UNIT – 1 (Question Bank)

1. Explain the function of OS.
2. Explain the architecture of UNIX with neat diagram.
3. Explain the various types of kernels.
4. List and explain different services provided by OS.
5. What is distributed and real-time systems?
6. Define a UNIX Shell. List its five responsibilities.
7. Write down five design issues of a distributed operating system.
8. What do you mean by Operating System? Explain various types of operating systems.
9. With a block diagram explains UNIX operating system organization and explain Kernel Shell Relationship.
10. What is the difference between UNIX and Windows Operating System?

UNIT – I (NOTES)

Introduction of Operating System

An operating system acts as an intermediary between the user of a computer and computer hardware. The purpose of an operating system is to provide an environment in which a user can execute programs in a convenient and efficient manner. An operating system is software that manages the computer hardware. The hardware must provide appropriate mechanisms to ensure the correct operation of the computer system and to prevent user programs from interfering with the proper operation of the system.

Operating System – Definition:

- An operating system is a program that controls the execution of application programs and acts as an interface between the user of a computer and the computer hardware.
- A more common definition is that the operating system is the one program running at all times on the computer (usually called the kernel), with all else being application programs.

- An operating system is concerned with the allocation of resources and services, such as memory, processors, devices, and information. The operating system correspondingly includes programs to manage these resources, such as a traffic controller, a scheduler, memory management module, I/O programs, and a file system.

Functions of Operating system – Operating system performs three functions:

1. **Convenience:** An OS makes a computer more convenient to use.
2. **Efficiency:** An OS allows the computer system resources to be used in an efficient manner.
3. **Ability to Evolve:** An OS should be constructed in such a way as to permit the effective development, testing and introduction of new system functions at the same time without interfering with service.

Operating system as User Interface –

1. User
2. System and application programs
3. Operating system
4. Hardware

Every general-purpose computer consists of the hardware, operating system, system programs, and application programs. The hardware consists of memory, CPU, ALU, and I/O devices, peripheral device, and storage device. System program consists of compilers, loaders, editors, OS, etc. The application program consists of business programs, database programs.

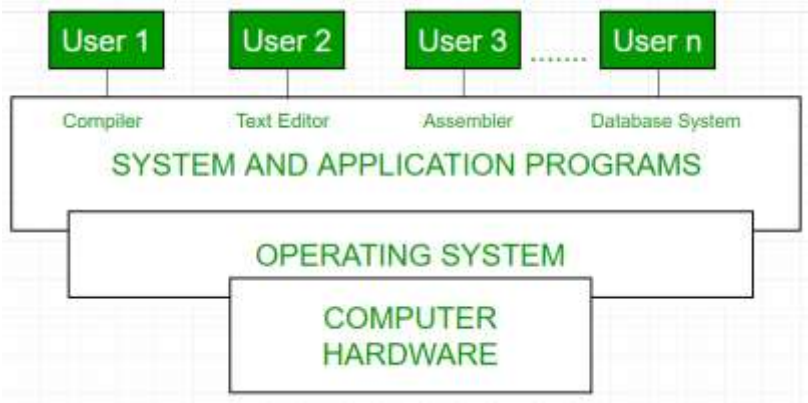


Fig. 1: Conceptual view of a computer system

Every computer must have an operating system to run other programs. The operating system coordinates the use of the hardware among the various system programs and application programs for various users. It simply provides an environment within which other programs can do useful work.

The operating system is a set of special programs that run on a computer system that allows it to work properly. It performs basic tasks such as recognizing input from the keyboard, keeping track of files and directories on the disk, sending output to the display screen and controlling peripheral devices.

OS is designed to serve two basic purposes:

1. It controls the allocation and use of the computing System's resources among the various user and tasks.
2. It provides an interface between the computer hardware and the programmer that simplifies and makes feasible for coding, creation, debugging of application programs.

The Operating system must support the following tasks.

The tasks are:

1. Provides the facilities to create, modification of programs and data files using an editor.
2. Access to the compiler for translating the user program from high level language to machine language.
3. Provide a loader program to move the compiled program code to the computer's memory for execution.
4. Provide routines that handle the details of I/O programming.

I/O System Management –

The module that keeps track of the status of devices is called the I/O traffic controller. Each I/O device has a device handler that resides in a separate process associated with that device. The I/O subsystem consists of,

- A memory Management component that includes buffering caching and spooling.
- A general device driver interface.

Drivers for specific hardware devices:

Assembler – The input to an assembler is an assembly language program. The output is an object program plus information that enables the loader to prepare the object program for execution. At one time, the computer programmer had at his disposal a basic machine that interpreted, through hardware, certain fundamental instructions. He would program this computer by writing a series of ones and Zeros (Machine language), place them into the memory of the machine.

Compiler – The High-level languages- examples are FORTRAN, COBOL, ALGOL and PL/I are processed by compilers and interpreters. A compiler is a program that accepts a source program in a "high-level language "and produces a corresponding object program. An interpreter is a program that appears to execute a source program as if it was machine language. The same name (FORTRAN, COBOL, etc.) is often used to designate both a compiler and its associated language.

Loader – A Loader is a routine that loads an object program and prepares it for execution. There are various loading schemes: absolute, relocating and direct-linking. In general, the loader must load, relocate and link the object program. The loader is a program that places programs into memory and prepares them for execution. In a simple loading scheme, the assembler outputs the machine language translation of a program on a secondary device and a loader places it in the core. The loader places into memory the machine language version of the user's program and transfers control to it. Since the loader program is much smaller than the assembler, those make more core available to the user's program.

History of Operating system – Operating system has been evolving through the years. Following Table shows the history of OS.

GENERATION	YEAR	ELECTRONIC DEVICE USED	TYPES OF OS DEVICE
First	1945-55	Vaccum Tubes	Plug Boards
Second	1955-65	Transistors	Batch Systems
Third	1965-80	Integrated Circuits(IC)	Multiprogramming
Fourth	Since 1980	Large Scale Integration	PC

Types of Operating System – Batch Operating System- Sequence of jobs in a program on a computer without manual interventions.

- Time sharing operating System- allows many users to share the computer resources. (Max utilization of the resources).
- Distributed operating System- Manages a group of different computers and make appear to be a single computer.
- Network operating system- computers running in different operating system can participate in common network (It is used for security purpose)
- Real time operating system – meant applications to fix the deadlines.

Examples of Operating System are;

- Windows (GUI based, PC)
- GNU/Linux (Personal, Workstations, ISP, File and print server, Three-tier client/Server)
- Mac OS (Macintosh), used for Apple's personal computers and work stations (MacBook, iMac).
- Android (Google's Operating System for smartphones/tablets/smartwatches)
- iOS (Apple's OS for iPhone, iPad and iPod Touch)

Needs of OS:

- An Operating system is a set of programs which acts as an interface between the user and the computer.

- Operating systems manages these resources, resolves conflicts and tries to optimise the performance of the system.
- Generally part of the operating system free sites in memory and supervises execution of all other programs executed on the computer.
- The user communicates with the computer the operating system commands and program instructions.
- It plays an important role in loading programs from disk into memory, displaying messages, translating programs and in outputting results.
- Basically it executes the user programs without letting the user bother about the detailed steps involved in executing them.

Services of O.S.

Here is a list of common services offered by an almost all operating systems:

- User Interface
- Program Execution
- File system manipulation
- Input / Output Operations
- Communication
- Resource Allocation
- Error Detection
- Accounting
- Security and protection

Types of O.S.

1. Simple Batch System
2. Multiprogramming Batch System
3. Multiprocessor Systems-Parallel System
4. Distributed Operating System
5. Real Time Operating System

1. Simple Batch Systems

- In this type of system, there is no direct interaction between user and the computer.
- The user has to submit a job (written on cards or tape) to a computer operator.
- Then computer operator places a batch of several jobs on an input device.
- Jobs are batched together by type of languages and requirement.
- Then a special program, the monitor, manages the execution of each program in the batch.
- The monitor is always in the main memory and available for execution.

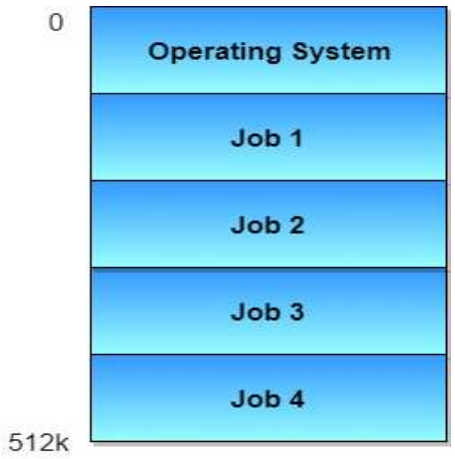
Advantages of Simple Batch Systems

1. No interaction between user and computer.
2. No mechanism to prioritise the processes.

2. Multiprogramming Batch Systems

- In this the operating system picks up and begins to execute one of the jobs from memory.
- Once this job needs an I/O operation operating system switches to another job (CPU and OS always busy).
- Jobs in the memory are always less than the number of jobs on disk(Job Pool).
- If several jobs are ready to run at the same time, then the system chooses which one to run through the process of CPU Scheduling.
- In Non-multiprogrammed system, there are moments when CPU sits idle and does not do any work.
- In Multiprogramming system, CPU will never be idle and keeps on processing.

Time Sharing Systems are very similar to Multiprogramming batch systems. In fact time sharing systems are an extension of multiprogramming systems. In Time sharing systems the prime focus is on minimizing the response time, while in multiprogramming the prime focus is to maximize the CPU usage.



3. Multiprocessor Systems-Parallel System

A Multiprocessor system consists of several processors that share a common physical memory. Multiprocessor system provides higher computing power and speed. In multiprocessor system all processors operate under single operating system. Multiplicity of the processors and how they do act together are transparent to the others.

Advantages of Multiprocessor Systems

1. Enhanced performance.
2. Execution of several tasks by different processors concurrently, increases the system's throughput without speeding up the execution of a single task.
3. If possible, system divides task into many subtasks and then these subtasks can be executed in parallel in different processors. Thereby speeding up the execution of single tasks.

4. Distributed Operating System

The motivation behind developing distributed operating systems is the availability of powerful and inexpensive microprocessors and advances in communication technology.

These advancements in technology have made it possible to design and develop distributed systems comprising of many computers that are inter connected by communication networks. The main benefit of distributed systems is its low price/performance ratio.

Advantages Distributed Operating System

1. As there are multiple systems involved, user at one site can utilize the resources of systems at other sites for resource-intensive tasks.
2. Fast processing.
3. Less load on the Host Machine.

Types of Distributed Operating Systems;

Following are the two types of distributed operating systems used:

1. Client-Server Systems
2. Peer-to-Peer Systems

5. Real Time Operating System

It is defined as an operating system known to give maximum time for each of the critical operations that it performs, like OS calls and interrupt handling. The Real-Time Operating systems which guarantees the maximum time for critical operations and complete them on time are referred to as Hard Real-Time Operating Systems. While the real-time operating systems that can only guarantee a maximum of the time, i.e. the critical task will get priority over other tasks, but no certain of completing it in a defined time. These systems are referred to as Soft Real-Time Operating Systems. Note: above notes for system administration is very much summarised because we well studies the O.S as a primary subject in previous semester.

Overview of UNIX:

- UNIX is a computer operating system.
- An operating system is the program that controls all the other parts of a computer system, both the hardware and the software. It allocates the computer's resources and schedules tasks. It allows you to make use of the facilities provided by the system. Every computer requires an operating system.
- UNIX is a multi-user, multi-tasking operating system. Multiple users may have multiple tasks running simultaneously. This is very different from PC operating systems such as MS-DOS or MS-Windows (which allows multiple tasks to be carried out simultaneously but not multiple users).

- UNIX is a machine independent operating system. Not specific to just one type of computer hardware. Designed from the beginning to be independent of the computer hardware.
- UNIX is a software development environment. Was born in and designed to function within this type of environment.
- The "UNIX" trademark, previously owned by AT&T and then deeded to UNIX Systems Laboratories (USL), an AT&T subsidiary, passed to Novell when it acquired USL. After a brief period of negotiations with rival Unix vendors, namely, Sun Microsystems, Santa Cruz Operation, International Business Machines, and Hewlett-Packard, Novell granted exclusive licensing rights of the UNIX trademark to X/Open Co. Ltd., an Open Systems industry standards branding agent based in the United Kingdom.

History of UNIX

- 1969: Developed at AT&T Bell Labs in Murray Hill, New Jersey, one of the largest research facilities in the world. Created in an environment when most computer jobs were fed into a batch system.
- Developed by researchers who needed a set of computing tools to help them with their projects and their collaborators. Allowed a group of people working together on a project to share selected data and programs.
- 1975: AT&T makes UNIX widely available - offered to educational institutions at minimal cost. Becomes popular with university computer science programs. AT&T distributes standard versions in source form: Version 6 (1975), Version 7 (1978), System III (1981).
- 1984 to date: University of California, Berkeley adds major enhancements, creates Berkeley Standard Distribution (BSD)
- 1984 to date: Many Berkeley features incorporated into new AT&T version: System V
- UNIX has become the operating system of choice for engineering and scientific workstations.
- Two variations maintain popularity today, AT&T System V based and the Berkeley Standard Distribution.
- Current versions (1/95)are System V release 4.2 .and 4.4 BSD

- Work is in progress to develop a Portable Operating System specification based on UNIX (IEEE POSIX committee).

UNIX Philosophy

- Make each program do one thing well. Reusable software tools: 1 tool = 1 function
- Expect the output of every program to become the input of another, yet unknown, program to combine simple tools to perform complex tasks
- Prototyping: get something small working as soon as possible and modify it incrementally until it is finished
- Use terse commands and messages: reduces typing and screen output

Why UNIX?

- **Hardware independence**
    - operating system code is written in C language rather than a specific assembly language
    - operating system software can be easily moved from one hardware system to another
    - UNIX applications can be easily moved to other UNIX machines. Porting is usually as simple as transfer of the source and a recompile.

- **Productive environment for software development**
    - Rich set of tools
    - Versatile command language.

- **Distributed processing and multi-tasking**

Features of UNIX

The following are the advantages of Unix Features.

Portability:

The system is written in high-level language making it easier to read, understand, change and, therefore move to other machines. The code can be changed and complied on a new machine. Customers can then choose from a wide variety of hardware vendors without being locked in with a particular vendor.

Machine-independence:

The System hides the machine architecture from the user, making it easier to write applications that can run on micros, mins and mainframes.

Multi-Tasking:

UNIX is a powerful multi-tasking operating system; it means when a active task in in process, there can be a simultaneous background process working too. Unix handles these active and background threads efficiently and manages the system resources in a fair-share manner.

Multi-User Operations:

UNIX is a multi-user system designed to support a group of users simultaneously. The system allows for the sharing of processing power and peripheral resources, white at the same time providing excellent security features.

Hierarchical File System:

UNIX uses a hierarchal file structure to store information. This structure has the maximum flexibility in grouping information in a way that reflects its natural state. It allows for easy Maintenance and efficient implementation.

UNIX shell:

UNIX has a simple user interface called the shell that has the power to provide the services that the user wants. It protects the user from having to know the intricate hardware details.

Pipes and Filters:

UNIX has facilities called Pipes and Filters which permit the user to create complex programs from simple programs.

Utilities:

UNIX has over 200 utility programs for various functions. New utilities can be built effortlessly by combining existing utilities.

Software Development Tools:

UNIX offers an excellent variety of tools for software development for all phases, from program editing to maintenance of software,

Comparison between UNIX and Windows

The main differences between Windows and UNIX are as follows:

1. UNIX is a Command Line User Interface and Windows is Graphic User Interface operating system.
2. UNIX is command based and Windows is menu based operating system.
3. Windows is event driven whereas this feature is absent in Unix operating system.
4. File system in Unix is (STD.ERR,STD.IO), and in Windows it is (FAT32,NTFS).
5. In UNIX multiprocessing is possible whereas it is not possible in Windows.
6. In terms of security, UNIX is more secure than Windows as we can restrict the permission of each user.
7. Windows operating system support plug and play and this feature is not available in UNIX.
8. Windows is licensed operating system and Unix is free source operating system.

Multi-threading: It is one of the important feature of operating system which allows its use by more than one user or accepting multiple requests at a time. Yes, Windows support multi-threading.

Components of UNIX:

- **Kernel**

- The core of the UNIX system. Loaded at system start up (boot). Memory-resident control program.
- Manages the entire resources of the system, presenting them to you and every other user as a coherent system. Provides service to user applications such as device management, process scheduling, etc.
- Example functions performed by the kernel are:
  - Managing the machine's memory and allocating it to each process.
  - Scheduling the work done by the CPU so that the work of each user is carried out as efficiently as is possible.
  - accomplishing the transfer of data from one part of the machine to another
  - interpreting and executing instructions from the shell
  - enforcing file access permissions
- You do not need to know anything about the kernel in order to use a UNIX system. These details are provided for your information only.

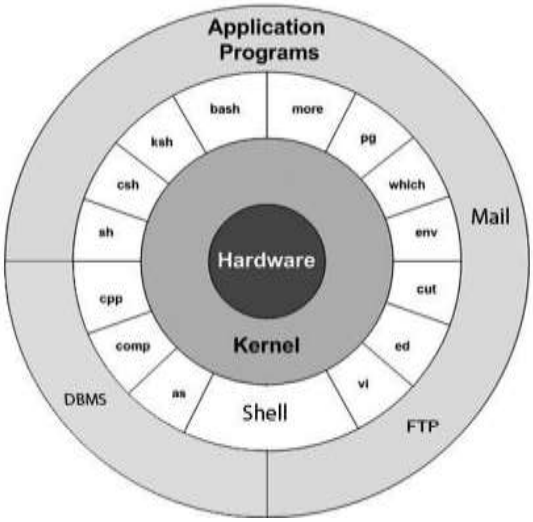
- **Shell**
  - Whenever you login to a UNIX system you are placed in a shell program. The shell's prompt is usually visible at the cursor's position on your screen. To get your work done, you enter commands at this prompt.
  - The shell is a command interpreter; it takes each command and passes it to the operating system kernel to be acted upon. It then displays the results of this operation on your screen.
  - Several shells are usually available on any UNIX system, each with its own strengths and weaknesses.
  - Different users may use different shells. Initially, your system adminstrator will supply a default shell, which can be overridden or changed. The most commonly available shells are:
    - Bourne shell (sh)
    - C shell (csh)
    - Korn shell (ksh)

- TC Shell (tcsh)
- Bourne Again Shell (bash)

- Each shell also includes its own programming language. Command files, called "shell scripts" are used to accomplish a series of tasks.

