. This puts a '!' in front of the encrypted password and effectively disables it.

`-U`

This option is used to unlock a user's password. This removes the '!' in front of the encrypted password.

Note : usermod does not allow the name of a user who is logged in to be changed. It should be ensured that the named user is not executing any process when this comand is issued. The owner of any crontab files should be changed manually. Changes involving NIS should be made on the NIS server.

Related files to be updated when the usermod command is used are

```
       /etc/passwd - user account information
       /etc/shadow - secure user account information
       /etc/group - group information
```

Practice 8.5

```
# usermod – G 515,517 raja

# usermod – s /bin/bash sudha

# usermod – l ravi

# usermod – u ravi
```

The above command will modify user raja by adding two secontary groups. For user sudha we are changing the default login shell to bash. Using the usermod locking and unlocking the user ravi.

The Userdel command

The userdel command modifies the system account files, deleting all entries that refer to login. The named user must exist.

Syntax

```
userdel [-r] login
```

There is only one option that is used with this command.

`-r`

This option is used to remove files in the user's home directory along with the home directory. Files located in other file systems will have to be searched and deleted manually.

Note : Userdel does not allow an account to be removed if the user is currently logged in. Any running process, which belongs to an account that is being deleted has to be killed. Any NIS attributes on an NIS client need not be removed. This must be performed on the NIS server.

Related files to be updated when the userdel command is used

```
/etc/passwd - user account information
/etc/shadow - secure user account information
/etc/group - group information
```

Practice 8.6

```
# userdel – r dummy1
# userdel – r dummy2
# userdel – r dummy3
```

The above command userdel will delete the user dummy1,dummy2 and dummy3 and also removing the respective users home directories and files.

The Password File

This is used to change passwords for user and group accounts. A normal user may only change the password for his own account, the superuser may change the password for any account. The administrator of a group may change the password for the group. passwd also changes the account information, such as the full name of the user, the login shell, or password expiry dates and intervals.

Password Changes

The user is first prompted for the old password. This password is then encrypted and compared against the stored password. The user has only one chance to enter the correct password. The super user is permitted to bypass this step so that the forgotten passwords may be changed. After the password has been entered, password-aging information is checked to see if the user is permitted to change the password at that particular time. If not, passwd refuses to change the password and exits. The user is then prompted for a replacement password. This password is tested for complexity.

As a general guideline, passwords should consist of

6 to 8 characters including one or more from each of the following sets:

- Lower case alphabets
- Upper case alphabets
- Digits 0 - 9

Care must be taken not to include the system default erase or kill characters. passwd will reject any password which is not suitably complex. If the password is accepted, passwd will prompt again for the password and compare the second entry against the first. Both entries are required to match in order to change the password.

Syntax

```
Passwd [options] username
```

The options that can be used with this command are

```
-l
```

This option is used to lock the specified account and it is available to only the root user. The locking is performed by rendering the encrypted password into an invalid string (by prefixing the encrypted string with an !).

```
-u
```

This is the reverse of the previous option. It unlocks the account password by removing the ! prefix. This option is available to only the root user. By default, passwd will refuse to the create a passwordless account (it will not unlock an account that has only "!" as a password). The force option -f will override this protection.

-d

This is a quicker way of disabling a password for an account. It will set the named account passwordless. This option is only available to the root user.

-S

This will output a short information about the status of the password for a given account. It is available to the root user only.

Related files to be updated when the passwd command is used are

/etc/pam.d/passwd

Note : In the case of SUN and SCO /etc/passwd

Practice 8.7

```
# passwd –L sudha
# passwd –U sudha
# passwd –d ramesh
# passwd –s ragu
```

The above command passwd will lock and unlock user sudha, disable the user ramesh and print the status for the user ragu.

The shadow file

The /etc/shadow file is an access-restricted ASCII system file that stores the users encrypted passwords and related information. The shadow file can be used in conjunction with other shadow sources, including the NIS maps, passwd by name, passwd uid and the NIS+ table passwd . The fields for each user are separated by colons. Each user is separated from the next by a new line. Unlike the /etc/passwd file, /etc/shadow does not have general read permission.

Each entry in the shadow file has the form:

```
username:password:lastchg:min:max:warn:inactive:expire:
```

flag

The fields are defined as follows:

username

This specifies the user's login name (UID).

password

This specifies a 34-character encrypted password for the user, a *lock* string to indicate that the login is not accessible, or no string, which shows that there is no password for the login.

lastchg

The number of days between January 1, 1970, and the date that the password was last modified by the user.

min

The minimum number of days within which a user can change password.

max

The maximum number of days during which the password remains valid.

`warn`

The number of days before the password expires that the user is warned. Indicates the expiry of password.

`inactive`

This indicates the number of days of inactivity allowed for the user.

`expire`

This indicates an absolute date specifying when the login may no longer be used.

`flag`

This indicates that it is reserved for future use, and set to zero. Currently it is not used.

Related files get updated when the passwd command is used

```
/etc/pam.d/passwd
/etc/shadow - shadow password file
```

Note : In the case of SUN and SCO. /etc/passwd. The encrypted password consists of 34 characters that are chosen from a 64-character alphabet (., /, 0-9, A-Z, a-z). In order to make the system administration manageable, /etc/shadow entries should appear exactly in the same order as /etc/passwd entries; this includes "+" and "-" entries if the compute source is being used.

8.4 Short Summary

- The groupadd command creates a new group account by using the values specified on the command line and the default values from the system
- The groupmod command modifies the system account files to reflect the changes that are specified on the command line
- The groupdel command modifies the system account files, deleting all entries that refer to the group
- The tool which is used to create user account in LINUX is called useradd
- The usermod command modifies the system account files to reflect the changes that are specified on the command line
- The userdel command modifies the system account files, deleting all entries that refer to login
- The password files changes the passwords for user and group accounts
- The shadow file is an access–restricted ASCII system file that stores the user's encrypted passwords and related information

8.5 Brain Storm

1. How are groups added?
2. How are groups modified?
3. How are secondary groups added?
4. What is the use of the /etc/group file?
5. How are the users added?
6. What is the relationship between /etc/password and /etc/shadow files?
7. How are passwords set and validated?

Lecture 9

Device and Disk Management

Objectives

In this lecture you will learn the following

- Knowing about the Devices and Disk management
- About the geometry of the Hard disk.
- Able to create partition and Formatting the hard disk
- About Device naming

Coverage Plan

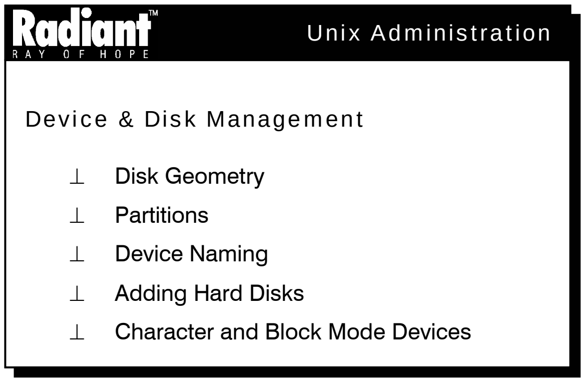
Lecture 9

9.1 Snap Shot

In UNIX, everything attached to the computer is treated as a device.It does not matter whether the device is a terminal, a harddisk, a printer, a CDROM drive or a modem. Everything that accepts and sends data to the operating system is a device.

Device management in LINUX environment includes adding and removing peripheral devices from systems, possibly adding device drivers to support a device and displaying the system configuration information.

9.2 Device and Disk Management



Radiant™ RAY OF HOPE — Unix Administration

Device & Disk Management

- ⊥ Disk Geometry
- ⊥ Partitions
- ⊥ Device Naming
- ⊥ Adding Hard Disks
- ⊥ Character and Block Mode Devices

Disk Geometry

Hard disks are made of one or more magnetic disks, called PLATTERS, which rotate around a central shaft. Each platter contains millions of magnetic particles, whose magnetic field determines the storage of information.

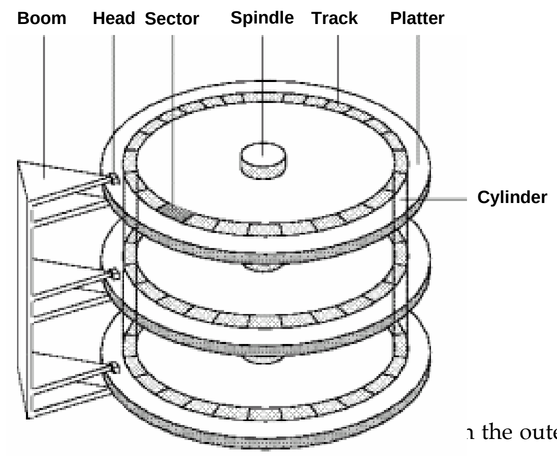


Fig 9.1

ı the outer edge of the disk to the central shaft. At the end of each arm is a read-write head that can access any point on the disk's surface. All the arms are connected together so that they move in unison.

Each platter contains a set of blocks or sectors, which are used to store data. Sector is the smallest unit of one platter. One sector contains 512 bytes. The blocks are organized into circular rings called tracks. A group of sectors together form a track.

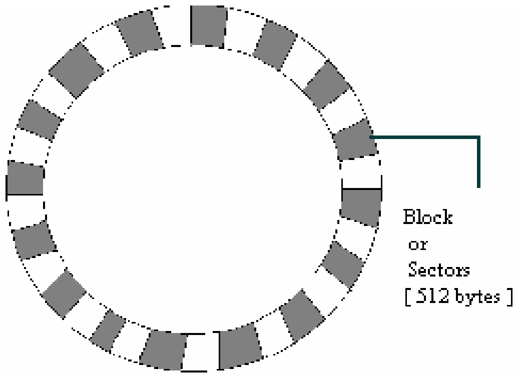


Fig 9.2

Tracks on different platters that are located at the same distance from the edge of their respective disk platter, are combined to form a cylinder. This is diagrammatically represented in Figure No. 9.3

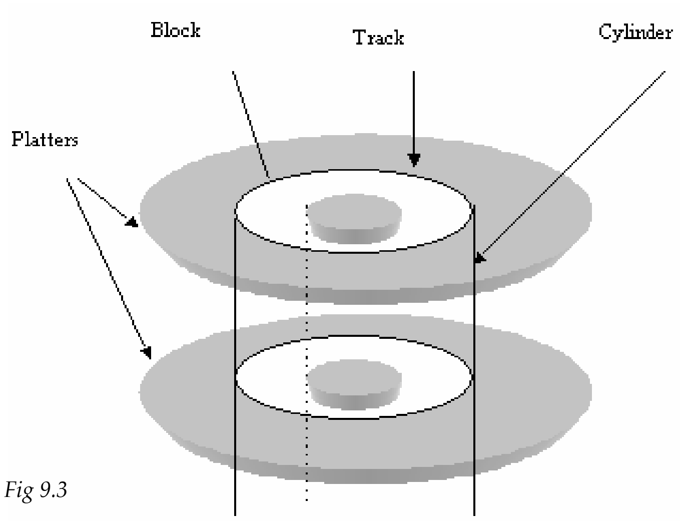


Fig 9.3

Since all the tracks of a cylinder are accessed by the read-write head at the same time, the delay in accessing the data stored on different platters but in the same cylinder is much lesser than if the data is located in different cylinders.

Partitions

A hard disk can be divided into several partitions. Each partition functions as if it were a separate hard disk. So, one hard disk can be divided into two partitions to have two operating systems on it. Each operating system uses its partition according to its preference without disturbing the other. In this way the two operating systems can co-exist peacefully on the same hard disk.

Floppies are not partitioned. There is no technical reason against this, but since they are so small, partitions would be useful only very rarely. CD-ROMs are also not partitioned, also since it's easier to use them as one big disk, and there is seldom a need to have several operating systems on one.

The MBR, Boot Sectors and Partition Table

The information about how a hard disk has been partitioned is stored in its first sector (that is, the first sector of the first track on the first disk surface). The first sector is the master boot record (MBR) of the disk. This is the sector that the BIOS reads and starts when the machine is first booted. The master boot record contains a small program that reads the partition table, checks the type of partition that is active (that is, marked bootable), and reads the first sector of that partition, the boot sector (the MBR is also a boot sector, but it has a special status). This boot sector contains a smaller program that reads the first part of the operating system that is stored on that partition (assuming it is bootable), and then starts it.

The partitioning scheme is not built into the hardware, or the BIOS. It is only a convention that many operating systems follow. Some operating systems support partitions, but they occupy one partition on the hard disk, and use their internal partitioning method within that partition. The latter type co-exists with other operating systems (including Linux), and does not require any special measures. But an operating system that does not support partitions cannot co-exist with any other operating system on the same disk .

It is a wise practice to create a physical record of the partition table, so that it can be retrieved when the partition gets corrupted. The relevant information is given by the fdisk -l command:

```
# fdisk -l /dev/hda

Disk /dev/hda: 15 heads, 57 sectors, 790 cylinders
Units = cylinders of 855 * 512 bytes


Device Boot       Begin    Start  End  Blocks   Id  System
/dev/hda1           1        1     24   10231+  82  Linux swap
/dev/hda2          25       25     48   10260   83  Linux native
/dev/hda3          49       49    408  153900   83  Linux native
/dev/hda4         409      409    790  163305    5  Extended
/dev/hda5         409      409    744  143611+  83  Linux native
/dev/hda6         745      745    790   19636+  83  Linux native
#
```

Extended and Logical Partitions

Originally, the partitioning scheme for PC hard disks allowed only four partitions which turned out to be insufficient when more than four operating systems (Linux, MS-DOS, OS/2, Minix, FreeBSD, NetBSD, or Windows/NT, to name a few) were required. Sometimes it is wise to have several partitions for one operating system. For example, in Linux, placing the swapspace in it's own partition increases its speed than placing it in the main partition.

To overcome this design problem, extended partitions were invented. This allows partitioning a primary partition into sub-partitions. The primary partition thus subdivided is the extended partition; the sub partitions are logical partitions. They behave like primary partitions, but are created differently. There is no difference in speed between them.

The partition structure of a hard disk appears as in the following figure below. The disk is divided into three primary partitions, the second of which is divided into two logical partitions. But part of the disk is not partitioned. The disk as a whole and each primary partition has a boot sector.

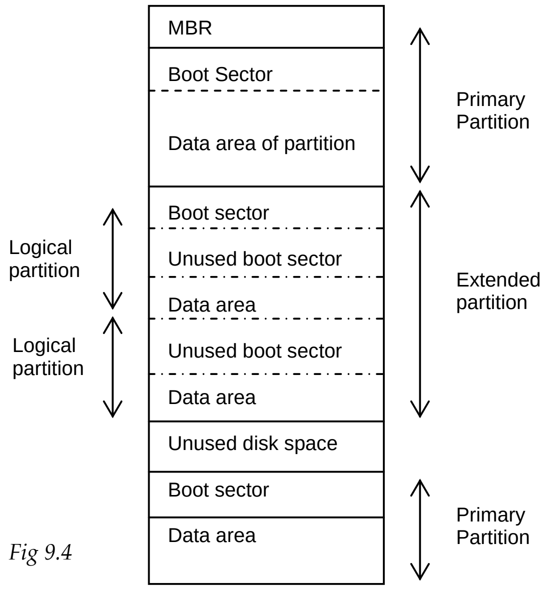


Fig 9.4

Partitioning a Hard Disk

There are many programs to create and remove partitions. Many of the programs are called fdisk, in Linux and its variations. Details on using the Linux fdisk are given on its man page. The cfdisk command is similar to fdisk, but has a full-fledged (full screen) user interface.

When using IDE disks, the boot partition (the partition with the bootable kernel image files) must be completely within the first 1024 cylinders. This is due to the fact that the disk is utilized through the BIOS during boot (before the system goes into protected mode), and BIOS cannot handle more than 1024 cylinders. It is sometimes possible to use a boot partition that is only partly within the first 1024 cylinders. This works as long as all the files that are read with the BIOS are within the first 1024 cylinders. Since this is difficult to arrange, the user might not know when a kernel update or disk defragmentation would result in an unbootable system. Therefore, the boot partition should be completely within the first 1024 cylinders.

Newer versions of the BIOS and IDE disks can handle disks with more than 1024 cylinders.

Each partition should have an even number of sectors. Since the Linux filesystems use a 1-kilobyte block size, that is, two sectors, an odd number of sectors will result in the last sector being unused.

Changing a partition's size usually requires first backing up the files to be saved from that partition (preferably the whole disk, just in case), deleting the partition, creating a new partition, then restoring everything to the new partition. If the partition grows, the sizes (and backup and restore) of the adjoining partitions need to be adjusted as well.

Since changing the size of partitions is tedious, it is preferable to get the partitions right the first time, or have an effective and easy-to-use backup system. If installation is done from a media that does not require much human intervention (for instance, from CD-ROM, as opposed to floppies), it is often easy to play with different configurations. Since there is no data to be backed up it is not tedious to modify the size of the partition several times.

There is a program for MS-DOS, called fips, which resizes an MS-DOS partition without requiring the backup and restore, but for other filesystems it is still necessary.

Note : In Sun Solaris, the format command can be used to create or modify partitions in a disk. After selecting the disk, the format menu is displayed, partition should be selected to display the partition menu. The required commands are provided in this menu to changeand modify the partitions.

Device Files and Partitions

Each partition and extended partition has its own device file. The naming convention for these files is that a partition's number is appended after the name of the whole disk, with the convention that 1-4 are primary partitions (regardless of the number of primary partitions that are available) and 5-8 are logical partitions (regardless of within which primary partition they reside). For example, /dev/hda1 is the first primary partition on the first IDE hard disk, and /dev/sdb7 is the third extended partition on the second SCSI hard disk.

Formatting

Formatting is the process of writing marks on the magnetic media that are used to mark tracks and sectors. Before a disk is formatted, its magnetic surface is a complete mess of magnetic signals. When it is formatted, some order is brought into the chaos by essentially drawing lines where the tracks go, and where they are divided into sectors. The actual details are not quite exactly like this, but that is irrelevant. A disk cannot be used unless it is formatted.

In MS-DOS, the word formatting is also used to refer to the process of creating a file-system. There, the two processes are often combined, especially for floppies. When the distinction needs to be made, the real formatting is called low-level formatting, while making the file-system is called high-level formatting. In UNIX, the two are called formatting and making a file-system.

For IDE and some SCSI disks the formatting is actually done at the factory and does not need to be repeated. In fact, formatting a hard disk can cause it to work less efficiently, for example because a disk might need to be formatted in a unique way to allow automatic bad sector replacement to work.

Disks that need to be or can be formatted often require a special program because the interface to the formatting logic inside the drive differs from drive to drive. The formatting program is often either on the controller BIOS, or is supplied as an MS-DOS program; neither of these can easily be used from within Linux.

During formatting bad spots might be encountered on the disk, called bad blocks or bad sectors. These are sometimes handled by the drive itself, but if more of them develop parts of the disk should not be used. The logic behind this is built into the file-system. Alternatively, one might create a small partition that covers just the bad part of the disk. This approach might be a good idea if the bad spot is very large, since filesystems can sometimes have trouble with very large bad areas.

Floppies are formatted using fdformat. The floppy device file that is to be used is given as the parameter. For example, the following command would format a high density, 3.5 inch floppy in the first floppy drive:

```
# fdformat /dev/fd0H1440
Double-sided, 80 tracks, 18 sec/track. Total capacity 1440 kB.
Formatting ... done
Verifying ... done
#
```

The above command would format a high density, 3.5 inch floppy in the first floppy drive.

To use an autodetecting device (e.g., /dev/fd0), the parameters of the device have to be set with setfdprm first. To achieve the same effect as above, the following has to be performed:

```
# setfdprm /dev/fd0 1440/1440
# fdformat /dev/fd0

Double-sided, 80 tracks, 18 sec/track. Total capacity 1440 kb.
Formatting ... done
Verifying ... done
#
```

Here the setfdprm, will automatically set the device and when using the fdformat command there is no need to specify particulars of the device.

It is usually more convenient to choose the correct device file that matches the type of the floppy. Note that it is unwise to format floppies to contain more information than what they are designed for.

fdformat will also validate the floppy, i.e., check for bad blocks. It will try a bad block several times (the drive noise changes dramatically). If the floppy is only marginally bad (due to dirt on the read/write head, some errors or false signals), fdformat would not complain, but a real error will abort the validation process. The kernel will print log messages for each I/O error that it finds. These messages will go to the console or, if syslog is being used, to the file /usr/log/messages. fdformat itself does not indicate the error.

```
# fdformat /dev/fd0H1440
Double-sided, 80 tracks, 18 sec/track. Total capacity 1440 kB.
Formatting ... done
Verifying ... read: Unknown error
#
```

Here the errors indicate the bad floppy and the format process is incomplete. Hence the floppy cannot be used.

The badblocks command can be used to search any disk or partition for bad blocks (including a floppy). It does not format the disk, so it can be used to check even existing filesystems. The example below checks a 3.5 inch floppy with two bad blocks.

```
# badblocks /dev/fd0H1440 1440
718
719
#
```

Badblocks command outputs the block numbers of the bad blocks that are found. Most filesystems can avoid such bad blocks. They maintain a list of known bad blocks, which is initialized when the filesystem is made, and can be modified later. The initial search for bad blocks can be done by the mkfs command (which initializes the filesystem), but later checks should be done with badblocks and the new blocks should be added with fsck.

Many modern disks automatically notice bad blocks, and attempt to fix them by using a special, reserved good block. This is invisible to the operating system. Even such disks can fail, if the number of bad blocks grows very large.

A Note on Formatting of Disks

Earlier disks were required to be formatted and checked for bad blocks. The procedure of formatting entailed writing the head, track, and sector in a sector preamble and a checksum in the post amble to every sector on the disk.

At the same time, sectors that were unusable due to flaws in the disk surface were marked and depending on the type of disk, an alternate sector mapped into its place.

But now-a-days both the SCSI and IDE disks are pre-formatted from the factory. Moreover, they transparently handle bad blocks on the disk and remap them without any assistance from the operating system.

Device Naming

Most of the implementations of Unix automatically create the correct device entry when the system is made to boot with the new drive attached. Once this entry is created, permissions should be checked. Only root should be given read and write access to it.

Comparison of device names

	Linux	**Sco**	**Solaris 2.X**
Disk *	/dev/hda0 /dev/sda0	/dev/[r]dsk/c#b#t#d#s# /dev/[r]dsk/c#b#t#d#s#	/dev/dsk/c0t0d0s0
Floppy	/dev/fd0		/dev/fd0, /dev/diskette0
Tape, rewinding **	/dev/ht0 /dev/st0		/dev/rmt/0
Tape, non-rewinding ** **	/dev/nht0 /dev/st0 /dev/tape		/dev/rmt/0n

Table 9.1

dmesg command - Identifies the devices that are connected to the system, from the output of *dmesg* command the logical disk names can be identified.

Adding Hard disks

Hard disks can be added to a system as follows:

- The system with the new hard disk installed is rebooted.
- Linux will auto detect the disk.
- Use the‘dmesg’ command to find out the device that is being used by the disk.

For IDE machines

```
primary master =>/dev/hda
primary slave =>/dev/hdb
secondary master=>/dev/hdc
secondary slave =>/dev/hdd
```

- Use cfdisk to partition the disk.
- Use mke2fs to create a file system
- Edit fstab to make mounting easier.

SCSI hard disks are named with sdx, where x is a hard disk letter. The disk with the lowest SCSI ID on the first controller will become sda, the next one will become sdb, and so on.

Hard disks, are called hda, hdb, hdc, and so on. Hda is the master disk on the first controller, hdb is the slave disk on the first controller, hdc is the master disk on the second controller, and so on.

GNU/Linux systems generally use a partition scheme where a hard disk can have up to four primary partitions. To make more partitions one of these should be made as an extended partition where several logical partitions can be made. The partitions take the name of the disk they belong to, and a number. The first primary partition on the first SCSI disk is therefore sda1, the second primary partition is sda2, and so on. The first and second logical partition on an extended partition on the first SCSI disk are sda5 and sda6, and so on.

cfdisk can be started from the command line with the command cfdisk /dev/sdx

where x is the SCSI hard disk letter, like a, b, c, d, etc. So to partition the first hard disk on the SCSI controller, the command cfdisk /dev/sda should be used.

When cfdisk is started an interface is obtained where the current partition table is listed with the names and, is started command buttons on the bottom of the screen. To change between partitions, the up and down arrow keys have to used. To change between commands, the left and right arrow keys are to be used.

To delete an existing partition, the up and down keys should be highlighted, the delete command should be selected with the left and right arrows keys, and Enter should be pressed.

To make a new partition, the New command should be selected with the left and right arrow keys, and entered. Choice is obtained between a primary and a logical partition. For a logical partition, the program will automatically make an extended partition. Then the size of the partition (in MB) must be chosen. If the value cannot be entered in MB, the user must return to the main screen with the Esc key, and select MB with the Units command.

To set the type of a partition, for bootable PReP, Linux swap or Linux ext2, the actual partition needs to be selected, and Type command must be selected. A list of different types is obtained. On pressing space more choice is obtained. The type that is needed is to be typed, and the number at the prompt should be entered.

To be able to boot from a primary partition, it should be made bootable. The actual partition should be highlighted and the Bootable command should be selected.

After choosing the layout of the disk, the Write command should be selected. The partition table will be written to disk that will destroy all data on partitions has been deleted or changed.

To exit the program, the Quit command should be selected.

Note : In Solaris, if a new hard disk is added, a reconfiguration file is created in the root directory to detect the presence of new hardware, then the system should be shutdown. Connect the new disk to the system and reboot. The disk will be identified by the system. If required partitions should be created using format command, file system should be created using newfs. Make necessary changes in the /etc/vfstab for mounting the new created file system.

The following program shows how to create a /reconfiguare file.

i) Create a /reconfigure file
   # touch /reconfigure
   #init 0

ii) Connect the new disk to the system

iii) Turn on the power
iv) Boot the system

v) Create partitions on the disk using the format command (if required)
```
# format
```

vi) Create file system
```
# newfs/dev/rdsk/c # t # d # s #
```

vii) Make necessary changes in the /etc/vfstab file

Instead of creating a /reconfigure file a reconfigure boot can be performed by using b –r command at the PROM level in order to make the system recognize the presence of a new device. Hard disk can be added without shutting down the system. In this case•, drvconfig and disks commands have to be used to identify the new disk. These commands should be invoked with no parameters. Never attempt a low-level format on IDE disk. It will destroy the disk itself.

Character and Block Mode Devices

Any device that is attached to the system that is running on Linux is treated as a device by the operating system. It does not matter whether the device is a terminal, a hard disk, a printer, a CD-ROM drive, or a modem. Everything that accepts or sends data to the operating system is a device.

The concept of treating anything on the system as a device is one of the benefits of the UNIX architecture. Each device has a special section in the kernel, called a device driver, which includes all the instructions necessary for Linux to communicate with the device. When a new device is developed, it can be used with Linux by writing a device driver, which is usually a set of instructions that explains about ways to send and receive data.

Device drivers allow the Linux kernel to include only the operating system and support software. By having the instructions for talking to devices within a set of files, they can be loaded as needed (in the case of rarely-used devices), or kept in memory all the time when the operating system boots. As refinements are made to a peripheral, small changes to the device driver file can be linked to the kernel to keep the operating system informed of the new features and capabilities.

When an application instructs a device to perform an action, the Linux kernel passes the request to the device driver and allows it to handle the communications.

Linux keeps device files in the /dev directory by default and convention. It is permissible to keep device files anywhere on the file system, but keeping them all in /dev makes it obvious that they are device files.

Every type of device on the Linux system communicates in one of the two ways: character by character or as a set of data in a predefined chunk or block. Terminals, printers, and asynchronous modems are character devices, using characters sent one at a time and echoed by the other end. Hard drives and most tape drives, on the other hand, use blocks of data, because this is the fastest way to send large chunks of information. These peripherals are called either character mode or block mode devices, based on the mode of communication.

Note : Another way to differentiate between character and block mode devices is by the way in which buffering to the device is handled. Character mode devices do their own buffering. Block mode devices, which usually communicate in chunks of 512 or 1,024 bytes, have the kernel perform the buffering. Some devices can be both character and block mode devices. Some tape drives, for example, can handle both character and block modes, and therefore have two different device drivers. The device driver that is used depends on how the user wants to write data to the device.