- Utilities
  - UNIX provides several hundred utility programs, often referred to as commands.
  - Accomplish universal functions
    - editing
    - file maintenance
    - printing
    - sorting
    - programming support
    - online info

Structure of UNIX:

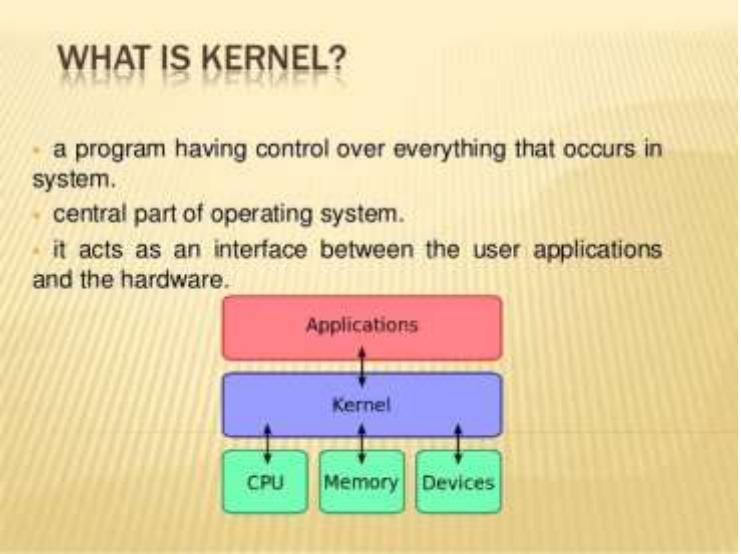


Structure of kernel

Kernel:

The kernel is a computer program that is the **core of a computer's operating system,** with complete control over everything in the system. It manages following resources of the Linux system:

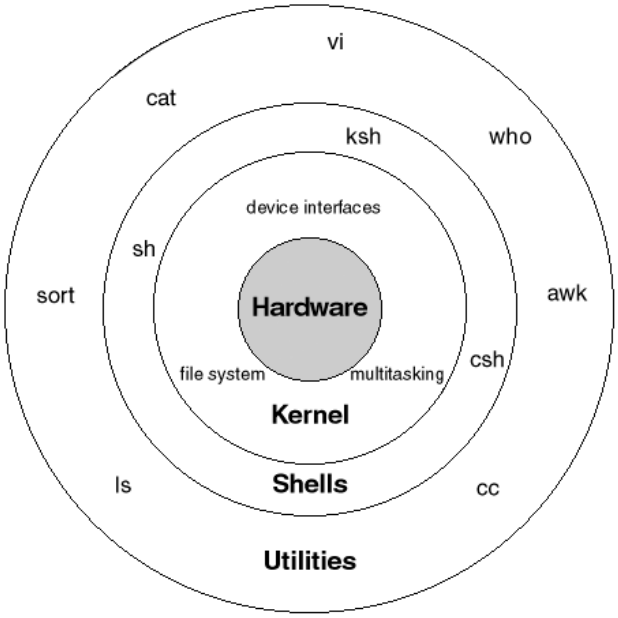
- File management
- Process management
- I/O management
- Memory management
- Device management etc.



Structure of Shell

Shell:

- A shell is special user program which provide an interface to user to use operating system services.
- Shell accepts human readable commands from user and converts them into something which kernel can understand.
- It is a command language interpreter that executes commands read from input devices such as keyboards or from files.
- The shell gets started when the user logs in or starts the terminal.



Shell is broadly classified into two categories:

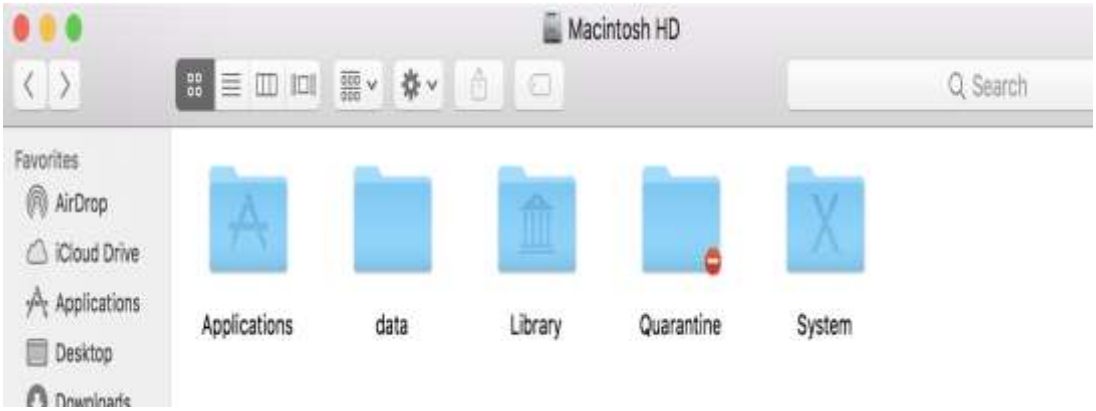
- Command Line Shell
- Graphical shell

Command Line Shell

- Shell can be accessed by user using a command line interface.
- A special program called `Terminal` in **Linux/MAC OS** or **Command Prompt** in Windows OS is provided to type in the human readable commands such as "cat", "ls" etc. and then it is being execute.
- The result is then displayed on the terminal to the user. A terminal in Mac OS:
- Working with command line shell is very powerful, it allows user to store commands in a file and execute them together.
- This way any repetitive task can be easily automated.
- These files are usually called `batch files` in Windows and `Shell Scripts` in Linux/MAC OS systems.

Graphical Shell

- Provide means for manipulating programs based on graphical user interface (GUI), by allowing for operations such as opening, closing, moving and resizing windows, as well as switching focus between windows.
- Mac OS or Ubuntu OS can be considered as good example which provides GUI to user for interacting with program.
- User do not need to type in command for every actions.
- A GUI in Mac OS will look like below.



There are several shells are available for Linux systems. Each shell does the same job but understands different commands and provide different built in functions.

- **BASH (Bourne Again Shell)** – It is most widely used shell in Linux systems. It is used as default login shell in Linux systems and in MAC OS. It can also be installed on Windows OS.
- **CSH (C Shell)** – The C shell's syntax and usage are very similar to the C programming language.
- **KSH (Korn Shell)** – The Korn Shell also was the base for the POSIX Shell standard specifications etc.

UNIT – 2 (Question Bank)

1. Explain VI editor in detail.
2. Explain with example various file handling commands.
3. Explain various process management.
4. Explain the output from $wc temp > temp.
5. Describe different ways of saving and quitting VI editor.
6. In which mode of operation of VI editor Search-cum-Replace feature is supported. Explain how it can be utilized?
7. Explain the three modes of the Vi-Editor.
8. What is a pipe in UNIX? When do you required nameless pipe?
9. Write a shell script which displays names of directories in PATH in one line each.
10. What is difference between Message Queue and Pipes?

UNIT – 2 (NOTES)

UNIX DIRECTORY SYSTEM

Introduction to Unix File System

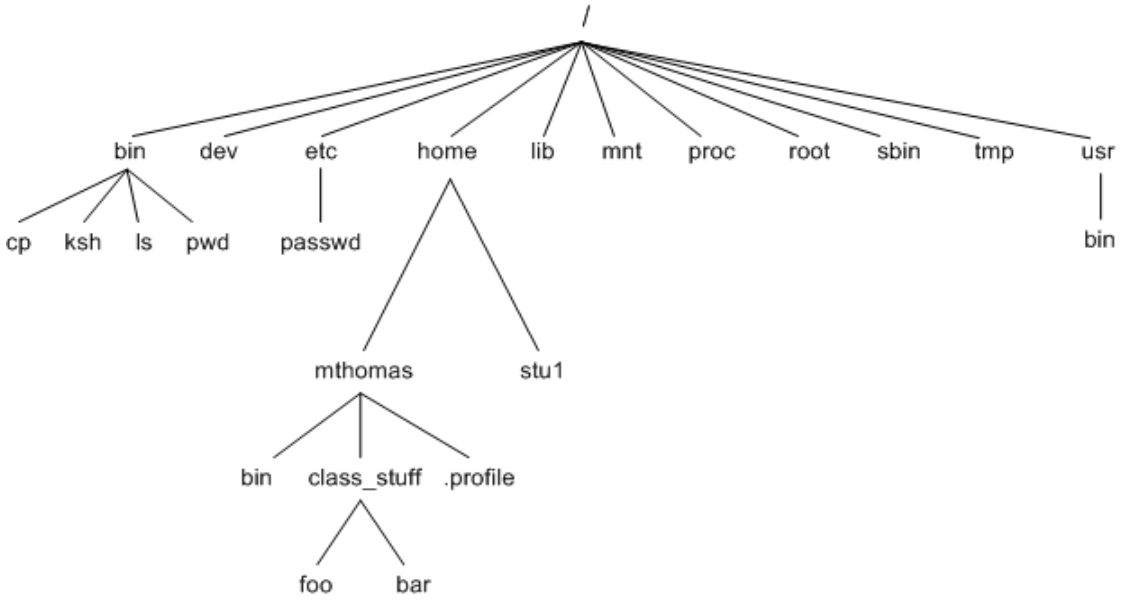
The Unix file system is a methodology for logically organizing and storing large quantities of data such that the system is easy to manage. A **file** can be informally defined as a collection of (typically related) data, which can be logically viewed as a stream of bytes (i.e. characters). A file is the smallest unit of storage in the Unix file system. By contrast, a **file system** consists of files, relationships to other files, as well as the attributes of each file. File attributes are information relating to the file, but do not include the data contained within a file. File attributes for a generic operating system might include (but are not limited to):

- a file type (i.e. what kind of data is in the file)
- a file name (which may or may not include an extension)
- a physical file size
- a file owner

- file protection/privacy capability
- file time stamp (time and date created/modified)

Additionally, file systems provide tools which allow the manipulation of files, provide a logical organization as well as provide services which map the logical organization of files to physical devices. From the beginners' perspective, the Unix file system is essentially composed of files and **directories**. Directories are special files that may contain other files. The Unix file system has a hierarchical (or tree-like) structure with its highest level directory called root (denoted by /, pronounced slash). Immediately below the root level directory are several subdirectories, most of which contain system files. Below this can exist system files, application files, and/or user data files. Similar to the concept of the process parent-child relationship, all files on a Unix system are related to one another. That is, files also have a parent-child existence. Thus, all files (except one) share a common parental link, the top-most file (i.e. /) being the exception.

Below is a diagram (slice) of a "typical" Unix file system. As you can see, the top-most directory is / (slash), with the directories directly beneath being system directories. Note that as Unix implementations and vendors vary, so will this file system hierarchy. However, the organization of most file systems is similar.



While this diagram is not all inclusive, the following system files (i.e. directories) are present in most Unix file systems:

- bin - short for binaries, this is the directory where many commonly used executable commands reside
- dev - contains device specific files
- etc - contains system configuration files
- home - contains user directories and files
- lib - contains all library files
- mnt - contains device files related to mounted devices
- proc - contains files related to system processes
- root - the root users' home directory (note this is different than /)
- sbin - system binary files reside here. If there is no sbin directory on your system, these files most likely reside in etc
- tmp - storage for temporary files which are periodically removed from the filesystem
- usr - also contains executable commands

UNIX COMMAND

In simple words we can that UNIX commands are used to access and modify the system files and operational or available database.

A **Unix** shell is a **command-line** interpreter or shell that provides a **command line** user interface for **Unix**-like operating systems. ... All **Unix** shells provide filename wildcarding, piping, here documents, **command** substitution, variables and control structures for condition-testing and iteration.

Uses

- Establish is secure or valid connection
- Organisation and maintenance
- Uploading and downloading file
- manage files or directories

- confirm current system status
- allow you to navigate your Unix or Linux system

User access commands

Linux is a clone of UNIX, the **multi-user operating system** which can be accessed by many users simultaneously. Linux can also be used in mainframes and servers without any modifications. But this raises security concerns as an unsolicited or **malign user** can **corrupt, change or remove crucial data**. For effective security, Linux divides authorization into 2 levels.

1. Ownership
2. Permission

The concept of **permissions** and **ownership** is crucial in Linux. Here, we will discuss both of them. Let us start with the **Ownership.**

Ownership of Linux files

Every file and directory on your Unix/Linux system is assigned 3 types of owner, given below.

User

A user is the owner of the file. By default, the person who created a file becomes its owner. Hence, a user is also sometimes called an owner.

Group

A user- group can contain multiple users. All users belonging to a group will have the same access permissions to the file. Suppose you have a project where a number of people require access to a file. Instead of manually assigning permissions to each user, you could add all users to a group, and assign group permission to file such that only this group members and no one else can read or modify the files.

Other

Any other user who has access to a file. This person has neither created the file, nor he belongs to a user group who could own the file. Practically, it means everybody else. Hence, when you set the permission for others, it is also referred as set permissions for the world. Now, the big question arises how does **Linux distinguish** between these three user types so that a user 'A' cannot affect a file which contains some other user 'B's' vital information/data. It is like you do not want your colleague, who works on your Linux computer, to view your images. This is where **Permissions** set in, and they define **user behaviour**. Let us understand the **Permission system** on Linux.

Permissions

Every file and directory in your UNIX/Linux system has following 3 permissions defined for all the 3 owners discussed above.

- **Read:** This permission give you the authority to open and read a file. Read permission on a directory gives you the ability to lists its content.
- **Write:** The write permission gives you the authority to modify the contents of a file. The write permission on a directory gives you the authority to add, remove and rename files stored in the directory. Consider a scenario where you have to write permission on file but do not have write permission on the directory where the file is stored. You will be able to modify the file contents. But you will not be able to rename, move or remove the file from the directory.
- **Execute:** In Windows, an executable program usually has an extension ".exe" and which you can easily run. In Unix/Linux, you cannot run a program unless the execute permission is set. If the execute permission is not set, you might still be able to see/modify the program code(provided read & write permissions are set), but not run it.

Let's see this in action

ls - l on terminal gives

ls - l

Here, we have highlighted **'-rw-rw-r--'**and this weird looking code is the one that tells us about the permissions given to the owner, user group and the world.

Here, the first '-' implies that we have selected a file.p>

Else, if it were a directory, **d** would have been shown.

The characters are pretty easy to remember.

r	=	read	permission
w	=	write	permission
x	=	execute	permission

- = no permission

Let us look at it this way.

The first part of the code is **'rw-'**. This suggests that the owner 'Home' can:

- Read the file
- Write or edit the file
- He cannot execute the file since the execute bit is set to '-'.

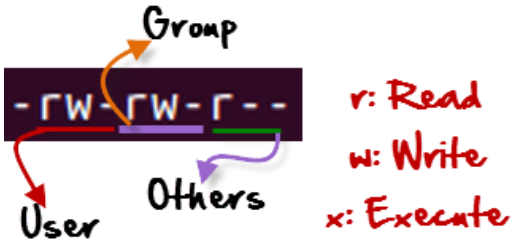
By design, many Linux distributions like Fedora, CentOS, Ubuntu, etc. will add users to a group of the same group name as the user name. Thus, a user 'tom' is added to a group named 'tom'.

The second part is **'rw-'.** It for the user group 'Home' and group-members can:

- Read the file
- Write or edit the file

The third part is for the world which means any user. It says **'r--'.** This means the user can only:

- Read the file



Changing file/directory permissions with 'chmod' command

Say you do not want your colleague to see your personal images. This can be achieved by changing file permissions. We can use the '**chmod**' command which stands for 'change mode'. Using the command, we can set permissions (read, write, execute) on a file/directory for the owner, group and the world. **Syntax:**

chmod permissions filename

There are 2 ways to use the command -

1. **Absolute mode**
2. **Symbolic mode**

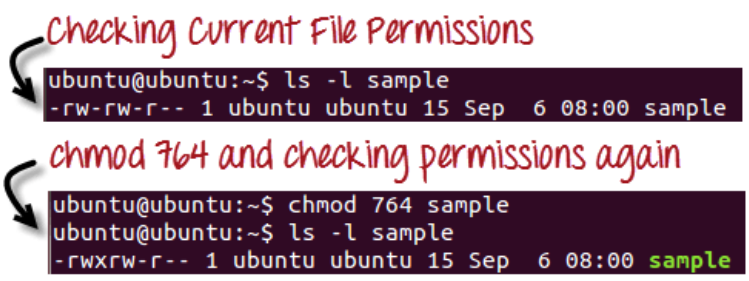
Absolute (Numeric) Mode

In this mode, file **permissions are not represented as characters but a three-digit octal number**.

The table below gives numbers for all for permissions types.

Number	Permission Type	Symbol
0	No Permission	---
1	Execute	--x
2	Write	-w-
3	Execute + Write	-wx
4	Read	r--
5	Read + Execute	r-x
6	Read +Write	rw-
7	Read + Write +Execute	rwx

Let's see the chmod command in action.



In the above-given terminal window, we have changed the permissions of the file 'sample to '764'.

'764' absolute code says the following:

- Owner can read, write and execute
- Usergroup can read and write
- World can only read

This is shown as '-rwxrw-r-

This is how you can change the permissions on file by assigning an absolute number.

Symbolic Mode

In the Absolute mode, you change permissions for all 3 owners. In the symbolic mode, you can modify permissions of a specific owner. It makes use of mathematical symbols to modify the file permissions.

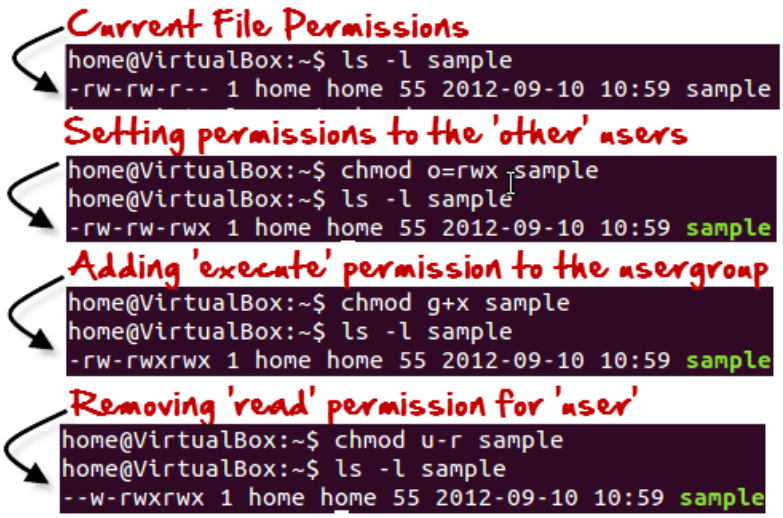
Operator	Description
+	Adds a permission to a file or directory
-	Removes the permission
=	Sets the permission and overrides the permissions set earlier.

The various owners are represented as -

User Denotations	
U	user/owner
G	Group
O	Other

A	All

We will not be using permissions in numbers like 755 but characters like rwx. Let's look into an example,



Changing Ownership and Group

For changing the ownership of a file/directory, you can use the following command:

chown user

In case you want to change the user as well as group for a file or directory use the command

Summary:

- Linux being a multi-user system uses permissions and ownership for security.
- There are three user types on a Linux system viz. User, Group and Other
- Linux divides the file permissions into read, write and execute denoted by r,w, and x
- The permissions on a file can be changed by 'chmod' command which can be further divided into Absolute and Symbolic mode
- The 'chown' command can change the ownership of a file/directory. Use the following commands: chown user file or chown user:group file
- The 'chgrp' command can change the group ownership **chrgrp group filename**

2.3 DIRECTORY COMMANDS

Directory commands are those commands that are used by user to complete the following takes…

- Create a new file
- Modifies the existing file
- Update the files
- Delete the files

All commands are used to perfume upon directory in linux. We can see the available directory by **dir** command.

Making Directories

mkdir (make directory)

We will now make a subdirectory in your home directory to hold the files you will be creating and using in the course of this tutorial. To make a subdirectory called unixstuff in your current working directory type - **% mkdir unixstuff**

To see the directory you have just created, type

% ls

Changing to a different directory

cd (change directory)

The command **cd** *directory* means change the current working directory to 'directory'. The current working directory may be thought of as the directory you are in, i.e. your current position in the file-system tree. To change to the directory you have just made, type

% cd unixstuff

Type **ls** to see the contents (which should be empty)

The parent directory (..)

(**..**) means the parent of the current directory, so typing

% cd ..

pwd (print working directory)

Pathnames enable you to work out where you are in relation to the whole file-system. For example, to find out the absolute pathname of your home-directory, type **cd** to get back to your home-directory and then type

% pwd

File manipulation commands

This lesson will introduce you to the following commands:

- **cp** - copy files and directories
- **mv** - move or rename files and directories
- **rm** - remove files and directories
- **mkdir** - create directories

These four commands are among the most frequently used Linux commands. They are the basic commands for manipulating both files and directories. Now, to be frank, some of the tasks performed by these commands are more easily done with a graphical file manager. With a file manager, you can drag and drop a file from one directory to another, cut and paste files, delete files, etc. The answer is power and flexibility. While it is easy to perform simple file manipulations with a graphical file manager, complicated tasks can be easier with the command line programs. For example, how would you copy all the HTML files from one directory to another, but only copy files that did not exist in the destination directory or were newer than the versions in the destination directory? Pretty easy with the command line:

1.cp

The **cp** program copies files and directories. In its simplest form, it copies a single file:

[me@linuxbox me]$ **cp file1 file2**

It can also be used to copy multiple files (and/or directories) to a different directory:

[me@linuxbox me]$ **cp file... directory**

2.mv

The **mv** command moves or renames files and directories depending on how it is used. It will either move one or more files to a different directory, or it will rename a file or directory. To rename a file, it is used like this:

[me@linuxbox me]$ **mv filename1 filename2**

To move files (and/or directories) to a different directory:

[me@linuxbox me]$ **mv file... directory**

3.rm

The **rm** command removes (deletes) files and directories.