The device file has all the details about whether the device is a character mode or block mode device. There is an easy way to find out the type of device of a peripheral. The output of the listing command shows file permissions (such as ls -l). If the first character is a b, the device is a block mode device; a c indicates a character mode device.

Device files are usually named to indicate the type of device. Most terminals, for example, have a device driver with the name tty followed by two or more letters or numbers, such as tty1, tty1A, or tty04. The letters tty identify the file as a terminal (tty stands for teletype), and the numbers or letters identify the specific terminal referred to. When coupled with the directory name /dev, the full device driver name becomes /dev/tty01.

Major and Minor Device Numbers

There might be more than one device of the same type on a system. For example, a Linux system might have a multiport card (multiple serial ports) with 10 Wyse 60 terminals hanging off it. Linux can use the same device driver for each of the terminals because they are all the same type of devices.

However, there must be a method for the operating system to differentiate which of the 10 terminals is to be addressed. This is where device numbers are used. Two device numbers identify each device: The major number identifies the device driver to be used, and the minor number identifies the device number. For example, the 10 Wyse 60 terminals on the multiport card can all use a device file with the same major number, but each will have a different minor number, thereby uniquely identifying it to the operating system.

Every device on the system has both major and minor device numbers assigned uniquely. If two devices are assigned the same number, Linux cannot properly communicate with them.

Some devices use the major and minor device numbers in a strange way. Some tape drives, for example, use the minor number to identify the density of the tape and adjust its output in that manner.

Device files are created with the command mknod (make node) and removed with the standard rm command.

The mknod Command

The mknod (make node) command is used for several different purposes in Linux. It can create a FIFO (first in first out) pipe or a character or block mode device file. The format of this command is

```
mknod [options] device b|c|p|u major minor
```

The options can be one of the following:

– help	—	displays help information and then exits.
-m [mode]	—	sets the mode of the file to mode instead of the default 0666 (only symbolic notation is allowed).
– version	—	displays version information, then exits.

The argument after the device or pathname specifies whether the file is a block mode device , character mode device , FIFO device (p), or unbuffered character mode device (u). One of these arguments must be present on the command line.

Following the type of file argument are two numbers for the major and minor device numbers assigned to the new file. Every device on a UNIX system has a unique number that identifies the type of device (the major number) and the specific device itself (the minor number). Both the major and minor numbers must be specified for any new block, character, or unbuffered mode device. Device numbers are not specified for a type p device.

9.3 Short Summary

- The hard disk is divided into several partitions
- LINUX automatically detects the presence of hard disk.
- cfdisk utility is used to partition disks
- fdformat is used to format the floppy disk.

9.4 Short Summary

1. Describe the hard disk geometry.
2. Explain the cfdisk utility in Linux.
3. What does the dmesg command give?

Lecture 10

Introduction to File System

Objectives

In this lecture you will learn the following

- About File System
- Knowing about Local Based File System, RAM Based File System and Network File System
- Understand the Types of File Systems
- Mounting the Local Based File System

Coverage Plan

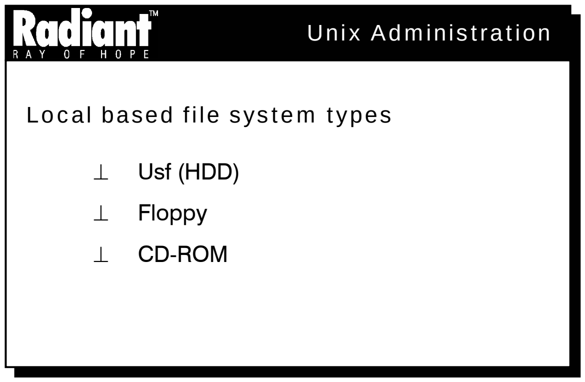
Lecture 10

10.1 Snap Shot

The file system is the primary means of file storage in UNIX. Each file system houses directories, which, as a group, can be placed almost anywhere in the UNIX directory tree. The topmost level of the directory tree, the root directory, begins at /. In Linux the file system that is used is *ext2.*

A file system, however keeps its data on disks.. These disks are then broken into partitions, each varying in size depending on the needs of the administrator. It is on each partition that the actual file system is laid out. Within the file system, there are directories, subdirectories, and, finally, the individual files.

10.2 Local Based File System Types

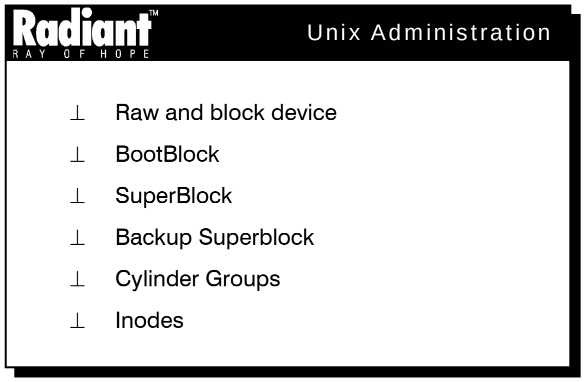

Radiant™ RAY OF HOPE — Unix Administration

Local based file system types

- ⊥ Usf (HDD)
- ⊥ Floppy
- ⊥ CD-ROM

These File System types are implemented on the hard disk. (Disk based,CD-ROM, or Floppy). Ext2, is the default Unix File System in the case of Linux . It is used to optimize disk performance through the use of cylinder groups, sectors, tracks and data Blocks.

10.3 The ext2 File System


Radiant™ RAY OF HOPE — Unix Administration

- ⊥ Raw and block device
- ⊥ BootBlock
- ⊥ SuperBlock
- ⊥ Backup Superblock
- ⊥ Cylinder Groups
- ⊥ Inodes

Raw and Block Devices

Raw device has a defined beginning and size. But no File System is created for it. When a File System is created for a raw device, it becomes a Block device. Hence, a Block device has a File System.

The Boot Block

This boot block can be found only in the root File System. There is some space allocated for the boot Block at the beginning of each File System.

The SuperBlock

The SuperBlock is located next to the boot Block. The SuperBlock contains a table of information about the File System. The following information is included in the SuperBlock:

- Number of data Blocks, cylinder groups
- Size of the data Block
- Mount, where the File System gets mounted

Backup SuperBlocks

This contains the backup of the superBlock. It is replicated in each and every dataBlock and cylinder group. This enables to protect against data loss. These Blocks are created when File Systems are created.

Cylinder Groups

The HDD are partitioned by cylinder groups. The data of the File System will get stored across several cylinder groups. The cylinder group contains inodes, dataBlocks, free Blocks and free inodes.

Inodes

An inode maintains information about each file The inode contains information about the type of file, the access mode of the file, the user and group to which the file belongs, the size of the file, the time at which it was last accessed and modified, the number of data Blocks which are allocated and used for that particular file and pointers.

Depending on the type of File System, the inode can contain more than 40 pieces of information. The fields that are of concern for the system administrator are as follows:

mode	—	Indicates the permission mask and type of the file.
link count	—	Indicates the number of directories that contain an entry with this inode number.
user ID	—	Indicates the ID of the file’s owner.
group ID	—	Indicates the ID of the file’s group.
size	—	Indicates number of bytes in this file.
access time	—	Indicates the time at which the file was last accessed.
mod time	—	Indicates the time at which the file was last modified.
inode time	—	Indicates the time at which this inode structure was last modified.
Block list	—	Indicates a list of disk Block numbers which contain the first segment of the file.
indirect list	—	Indicates a list of other Block lists.

The fields mode, link count, user ID, group ID, size, and access time are used when generating file listings.

Note : The inode does not contain the name of the file. This information is held in the directory file .

Pointers

There are two types of pointers: Direct pointer and indirect pointer.

Direct Pointer

These pointers directly refer to the data Blocks. There are 12 direct pointers that directly reference the data Blocks of a file up to 96Kb.

Indirect Pointer

There are three types of indirect pointers. They are described below:

Single Indirect

It refers to a File System Block containing pointers to data Blocks. It will contain 4-Kbyte data Blocks and it will point up to an additional 8 Mb of data.

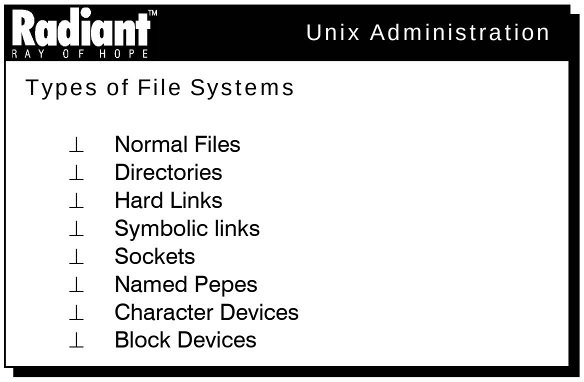
Double Indirect

These pointers point up to an additional 16 Gbytes of data.

Triple Indirect

These pointers can refer up to an additional 35 Tbytes of data.

10.4 Types of File Systems



Unix Administration

Types of File Systems

⊥ Normal Files
⊥ Directories
⊥ Hard Links
⊥ Symbolic links
⊥ Sockets
⊥ Named Pepes
⊥ Character Devices
⊥ Block Devices

Generally speaking, each operating system has its own unique filesystem. Files created on one operating system are not readable on any other operating system. At present, Unix supports more filesystems than any other OS. When Unix is installed the filesystem is created on the hard disk. As far as Unix is concerned, a filesystem is a device that is formatted to store files that can be randomly accessed. This includes the hard disk partitions, floppy disks, CD-ROMs, but not tape drives (which are accessed sequentially and thus cannot contain a filesystem per se).

The current list of major filesystems supported by Unix are as follows:

- Linux Swap Filesystem - **swap**
- MS-DOS Filesystem - **msdos**
- Network File System (NFS) - **nfs**
- Novell Filesystem - **ncpfs**
- NT - **ntfs**
- Second Extended Filesystem - **ext2** (Linux standard filesystem)
- Uniform Filesystem - **ufs** Used by BSD , SunOS

Types of Files are available in 8 flavors:

- Normal Files
- Directories
- Hard Links
- Symbolic links
- Sockets
- Named Pipes
- Character Devices
- Block Devices

Normal Files

These are the files that are used the most. They can be either text or binary files. However, their internal structure is irrelevant from a System Administrator's standpoint. The characteristics of a file are specified by the inode in the File System that describes it. An `ls -l` on a normal file will look as follows:

```
-rw———    1 kishore  unix         42 May 12 13:09 hello
```

Directories

These are special kind of files that contain a list of other files. Although there is a one-to-one mapping of inodes to disk Blocks, there can be a many-to-one mapping from a directory entry to an inode. When viewing a directory listing using the `ls -l` command, the directories can be identified by their permissions starting with the `d` character. An `ls -l` on a directory will look as shown below:

```
drwx———     2 kishore    unix        512 May 12 13:08 public_html
```

Hard Links

A hard link is actually a normal directory entry except that instead of pointing to a unique file , it points to an already existing file . This gives the illusion that there are two identical files when a directory listing is done. Since the system sees the hard-linked file as another file, it treats it as such. This is most apparent during backups because hard-linked files get backed up as many times as there are hard links to them. Because a hard link shares an inode, it cannot exist across File Systems. Hard links are created with the `ln` command. For example, when a directory listing is performed using `ls -l`, the result would be as shown below:

```
-rw———     1  kishore    unix       42 May 12 13:04 hello
```

When `ln hello goodbye` is typed and another directory listing is again performed using `ls -l`, the following can be observed:

```
-rw———     2 kishore    unix        42 May 12 13:04 goodbye
-rw———     2 kishore    unix        42 May 12 13:04 hello
```

Notice how this appears as two separate files that just happen to have the same file lengths. Also note that the link count (second column) has increased from one to two. It can be verified that both the files are actually the same by using `ls -il`:

```
13180 -rw———     2 kishore    unix         42 May 12 13:04 goodbye
13180 -rw———     2 kishore     unix        42 May 12 13:04 hello
```

It can be seen that both point to the same inode, 13180.

Note : Care should be taken when creating hardlinks, especially when hardlinking to a directory. It is possible to corrupt a filesystem by doing so since the hardlink does not contain the fact that the i-node being pointed to needs to be treated as a directory.

Symbolic Links

A symbolic link (sometimes referred to as a *symlink*) differs from a hard link because it does not point to another inode but to another filename. This allows symbolic links to exist across File Systems as well as be recognized as a special file by the operating system:

Example

```
drwx———     2 kishore       icg       512 May 12 13:08 public_html
lrwx———     1 kishore       icg       11 May 12 13:08 www -> public_html
```

Sockets

Sockets are the means for UNIX to network with other machines. Typically, this is done using network ports. However, the File System has a provision to allow for interprocess communication through socket files. If a socket file needs to be removed, the *rm* command has to be used. Socket files are identified by their permission settings beginning with an `s` character. .

Example

```
srwxrwxrwx   1 root     admin           0 May 10 14:38 X0
```

Named Pipes

Similar to sockets, named pipes enable programs to communicate with one another through the File System. The `mknod` command can be used to create a named pipe. Named pipes are recognizable by their permission settings beginning with the `p` character.

Example

```
prw———   1 kishore   unix          0 May 12 22:02 mypipe
```

Character Devices

These special files are typically found in the */dev* directory and provide a mechanism for communicating with system device drivers through the File System one character at a time. They are easily noticed by their permission bits starting with the `c` character.

Example

```
crw-rw-rw-   1 root     wheel     21,   4 May 12 13:40 ptyp4
```

Block Devices

Block devices also share many characteristics with character devices. That is, they too exist in the `/dev` directory and are used to communicate with device drivers, and have major and minor numbers. The key difference is that Block devices typically transfer large Blocks of data at a time versus one character at a time.

Example

```
brw———   2 root     staff     16,   2 Jul 29  1992 fd0c
```

Working with Directories

The directory tree of the Unix system is well organized. Some important directories are:

/	–	Root directory, start of the directory tree
/bin	–	Commands needed to run the system
/dev	–	Device files that represent the system hardware
/etc	–	Important system configuration files
/home	–	The (private) directories of the users
/lib		Shared libraries
/opt	–	Optional software, large systems
/proc	–	The process File System
/sbin	–	Commands reserved for the superuser and needed for system start
/tmp	–	Temporary files
/usr	–	User commands and applications, Configuration files, can be mounted read-only
/usr/bin	–	Publicly accessible commands
/usr/sbin	–	Commands reserved for the superuser
/var	–	Configuration files (linked from /usr)

Creation of new File System

Any partition on a newly partitioned disk needs to have a File System created on it before adding data. The `mkfs` (for make File System) command is used to create the File System on the partition. To create a File System on the disk drive partition, for example, the following command has to be used:

```
# mkfs  [-t filesytemtype]  [-c] [device]
```

The option -t indicates the type of the File System and the option -c indicates the device path on which the new File System is to be created.

Practice 10.1

The following example illustrates how to create a new File System.

```
#   mkfs -t  ext2  -c  /dev/hda*
```

Blocksize=4096,611648

Inodes 1220932 Blocks

61046 Blocks(5.00%) reserved for Super user

32768 inodes per group.

Super Block backups states on Blocks (for fsck)

32768,98304,163840,229376,294912,819200,884736.

mkfs is the command which is used to create a new File System on the device called */dev/hda12.*

Once a new File System is created by using mkfs and the new File System has to be pointed on the mount point. The mount point should be unique.

Note :
In Solaris and SCO *mkfs* is the commmand which is used to create a new File System, *newfs* is a user-friendly command which is used to create a New File System on Solaris.

Syntax:

```
# newfs –s [size of the File System] –d [ device]
```

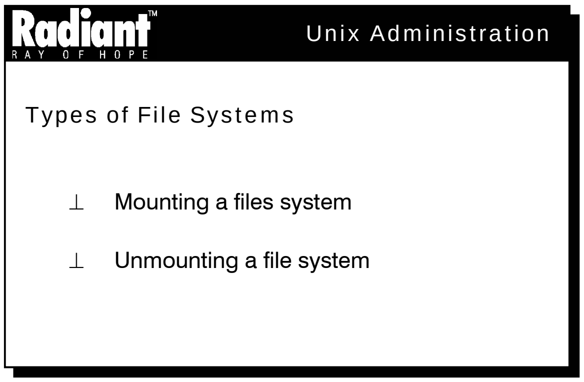
Options:

-s	Indicates the size of the File System.
-d	Indicates the device path where a new filesystem has to be created.

Consider the following example:

```
#  newfs  -s 2000 –d /dev/rdsk/c0d0s0
```

10.5 Mounting the Local Based File System



Radiant™ RAY OF HOPE Unix Administration

Types of File Systems

- ⊥ Mounting a files system
- ⊥ Unmounting a file system

Mounting a File System

Before a File System is mounted a mount point has to be selected. A mount point may exist anywhere in the File System. Mount points are directories. File Systems can be mounted using the **mount** command (which can only be invoked by '`root`', unless the normal users are given permission to use it. All the mounted File Systems are found in ***/etc/mtab***. If **mount** is invoked without any argument the contents of this list is printed to screen. This corresponds to all mounted File Systems. Mounting and Unmounting of the File Systems occur during booting and shutting the system respectively. These are based on the entries in */etc/fstab* .

The *mount* command is used to mount local and remote File Systems. It is used to display the list of File Systems that are mounted as shown below:

```
# mount /dev/device /directory/to/mount
```

`/dev/device` is the name of the device that is to be mounted

`/directory/to/mount` is the directory to be overlayed in the local File System.

There are options that can be passed to the *`mount`* command. The options are

-a	Mount all the File Systems
-v	Print the output in verbose mode
-V	Print the help message to use *Mount* command
-r	Mount the File System with read only permission
-w	Mount the File System with write only permission

There are options that can be passed to the `mount` command. The most important characteristics are specified in the `-o` option. These characteristics are:

rw	read/write
ro	read only
bg	background mount (if the mount fails, place the process into the background and keep trying until success.)

Consider the following example:

```
# mount  -o  rw,bg  /dev/hda4  /mnt
```

All the mounted File System information will stored in ***/etc/mtab..*** /etc/fstab virtual File System table provides entries for mounting File Systems at the time of booting the system.

mount is the command which is used to mount a File System in Solaris. Mounted File Systems are found in */etc/vfstab* in Sun Solaris and in SCO Unix.

Syntax

```
# mount –F [File System type] –o [options] [device to mount]
[mountpoint]
```

Options:

-F	Indicates File System which is used for mounting.*ufs* is the file type in Solaris. *Pcfs* is the File System to access floppy, *hsfs* is the File System to accessing Cd-Rom.
-o	Indicates the permission.
`rw:`	`Read and write permission`
`ro:`	`read only permission.`

```
Consider the following example:
# mount –F ufs –o rw /dev/dsk/c0d0s0 /mnt
```

Mounting a Floppy disk and CD ROM

The floppy disk and CD ROM can be mounted as shown below:

Syntax

```
# mount /dev/device /directory/to/mount
```

Practice 10.2

The following example illustrates the mounting of a floppy disk.

```
# mount  /dev/fd0 /mnt
```

/dev/fd0 is the name of the floppy device to be mounted and the floppy is mounted on the */mnt* directory.

Practice 10.3

The following example illustrates the mounting of a CD ROM disk

```
#  mount  /dev/cdrom0 /mnt
```

`/dev/cdrom0` is the CD ROM device to be mounted and the CD ROM is mounted on the */mnt* directory.

To access the CD ROM disk the following should be done:

```
# cd  /mnt
# ls
# umount /mnt
# eject cdrom0
```

Note :
The eject command is used to eject the CD ROM. Before ejecting the CD ROM, the CD has to be unmounted using the umount command.

All the mounted File System information will be stored in ***/etc/mnttab*** in Sun Solaris and in ***/etc/mnttab*** in SCO Unix.

Linux uses a special file called `/etc/fstab.` This file lists all the partitions that need to be mounted at boot time and the directory where they need to be mounted. Along with that information parameters to the `mount` command can be passed.

Each File System to be mounted is listed in the */etc/fstab* file in the following format:

```
/dev/device     /dir/to/mount   ftype      parameters   fs_freq     fs_passno
```

Practice 10.4

The following example illustrates how to mount a File System.

```
/dev/hda1   /home/kishore          /ext2    rw         0      0
```

/dev/device	–	The device to be mounted, for instance, /dev/hda4.
/dir/to/mount	–	The location at which the File System should be mounted on your directory tree.
Ftype	–	The File System type. This should be 4.2 under SunOS, ufs under Solaris, ext2 under Linux, nfs for NFS mounted File Systems, swap for swap partitions, and proc for the /proc File System. Some operating systems, such as Linux, support additional filesystem types, although they are not as likely to be used.
Parameters	–	The parameters that are passed to mount using the -o option. They follow the same comma-delineated format. An example entry would look like rw,intr,bg.
fs_freq	–	Used by dump to determine whether a File System needs to be dumped.
fs_passno	–	Used by the fsck program to determine the order to check disks at boot time.

Note :
In Sun Solaris, /etc/vfstab is the virtual File System table that provides entries for mounting the File System at the time of booting the system.
#vi /etc/vfstab

<device to mount> <device to *fsck*> < mount point> <*fs* type> <*fsck* pass> <mount at boot> <mount options>

The fields are described below:

Device to mount	:	Is the device to be mounted.
Device to fsck	:	What is the name of the raw device to `fsck`; for a remote mount, the parameter is not applicable, and the entry should be - .
mountpoint	:	Which is the mount point on which the resource is to be mounted. (*/mnt*).

Fstype	:	Which is the type of File System of the resource to be mounted. (*ufs*)
fsckpass	:	Which is the pass number to use for multiple `fsck`'s. For a remote mount, the parameter is not applicable, and the entry should be - .
mntopts	:	The options passed to the `mount` command

Unmounting a File System

Unmounting a File System makes the File System, files and directories inaccessible to the users. U*mount* is the command which is used to unmount the filesystem.

Syntax:

```
# umount [mount point]
```

Practice 10.5

The following command illustrates how to unmount a File System.

```
# umount  /var
```

/var is the File System to be unmounted.

10.6 Common Commands for File System Management

Radiant™ RAY OF HOPE — Unix Administration

Common commands for file System Management

⊥ Managing Disk Use (Tesks)

Managing Disk Use (Tasks)

This chapter describes how to optimize Disk space can be optimized by locating unused files and large directories. This is a list of the step-by-step instructions in this chapter.

1. Displaying Blocks and Files Used
2. Displaying the size of directories
3. Finding Large Files
4. Finding Files That Exceed a Given Size Limit
5. Checking the Size of Directories:
6. Finding and Removing Old and Inactive Files

7. Listing the Newest Files
8. Finding and Remove Old or Inactive Files
9. Remove the inactive files that you listed in the previous step
10. Clearing out temporary directories
11. Finding and delete core files
12. Deleting crash dump files
13. What is fsck?
14. The fsck Utility
15. Where is fsck?
16. When should an fsck run?
17. How to use ***fsck***?

Displaying Blocks and Files Used

The `df` command and its options can be used to report the number of free disk blocks and files.

Syntax:

df [option]

option:

-k - prints the free disk space in terms of k bytes.

Consider the following example

```
# df

Filesystem  1024-blocks   Used    Available       Capacity   Mounted on
/dev/hda3   247871         212909  22161            91%       /
/dev/hda6   50717          15507   32591            32%       /var
/dev/hda7   481998         15      457087           0%        /local

# df  -k

Filesystem   1k-blocks      Used    Available   Use%   Mounted on
/dev/hda7    396623         71930   304212             19%    /
/dev/hda1    101089         2453    93417              3%     /boot
/dev/hda6    1517920        12952   1427860     1%     /home
/dev/hda5    1517920        1276308 164504             89%    /usr
/dev/hda9    54416          18548   33059              36%    /var
```

The columns reported show:

Filesystem	–	Refers to the file system being shown. File systems mounted using NFS are shown as *hostname:/dir/that/is/mounted*
1024-blocks	–	The number of 1 KB blocks the file system consists of. (Its total size.)
Used	–	The number of blocks used.
Available	–	The number of blocks available for use.
Capacity	–	Percentage of partition currently used.

Mounted on – The location in the directory tree where this partition has been mounted on.

The `du` command summarizes disk usage by directory. It recurses through all subdirectories and shows disk usage by each subdirectory with a final total at the end.

Displaying the size of directories

The following command illustrates the disk usage by the directoy.

Syntax

```
du [options] <directory-name>
```

Options

Directory	–	Show usage for the specified directory. The default is the current directory.
-a	–	Show usage for all files, not just directories.
-s	–	Show only the total disk usage.

du

```
409     ./doc
945     ./lib
68      ./man
60      ./m4
391     ./src
141     ./intl
873     ./po
3402  .
```

The first column shows the blocks of disk used by the subdirectories.

The second column shows the names of the subdirectory being evaluated and to see how many kilobytes each subdirectory consumes.

Displaying the Size of Files

You can check the size of files and sort them by using the *ls* command. You can find files that exceed a size limit by using the **find** command.

Change the directory to where the files you want to check are located.

Display the size of the files.

Consider the following example

The following example shows that lastlog, wtmp, and `wtmpx` are substantially larger than the other files in the **/var/adm** directory.

```
#cd /var/adm
#ls –l
total 434
-r—r—r—         1 root other  585872 Jan 28  14:53    lastlog
drwxrwxr-x      2 adm  adm    512  Dec  1    16:35    log
-rw-r—r—        1 root other  562  Jan  2    13:13    messages.3
drwxrwxr-x      2 adm  adm    512  Dec  1    16:35    passwd
drwxrwxr-x      2 adm  sys    512  Jan 28    11:38    sa
```

```
-rw-rw-r—       1 adm  adm 125736  Jan 28    14:53    wtmpx
```

Finding Large Files

1. Change directory to the location which is to be searched
2. The size of files in blocks from largest to smallest can be displayed as shown below.

```
# ls –s | sort -nr | more
```

here

sort –nr Sorts the list of files by block size from smallest to largest.

Consider the following example

In the following example, **wtmpx** and **lastlog** are the largest files in the **/var/adm** directory.

```
# cd /var/adm
# ls -s | sort -nr | more
 320 wtmpx
 128 lastlog
  74 pacct
  56 messages
  30 wtmp
   6 utmpx
   2 utmp
   2 sulog
   2 sa
   2 passwd
   2 log
   0 spellhist
total 624
```

Finding files that exceed a given size limit

To locate and display the names of files that exceed a specified size, the **find** command can be used as shown below.

# find *directory* **-size +***nnn*

Directory – Where is the directory to be searched

-size +*nnn* – Is a number of 512-byte blocks. Files that exceed the size indicated are listed.

Consider the following example

Finding files that exceed a given size limit

The following example shows how to find files with more than 400 blocks in the current working directory.

```
# find . -size +400 –print
./Howto/howto.doc
./Howto/howto.doc.backup
./Howto/howtotest.doc
./.record
./Config/configMailappx.doc
```

```
./Config/configMailconcepts.doc
```

Checking the Size of Directories

Subdirectory and files can be displayed by using the `du` command and its options.The amount of disk space taken up by user accounts on local file system can be found. Sizes are displayed in 512-byte blocks.

Consider the following example

```
# du /var/log /var/cron
4       /var/log
3250    /var/cron
```

Consider the following example displays the sizes of two directories, all of the subdirectories and files they contain, and the total number of blocks contained in each directory.

```
# du -a /var/log /var/cron
0       /var/log/authlog
0       /var/log/syslog
2       /var/log/sysidconfig.log
4       /var/log
3248    /var/cron/log
3250    /var/cron
```

Consider the following example displays the total sizes of two directories.

```
# du -s /var/log /var/cron
4      /var/log
3250   /var/cron
```

Finding and Removing Old and Inactive Files

Part of the job of cleaning up heavily loaded file systems involves locating and removing files that have not been used recently. Unused files can be located using the *`ls`* or *`find`* commands. Other ways to conserve disk space include emptying temporary directories such as the ones located in *`/var/tmp`* `or` *`/var/spool`*, and deleting **`core`** and crash dump files.

Listing the Newest Files

Files can be used by displaying the most recently created or changed files first, by using the *`ls -t`* command.

Here

```
# ls -t [directory]
```

-t	– sorts listings by latest time stamp first and
Directory	– refers to the directory that is to be searched.

Consider the following example to listing the newest files

The following example shows how to use *`ls -t`* to locate the most recent files within the *`/var/adm`* directory. *`sulog`*, *`messages`*, *, and* *`lastlog`* were created or edited most recently. This is verified using output from *`ls -l`*, which shows that these three files were created or edited in March, while the other files in *`/var/spool`* were created or edited earlier.

```
# ls –t /var/adm

sulog      wtmpx        wtmp        messages.1  vold.log   spellhist
messages   utmp         sa          messages.2  log        aculog
utmpx      lastlog      messages.0  messages.3  acct       passwd
# ls -l /var/adm
total 686
drwxr-xr-x     5 adm       adm         512 Feb 13 16:20 acct
-rw———         1 uucp      bin           0 Feb 13 16:04 aculog
-r—r—r—        1 root      other      8456 Mar 27 10:34 lastlog
drwxr-xr-x     2 adm       adm         512 Feb 13 16:36 log
-drwxr-xr-x    2 adm       adm         512 Feb 13 16:03 passwd
drwxr-xr-x     2 adm       sys         512 Mar 20 06:59 sa
```

Finding and Removing old or inactive Files

In order to find inactive files, the user should first become the super user

Files that have not been accessed for a specified number of days can be found and listed in a file as follows.

`# find` ***`directory`*** ` -type f` **`[`**`-atime +` ***`nnn`*** **`][`**`-mtime +` ***`nnn`*** **`]`**`-print >` ***`filename`***

Here

Directory	–	Refers to the directory that is to be checked .Directories below this also will be checked.
-atime +*nnn*	–	Finds files that have not been accessed within the specified number of days .
-mtime +***nnn***	–	Finds files that have not been modified within the specified number of days you specify.
Filename	–	refers to the file containing the list of inactive files.

The inactive files can now be deleted as follows

# rm 'cat *filename*'

Here

filename – refers to the file created by this command which contains the list of inactive files.

Example — Finding and Removing Old or Inactive Files

The following example locates regular files in **`/var/adm`** and its directories that have not been accessed in the last 60 days and saves the list of inactive files in **`/var/tmp/deadfiles`**. These files are then removed with the **`rm`** command.

```
# find /var/adm -type f -atime +60 –print > /var/tmp/deadfiles &
# more /var/tmp/deadfiles
/var/adm/wtmp
/var/adm/wtmpx
/var/adm/sulog
# rm 'cat /var/tmp/deadfiles'
```

Clearing out Temporary Directories

To enable the clearing of temporary directories the user should become the superuser and then the directory should be changed as shown below.

```
# cd /var/tmp
```

CAUTION

It should be ensured that the user is in the right directory before completing the following step.The next step deletes all files in the current directory.

The files and subdirectories in the current directory can be deleted as follows.

```
# rm -r *
```

The user can change to other directories containing temporary or obsolete subdirectories and files and delete them by repeating Step 3 above.

Consider the following example

The following example shows how to clear out the */var/tmp* directory, and verifies that all files and subdirectories were removed.

```
# cd /var/tmp
# ls
deadfiles          wxconAAAa0003r:0.0  wxconAAAa000NA:0.0
test_dir           wxconAAAa0003u:0.0  wxconAAAa000cc:0.0
wxconAAAa000zs:0.0
# rm -r *
# ls
```

Finding and deleting `core` files

In order to find a delete core files, the user should become a super user and then change the directory which is to be searched.Any core files in this directory and its subdirectoriescan be found and removed as follows.

Consider the following example shows how to find and remove **`core`** files from the user account belonging to **`jones`** using the **`find`** command.

```
# cd /home/jones
# find . -name core -exec rm {} \;
```

Deleting crash dump files

Crash dump files can be very large, so if the user has enabled the system to store these files, they should not be retained for longer than necessary.

The user should become the superuser to delete the crash dump files.Thereafter ,the user should change the directory where crash dump files are as shown below.

Change to the directory where crash dump files are stored.

```
# cd /var/crash/system
```

System – refers to the system that created the crash dump files.

CAUTION

It should be ensured that the user is in the right directory before completing the following step. The next step deletes all files in the current directory.

The crash dump files can be removed as shown below.

```
# rm *
```

It can be verified if the crash dump files are removed as follows

```
# ls
```

Consider the following example

The following example shows how to remove crash dump files from the system *lambent*, and how to verify that the crash dump files were removed.

```
# cd /var/crash/lambent
# rm *
# ls
```

File System Check(fsck)

File system check (***fsck***),uses the known parameters and redundant information to audit and check the file systems for inconsistencies.

The fsck Utility

The `fsck` utility takes its understanding of the internals of the various UNIX file systems and attempts to verify that all the links and blocks are correctly tied together. It runs in five passes, each of which checks a different part of the linkage *fsck* walks through the file system, starting with the superblock. It then deals with the allocated disk blocks, pathnames, directory connectivity, link reference counts, and the free list of blocks and inodes.

The ***fsck*** command checks for the following inconsistencies:

- Blocks or fragments allocated to multiple files
- i-nodes containing block or fragment numbers that overlap
- i-nodes containing block or fragment numbers out of range
- Discrepancies between the number of directory references to a file and the link count of the file
- Illegally allocated blocks or fragments
- i-nodes containing block or fragment numbers that are marked free in the disk map
- i-nodes containing corrupt block or fragment numbers
- A fragment that is not the last disk address in an i-node. This check does not apply to compressed file systems
- Files larger than 32KB containing a fragment. This check does not apply to compressed file systems
- Size checks
- Incorrect number of blocks

- Directory size not a multiple of 512 bytes

CAUTION

These checks do not apply to compressed file systems.

- Directory checks
- Directory entry containing an i-node number marked free in the i-node map
- i-node number out of range
- Dot (.) link missing or not pointing to itself
- Dot dot (..) link missing or not pointing to the parent directory
- Files that are not referenced or directories that are not reachable
- Inconsistent disk map
- Inconsistent i-node map

fsck command does not make corrections to a mounted file system

fsck command can be run on a mounted file system for reasons other than repairs. However, inaccurate error messages may be returned when the file system is mounted

Where is fsck?

When `fsck is run`, an executable in either the `/usr/sbin` or /bin directory called `fsck is run` but this is not the real `fsck`. It is just a dispatcher that invokes a file system type-specific `fsck` utility.

When Should an fsck run?

Normally, the user need not run `fsck`. The system runs it automatically when the user tries to mount a file system at boot time that is corrupted. However, problems can creep up. Software and hardware glitches do occur from time to time. It is advicable to run `fsck` just after performing the monthly backups.

CAUTION

It is better to run `fsck` after the backups rather than before. If **fsck** finds major problems, it could leave the file system in a worser shape than it was prior to running.

How to use *fsck*?

To use *fsck* the system should be brought down to single user mode.In single user mode has to be invoked **fsck,** giving it the options to force a check of all file systems, even if they are already stable.

With out any arguments,the fsck programe would check only those entries in the */etc/fstab* that have an entry *fspassno* (Linux),*/etc/vfstab (sun)* file that have an entry *for the devtofsck* field and have a nonzero numeric value entry in the *fsckpass* field.

Syntax

```
  fsck [options] [ device or mount point ]
```

```
Options
```

A	Runs *fsck* for all the file system,
-N	Prints out the help message.
-V	(Solaris) Prints the help message.
-t	Indicates the file system for which fsck is to be run.

It checks and fixes the file system in a noninteractive mode and exists immediately if there is a problem that needs user's intervention.

Consider the following example

```
# fsck  /opt
```

Running fsck on a mounted filesystem may cause

SEVERE filesystem damage.Do you really want to continue (y/n)? yes/dev/hda9 was not cleanly unmounted, check forced.

Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information

Block bitmap differences: -98858 -98859 -98860 –98861

Fix<y>? yes

Free blocks count wrong for group #12 (7635, counted=7639).

Fix<y>? yes

Free blocks count wrong (236509, counted=236513).

Fix<y>? yes

/dev/hda9: ***** FILE SYSTEM WAS MODIFIED *****

/dev/hda9: 8422/78312 files (0.8% non-contiguous), 76723/313236 blocks

Checking ufs File Systems For `ufs` file systems, `fsck` is a five-phase process. `fsck` can automatically correct most of these errors and does so if invoked at boot time to automatically check a dirty file system.

10.7 Short Summary

- *mkfs* is the command which is used to create a new file system
- *mount* is the command which is used to mount the file system
- *umount* is the command which is used to unmount the file system
- */etc/mttab* contains information about all the mounted file systems
- */etc/fstab* contains the list of all the partitions that need to be mounted at boot time and the directory where they need to be mounted

10.8 Brain Storm

1. Explain the term file system.
2. Explain the terms: boot block and super block.
3. What are raw devices?
4. What are block devices?
5. Explain the fields in */etc/fstab*.

Lecture 11

Network File System

Objectives

In this lecture you will learn the following

- Understanding the concept of NFS
- Able to Start and Stop the nfs Deamons
- Mounting the File System
- Able to Configure nfs Servers and Clients

Coverage Plan

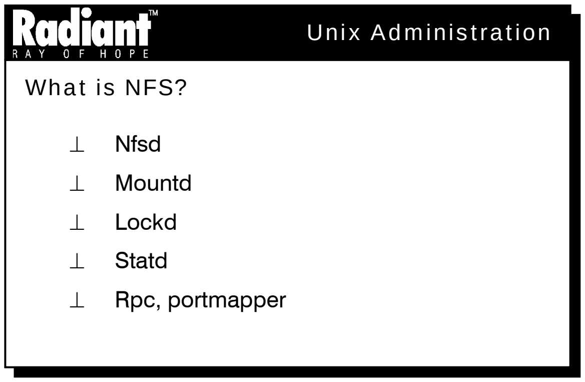
Lecture 11

11.1 Snap Shot

The NFS service enables computers of different architectures running different operating systems to share file systems across a network.The NFS environment can be implemented on different operating systems because it defines an abstract model of a file system, rather than an architectural specification. Each operating system applies the NFS model to its file system semantics. This means that file system operations like reading and writing function as though they are accessing a local file. The benefits of the NFS service are

- Allows multiple computers to use the same files, so everyone on the network can access the same data
- Reduces storage costs by having computers share applications instead of needing local disk space for each user application
- Provides data consistency and reliability because all users can read the same set of files

11.2 What is NFS?



Unix Administration

What is NFS?

⊥ Nfsd

⊥ Mountd

⊥ Lockd

⊥ Statd

⊥ Rpc, portmapper

The Network File System (NFS) was developed to allow machines to mount a disk partition on a remote machine as if it were on a local hard drive. This allows for fast sharing of files across a network. The version of *nfs* is 2.

It also gives the potential for unauthorised access of the hard drive over the network (and thereby possibly read the users email and delete files as well as break into the system) NFS is a ***stateless protocol,***which means that the request that is made between the client and server is complete in itself and doesn't require keys of prior transactions.

NFS services are activated in runlevel 3, *nfs* is the service which allows the user to share or access the remote files, directories and file systems , *nfs* service controls four daemons,they are

nfsd

This is the Network File System (NFS) daemon. It runs on the file server, and is responsible for handling client requests.

mountd

This daemon runs on the NFS server and is responsible for responding to client NFS mount requests.

lockd

This is run by both the client and the server, this daemon handles file locks.

statd

This is run by both the client and the server, statd maintains the status of currently enforced file locks.

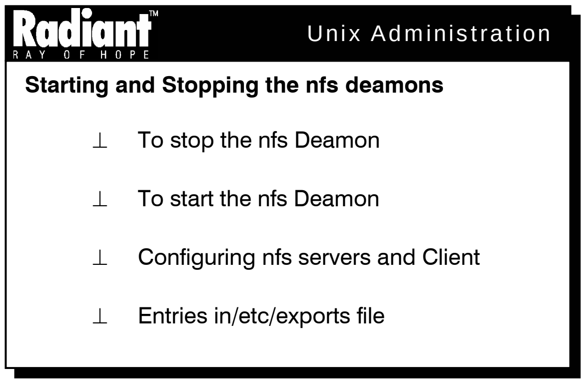
Rpc.portmapper

These daemons do not directly provide *nfs* services. They map calls made from other machines to the correct *nfs* daemons. Listing of daemons that are invoked by NFS on both the client and the server hosts are given below. It should be noted that both **lockd** and **statd** daemons are invoked on both the hosts.

Client Daemons	Server Daemons
	nfsd
lockd	lockd
statd	statd
	mountd

Table 9.1

11.3 Starting and Stopping the *nfs* Daemons



Radiant™ RAY OF HOPE Unix Administration

Starting and Stopping the nfs deamons

⊥ To stop the nfs Deamon

⊥ To start the nfs Deamon

⊥ Configuring nfs servers and Client

⊥ Entries in/etc/exports file

There would be instances where it might be required to stop *the nfs* and restart it later.This can be done by using the startup scripts that are executed at boot time and shutdown. *nfs* scripts are in */etc/rc.d/nfs.*

Note : In Solaris */etc/init.d/nfs.server* . is the service that provides the network file system services, for accessing and sharing the remote file system.

To stop the *nfs* Daemon

```
# /etc/rc.d/init.d/nfs stop
```

To start the *nfs* Daemon

```
# /etc/rc.d/init.d/nfs start
```

Configuring *nfs* Servers and Clients

There are two key files to nfs .They are */etc/exportfs* and */etc/fstab*. */etc/exportfs* is used to share files, directories and file system to nfs clients.This file is configured on the server. It specifies the files that are to be shared with a specific client and the client access rights.*/etc/fstab* is configured on the client side and specifies the servers to be contacted for certain directories as well as where to place them in their machines.

Entries in */etc/exports* file

This file contains a list of entries; each entry indicates a volume that is shared and how it is shared. An entry in /etc/exports will typically look like this:

```
directory machine1(option11,option12) machine2(option21,option22)
```

where

directory

Specifies the directory to be shared. If an entire volume is shared then all directories under it within the same file system will be shared as well. machine1 and machine2.Specifies the client machines that will have access to the directory. The machines may be listed by their IP addresses or their DNS addresses (e.g., *machine.company.com* or *192.168.0.8*). Using IP addresses is more reliable and more secure.

optionxx

The option listing for each machine will describe what kind of access that machine will have. Important options are:

ro: Indicates that the directory is shared read only; the client machine will not be able to write to it. This is the default.

rw: Indicates that the client machine will have read and write access to the directory.

Consider the following example

```
/home/abchost1(rw) host2(ro) host3(ro)
```

Sharing the file system

exportfs is the command which is used to export the files, directories and filesystem to the clients.This command makes local directories available for Network File System (NFS) clients to mount. This command is normally invoked during system startup by the **run level 3** and uses information in the */etc/exports* file to export one or more directories, which must be specified with full path names.

The **/etc/rmtab** file lists directories that are currently exported. To display this file, the *exportfs* command has to be entered without flags or arguments. To alter the file or to alter the characteristics of one of its directories, root users can edit the /etc/exports file and run the *exportfs* command.

CAUTION

Never edit the */etc/rmtab* manually.

Note : */etc/rmttab* is the file That lists directories that are currently exported.

Syntax

```
exportfs [ options ]
```

options

-a	Exports all directories listed in the **/etc/exports** file.
-v	Prints the name of each directory as it is exported or unexported.
-u	Unexports the directories that are specified. When used with the **-a** flag, unexports all directories listed in the **/etc/exports** file.
-I	Allows the exporting of directories not specified in the exports file or ignores the options in the **/etc/exports** file. Normally the **exportfs** command consults the **/etc/exports** file for the options associated with the exported directory.
-f file	Specifies an export file, other than the **/etc/exports** file, that contains a list of directories that can be exported. This file should follow the same format as the **/etc/exports** file. NOTE: This alternate file will not be used for exporting directories automatically when the system and NFS is started. The **/etc/exports** file is the only file that is supported for specifying directories to export at system start.
-o options	Specifies optional characteristics for the exported directory. More than one variable can be entered by separating them with commas.
Ro	Exports the directory with read-only permission. Otherwise, if not specified, the directory is exported with read-write permission
Rw = *Client [:Client]*	Exports the directory with read-write permission to the machines specified by the *Client* parameter and read-only to all others. The *Client* parameter can be either the host name or the network name. If a *rw* host name is not specified, the directory is exported with read-write permission to all.

/etc/exports format:

```
# vi /etc/exports
```

[directory] manchine1[option] machine2[option]

```
/home/kishore   radiant1(rw) radiant2(ro)
```

In the above example the directory called */home/kishore* is available to the hosts *radiant1* and *radiant2* .For *radiant1* the access is both read and write and *for radiant2* the access is only for reading.After the entry is made the *nfs* services have to be restarted.These services are normally started at boot time by the *run level3*.These services can also be run manually to stop and start the *nfs* system.To stop *nfs,* the user will have to log in as root and enter the following:

Consider the following example and can be used to start and stop *nfs* daemons

```
# /etc/rc.d/init.d/nfs stop
```

The *nfs* services can be started as follows:

```
# /etc/rc.d/init.d/nfs start
```

The following command can be used to check whether the *nfs* daemons are currently running:

```
# /etc/rc.d/init.d/nfs status
```

Once an entry is made in */etc/exports* file to share the directory,the *exportfs* command has to be run to check whether the directory is shared.

The following examples show how to user the *exportfs* command.

```
# exportfs
/home/kishore  radiant1
/home/rams     radiant2
```

In the above example */home/kishore* is exported to *radiant1* and /home/rams is exported to radiant2

To export entries in */etc/exports* the following can be done:

```
#  exportfs -a
```

-a exports all the files and directories from */etc/exportfs* to *nfs* clients.

To unexport all exported files and directories the following is done:

```
#  exportfs   -ua
```

-u Unexports all the files and directories from */etc/exportfs* to *nfs* clinets.

Note : *unshare* is the command which is used to unexport the file sytem to clients.

unshare the file system

```
#  unshare [ file system]
#  unshare /home/kishore
```

In the above example /home/kishore is unexported to *nfs* clients.

To export /*home/kishore* in read-only mode the following can be done:

```
# exportfs -o ro: /home/kishore
```

Here the options

-oSpecifies a comma-separated list of optional characteristics for the directory being exported

To export /*home/kishore* read and write mode to the *radiant1,radiant2* the following can be done:

```
# exportfs -o rw= radiant1:radiant2      /home/kishore
```

In the above example */home/kishore* is shared in read and write mode to the clients of *radiant1*and *radiant2.*

uy

Note : *share* is the command which is used to share the file system to the *nfs* clients.

Share the filesystem

```
# share –F [File system type] –o [options] hostname –d [Description]
options
```

-F – Indicates the filesystem that is being used for exports.Here the filesystem type is *nfs.*

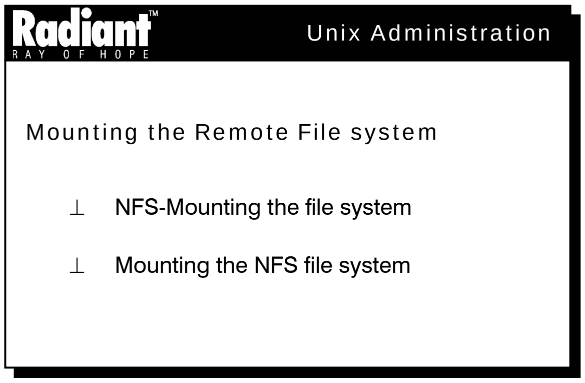
-o	–	Indicates the access rights to the directories.
rw	–	Read and write access to the directories.
ro	–	Read only access to the directories.
hostname	–	Indicates the *nfs* client.
-d	–	Indicates the description for the directories which are exported to the *nfs* clients.

Consider the following example

```
# share –F nfs –o rw:radiant1:radiant2 /home/kishoe –d "only for Admin.pupils"
```

In the above example *share* is the command which is used to export the */home/kishore* directory to the *nfs* clients called *radiant1 & radiant2.*

11.4 Mounting the Remote File System



Radiant™ RAY OF HOPE Unix Administration

Mounting the Remote File system

⊥ NFS-Mounting the file system

⊥ Mounting the NFS file system

NFS-Mounting the File System

The following steps have to be followed for each workstation from which NFS access to the Share file system is required

1. Identify or create a Unix mount point, which is an empty Unix directory on the workstation. Default is */mnt.*
2. The installed bin directory must be in the (super-user's) path.
3. Run the script mnt, which takes 2 arguments: the name of a host, and the pathname for the mount.
4. Check the mount. Type *mount* and see if the mount point shows up in the mount table.

Mounting the NFS file System

mount is the command which is used to mount the file system and display the mounted file system information.

```
# mount –t [file system type] –o [option] [hostname  of NFS server:remote dir
]  [mountpoint]
```

options

-t	Indicates the filesystem type .

There are options that can be passed to the `mount` command. The most important characteristics are specified in the `-o` option. These characteristics are:

rw	–	read/write.
ro	–	read only.
bg	–	background mount (if the mount fails, place the process into the background and keep trying until success).
fg	–	foreground mount (if the mount fails, place the process into the background and keep trying until success).
soft	–	The soft option gives an error if the server doesn't respond.Do not use this option on writable file systems.
hard	–	Continue retrying a request until the server responds.Use this option on all file systems mounted with read-write permission.

Consider the following example

```
#mount –t  nfs –o rw,fg,soft  radiant1:/home/kishore /mnt

#mount –t  nfs –o rw radiant1:/home/kishore  /mnt
```

Automating the Mount Process

UNIX uses a special file called **/etc/fstab.** This file lists all the partitions that need to be mounted at boot time and the directory where they need to be mounted. Along with that information parameters can be passed to the **mount** command.

Format of /etc/fstab:

```
/dev/device /dir/tomount  ftype parameters   fs_freq    fs_passno
```

where:

/dev/device	–	Is the device to be mounted, for instance /dev/hda4.
/dir/to/mount	–	Specifies the location at which the file system should be mounted on the directory tree.
Ftype	–	Specifies the file system type for mounting.
Parameters	–	Specifies the parameters that are passed to mount using the -o option. They follow the same comma-delineated format. An example entry would look like rw,intr,bg.
Fs_freq	–	Used by dump to determine whether a file system needs to be dumped.
Fs_passno	–	Used by the *fsck* program to determine the order of checking the disks at boot time.

The system administrator can tell the OS about any filesystem the machine may have access to in the */etc/fstab* . It also allows default parameters to be provided for each filesystem.

```
local mount

/dev/device /dir/to/mount    ftype   parameters  fs_freq fs_passno
/dev/hda1/  /home            ext2    rw          0       0
Remote mount
```

```
/dev/device /dir/to/mount    ftype  parameters fs_freq  fs_passno
lambent:/home/kishore /mnt   nfs    rw         0        0
```

The Structure of */etc/fstab*

The **first field** (*lambent:/home/kishore*) is the remote filesystem which is to be described.

The **second field** (/ *mnt*) specifies the mount point where the filesystem will be mounted.

The **third field** (*nfs*) is the type of filesystem on the device from the first field.

The **fourth field** (*rw*) is a options which mount should use when mounting the filesystem.

The **fifth field** (0) is used by dump (a backup utility) to decide if a filesystem should be backed up. If zero then dump will ignore that filesystem.

The **sixth field** (0) is used by fsck (the filesystem check utility) to determine the order in which filesystems should be checked.

Note : In Solaris **/etc/vfstab** is the file that lists all the partitions that need to be mounted at boot time and the directory where they need to be mounted.

```
#vi /etc/vfstab
<device to mount>  <device to fsck> < mount point>  <fs type> <fsck pass>
<mount at boot> <mount options>

# remote mount

lambent:/home/kishore - /mnt    nfs    -   yes    -
```

The fields are described below:

Device to mount	–	This is the name of the server sharing the resource the client wants to mount, if is the resource is an *Nfs* resource followed by a colon, then thename of the resource to be mounted. (*lambent:/home/kishore*).***lambent*** is the name of the server and ***/home/kishore*** is the name of the remote directory.
Device to fsck	–	Specifies the name of the raw device to `fsck`; for a remote mount, the parameter is not applicable, and the entry should be - .
mountpoint	–	which is the mount point on which the resource is to be mounted.(*/mnt*).
fstype	–	Specifies the file system type of the resource that is to mount.(*nfs*)
fsckpass	–	This is the pass number to use for multiple *`fsck`'s*. For a remote mount, the parameter is not applicable, and the entry should be -
mntopts	–	Specifies the options passed to the *`mount`* command;.

There are options that can be passed to the *mountoption* field(7th coloum). The most important characteristics are specified:

rw	–	read/write.
ro	–	read only.

bg	–	background mount (if the mount fails, place the process into the background and keep trying until success.)
fg	–	foreground mount (if the mount fails, place the process into the background and keep trying until success.)
soft	–	The soft option gives an error if the server doesn't respond. Don't use this option on Writable file system.
hard	–	Continue retrying a request until the server responds.Use this option on all file systems mounted with read-write permission.

End Note:

showmount queries the mount daemon on a remote host for information about the state of the NFS server on that machine. With no options *showmount* lists the set of clients who are mounting from that host.

Practice 11.1

The following example shows the list of clients on which file system are mounted

```
# showmount
```

```
hostname on radiant1:/home/kishore
```

Here **radiant1** is the client on which */home/kishore* file system is mounted.

Options

-a or —all

Lists both the client hostname and mounted directory in *host:dir* format.

-d or —directories

Lists only the directories mounted by some client.

-e or —exports

The file **/var/lib/nfs/rmtab** contains a record of all clients that mount remote file systems from the current machine. Whenever a remote mount is done, an entry is made in the rmtab file of the machine serving that file system. umount removes the entry of a remotely mounted file system. **umount** *-a* broadcasts to all servers that they should remove all entries from **rmtab. rmtab** contains a table of filesystems that are remotely mounted by **NFS** clients. This file is maintained by **mountd** the mount daemon. The data in this file should be obtained only from *mountd* the mount daemon The file contains a line of information for each remotely mounted filesystem.

Consider the following example

```
# vi  /var/lib/nis/rmtab
```

```
radiant1    /home/kishore
radiant2    /home/rams
```

In the above file */home/kishore* is being accessed by *radiant1,/home/rams* is being accessed by *radiant2.*

11.5 Short Summary

- *nfs* is the service which allows the user to share and access the remote file systems
- *exportfs* is the command which is used to export a directory
- *showmount –e* is the command is used to list all the currently exported directories
- */etc/exportfs* is the file that contains a list of directories that are expored
- */etc/fstab* file allows to mount a file system automatically when the system is booted

11.6 Review Questions

1. What are the daemons that are controlled by *the nfs* service?
2. What is the name of the file that contains all the currently exported directories?
3. What is the command that is used to unexport all the filesystems?
4. Which file contains the record of all clients that have mounted remote file system?
4. Define the fields in */etc/fstab.*

Lecture 12

Virtual File System

Objectives

In this lecture you will learn the following

- Knowing the virtual file system and its implementation methods
- Understanding the type of Virtual File system
- About the Process file System

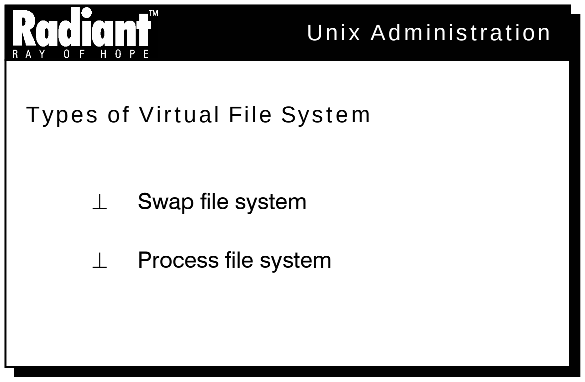
Coverage Plan

Lecture 12

12.1 Snap Shot

Virtual file system is an extension of physical memory.When physical memory is not enough to execute an application, some memory from the free space of the hard disk has to be allocated temporarily. The temporary space is referred to as Virtual File System(VFS). This will be beneficial when the system is in an active state.