[me@linuxbox me]$ **rm file...**

It can also be used to delete directories:

[me@linuxbox me]$ **rm -r directory...**

Be careful with rm!

Linux does not have an undelete command. Once you delete something with **rm**, it's gone. You can inflict terrific damage on your system with **rm** if you are not careful, particularly with wildcards.

4.mkdir

The **mkdir** command is used to create directories. To use it, you simply type:

[me@linuxbox me]$ **mkdir directory...**

Security and Protection Commands

There are many aspects to security on Unix systems – from setting up accounts to ensuring that legitimate users have no more privilege than they need to do their jobs. This is look at some of the most essential security commands for day-to-day work on Unix systems.

Sudo

Running privileged commands with sudo – instead of switching user to root – is one essential good practice as it helps to ensure that you only use root privilege when needed and limits the impact of mistakes. Your access to the sudo command depends on settings in the /etc/sudoers and /etc/group files.

If you run sudo and ask who you are, for example, you'll get confirmation that you're running the command as root.

$ sudo whoami

root

If you manage the sudo setup for users, you also need to be comfortable with the visudo command.

visudo

The visudo command allows you to make changes to the /etc/sudoers file by opening the file in a text editor and checking your changes for syntax. Run the command with “sudo visudo” and make sure that you understand the syntax. Privileges can be assigned by user or by group. On most Linux systems, the /etc/sudoers file will already be configured with groups like those shown below that allow the privileges to be assigned to groups set up in the /etc/group file. In those cases, you don’t need to use the visudo command at all – just be familiar with the groups that bestow root privileges in this way, and make your updates to the /etc/group file.

%admin ALL=(ALL) ALL

%sudo ALL=(ALL:ALL) ALL

%wheel ALL=(ALL:ALL) ALL

Note that group names are preceded by the % sign.

You can probably display the group providing sudo access in your /etc/group file like this since it is probably one of these:

$ egrep "admin|sudo|wheel" /etc/group

sudo:x:27:shs,jdoe

The easiest way to give someone sudo privilege is to add them to the empowered group in /etc/group. However, that means that they can run any command as root. If you want some users to have root authority for a limited set of commands (e.g., adding and removing accounts) you can define the commands you want them to be able to run through a command alias like this:

Cmnd_Alias ACCT_CMDS = /usr/sbin/adduser, /usr/sbin/deluser

who and w

The who and w commands show you who is logged into the system though w shows more information such as where they logged in from, when they logged in and how long they've been idle.

```
$ w
 18:03:35 up 9 days, 22:48,  2 users,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
joe      tty2     /dev/tty2        27Apr18  9days  7:34   0.09s /usr/lib/x86_64-linux
shs      pts/1    192.168.0.15     09:50    7.00s  0.28s  0.00s w
```

Use the "sudo update-alternatives – config editor" command if you don't like the default editor that's called into play when you run the visudo command. It will offer a number of editors as options and change your setting.

last

The last command shows you recent logins for users and is often helpful when you're trying to track down changes or other activity.

```
$ last nemo
nemo  pts/1   192.168.0.15   Wed May  2 07:01 - 08:29  (01:27)
wtmp begins Tue May  1 10:21:35 2018
```

Nemo hasn't logged in for a while. He might be on vacation (maybe fishing?) or have just recently left the company. This kind of information can be useful in deciding whether you need to follow up on this.

find

The find command is used for many types of searches. When it comes to security, you might find yourself looking for files that have no owners (no corresponding accounts) or are both world-writable and executable. Find commands are easy to compose but require some familiarity with its many options for defining what you're looking for. The first of these two commands will find files with no currently defined owners. The second will find files that likely anyone can both run and modify.

$ sudo find /home -nouser
$ sudo find / -perm -o=wx

Keep in mind that the -o in the second command refers to the "other" group – not the owner and not the group associated with the files.

file

The file command looks at a file and determines what kind of file it is based on its contents, not its name. Many files (like jpeg files) contain identifiers near the beginnings of the files that identify them. The ".jpg" file in the example below is clearly not a true jpeg file but an executable – in spite of its name.

passwd [option(s)] [username]

Users may change their own passwords at any time using this command. Furthermore, the administrator root can use the command to change the password of any user on the system.

What is meant by Intermachine communication?

Intermachine communication is the communication between two machines. With respect to the context of UNIX operating system there are many commands available for achieving this intermachine communication. Some of them are listed below namely:

uucp:

This command is used for transferring of files between UNIX machines. There are many options available in this command to name one of which is uucp with a –n options notifies or messages the user on the system that the file is sent.

send:

It is used for sending a message from a socket.

cu:

This is used for communicating by dialing a phone number and there by useful for connecting to another UNIX system and in fact even to a non-UNIX system. Using this command interactive message can be shared and also files can be transferred. In this command first a connection is established between the two systems which are to have communication and then after establishing the connection communication is made. This command also has number of options in it and the generally used option is cu with –d which displays trace messages.

ct:

This creates connection between two terminals by dialing a phone number of a modem that is attached to a terminal and there by creates a login process to that terminal.

Process Management Commands

Generally, an application process' lifecycle has three main states: start, run, and stop. Each state can and should be managed carefully if we want to be competent administrators. These eight commands can be used to manage processes through their lifecycles

1.Starting a process

The easiest way to start a process is to type its name at the command line and press Enter. If you want to start an Nginx web server, type **nginx**. Perhaps you just want to check the version.

alan@workstation:~$

2.Viewing your executable path

The above demonstration of starting a process assumes the executable file is located in your executable path. Understanding this path is key to reliably starting and managing a process. Administrators often customize this path for their desired purpose. You can view your executable path using **echo $PATH**.

alan@workstation:~$ echo $PATH

/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin

WHICH

Use the which command to view the full path of an executable file.

alan@workstation:~$ which nginx

/opt/nginx/bin/nginx

I will use the popular web server software Nginx for my examples. Let's assume that Nginx is installed. If the command **which nginx** returns nothing, then Nginx was not found because which searches only your defined executable path. There are three ways to remedy a situation where a process cannot be started simply by name. The first is to type the full path.

alan@workstation:~$ /home/alan/web/prod/nginx/sbin/nginx -v

nginx version: nginx/1.14.0

The second solution would be to install the application in a directory in your executable's path. However, this may not be possible, particularly if you don't have root privileges. The third solution is to update your executable path environment variable to include the directory where the specific application you want to use is installed. This solution is shell-dependent. For example, Bash users would need to edit the PATH= line in their .bashrc file.

PATH="$HOME/web/prod/nginx/sbin:$PATH"

Now, repeat your echo and which commands or try to check the version. Much easier!

alan@workstation:~$ echo $PATH

/home/alan/web/prod/nginx/sbin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin

alan@workstation:~$ which nginx

/home/alan/web/prod/nginx/sbin/nginx

alan@workstation:~$ nginx -v

nginx version: nginx/1.14.0

Keeping a process running

NOHUP

A process may not continue to run when you log out or close your terminal. This special case can be avoided by preceding the command you want to run with the nohup command. Also, appending an ampersand (&) will send the process to the background and allow you to continue using the terminal. For example, suppose you want to run myprogram.sh.

nohup myprogram.sh &

One nice thing nohup does is return the running process's PID. I'll talk more about the PID next.

Manage a running process

Each process is given a unique process identification number (PID). This number is what we use to manage each process. We can also use the process name, as I'll demonstrate below. There are several commands that can check the status of a running process. Let's take a quick look at these.

PS

The most common is ps. The default output of ps is a simple list of the processes running in your current terminal. As you can see below, the first column contains the PID.

alan@workstation:~$ ps

PID TTY TIME CMD

23989 pts/0 00:00:00 bash

24148 pts/0 00:00:00 ps

PGREP

The pgrep command was created to further simplify things by removing the need to call grep separately.

alan@workstation:~$ pgrep nginx

20520

20521

PIDOF

Another nifty one is pidof. This command will check the PID of a specific binary even if another process with the same name is running. To set up an example, I copied my Nginx to a second directory and started it with the prefix set accordingly. In real life, this instance could be in a different location, such as a directory owned by a different user. If I run both Nginx instances, the **ps -ef** output shows all their processes.

alan@workstation:~$ ps -ef |grep nginx

alan 20881 1454 0 11:18 ? 00:00:00 nginx: master process ./nginx -p /home/alan/web/prod/nginxsec

alan 20882 20881 0 11:18 ? 00:00:00 nginx: worker process

alan 20895 1454 0 11:19 ? 00:00:00 nginx: master process nginx

alan 20896 20895 0 11:19 ? 00:00:00 nginx: worker process

TOP

The top command has been around a long time and is very useful for viewing details of running processes and quickly identifying issues such as memory hogs. Its default view is shown below.

```
alan@workstation:~$ top -p20881 -p20882 -p20895 -p20896
Tasks:  4 total,  0 running,  4 sleeping,  0 stopped,  0 zombie
%Cpu(s):  2.8 us,  1.3 sy,  0.0 ni, 95.9 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 16387132 total, 10856008 free,  1857648 used,  3673476 buff/cache
KiB Swap:       0 total,       0 free,       0 used. 14177928 avail Mem
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
20881 alan      20   0   12016    348      0 S   0.0  0.0   0:00.00 nginx
20882 alan      20   0   12460   1644    932 S   0.0  0.0   0:00.00 nginx
20895 alan      20   0   12016    352      0 S   0.0  0.0   0:00.00 nginx
20896 alan      20   0   12460   1628    912 S   0.0  0.0   0:00.00 n
```

3. Stopping a process

KILL

Interestingly, there is no stop command. In Linux, there is the kill command. Kill is used to send a signal to a process. The most commonly used signal is "terminate" (SIGTERM) or "kill" (SIGKILL). However, there are many more. Below are some examples. The full list can be shown with **kill -L**.

1) SIGHUP 2) SIGINT 3) SIGQUIT 4) SIGILL 5) SIGTRAP

6) SIGABRT 7) SIGBUS 8) SIGFPE 9) SIGKILL 10) SIGUSR1

11) SIGSEGV 12) SIGUSR2 13) SIGPIPE 14) SIGALRM 15) SIGTERM

Notice signal number nine is SIGKILL. Usually, we issue a command such as **kill -9 20896**. The default signal is 15, which is SIGTERM. Keep in mind that many applications have their own method for stopping. Nginx uses a **-s** option for passing a signal such as "stop" or "reload." Generally, I prefer to use an application's specific method to stop an operation. However, I'll demonstrate the kill command to stop Nginx process 20896 and then confirm it is stopped with pgrep. The PID 20896 no longer appears.

alan@workstation:~$ kill -9 20896

alan@workstation:~$ pgrep nginx

20881

20882

20895

22123

PKILL

The command pkill is similar to pgrep in that it can search by name. This means you have to be very careful when using pkill. In my example with Nginx, I might not choose to use it if I only want to kill one Nginx instance. I can pass the Nginx option **-s stop** to a specific instance to kill it, or I need to use grep to filter on the full ps output.

/home/alan/web/prod/nginx/sbin/nginx -s stop

/home/alan/web/prod/nginxsec/sbin/nginx -s stop

If I want to use pkill, I can include the **-f** option to ask pkill to filter across the full command line argument. This of course also applies to pgrep. So, first I can check with **pgrep -a** before issuing the **pkill -f**.

alan@workstation:~$ pgrep -a nginx

20881 nginx: master process ./nginx -p /home/alan/web/prod/nginxsec

20882 nginx: worker process

20895 nginx: master process nginx

20896 nginx: worker process

Most of these commands have many command line options, so I always recommend reading the man page on each one. While most of these exist across platforms such as Linux, Solaris, and BSD, there are a few differences. Always test and be ready to correct as needed when working at the command line or writing scripts.

Linux I/O Redirection

Redirection can be defined as changing the way from where commands read input to where commands sends output. You can redirect input and output of a command. For redirection, meta characters are used. Redirection can be into a **file** (shell meta characters are angle **brackets** '<', '>') or a **program** (shell meta characters are **pipe**symbol '|').

Standard Streams In I/O Redirection

The bash shell has three standard streams in I/O redirection:

- **standard input (stdin) :** The stdin stream is numbered as stdin (0). The bash shell takes input from stdin. By default, keyboard is used as input.
- **standard output (stdout) :** The stdout stream is numbered as stdout (1). The bash shell sends output to stdout. Output goes to display.
- **standard error (stderr) :** The stderr stream is numbered as stderr (2). The bash shell sends error message to stderr. Error message goes to display.

Redirection Into A File

Each stream uses redirection commands. Single bracket '>' or double bracket '>>' can be used to redirect standard output. If the target file doesn't exist, a new file with the same name will be created.

Overwrite

Commands with a single bracket **'>' overwrite** existing file content.

- > : standard output
- < : standard input
- 2> : standard error

Note: Writing **'1>'** or **'>'** and **'0<'** or **'<'** is same thing. But for stderr you have to write **'2>'**.

Syntax:

1. cat **> <fileName>**

   Example:

1. cat **>** sample.txt

Look at the above snapshot, command "cat > sample.txt" has created 'sample.txt' with content 'a, b, c'. Same file 'sample.txt' is created again with command **"cat > sample.txt"** and this time it overwrites earlier file content and only displays 'd, e, f '.

Append

Commands with a double bracket **'>>' do not overwrite** the existing file content.

- >> - standard output
- << - standard input
- 2>> - standard error

Syntax:

1. cat **>> <fileName>**

   Example:

1. cat **>>** sample.txt

Look at the above snapshot, here again we have created two files with the same name using '>>' in command **"cat >> sample.txt"**. But this time, content doesn't overwrite and everything is displayed.

Redirection into a Program

Pipe redirects a stream from one **program** to another. When pipe is used to send standard output of one program to another program, first program's data will not be displayed on the terminal, only the second program's data will be displayed. Although the functionality of pipe may look similar to that of '>' and '>>' but has a significance difference. Pipe redirects data from one program to another while brackets are only used in redirection of files.

Example:

1. ls *.txt | cat **>** txtFile

Look at the above snapshot, command **"ls *.txt | cat > txtFile"** has put all the '.txt' files into a newly created file 'txtFile'.

Piping in UNIX

A pipe is a form of redirection (transfer of standard output to some other destination) that is used in Linux and other Unix-like operating systems to send the output of one command/program/process to another command/program/process for further processing. The Unix/Linux systems allow stdout of a command to be connected to stdin of another command. You can make it do so by using the pipe character **'|'**. Pipe is used to combine two or more commands, and in this, the output of one command acts as input to another command, and this command's output may act as input to the next command and so on. It can also be visualized as a temporary connection between two or more commands/ programs/ processes. The command line programs that do the further processing are referred to as filters.

This direct connection between commands/ programs/ processes allows them to operate simultaneously and permits data to be transferred between them continuously rather than having to pass it through temporary text files or through the display screen. Pipes are unidirectional **i.e data flows from left to right through the pipeline**

Syntax :

command_1 | command_2 | command_3 | .... | command_N

Listing all files and directories and give it as input to more command.

$ ls -l | more

the output of ls -l is displayed one screen at a time. The pipe acts as a container which takes the output of ls -l and gives it to more as input. This command does not use a disk to connect standard output of ls -l to the standard input of more because pipe is implemented in the main memory.

In terms of I/O redirection operators, the above command is equivalent to the following command sequence.

$ ls -l -> temp

more -> temp (or more temp)

[contents of temp]

rm temp

2. Use sort and uniq command to sort a file and print unique values.

$ sort record.txt | uniq

This will sort the given file and print the unique values only.

3. Use head and tail to print lines in a particular range in a file.

$ cat sample2.txt | head -7 | tail -5

This command selects first 7 lines and last 5 lines from the file and print those lines which are common to both of them.

4. Use ls and find to list and print all lines matching a particular pattern in matching files.

$ ls -l | find ./ -type f -name "*.txt" -exec grep "program" {} \;

This command selects files with **.txt** extension in the given directory and search for pattern like “program” in the above example and prints those ine which have program in them.

5. Use cat, grep, tee and wc command to read the particular entry from user and store in a file and print line count.

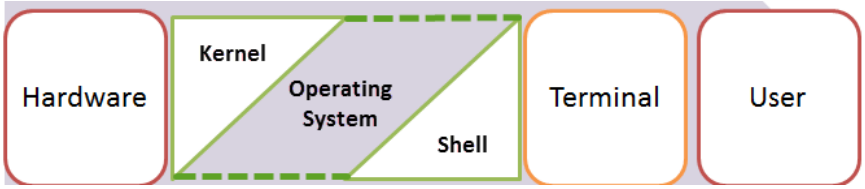
$ cat result.txt | grep "Rajat Dua" | tee file2.txt | wc -l

This command select **Rajat Dua** and store them in file2.txt and print total number of lines matching Rajat Dua. Difference b/w Shell scripting and shell programming. There's not much of the difference b/w doing shell scripting and programming except for the fact that in programming you are using the actual processor to do something but in shell scripting you command the utilities provided by the operating system which in turn use processor to perform certain functions.

What is a Shell?

An Operating is made of many components, but its two prime components are -

- Kernel
- Shell



A Kernel is at the nucleus of a computer. It makes the communication between the hardware and software possible. While the Kernel is the innermost part of an operating system, a shell is the outermost one. A shell in a Linux operating system takes input from you in the form of

commands, processes it, and then gives an output. It is the interface through which a user works on the programs, commands, and scripts. A shell is accessed by a terminal which runs it. When you run the terminal, the Shell issues **a command prompt (usually $),** where you can type your input, which is then executed when you hit the Enter key. The output or the result is thereafter displayed on the terminal. The Shell wraps around the delicate interior of an Operating system protecting it from accidental damage. Hence the name **Shell**.

Types of Shell

There are two main shells in Linux:

1. The Bourne Shell: The prompt for this shell is $ and its derivatives are listed below:

- POSIX shell also is known as sh
- Korn Shell also knew as sh
- **B**ourne **A**gain **SH**ell also knew as bash (most popular)

2. The C shell: The prompt for this shell is %, and its subcategories are:

- C shell also is known as csh
- Tops C shell also is known as tcsh

We will discuss bash shell based shell scripting in this tutorial.

What Is Shell Scripting?

SHELL SCRIPTING is writing a series of commands for the shell to execute. It can combine lengthy and repetitive sequences of commands into a single and simple script, which can be stored and executed anytime. This reduces the effort required by the end user. Let us understand the steps in creating a Shell Script.

1. **Create a file using** a **vi** editor(or any other editor). Name script file with **extension .sh**
2. **Start** the script with **#! /bin/sh**
3. Write some code.
4. Save the script file as filename.sh
5. For **executing** the script type **bash filename.sh**

"#!" is an operator called shebang which directs the script to the interpreter location. So, if we use"#! /bin/sh" the script gets directed to the bourne-shell.

Let's create a small script -

```
#!/bin/sh
ls
```

Let's see the steps to create it -

Command 'ls' is executed when we execute the scrip sample.sh file.

Adding shell comments

Commenting is important in any program. In Shell programming, the syntax to add a comment is

```
#comment
```

Let understand this with an example.

What are Shell Variables?

As discussed earlier, Variables store data in the form of characters and numbers. Similarly, Shell variables are used to store information and they can by the shell only.

For example, the following creates a shell variable and then prints it:

```
variable ="Hello"
echo $variable
```

Below is a small script which will use a variable.

```
#!/bin/sh
echo "what is your name?"
read name
echo "How do you do, $name?"
read remark
echo "I am $remark too!"
```

Let's understand, the steps to create and execute the script

As you see, the program picked the value of the variable 'name' as Joy and 'remark' as excellent.

This is a simple script. You can develop advanced scripts which contain conditional statements, loops, and functions. Shell scripting will make your life easy and Linux administration a breeze.

Administration Commands

NOTE

For details on specific commands, including syntax and examples, click on the specific command to go to its reference page.

Name	Description
clean	Internal namespace administration command.
cloneCollection	Copies a collection from a remote host to the current host.
cloneCollectionAsCapped	Copies a non-capped collection as a new capped collection.
collMod	Add options to a collection or modify a view definition.
compact	Defragments a collection and rebuilds the indexes.
connPoolSync	Internal command to flush connection pool.
convertToCapped	Converts a non-capped collection to a capped collection.
create	Creates a collection or a view.
createIndexes	Builds one or more indexes for a collection.

Name	Description
currentOp	Returns a document that contains information on in-progress operations for the database instance.
drop	Removes the specified collection from the database.
dropDatabase	Removes the current database.
dropConnections	Drops outgoing connections to the specified list of hosts.
dropIndexes	Removes indexes from a collection.
filemd5	Returns the md5 hash for files stored using GridFS.
fsync	Flushes pending writes to the storage layer and locks the database to allow backups.
fsyncUnlock	Unlocks one fsync lock.
getParameter	Retrieves configuration options.
killCursors	Kills the specified cursors for a collection.

Name	Description
killOp	Terminates an operation as specified by the operation ID.
listCollections	Returns a list of collections in the current database.
listDatabases	Returns a document that lists all databases and returns basic database statistics.
listIndexes	Lists all indexes for a collection.
logRotate	Rotates the MongoDB logs to prevent a single file from taking too much space.
reIndex	Rebuilds all indexes on a collection.
renameCollection	Changes the name of an existing collection.

Name	Description
setFeatureCompatibilityVersion	Enables or disables features that persist data that are backwards-incompatible.
setParameter	Modifies configuration options.
shutdown	Shuts down the mongod or mongos process.

What is the VI editor?

The VI editor is the most popular and classic text editor in the Linux family. Below, are some reasons which make it a widely used editor –

1) It is available in almost all Linux Distributions

2) It works the same across different platforms and Distributions

3) It is user-friendly. Hence, millions of Linux users love it and use it for their editing needs

Nowadays, there are advanced versions of the vi editor available, and the most popular one is **VIM** which is **V**i **Im**proved. Some of the other ones are Elvis, Nvi, Nano, and Vile. It is wise to learn vi because it is feature-rich and offers endless possibilities to edit a file.

To work on VI editor, you need to understand **its operation modes**. They can be divided into two main parts.

Command mode:

- The vi editor opens in this mode, and it only **understands commands**

- In this mode, you can, **move the cursor and cut, copy, paste the text**
- This mode also saves the changes you have made to the file
- **Commands are case sensitive.** You should use the right letter case.

Insert mode:

- This mode is for inserting text in the file.
- You can switch to the Insert mode from the command mode **by pressing 'i' on the keyboard**
- Once you are in Insert mode, any key would be taken as an input for the file on which you are currently working.
- To return to the command mode and save the changes you have made you need to press the Esc key

Starting the vi editor

To launch the VI Editor -Open the Terminal (CLI) and type

vi <filename_NEW> or <filename_EXISTING>

And if you specify an existing file, then the editor would open it for you to edit. Else, you can create a new file.

VI Editing commands

- i - Insert at cursor (goes into insert mode)
- a - Write after cursor (goes into insert mode)
- A - Write at the end of line (goes into insert mode)
- ESC - Terminate insert mode
- u - Undo last change
- U - Undo all changes to the entire line
- o - Open a new line (goes into insert mode)
- dd - Delete line
- 3dd - Delete 3 lines.
- D - Delete contents of line after the cursor

- C - Delete contents of a line after the cursor and insert new text. Press ESC key to end insertion.
- dw - Delete word
- 4dw - Delete 4 words
- cw - Change word
- x - Delete character at the cursor
- r - Replace character
- R - Overwrite characters from cursor onward
- s - Substitute one character under cursor continue to insert
- S - Substitute entire line and begin to insert at the beginning of the line
- ~ - Change case of individual character

Note: You should be in the "**command mode" to execute these commands**. VI editor is **case-sensitive** so make sure you type the commands in the right letter-case. Make sure you press the right command otherwise you will end up making undesirable changes to the file. You can also enter the insert mode by pressing a, A, o, as required.

Moving within a file

- k - Move cursor up
- j - Move cursor down
- h - Move cursor left
- l - Move cursor right

You need to be in the command mode to move within a file. The default keys for navigation are mentioned below else; You can **also use the arrow keys on the keyboard**.

Saving and Closing the file

- Shift+zz - Save the file and quit
- :w - Save the file but keep it open
- :q - Quit without saving
- :wq - Save the file and quit

Summary:

- The vi editor is the most popular and commonly used Linux text editor
- It is usually available in all Linux Distributions.
- It works in two modes, Command and Insert
- Command mode takes the user commands, and the Insert mode is for editing text
- You should know the commands to work on your file easily
- Learning to use this editor can benefit you in creating scripts and editing files.

File Handling commands

Basic Navigation commands

"cd" The change directory. This command will navigate you from your current location to the path specified after the command.

"cd /" will take you to the root directory of your system.

"pwd" This command is issued to display your current working directory.

If you issued the above change directory command, your pwd command will now indicate you are at the root directory location.

john@john-desktop:~$ cd /

john@john-desktop:/$ pwd

/

Absolute Paths and Relative Paths

In the above example we passed the path / to the cd command. It tells cd where to go. The "/" argument is an absolute path meaning that it specifies a location relative to the root of the filesystem tree. There are many other absolute paths. One easy way to recognise an absolute path is that they all start with a "/". For example, If I navigate to the path /home/john, we would be telling the cd command to navigate to the "/"location then the home location and finally the john area.

Listing files with the "ls" command

One of the most useful and common commands that you will encounter is the "ls" command. (Literally meaning list). When "ls" is run alone, it will list any normal files within the current directory. If you add the "-a" parameter to the "ls" command thus making "ls -a", any files including hidden files will be displayed in the current directory. In Linux, hidden files are file names that begin with a dot ".". If you supply the "-l" option to the "ls" command, then we get an extended or long view of the files in the current directory. It is also possible to pass a directory to be displayed to the "ls" command. "ls -l /home/john" this would display all files in the /home/john area. Another useful parameter that can be passwd to the "ls" command is the "-t" option. The "-t" option sorts by modification time (newest first). Passing the "-r" option will reverse the order. Multiple options can be added together "ls -rtla". This command lists all files including hidden in reverse time order with the extended listing option!

$ pwd
/home/landoflinux
$ ls
examples.desktop visiblefile
$ ls -a
. .bash_logout examples.desktop .profile
.. .bashrc .hiddenfile visiblefile

Long - extended file listings "-l"

When you issue the "-l" option, much more detail is displayed about your files. The very first column displays the file type and permissions of that file within the listing. The next column indicates how many links there are to this file system object. The third and fourth column indicate the file owner and the group. (These details will be covered later in more detail). The fifth column indicates the size. The sixth column is the modification time/date. And finally the last column is the name of the file. If you see a "->" then this indicates there is a symbolic link to another file. (This will be covered in more detail later).

Display directory information "-d"

As we have seen earlier, if we pass a directory path to the "ls" command, the contents of that directory are displayed. However, sometimes it useful to display information about the directory without display its contents. To achieve this, we pass the "-d" flag". "ls -ld /home/john" would display information relating to the directory and not its contents:

$ cd /

$ pwd

/

$ ls -ld /home/john

drwxr-xr-x 70 john john 12288 Jan 17 19:52 /home/john

Recursive Listings "-R"

Sometimes you may wish to display directory information and also any other files and directories underneath. In the following example we use the "-ld" option to display directory information. Next we use the "-R" recursive listing option to display both directory and sub directory files and information

Creating Directories - "mkdir"

The "mkdir" command is used to create new directories. Directories allow you to organise your files and keep your system tidy. You will notice while navigating around your linux system that there are literally hundreds if not thousands of directories. Important system files and users files are stored within. In most cases, we will be working within our home area /home/user. In the example that follows we will be using /home/landoflinux. Simply issuing the "mkdir" command followed by a name will create you a single directory: "mkdir mydir1". If we wanted to create a directory with sever sub-directories, then we can not just issue "mkdir one/two/three" because we will receive an error "mkdir: cannot create directory `one/two/three': No such file or directory". To create nested directories, we would have to pass a "-p" flag to our command. This option will then create any missing directories. So now our command would be "mkdir -p one/two/three"

$ pwd

/home/landoflinux

$ mkdir mydir1

$ ls -l

total 16

Copying and Renaming Files - "cp" and "mv"

In this section we will learn how to create a file, copy the file and then rename the file. Firstly, we need to look at how we can create a file. There are numerous ways of creating files, either by a text editor of your choice or from simple commands or programs. In the examples below we will look at using the "touch" command, the "echo" command and simple redirection commands to generate a file.

Deleting Files and Directories - "rm" and "rmdir"

So far we have learnt some basic ways of creating files and directories. Now we need to learn how to remove these files and directory structures. The most common way of removing a file is to use the "rm" remove command. In its simplest form "rm file1" will remove the file specified. However, you can specify more than one file after the "rm" command. "rm file1 file2". It is also possible to use pattern matching "rm file*".

grep command in Unix/Linux

The grep filter searches a file for a particular pattern of characters, and displays all lines that contain that pattern. The pattern that is searched in the file is referred to as the regular expression (grep stands for globally search for regular expression and print out).

Syntax:

grep [options] pattern [files]

Options Description

-c : This prints only a count of the lines that match a pattern

-h : Display the matched lines, but do not display the filenames.

-i : Ignores, case for matching

-l : Displays list of a filenames only.

-n : Display the matched lines and their line numbers.

-v : This prints out all the lines that do not matches the pattern

-e exp : Specifies expression with this option. Can use multiple times.

-f file : Takes patterns from file, one per line.

-E : Treats pattern as an extended regular expression (ERE)

-w : Match whole word

-o : Print only the matched parts of a matching line,
with each such part on a separate output line.

Sample Commands

Consider the below file as an input.

$cat > geekfile.txt

unix is great os. unix is opensource. unix is free os.
learn operating system.
Unix linux which one you choose.
Unix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.

1. Case insensitive search : The -i option enables to search for a string case insensitively in the give file. It matches the words like “UNIX”, “Unix”, “unix”.

$grep -i "UNix" geekfile.txt

Output:

unix is great os. unix is opensource. unix is free os.
Unix linux which one you choose.
uNix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.

2. Displaying the count of number of matches : We can find the number of lines that matches the given string/pattern

$grep -c "unix" geekfile.txt

Output:

2

3. Display the file names that matches the pattern : We can just display the files that contains the given string/pattern.

$grep -l "unix" *

or

$grep -l "unix" f1.txt f2.txt f3.xt f4.txt

Output:

geekfile.txt

4. Checking for the whole words in a file : By default, grep matches the given string/pattern even if it found as a substring in a file. The -w option to grep makes it match only the whole words.

$ grep -w "unix" geekfile.txt

Output:

unix is great os. unix is opensource. unix is free os.

uNix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.

5. Displaying only the matched pattern : By default, grep displays the entire line which has the matched string. We can make the grep to display only the matched string by using the -o option.

$ grep -o "unix" geekfile.txt

Output:

unix

unix

unix

unix

unix

unix

6. Show line number while displaying the output using grep -n : To show the line number of file with the line matched.

$ grep -n "unix" geekfile.txt

Output:

1:unix is great os. unix is opensource. unix is free os.

4:uNix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.

7. Inverting the pattern match : You can display the lines that are not matched with the specified search sting pattern using the -v option.

$ grep -v "unix" geekfile.txt

Output:

learn operating system.

Unix linux which one you choose.

8. Matching the lines that start with a string : The ^ regular expression pattern specifies the start of a line. This can be used in grep to match the lines which start with the given string or pattern.

$ grep "^unix" geekfile.txt

Output:

unix is great os. unix is opensource. unix is free os.

9. Matching the lines that end with a string : The $ regular expression pattern specifies the end of a line. This can be used in grep to match the lines which end with the given string or pattern.

$ grep "os$" geekfile.txt

10.Specifies expression with -e option. Can use multiple times :

$grep –e "Agarwal" –e "Aggarwal" –e "Agrawal" geekfile.txt

11. -f file option Takes patterns from file, one per line.

$cat pattern.txt

Agarwal

Aggarwal

Agrawal

$grep –f pattern.txt geekfile.txt

UNIT – 3 (Question Bank)

1. What do you understand by System Administration?
2. Explain the functional activities of system administration in details.
3. Write short note on process management and monitoring.
4. List and elaborate the functional activities of UNIX system administration. Also explain the process for starting up the system.
5. Explain the process to recover from a system crash in a UNIX environment.
6. How users can be added and removed in UNIX system? Explain the process of changing the groups and pass words.
7. How the synchronization in the Kernel does take place?
8. What are the functional activities of system administrator?
9. How do we mount and unmount file system in UNIX?
10. Write the process of maintaining the security in UNIX.
11. Explain the process of setting up of remote communication in UNIX?
12. Write the procedure of installing printer and peripheral devices on Unix.

UNIT – 3 (NOTES)

SYSTEM ADMINISTRATION

System administration refers to the management of one or more hardware and software systems. The task is performed by a system administrator who monitors system health,

monitors and allocates system resources like disk space, performs backups, provides user access, manages user accounts, monitors system security and performs many other functions. System administration is a job done by IT experts for an organization. The job is to ensure that computer systems and all related services are working well. The duties in system administration are wide ranging and often vary depending on the type of computer systems being maintained, although most of them share some common tasks that may be executed in different ways. Common tasks include installation of new hardware or software, creating and managing user accounts, maintaining computer systems such as servers and databases, and planning and properly responding to system outages and various other problems. Other responsibilities may include light programing or scripting to make the system workflows easier as well as training computer users and assistants.

Process Management

A process is an instance of a program running in a computer. It is close in meaning to task, a term used in some operating systems. In UNIX and some other operating systems, a process is started when a program is initiated (either by a user entering a shell command or by another program).

Attributes of a process

The Attributes of the process are used by the Operating System to create the process control block (PCB) for each of them. This is also called context of the process. Attributes which are stored in the PCB are described below.

1. Process ID

When a process is created, a unique id is assigned to the process which is used for unique identification of the process in the system.

2. Program counter

A program counter stores the address of the last instruction of the process on which the process was suspended. The CPU uses this address when the execution of this process is resumed.

3. Process State

The Process, from its creation to the completion, goes through various states which are new, ready, running and waiting.

4. Priority

Every process has its own priority. The process with the highest priority among the processes gets the CPU first. This is also stored on the process control block.

5. General Purpose Registers

Every process has its own set of registers which are used to hold the data which is generated during the execution of the process

6. List of open files

During the Execution, Every process uses some files which need to be present in the main memory. OS also maintains a list of open files in the PCB.

7. List of open devices

OS also maintain the list of all open devices which are used during the execution of the process.

Process management:

A Program does nothing unless its instructions are executed by a CPU. A program in execution is called a process. In order to accomplish its task, process needs the computer resources. There may exist more than one process in the system which may require the same resource at the same time. Therefore, the operating system has to manage all the processes and the resources in a convenient and efficient way. Some resources may need to be executed by one process at one time to maintain the consistency otherwise the system can become inconsistent and deadlock may occur. The operating system is responsible for the following activities in connection with process management are;

1. Scheduling processes and threads on the CPUs.
2. Creating and deleting both user and system processes.
3. Suspending and resuming processes.
4. Providing mechanisms for process synchronization.
5. Providing mechanisms for process communication.

Process monitoring:

Monitoring the performance of operating systems and processes is essential to debug processes and systems, effectively manage system resources, making system decisions, and evaluating and examining systems. These tools are primarily divided into two main categories:

1) **Real time:** Real time monitoring tools are concerned with measuring the current system state and provide up to date information about the system performance.
2) **Log-based:** Log-based monitoring tools record system performance information for post-processing and analysis and to find trends in the system performance.

Unix is a multi-user system where the same resources can be shared by different users. It provides multi-tasking, wherein each user can execute many processes at the same time. It was the first operating system that was written in a high-level language (C Language). This made it easy to port to other machines with minimum adaptations. It provides a hierarchical file structure which allows easier access and maintenance of data. Unix has built-in networking functions so that different users can easily exchange information. Unix functionality can be extended through user programs built on a standard programming interface.

FUNCTIONAL ACTIVITIES OF SYSTEM ADMINISTRATION

There are some essential functions by system administration:

Adding and Removing Users:

The system administrator adds accounts for new users and removes the accounts of users that are no longer active. These processes of adding and removing users can be automated.

Adding and Removing Hardware:

When there is a change in hardware configuration, System administrator should handle the new device drivers, configuration of the device.

Performing Backups:

Performing backups is one the most important role of a System Administrator; to avoid chaos and data loss, System administrator should take backups. Taking backups can be automated.

Installing Software:

When a new software is installed, it should be tested many times before informing users. Local software should be installed in separate folders than system programs to avoid problems during, upgrade of the system or kernel, in case local software won't be overwritten by System Software.

Monitoring the System:

Unix systems require supervision. Daily activities include making sure that, email, web services, network services are working correctly. Writing shell script and watching log files for signs of trouble and keeping an eye on system resources.

Trouble Shooting:

Unix System administration has been like chaos. Finding the problem is sometimes harder than solving it. Trouble shooting is a key role of system administrator and it comes with experience.

Maintaining Local Documentation:

Updating and organizing man pages are yet another role of the SysAdmin.

Auditing Security:

Security is a very big issue. As the systems get bigger, security becomes more important. SysAdmin must implement security audits, run sniffers and script to ensure that network, system and file system is secure. A crash refers to as a system crash. It's a term used to describe a software program or hardware problem that is encountered without warning.

STARTING UP AND SHUTTING DOWN THE SYSTEM

Understanding the Boot Operation

When you boot the operating system, you initiate a set of tasks that the system must perform to operate successfully. The system is vulnerable during startup since it is loading the kernel into memory and initializing routines that it depends on for operation. Consequently, you should understand what is happening during the system boot, and be prepared to respond if problems occur.

Although certain boot operations are hardware dependent, some features typically apply to all systems. For example:

- The system always boots either automatically or manually.

  In an automatic boot, the system controls the entire operation. When you boot the system to multiuser mode, or shut down the system with the reboot flag, or when the system panics and recovers, you are relying on an automatic boot. With an automatic boot, the system begins the initialization process and continues until completion or failure.

  Manual intervention may be required if the automatic boot fails for some reason, for example, if the fsck command fails.

  In a manual boot, the system controls the initial operation, turns control of the procedure over to you, then reinstates control to complete the operation. When you boot the system to single-user mode, you are relying on a manual boot. In an automatic or a manual boot, the operation either succeeds or fails:

  - If the boot operation succeeds, the system is initialized. In single-user mode, the system displays the root prompt (#) on the console or on the terminal screen. In multiuser mode, the system displays the login prompt or a startup display. The prompt or startup display differs according to hardware capability and available startup software.
  - If the boot operation fails, the system displays an error message followed by a console firmware prompt (>>>). In the worst case, the system hangs without displaying a console prompt.
- The system boots to either single-user or multiuser mode.
  - In a boot to single-user mode, the software loads the kernel and proceeds through the initialization tasks associated with process 0 (initialization) and

process 1 (init). The init program creates a Bourne shell (sh), turns control over to you, and waits for you to exit from the shell with the exit command or Ctrl/d before continuing with its startup tasks.

Because init does not invoke the startup script prior to turning control over to you, the root file system is mounted read only, startup of the network and other daemons does not occur, file checking and correction are not enabled, and other operations necessary for full system use are not automatically available to you.

Usually you boot to single-user mode to perform specific administrative tasks that are best accomplished without the threat of parallel activity by other users. You perform these tasks manually before exiting from the Bourne shell. For example, you might check new hardware, mount and check aberrant file systems, change disk partitions, or set the system clock. When you finish your work, you return control to the system, and initcontinues with its startup tasks and boots to multiuser mode.