12.2 Types of Virtual File System



Radiant™ RAY OF HOPE Unix Administration

Types of Virtual File System

⊥ Swap file system

⊥ Process file system

The Virtual File sytem supported in the Unix Operating System environment are described below:

Swap file system?

This section deals with the following:

- Virtual Memory (or) Swap Space
- Adding more swap space
- Creating a swap file

Virtual Memory (or) Swap Space

Unix supports virtual memory, that is,using a part of the hard disk as an extension of RAM so that the effective size of usable memory grows correspondingly. The kernel will write the contents of a not-right-now used part of memory to the hard disk so that the memory can be used for another purpose. When the original contents are needed again, they are read back into memory (not necessarily to the same place). This is all completely transparent to the user; programs running under Unix only see the larger amount of memory available and do not notice that parts of them reside on the hard disk from time to time. Of course, reading and writing the hard disk is slower (on the order of a thousand times slower) than using real memory, so the programs do not run fast. The part of the hard disk that is used as **virtual memory** (as this is called) is called the **swap space**.

Adding More Swap Space

As system configuration changes and new software packages are installed more *swap* space might have to be added. .The preferred way to add swap is by using the *touch* and *mkswap,swap* commands to designate a port on an existing file system.

Creating a swap file

The following general steps are involved in creating a swap file system.

- Create a swap file by using *mkswap* command
- Activitate the swap with the *swapon* command

***mkswap* Command:**

The *mkswap* command is used to add a swap file.

Syntax

```
mkswap [device ] [ block ]
```

The options for the *mkswap* command:

-c:	Checks the device for bad blocks before a swap fileis created.
-f:	Forcibly creates a swap file.

Consider the following example to create a swap file:

```
# mkswap /dev/hda9 2000
```

mkfile is the command which is used to create a swap file, *swap* is the commands which is used to activate and check the swap staus in Solaris.

Syntax:

```
# mkfile nnn[k/b/m] filename
```

Consider the following example to create a swap file:

```
# mkfile 24m /opt/abc
```

The swap file of the size nnn(in Kbytes,bytes, or Mbytes) and the specified name is created

Swapon Command:
swapon is the command which is used to activate the swap on the device.

Syntax:

```
# swapon [options] [ device ]
```

Options:

–h	Prints the help information.
-s	Prints the swap usage information.

Consider the following example

```
# swapon  /dev/hda9
```

swap is the command which is used to activate and check the status of *a swap* file system in Solaris.

Consider the following example
Activate the swap file:

Syntax:

```
#  swap -a  [file name]
```

The options of *swap* command:

–a:	It is used to add swap file.
-l:	It is used to verify the swap file is added.
-s:	It is used to list the status of swap file.
-d:	It is used to remove the swap file from use.

Consider the following example

# swap -a /opt/kishore

Deleting the swap space

The swap space can be deleted by using the swapoff command.

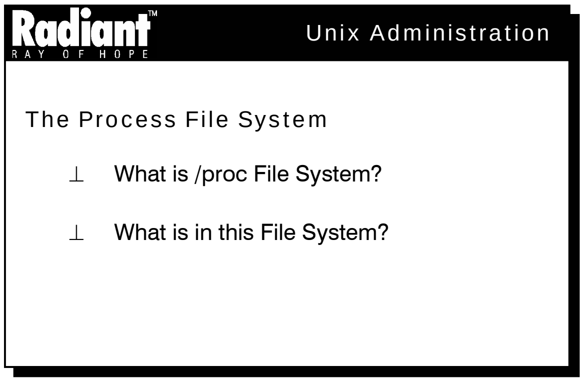
Syntax:

```
swapoff  [device]
```

Consider the following example

```
# swapoff /dev/hda9
```

12.3 The Process File System



Radiant™ RAY OF HOPE

Unix Administration

The Process File System

⊥ What is /proc File System?

⊥ What is in this File System?

This file system contains a list of active processes, named according to the process number,in the */proc* directory.

/proc file System?

The */proc* file system is not a real file system; it is a virtual file system without the physical presence that a disk , in which the information about all the active process in the system are stored.

This File System contains everything that happens in Unix. Every single program that is running, the entire contents of memory, the internal workings of the kernel—all the processes currently running on the system are contained in the */proc* file system. proc is an abbreviation for process.

Some common Files in */proc*

- *pid/* contains information about process number *pid*. The kernel maintains a directory containing process information for each process
- *pid/cmdline* contains the command that was used to start the process (using null characters to separate arguments)
- *pid/environ* contains a list of the environment variables that are available to the process
- *pid/exe* contains a link to the program that is running in the process
- *pid/fd/* is a directory containing a link to each of the files that the process has opened
- *pid/mem* contains the memory contents of the process
- *pid/stat* contains the process status information
- *pid/statm* contains the process memory usage information

The */proc* file system is not a file system in the standard sense. The large files found in */proc* are the address spaces of running processes, just the PID of the running processes. The owner and group owner of each file are the real-UID and primary group of the process's owner.

A sample listing from /proc is as follows

```
ls -l /proc
total 43384
-rw———   1 root     root           0 Apr  2 20:07 00000
-rw———   1 root     root      393216 Apr  2 20:07 00001
-rw———   1 root     root           0 Apr  2 20:07 00002
-rw———   1 root     root           0 Apr  2 20:07 00003
-rw———   1 root     root     1695744 Apr  2 20:07 00081
-rw———   1 root     root     1597440 Apr  2 20:07 00083
-rw———   1 root     root     1777664 Apr  2 20:08 00096
```

12.4 Short Summary

- */proc* is the file system that contains a list of active processes, named according to process numbers
- */swap* is a file system that is a part of the memory on the hard disk, which is used to queue processes
- *mkswap* is the command which is used to create a swap file
- *swap* [options] command is used to check the swap status

12.5 Brain Storm

1. How can the status of a *swap* file system be found?
2. Which command is used to create a *swap* file system?
3. Explain */proc* file system?

Lecture 13

Security

Objectives

In this lecture you will learn the following

- Knowing the various security measures incorporated by the UNIX OS.
- About the types of UNIX Security

Coverage Plan

Lecture 13

13.1 Snap Shot

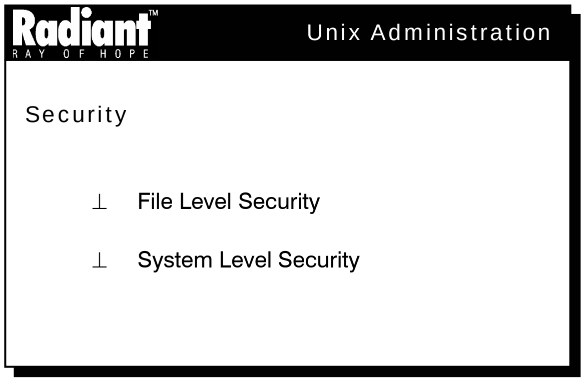
Defining "computer security" is not trivial. The difficulty lies in developing a definition that is broad enough to be valid regardless of the system being described, yet specific enough to describe what security really is. In a generic sense, security is "freedom from risk or danger." In the context of computer science, security is the prevention of, or protection against,

- access to information by unauthorized recipients

- _intentional but unauthorized destruction or alteration of that information.

This can be re-stated: "Security is the ability of a system to protect information and system resources with respect to confidentiality and integrity." It should be noted that the scope of this second definition includes system resources, which include CPUs, disks, and programs, in addition to information.

13.2 Security



Radiant™ RAY OF HOPE — Unix Administration

Security

⊥ File Level Security

⊥ System Level Security

Being a multi-user operating system , UNIX has incorporated various security measures at different levels to ensure complete safety from any unauthorized access to the system. This session deals with two types of UNIX security, 1) File level security and 2) System level security.

13.3 File Level Security

- Specify Owner, Group and Others
- Types of Permissions
- Advanced file permissions

In the UNIX operating system the file access by users is controlled with respect to the owner, group and others .

Specify Owner Group and Others

The term user refers to the owner of the file (who created the file) and the term group is the group (or groups, in the case of secondary groups) to which the owner belongs. Other belongs categories include all users except the owner and the members in the owner group who have access permission in the system

Types of permissions

Normal permissions

Each file or directory has permission that apply to one of three types of user:

u - Tthe user who owns the file
g - Tthe group to which the file belongs
o - Eeverybody else

For each of these users, a file or directory has three type of permissions:

r - Read the file or directory
w - Write to (or remove)the file or directory
x - Execute the file or list the directory

Each of these permissions can be listed as 9 characters, listed by user, group and others as shown below:

```
 user     group     others
 r w x    r w x     r w x
 |___|    |___|     |___|
   |        |         |
   |        |         — — Access granted to non-owner, non-group
   |        |— — — — — — — — Access granted to group members
   |— — — — — — — — — — — — — — — — Access granted to file’s owner
```

Octal & Symbolic Notations

Octal	Symbolic	Description
0	—-	No permission
1	—x	Execute permission only
2	-w-	Write permission only
3	-wx	Write & execute only
4	r—	Read permission only
5	r-x	Read & execute permissions
6	rw-	Read & write permissions
7	rwx	All permissions

Table 13.1

Consider the following example to identify the mode permission for files or directories

The “ ls “ command can be used to show the current permissions of a file or directory:

```
ls -l filename
        -or-
ls -ld directories
```

This displays the filename (or directory) in a one line summary as shown below:

```
-rwxr-xr-x  1 radiant     683 May  1 11:37 text.txt
                   -or-
drwxr-xr-x  2 radiant     512 May  1 11:37 bbl
```

Default permissions for files and directories

When a user creates a file , the system assigned permissions are -rw-rw-rw- (ie 666 in octal notation) and for a directory it is drwxr-xr-x (ie 777 in octal notation). But there is an environment variable UMASK that comes into the picture. It stands for users mask that will have a default value 022 and it is subtracted from the default permissions. For a file 666 – 022 yields 644. This will make the file –rw-r—r— . For a directory 777 – 022 yields 755. This will make the directory drwxr-xr-x.

Normally the umask value assigned to users by default is 022 which grants permissions to newly created files or directory. By changing the value of umask the permissions granted to a file or directory can be controlled on creation.

After changing the value of the umask from the command line or in the .profile file the user has to logout from the current session and again login into the system in order to make the umask settings effective

Changing permissions using chmod command

Once a file is created the umask value no longer has any affect on it. In order to change the permissions of a file the chmod command must be used. Only the owner (or root) of a file can change its permissions.

chmod has two modes, " symbolic " and " absolute " .

In symbolic mode, permissions can be granted or revoked by using the symbols " +, -, = " .

In absolute mode, permissions are set by the desired octal value for the file permissions.

Access to a file may also be affected by using the CHMOD command. Only the owner (or root) of a file can change its group membership.

The file owner can change the group of a file only if the owner is a member of the group.

Syntax

```
chmod  [ -fR ] <absolute-mode> file ...
chmod  [ -fR ] <symbolic-mode-list> file ...
```

chmod changes or assigns the mode of a file. The mode of a file specifies its permission and other attributes. The mode may be absolute or symbolic.

Absolute mode: An absolute mode is specified using octal numbers

chmod nnnn file ...

where

n is a number from 0 to 7.

For directories, files are created with BSD semantics for propagation of the group ID. With this option, files and subdirectories created in the directory inherit the group ID of the directory, rather than that of the current process. It may be cleared only by using symbolic mode.

Options

The following options are supported:

-f – Forces the change, chmod will not complain if it fails to change the mode of a file

-R – Recursively descends through directory arguments, setting the mode for each file as described above

When symbolic links are encountered, the mode of the target file is changed, but no recursion takes place

Consider the following examples

1. Deny execute permission to everyone:
   # chmod a-x filename

2. Allow only read permission to everyone:
   # chmod 444 filename

3. Make a file readable and writable by the group and others:
   # chmod go+rw filename
   # chmod 066 filename

Changing ownership of a file

chown command is used to change the ownership of a file.

Using ***chown*** the group ownership of a file can also be changed as follows

Syntax

```
chown  [ -fhR ] owner  [ : group  ] file ...
```

Options

-f – When this option is used errors are not reported

-h – If the file is a symbolic link, changes the owner of the symbolic link. Without this option, the owner of the file referenced by the symbolic link is changed

-R – Recursively descends through the directory, and any subdirectories, setting the ownership ID as it proceeds. When a symbolic link is encountered, the owner of the target file is changed (unless the -h option is specified), but no recursion takes place.

The following operands are supported:

owner[:*group*]	–	Specifies a user and optional group ID to be assigned to a file . The *owner* portion of this operand must be a user name from the user database or a numeric user ID
file	–	*Specifies a path name of a file whose user ID is to be modified*

To change ownership of all files in the hierarchy, including symbolic links, but not the targets of the links the following can be used:

```
$ chown -R -h   owner  [:group]  file...
```

Note : Only the owner of a file or the super-user may change the owner of that file. But by default the owner of the file is prevented from changing the owner ID of the file. Only the super-user can arbitrarily change owner IDs to make the user to change the ownership of a file, include the following line in /etc/system in Solaris:

```
set rstchown = 1
```

To disable this option, the following line in /etc/system:

```
set rstchown = 0
```

Changing Group Ownership of a File

Using the chgrp command the group ownership of a file can be changed.Only the owner of a file or the super-user may change the owner of a file. But by default the owner of the file is prevented from changing the group ID of the file.

Syntax

chgrp [-fhR] group file

Options

-f	– Forces the change. Errors are not reported.
-h	– If the file is a symbolic link, changes the group of the symbolic link. Without this option, the group of the file referenced by the symbolic link is changed
-R	– Recursively descends through the directory , and any subdirectories, setting the specified group ID as it proceeds. When a symbolic link is encountered, the group of the target file is changed (unless the -h option is specified), but no recursion takes place

The following operands are supported:

group	– Indicates the group name from the group database or a numeric group ID. Either of these specifies a group ID to be given to each file named by one of the file operands. If a numeric group operand exists in the group data base as a group name, the group ID number associated with that group name is used as the group ID
file	– Indicates the path name of a file whose group ID is to be modified

Umask Filter

The umask utility sets the file mode creation mask of the current shell execution environment to the value specified by the mask operand. This mask affects the initial value of the file permission bits of subsequently created files.If we are giving the command umask alone ($umask) is given , the current value of the mask is printed.

umask is recognized and executed by the shell.

umask can be included in the user's .profile file and invoked at login to automatically set the user's permissions on files or directories created.

umask 022 removes write permission for group and other (files normally created with mode 777 become mode 755; files created with mode 666 become mode 644).

The following options are supported:

-S Produce symbolic output.

Consider the following example

```
# umask 027 ( for the current session)
 or
   $ vi .profile
 umask 027
:wq!
```

In the first case above the umask value is changed temporarily. If the user logs out and then again logs in the umask value goes back to the default value. In the second case the new umask value is entered into the .profile file of the user which permanently changes the umask value and hence changes the file permissions accordingly.

After changing the value of UMASK in .profile file the user have to logout from the current session and again he have to login in order to initialize umask's new value.

Advanced File Permissions

Setuid & setgid

The owner and the super user can also set setuid and setgid permissions on a file and setgid permission on directory. These special permissions enable to control the modification of files and shared directories.

If an executable file has setuid permission, any user who has execution permission on the file will be treated as the owner of the file.

If a file has setgid permission, any user who has execution permission on this file will be treated as the user who belongs to the group of the executable file.

Executable programs with setuid or setgid permission get their UID's or GID's from the owner and group of the program file, instead of inheriting their UID's and GID's from the process (usually a shell) that started them. This is used when a program must access files that are normally only accessible to the owner or group owner of the program.

For example

```
#ls -l /usr/bin/passwd /etc/shadow

-r————         1 root     root    957   May 23 18:26 /etc/shadow
-r-s—x—x       1 root     root    12244 Feb  7  2000 /usr/bin/passwd
#
```

From the above example it is clear that in the shadow file, only root has read permission and all others do not have any permissions. But any user can change their password using /usr/bin/passwd command, that will make changes in the /etc/shadow file. This happens because /usr/bin/passwd is a setuid program

Directories that have setgid permission will propagate their GID to files created below them, i.e., new files and directories will belong to the same group as the parent directory. It is very useful in the case of a shared project directory

Enabling SETUID and SETGID Permissions

Setuid and setgid permissions can be set with the chmod command by using symbolic or numeric notation for files. Numeric notation requires four octal numbers when specifying the setuid or setgid and uses the left-most number to refer to these special permissions.

OCTAL	NOTATIONS
4	setuid
2	setgid
1	save text attribute

Table 13.2

For executable files

# chmod 4755 setuid_executable file

# chmod 2755 setgid_executable file

For shared directories

#chmod g+s shared_directory

The setgid bit on a directory must be set or changed using symbolic notation.

The file or directory that is set to a setuid/setgid program can be checked using the ls command as shown below.

$ls –l setuid_executable file

```
-rwsr-xr-x       1           root      other           567   Mar     18   19.25
setuid_executable file
-rwxr-sr-x       1           root      other           678   Mar     18   19.27
setgid_executable file
```

Note: In some cases "S" (capital S) instead of "s" can be found in the output of ls –l command for a setuid or setgid file which shows an error condition that the setuid or setgid bit is set but the execute bit is off.

Save Text Attribute in LINUX

If a directory is publicly writable and has the sticky bit set, files within that directory can be removed or renamed only by the owner of the file or the directory (owner of the public directory) or by the root or the file is writable by the user.

This prevents the users from deleting files of others users from public directories

The Save Text Attribute permission can be set as follows.

```
#chmod 1777 projectdirectory
OR
#chmod  a-rwxt projectdirectory
#ls –ld projectdirectory
```

```
drwsrwsrwt       2     root     other    512      Sep   18  16:27  projectdirectory
```

In some cases a "T" in the instead of "t" can be found in the output, this indicates an error condition that, the save text attribute bit is on but the execute permission bit is off.

In Solaris and SCO OS the save text attribute is known as a sticky bit.

Access Control Lists (ACL)* (In Solaris O/S)

In traditional Unix file protection provides read, write and execute permissions for the three user classes; file's owner, file's group and other. An ACL provides better file security by enabling file permissions to be defined for the owner, group, other, specific users, specific group, and default permissions.

For each file that is specified, setfacl will either replace its entire ACL, including the default ACL on a directory, or it will add, modify, or delete one or more ACL entries, including default entries on directories.

The setfacl command is used to set or modify ACLs. It supports the following options

-d	–	Deletes the specified ACL entries
-m	–	Adds/changes the specified ACL entries
-s	–	Replaces the whole ACL with specified entries

The following example replaces the entire ACL for the file, which gives the user shea read access, the file owner all access, the file group owner read access only, the ACL mask read/write access, and others no access.

```
#setfacl-s user:shea:rwx,user::rwx,group::rw-,mask:r—,other:—abc
OR
# setfacl -s u:shea:7,user::7,group::6,mask:4,other:0 abc
```

Note that after this command, the file permission bits are rwxr— —. Even though the file group owner was set with read/write permissions, the ACL mask entry limits it to have only read permissions. The mask entry also specifies the maximum permissions available to all additional user and group ACL entries. Once again, even though the user shea was set with all access, the mask limits it to have only read permissions. The ACL mask entry is a quick way to limit or open access to all the user and group entries in an ACL. For example, by changing the mask entry to read/write, both the file group owner and user shea would be given read/write access.

The following example adds one ACL entry to the file abc, which gives user shea read permission only.

```
# setfacl -m user:shea:r—abc
             OR
# setfacl -m u:shea:4 abc
```

The ACLs for a file or a directory can be displayed using getfacl command.

```
# getfacl –d abc
```

The asroot command can be used in SCO, as there is no concept of ACLs in SCO. After setting the appropriate "setgid" using chmod, the files in /tcb/files/rootcmds/... have to be edited to enable the users to run as root. Refer the asroot man pages for more details.

13.4 System Level Security

- The superuser account
- Unix System Files

The Superuser Account

Keeping the system's information secure is one of the primary tasks of the system administrator's primary tasks. System security involves protecting data against a disaster or system failure. In addition, it is the duty of a system administrator to protect the system from the threat of an unauthorized intruder and protect the data on the system from unauthorized users

This section discusses how to safeguard the system against unauthorized access, such as how to prevent an intruder from logging into the system, how to maintain the passwd files and how to prevent unauthorized super user access to sensitive system files and programs

Security barriers can be setup on a system. The first security is the logging program. To cross the barrier and gain access to a system, a user must be supplied with a user name and the corresponding passwd known by the system.

The second security barrier is in ensuring that only the superuser replaces the system programs and files

The user with UID of 0 is granted read and writes access to all files stored in the system's local disk and can send signals to all processes under the control of the system's CPU. There are no restrictions for the super user account

The root account is created automatically when the Unix/Linux operating systems is installed.

The Superuser can

- Shutdown the system
- Back up and restore the file systems
- Mount and unmount file systems
- Add and delete users
- Maintain password aging of the users

It is recommended to change the root password on a regular basis

The root account should be only used only while performing system administration tasks.For everyday use ,it is not advisable to use the root account. This helps to protect the system from unauthorized access and also , critical mistakes are less likely to occur if routine work is done as an alternate user.

Unix System Files (/etc/passwd,/etc/shadow,/etc/group)

`/etc/passwd`

The password file is arguably the most critical system file in Linux (and most other UNIX's). It contains the mappings of username, user ID and the primary group ID that a person belongs to. It may also contain the actual password However, it is more likely (and much more secure) to use shadow passwords to keep the passwords in /etc/shadow. This file MUST be globally readable, otherwise commands even as simple as ls will fail to work properly. The GECOS field can contain data such as the real name, phone number etc for the user. The home directory is the default directory the users get placed in if they log in interactively, and the login shell must be an interactive shell (such as bash, or a menu program) and listed in /etc/shells for the user to log in. The format is:

```
username:encrypted_password:UID:GID:GECOS_field(comment):home_directory:login_
shell
```

Passwords are stored utilizing a one way hash (the default hash used is crypt, newer distributions support MD5 which is significantly stronger). Passwords cannot be recovered from the encrypted result. However, a password can be found by using brute force to hash strings of text and compare them. Once a match is found the user knows the password. This in itself is generally not a problem, the problem occurs when users choose passwords that can be easily guessed. The recent survey results have shown that 25% of passwords can be broken in under an hour, and what is even worse is that 4% of users choose their own names as the password. Blank fields in the password field are left empty, a "::"symbol can be seen; this is something that is critical for the first four fields (name, password, uid and gid).

`/etc/shadow`

The shadow file holds the username and password pairs, as well as account information such as expiry date, and any other special fields. This file should be protected at all cost and only the root user should have read permission to it.

`/etc/groups`

The groups file contains all the group membership information, and optional items such as group password (typically stored in gshadow on current systems). This file must be globally readable for the system to behave correctly. The format is:

```
groupname:encrypted_password:GID:member1,member2,member3
```

A group may contain no members (i.e. it is unused), a single member or multiple members, and the password is optional (and typically not used).

Restricted Shell

There are restricted versions of the Bourne and Korn shells (rsh and rksh), that prohibit changing directory with cd, setting the value of $PATH, using command names containing slashes, and redirecting output using > and >>.

`/etc/login.defs`

This file (/etc/login.defs) allows to define some useful default values for various programs such as useradd and password expiry. It tends to vary slightly across distributions and even versions, but typically is well commented and tends to contain sane default values.

`/etc/shells`

The shells file contains a list of valid shells, if a user's default shell is not listed here they may not log in interactively.

`/etc/securetty`

This file contains a list of tty's that the root can log in from. Console tty's are usually /dev/tty1 through /dev/tty6. Serial ports are /dev/ttyS0 and up typically. If root is to be allowed to login via the network then add /dev/ttyp1 have to added and up. Generally only root should be allowed to login from /dev/tty1, and it is advisable to disable the root account altogether. Before doing this sudo or program has to be installed that allows root access to commands.

Note :
How to temporarily disable user logins (Solaris)
The following procedures show how to temporarily disable user logins in Solaris
Become superuser.
Create the /etc/nologin file using an editor.
```
# vi /etc/nologin
```
Include a message regarding system availability.
Close and save the file.

Example—Disabling User Logins

This example shows how to notify users of system unavailability.

```
# vi /etc/nologin
```

(Add system message here)

```
# cat /etc/nologin
```

No logins permitted

The system will be unavailable until 12 noon

Note : With regard to disabling logins, in Solaris it will work and there is no "nologin" file concept in SCO. However, the /etc/profile can be edited with an appropriate message with the last line as /bin/true and exit. This will throw all the users who are trying to login. To ensure that it does not throw the root, a condition by checking the LOGNAME for non-root has to be added.

Locking and Unlocking of Accounts

Using the passwd -l command the root can lock the passwd of a user. The locking is performed by rendering the encrypted password into an invalid string (by prefixing the encrypted string with an !).

Consider the following example.

```
[root@icg3 unix]# passwd -l unix
```

Changing password for user unix
Locking password for user unix
passwd: Success
[root@icg3 unix]#

The "!" mark can be found in the beginning of the encrypted password string in the /etc/shadow file.

```
[root@icg3 unix]# cat /etc/shadow | grep unix
unix:!$1$aya1el49$x9PHEvual1SjovjT0uiQe/:11467:0:99999:7:-1:-1:134532716
[root@icg3 unix]#
```

For unlocking purpose use –u option has to be used , the –u option will unlock the account password by removing the! .

prefix. This option is available to root only.

```
[root@icg3 unix]# passwd -u unix
Changing password for user unix
Unlocking password for user unix
passwd: Success
[root@icg3 unix]#
```

In the above examples, the password of the user named unix has been and the output show that it is successful. The second step checks the /etc/shadow file for the !. The third step unlocks the password of the user and here again it indicates that is successful.

In Solaris environment, a user's password can be locked by using "-l " option with the passwd command.

Example

```
# passwd -l unix  <—  to lock the account named unix
# passwd   unix <—  to unlock the account by issuing a new password.
```

13.5 Short Summary

- Security in Unix can be classified into two File level security and System level security

- For each file there are three classes of users- owner (the user who creates the file), group (members who belong to owner's group), others (Rest of the users)
- There are three types of permissions –read, write and execute
- The default permissions of a file or a directory are controlled by the umask value
- Chmod command is used to change the access permissions of a file
- Chown command is used to change the ownership of a file and chgrp command is used to change a group of a file
- Setuid and setgid permissions can be set on executable files and setgid permission can be set on directories
- Save text attribute (sticky bit) can be set on public directories for the purpose of security
- Access of a system is controlled by login name and password assigned to the users which can be found in /etc/passwd and /etc/shadow files
- Root account is automatically created when the operating system is installed
- Root can temporarily disable user logins with the help of /etc/nologin file

13.6 Brain Storm

1. Explain how the access permissions are given to users.
2. What is the purpose of "umask" value?
3. Explain Chown and chmod commands.
4. Find out the difference between setuid and setgid permission.
5. What is the purpose of asroot command?

Lecture 14

Printer Management

Objectives

In this lecture you will learn the following

- Able to configure the Print Service
- About Print Service Architecture
- Able to print a file

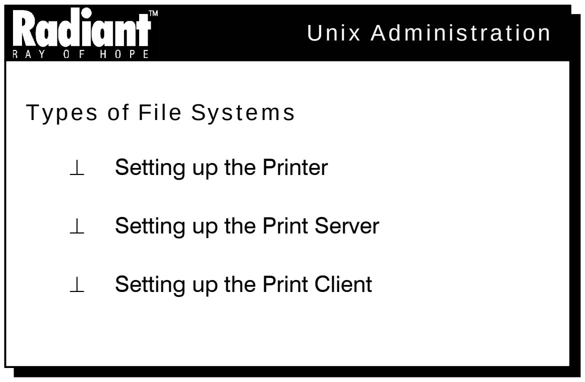
Coverage Plan

Lecture 14

14.1 Snap Shot

Printer Manager allows the easy set up of both the client and server side of the SVR4 print subsystem. Printer Manager is used for creating new print servers and administering the connection of the clients to their print servers. Printer Manager supports use the LP print service software. Using the LP software gives the users complete access to all of the printer manager functionality including simplified and centralized printer administration through the use of name services.

14.2 Configuring Print Services



Radiant™ RAY OF HOPE — Unix Administration

Types of File Systems

⊥ Setting up the Printer

⊥ Setting up the Print Server

⊥ Setting up the Print Client

Configuring Printer services in Unix involves three main tasks: setting up the printer, setting up the printer server and setting up the print client.

Setting up the Printer

A printer is physically connected to the system and the switches and other settings are set.

Sfetting up the Print Server

A print server is configured to provide access to the local printer by editing the */etc/printcap* file in Linux.

Setting up the Print Client

A print client is configured to provide access to the remote printer.

Print server requirements

- Minimum of 20 – 50 Mbytes in spooling directory
- At least 32 Mbytes of RAM

The system to which the printer is attached is known as the print server for that printer. Any networked system that meets the following minimum requirements can be a print server:

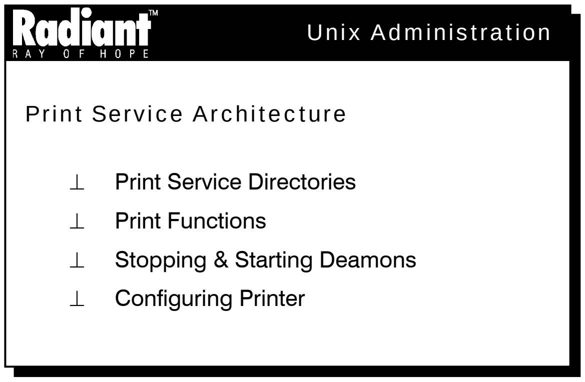
Spooling directory space of 20 – 50 Mbytes The Spooling space (*/var/spool)* is the most important factor. The spool area is used to store and process requests in the print queue. The amount of space required depends on the printing needs.

Daemon:

```
# lpd is the daemon which provides the print service.
```

Note : A printer can be configured in Solaris using *lpadmin, accept, enable* commands. *lp* is the daemon which provides the print service in Solaris and SCO. These services are invoked in run level 2.

14.3 Print Service Architecture


Radiant™ RAY OF HOPE Unix Administration

Print Service Architecture

⊥ Print Service Directories
⊥ Print Functions
⊥ Stopping & Starting Deamons
⊥ Configuring Printer

Print Service Directories

The print service directories are as follows

/usr/bin	The print service user commands
/usr/share/lib	The print server configuration
/var/spool	Spooling directory for pending print requests
/usr/sbin	The print service administration commands

Print Functions

The print functions are as follows
- Queuing
- Tracking
- Initialization

Queuing

When the print requests are spooled, the jobs are lined up with other jobs that are waiting to be printed. This process of lining up of jobs is called queuing.

Tracking

The print service tracks the status of every job to enable users to remove jobs and system administrators to manage jobs.

Initialization

The print service initializes a printer before sending it a print job to ensure that it is in a known state.

Stopping & Starting Daemons

The print daemon can be stopped as follows:

```
# /etc/rc.d/init.d/lpd stop
```

The print daemon can be started as follows:

```
# /etc/rc.d/init.d/lpd start
```

Configuring Printer

The `/etc/printcap` file explains the Linux OS about a printer. Each entry defines a printer, provides a name to be addressed and explains the OS about ways of handling output to it. Multiple "printers" that access the same physical printer can be installed in a system.

Practice 14.1

The following example illustrate how to configure a printer.

The */etc/printcap* file is first edited.

```
# vi /etc/printcap
#Local Printer
hp|Our printer:\
:sd=/var/spool/lpd/hp:\
:mx#0:\
:sh:\
:hp=/dev/lp0:\
s
```

The printer name

The first line of a */etc/printcap* entry is the printer name. It contains the printer name, which is followed by a " | " character, a text description of the printer, and ends with a ":" character. For example:

Hp:\

is a printer called "HP" with the description "OurPrinter "

The spool directory

This is a directory for print spooling. The tag "`sd=`" identifies the spool directory for the printer.

Additional Tags

There are three additional things to be put in the `printcap` entries. They can be seen below:

```
:mx#0:\

:sh:\

:sf:
```

The "*`mx#0`* "means "don't limit the size of the file we can print". The "`sf`' prevents a form-feed from being sent after the document ends. Finally, the ``` ``sh'' ``` prevents a header page from being sent at the beginning of each job. These may be added to the `printcap` entries.

The printer device

The tag "`hp=`" takes the name of the printing device. In this case, it is `/dev/lp1`, the parallel port. There are printers that connect to the serial port.

The Print Spool Daemon, `lpd`

The program that actually does the spooling and printing of files is `lpd`. It reads the */etc/printcap* file. You can also execute *lpd* can also be execute as a background job from the command line if logged on as root. *lpd* is the line printer daemon (spool area handler) and is normally invoked at boot time.

Example

```
# lpd –l  /dev/lp0
```

Available options:

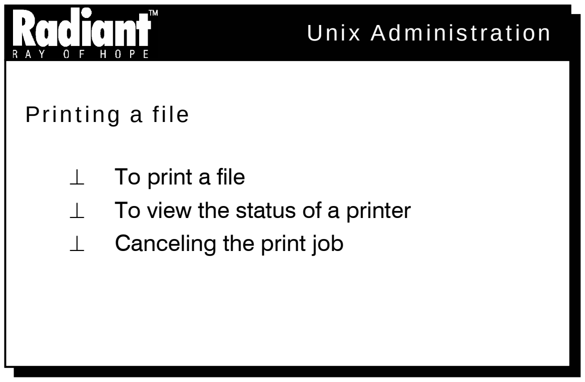
-l	– The **-**l flag causes *lpd* to log valid requests that are received from the network.
port#	– The port indicates under which port the request has been accepted.
lpq	– The spool queue examination program **lpq** examines the spooling area used by lpd for printing files on the line printer, and reports the status of the specified jobs or all jobs associated with a user. l*pq,* that is invoked without any arguments, reports on any jobs currently in the queue.

Note : *lpadmin* is the command that is used to configure the printer in Solaris. *lp* is the service which provides the print services. *lpadmin* configures the LP print service by defining printers and devices. It is used to add and change printers, to remove printers from service, to set or change the system default destination**.**

Example

# lpadmin -p printer options
# lpadmin –p epson –v /dev/lp0
-p Indicates the name of the printer
-v Indicates the port

14.4 Printing a File



Radiant™ RAY OF HOPE Unix Administration

Printing a file

- To print a file
- To view the status of a printer
- Canceling the print job

To Print a File

A file can be printed by using the following command

```
# lpr [-l] [-Pprinter] [job # ...]
```

Options:

-P - Specifies a particular printer.

-l - Information about each of the files comprising the job entry is printed.

Practice 14.2

Consider the following example illustrate to print file.

```
# lpr –Php /radiant/kishore
```

The above example shows the usage of the *lpr* command , -p inidcates the name of the printer , */radiant/kishore* is the name of the file to be printed.

The */var/spool/lpd/hp* directory is the spool directory where all the jobs of a printer queue are stored,

# */var/spool/lpd/hp/status* is a file that contains the status of printer.

Example

```
# vi /var/spool/lpd/hp/status
```

Note :

Hp enabled and accepting requests

`# /var/spool/lpd/hp/lock file contains the print jobs.`

Lock file is used to obtain the pid of the current daemon and the job number of the currently active job.

lp is used to print a file in Solaris and SCO.

Syntax

```
lp [file name]
```

Consider the following example.

```
# lp /kishore
```

To View the Status of a Printer

lpc is a line printer control program that is used by the system administrator to control the operation of the line printer system. For each line printer configured in /etc/printcap, *lpc* may be used to disable or enable a printer, disable or enable a spooling queue of a printer.

Syntax

```
/usr/etc/lpc  [command  [argument...] ]
```

# lpc abort { all | printer ... } Terminates an active spooling daemon on the local host immediately and then disables printing (preventing new daemons from being started by *lpr*) for the specified printers.

# lpc clean { all | printer ... } Removes any temporary files, data files, and control files that cannot be printed (i.e., do not form a complete printer job) from the specified printer queue(s) on the local machine.

#lpc disable { all | printer ... } Turns the specified printer queues off. This prevents new printer jobs from being entered into the queue by *lpr*.

# lpc down { all | printer } message ... Turns the specified printer queue off, disable printing and puts *message* in the printer status file.

# lpc enable { all | printer ... } Enables spooling on the local queue for the listed printers. This will allow *lpr* to put new jobs in the spool queue.

# lpc restart { all | printer ... } Attempts to start a new printer daemon. This is useful when some abnormal condition causes the daemon to die unexpectedly leaving jobs in the queue. *Lpq* will report that there is no daemon present when this condition occurs. If the user is the super-user, it will try to abort the current daemon first (i.e., kill and restart a stuck daemon).

# lpc start { all | printer ... } Enables printing and starts a spooling daemon for the listed printers.

# lpc status { all | printer ... } Displays the status of daemons and queues on the local machine.

# lpc stop { all | printer ... } Stops a spooling daemon after the current job completes and disables printing.

# lpc up { all | printer ... } Enables everything and starts a new printer daemon. Undoes the effects of *down*.

Note:
lpstat is used to check the status of the print service in Solaris.

```
# lpstat [ printer ]
```

Canceling the Print job

Lprm removes a job, or jobs, from a printer's spool queue. Since the spooling directory is protected from users, using *lprm* is normally the only method by which a user may remove a job.

Syntax

```
lprm [ -P printer ] [ - ] [ job # ... ] [ user ... ]
```

Options

-P *printer* - Specifies the associated queue with a specific *printer* (otherwise the default printer is used).

user - Causes **lprm** to attempt to remove any jobs that are queued and belonging to that user (or users). This form of invoking **lprm** is useful only to the super-user.

- - A single '**-**' is given, **lprm** will remove all jobs of a user. If the super-user employs this flag, the spool queue will be emptied entirely.

job # - A user may remove an individual job by specifying its job number.

Note :
cancel is the command which is used to cancel the print request in Solaris.

```
# cancel [ request-ID ... ]
# cancel  epson-1
```

User commands

Linux	**Solaris 2.7**	**Description**
lpr	lp	Submits a request to the printer
lpq	lpstat	Reports on the status of the print request and service
lprm	cancel	Cancels a print request

Administrative Commands:

Administrative commands are located in **/usr/lib**. These files are symbolic links to the actual files residing in **/usr/sbin** and **/usr/lib/lp**. The commands **accept** and **reject** are in **/usr/sbin**, and the commands **enable** and **disable** are in **/usr/bin**.

Administrative Commands

linux	**SunOS 5.X**	**Description**
lpc enable	enable	Enables the queue for the named printer
lpc disable further	disable	Disables the queue of the named printer for requests
lpc	**lpadmin**	**Configure the print service**

14.5 Short Summary

- *Lpd* (Linux) is the daemon that controls the print services
- */etc/printcap (*linux) is the file which contains the printer details
- */var/spool/lpd* (linux)is the spool directory which is used to store the print requests
- *Lpc* (linux) is used to control the operation of the printer
- *Lprm* (Linux) is used to cancel the print requests

14.5 Brain Storm

1. What are the Daemons that control the Print services?
2. How can Print services be started and stopped?
3. What are the default spool directories?
4. How can the printer be enabled and disabled ?
5. How can the print requests be cancelled ?

Lecture 15

Backup and Recovery

Objectives

In this lecture you will learn the following

- Understand about Backup utility
- Able to apply the Backup command

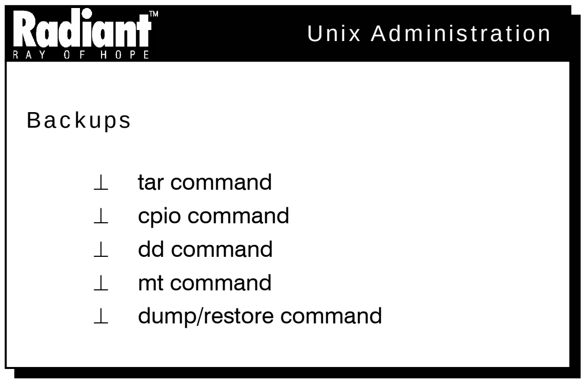
Coverage Plan

Lecture 15

15.1 Snap Shot

Backing of files is one of the most crucial functions of system administration. It must be planned and carried regularly at scheduled intervals.Backing of files means making copies of them, usually on removable media, as safeguard in case the originals get lost or damaged. Backups should be performed on a regular basis.

15.2 Backups



Radiant™ RAY OF HOPE — Unix Administration

Backups

⊥ tar command
⊥ cpio command
⊥ dd command
⊥ mt command
⊥ dump/restore command

Backup is the procedure of taking a copy of relevant and important information on the workstation or server systems.Backing of files is one of the most crucial functions of system administration. It must be planned and carried regularly at scheduled intervals for the following five major reasons.

Accidental file removal	–	A user or superuser may accidentally removed an important file.
External failure of system	–	Power failure may lead to file system corruption and data loss.
Internal failure of systems	–	Hard disk crash or failure in some other system components which may lead to data loss.
Environmental damages	–	Natural disaster may destroy the local site.

So backups should be protected against these types of dangers.

Backing of files means making copies of them, usually on removable media, as safeguard in case the originals get lost or damaged. Backups should be performed on a regular basis. If backups are automated sufficiently, the only limitation on how often backups are performed should be the size and availability of data.

tar command

One of the oldest and most often used commands for archiving and backing up files is the tar command. Due to its simplicity and ease of use, this becomes the most common format for tape and disk archives today.

One of the best features of tar is the flexibility to save any medium, since it treats file and tape (or other backup media) device targets the same.

The tar command helps to backup single or multiple files in a directory hierarchy.

The format of the tar command is

```
# tar options [ argument ] (tar filename ….) (files to backup or restore)
```

Options

c - Creates a new archive; writing begins at the beginning of the archive, and not after the last file.

t - The names of the specified files are listed each time that they occur on the archive. If no files argument is given, all the names on the archive are listed.

v - Normally, tar does its work silently. The v (verbose) function modifier causes tar to display the name of each file it treats, preceded by the function letter. With the t function, v gives more information about the archive entries than just the name.

f - Causes tar to use the next argument as the name of the archive instead of the default device listed in /etc/default/tar. If the name of the file is a dash (-), tar writes to the standard output or reads from the standard input, whichever is appropriate. Thus, tar can be used as the head or tail of a pipeline. tar can also be used as archive file or device F (default /dev/rmt0) used to move hierarchies with the command:

```
cd fromdir; tar cf - . | (cd todir; tar xf -)
```

x - The named files are extracted from the archive. If a named file matches a directory whose contents had been written onto the archive, this directory is (recursively) extracted. The owner, modification time, and mode are restored (if possible). If no files argument is given, the entire contents of the archive are extracted. If several files with the same name are on the archive, the last one overwrites all earlier ones. There is no way to ask for the nth occurrence of a file.

t - The names of the specified files are listed each time that they occur on the archive. If no files argument is given, all the names on the archive are listed.

r - The named files are written to the end of an existing archive. This function letter is only valid for appending files to disk archives. When specifying the absolute path of an archive device with the function modifier, use the n function modifier to indicate that the device is not a magnetic tape. this function letter cannot be used with tape devices.

The tar command is unaware of the file systems; however if a directory is specified as a tar argument, it copies the entire hierarchy below a directory. use a ./ in front of the directory name so that it can be restored relative to a current working directory.

Copying Files to Diskette Using Tar

To copy files test1 to test5 under home directory to a diskette.

Consider the following example

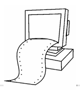
The user must be login as single user mode. This is inevitable in view of maintaining consistency of files.

```
#shutdown –g 10 -y
```

Practice 15.1

Consider the following command illustrates a method to take backup.

```
# tar cvf /dev/fd0 ./test*
```



```
./test
./test1
./test2
./test3
```

In the above example *tar* is the command which is used to take backup, */dev/fd0* is the device used to take backup and ./*test** is the directory.

c - Creates a new archive; writing begins at the beginning of the archive, instead of after the last file.

v - Normally, tar does its work silently. The v (verbose) function modifier causes tar to display the name of each file it treats, preceded by the function letter. With the t function, v gives more information about the archive entries than just the name.

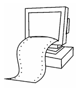
f - Causes tar to use the next argument as the name of the archive instead of the default device listed in /etc/default/tar. If the name of the file is a dash (-), tar writes to the standard output or reads from the standard input, whichever is appropriate. Thus, tar can be used as the head or tail of a pipeline.

Note :
To use tape archive as device file use /dev/st0 (st0 – indicates the first tape drive) is to be used.
In Solaris, it is different.
`/dev/rmt/0`
`/dev/rmt-logical device name of tape`
0-indicate first tape drive

Practice 15.2

The following example illustrates how to display the content of the floppy using tar.

```
# tar tvf /dev/fd0
```



```
-rw-r—r— root/root17 2000-10-18 08:25:15 ./test
-rw-r—r— root/root17 2000-10-18 08:25:25 ./test1
-rw-r—r— root/root17 2000-10-18 08:25:32 ./test2
-rw-r—r— root/root17 2000-10-18 08:25:41 ./test3
```

Here t represents the names of the specified files are listed each time when they occur on the archive. If the files argument is not given, all the names on the archive are listed. v means normally, tar does its work silently. The v (verbose) function modifier causes tar to display the name of each file it treats, preceded by the function letter. The function, gives more information about the archive entries than just the name. f shows causes tar to use the next argument as the name of the archive instead of the default device listed in /etc/default/tar. If the name of the file is a dash (-), tar writes to the standard output or reads from the standard input, whichever is appropriate. Thus, tar can be used as the head or tail of a pipeline.

Note : If we use tape archive as device file use /dev/rmt0.

Restoring Files Using Tar

Extracting files under home directory,

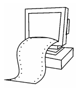
move to the directory,that restores the files.

```
#cd /home
```

Practice 15.3

The following example illustrates about restoring the file.

```
# tar xvf /dev/fd0 ./test*
```



```
./test
./test1
./test2
./test3
```

Here, x represents the named files are extracted from the archive. If a named file matches a directory whose contents had been written onto the archive, this directory is (recursively) extracted. The owner, modification time, and mode are restored (if possible). If no files argument is given, the entire contents of the archive are extracted. If several files with the same name are on the archive, the last one overwrites all earlier ones. Normally, tar does its work silently. The v (verbose) function modifier causes tar to display the name of each file it treats, preceded by the function letter. With the t function, v gives more information about the archive entries than just the name. f means causes tar to use the next argument as the name of the archive instead of the default device listed in /etc/default/tar. If the name of the file is a dash (-), tar writes to the standard output or reads from the standard input, whichever is appropriate. Thus, tar can be used as the head or tail of a pipeline.

cpio command

One of the more popular generic backup utilities in use today is the cpio command. In large part, its popularity is due to its capability to append backup volumes and span tapes, allowing to create incremental backup sets and full system backups without losing data integrity.

cpio allows to copy files into and out of a cpio archive. The term cpio stands for "copy in/out". cpio copies files to and from an archive file or another directory hierarchy. There are two main options to cpio which determine its mode of operation.

These modes are used to create an archive (cpio -o), extract files from an archive (cpio -i)

c - For portability, write header information in ASCII Character form. Always use this option should be used when the origin and destination machines are of different types.

c - (copy out) Reads a list of pathnames from the standard input, and copies those files onto the standard output together with pathname and status information. Output is padded to a 512- byte boundary by default.

i - (copy in) Extracts files from the standard input, which is assumed to be the product of a previous cpio -o. Only files with names that match the wildcard patterns are selected.

v - Lists the files processed, or with -t, give an 'ls -l' style table of contents listing.

t - Prints a table of contents of the input. No files are created.

b - Sets the I/O block size to 5120 bytes.Initially the block size is 512 bytes.

m - MESSAGE, —message=MESSAGE Prints MESSAGE when the end of a volume of the backup media (such as a tape or a floppy disk) is reached, to prompt the user to insert a new volume.

If MESSAGE contains the string "%d", it is replaced by the current volume number (starting at 1).

-a - Suppresses absolute filenames. A leading "/" character is removed from the filename during copy-in.

If a pattern is provided, it should match the relative (rather than the absolute) pathname.

Note : In linux this -a, option is different, here it is used to append to an existing archive. Only works in copy- out mode. The archive must be a disk file specified with the -O or -F (—file) option

Practice 15.4

The following example illustrates the copying of files using cpio.

```
#cd /home
# ls tes*|cpio -ovc >/dev/fd0
```

```
test
test1
test2
test3
2 blocks
```

In the above example, files were copied under the directory called /home/test by using *cpio* command. o represents (copy out) Reads a list of pathnames from the standard input, and copies those files onto the standard output together with pathname and status information. Output is padded to a 512- byte boundary by default. v means list the files processed, or with -t, give an 'ls, l shows style table of contents listing, and c means for portability, write header information in ASCII Character form. Always use this option when the origin and destination machines are of different types >/dev/fd0-Output is redirected to a floppy disk in the first drive.

Restoring the files using cpio

The user has to move to the respective directory where the files are to be restored.

Practice 15.5

Consider the following example illustrate how to restore the files under home directory.

```
#cd /home
# cpio -ivc </dev/fd0
```

```
test
test1
test2
test3
2 blocks
```

Here, i represents (copy in) Extracts files from the standard input, which is assumed to be the product of a previous cpio -o. Only files with names that match the wildcard patterns are selected, v shows list the files processed, or with -t, give an 'ls, l shows style table of contents listing, c means for portability, write header information in ASCII Character form. Always use this option when the origin and destination machines are of different types `>/dev/fd0`-Output is redirected from a floppy disk in the first drive.

Comparision between tar and cpio

CPIO	**TAR**
Designed to backup a set of files	Designed to backup subdirectories
Can handle backups that span over different different ways	Not with this option but can be achieved tapes in
Cpio skips over bad sections on tape and continues	It will not skip over bad sections, so tar crashes and burns

dd command

The purpose of the Unix `dd` command is to copy data from one device to another; the command stands for *device dump*. The `dd` command transfers raw data between devices.

Syntax

```
dd [OPTION]...
```

This command copies a file, convert and formatts according to the following options.

Options:

if=FILE	read from FILE instead of stdin
of=FILE	write to FILE instead of stdout
bs=BYTES	force ibs=BYTES and obs=BYTES
ibs=BYTES	read BYTES bytes at a time
obs=BYTES	write BYTES bytes at a time

Practice 15.6

Consider the following example illustrates how to copy the contents of one floppy disk to another, using /tmp as a temporary storage area.

The source disk is inserted in the drive, and the following command entered:

```
#dd if=/dev/fd0 of=/tmp/tmpfile bs=50k

28+1 records in
28+1 records out
```

In the above example *dd* is the command which is used to copy from one disk to another disk.Numbers of full and partial blocks read in the current input block size.ie bs=50k 28+1 records out means.Numbers of full and partial blocks written in the current output block size.

```
Where if=FILE
read from FILE instead of stdin
of=FILE
write to FILE instead of stdout
bs=BYTES
force ibs=BYTES and obs=BYTES
```

Practice 15.7

This example shows how to copy the data in temporary file, /tmp/tmpfile to the destination that is inserted after removing the source.

```
#dd if=/tmp/tmpfile of=/dev/fd0


28+1 records in
28+1 records out

where 28+1 records in means
Numbers of full and partial blocks read in the current input block size. ie
28+1 records out means
Numbers of full and partial blocks written in the current output block size.
Where if=FILE
read from FILE instead of stdin
of=FILE
write to FILE instead of stdout
  bs=BYTES
force ibs=BYTES and obs=BYTES
```

Finally remove the temporary file:

```
#rm /tmp/tempfile
```

mt command

The mt command enables direct tape manipulation.

Syntax

```
mt[ -f tape-device-name ] command [count ]
```

Commands

status	Displays status information about the tape drive
rewind	Rewinds the tape
retention	Rewinds the cartridge tape completely
erase	Erases the entire tape
fsf	Forward skips count tape files
bsf	Backward skips count tape files

eom	Skips to the end of the records cd media

Table 15.1

Note :

Consider the following example
How do the tar and mt commands are used together in Solaris?

```
#tar cvf/dev/rmt/0ntest
a test/ 0 tape blocks
a test/test1 3 tape blocks
in this example we are copying a directory test to tape with no
rewind option
/dev/rmt/0 – logiacl device name of tape drive
n – no rewind option
#tar cvf/dev/rmt/0n oracle
a oracle/ 0 tape blocks
a oracle/initorcl.ora 6 tape blocks
here also we are copying a directory oracle with no rewind option

#tar cvf/dev/rmt/0 doc
a doc/ 0 tape blocks
a doc/doc1 2 tape blocks
a doc/doc24 tape blocks
```

here doc directory for copied with no rewind option so the tape rewind automatically

now if I want to restore the doc directly I want move to the third archive

because the first two tape archives contains test and oracle directory respectively

use mt to move to the third archive as show below:

```
mt –f /dev/rmt/0n fsf 2
 -f-dump file(device we use to take the backup,here it is the
tape)
/dev/rmt/0n – the tape which we used to take the backup
n-no rewind option
fsf 2 - Forward Skip File-skips 2 files
now use tar to restore doc directory

tar xvf /dev/rmt/0
x doc, 0 bytes, 0 tape blocks
x doc/doc1, 500 bytes, 2 tape blocks
x doc/doc2, 1000 bytes, 4 tape blocks
```

Dump and Restore

A favorite amongst many System Administrators, dump is used to perform backups and restore is used to retrieve information from the backups.

dump

There is a version of dump for Linux. RedHat 5.0 includes an RPM package which includes dump. If the system does not have dump and restore installed it should be installed. RedHat provides a couple of tools to install these packages: rpm and glint. glint is the GUI tool for managing packages. Refer to the RedHat documentation for more details on using these tools.

The dump package can be found under the Utilities /System folder. Before the dump package is installed the rmt package.

dump

dump is generally used to backup an entire partition (file system). If given a list of filenames, dump will backup the individual files.

Syntax:

```
dump [ options [ arguments ] ] file system
dump [ options [ arguments ] ] filename
```

Arguments must appear after all options and must appear in a set order.

Using dump command the user can take full or incremental backups, dump works on the concept of levels (it uses 9 levels). A dump level of 0 (full backup) means that all files will be backed up. A dump level of 1...9 (incremental) means that all files that have changed since the last dump of a lower level will be backed up.

If the –u option is specified with the dump command it will update the dump information in the /etc/dumpdates file, which will help the system to perform the next incremental backup. This file, after the dump performed, contains the date and time of backup and the level specified for the backup.

Options

0-9	-	Dump level.
a archive-file	-	Refers the Archive-file will be a table of contents of the archive
f dump-file	-	Specifies the file (usually a device file) to write the dump to, a - specifies standard output
u	-	Updates the dump record (/etc/dumpdates)
v	-	After writing each volume, rewinds the tape and verifies. The file system must not be used during dump or the verification.

Arguments for dump

Consider the following example

```
dump 0dsbfu 54000 6000 126 /dev/rst2 /usr
```

The above command leads to full backup of /usr file system on a 2.3 Gig 8mm tape connected to device rst2 The numbers here are special information about the tape drive on which the backup is being written on.

The restore command

The purpose of the restore command is to extract files archived using the dump command. This command restore provides the ability to extract single individual files, directories and their contents and even an entire file system.

Syntax

```
restore -irRtx [ modifiers ] [ filenames ]
```

The restore command has an interactive mode where commands like ls etc can be used to search through the backup.

Arguments

-I	-	Interactive, directory information is read from the tape after which the user can browse through the directory hierarchy and select files to be extracted.
-r	-	Restore the entire tape. Should only be used to restore an entire file system or to restore an incremental tape after a full level 0 restore.
-t	-	Table of contents, if no filename provided, root directory is listed including all subdirectories (unless the h modifier is in effect)
-x	-	Extract named files. If a directory is specified, it and all its sub-directories are extracted.

Arguments for the restore Command.

Modifiers	**-**	**Purpose**
a archive-file	-	Uses an archive file to search for a file's location. Converts contents of the dump tape to the new file system format
d	-	Turns on debugging
h	-	Prevents hierarchical restoration of sub-directories
v	-	Verbose mode
f dump-file	-	Specifies dump-file to use, - refers to standard input
s n	-	Skips to the nth dump file on the tape

Argument modifiers for the restore Command.

Using dump and restore without a tape

Practice 15.8

For all our experimentation with the commands in this chapter we are going to work with a practice file system. Practising backups with hard-drive partitions is not going to be all that efficient as they will almost certainly be very large. Instead we are going to work with a floppy drive.