- In a boot to multiuser mode, the system loads the kernel and moves through various phases such as hardware and virtual memory initialization, resource allocation, scheduling, configuration, module loading, and so on. At the conclusion of the main initialization tasks (process 0), init (process 1) starts an additional set of tasks that includes reading the /etc/inittab file, acting on instructions found there, and executing the relevant run command scripts. These scripts contain entries that initiate activities such as mounting and checking file systems, removing temporary files, initializing the clock daemon, initializing the network daemon, setting up printer spooling directories and daemons, enabling error logging, and performing other tasks specified within the scripts or in related directories. At the conclusion of these activities, the system is enabled and accessible to users.

The Digital UNIX operating system allows you to boot an alternate kernel. For example, if you cannot boot your system, you could boot /genvmunix to troubleshoot the problem with your

system. You could also boot an alternate kernel to test new drivers or to add options to the existing kernel.

Preparing to Boot the Installed System

As the system administrator, you set up or encounter various preboot or postshutdown states. This section describes and recommends procedures for preparing and initiating a reboot from a variety of system states. The states discussed here include the following:

- A powered-down system
- A powered-up, halted system
- A powered-up system in single-user mode
- A crashed system

Preparing to Boot a Powered-Down System

A system is powered down when the hardware (processor, devices, and peripherals) is turned off. Administrators power down the hardware periodically for routine maintenance or to configure new devices.

If you are preparing to reboot your system from a powered-down state, follow these steps:

1. Confirm that the hardware and all peripheral devices are connected. Refer to the operator's guide for your hardware for information and instructions for interpreting diagnostic output.
2. Power up the hardware and peripheral devices. Remember to power up all devices that you powered down earlier. Refer to the operator's manual or the hardware user's guide for instructions on starting your hardware and peripherals.
3. Confirm that the hardware completed its restart and diagnostic operations. Most hardware provides a diagnostic check as a routine part of its startup operation. Refer to the operator's manual for your hardware for information about your hardware's restart and diagnostic operations.
4. Wait for the console prompt (>>>). If you have enabled your system to boot automatically upon powerup, press the halt button to display the console prompt. Refer to the hardware operator's guide for the location of the halt button on your system.

5. Decide which startup mode you want to initiate:
    - If you have tasks you need to accomplish and want the system to restrict access to all users but root, plan to boot to single-user mode.
    - If you do not require single-user access and you want the system to initialize full functionality, plan to boot to one of the multiuser modes: multiuser without networking or multiuser with networking.
6. Enter the boot command that corresponds to the desired startup mode.

Preparing to Boot a Powered-Up, Halted System

When your machine is powered up and enabled but the processor is halted, the system is in console mode. For example, after you shut down the processor with the shutdown -h command or when you run the halt command, your system displays the console prompt (>>>). When the system displays the console prompt, follow these steps to prepare to boot your system:

1. Decide which startup mode you want to initiate:
    - If you have tasks you need to accomplish and you want the system to restrict access to all users but root, plan to boot to single-user mode.
    - If you do not require single-user access and you want the system to initialize full functionality, plan to boot to one of the multiuser modes: multiuser without networking or multiuser with networking.
2. Enter the boot command that corresponds to the desired startup mode.

Preparing to Transition from Single-User Mode

When your machine is powered up and enabled, the processor is running, and access is limited to root, the system is in single-user mode.

When the system displays the single-user prompt (#), follow these steps to prepare to go to multiuser mode:

1. Decide if you should continue in single-user mode or if you should go to multiuser mode:
    - If you have additional tasks that you need to perform and you want the system to deny access to all users but root, plan to continue in single-user mode.

- If you do not require single-user access, or if you have completed your tasks and you want the system to initialize full functionality, plan to go to one of the multiuser modes: multiuser without networking or multiuser with networking.

2. When you are ready to go to multiuser mode, press Ctrl/d.

Preparing to Boot a Crashed System

If your system crashes and is unable to recover automatically and reboot itself, follow these steps to prepare to boot the system:

1. Confirm that the hardware and all peripheral devices are connected.
2. Power up the hardware, if necessary. Always power up peripherals and devices before the processor.
3. Monitor the hardware restart and diagnostic operations. Refer to the operator's guide for your hardware for information and instructions for interpreting diagnostic output.
    - In the unlikely event that the diagnostic test indicates hardware failure, contact your Digital field representative. Because hardware damage is a serious problem, do not continue or try to bypass the defective hardware.
    - If you have enabled your system to boot automatically, press the halt button to display the console prompt. Refer to the hardware operator's guide for the location of the halt button on your system.
4. Decide which startup mode you want to initiate:
    - If you need to deny access to all users but root, plan to work in single-user mode. After a crash, it is wise to work initially in single-user mode. You should check all file systems thoroughly for inconsistencies and perform other postcrash operations before enabling system access to other users.
    - If you need to allow access to you and to all other users with login permission, plan to boot to one of the multiuser modes: multiuser without networking or multiuser with networking.
5. Enter the required boot command.

SHUTTING DOWN THE SYSTEM

The following sections describe the shutdown procedures and the recovery strategies that you use in both controlled and unexpected shutdowns. The first part discusses procedures for handling controlled shutdowns. The second part discusses guidelines and recommendations for handling and recovering from unexpected shutdowns.

There are several good reasons to stop the system in a controlled shutdown. For example:

- You need to upgrade your software or add new hardware to your configuration. You shut down the system to set up the new additions, make the necessary adjustments to your configuration files, and build a new kernel.
- You have been monitoring the hardware error log and have noticed repeated warnings. You suspect that your hardware may soon fail so you shut down the system and examine the problem.
- You notice that system performance is degrading rapidly. You check the system statistics and conclude that some changes to the system would improve performance. You shut down and tune the system.
- You notice signs of possible file system corruption. You shut down the system and run the fsck program to fix problems or to confirm that none exist.

In each of these and similar situations a variety of options are available to you. Regardless of how you decide to resolve the situation, your first step is to initiate a controlled shutdown of the system. There are practical and reasonable ways to shut down your system from single-user mode or multiuser mode. A system that has panicked or crashed presents you with a different set of circumstances than a system that has shut down in an orderly fashion. However, this chapter discusses orderly shutdowns only.

MAINTAINING LOGIN CONTROL

You also must prevent unauthorized logins to a system or the network, which you can do through password assignment and login control. All accounts on a system should have a password. A password is a simple authentication mechanism. An account without a password makes your entire network accessible to an intruder who guesses a user name. A strong password algorithm protects against brute force attacks. When a user logs in to a system,

the login command consults the appropriate database according to the information that is listed in the /etc/nsswitch.conf file. This file can include the following entries:

- files – Designates the /etc files on the local machine.
- nis – Designates the NIS database on the NIS master server.
- nisplus – Designates the NIS+ database on the NIS+ root server.
- ldap – Designates the LDAP directory service on the LDAP server.

The login command verifies the user name and password that were entered. If the user name is not in the password file, the login command denies access to the machine. If the password is not correct for the user name that was entered, the login command denies access to the machine. When the user supplies a valid user name and its corresponding password, the system grants the user access to the machine.

Managing Password Information

When users log in to a system, the users must enter both a user name and a password. Although logins are publicly known, passwords must be kept secret. Passwords should be known only to each user. You should ask your users to choose their passwords carefully, and users should change their passwords often.

Passwords are initially created when you set up a user account. To maintain security on user accounts, you can set up password aging to force users to routinely change their passwords. You can also disable a user account by locking the password.

Local Passwords

If your network uses /etc files, the password information is kept in the system's /etc/passwd and /etc/shadow files. The user name and other information are kept in the password file /etc/passwd, while the encrypted password itself is kept in a separate **shadow** file, /etc/shadow. This security measure prevents a user from gaining access to the encrypted passwords. While the /etc/passwd file is available to anyone who can log in to a machine, only superuser can read the /etc/shadow file. You can use the passwd command to change a user's password on a local system.

NIS and NIS+ Passwords

If your network uses NIS+, the password information is kept in the NIS+ database. Information in the NIS+ database can be protected by restricting access to authorized users. You can use the passwd command to change a user's password that is stored in a NIS+ database.

If your network uses NIS, the password information is kept in the NIS password map. NIS does not support password aging. You can use the passwd command to change a user's password that is stored in the NIS password map.

LDAP Passwords

The Solaris LDAP Naming Service stores the password information and the shadow information in the ou=people container of the LDAP directory tree. On the Solaris LDAP naming service client, you can use the passwd –r ldap command to change a user's password. The LDAP naming service stores the password in the LDAP repository.

RECOVERY FROM COMPUTER SYSTEM CRASH

The computer may crash suddenly. It crashes by a variety of reasons. Here are six common reasons:

1. Corrupted system registry files.
2. Disorganized files.
3. Malicious software.
4. Too little available memory.
5. Device driver.
6. Hardware problem like overheating.

How to recover files after your computer crashed?

Your computer slightly crash

Run safe mode

Safe mode is a diagnostic mode in the Windows-based computer. It can load the operating system with minimal options. If your computer slightly crashes, you can run safe mode.

Step 1. Start your computer, and press the F8 key repeatedly before showing the Windows logo

Step 2. Under the Advanced Boot Options window, select Safe Mode.

Step 3. Shutdown and restart the computer, then see if the problem is solved.

Do system restore in safe mode

System restore can restore your computer to an early date, including Windows files and settings. If your computer crash is due to bad software or drivers, you can do system restore in safe mode.

Step 1. Start your computer in safe mode.

Step 2. Summon the system restore window. You can click Start menu > All Programs > Accessories > System Tools > System Restore.

Step 3. Choose a restore point and click Next. In this window, you should tick the recommended restore option. But if the restore point is not working properly at that time, you still can tick Choose a different restore point.

Step 4. Confirm the settings are correct and click Finish.

Start your computer with Last known good configuration option. Last known good configuration is an option, which is available on the Boot Menu. If you change the settings like driver and your computer does not start up, you can start your computer with this option. And it does not delete anything.

Step 1. Enter the Advanced Boot Options window.

Step 2. Select Last Known Good Configuration option and press Enter.

Last Known Good Configuration

Step 3. Wait for your computer to start and check if the problem is solved.

If your computer is completely crash, the above ways will not work for you. Many users will turn to the data recovery software. At the same time, some users prefer to restore computer with a backup image created in advance. See below, I will show you both.

Using Data recovery software

To recover files after Windows crashed, you need USB cable, a storage device like USB flash drive and working computer.

Step 1. Take out your hard drive.

Step 2. Connect the crashed hard drive to a working computer via USB cable. If you can access the content on the hard drive, you should copy it to your USB flash hard drive. If not, you can employ data recovery software.

Note: You cannot install any data recovery software on the hard drive you want to recover files from. The operation might actually overwrite your data. Thus, you need to install the data recovery software on a different disk or computer.

Step 3. Download the free data recovery software TextDisk, and open it.
Step 4. Create a new log. You can click Create under this window.
Step 5. Select the drive you want to scan, and then click Proceed.
Step 6. Select the partition table type: Intel
Step 7. Choose the Advanced option to select a partition with NTFS or FAT32, then select Undelete at the bottom.
Step 8. Select files via : or a
step 9. Press c or C to copy files.
Step 10. Select a path to save the recovered files. And wait for the final result.

With the help of free data recovery software, you can restore your files. But most data recovery software cannot retrieve all kinds of data loss unless you have a comprehensive software

MANAGE DISK SPACE IN UNIX SYSTEM

The „du" Command Line

The „du" (**D**isk **U**sage) command line is a standard command under Unix and Linux. It is used to list the disk space used by files on a machine and crucial for disk space management on unix and linux systems. Several paramaters enable users to format and filter the results.

Using **du** is a good starting point when trying to clean up unsed disk space under Unix.

Many smaller machines run Microsoft Windows. There is a wide range of graphical disk space managers for Windows, but few will enable you to scan Linux and Unix servers without having to resort to Samba. The Enterprise Edition of SpaceObServer will scan Unix and Linux servers via SSH.

Using du

To analyze disk usage in a specific directory tree, simply enter the command line.

root@server:/# du

Adding **/*** will show all directories and files at the top level of the file system:

root@server:/# du -s /*

You can also specify which folder shall be scanned:

root@server:/# du -s /var

There are several ways to filter the information and gain the insights required to proceed.

Parameters

A wide range of parameters can be used to refine the results. Many flags can be used in combination. This is a (by no means complete) selection of useful arguments.

du -s

Display the disk space used by all files in the selected directory by adding **-s**. You cannot limit the folder depths while using this argument.

```
root@server:/# du -s /*
32      /allout.txt
6768    /bin
19624   /boot
0       /dev
5832    /etc
3098116 /home
4       /html
0       /initrd.img
127284  /lib
4       /lib64
16      /lost+found
20      /media
4       /mnt
4       /opt
0       /proc
30700   /root
2040    /run
5848    /sbin
4       /selinux
4       /srv
0       /sys
8       /tmp
819524  /usr
1022520 /var
0       /vmlinuz
```

MOUNT AND UNMOUNT FILE SYSTEM

.The mount command instructs the operating system that a file system is ready to use, and associates it with a particular point in the overall file system hierarchy (its mount point) and sets options relating to its access. Mounting makes file systems, files, directories, devices and special files available for use and available to the user. Its counterpart umount instructs the operating system that the file system should be disassociated from its mount point, making it no longer accessible and may be removed from the computer. It is important to umount a device before removing it since changes to files may have only partially been written and are completed as part of the umount.

The mount and umount commands require root user privilege to effect changes. Alternately, specific privileges to perform the corresponding action may have been previously granted by the root user. A file system can be defined as user mountable in the /etc/fstab file by the root user.

Description: mount

All files accessible in Unix, or a Unix-style system such as Linux, are arranged in one big tree: the file hierarchy, rooted at /. These files can be spread out over several devices. The mount command attaches a filesystem, located on some device or other, to the file tree. Conversely, the umount command will detach it again.

The standard form of the mount command is:

mount -t type device dir

This tells the kernel to attach the filesystem found on device (which is of type type) at the directory dir. The previous contents (if any), owner, and mode of dir become invisible, and as long as this filesystem remains mounted, the pathname dir refers to the root of the filesystem on device.

If only directory or device is given, for example:

mount /dir

then mount looks for a corresponding mountpoint (and then, if not found, for a corresponding device) entry in the /etc/fstab file, and attempts to mount it.

Description: umount

The umount command detaches the specified file system(s) from the file hierarchy. A file system is specified by giving the directory where it has been mounted. Giving the special device on which the file system lives may also work, but is an obsolete method, mainly because it will fail in case this device was mounted on more than one directory.

Note that a file system cannot be unmounted when it is busy - for example, when there are open files on it, or when some process has its working directory there, or when a swap file on it is in use. The offending process could even be umount itself - for example, umount opens the common C libraries, which in turn may open (for example) locale files, which, if they are located on the device in question, will prevent it from being unmounted. A "lazy" unmount attempts to unmount even if this conflict arises.

ADD AND REMOVE USERS IN UNIX SYSTEM

One of the most basic tasks that you should know how to do on a fresh Linux server is add and remove users. When you create a new system, you are often (such as on DigitalOcean Droplets) only given the **root** account by default. While running as the **root** user gives you a lot of power and flexibility, it is also dangerous and can be destructive. It is almost always a better idea to add an additional, unprivileged user to do common tasks. You also should create additional accounts for any other users you may have on your system. Each user should have a different account. You can still acquire administrator privileges when you need them through a mechanism called `sudo`. In this guide we will cover how to create user accounts, assign `sudo` privileges, and delete users.

How To Add a User

If you are signed in as the **root** user, you can create a new user at any time by typing:

- adduser newuser
-

If you are signed in as a non-root user who has been given sudo privileges, as demonstrated in the initial server setup guide, you can add a new user by typing:

- sudo adduser newuser
-

Either way, you will be asked a series of questions. The procedure will be:

- Assign and confirm a password for the new user
- Enter any additional information about the new user. This is entirely optional and can be skipped by hitting **Enter** if you don't wish to utilize these fields.
- Finally, you'll be asked to confirm that the information you provided was correct. Enter **Y** to continue.

Your new user is now ready for use! You can now log in using the password you set up.

How to Delete a User

In the event that you no longer need a user, it is best to delete the old account.

You can delete the user itself, without deleting any of their files, by typing this as root:

- deluser newuser
-

If you are signed in as another non-root user with sudo privileges, you could instead type:

- sudo deluser newuser
-

If, instead, you want to delete the user's home directory when the user is deleted, you can issue the following command as root:

- deluser --remove-home newuser
-

If you're running this as a non-root user with sudo privileges, you would instead type:

- sudo deluser --remove-home newuser

CHANGING GROUP AND PASSWORD

Both Linux and UNIX-like operating systems use the passwd command to change user password. The passwd is used to update a user's authentication token (password) stored in /etc/shadow file. The passwd change passwords for user and group accounts. A normal user may only change the password for his/her own account, the super user (or root) may change the password for any account. The administrator of a group may change the password for the group. passwd also changes account information, such as the full name of the user, user login shell, or password expiry date and interval.

LINUX SET USER PASSWORD

Type following passwd command to change your own password:

```
$ passwd
```

Sample Outputs:

```
Changing password for vivek
(current) UNIX password:
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

The user is first prompted for his/her old password if one is present. 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 forgotten passwords may be changed. A new password is tested for complexity. As a general guideline, passwords should consist of 10 to 20 characters including one or more from each of following sets:

1. Lower case alphabetics
2. Upper case alphabetics
3. Digits 0 thru 9
4. Punctuation marks/spacial characters

Linux change password for other user account

You need to login as the root user, type the following command to change password for user vivek:

```
# passwd vivek
```

OR

```
$ sudo passwd vivek
```

Sample putput:

```
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

MANITAIN SECURITY IN UNIX SYSTEM

The different vendors of UNIX and Linux make it difficult to write one comprehensive, step by step, guide for securing your UNIX/Linux system. We can, however, generalize some basic concepts about securing the environment that you can customize to your specific installation to produce a more secure operating system. Following these guidelines will help reduce your chance of compromise, make the forensics easier if you do get compromised, and hopefully get your system back online faster.

Step 1: Pick a good, supported operating system.

IS&T currently supports:

- BU Linux (CentOS) on x86
- MacOS X

In addition, we have a lot of experience with IBM's AIX, and SGI's IRIX operating systems that we can share with you. While you are free to pick the operating system of your choice that best fits your needs, sticking with an operating system that IS&T can support will likely result in a more secure environment. If you are picking from outside our list make sure that you select on operating system with a good track record for supplying security fixes in a timely fashion and join their mailing lists so you can be sure to know about critical patches when they are released.

Step 2: Stay Current with Patches

Nearly every system compromise is the result of an unpatched system. It is critical that you not only get all the current patches when you install your system but that you maintain the current patch level. This may mean scheduling routine and emergency downtime to install new versions of the kernel and other core components when the patches can't be applied live. IS&T has provided tools for Solaris and BU Linux to help with automating patch installation.

Third party applications must also be kept secure. Numerous compromises start with an old version of a web server or web application, a forgotten SSH server or overlooked sendmail. When these components are part of the operating system the vendor may help you keep current, but when you start adding third party services to your host, you must make a procedure by which you ensure you stay current. This is increasingly true with web applications which have become a very popular target in the last few years.

Step 3: Use a Firewall

Nothing protects a service from compromise like a well configured firewall! Sure, your public web server has to be accept traffic from everything on the Internet, but do you really have users that need to SSH in from other continents? Some systems do, but for most systems it is totally unnecessary to SSH into the system if you're not on campus (or connected to the VPN server from off-campus!) Configuring a software firewall is easy and supported on most platforms. See our host-based firewalls section for more information.

Step 4: Use File Integrity Monitoring and Change Auditing

File Integrity Monitoring and Change Auditing is using software to detect changes to the contents or attributes of your filesystems. While files in home directories and temp space change constantly, many sections of your system are static such as /usr or behave in predictable ways like /var. By monitoring these sections of the filesystem, you can easily detect when something changes and determine if it was the work of a patch or systems administrator, or if it is indicative of a compromise. Tripwire is the most well-known public offering, but Boston University has developed its own application known as Baseline, which is available to the BU community.

Step 5: Keep your clocks in sync!