The first step then is to format a floppy with the ext2 file system. By now you should know how to do this. Here's what I did to format a floppy and put some material on it.

```
[root@icg]# /sbin/mke2fs /dev/fd0
mke2fs 1.10, 24-Apr-97 for EXT2 FS 0.5b, 95/08/09
Linux ext2 filesystem format
Filesystem label=
360 inodes, 1440 blocks
72 blocks (5.00%) reserved for the super user
First data block=1
Block size=1024 (log=0)
Fragment size=1024 (log=0)
```

```
1 block group
8192 blocks per group, 8192 fragments per group
360 inodes per group
```

Writing inode tables: done

Writing superblocks and filesystem accounting information: done

```
[root@icg]# mount -t ext2 /dev/fd0 /mnt/floppy
[root@icg]# cp /etc/passwd /etc/issue /etc/group /var/log/messages /mnt/floppy
[root@icg dump-0.3]#
```

Doing a level 0 dump

Let's assume If we want to do a level 0 dump of the /mnt/floppy file system.

[root@icg]# /sbin/dump 0f /tmp/backup /mnt/floppy

```
DUMP: Date of this level 0 dump: Sun Jan 25 15:05:11 1998
DUMP: Date of last level 0 dump: the epoch
DUMP: Dumping /dev/fd0 (/mnt/floppy) to /tmp/backup
DUMP: mapping (Pass I) [regular files]
DUMP: mapping (Pass II) [directories]
DUMP: estimated 42 tape blocks on 0.00 tape(s).
DUMP: dumping (Pass III) [directories]
DUMP: dumping (Pass IV) [regular files]
DUMP: DUMP: 29 tape blocks on 1 volumes(s)
DUMP: Closing /tmp/backup
DUMP: DUMP IS DONE
```

The arguments to the dump command are

0	-	This tells dump I wish to perform a level 0 dump of the file system.
F	-	This is telling dump that I will tell it the name of the file that it should write the backup to.
/tmp/backup	-	This is the name of the file I want the backup to go to. Normally, this would be the device file for a tape drive or other backup device. However, since I don't have one I'm telling it a normal file.
/mnt/floppy	-	This is the file system I want to backup.

What this means is that I have now created a file, /tmp/backup, which contains a level 0 dump of the floppy.

```
[root@icg]# ls -l /tmp/backup
-rw-rw-r— 1 root tty 20480 Jan 25 15:05 /tmp/backup
```

Restoring the backup

Now that we have a dump archive to work with, we can try using the restore command to retrieve files.

```
[root@icg dump-0.3]# /sbin/restore -if /tmp/backup
restore > ?
```

Available commands are:

```
ls [arg] - list directory
cd arg - change directory
pwd - print current directory
add [arg] - add ‘arg’ to list of files to be extracted
delete [arg] - delete ‘arg’ from list of files to be extracted
extract - extract requested files
setmodes - set modes of requested directories
quit - immediately exit program
what - list dump header information
verbose - toggle verbose flag (useful with “ls”)
help or ‘?’ - print this list
If no ‘arg’ is supplied, the current directory is used
restore > ls
.:
group issue lost+found/ messages passwd

restore > add passwd
restore > extract
You have not read any tapes yet.
Unless you know which volume your file(s) are on you should start
with the last volume and work towards towards the first.
Specify next volume #: 1
Mount tape volume 1
Enter “none” if there are no more tapes
otherwise enter tape name (default: /tmp/backup)
set owner/mode for ‘.’? [yn] y
restore > quit
[root@icg]# ls -l passwd
-rw-r—r— 1 root root 787 Jan 25 15:00 passwd
```

Alternative

Rather than backup to a normal file on the hard-drive the user could choose to backup files directly to a floppy drive (i.e. use /dev/fd0 rather than /tmp/backup). One problem with this alternative is that the user limited to 1.44Mb. According to the “known bugs document” distributed with Linux dump it does not yet support multiple volumes.

Note: In Solaris environment the ufsdump and ufsrestore commands which are almost similar to the dump/restore commands in Linux can be used.

15.3 Short Summary

- To ensure a usable backup, the file system should be unmounted or the the system should be in single user run level before the backup is performed
- The tar command enables user to backup single or multiple files in a directory hierarchy
- The tar command is used to create a tape archive or extract file from the archive
- The mt command is used to control magnetic tape operations, includes positioning of the tape to the beginning of a data set, rewinding the tape and even erasing the tape
- Cpio command allows to copy files into and out of cpio archive
- The dd command is used to copy data from one disk to another (cloning)
- Using dump command the user can take full or incremental backups
- Dump works on the concept of levels (it uses 0-9 levels)

- The previous dump information are stored in the /etc/dumpdates file
- The restore command is to extract files archived using the dump command
- Restore provides the ability to extract single individual files, directories and their contents and even an entire file system

15.4 Brain Storm

1. What are the differences between tar and cpio commands?.
2. What is the purpose of the mt command?
3. Why is unmounting of a file system necessary before performing a backup?
4. Why should the data in a temporary directory be restored?
5. Explain dump/restore commands.

Lecture 16

Space Management

Objectives

In this lecture you will learn the following

- Understand about Space Management
- Able to set Quota for a user
- Knowing to check Quota consistency
- Able to check quota on a file system

Coverage Plan

Lecture 16

16.1 Snap Shot

In this chapter we are discussing how to specify the limits on two aspects of disk storage. The number of inodes a user or a group of users may possess the number of disk blocks that may be allocated to a user or a group of users. UNIX forces users to stay under their disk consumption limit, taking away their ability to consume unlimited disk space on a system. If there is more than one file system which a user is expected to use, then space must be set for each file system separately.

16.2 Space Management

Radiant™ RAY OF HOPE Unix Administration

- ⊥ What is a quota?
- ⊥ Quotas set up for a user
- ⊥ Turning quotas on
- ⊥ Settingup quotas for single user
- ⊥ Setting quotas for multiple users
- ⊥ To check quota consistency
- ⊥ Checking quotas on a file system

Quota

Quota allows to specify limits on two aspects of disk storage: the number of inodes a user or a group of users may possess; and the number of disk blocks that may be allocated to a user or a group of users. Quota forces users to stay under their disk consumption limit, taking away their ability to consume unlimited disk space on a system. Quota is handled on a per user, per file system basis. If there is more than one file system which a user is expected to use, then quota must be set for each file system separately.

Quotas Setup for User

In Linux */etc/fstab* virtual file system table provides entries for mounting a file system at the time of booting the system.

Linux uses a special file called /etc/fstab (/etc/vfstab under Solaris). This file lists all the partitions that need to be mounted at boot time and the directory where they need to be mounted. Along with that information the parameters to the *mount* command can be passed.

Consider the following example.

To enable user quota support on a file system, *usrquota* has to be added to the fourth field containing the word "defaults as shown in the following example.

```
Modify /etc/fstab
# vi /etc/fstab
/dev/device /dir/to/mount ftype parameters        fs_freq fs_passno
/dev/hda6   /home         ext2   defaults,usrquota   1         1
```

Consider the following example.

To enable group quota support on a file system, *grprquota* has to be added to the fourth field containing the word "defaults as shown in the following example:

```
Modify /etc/fstab
```

```
# vi /etc/fstab
/dev/device /dir/to/mount ftype  parameters   fs_freq   fs_passno

/dev/hda6   /home          ext2   defaults,    grpquota   1      1
```

where:

/dev/device	–	Is the device to be mounted, for instance, /dev/hda4.
/dir/to/mount	–	Is the location at which the file system should be mounted on the directory tree.
Ftype	–	Is the file system type. This should be 4.2 under SunOS, ufs under Solaris, ext2 under Linux, nfs for NFS mounted file systems, swap for swap partitions, and proc for the /proc file system. Some operating systems, such as Unix, support additional filesystem types, although they are not as likely to be used.
Parameters	–	Are the parameters we passed to enable the quota using the *usrquota,grpquota* option. They follow the same comma-delineated format. An example entry would look like usrquota, grpquota.
fs_freq	–	Is used by dump to determine whether a file system needs to be dumped.
fs_passno	–	Is used by the fsck program to determine the order to check disks at boot time.

Create quota record “quota.user” and “quota.group”

Both the quota record files, quota.user and quota.group, should be owned by root, and read-write permission for root and none for anybody else. Login as root. Go to the root of the partition for which quota has to be enabled, then create quota.user and quota.group.

Configuring Quotas for User

Syntax:

```
# touch   [ file name ]
```

Consider the following examples.

```
# touch  /home/quota.user
```

To change the permission to read/write for root only the following can be done.

```
# chmod 600 /home/quota.user
```

Configuring quotas for group:

Syntax:

```
# touch   [ file name ]
```

Consider the following example.

```
# touch  /home/quota.group
```

The permission can be changed to read/write for root only as follows.

```
# chmod 600 /home/quota.group
```

Now the system has to be rebooted for the changes that have been made to take effect.

Note : Quota Setup on Sun Solaris: In Sun Solaris */etc/vfstab* virtual file system table provides entries for mounting file system at systems boot time.

To enable user quota support on a file system, *rq* has to be added to the seventh field containing the mount options:

```
Modify /etc/vfstab

# vi /etc/vfstab

devicetomount devicetofsck  mountpoint  fstype fsckpass mountatboot
mountoptions
/dev/dsk/c0t3d0s7/dev/rdsk/c0t3d0s7 /export/home ufs2 yes rq
```

The fields are described below:

device to mount	:	The name of the resource to be mounted.
Device to fsck	:	The name of the raw device to fsck; for a remote mount, the parameter is not applicable, and the entry should be -.
Mountp	:	The mount point on which the resource is to be mounted.
fstype	:	The file system type of the resource to be mounted.
fsckpass	:	the pass number to be used for multiple fsck's. For a remote mount, the parameter is not applicable, and the entry should be 0
automnt	:	Indicates whether the entry should be automounted by /sbin/mountall (yes) or not (no) when the client is booted or enters the appropriate run level.
Mntopts	:	Adds the parameters that are passed to enable the quota using the *rq* option.

Configuring quotas for user.

Turning Quotas on

quotaon is used to turn on quota accounting; quotaoff to turn it off. Actually both files are similar. They are executed at system startup and shutdown respectively.

To turn on quotas the following command should be used:

Syntax:

```
quotaon [options] [filesystem]
```

Options for *quotaon* command are as follows:

-v	verbose option.
-a	Turns quotas on for all the file systems with an *usrquta,grpquota* entry in */etc/fstab* file.

Consider the following example.

```
#quotaon  -va    /home
dev/hda6 – quota turned on
```

To turn off quotas the following command should be used:

```
quotaoff  [options]  [filesystem]
```

Options for *quotaoff* command:

-v	verbose option.

-a Turns quotas off for all the file system with an *usrquta,grpquota* entry in */etc/fstab* file.

Consider the following example.

```
#quotaoff  -va    /home
/dev/hda6 – quota turned off
```

Settingup quotas for a single user

```
# edquota –u  [ username ]
```

The following example shows how to set up quotas for a single user!

```
# edquota –u  kishore
Quotas for user kishore:
/dev/hda6: blocks in use: 0, limits (soft = 0, hard = 0)
         inodes in use: 0, limits (soft = 0, hard = 0)
```

Here blocks in use is the total number of blocks (in kilobytes) a user has consumed on a partition, inodes in use is the total number of files a user has on a partition.

Only the soft and hard limit values should be edited. The value of 0 means no limits will be imposed.

Soft Limit

Soft limit indicates the maximum amount of disk usage a quota user has on a partition. When combined with grace period, it acts as the borderline, which a quota user is issued warnings about his impending quota violation when passed.

Hard Limit

Hard limit works only when the grace period is set. It specifies the absolute limit on the disk usage, which a quota user cannot go beyond.

Settingup quotas for multiple users

To rapidly set quotas for, say 5 users, on the user's system to the same value the user's quota information has to be edited manually as shown below:

Syntax:

```
#edquota –p [ prototype-user ] [member list ]
```

Consider the following example.

```
#edquota  -p  kishore rams senthil rangarajan sekar
```

In the above example *kishore's* quota limit has been set to all the other users.

To Check quota consistency

Syntax:

```
#quotacheck  [ options ] [ file system ]
```

Options for the *quotacheck* command:

-v verbose option.

-a Turns quotas off for all the file systems with an *usrquta,grpquota* entry in */etc/fstab* file.

```
# quotacheck  -va /home
```

Checking quotas on a File System

To check the quotas in what are the file system it has been applied.

Syntax:

```
repquota  [ options ] [ file system ]
```

Options:

```
-v - verbose mode
```

Consider the following example.

```
# repquota  -va   /home
*** Report for user quotas on /dev/hda6 (/home)
                  Block limits                       File limits
User        used                 soft      hard      used      soft     hard
root        10224      0         0         1648      0         0
Kishore     184        1         4         46        1         5
```

In the above example, the quotas for the users in the file system /home is displayed in the output.

Block Limits:

Used	Is the current block usage
Soft	Is the soft block limit
Hard	The hard block limit

File Limits:

Used	Is the current block usage
Soft	Is the soft block limit
Hard	The hard block limit

16.3 Short Summary

- Quota allows to specify limits on two aspects of disk storage: the number of inodes a user or a group of users may possess; and the number of disk blocks that may be allocated to a user or a group of users.
- Allocation of space for users or groups on a file system
- Report about quotas applied on filesystem

16.4 Brain Storm

1. What is a *quota*?
2. How is quota activated?
3. How is the quota for a user assessed?
4. How is the quota status of a file system checked?
5. How is quota turned off?

Lecture 17

Scheduling of System Events

Objectives

In this lecture you will learn the following

- About Scheduling Events
- Knowing the types of scheduling

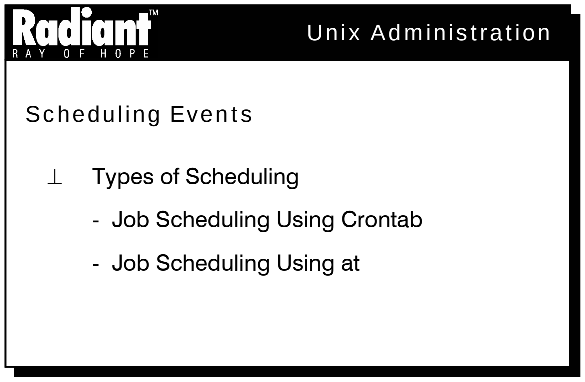
Coverage Plan

Lecture 17

17.1 Snap Shot - Introduction

UNIX gives the ability to schedule scripts and commands for execution at a later point in time.The exact time at which the command should be run can be specified. UNIX also provides a way of reporting on the scheduled jobs and removing them if the user does not want to execute them.

17.2 Scheduling Events



Radiant™ RAY OF HOPE Unix Administration

Scheduling Events

⊥ Types of Scheduling

- Job Scheduling Using Crontab
- Job Scheduling Using at

Scheduling of repetitive execution of system events can be accomplished using the crontab command. A single system event can be scheduled at a specified time by using the at command.

Job Scheduling Using Crontab

Cron is available with every Linux distribution. It has two main programs, `crond` and `crontab`. crond is the daemon that runs in the background and ensures that programs run as scheduled. crontab is the program that allows the programs that are scheduled to be run to be modified. The `/etc/crontab` file stores system scheduling information; each user has their own crontab file in `/var/spool/cron.`

Note : In Solaris environment the crontab file is stored in the /var/spool/cron/crontabs.

Cron has information about the tasks to be executed and their schedule by setting up the crontab files. Cron has to be informed about the task to be run and the exact time at which is has to be executed in minutes, hours, days, months, and years. Asterisks can be used to indicate all possible values. The **'crontab -e' command** can be used to edit the crontab file. `This command` will bring up an editor window and allow the crontab file to be edited.

A crontab file can be created without using crontab. But crontab has to be used to set up a particular file to be the crontab file. The command 'crontab file', has to be used. Here 'file' refers to the file that is to be used. This will replace the old crontab file and not append to it.

`The'crontab -e'` command uses vi by default. If the user prefers to use a different editor, the EDITOR environment variable has to be set. For example, to use emacs, in bash, use 'export EDITOR=emacs'.

Crond

Crond is a daemon. This program is executed when the Linux system is initially booted. It scans the /etc/crontab file and the /var/spool/cron directory, looking for the regularly scheduled jobs entered by root or other system users.

Crontab

The crontab command, not to be confused with the /etc/crontab file, is used by the users of the system to schedule personal events. The cron files are stored under the /var/spool/cron directory. The system administrators can control whether or not this facility exists on the system through the /etc/cron.allow or /etc/cron.deny files. All current cron jobs are listed with the crontab -l option. The -e option has to be used to create or edit a job and -r to remove a job

Syntax:

```
crontab [option] [username]
```

Practice 17.1

The following example shows how to edit a crontab file for a user to run "welcome" at midnight of every Sunday.

```
#  crontab –e kishore
```

```
0 0 0 * * banner welcome > /dev/console
```

This indicates that the command should be run on the zeroth minute of the zeroth hour (midnight) of the zeroth day (Sunday) of every week.

Once the crontab file has been set up (and that of the system's, if the user is the root user), there is no necessity to worry about it again, until the configuration has to be changed.

To list the current crontab entries the following command should be used:

```
# crontab –l  [username]
```

Practice 17.2

The following example illustrates how to list out the crontab entries for the user.

```
# crontab –l kishore
```

```
30 10 5 * *  rm /a*
```

The above example shows that at the thirtieth minute of the tenth hour of the fifth day of every month and every year, the files whose names begin with "a" under the root will be removed.

To remove the crontab entiries for a particular user the following should be used:

Consider the following example

```
# crontab –r kishore
```

The above command would remove the crontab entries for the user kishore.

Note : If **`crontab`** command is accidentally entered without any option, the interrupt character for the editor has to be pressed. This allows the user to quit without saving changes. Exiting the file and saving changes at this point would overwrite an existing **`crontab`** file with an empty file.

Controlling Access to crontab

Access to `crontab` can be controlled by using two files in the `/etc/cron.d` directory: `cron.deny` and `cron.allow`. These files permit only specified users to perform crontab tasks such as creating, editing, displaying, or removing their own crontab files.

The cron.deny and cron.allow files consist of a list of user names, one per line. These access control files work together as follows:

If `cron.allow` exists, only the users listed in this file can create, edit, display, or remove `crontab` files.

If `cron.allow` does not exist, all users may submit `crontab` files, except for users listed in `cron.deny`.

If neither `cron.allow` nor `cron.deny` exist, superuser privileges are required to run `crontab.`

Superuser privileges are required to edit or create `cron.deny` and `cron.allow`.

Note : In Solaris environment access to `crontab` can be controlled by using two files in the `/etc/cron.d` directory: `cron.deny` and `cron.allow`. These files permit only specified users to perform `crontab` tasks such as creating, editing, displaying, or removing their own `crontab` files.

Job Scheduling Using at

at

The "at" command can be used to schedule a task or job to run at the time specified on the command line or in a file.

atq

The "atq" command can be used to list the queue of waiting jobs. The "atq" command prints a list of all waiting jobs for the at command. These jobs can be found in the /var/spool/at directory.

atrm

The "atrm" command can be used to remove a specified job. The atrm command removes one or several jobs waiting in the at queue. The "atrm" command can be used by the users or by the root to delete the pending events.

Note : The output from the at command or script is important. The user should ensure to direct it to a file for later examination. In Solaris environment "at –r " command can be used to delete at jobs.

Practice 17.3

The following example shows the use of the at command.

```
[root@icg3 /etc]# at 11:45 pm
at> rm /home/kishore/*core*
at> <EOT>
```

```
warning: commands will be executed using /bin/sh
```

```
job 1 at 2001-05-26 23:45
[root@icg3 /etc]#
```

In the above example the `at` job removes `core` files from the user account belonging to kishore near midnight.

Creating an *at* Job

The at command allows the user to schedule a command for execution at a specified time, display a list of scheduled jobs and remove jobs from the scheduled jobs list. Jobs can be scheduled by specifying either the absolute time or a time relative to the current time.

Practice 17.4

…ving example shows how an at job can be created.

```
$ at -m 1930
at> rm /home/kishore/*.backup
at> Press Control-d
```

```
warning: commands will be executed using /bin/sh
job 5 at 2001-05-23 19:30
Send mail to the user
```

In the above example the user kishore has created an at job to remove his backup files at 7:30 p.m. The `-m` option enables the user to receive a mail message after the job completed. The user will receive a mail message when the job is completed, even if there is no output.

Controlling Access to `at`

A file can be set up to control access to the `at` command, permitting only specified users to create, remove, or display queue information about their `at` jobs. The file that controls access to `at,` is `/etc/at.deny.` The users listed in this file cannot access "`at`" commands. Root permissions are required to edit this file. Moreover, the at.allow file can be created in the /etc directory, in this case only users listed in the at.allow file can execute at command other than root.

The same files are available in the Solaris and SCO environments in the /etc/cron.d/ directory.

Displaying *at* Jobs

The at jobs can be displayed by using the following:

Syntax:

```
at -l [username]
```

Practice 17.5

The following example shows how to display information about the execution times of the `at` jobs for a user.

```
# at –l kishore
```

```
1 2001-05-24 09:45 a
2 2001-05-23 19:30 b
```

The above example shows the status information on all jobs submitted by the user kishore.

Linux does not return any message to indicate that the job has been cancelled, but the job will not be listed in the queue. Users can remove only their own jobs (root can remove any).

Some Linux versions support the atrm command as well as the -r option.

	LINUX	**SOLARIS**	**SCO**
Daemon	Crond	Crond	Crond
Command used for repetitive scheduling			
i) edit	Crontab –e	Crontab –e	Crontab –e
ii) list crontab entries	crontab –l	crontab –l	crontab –l
Location of crontab files	/var/spool/cron	/var/spool/cron/crontabs	
	/usr/spool/cron/crontabs		
Files to control	/etc/cron.deny	/etc/cron.allow	/etc/cron.d/cron.deny
	/etc/cron.d/cron.allow		/etc/cron.d/cron.deny
	/etc/cron.d/cron.allow		
One time scheduling	At	at	At
To list at jobs	at –l	at -l	at –l
To verify at commands waiting to execute	Atq	atq	Atq
To remove an at job	Atrm <job id>	at –r <job id>	atrm <job id>

Table 17.1

17.3 Short Summary

- Jobs can be scheduled to execute repetitively using the crontab command and at command is used to schedule a job to be executed once
- Crond is the daemon responsible for the execution of scheduled events
- The at command is used to schedule a task or job to run at a specified time
- Like the crontab command the access to the at command is controlled by the files at.allow and at.deny in /etc dir for Linux

17.4 Brain Storm

1. How does the crond daemon work?
2. What are the differences between cron.allow and cron.deny files?
3. How many ways are there to specify the time for the at command?
4. What is the purpose of –m option with at command?

Lecture 18

Performance Monitoring

Objectives

In this lecture you will learn the following

- Able to monitor the system performance
- Knowing Process Management Commands
- Knowing the tools to Monitor the System Performance in Sun Solaris

Coverage Plan

Lecture 18

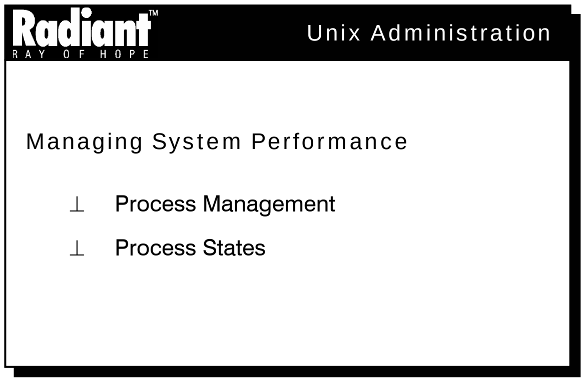
18.1 Snap Shot

The performance of a computer system depends upon how the system uses and allocates its resources.

System resources that affect performance include

- Central processing unit (CPU)
- Input/output (I/O) devices
- Memory

18.2 Managing System Performance



Radiant™ RAY OF HOPE Unix Administration

Managing System Performance

⊥ Process Management

⊥ Process States

The performance of a computer system depends upon how the system uses and allocates its resources. It is important to monitor the performance of the system regularly because this would enable to know about its behavioral pattern under normal conditions. The system administrator should essentially foresee and be able to recognize a problem when it occurs.

System resources that affect performance include

- Central processing unit (CPU) - The CPU processes instructions, receiving instructions from memory and executing them
- Input/output (I/O) devices - I/O devices transfer information into and out of the computer. Such a device could be a terminal, keyboard, a disk drive, or a printer
- Memory - Physical (or main) memory is the amount of memory (RAM) on the system

Process States

As soon as a process is created, the system assigns it a state. A process can be in one of several states. The state of the processes can be viewed on a system using the **ps(C)** command with the -el options. The "S" field displays the current state as a single letter.

The important states for performance tuning are:

O

On processor – This indicates that the processor is executing on the CPU in either user or system mode.

R

Runnable – This indicates that the process is on a run queue and is ready-to-run. A runnable process has every resource that it needs to execute except the CPU itself.

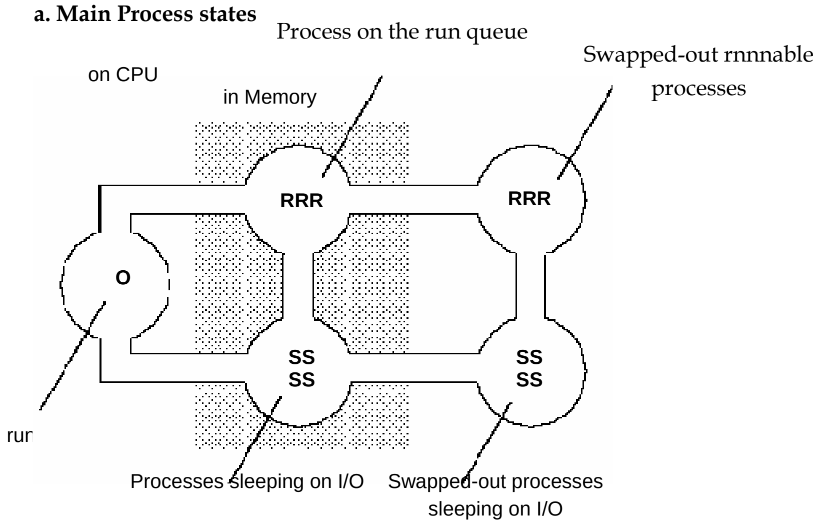
S

Sleeping – This indicates that the process is waiting for some I/O event to complete such as keyboard input or a disk transfer. Sleeping processes are not runnable until the I/O resource becomes available.

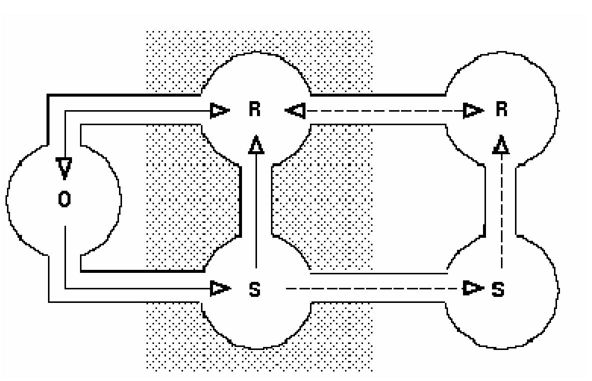
The figure 18.1 represents these states and the possible transitions between them.

On single CPU systems only one process can run on the CPU at a time. All other runnable processes have to wait on the run queue. A portion of the Kernel known as the scheduler chooses which process to run on the CPU(s). When the scheduler wants to run a different process on the CPU, the run queue is scanned from the highest priority to the lowest for the first runnable process. When a process becomes runnable, its priority is calculated by the Kernel and is placed on the run queue at that priority. While it remains runnable, the priority of the process is recalculated once every second, and its position in the run queue is adjusted. When there are no higher-priority runnable processes on the run queue, the process is placed on the CPU to run for a fixed amount of time known as a *time slice*.

a. Main Process states


Process on the run queue
Swapped-out rnnnable processes
on CPU
in Memory
RRR
RRR
O
SS SS
SS SS
rur
Processes sleeping on I/O
Swapped-out processes sleeping on I/O

b. Transistions between process states

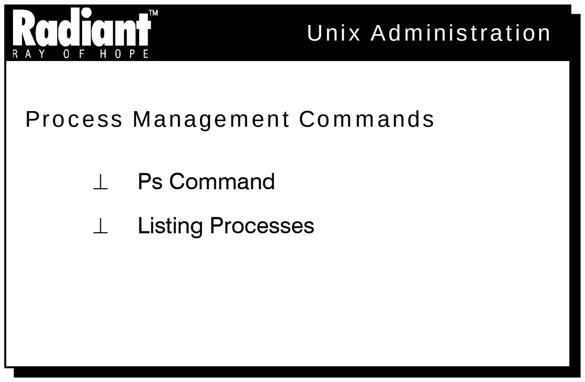


──▶ Main Flow

............▶ Swapping

Fig 18.1

18.3 Process Management Commands



Radiant™ RAY OF HOPE

Unix Administration

Process Management Commands

⊥ Ps Command

⊥ Listing Processes

This section describes the commands that are used to manage process information.

ps Command

The ps command enables to check the status of active processes on a system, as well as display technical information about the processes. This data is useful for administrative tasks such as determining how to set process priorities.

Depending on the option that is chosen, ps reports the following information:

- Current status of the process
- Process ID
- Parent process ID
- User ID
- Scheduling class
- Priority
- Address of the process
- Memory used
- CPU time used

Listing Processes

The ps command can be used to list all the processes being executed on a system.

```
$ ps [-ef]
      -e    Selects all processes.
      -f    Does full listing.
```

ps	displays only the processes associated with the user's login session.
-ef	displays full information about all the processes being executed on the system.

Practice 18.1

The following example shows the output from the ps command when no options are used.

```
#ps
```

```
PID   TTY      TIME        CMD
859   pts/1    00:00:00    login
883   pts/1    00:00:00    su
884   pts/1                00:00:00 bash
918   pts/1                00:00:00 ps
```

Here, PID refers to the process ID. TTY refers to the terminal from which the process (or its parent) is started. A question mark indicates that there is no controlling terminal. TIME indicates the total amount of CPU time used by the process since it began. COMMAND shows the command that has generated the process.

Practice 18.2

The following example shows the output from ps –ef. Only part of the output is given below.

```
# ps –ef

UID      PID      PPID   C    STIME      TTY    TIME      CMD
root     1        0      2    05:41      ?      00:00:04  init
root     2        1      0    05:41      ?\     00:00:00  [kflushd]
root     3        1      0    05:41      ?      00:00:00[kupdate]
root     4        1      0    05:41      ?      00:00:00  [kpiod]
root     5        1      0    05:41      ?      00:00:00  [kswapd]
root     6        1      0    05:41      ?      00:00:00  [mdrecoveryd]
bin      330      1      0    05:41      ?      00:00:00  portmap
root     345      1      0    05:41      ?      00:00:00  [lockd]
root     346      345    0    05:41      ?      00:00:00  [rpciod]
root     355      1      0    05:41      ?      00:00:00  rpc.statd
root     449      1      0    05:41      ?      00:00:00  syslogd -m 0
root     458      1      0    05:41      ?      00:00:00  klogd
nobody   472      1      0    05:41      ?      00:00:00  identd -e –o
nobody   475      472    0    05:41      ?      00:00:00  identd -e –o
nobody   476      475    0    05:41      ?      00:00:00  identd -e –o
nobody   478      475    0    05:41      ?      00:00:00  identd -e –o
nobody   479      475    0    05:41      ?      00:00:00  identd -e –o
daemon   490      1      0    05:41      ?      00:00:00  /usr/sbin/atd
root     504      1      0    05:41      ?      00:00:00  crond
root     522      1      0    05:41      ?      00:00:00  inetd
root     536      1      0    05:41      ?      00:00:00  lpd
```

Here UID refers to the effective user ID of the owner of the process. PPID indicates the ID of the parent process. C shows the processor utilization for scheduling. This field is not displayed when the –c option is used. STIME indicates the starting time of the process (in hours, minutes, and seconds). TIME indicates the total CPU time that has been used by the process since it began. CMD indicates the command that has generated the process.

Displaying Information about Processes

The output from the ps command can be used (optional) to obtain the identification number of the process about which more information needs to be displayed. This can be done as follows:

```
# ps -e | grep process
```

Here process indicates the name of the process about which more information needs to be displayed.

Practice 18.3

The following example shows how to display information about a process.

```
# ps -e|grep init
1 ?        00:00:04 init
```

Here first field represents the process-id(PID), the second field shows the terminal type (TTY), then the third field shows the total amount of CPU time that has been used by the process since it began (TIME), and the final field shows the command that has generated the process (CMD).

Displaying the Global Priority of a Process

The global priority of a process can be displayed by using the ps command as shown below.

```
$ ps –ecl
```

-e	Selects all processes.
-c	Selects by command name.
-l	Long format.

Practice 18.4

The following shows how to display the global priority of a process.

```
# ps –ecl
```

```
  F    S    UID    PID    PPI    CLS  PRI  ADDR   SZ     WCHAN     TTY     TIME       CMD
100    S    0      1      0      -     9    -     280    do_sel    ?       00:00:04   init
040    S    0      2      1      -     39   -     0      bdflus    ?       00:00:00   flushed
040    S    0      3      1      -    39   -     0      kupdat    ?       00:00:00   0
040    S   0      4      1      -     39   -     0      kpiod     ?       00:00:00   kpiod
040    S    0      5      1      -    39   -     0      kswapd    ?       00:00:00   kswapd
040    S    0      6      1      -    59   -     0      md_thr    ?       00:00:00  drecovery
140    S    1     330     1      -    39   -     303    do_sel    ?       00:00:00   portmap
040    S    0      345    1      -    39   -     0      end       ?       00:00:00   lockd
040    S    0      346    345    -    39   -     0      end    ?          00:00:00   rpciod
140    S    0      355     1     -    39   -     289    do_sel ?          00:00:00   pc.statd
140    S    0      369    1      -    39   -     276    do_sel ?          00:00:00   apmd
040    S    0      396    1      -    39   -     302    pipe_r ?          00:00:00 automount
140    S    0      449    1      -    39   -     293    do_sel ?          00:00:00 syslogd
140    S    0      458     1     -    39   -     357    do_sys ?          00:00:00   klogd
140    S    99     472    1      -    39   -     323    wait_f ?          00:00:00   identd
```

```
040   S   99    475   472    -   39   -     323   do_pol ?  00:00:00          identd
040   S    99   476   475    -   39   -     323   rt_sig ?         00:00:00  identd
040   S   99    478   475    -   39   -     323   rt_sig    ?      00:00:00  identd
040   S   99    479   475    -   39   -     323   rt_sig    ?      00:00:00  identd
040   S   2     490    1     -   39   -     286   nanosl    ?      00:00:00  atd
040   S   0     504    1     -   39   -     332   nanosl    ?      00:00:00  crond

140   S   0     522   1      -   39   -     286   do_sel    ?      00:00:00   inetd
140   S   0     536    1     -   39   -     301   do_sel    ?      00:00:00  lpd
140   S   0     584    1     -   39   -     532   do_sel    ?        00:00:00  sendmail
140   S   0     599    1     -   39   -     288   do_sel ttyS0     00:00:00   gpm
040   S   43    683    1     -   39   -     926   do_sel    ?      00:00:00   xfs
100   S   0     723   1      -   39   -     556   wait4  tty1      00:00:00   login
100   S   0     724   1      -   39   -     273   read_c tty2      00:00:00   mingetty
100   S   0     725   1      -   39    -    273   read_c tty3      00:00:00   mingetty
100   S   0     726   1      -   39    -    273   read_c tty4      00:00:00  mingetty
100   S   0     727    1     -   39    -    273   read_c tty5      00:00:00  mingetty
100   S   0     728   1      -   39   -     273   read_c tty6      00:00:00   mingetty
100   S   0     729    1     -   39    -    686   do_pol    ?      00:00:00   gdm
100   S   0     739   729     -  39    -    3930  do_sel    ?      00:00:01   X
140   S   0     740   729     -  39    -    856   wait4     ?      00:00:00   gdm
100   S   0     752   740    -   39   -     1405  do_pol    ?      00:00:00 gnome-sess
100   S   101   878   877     -  37   -     423   wait4 pts/0      00:00:00  sh
000   S   0     895   878     -  37   -     539   wait4 pts/0      00:00:00   su
100   S   0     896   895    -   29   -     432   wait4 pts/0      00:00:00   bash
100   R   0     909   896    -   28   -     636   -     pts/0      00:00:00   ps
```

CLS represents the scheduling class to which the process belongs: real-time, system, or timesharing. This field is included only with the -c option. PRI means the Kernel thread's scheduling priority. Higher numbers mean higher priority. ADDR indicates the address of the proc structure. SZ shows the virtual address size of the process. WCHAN indicates the address of an event or lock for which the process is sleeping. Data in the PRI column show that kflushd has the highest priority, while sh has the lowest. This output clearly shows that the system processes will always have higher priority than user processes.

Commands for Managing Processes

Processes can be managed using the ps command and the nice command.

The ps command can be used to check the active processes on a system and to display detailed information about the processes. The nice command is used to change the priority of a timesharing process.

Nice executes a command with a lower CPU scheduling priority. It is used if the increment argument (in the range 1-20) is given; if not, an increment of 10 is assumed. The invoking process (generally the user's shell) must be in the time-sharing scheduling class. The super-user may run commands with priority which is higher than normal by using a negative increment. For example, —10.

Finding the Load Average

The uptime command can be used to find the load average. This command tells how long the system has been running.

It gives a one line display of the following information: the current time, the duration for which the system has been running, the number of users who are currently logged on and the system load averages for the past 1, 5, and 15 minutes. -V indicates the version number of the command.

Practice 18.5

The following example shows the usage of the uptime command.

```
# uptime
```

```
2:06pm  up 1 min,  1 user,  load average: 0.62, 0.20, 0.05
0.62 for 1minutes
0.20 for  5 minutes
0.05 for 15 minutes
```

The output indicates the current time. It shows that there is one user and that the system has been up for 1 minute. It also gives the load averages for the past 1, 5 and 15 minutes respectively. The optimal load average values are 1 for 1 minute, 2 for 5 minutes and 10 for 15 minutes. The load average has to be checked. If this load average is abnormal then different areas in the system such as disk , CPU and RAM need to be checked.

Monitoring the Virtual Memory Statistics

The umstat command is used to monitor virtual memory statistics.

Practice 18.6

The following example shows the usage of the vmstat commands.

```
# vmstat
```

```
   procs      memory   swap      io        system        cpu
r  b    w  swpd  free si  so    bi  bo   in    cs  us    sy     id
0  0    0  680   1544 1   1     35  96   418   27  1     4      96
```

Here, under procs there are 3 fields. i.e. the number of processes waiting for run time (r), the number of processes in uninterruptable sleep(b) and the number of processes swapped out but otherwise runnable (w), the memory field reports on usage of real and virtual memory. Here there are 3 fields: The amount of virtual memory used in kB (swpd), the amount of idle memory in kB (free). and the amount of memory used as buffers in kB (buff). Under Swap we are having 3 fields: Amount of memory swapped in from disk in kB/s (si), amount of memory swapped to disk in kB/s (so) and the blocks sent to a block device in blocks/s (IO), the blocks received from a block device in blocks/s (bo). Under system there are 2 fields: The number of interrupts per second, including the clock (in), and the number of context switches per second (cs), and these are percentages of total CPU time, user time(us),Under cpu there are 2 fields: system time (sy) and idle time (id).

18.4 Tools to Monitor the System Performance in Sun Solaris

System Activity Report (SAR)

SAR, in the first instance, samples cumulative activity counters in the operating system at n intervals of t seconds, where t should be 5 or greater. If the -o option is specified, it saves the samples in the file in binary format. The default value of n is 1. In the second instance, with no sampling interval specified, SAR extracts data from a previously recorded file, either the one specified by -f option or, by default, the standard system activity daily data file /var/adm/sa/sadd for the current day dd. The starting and ending times of the report can be bound through the -s and -e time arguments of the form hh[:mm[:ss]]. The -i option selects records at second intervals. Otherwise, all the intervals found in the data file are reported.

In either case, subsets of data to be printed are specified by the following options:

-a	Reports use of file access system routines.
-b	Reports buffer activity.
-c	Reports system calls.
-d	Reports activity for each block device, that is, disk drives.
-g	Reports graphics activity.
-m	Reports message and semaphore activities.
-p	Reports paging activities.
-q	Reports average queue length while occupied, and % of time occupied.
-r	Reports unused memory pages and disk blocks.
-u	Reports CPU utilization (the default).
-v	Reports status of process, i-node, file tables and record lock tables.
-w	Reports system swapping and switching activity.
-y	Reports TTY device activity.

Practice 18.7

The following example is an illustration of the SAR command.

```
# SAR -a 22
17:56:20  iget/s namei/s dirbk/s (-a)
17:56:22       0       0       0
17:56:24       4       1       0
Average        2       1       0
```

Here 2 samples are taken at an interval of two seconds.

iget/s indicates the number of requests made for the inodes that were not in the directory name lookup cache(dnlc). namei/s indicates the number of file system path searches per second. dirbk/s indicates this is the number of directory block reads issued per second.

Practice 18.8

The following example shows how to check buffer activity using SAR

```
# SAR-b 2 2
17:56:37 bread/s lread/s %rcache bwrit/s lwrit/s %wcache pread/s pwrit/s(-b)
17:56:39     0     2    100       8       2       0       0     0
17:56:41     2     0      0       0       0       0       0     0
Average      1     1     20       4       1       0       0     0
```

Here bread/s indicates the average number of reads per second submitted to the buffer cache from the disk. lread/s indicates average number of logical reads per second from the buffer cache. %rcache indicates the fraction of logical writes found in the buffer cache (100% minus the ratio of bread/s to lread/s). bwrite average number of physical blocks (512 blocks) written from the buffer cache to disk, per second. lwrite/s indicates the average number of logical writes to the buffer cache, per second. %wcache indicates the fraction of logical writes found in the buffer cache (100% minus the ratio of bwrite/s to lwrite/s). pread/s indicates the average number of physical reads, per second, using character device interface. pwrite/s indicates the average number of physical write requests, per second, using character device interfaces.

Practice 18.9

le illustrates how to display disk activity statistics with the SAR –d command

```
# SAR -d 22
17:57:14  device %busy  avque   r+w/s   blks/s  avwait  avserv (-d)
17:57:16
17:57:18  wd-0    0.50  1.00    1.00    1.99    0.00    5.00
Average  wd-0    0.25   1.00    0.50    1.00    0.00    5.00
```

Here, devices represents the name of the disk device being mounted. %busy indicates the percentage of time the device has spent in servicing a tranfer request. avque indicates the sum of the average wait time plus average service time. r+w/s shows the number of read and write transfers to the device per second. blks/s indicates the number of 512-byte blocks transferred to the device per second. avwait indicates the average time in milliseconds that transfer to the device per second. avserv indicates the average time in milliseconds, for a transfer request to be completed by the device (for disk). pwrite/s shows the average number of physical write requests, per second, using character device interfaces.

Practice 18.10

The following example shows how to display Inter-process communication activities using SAR -m

```
# SAR -m 2 2
17:58:37   msg/s  sema/s (-m)
17:58:39    0.00    0.00
17:58:41    0.00    0.00
Average     0.00    0.00
```

Here, msg/s represents the number of message operations (send and receives) per second., sema/s shows the number of semaphore operations per second.

Practice 18.11

The following example illustrates how to monitor unused memory.

```
# SAR -r 2 2

17:59:24 freemem freeswp (-r)
17:59:26   25807   96000
17:59:28   25830   96000
Average    25818   96000
```

Here, Freemem refers to the free ram space. Freeswp indicates the free swap space.

Checking CPU Utilization

Note : The SAR command without any options is equivalent to SAR (-u).

The processor might be either busy or idle. When the processor is busy it is either in user mode or system mode. When idle, it is either waits for I/O completion or is inactive.

Practice 18.12

The following example shows how to check if the processor is busy or idle.

```
# SAR -u 2 2
17:59:40  %usr    %sys    %wio   %idle (-u)
17:59:42   0       0        0      100
17:59:44   0       0        0      100
Average    0       0        0      100
```

Here, %sys represents the percentage of time that the processor is in system mode. %user represents the percentage of time the processor is in user mode. %wio indicates the percentage of time the processor is idle and waiting for I/O completion. %idle shows the percentage of time the processor is idle and is not waiting for I/O. SAR reports may indicate that the processor takes more time in processing the user requests. There might be some programs that use many unwanted system calls, that is, some shell scripts that might refer unwanted files which results in unwanted system calls.

18.5 Short Summary

- A process is defined as an instance of a program in execution
- The important states for performance tuning are O – On-proccesor, R- Runnable, and S- Sleeping
- The ps command gives a list of all active processes
- The information about all active processes are found in the /proc file system
- The commonly used commands to monitor system performance are *SAR,* (in Solaris) *vmstat*

18.6 Brain Storm

1. What are the two types of memory?
2. Explain how optimization of the system performance is done.
3. Explain the *ps* command.
4. Explain the *vmstat* command.

Lecture 19

Network Management

Objectives

In this lecture you will learn the following

- Knowing about LAN Fundamentals
- Understanding the concept of LAN Interconnection
- About Reference Model
- Able to test the TCP/IP using IPCONFIG and PING

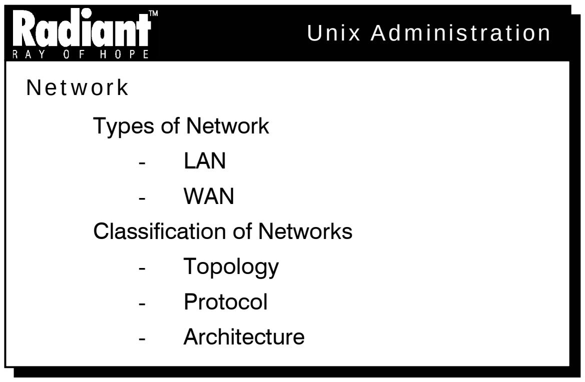
Coverage Plan

Lecture 19

19.1 Snap Shot

Networking is establishing, maintaining and utilizing a broad network of contacts (e.g., in government, media, and the community) in order to keep a pulse on public, political and internal issues and to make informed decisions. It includes identifying who to involve, as well as when and how to involve them in order to accomplish objectives and minimize obstacles.

19.2 Network



Radiant™ RAY OF HOPE — Unix Administration

Network

Types of Network
- LAN
- WAN

Classification of Networks
- Topology
- Protocol
- Architecture

A network is a group of two or more computers that are linked together to share the resources like printers, data, disk drives, cdrom drives and tapedrive.

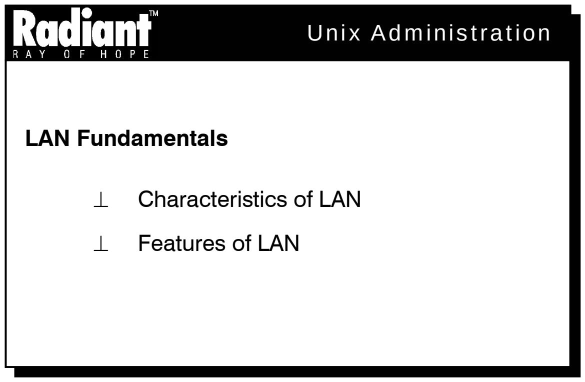
Types of Network

Many types of network exist, but the most common types of networks are Local-Area Networks (LANs) and Wide-Area Networks (WANs). In a LAN, computers are connected together within a "local" area (for example, an office or home). In a WAN, computers are geographically distributed and are connected through telephone/communication lines, radio waves, or other means of connection.

Classification of Networks

Networks are usually classified using three properties: Topology, Protocol, and Architecture. Topology specifies the geometric arrangement of the network. Common topologies are bus, ring, and star. Protocol specifies a common set of rules and signals, which the computers on the network use to communicate. Most networks use Ethernet, but some networks may use IBM's Token Ring protocol. Architecture refers to one of the two major types of network architecture: Peer-to-Peer or client/server. In a Peer-to-Peer networking configuration, computers simply connect with each other in a workgroup to share files, printers, and Internet access. This is most commonly found in home configurations, and is only practical for workgroups of a dozen or less computers. In a client/server network, an NT Domain Controller, is usually present to which all the computers log on to. This server can provide various services, including centrally routed Internet access, mail (including e-mail), file sharing, and printer access, as well as ensuring security across the network. This is most commonly found in corporate configurations, where network security is essential.

19.3 LAN Fundamentals


Unix Administration

LAN Fundamentals

- ⊥ Characteristics of LAN
- ⊥ Features of LAN

This section provides the background required to understand LAN products:

Characteristics of LAN

A local area network (LAN) consists of hardware and software that provide the capability to interconnect a variety of data-communicating devices within a limited area. LANs have become an integral part of computing and data communication. Today, most computer manufacturers provide some means for connecting their equipment to a LAN. It is expected that, in the future, installation and use of LANs will increase dramatically.

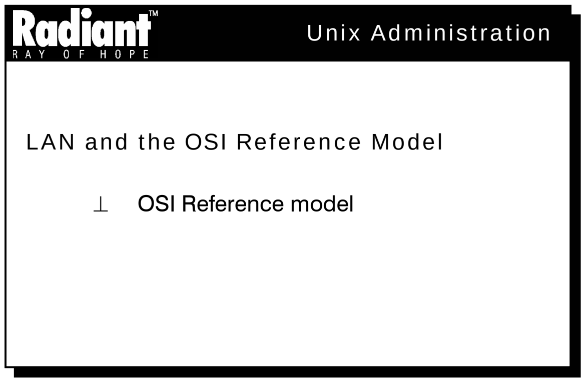
Users connected to a LAN can share devices such as processors, printers, and disk storage. The most efficient use of each device can be achieved by allowing access to it by multiple users; the need to obtain individual devices for each user is eliminated. For example, through a LAN, several workstations can access a single disk.

While networks as a whole permit a variety of intelligent devices to exchange data, several important features distinguish a LAN from a wide area network (WAN).

Features of LAN

- **Service to a limited area.** LANs provide network services to a limited area such as a single building or a group of buildings known as a campus, whereas WANs may provide national or international service
- **Private ownership and administration.** Unlike wide area networks, which may be subject to the rules of a regulatory agency such as the Federal Communications Commission in the United States or the Office of Telecommunications (OFTEL) in Great Britain, LANs are owned and administered privately. This means a LAN can be designed, installed, maintained, and expanded without external involvement or approval
- **Complete device connectivity.** Each device that is connected to a LAN potentially has equal access to every other network device
- **Self-contained.** LANs are self-contained entities that can be constructed with relatively fewer equipments. The geographic coverage that a wide area network provides requires a large array of telecommunication equipments (telephone lines, satellites, and microwave links, to name a few). LANs often connect to a wide area network through a device known as a *gateway*, but the LAN is always a distinct and independent entity

19.4 LANs and the OSI Reference Model


Radiant™ RAY OF HOPE Unix Administration

LAN and the OSI Reference Model

⊥ OSI Reference model

The International Standards Organization (ISO) has defined an architecture that describes how devices can communicate over a network. The architecture, referred to as the Open Systems Interconnection (OSI) Reference Model, divides the communication functions into seven distinct and independent layers. Figure 17.1 shows the OSI model.

OSI reference model

Each layer receives services from the underlying layer, the service provider, through service access points (SAPs). A layer may have several SAPs with each SAP having its own address. Software or hardware entities implement the services defined by each layer. Essentially, communication takes place between peer layers on each machine. Protocols specify how the communication takes place between the peer layers.

Application

The application layer defines how a user or process accesses the network. File transfer and mail facilities are the typical examples of application-layer processes.

Presentation

The presentation layer defines the format, including syntax and semantics of the data exchanged between devices. The application layer may deliver the data in a particular format (ASCII, for example) to the presentation layer, where it is transformed into a format suitable for transmission over the network.

Session

The session layer defines how communication (known as a session) is established between machines. The services in this layer also manage different aspects of the session such as synchronization and traffic direction.

Transport

The transport layer defines the connection between the source and destination machines (end-to- end). It ensures that the data from upper-layer processes, usually encapsulated in packets, arrives at its destination error-free and in the correct order.

The following paragraphs define the services provided by each layer of the OSI model:

OS Reference Model

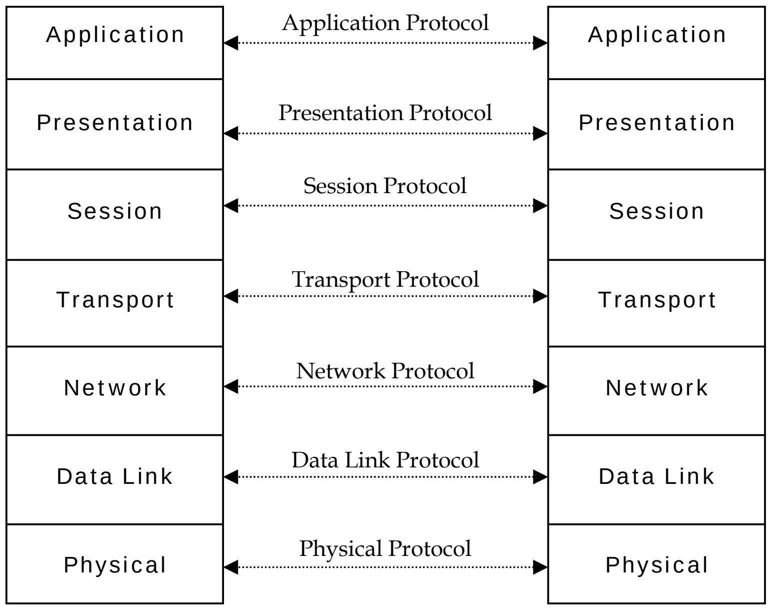


Figure 19.1

Network

The network layer defines how the packets are routed through the network to their final destination. This layer isolates the upper layers from the underlying aspects of the network.

Data Link

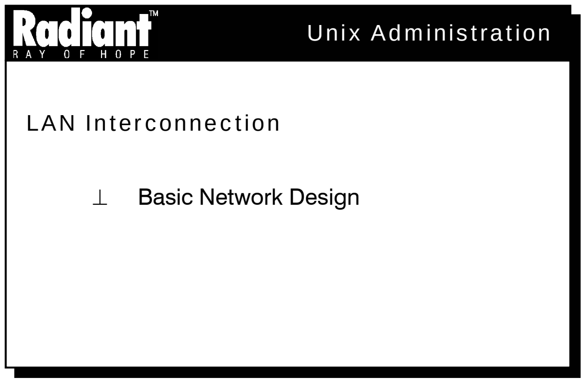
The data link layer defines how blocks of data (packets) are reliably transferred between adjacent network nodes. It is also responsible for error detection and flow control.

Physical

The physical layer defines how the raw bits of upper-level data are physically transferred over the network medium (Physical cable). This layer concerns itself with the electrical and mechanical aspects of data transmission.

As shown in Figure 17.1, LANs implement the services defined by the physical and data link layers of the OSI reference model. Additional communications standards, such as the BSD/DOD application utilities and the TCP/IP protocols, must be implemented in some manner to achieve the services defined by the upper layers of the OSI reference model.

19.5 LAN Interconnection



Unix Administration

LAN Interconnection

⊥ Basic Network Design

The existence of separate LANs within an organization can be necessitated by a number of factors (such as too many users for one network to support, too large a geographic distance to cover, different network needs of various work groups, or a need for special network security). Organizations containing separate LANs often need to pass data back and forth between them. The generic term "relay" refers to the equipment that connects LANs. Relays are usually implemented in the lower three layers of the OSI reference model.

At the Physical Layer (Layer 1), relays are repeaters that copy signals from one LAN segment to another. Repeaters extend LANs, thus inexpensively solving distance problems at a single location.