If you use Kerberos authentication it is essential that you keep your system clock synchronized to our central servers, but even if you don't it's still a good idea. It is very easy to do — just configure your system's network time protocol (ntp) subsystem to synchronize against the BU time servers:

ntp1.bu.edu

ntp2.bu.edu

ntp3.bu.edu

If you use BU Linux or installed Solaris from our Jumsptart servers, NTP should have been properly configured for you by default.

Step 6: Copy your logs to our Central Log Server

Every time something interesting happens on your servers the system records it via the syslog facility to log file located (generally) in /var/log. When a system gets compromised, the first thing an intruder often does is erase those logs so you don't see all the interesting things that lead up to the compromise. You can thwart this by storing your logs on more than one system. Syslog provides a mechanism for forwarding the log data to another host, and IS&T runs a log server that will gladly accept a copy from you. We manage the logs, store them, back them up, and do all the necessary steps to make sure they are available if and when you need them. Access to these logs is limited to the smallest set of administrators necessary for the log server and are used internally only to look for signs of compromise. Setting up logging is easy to do, you only need to add one line to your syslog.conf and restarting syslog. See our complete instructions for more details on configuring remote logging.

Step 7: Follow our Global UID System

MONITORING UNIX SYSTEM SECURITY

Securing any computer system involves many steps, including:

- Knowing how the system may be attacked
- Knowing how to protect against these attacks

- Monitoring your system
- Teaching users how to be "safe" and getting them to work with you
- Knowing what to do when you detect you've been hit

Knowing how the system may be attacked

Detecting an attack

Common methods of attack include:

- System crashes
    - Repeat crashes usually hardware related
    - Read log files/console for panic reasons
    - Any patterns (e.g., time of day) should raise suspicion
- New accounts
    - Especially with UID 0
- New files
    - With setuid/setgid bits or root ownership
    - Hidden directories (e.g., "..." or with control characters in the name, such as backspace)
- Modified file
    - System programs (e.g., login, sh, csh, ps)
    - Configuration files (e.g., /etc/rc, /etc/aliases, /etc/hosts.equiv)
    - User files (e.g., .login, .cshrc, .history, .rhosts)
- Poor Performance
    - May indicate password cracking or packet sniffing programs
    - Programs to use: ps, iostat, vmstat, pstat, sar, netstat
- Denial of service
    - Exhaustion of finite resources
        - File space on partitions (e.g., filling /tmp)
        - inode table
        - process table
        - open files
        - virutal memory

        - network services
    - Destruction of data
        - Deleting user files/system configuration files/system programs.

ACCOUNTING SYSTEM USAGE AND BILLING

Accounting reports created by the system accounting service present the *NIX administrator with the information to assess current resource assignments, set resource limits and quotas, and predict future resource requirements. This information is also valuable to the forensic analyst and allows for the monitoring of system resourcing. This data can be a means of finding what processes and resources have been used and by which user. When the system accounting has been enabled on a *NIX system, the collection of statistical data will begin when the system starts or a least from the moment that the accounting service is initiated. The standard data collected by system accounting will include the following categories:

- Connect session statistics
- Disk space utilization
- Printer use
- Process use

The accounting system process starts with the collection of statistical data from which summary reports can be created. These reports can assist in system performance analysis and offer the criteria necessary to establish an impartial customer charge back billing system or many other functions related to the monitoring of the system.

Connect Session Statistics

Connect session statistics allow an organization to bill, track or charge access based on the tangible connect time. Connect-session accounting data, associated with user login and logout, is composed by the init and login commands. When a user logs into the *NIX system, the login program makes an entry in the "wtmp" file. This file will contain the following user information:

- Date of login/logout
- Time of login/logout

- Terminal port
- User name

This data may be can be utilized in the production of reports containing information valuable to the computer forensic investigator, system security tester and system administrator. Some of the information that can be extracted includes:

- Connect time seconds used,
- Date and starting time of connect session,
- Device address of connect session,
- Login name,
- Number of prime connect time seconds used,
- Number of nonprime connect time seconds used,
- Number of seconds elapsed from Jan 01st 1970 to the connect-session start time,
- Process Usage,
- User ID (UID) associated with the connect-session.

It is also possible to gather statistics about individual processes using system accounting. Some areas that may be collected include:

- Elapsed time and processor time consumed by the process,
- First eight characters of the name of the command,
- I/O (Input/output) statistics,
- Memory usage,
- Number of characters transferred,
- Number of disk blocks read or written by the process ,
- User and group numbers under which the process runs.

Many *NIX systems maintain statistical information in a "pacct" or process account database or accounting file. This database is commonly found in the "/var/adm/pacct" file, but like many *NIX log files, this will vary from system to system. The accounting file used by many of the system and process accounting commands. When a process terminates, the kernel writes information explicit to the particular process into the "pacct" file. This file consists of the following information:

- Command used to start the process
- Process execution time
- Process owner's user ID

When system accounting is installed and running on a *NIX system, commands to display, report, and summarize process information will be available. Commands such as "*ckpacct*" can be used by the administrator or system security tester to ensure that the process accounting file ("*pacct*") remains under a set size and thus is stopped from either growing too large or possibly impacting system performance in other ways.

Disk Space Utilization

System accounting provides the ability for the system security tester to receive information concerning the disk utilization of the users. As it is possible to restrict users to a specified disk usage limit, the system security tester may need to validate usage through a disk quota system. This may be monitored and tested to ensure users are adhering to limits. This allows an unwary client to be charged fees that are correctly associated with another account. Disk usage commands perform three basic functions:

- Collect disk usage by filesystem,
- Gather disk statistics and maintain them in a format that may be used by other system accounting commands for further reporting,
- Report disk usage by user.

Printer Usage

Printer usage data is stored in the "qacct" file (this is commonly located in "/var/adm/qacct" on many systems though this varies). The "qacct" file is created using an ASCII format. The qdaemon writes ASCII data to the "qacct" file following the completion of a print job. This file records printer queue data from each print session and should at a minimum contain the following fields:

- User Name
- User number(UID)
- Number of pages printed

Automatic Accounting Commands

To accumulate accounting data, the *NIX system needs to have a number of command entries installed into the "crontab" file (e.g. the "/var/spool/cron/crontabs/adm" file on many *NIX'es but this will change from system to system). The cron file of the adm user is configured to own the whole of the accounting files and processes. These commands have been designed to be run using cron in a batch mode. It is still possible to execute these commands manually from a command line or script.

- ckpacct Controls the size of the /var/adm/pacct file. When the /var/adm/pacct file grows larger than a specified number of blocks (default = 1000 blocks), it turns off accounting and moves the file off to a location equal to /var/adm/pacctx (x is the number of the file). Then ckpacct creates a new /var/adm/pacct for statistic storage. When the amount of free space on the filesystem falls below a designated threshold (default = 500 blocks), ckpacct automatically turns off process accounting. Once the free space exceeds the threshold, ckpacct restarts process accounting.
- dodisk Dodisk produces disk usage accounting records by using the diskusg, acctdusg, and acctdisk commands. By default, dodisk creates disk accounting records on the special files. These special filenames are usually maintained in "/etc/fstab" or "/etc/filesystems".
- monacct Uses the daily reports created by the commands above to produce monthly summary reports.
- runacct Maintains the daily accounting procedures. This command works with the acctmerg command to produce the daily summary report files sorted by user name.
- sa1 System accounting data is collected and maintained in binary format in the file /var/adm/sa/sa{dd}, where {dd} is the day of the month.
- sa2 The sa2 command removes reports from the "../sa/sa{dd}" file that have been there over a week. It is also used to write a daily summary report of system activity to the "../sa/sa{dd}" file

SETTING UP REMOTE COMMUNICATION IN UNIX SYSTEM

Accessing machines remotely became a necessity a long time ago and we can barely imagine how it would be if we couldn't control computers from remote locations. There are many ways to establish a connection with a remote machine depending on the operating system you are running, but the two most used protocols are:

- Secure Shell (SSH) for Linux-based machines
- Remote Desktop Protocol (RDP) for Windows-based machines

The two protocols use the client and server applications to establish a remote connection. These tools allow you to gain access and remotely manage other computers, transfer files, and do virtually anything you can do while physically sitting in front of the machine.

Prerequisites

Before you can **establish a secure remote desktop protocol** with a remote machine, there are a few basic requirements to meet:

- The remote computer must be turned on at all times and have a network connection.
- The client and server applications need to be installed and enabled.
- You need the IP address or the name of the remote machine you want to connect to.
- You need to have the necessary permissions to access the remote computer.
- Firewall settings need to allow the remote connection.

INSTALL PRINTERS AND PERIPHERAL DEVICES

The peripheral devices are those devices which are connected to the computer and it helps the computer function. These devices contain both the input devices, which are used to give command to the computer and the output devices, which help computer showing the result to the user. The computer works with many peripheral devices so it's important to know that how much these devices can be installed in the computer and how can a persona configure

them. Following are the devices which are used by the computer and the ways they can be configured;

Input devices:

The device which are connected to computer and they are used to send the data to the computer internally, are known as the input devices. Following are some important input devices;

Mouse: Mouse is the one of the most common device that one would find in a desktop. It is normally plugged with the USB connection. It would be connected through the PS/2 port which is located on the back of the computer. In older system, one might find the serial port which was used to move the move around. The old mouse contained ball inside of it which helped it move. But nor the trend has changed the mouse that one uses contains some LED light. They are called the laser mouse. The LED light now emits out of the mouse sensing that where the mouse is being moved. The installation of mouse is easy, one can just plug the mouse and the drivers would be searched by windows itself. The mouse can be configured from the control panel where the speed of it can be improved.

Keyboard: it is the famous device that is being used today and it is connected through a USB connection at the back of the computer. Old keyboards used to have the Mini DIN PS/2 connection with them; one would have to go through the manufacturer's website to know that what are the drivers for the keyboard and then can get them installed when he connects the keyboard with the computer.

Touch screen: Touch screen is becoming common as well. They are normally used on the mobile phones and other smart devices. Also, the tablets also contain this feature. Now it is even common to find a computer's large screen which works with the touch screen system. It makes it very easy to type and navigate and one can bring the keyboard on the screen and type it there as well hence one doesn't have to install the keyboard. The configurations and the installation of the touch screen are also done in a display when it is bought.

Scanner: At a time, the fax machines were widely used by the people and till now, they use it. But now the tables have turned and people tend to use the scanners. The scanners have the ability to turn the paper's image into a digital one hence sending the data from one place to another while keeping the same paper with us is now an easy task and the data can be sent more easily through the internet. The scanners are now used a lot and one can simply scan a paper and save it in any format he wants to.

Barcode reader: If one goes to some super market or the retail environment, one would see the barcode. It's a machine that is used to read the code which is printed at the back of the products. The connection of the bar code is normally made through the USB connection. Some old versions of the bar codes also use the PS.2 connections to get connected to the computer. The barcodes use the laser technology to read the barcode. The light comes and goes when the code is exposed to the barcode reader. This eye can directly damage one's eye as well so one has to be careful while using this machine. The barcode contains a specific driver when it is connected, the driver is detected by the windows and hence it can work properly.

Microphone: the putting of voice in our computer is very common. It is done by the chat or it is even done for the voice reorganization software's which are available in the market. Microphones on not only at the headset, but they can be found separately and are there in the laptops as well. They also have the specific drivers and the configuration can be made through the control panel.

Biometric devices: these devices are used as the security tools. They are mostly used on some portable devices which tend to go away from one. If one tries to use the laptop, he might find there is a finger print reader there and hence it is protected since only the administrator who has set it up can log in to the computer.

Game pads: Game pad is the wonderful thing for the gamers. It allows the gamers to have some control over the gaming experience.

Joysticks: it is the gaming input which is widely used. It is a stick which has many buttons on it.

Digitizer: the digitalizing pad has some specialized pins that allow the user to draw on it very carefully and accurately. They are for those who have got some artistic abilities and they are installed with the driver.

Output devices:

Output devices are those devices which do not send the data into the computer, in fact through these devices, the computer communicates with the user. They contain the devices like displays, printers etc. Since they are very vital part of the computer system, so one should have enough knowledge about them. Some of these output devices are as follows;

Printers: if one has the data in the computer, it is obvious that he has to take it out of the computer. There are several ways for it and one of them is the painters. These printers are available not only in the office environment and in some retail environment, but at the homes as well. One might need the printer to take out the prints of the important information he finds on the internet or the assignment which he creates. At office, many important documents are sent and received in the digitalized format and they have to be converted to the hard copy so they can be read easily. At the shops, one might find the painters which are used for printing out the receipts which are handed over to the customers. Hence printers are used in every aspect of the busy life. Printers have various types and the laser printers are the most famous type since they print really good. For printers there is much software and one must install the appropriate drivers to get them work. The configuration of the pinner can be done manually once it is connected to the computer.

Speakers: many of the features which are used by us are mainly related to the audio. Doesn't matter whether one listens to the podcasts or plays music, He would need the audio to be heard. Even while watching the movie or some educational video, hearing the voice is really important part and for that purpose, one needs the speakers. The laptop contains the right left speakers which are located at these both sides. There is also a subwoofer which can be

used for the mixing of the voice. Also, one can have some advanced computer speaker which can be plugged into the computer externally for the sake of better quality of the sound. The laptops have various options which ate provided to user so that he can hear the sound with different options.

Display devices: one of the most integral parts of the PC is the display devices which are used. The human beings need to see that what's going on the operating system and to get in touch with the interface, they must have some display device which can show them the appropriate amount of data. Normally the display devices re pretty thin and one has many options for the display devices and they vary in prices range as well. These device can be both non touchable and the touchscreen. These display devices are not just connected to the computers; instead they are connected to the mobile phones as well. Also the Tablets contain some display devices as well. These displays show us what is exactly going on there and how we can control the computer.

UNIT – 4 (Question Bank)

1. Write a shell program in UNIX.
2. Explain the WWW server under UNIX in detail.
3. What is system programming? Explain its evolution history in detail.
4. Explain shell programming and also, focus interactive shell scripts.
5. Define Shell script. How does it differ from a program coded in a specific language?
6. Explain with suitable example the networking commands used in UNIX.
7. Write a note on wild card characters.
8. Write notes on the following:
   (a) DNS
   (b) Setting up of WWW servers under UNIX

UNIT – 4 (NOTES)

Shell programming as a Language

What is shell?

Shell is a UNIX term for the interactive user interface with an operating system. The shell is the layer of programming that understands and executes the commands a user enters. In some systems, the shell is called a command interpreter. A shell usually implies an interface with a command syntax (think of the DOS operating system and its "C:>" prompts and user commands such as "dir" and "edit"). As the outer layer of an operating system, a shell can be contrasted with the kernel, the operating system's inmost layer or core of services.

Following important point to understand shell.

- "Shell" is a program, which facilitates the interaction between the user and operating system (kernel). There are many shells available, like sh, bash, csh, zsh...etc.

- "Shell" scripting is a way of automating things, in the form of collection of commands. The control of execution is steered by the predefined control statements.
- "Bash shell" scripting is, a kind of shell scripting only. You can say, its a subset of "shell" scripting.
- "sh" is the original (Bourne) shell, having its root from the old Unix. "bash", is "Bourne Again SHell", which is rewritten "sh".
- Bash is the most widely used shell. It comes with Linux by default, having backward compatibility with sh (though sh is also there). But, you can choose any shell you want.

Category of Shell (or we can say most of the O.S Shell falling category)

- Command-line shells
- Graphical shells

What is Command-line shells?

A command-line interface (CLI) is an operating system shell that uses alphanumeric characters typed on a keyboard to provide instructions and data to the operating system, interactively. For example, a teletypewriter can send codes representing keystrokes to a command interpreter program running on the computer; the command interpreter parses the sequence of keystrokes and responds with an error message if it cannot recognize the sequence of characters, or it may carry out some other program action such as loading an application program, listing files, logging in a user and many others.

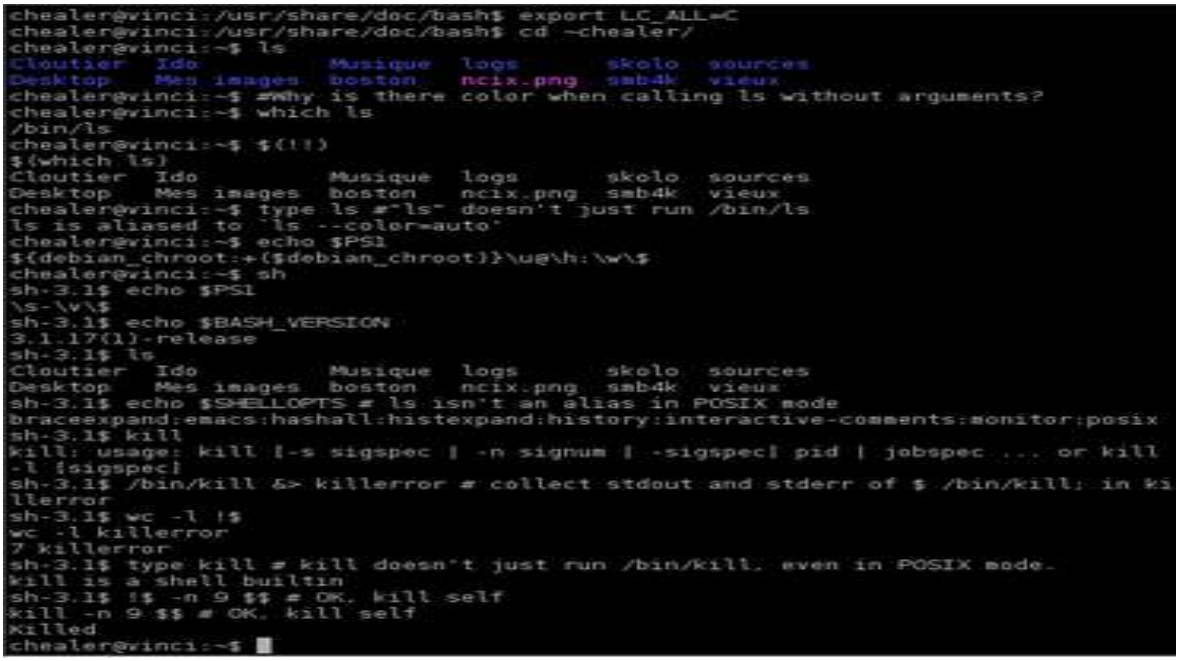


Fig: Bash, a widely adopted Unix shell.

Operating systems such as UNIX have a large variety of shell programs with different commands, syntax and capabilities.

- Some operating systems had only a single style of command interface
- commodity operating systems such as MS-DOS came with a standard command interface
- Application programs may also implement a command-line interface. Like telnet.

Graphical shell

A graphical user interface (GUI) provides means for manipulating programs graphically, by allowing for operations such as opening, closing, moving and resizing windows, as well as switching focus between windows. Graphical shells may be included with desktop environments or come separately, even as a set of loosely coupled utilities. Most graphical user interfaces develop the metaphor of an "electronic desktop", where data files are represented as if they were paper documents on a desk, and application programs similarly have graphical representations instead of being invoked by command names.

Shell Types:

In UNIX there are two major types of shells:

1. The Bourne shell. If you are using a Bourne-type shell, the default prompt is the $ character.
2. The C shell. If you are using a C-type shell, the default prompt is the % character.

There are again various subcategories for Bourne Shell which are listed as follows:

- Bourne shell (sh)
- Korn shell (ksh)
- Bourne Again shell (bash)
- POSIX shell (sh)

The different C-type shells follow:

- C shell (csh)
- TENEX/TOPS C shell (tcsh)

The original UNIX shell was written in the mid-1970s by Stephen R. Bourne while he was at AT&T Bell Labs in New Jersey.

Program execution in shell.

Steps to write and execute a script

- Open the terminal. Go to the directory where you want to create your script.
- Create a file with **.sh** extension.
- Write the script in the file using an editor.
- Make the script executable with command **chmod +x** <**fileName**>.
- Run the script using ./<**fileName**>.

Note: In the last step you have to mention the path of the script if your script is in other directory.

Hello World script

Here we'll write a simple programme for Hello World.

First of all, create a simple script in any editor or with echo. Then we'll make it executable with **chmod +x** command. To find the script you have to type the script path for the shell.