At the Data Link Layer (Layer 2), relays are bridges that store and forward data packets. LANs use 48-bit source and destination addresses that are administered by the Institute of Electrical and Electronic Engineers (IEEE) so that each system connected to a LAN has a globally unique source address. A bridge examines the source and destination addresses in each data packet transmitted over the LANs to which it is connected. It constructs (and frequently updates) an address table that associates each source address with the LAN on which it is seen. Each packet's destination address is compared to the source-address table. If the address is found to be associated with the LAN on which the packet was found, the packet is filtered (discarded). Otherwise, the packet is forwarded. Some bridges can connect different LAN types and are therefore called translation bridges.

Bridges have a number of advantages as a means of connecting LANs, the biggest advantage is that they do not require end-user systems to execute any special routing protocols or services. Consequently, bridges are said to be transparent. By filtering out some data packets, bridges can significantly reduce traffic on associated LANs, thus increasing LAN throughput. Many bridges allow network managers to further optimize, or increase the security of their networks by providing additional filters that discard specified types of packets or packets that have particular destination addresses.

The greatest disadvantage of connecting LANs with a bridge is that, when filtering is inadequate, the bridge can saturate the network with propagated packets, thus degrading network performance. However, bridges are a low-cost, high-performance means of connecting LANs if the networks contain a limited number of LAN nodes that are not spread over too wide an area.

At the Network Layer (Layer 3), relays are either gateways or routers. A router stores and forwards data packets that are directly addressed to it by the user systems. For a router and the source and destination systems must adhere to a common routing protocol. Unlike bridges, routers require the active participation of all end users.

Basic Network Design

The most common form of network is Ethernet. This is a bus-like network that uses Carrier-Sense Multiple Access with Collision Detection (CMSA-CD). In this network stations apply a voltage to the bus when they wish to send data, by sensing the bus for this voltage it is determined if the bus is in use. Multiple access implies that many hosts may be on this bus. Collision detect is used to detect multiple hosts sending data at the same time. Initially, it would seem unnecessary to need collision detection, after all, a station sends data on the bus when nothing is being sent. Due to the propagation delay of electrical signals we can have stations that decide to send data at the same time. When each station looks at the bus it is clear. However, before the data they send reaches its destination they will collide and the result will be garbage. The collision detection circuitry monitors the line to verify if there were no collisions and the data does not need to be re-sent.

Understanding the CMSA-CD concept is fundamental to understanding how ethernet works. All limitations found on the design of ethernet networks are due to issues surrounding CMSA-CD. The biggest design limitation is that reading data on an ethernet is a passive operation, the sending stations have no way to “sense” when this has happened. However, the sending station must perform collision detection until it knows the receiving station has received the packet! To perform this task, lengthy restrictions must be developed so that a sending station knows that within a finite time the receiving stations must have received the packet. This limitation controls most of the aspects of network design.

Ethernet Hardware

Ethernet has evolved over time. Ethernet version 2 released in 1982 was originally developed by Xerox-Intel-Dec. In 1985 the IEEE released a new standard for ethernet. This standard is named IEEE 802.2. In general, these two versions of ethernet can inter-operate, however there are a few minor differences. The first difference is that in the ethernet packet header Version 2 defined a two byte Type field while IEEE created a 2 byte length field in that location. Luckily, values for type cannot conflict with valid length values and most systems can determine the Ethernet Frame type by examining this field. A second difference is that the Ethernet version 2 specification required that a transceiver send a heartbeat signal each second. The IEEE 802.2 specification removed this. This has resulted in most vendors offering transceivers that have a switch to enable or disable heart-beat. It should be off unless connected to a piece of equipment using the ethernet version 2 specification. Fortunately, all new devices are built to conform to the 802.2 specification. However, occasionally devices are found that were installed years ago that still need this.

In either specification, ethernet uses a 48-bit identifier to uniquely identify each source and destination device. A range of addresses is assigned to each manufacturer of ethernet equipment.

There are basically two categories of ethernet components, one type that passes the signal onto other devices, generally these are known as repeaters. A second type of device which takes the signal and regenerates the signal onto a new network, these types of devices are generally known as bridges or routers. Repeaters are useful for propagating a network signal, a signal comes in on an input port is often output to many ports. However, since they add some delay to the transmittal of packets they reduce the maximum size of a segment. However, repeaters can simplify the design of a network.

Devices such as bridges and routers, which regenerate the signal, allow to build larger networks. Since the signal is regenerated, it becomes the responsibility of the bridge or router to guarantee the packets arrival at the destination (or the next router or bridge). Bridges and routers work at different levels of the network. Bridges work at the ethernet frame level while routers work at the protocol level. In both cases, the bridge or router, have the property of filtering traffic and only transmitting the signal onto networks. Thus, in each case they have the effect of reducing unnecessary traffic.

Types of Media used with Ethernet

The IEEE 802.2 specification defines the general properties of ethernet. Subsequent standards define how each media type will operate. At present, ethernet can be run over voice grade twisted pair (10BASE-T), thin-wire coaxial cable (10Base-2), thick-wire coaxial cable (10Base-5), and fiber optic cable (10Base-F). The overwhelming majority of connections made today use twisted-pair wiring. This option is now offered as standard equipment on many workstation models.

Each media type has different signal properties and limits. For example, (10BASE-T) only supports one machine per segment and is limited in distance to 100 meters. Thinwire (10BASE-2) can support up to 29 stations and is limited to a maximum distance of 185 meters. Fiber optic cabling can support 1024 devices and can operate at distances up to 2 Kilometers. Thickcoaxial cable (10BASE-5) can operate up to 500 meters and support up to 1024 stations.

Trancievers often allow dissimilar devices to be attached together. Many machines have a 15 pin Ethernet AUI interface. Tranceivers exist which allow the users to adapt the AUI interface to whatever media you have running to the desktop.

Designing Ethernet Networks

The goal of designing networks is to maximize reliability while minimizing cost. These are usually conflicting goals and hence tradeoffs must be made. The user can try to follow these guidelines:

- Use twisted pair connections for all desktop connections. This is cost effective and provides an easy way to troubleshoot problems
- Build networks such that wherever possible servers and clients are on the same network
- Use routers to build enterprise networks. Routers are more effective at isolating and controlling traffic among networks. Use bridges to seperate traffic within a network
- Adopt the Simple Network Management Protocol (SNMP) as a management standard and only purchase equipment supporting that standard
- It is wise to purchase machines with an AUI interface and then use transceivers to connect the machine to whatever media you have

Ensure and follow the design limitations for each media type that is being used before designing networks. The ethernet standard is conservative by nature and the user should strictly adhere to the standards.

Often referred to as “Thicknet,” 10Base5 technology was the first incarnation of Ethernet. It was used in the 1980s until 10Base2 “Thinnet” with more flexible cabling appeared. (At five millimeters, Thinnet is one-half the thickness of Thicknet.) The most common form of traditional Ethernet, however, is 10Base-T due to the inherent advantages of unshielded twisted pair (UTP) over coaxial cabling and its low cost compared to alternatives like fiber.

The following table lists these well-known forms of Ethernet technology. Besides the type of cable involved, another important factor in Ethernet networking is the *segment length*. A single uninterrupted network cable can only span a certain physical distance before its electrical characteristics are critically affected by factors such as line noise or reduced signal strength.

Name	**Segment Length (Max.)**	**Cable**
10Base5	500m / 1640ft.	RG-8 or RG-11 coaxial
10Base2	185m / 606ft.	RG 58 A/U or RG 58 C/U coaxial

10Base-T	100m / 328ft.	Category 3 or better unshielded twisted pair

Table 19.1

Several other less well-known Ethernet standards exist, including 10Base-FL, 10Base-FB, and 10Base-FP for fiber optic networks and 10Broad36 for broadband (CATV) cabling.

Fast Ethernet

In the mid-1990s, Fast Ethernet achieved its design goal of increasing the performance of traditional Ethernet while avoiding the need to completely re-cable existing networks. Fast Ethernet comes in two major varieties:

- 100Base-T (using unshielded twisted pair cable)
- 100Base-FX (using fiber optic cable)

By far the most popular of these is 100Base-T, a standard that includes *100Base-TX* (Category 5 UTP), *100Base-T2* (Category 3 or better UTP), and *100Base-T4* (100Base-T2 cabling modified to include two additional wire pairs).

Physical Devices

Standard Hubs

The criteria to be ensured before purchasing a standard hub is that it should be capable of handling network expansions

Standalone Hubs

Standalone hubs are single products with a number of ports. Standalone hubs usually include methods for linking them to other standalone hubs for network expansion. Standalone hubs are usually the least expensive type of hub and are best suited for small, independent workgroups, departments, or offices. Standalone hubs usually include methods for linking them to other standalone hubs for network expansion. typically with fewer than 12 users per LAN.

Stackable Hubs

Stackable hubs work just like standalone hubs, except that several of them can be “stacked” (connected) together, usually by short lengths of cable. When they are connected together, they act like a modular hub, because they can be managed as a single unit. These hubs are ideal want to start with a minimal investment, but it should be realized the LAN will grow.

Modular Hubs

Modular hubs are popular in networks because they are easily expanded and always have a management option. A modular hub is purchased as a chassis, or card cage, with multiple card slots, each of which accepts a communications card, or module. Each module acts like a standalone hub and usually has 12 twisted pair ports. Modules supporting different types of network cabling, like coaxial or token ring, can also be purchased.

For a home or small office network, it is preferable to purchase a standalone or stackable hub. For a medium to large sized company, a modular hub will probably fit the needs more efficiently. The other types of network hardware are discussed below.

Bridges, Routers and Switches

Bridges and routers are devices used for linking different LANs or LAN segments together. There are many companies that have LANs at various offices across the world. Routers were originally developed to allow connection of remote LANs across a wide area network (WAN). Bridges can also be used for this purpose. By setting up routers or bridges on two different LANs and connecting them together, a user on one LAN can access resources on the other LAN as if they were on the local LAN.

Bridges

Bridges are simpler and less expensive than routers. Bridges make a simple do/don't decision on which packets to send across two segments they connect. Filtering is done based on the destination address of the packet. If a packet's destination is a station on the same segment where it originated, it is not forwarded. If it is destined for a station on another LAN, it is connected to a different bridge port and forwarded to that port.

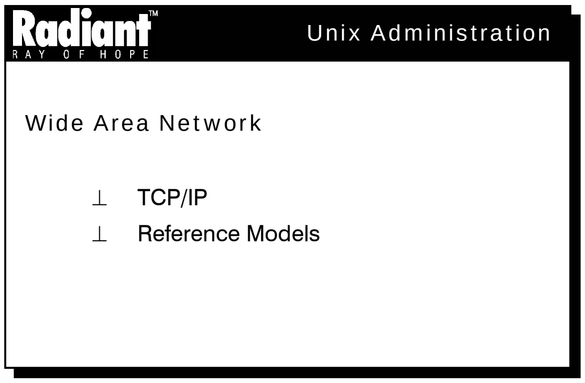
Routers

Routers are more complex and expensive than bridges. They use information within each packet to route it from one LAN to another, communicate with each other and share information that allows them to determine the best route through a complex network of many LANs.

Switches

Switches are another type of device used to link several LANs and route packets between them. A switch has multiple ports, each of which can support either a single station or an entire Ethernet or Token Ring LAN. With a different LAN connected to each of the ports, it can switch packets between LANs as needed.

19.6 Wide Area Network



Radiant™ RAY OF HOPE — Unix Administration

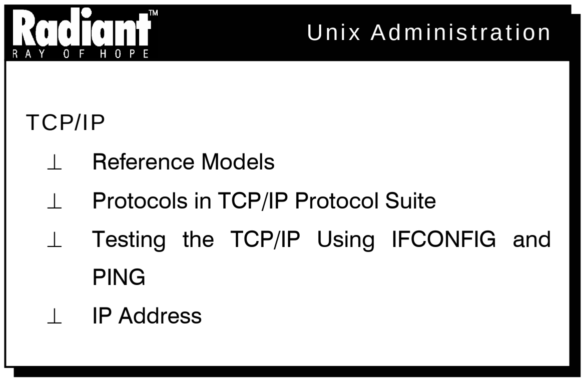
Wide Area Network

- TCP/IP
- Reference Models

As the term implies, a wide-area network spans a large physical distance. A WAN, like the Internet spans most of the world. A WAN is a geographically distributed collection of LANs. A network device called a router connects LANs to a WAN. In IP networking, the routers maintain both the addresses of LAN and WAN.

WANs differ from LANs in several important ways. Like the Internet, most WANs are not owned by any one organization but rather exist under collective or distributed ownership and management. WANs use technology like ATM, Frame Relay and X.25 for connectivity.

19.7 TCP/IP - Transmission Control Protocol/Internet Protocol



Radiant™ RAY OF HOPE Unix Administration

TCP/IP

- ⊥ Reference Models
- ⊥ Protocols in TCP/IP Protocol Suite
- ⊥ Testing the TCP/IP Using IFCONFIG and PING
- ⊥ IP Address

It is a communication protocol that can be used by an application to package its information for sending across a network or networks. It is also referred as a protocol suite.

An entire collection of protocols is termed as a protocol suite. The collection includes

- E-mail
- File Transfers
- Terminal Emulation

Reference Models

A reference model is a blueprint that states how communication should accomplished. It gives addresses for all the processes that are required for effective communication. Here each of the processes are divided into logical groupings called layers. It can be referred as layered architecture. Each layer has its own protocol and supplies the necessary parameter to all layers.

Advantages of using Reference Model

- Software developers can focus on the functions of only one layer since they know that another layer will handle the functions on which they are not currently working
- If changes are made to one layer, the contents of other layer do not get changed
- The reference model also promotes compatibility. If the software developer adheres to the specification that is outlined in the reference model, all the protocols written to conform to that model will work together. Compatibility creates the potential for a large number of protocols to be written and used

Types of Reference Model

The International Standard Organization – Open System Interconnection (ISO-OSI) is defined for an open protocol suite (support heterogeneous networks). The Department of Defense reference model (DOD Model) is a condensed version of the OSI Model. It is also referred to as four layer reference model.

OSI Reference Model	DOD Reference Model

The OSI reference model has seven layers. They are Application, Presentation, Session, Transport, Network, DataLink and Physical layer.	This model has four layers.These layers are Process/Application,Host-to-Host (Transport), Internet and Network Access(Network layer).This model is referred to as the internet protocol suite.
In the OSI model the top three layers such as the application, presentation and session layers deal with functions that aid applications in communicating with other applications.They specifically deal with tasks like filename formats, code sets, user interfaces, compression, encryption and other functions relating to the exchange occurring between applications.	In the DOD model the process/application layer performed by the top three layers of the OSI model. This layer contains a vast array of protocols that combine to integrate the various activities and duties performed by the OSI's presentation, Session and Application layers. The Process/Application layer defines protocols for application communication .It also controls user interface applications.
The transport layer deals with the logical transmission of data. The layer takes care of sizing of the packets sent by each application. The transport layer ensures error-free delivery of data.	The Host-to-Host layer parallels the function of the transport layer of the OSI model. This layer defines a protocol for setting up levels of transmission service for applications. It deals with issues like creating reliable end-to-end communication and ensuring error-free delivery of data. It handles packet sequencing and maintains data integrity.
The network layer sets the degree of reliability for packets reaching the destination and the logical address for each machine. When messages reach this layer from the ones above it, the network layer attaches a directive that includes both, the message's source and destination address, forming a packet ready for delivery. Next, the best route for the packet to take across the network to its destination must be chosen. This is known as routing and is handled by routers.	The internet layer corresponds to the network layer of OSI model. This layer defines a protocol relating to the logical transmission of packets over the entire network. It takes care of addressing the hosts by giving them an IP address. It also handles routing of packets among multiple networks. The internet layer controls the communication flow between applications.
The datalink and the physical layers are the bottom layers of the OSI model, which handles the physical transmission of data. The layers take what is passed down to them and put it into a format that can be sent over a variety of physical transmission media like cable, fiber optics, microwave and radio links. They encode data into different media signals to match the specific media over which they will be transmitted.	The network access layer monitors the of data exchange between the host and the network. This layer is equivalent of the data link and the physical layers of the OSI model. This layer defines the protocol for the physical transmission of data.

Table 19.2

Protocols in TCP/IP Protocol Suite

The Process/Application layer protocol addresses the ability of one application to communicate with another , regardless of hardware platforms, operating system and other features of the two hosts.

Protocols	Description
Telnet	Allows a user on the remote client machine, called Telnet client , to access the resource of another machine, the telnet server.
File Transfer Protocol(FTP)	Provides the facility of transferring files.
Trivial File Transfer Protocol(TFTP)	It is similar to FTP, however it has no directory browsing facilities and sends much smaller blocks of data than FTP.
Network File System(NFS)	Specializes in file-sharing abilities.
Line – Printer – Daemon(LPD)	This protocol is designed for sharing printers.
Simple Network Management Protocol(SNMP)	This protocol provides for the collection and manipulation of valuable network information needed in trouble shooting the network problems.
X Windows	Designed for client server operations, X windows defines a protocol for the writing of graphical user interface-based client/server application.

Host – to – Host Layer Protocols

Host-to-Host layer protocols shield the upper layer protocols from the complexities of the network. The two protocols are:

Transmission Control Protocol (TCP) – This protocol tests for errors , resends data if necessary and reports the occurrence of errors to the upper layers if it cannot manage to solve the problem itself. The protocol takes large amounts of information from an application and breaks them down into segments.

User Datagram Protocol (UDP) – This protocol is used instead of TCP. UDP is considered as a thin protocol since it does not occupy much space over the network.

Internet Layer Protocols

This protocol provides the functions such as

- Routing
- Providing a single network interface to the upper layers

The two main protocols of this layer are

- **Internet Protocol (IP)** – IP takes segments from the host-to-host layer and fragments them into datagrams (packets)
- Address Resolution Protocol (ARP) – It is used to find the destination's hardware address

Network Access Layer Protocol

The main duties of the network access layer are

- Receiving an IP datagram and framing it into a stream of bits (1's and 0's) for physical transmission of data

- Specifying the MAC address of the machine

- Ensuring that the stream of bits that make up the frame have been accurately received by calculating a Cyclic Redundancy Checksum(CRC) jellybean count

- Specifying the access methods to the physical networks, such as Token-passing for token ring, fiber distribution data distribution (FDDI) and polling for IBM Mainframes

- Specifying the physical media , the connector, electrical signaling and timing rules

Some of the technologies used to implement the network access layers are

- LAN-oriented protocols
- Ethernet (thick coaxial cable, thin coaxial cable and twisted-pair cable)
- Token Ring
- ARCnet
- WAN – Oriented protocols
- Point-to-Point protocol(PPP)
- X.25
- Frame Delay

Testing the TCP/IP Using IFCONFIG and PING

Ifconfig Utility

The IFCONFIG utility is used to verify the TCP/IP configuration parameters on the host computer. The configuration parameters include the IP address , subnet mask and default gateway. The IFCONFIG utility is useful in determining whether the configuration is initialized. It is also useful in determining the configuration of a duplicate IP address.

```
# ifconfig –a
```

PING Utility

The PING (Packet Internet Gopher) utility is used to test connectivity of the TCP/IP host, after the configuration is verified with the IFCONFIG utility. The PING utility is a diagnostic tool which is used to test TCP/IP configurations. It is also used to diagnose connection failures. The ICMP echo request and echo failure messages are used by PING to determine whether a particular TCP/IP host is available and functional.

```
# ping <ip-address>
```

IP Address

An IP address is a numeric identifier that is assigned to each machine on an IP network. It is used to identify the location of the device that is assigned to it on the network. The IP address of the machine is a software address and not a hardware address.

TCP/IP host is identified by a logical IP address. A unique IP address is required for each host and the network component uses TCP/IP for purposes of communication.

The IP address identifies the location of the system on the network and it is globally unique format.

NetworkID and HostID

Each IP address defines the networkID and hostID.

NetworkID is used to identify the systems that are located on the same physical network.

All the systems on the network must have the same networkID. It should be unique.

NetworkID is **130. 56. 30.57**

|_____|

|______________

HostID is used to identifies a workstation, server, router or other TCP/IP hosts within a network. It is used to uniquely identify each machine on the network. It has its own unique hostID for each and every machine

Example

HostID is **130.56.30.57**

|_____|

|______________

Each IP address is 32 bits long and is composed of four 8 bit fields called octets. The octet represents a decimal in the range 0-255.This format is called dotted decimal notation.

Binary Format	**Dotted Decimal Notation**
10000011 01101011 00000011 00011000	131.107.3.24

Address Classes

The address class defines which bits are used for the network ID and which bits are used for the Host ID's. It also defines the possible number of networks and the number of hosts per network The internet community has defined three address classes based on the network size.

There are three address classes:

Class A Address class

Class A addresses are assigned to networks with a very large number of hosts. The high order bit in a class. A network is always set to zero. The next seven bits (completing the first octet) complete the networkID. The remaining 24 bits (that from the last three octet) represent the hostID. There can be a maximum of 126 networks and approximately 17 million hosts per network in a class A address class.

Specification:

Class	A
Format	Net.Node.Node.Node
Leading Bit Pattern	0
Decimal range of first byte of network address	1-127
Maximum Networks	127
Maximum Nodes per Network	16,777,216

Class B Address Class

Class B addresses are assigned to medium-sized to large-sized networks. The two high-order bits in a class B network are always set to binary 10.The next 14 bits complete the networkID. The remaining 16 bits represent the host ID. There can be a maximum of 16,384 networks and approximately 65000 hosts per network in class B address class.

Specification:

Class	**B**
Format	Net.Net.Node.Node
Leading Bit Pattern	10
Decimal range of first byte of network address	128-191
Maximum Networks	127
Maximum Networks	16,384
Maximum Nodes per Network	65,534

Class C Address Class

Class C networks are used for small local area networks(LANs).The three high-order bits in a class C networks are always set to binary 110.The next 21 bits complete the networkID. The remaining 8 bits represent the hostID. There can be maximum of approximately 2 million networks and 254 hosts per network in class C address class.

Specification:

Class	**C**
Format	Net.Net.Net.Node
Leading Bit Pattern	110
Decimal range of first byte of network address	192-223
Maximum Networks	2,097,152
Maximum Nodes per Network	254

19.8 Short Summary

- A network is a group of two or more computers linked together to share the resources like printers, data, disk drives, cdrom drives & tapedrive
- The P2P acronym technically stands for “peer-to-peer” computer networking
- In a P2P type of network each workstation has equivalent capabilities and responsibilities. This differs from client/server architectures, in which some computers are dedicated to serving the others
- A local area network (LAN) consists of hardware and software that provide the capability to interconnect a variety of data-communicating devices within a limited area. LANs have become an integral part of computing and data communications
- A WAN is a geographically-dispersed collection of LANs. A network device called a router connects LANs to a WAN. In IP networking, the router maintains both a LAN address and a WAN address
- WANs differ from LANs in several important ways. Like the Internet, most WANs are not owned by any one organization but rather exist under collective or distributed ownership and management. WANs use technology like ATM, Frame Relay and X.25 for connectivity

- A reference model is a blueprint of how communication should be accomplished
- An IP address is a numeric identifier assigned to each machine on an IP network
- Transmission Control Protocol (TCP) tests for errors, resends data if necessary and reports the occurrence of errors to the upper layers if it cannot manage to solve the problem itself
- UDP is considered as a thin protocol since it does not occupy much space over the network
- The address class defines the type of bits are used for the network ID and which bits are used for the Host ID's
- The application layer defines how a user or process accesses the network
- The presentation layer defines the format, including syntax and semantics, of the data exchanged between devices
- The session layer defines how communication (known as a session) is established between the machines
- The transport layer defines the connection between the source and destination machines (end-to-end)
- The network layer defines how packets are routed through the network to their final destination
- The data link layer defines how blocks of data (packets) are reliably transferred between adjacent network nodes
- The physical layer defines how the raw bits of upper-level data are physically transferred over the network medium (Physical cable)
- The different types of Transmission Media are: twisted – pair cable, coaxial cable, Fiber Optic cables
- The Transmission Modes are: base band transmission, broad band transmission
- The different network topologies are Bus, Ring and Star

19.9 Brain Storm

1. What is a Peer-to-Peer Network?
2. What is a LAN Network?
3. Mention the difference between LAN and WAN.
4. What are the different types of topologies that are commonly used?
5. In which layer does ftp work?
6. Which layer establishes the session?
7. Which layer maintains the connection between systems?
8. In which layer does twisted-pair cable work?
9. Which is the device that is used to connect different networks across boundaries?
10. Mention the types of cable medium that the Ethernet Networks use.
11. What type of access method is used and why it is used?
12. What type of frame is used for Ethernet?
13. What is an IP address?
14. What is TCP?
15. What is UDP?

***

MS 3.5 UNIX Administration

Lecture 1 Introduction to Unix Administration

Introduction to Unix OS - Introduction to Operating System - History of Operating System - Features of Unix Operating System - Unix Architecture - Unix File System - System Administration

Lecture 2 Unix Refresher

Login - Logout - Unix Command – date –cal – finger – id – man - who

Lecture 3 Files and Directories Command

Unix Directories- File name Expansion - Working with Files - Comparing Files - Printing Files

Lecture 4 Working with IO Redirection, Pipes and Filters

IO Redirection, Pipes and Filters - Standard Input - IO Redirection – Pipes – Filters - Unix Process – Processes - Switching between Processes

Lecture 5 Introduction to Shell Programming

Shell Programming - Types of shell - Processing command by shells – Variable - Types of variables - Command substitution - Positional Parameters - The export command

Lecture 6 Advanced Shell Scripts

Advanced Shell Scripts - The echo command - Read command - The expr command - The if Statement - The for statement - The while statement - The until statement - The case statement - The break command - The Continue statement - The trap command

Lecture 7 Booting and Shutting

Booting - Types of booting - Boot Process - System Boot Sequence - Init process –Daemons - Run Levels - Overview of Run levels - Run levels functions - Run level identification - Run control scripts - Single/multi user mode - Shutting down

Lecture 8 User and Group Management

Managing Group - Groupadd command - Groupmod command - Groupdel command - Managing User - Useradd command - Usermod command - Userdel command

Lecture 9 Device and Disk Management

Device and Disk Management - Device Geometry – Partitions - Device naming - Adding hard disks - Character and block mode devices

Lecture 10 Introduction to File System

Local Based File System Types - Ofs (HDD) – Floppy - CD-ROM - The ext2 File System - Raw & Block device - Boot block - Super block - Backup super block - Cylinder groups – Inodes - Types of File System - Mounting the Local Based File System - Common Commands for File System Management - Managing Disk Use (Tasks)

Lecture 11 Network File system

Network File System (NFS) – *nsfd – mountd – lockd – statd - rpc.portmapper* - Starting and Stopping the nfs Daemons - To start and stop NFS Daemons - Configuring nfs Servers and Clients - Mounting the Remote File System - NFS-mounting the File System - Mounting the NFS File System

Lecture 12 Virtual File System

Virtual File System - Types of Virtual File System - Swap File System - Process File System - Process File System - What is /proc File System? - What is in this File System?

Lecture 13 Security

Security - Types of Security - File Server Security - System Level Security

Lecture 14 Printer Management

Printer Management - Configuring Print Services - Setting up the Printer - Setting up the Print server - Setting up the Print client - Print service Architecture - Print Service Directories - Print Functions - Starting and Stopping Daemons - Configuring Printer - Printing a file - To print a file - To view the status of a printer - Canceling the print job

Lecture 15 Backup and Recovery

Backups - *tar command - cpio command - dd command - mt command - dump/restore command*

Lecture 16 Space Management

Space Management – Quota - Quota set up for a user -Turning quotas on - Setting up quotas for single user - Setting quotas for multiple user - To check quota consistency - Checking quotas on a file system

Lecture 17 Scheduling of System Events

Scheduling of System Events - Types of Scheduling Events - Jobs Scheduling Using Crontab - Jobs Scheduling Using At

Lecture 18 Performance Monitoring

Managing System Performance - Process Management - Process States - Process Management Commands - ps command - Listing Processes

Lecture 19 Network Management

Network - Types of network - Classification of network - LAN Fundamentals - Characteristic of LAN - Features of LAN - LANs and OSI Reference Model - OSI Reference Model - LAN interconnection - Basic Network design - Wide Area Network - TCP/IP - Reference Models - Protocols in TCP/IP Protocol Suite - Testing the TCP/IP using IPCONFIG and PING - IP address