Look at the above snapshot, script **echo Hello World** is created with echo command as **hello_world.** Now command **chmod +x hello_world** is passed to make it executable. We have given the command. /**hello_world** to mention the hello_world path. And output is displayed.

Wild card characters

When we need to search for anything using shell commands then we need to define a pattern for searching. Wildcard characters are used to define the pattern for searching or matching

text on string data in the bash shell. Another common use of wildcard characters is to create regular expressions.

Wildcard characters:

The three main wildcard characters are,

- Star or Asterisk (*)
- Question mark (?)
- Square brackets ([])

Asterisk (*) is used to search for particular character(s) for zero or more times. Question mark (?) is used to search for a fixed number of characters where each question mark (?) indicates each character. Square brackets are used to match with the characters of a defined range or a group of characters. The uses of these characters are shown in the next part of this tutorial.

Use of asterisk (*)

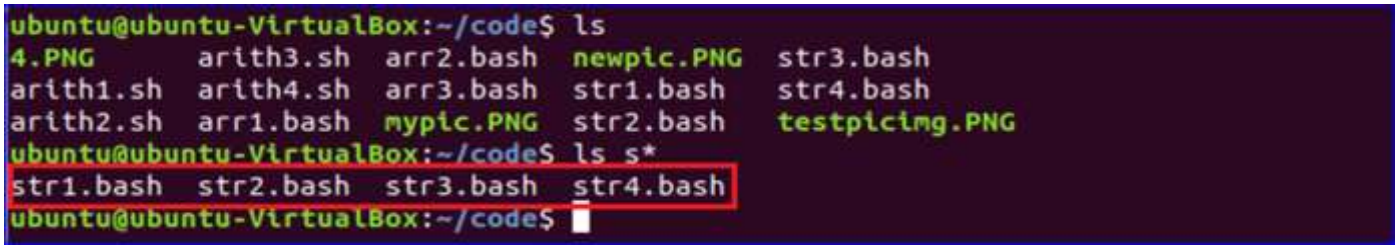
Asterisk (*) can be used in various ways with shell commands for searching files. Different use of asterisk (*) are shown in the following examples.

Example – 1: Searching specific file with filename and '*'

'ls' command is used to find out the list of files and folders of the current directory. **'ls s*'** command will search all files whose name start with **'s'**

$ ls

$ ls s*



Example – 2: Searching file with particular extension and '*'

You can search any file by using asterisk (*) and the file extension. If you want to search all files with **'.sh'** extension from the current directory then run the following command from the terminal. Here, the filename can be any character(s) and any number of characters.

$ ls *.sh

You can also search files of different extensions by using aterisk (*). The following command will search any files with extension **'.bash'** or **'.PNG'**

$ ls *.bash *.PNG

```
ubuntu@ubuntu-VirtualBox:~/code$ ls
4.PNG       arith3.sh  arr2.bash  newpic.PNG  str3.bash
arith1.sh   arith4.sh  arr3.bash  str1.bash   str4.bash
arith2.sh   arr1.bash  mypic.PNG  str2.bash   testpicimg.PNG
ubuntu@ubuntu-VirtualBox:~/code$ ls *.sh
arith1.sh  arith2.sh  arith3.sh  arith4.sh
ubuntu@ubuntu-VirtualBox:~/code$ ls *.bash *.PNG
4.PNG       arr2.bash  mypic.PNG   str1.bash  str3.bash  testpicimg.PNG
arr1.bash   arr3.bash  newpic.PNG  str2.bash  str4.bash
ubuntu@ubuntu-VirtualBox:~/code$
```

Example – 3: Removing file by partial match and '*'

You can use asterisk (*) for matching any filename partially. The following command will remove the file which contains **'img'** word in any part of the filename.

$ ls

$ rm *img*

$ ls

```
ubuntu@ubuntu-VirtualBox:~/code$ ls
4.PNG       arith3.sh  arr2.bash  newpic.PNG  str3.bash
arith1.sh   arith4.sh  arr3.bash  str1.bash   str4.bash
arith2.sh   arr1.bash  mypic.PNG  str2.bash   testpicimg.PNG
ubuntu@ubuntu-VirtualBox:~/code$ rm *img*
ubuntu@ubuntu-VirtualBox:~/code$ ls
4.PNG       arith2.sh  arith4.sh  arr2.bash  mypic.PNG   str1.bash  str3.bash
arith1.sh   arith3.sh  arr1.bash  arr3.bash  newpic.PNG  str2.bash  str4.bash
ubuntu@ubuntu-VirtualBox:~/code$
```

Use of question mark (?)

When you know the exact numbers of characters that you want to search then question mark (?) wildcard can be used. The following examples show the different use of question mark (?) wildcard.

Example – 1: Searching file with filename and '?'

Suppose, the file extension, the total number of characters of a file and some characters of the file are known, then you can use this wildcard to search the file. The command will search the file which has the extension **'.PNG'**, the word 'pic' is at the end of the filename and filename is six character long.

$ ls

$ ls ???pic.PNG

```
ubuntu@ubuntu-VirtualBox:~/code$ ls
4.PNG       arith2.sh  arith4.sh  arr2.bash  mypic.PNG   str1.bash  str3.bash
arith1.sh   arith3.sh  arr1.bash  arr3.bash  newpic.PNG  str2.bash  str4.bash
ubuntu@ubuntu-VirtualBox:~/code$ ls ???pic.PNG
newpic.PNG
ubuntu@ubuntu-VirtualBox:~/code$
```

Example -2: Searching file with extension and '?'

Suppose, you know the filename and the total number of characters of the file extension then you can use question mark (?) wildcard to search the file. The following command will search the file with filename **'mypic'** and the extension is three characters long.

$ ls

$ ls mypic.???

```
ubuntu@ubuntu-VirtualBox:~/code$ ls
4.PNG      arith2.sh  arith4.sh  arr2.bash  mypic.PNG   str1.bash  str3.bash
arith1.sh  arith3.sh  arr1.bash  arr3.bash  newpic.PNG  str2.bash  str4.bash
ubuntu@ubuntu-VirtualBox:~/code$ ls mypic.???
mypic.PNG
ubuntu@ubuntu-VirtualBox:~/code$
```

Use of square brackets ([])

Different range of characters or group of characters can be used within square brackets ([]) for searching files based on the range.

Example -1: Search files of any extension with two ranges values

The following command will search any file whose name contains any character within **'p-z'** and any digit within **'0-9'** and the file extension can be any characters.

$ ls

$ ls [p-z0-9]*.*

```
ubuntu@ubuntu-VirtualBox:~/code$ ls
4.PNG      arith2.sh  arith4.sh  arr2.bash  mypic.PNG   str1.bash  str3.bash
arith1.sh  arith3.sh  arr1.bash  arr3.bash  newpic.PNG  str2.bash  str4.bash
ubuntu@ubuntu-VirtualBox:~/code$ ls [p-z0-9]*.*
4.PNG  str1.bash  str2.bash  str3.bash  str4.bash
ubuntu@ubuntu-VirtualBox:~/code$
```

The basic use of wildcard characters is shown in this tutorial. You can also use wildcard characters for creating any regular expression pattern for doing validation work.

Different type of statements in shell;

- if-else-if statement
- Looping statement

- Switch statement
- Arithmetic-based statements
- String based statements

Conditional Statements | Shell Script

Conditional Statements: There are total 5 conditional statements which can be used in bash programming

1. if statement
2. if-else statement
3. if..elif..else..fi statement (Else If ladder)
4. if..then..else..if..then..fi..fi..(Nested if)
5. switch statement

Their description with syntax is as follows:

if statement

This block will process if specified condition is true.

Syntax:

```
if [ expression ]
then
  statement
fi
```

if-else statement

If specified condition is not true in if part then else part will be execute.

Syntax

```
if [ expression ]
```

```
then
  statement1
else
  statement2
fi
```

if..elif..else..fi statement (Else If ladder)

To use multiple conditions in one if-else block, then elif keyword is used in shell. If expression1 is true then it executes statement 1 and 2, and this process continues. If none of the condition is true then it processes else part.

Syntax

```
if [ expression1 ]
then
  statement1
  statement2
  .
  .
elif [ expression2 ]
then
  statement3
  statement4
  .
  .
else
  statement5
fi
```

if then else if then if if (Nested if)

Nested if-else block can be used when, one condition is satisfies then it again checks another

condition. In the syntax, if expression1 is false then it processes else part, and again expression2 will be check.

Syntax:

```
if [ expression1 ]
then
  statement1
  statement2
  .
else
  if [ expression2 ]
  then
    statement3
    .
  fi
fi
```

Switch statement

Case statement works as a switch statement if specified value match with the pattern then it will execute a block of that particular pattern. When a match is found all of the associated statements until the double semicolon (;;) is executed.

A case will be terminated when the last command is executed. If there is no match, the exit status of the case is zero.

Syntax:

```
case  in
  Pattern 1) Statement 1;;
  Pattern n) Statement n;;
Esac
```

Example Programs

Implementing if **statement**

filter_none

brightness_4

```
#Initializing two variables
a=10
b=20

#Check whether they are equal
if [ $a == $b ]
then
    echo "a is equal to b"
fi

#Check whether they are not equal
if [ $a != $b ]
then
    echo "a is not equal to b"
fi
```

Output

$bash -f main.sh

a is not equal to b

Example 2:

Implementing if.else statement

filter_none

brightness_4

```
#Initializing two variables
```

```
a=20
b=20

if [ $a == $b ]
then
    #If they are equal then print this
    echo "a is equal to b"
else
    #else print this
    echo "a is not equal to b"
fi
```

Output

$bash -f main.sh

a is equal to b

Example 3:

Implementing switch statement

filter_none

brightness_4

```
CARS="bmw"

#Pass the variable in string
case "$CARS" in
    #case 1
    "mercedes") echo "Headquarters - Affalterbach, Germany" ;;

    #case 2
    "audi") echo "Headquarters - Ingolstadt, Germany" ;;

    #case 3
    "bmw") echo "Headquarters - Chennai, Tamil Nadu, India" ;;
```

```
esac
```

Output

$bash -f main.sh

Headquarters - Chennai, Tamil Nadu, India.

Note: Shell scripting is a case-sensitive language, which means proper syntax has to be followed while writing the scripts.

Looping Statements

Looping Statements in Shell Scripting: There are total 2 looping statements which can be used in bash programming

- while statement
- for statement

To alter the flow of loop statements, two commands are used they are,

- break
- continue

Their descriptions and syntax are as follows

- **while statement**

  Here command is evaluated and based on the result loop will executed, if command raise to false then loop will be terminated.

Syntax

- while command
- do
- Statement to be executed

for statement

The for loop operate on lists of items. It repeats a set of commands for every item in a list. Here var is the name of a variable and word1 to word N are sequences of characters separated by spaces (words). Each time the for loop executes, the value of the variable var is set to the next word in the list of words, word1 to wordN.

Syntax

- for var in word1 word2 ...wordn
- do
- Statement to be executed

done

Example Programs

Example 1:

Implementing for loop with break statement

filter_none

brightness_4

```
#Start of for loop
for a in 1 2 3 4 5 6 7 8 9 10
do
    # if a is equal to 5 break the loop
    if [ $a == 5 ]
    then
        break
    fi
    # Print the value
    echo "Iteration no $a"
done
```

Output

$bash -f main.sh

Iteration no 1

Iteration no 2

Iteration no 3

Iteration no 4

Example 2:

Implementing for loop with continue statement

filter_none

brightness_4

```
for a in 1 2 3 4 5 6 7 8 9 10
do
    # if a = 5 then continue the loop and
    # don't move to line 8
    if [ $a == 5 ]
    then
        continue
    fi
    echo "Iteration no $a"
done
```

Output

$bash -f main.sh

Iteration no 1

Iteration no 2

Iteration no 3

Iteration no 4

Iteration no 6

Iteration no 7

Iteration no 8

Iteration no 9

Iteration no 10

Example 3:

Implementing while loop

filter_none

brightness_4

```
a=0
# -lt is less than operator

#Iterate the loop until a less than 10
while [ $a -lt 10 ]
do
    # Print the values
    echo $a

    # increment the value
    a=`expr $a + 1`
done
```

Output:

$bash -f main.sh

0

1

2

3

4

5

6

7

8

9

Note: Shell scripting is a case-sensitive language, which means proper syntax has to be followed while writing the scripts.

Switch Statement

The switch statement is another kind of conditional statement in shell scripting. It allows a programmer to compare several values against a variable in an easier way compared to if conditional statement. The syntax for switch statement is:

```
case  in
<pattern1>)
   ##series of code for pattern1
   ;;
<pattern2>)
   ##series of code for pattern2
   ;;
<patternN>)
   ##series of code for patternN
   ;;
*)
   ##default statements
esac
```

The pattern is the possible value of the variable. Each pattern is separated with a double semi-colon which serves as a break statement of the pattern.

```
#!/bin/bash
clear
read -p "Integer1: " int1
read -p "Integer2: " int2
echo "======================"
printf "Menu: \n[a] Addition\n[b]Subtraction\n[c]Multiplication\n[d]Division\n"
echo "======================"
read -p "Your choice: " choice
res=0
case $choice in
a)
   res=$((int1+int2))
;;
b)
   res=$((int1-int2))
;;
c)
   res=$((int1*int2))
;;
d)
   res=$((int1/int2))
;;
*)
   echo "Invalid input"
esac
echo "The result is: " $res
```

Arithmetic-based Condition

The shell provides several ways in declaring an arithmetic-based condition. First is by using mnemonics that can be used together with the old-style single-bracket syntax and the other

one is using mathematics-friendly symbols that can be used together with the double parenthesis

Below is the list of available mnemonics for arithmetic-based conditional statements in shell:

OperatorUsage/Description

-eq Equal
-ge Greater Than or Equal
-gt Greater Than
-le Less Than or Equal
-lt Less Than
-ne Not Equal

Let's create a script that accepts an integer from the user and determines whether the integer is a zero, negative, odd or an even number.

```
#!/bin/bash

read -p "Enter an integer: " int1
 if [ $int1 -eq 0 ]
 then
    echo "Zero"
 elif [ $int1 -lt 0 ]
 then
    echo "Negative"
 else
    if [ $((int1%2)) -eq 0 ]
    then
```

```
    echo "Even"
  else
    echo "Odd"
  fi
fi
```

String-based Condition

Making decisions based on a string user-input is also possible in the bash shell. The string-based condition returns a binary expression as a result meaning, it returns true if the specified condition is satisfied otherwise, it returns false. The following are the commonly-used string-based conditional operators:

Operator	Description
==	Returns true if the strings are equal
!=	Returns true if the strings are not equal
-n	Returns true if the string to be tested is not null
-z	Returns true if the string to be tested is null

Let's create a sample script using string-based conditional statement. The script will allow the user to input two strings and assess whether one of the strings is null, both string are equal and not equal.

```
#!/bin/bash

read -p "First String: " str1
 read -p "Second String: " str2
 if [ -z "$str1" ]
 then
    echo "The 1st string is null"
 elif [ -z "$str2" ]
```

```
then
   echo "The 2nd string is null"
else
   if [ $str1 == $str2 ]
   then
       echo "The strings are equal"
   else
      echo "The strings are not equal"
   fi
fi
```

Reserve keyword

Keywords are the words whose meaning has already been explained to the shell. the keywords cannot be used as variable names because of it is a reserved words with containing reserved meaning.

Echo	read	set	unset
readonly	shift	export	if
Fi	else	while	do
Done	for	until	case
Esac	break	Continue	exit
return	trap	Wait	eval
Exec	ulimit	Umask	

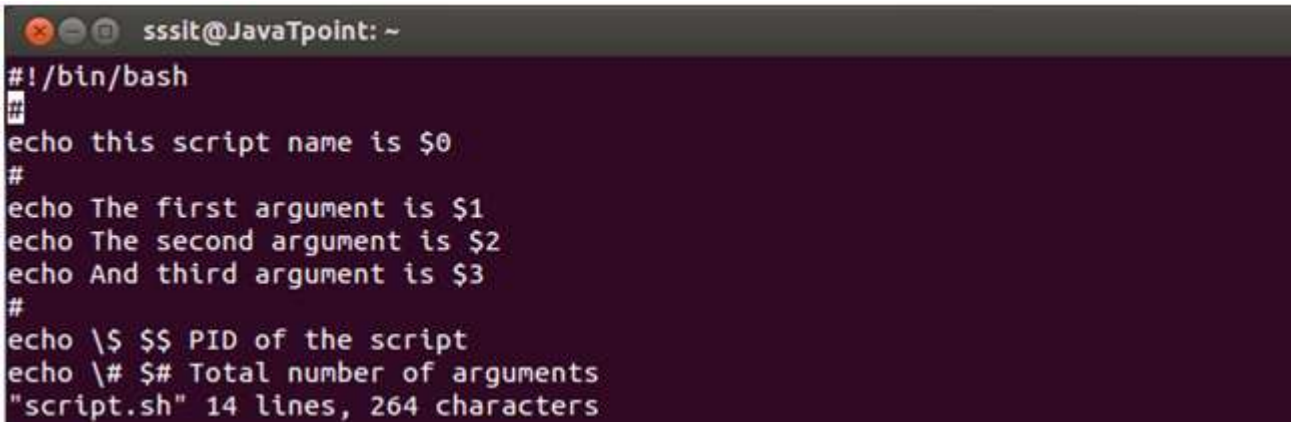
Shell Script Parameters

A bash shell script have parameters. These parameters start from **$1** to **$9.** When we pass arguments into the command line interface, a positional parameter is assigned to these arguments through the shell. The first argument is assigned as $1, second argument is assigned as $2 and so on. If there are more than 9 arguments, then **tenth** or onwards arguments can't be assigned as $10 or $11. You have to either process or save the $1 parameter, then with the help of **shift** command drop parameter 1 and move all other arguments down by one. It will make $10 as $9, $9 as $8 and so on.

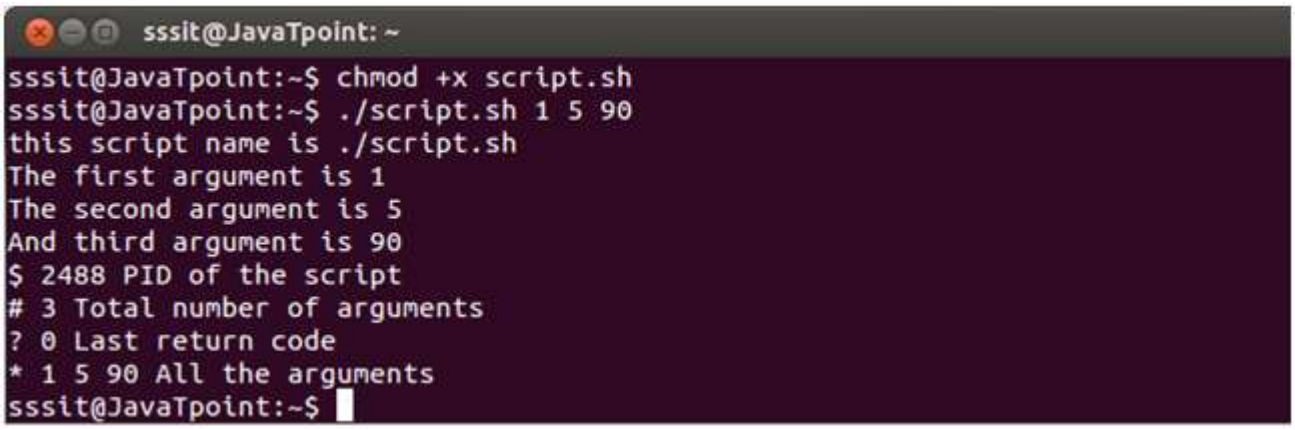
Shell Parameters

Parameters	Function
$1-$9	Represent positional parameters for arguments one to nine
${10}-${n}	Represent positional parameters for arguments after nine
$0	Represent name of the script
$*	Represent all the arguments as a single string
$@	Same as $*, but differ when enclosed in ('')
$#	Represent total number of arguments
$$	PID of the script
$?	Represent last return code

Example:



Look at the above snapshot, this is the script we have written to show the different parameters.



Look at the above snapshot, we have passed arguments **1, 5, 90**. All the parameters show their value when script is run.

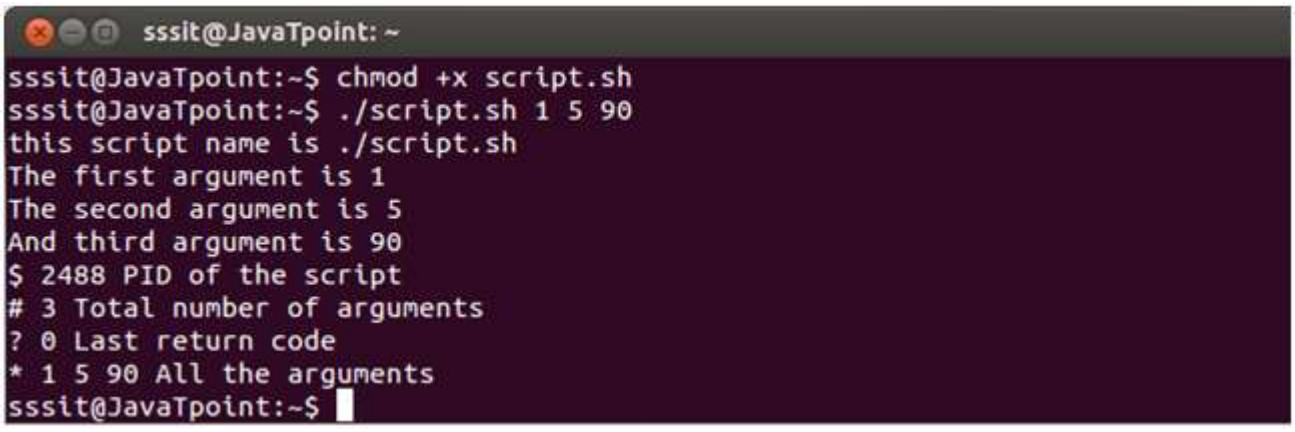
Shell Scripting Shift through Parameters

Shift command is a built-in command. Command takes number as argument. Arguments shift down by this number. For example, if number is 5, then $5 become $1, $6 become $2 and so on.

Example:

The shift command is mostly used when arguments are unknown. Arguments are processed in a while loop with a condition of **(($#))**. This condition holds true as long as arguments are not zero. Number of arguments are reduced each time as the shift command executes.

Look at the above snapshot, this is our script.



Look at the above snapshot, this is the output of the above script.

Read command

The read command allows a user to provide the runtime input.

Look at the above snapshot, this is our script using read command.

Look at the above snapshot, a user can enter the name in the shell.

UNIX and Networking

Once upon a time, there was something called a "standalone Computer" -- one that was used to perform some specific task and didn't require other devices or a network interface to do its job. It might have allowed its owner to write essays, play games, or manage family finances, but all data was stored and processed locally. And, if there was a printer involved, it was a "slave" printer -- one that hung off a port on the system and wasn't shared unless someone carried it to another computer. My first home computer was a standalone and my neighbors at the time thought I was some kind of freak just for having computer in my home, never mind having it sitting on a desk in the corner of my dining room. At the time, this identified me as being clearly abnormal, though I'd bet today these same people all have home computers and can't imagine going through life without them. Standalones may still exist today and their isolation may work to their advantage, but they are very rare. These days, nearly all computers are hooked to networks and are able to connect to systems around the globe. Unix/Linux systems are no exception. This kind of connectivity changes relationships between families, friends, and people with shared interests dramatically. I hear from my sister far more often than the 2,920 miles between us would allow if Facebook and Gmail didn't provide a

way for us to share the fairly trivial news from our day-to-day lives. And I recently met a 5th cousin once removed on LinkedIn. That's fairly incredible and something I'd never have had the patience or the tools to pull off 20 years ago.

The scene today.

Instead of a standalone computer, today most people I know now have a *home network*. They have multiple computers connected to a private LAN, maybe a printer or two, and maybe even a large capacity network drive for backups. And some of those home networks include a mix of Windows and Linux systems. So, what do you need to know if one or more of those computers is a Linux system? What do you need to know if you're going to be setting up network connections on Linux systems at work?

Setting up your network connection.

Whether at home or at work, if you're configuring a system to act as a Linux server, you will need to assign it what is called a "static" address. *Static* means it doesn't change. *Server* means that other systems need to find and connect to it. A static address is necessary so that clients, when they need to initiate connections, have a known address to use for these connections. For non-servers or "client" systems, addresses can be assigned dynamically and generally are -- whether they are assigned an address by a local router (as I have in my home) or by a DHCP server that manages this role within an organization. Even if your system is assigned an address dynamically, this doesn't mean that the address will change every time you boot it. Even DHCP addresses tends to be stable for a while. There is some likelihood that the same IP address will be assigned to a system for weeks, even months -- just no guarantee. As I like to tell my students, clients need to be able to find servers like shoppers need to find supermarkets. Servers never need to find their clients and supermarkets don't need to find their customers. Since clients initiate the connections, they can be coming from nearly anywhere.

DNS Network setup:

Introduction

Active Directory (AD) uses DNS in the background, to locate other DCs and services, such as Kerberos. Thus AD domain members and servers must be able to resolve the AD DNS zones. The following describes how to manually configure Linux clients to use DNS servers. If you are running a DHCP server providing DNS settings to your client computers, configure your DHCP server to send the IP addresses of your DNS servers.

Configuring the /etc/resolv.conf

Set the DNS server IP and AD DNS domain in your `/etc/resolv.conf`. For example:

```
nameserver 10.99.0.1
search samdom.example.com
```

Some utilities, such as Network Manager can overwrite manual changes in that file. See your distribution's documentation for information about how to configure name resolution permanently.

Testing DNS resolution

To verify that your DNS settings are correct and your client or server is able to resolve IP addresses and host names use the **lookup** command. The command is available on Linux and Windows

Forward Lookup

To resolve a host name its IP address:

```
# nslookup DC1.samdom.example.com
Server:         10.99.0.1
```

```
Address:        10.99.0.1#53

Name:   DC1.samdom.example.com
Address: 10.99.0.1
```

Reverse Lookup

To resolve a IP address to its host name:

```
# nslookup 10.99.0.1
Server:         10.99.0.1
Address:        10.99.0.1#53

1.0.99.10.in-addr.arpa          name = DC1.samdom.example.com.
```

Note that in a Samba AD, the reverse zone is not automatically configured. To set up a reverse zone, see DNS Administration.

Resolving SRV Records

Active Directory (AD) uses SRV records to locate services, such as Kerberos and LDAP. To verify that SRV records are resolved correctly, use the `nslookup` interactive shell:

```
# nslookup
Default Server:  10.99.0.1
Address:  10.99.0.1

> set type=SRV
> _ldap._tcp.samdom.example.com.
Server:  UnKnown
```

```
Address:  10.99.0.1


_ldap._tcp.samdom.example.com   SRV service location:
        priority      = 0
        weight        = 100
        port          = 389
        svr hostname   = dc1.samdom.example.com
samdom.example.com      nameserver = dc1.samdom.example.com
dc1.samdom.example.com  internet address = 10.99.0.1
```

Error Messages

- The DNS server is not able to resolve the host name:

```
** server can't find DC1.samdom.example.com: NXDOMAIN
```

- The DNS server is not able to resolve the IP address:

```
** server can't find 1.0.99.10.in-addr.arpa: NXDOMAIN
```

- The DNS server used is not available:

```
;; connection timed out; no servers could be reached
```

Setting Up of mails.

If you want to send or receive an email, you should have a mail server. In this post, we will discuss Linux mail server and how the SMTP (Simple Mail Transfer Protocol) works as well as

other mail-related protocols, like Post Office Protocol (POP) and Internet Message Access Protocol (IMAP) and the relationship between them.

What is Linux mail server ?.

Linux SMTP Server

SMTP defines how a mail is sent from one host to another, it is also system independent, which means the sender and receiver can have different operating systems. SMTP requires only that a server is able to send straight ASCII text to another server, and this is done by connecting to the server on **Port 25** which is the standard SMTP port. Most Linux distros today are shipped with two of the most common implementations of SMTP which are **sendmail** and **Postfix.** Sendmail is a famous and free mail server, but it has a little complex design and less secure. The Postfix took mail server implementation one step further, it was developed with security in mind.

Mail Service Components

The mail service on any mail server has three components:

Mail user agent (MUA): this component that the user sees and interacts with like Thunderbird and Microsoft Outlook, these user agents are responsible for reading mail and allowing you to compose mail.

Mail transport agent (MTA): this component is responsible for getting the mail from one site to another like Sendmail and Postfix.

Mail delivery agent (MDA): this component is responsible for distributing received messages on the local machine to the appropriate user mailbox like postfix-mail drop and Procmail.

How to Setup Email Server

We chose Postfix mail server, which is very popular and common among system administrators today.

Postfix is the default mail server on most modern Linux distros. First, check if it is installed on your system or not:

$ rpm -qa | grep postfix

If not installed, you can install Postfix mail server on Red Hat based distros like this:

$ dnf -y install postfix

Then start the postfix service and enable it on system startup:

$ systemctl start postfix

$ systemctl enable postfix

On Debian based distros like Ubuntu, you can install it like this:

$ apt-get -y install postfix

You will be prompted to select your Postfix mail server configuration type during the installation process.

Among the four choices No configuration, Internet site, Internet with smarthost, Satellite system and Local only, we will choose No configuration option.

WWW servers under UNIX

Source: IBM

Before moving forward for setting up a world wide based server. You have some basic requirement. installed the base UNIX operating system of your choice. you must also have Tomcat version 6 installed and running.

System requirements

- Your server should have at least 10GB of available hard disk space and a minimum of 512MB of RAM. You need the following additional tools installed on your UNIX server before you can begin:

- **A Web browser:** Any browser will do.

- Java 2 Standard Edition Runtime Environment (JRE) release version 5.0 **or later:** Apache Tomcat version 6 requires the JRE.

- **A** C **compiler:** Unfortunately, the base UNIX builds for flavors such as Solaris and IBM® AIX® do not come with such a compiler. See the link to the GNU GCC article in related topics for more information.

- **File-extraction tools:** On your server, you will likely need `gunzip`, `tar`, `bzcat`, and possibly GNU `make` and `tar` (`gmake`, `gtar`—available for download from the GNU site) to properly extract the files

- To check which tools if any are already running on your server, run the following command:

```
cd /usr
Find . –name *.tar (repeat for *make, *zip, etc.)
```

- **Tomcat code:** You need to download the Tomcat version 6 code from the Apache download site before proceeding. Save the tar.gz file to your server.

Preparing your server

Prepare your server by configuring your variables, editing your configuration files, creating users, and more.

Set your paths. In addition to the tools that must be installed on the server, you need to have all your PATH variable settings for your shell set up correctly. The PATH variables tell the shell where to look for certain software installations, tools, and so on.

Check your Java version

To check which version of Java technology you're using, use the command java -version. You should then see something like this:

JAVA_HOME

First, you must install the JRE according to the instructions included with the release. I typically choose to use the full Java software development kit (JDK) rather than just the JRE. If you do, too, set your JAVA_HOME environment variable to the path name of the directory into which you installed the JDK (for example, /usr/local/java/j2sdk5.0). Use the following code to set your JAVA_HOME variable:

```
1   bash-3.00#                                    export
    JAVA_HOME=/usr/jdk/instances/jdk1.5.0
```

You can also run:

```
1   JAVA_HOME=/usr/bin/java; export JAVA_HOME'
```

(there's really no difference).

Type an echo command to double-check that the command worked:

```
1   bash-3.00# echo $JAVA_HOME
2   /usr/jdk/instances/jdk1.5.0
```

CATALINA_HOME (Tomcat's default home directory)

CATALINA_HOME is used to refer to the full path name of the release directory. Use the following code to set this variable on your server:

```
1   bash-3.00#  export  CATALINA_HOME=/opt/apache-tomcat-
    6.0.16
```

This code pertains specifically to the Tomcat version you have installed. When in doubt, name it after the .tar file you extract, because that carries with it the correct version in the name.

Edit the shell and configuration file under your user

You probably noticed that I have a few references to *Bash* in the command samples. Without getting into boring details, users logged in to UNIX run out of shells on the server to execute a CLI that allows them to navigate and administer the server. This is not unlike other operating systems but is more flexible in that you can define that shell by typing something like Bourne shell (sh) or Bourne-again shell (bash). In addition, depending on which shell you're using, you can edit the custom shell profile for your CLI to automatically set PATH variables and even alias characters to represent useful common commands on the server. To do this, edit .bashrc, .profile (the default under the bash shell), and so on. Use the vi editor to create this file, then edit it appropriately under your root, personal, or Tomcat-specific user ID to ensure that the PATH variables will be set each time you log in to the server. Under your home directory, create the .profile file (shown in Listing 1), if it does not exist already, using the standard UNIX editor, vi.

Listing 1. Create your .profile using a standard UNIX editor

```
1  cd ~/ (home dir)
2  vi .profile
```

Type i (for insert), and then add each path you set above. An example may look something like this:

```
1  # /bin/sh
2  stty istrip
3  PATH=$PATH:/usr/bin:/usr/local/bin:/usr/ucb:/etc
4  export PATH
5  umask 077
6   export SHELL=/usr/bin/ksh
7  export ENV=$HOME/.kshrc
8  export EDITOR=vi
9  export FCEDIT=vi
10  #Tomcat specific PATHs
   export JAVA_HOME=/usr/jdk/instances/jdk1.5.0
```

```
11 export CATALINA_HOME=/opt/apache-tomcat-6.0.16
12
```

This may also be a good time to add additional PATH variables to cover all the areas in which you have tools installed that you're going to reference. For my setup, I also need to create the following two paths:

```
1 bash-3.00# PATH=/usr/ccs/bin:$PATH; export PATH (for make
  cmd)
2 bash-3.00# PATH=/usr/sfw/bin:$PATH; export PATH
```

Users and groups

If multiple users will use your server, you definitely want to set up user and group privileges to allow your different users access to various tools and file systems. This is also necessary if you're going to run your Tomcat installation as a non-root user (recommended for most production environments). I will get into the details of this in a later tutorial; however, here are some commands to chew on. To create a Tomcat group, use the code in Listing 2.

Listing 2. Create a unique group for Tomcat to run under

```
1 /usr/sbin/groupadd -g {specific gid. Leave this blank and the OS will
  a gid}
2 {group name}
3 ie.
4 /usr/sbin/groupadd -g 10004 tomcatgroup
```

To create a Tomcat user, use the code in Listing 3.

Listing 3. Create a unique user for Tomcat to run under

```
1  /usr/sbin/useradd -d {user home directory} -g {user primary group} -u
2  {specific UID. You can leave this blank, and the operating system wil
   a UID.)
3  -s {default shell path for this user} -c "{Description of the user}"
4  ie.
5  /usr/sbin/useradd -d /export/home/tomcat -g tomcatgroup -u 10010 -s /
6  "Main Tomcat Administrative User" tomcat
```

Download and extract the server installation file

Discover the various methods for extracting the server installation file.

Find and move the installation file

If you're using a GUI, open a terminal window by choosing **Launch > Applications > Utilities > Terminal**. If you saved this file to your desktop and you're still the root user, run the command cd /Desktop. Run ls -ltr to see if your Tomcat tar.gz file is there. Then, move this file under the /opt directory (which is where most new software is installed). To move the installation files, type:

```
1  mv *tar.gz /opt
```

Then, type Ls –ltr /opt to make sure it's there.

Set permissions

Permissions are the lifeblood of a UNIX system. Without the proper permissions, you can't do anything; without restricting permissions to certain things, any user can do anything. That is why you first must grant yourself higher-level permissions on the installation file so that you (or another user) can properly execute it. Next, use the umask command, shown in Listing 4, to make sure that you can write the files to all the temp and installation directories appropriately when you extract the installation code.

Listing 4. Set the proper permissions and umask

```
cd /opt
chmod +x *gz (same as chmod 775)
umask 007 (makes any new files your user creates to be created wi
permission.
Think chmod in reverse)
```

Extract the installation file

Most code packages come as a .tar file, with a .gz file to compact it even further. The gunzip command simply expands the package from its first layer of compression:

```
gunzip *.gz
```

Extract the .tar file

Finally, you must "untar" (unpack) the code. Doing so extracts the code exactly as it was packed—directories and all. This is why it's important to untar the file under the directory path in which you need the code installed. In UNIX, the best place would be /opt.

```
tar -xvf *.tar
```

This command extracts your application files. Type ls -latr to view the extracted files. If for some reason your installation came with a .bat or .exe file, you can remove those by typing:

```
rm *.exe
rm *.bat
```

When the Tomcat directory established, you can move on to configuring, compiling, and starting the server.

Configure, compile, and start; All Apache products that I've worked with include steps for configuring, compiling, and starting the product. This process gets the code ready, tells the software how and where to install, and so on.

Configure the software

To configure the Tomcat software, type:

```
cd $CATALINA_HOME/bin

./configure --with-java=/usr/java
```

or:

```
export JAVA_HOME
./configure
```

Compile the code

With the Tomcat software configured, it's time to compile it.

Build the binaries and libraries

Listing 5 shows the code for building the Tomcat binaries and libraries.

Listing 5. Build the binaries and libraries

```
# gunzip jsvc.tar.gz

# pwd
/opt/apache-tomcat-6.0.16/bin

# tar -xvf jsvc.tar

# gmake
```

Make sure that the path to gmake is in your PATHS (for example, ./sfw/bin/).

Note: The Tomcat site states that you should use the GNU make (gmake) instead of the native BSD make command on FreeBSD systems. The code in Listing 6 generates the executable file, .jsvc. This file is required to successfully run Tomcat as a daemon.

Quick definition of jsvc

Jsvc uses three processes: a launcher process, a controller process, and a controlled process. The controlled process is also the main Java thread; if the Java Virtual Machine (JVM) crashes,

the controller restarts it in the next minute. Jsvc is a daemon process, so it should be started as root; the -user parameter allows you to downgrade to an unprivileged user.

Listing 6. Set the permissions on jsvc and copy

```
1  chmod 775 jsvc
2  cp jsvc ..
3  cd ..
```

Start the server

You can start the server from the CLI or from a Java program as an embedded server. Furthermore, the server can run as a daemon, which will run automatically, similar to a service setup in a Windows environment.

Run the basic startup script

Listing 7 shows the basic startup script for Tomcat.

Listing 7. Run the basic startup script

```
1  cd $CATALINA_HOME/bin
2  ./startup.sh
3  cd ../logs
```

Check catalina.out for errors! If you use cat, vi, more, or less on this file, type shift G to go to the bottom of the file. Or, you can type something like:

```
1  tail -50 catalina.out
```

to check the last 50 lines of the file for errors.

Although doing so is beyond the scope of this tutorial, you can also customize the startup process by modifying the Tomcat code or by implementing your own Lifecycle Listeners.

Run the startup daemon

You can start the daemon using a variety of options, such as -user for non-root user, -pid to specify the .pid file location, and -errfile and -outfile to specify the error and output file logs, respectively. For a complete list, find your jsvc process, and type ./jsvc -help. Listing 8 shows an example of a jsvc startup script.

What's a daemon?

A *daemon* is a non-interactive server application that the operating system controls by a set of specified signals. Think *services* in Windows. This provides for a graceful shutdown of server applications; the operating system can notify a server application of its imminent shutdown, and the application has the ability to perform certain tasks before its process of execution is destroyed.

Listing 8. Example jsvc startup script

```
Bash#./jsvc –home /usr/jdk/instances/jdk1.5.0 \
–Dcatalina.home=/opt/apache-tomcat-6.0.16 \
-cp          ./bin/bootstrap.jar          -outfile
./logs/catalina.out \
-errfile ./logs/catalina.err \
       org.apache.catalina.startup.Bootstrap
```

It is also very helpful to use the Tomcat.sh script included under the jsvc/bin directory. However, you will need to edit the variables to make sure the paths and such match your environment.