F.Y.B.C.A. SCIENCE : SEMESTER-II

NEW SYLLABUS
CBCS PATTERN

OPERATING SYSTEMS CONCEPTS

Dr. SUNITA D. PATIL
Mrs. ARCHANA KOTHAWADE
GAJANAN DESHMUKH

NIRALI PRAKASHAN
ADVANCEMENT OF KNOWLEDGE

SPPU New Syllabus

A Book Of

OPERATING SYSTEMS CONCEPTS

For B.C.A.(Science) : Semester - II

[Course Code 123 : Credit - 4]

CBCS Pattern

As Per New Syllabus, Effective from June 2019

Dr. Sunita D. Patil
M.C.M., M.C.A., M.Phil (IT),
Ph.D. (Computer Management)
Dean Academics
Yashaswi Institute of Technology
Yashaswi Academy of Skills
Chinchwad, Pune-411033

Mrs. Archana Kothawade
B.E. Computer, MBS
Professor at Garaware College of Commerce
Pune

Mr. Gajanan Deshmukh
B.Sc. (Computer Science), MCA
HOD, Smt. Kashibai Navale College of Commerce
Pune

Price ₹ 165.00

NIRALI PRAKASHAN
ADVANCEMENT OF KNOWLEDGE

N5113

Operating Systems Concepts **ISBN 978-93-89533-72-9**

First Edition : **November 2019**

Published By :
NIRALI PRAKASHAN
Abhyudaya Pragati, 1312, Shivaji Nagar,
Off J.M. Road, Pune – 411005
Tel - (020) 25512336/37/39, Fax - (020) 25511379
Email : niralipune@pragationline.com

Polyplate

Printed By :
YOGIRAJ PRINTERS AND BINDERS
Survey No. 10/1A, Ghule Industrial Estate
Nanded Gaon Road
Nanded, Pune - 411041
Mobile No. 9404233041/9850046517

➢ DISTRIBUTION CENTRES

PUNE

Nirali Prakashan (For orders within Pune) : 119, Budhwar Peth, Jogeshwari Mandir Lane, Pune 411002, Maharashtra
Tel : (020) 2445 2044; Mobile : 9657703145
Email : niralilocal@pragationline.com

Nirali Prakashan (For orders outside Pune) : S. No. 28/27, Dhayari, Near Asian College Pune 411041
Tel : (020) 24690204; Mobile : 9657703143
Email : bookorder@pragationline.com

MUMBAI

Nirali Prakashan : 385, S.V.P. Road, Rasdhara Co-op. Hsg. Society Ltd.,
Girgaum, Mumbai 400004, Maharashtra; Mobile : 9320129587
Tel : (022) 2385 6339 / 2386 9976, Fax : (022) 2386 9976
Email : niralimumbai@pragationline.com

➢ DISTRIBUTION BRANCHES

JALGAON

Nirali Prakashan : 34, V. V. Golani Market, Navi Peth, Jalgaon 425001, Maharashtra,
Tel : (0257) 222 0395, Mob : 94234 91860; Email : niralijalgaon@pragationline.com

KOLHAPUR

Nirali Prakashan : New Mahadvar Road, Kedar Plaza, 1st Floor Opp. IDBI Bank, Kolhapur 416 012
Maharashtra. Mob : 9850046155; Email : niralikolhapur@pragationline.com

NAGPUR

Nirali Prakashan : Above Maratha Mandir, Shop No. 3, First Floor,
Rani Jhanshi Square, Sitabuldi, Nagpur 440012, Maharashtra
Tel : (0712) 254 7129; Email : niralinagpur@pragationline.com

DELHI

Nirali Prakashan : 4593/15, Basement, Agarwal Lane, Ansari Road, Daryaganj
Near Times of India Building, New Delhi 110002 Mob : 08505972553
Email : niralidelhi@pragationline.com

BENGALURU

Nirali Prakashan : Maitri Ground Floor, Jaya Apartments, No. 99, 6th Cross, 6th Main,
Malleswaram, Bengaluru 560003, Karnataka; Mob : 9449043034
Email: niralibangalore@pragationline.com

Other Branches : Hyderabad, Chennai

Preface ...

We take this opportunity to present this book entitled as **"Operating Systems Concepts"** to the students of Second Semester - BCA (Science). The object of this book is to present the subject matter in a most concise and simple manner. The book is written strictly according to the New Syllabus.

The book has its own unique features. It brings out the subject in a very simple and lucid manner for easy and comprehensive understanding of the basic concepts, its intricacies, procedures and practices. This book will help the readers to have a broader view on Operating Systems Concepts. The language used in this book is easy and will help students to improve their vocabulary of Technical terms and understand the matter in a better and happier way.

We sincerely thank Shri. Dineshbhai Furia and Shri. Jignesh Furia of Nirali Prakashan, for the confidence reposed in us and giving us this opportunity to reach out to the students of management studies.

We have given our best inputs for this book. Any suggestions towards the improvement of this book and sincere comments are most welcome on niralipune@pragationline.com.

Authors

■■■

Syllabus ...

1. **Introduction** **[08 Hrs]**
   - Introduction to Operating Systems, Different Services provided by Operating System to Users.
   - Introduce the Concept of Process, Process States, Process Control Block, User Interface, System Calls.
   - Introduction to Linux Operating System - Features of Linux, Architecture of the Linux, Introduction to File System and Process Environment.
   - Working with Linux - The Login Prompt, General Features of Commands/Command Structure, Command Arguments and Options.
   - Understanding of some Basic Commands such as echo, printf, is, who, date, passwd, cal, Combining Commands.
2. **Commands and Processes** **[08 Hrs]**
   - What is a Command? Meaning of Internal and External Commands, The Type Command: Knowing the Type of a Command and locating it, The man Command knowing more about Commands and using Linux online manual pages. The man with keyword option and what is.
   - Operating System Processes - Concept, Mechanism of Process Creation, Parent and Child Process, The ps command with its options, Executing a command at a specified point of time: at command.
   - The nice command, Background processes. The bg and fg commands, The kill command, The find command with illustrative example.
3. **File System** **[08 Hrs]**
   - Linux Files - Naming files, Basic File Types.
   - Organization of Files, Standard Directories, Parent Child Relationship, The Home Directory and the HOME Variable.
   - The PATH Variable, Manipulating the PATH, Relative and Absolute Pathnames, Directory Commands – pwd, cd, mkdir, rmdir Commands, The dot (.) and Double Dots (..) Notations to represent Present and Parent Directories and their Usage in relative path names.
   - File Related Commands – cat, mv, rm, cp, wc and od Commands, File Attributes and Permissions and knowing them, The ls Command with Options, Changing File Permissions: the Relative and Absolute Permissions changing methods.
4. **Using Shells and Vi Editor** **[08 Hrs]**
   - What is Shell? Different types of shells, the Shells Interpretive Cycle, Wild Cards and File Name Generation, Removing the Special Meanings of Wild Cards, Three Standard Files and Redirection.
   - Connecting Commands: Pipe, The grep, egrep commands.
   - Vi Editor - Introduction to the Vi editor, Different ways of invoking and quitting vi, Different modes of vi, Input mode commands, Command mode commands, The ex mode Commands, Illustrative Examples Navigation Commands.
5. **Security and Networking** **[08 Hrs]**
   - Security Understanding Linux Security, Uses of root, pseudo command, working with passwords, Bypassing user authentication, Understanding ssh.
   - Networking Basic introduction to Networking, Network protocols: http, ftp etc., IP address, DNS.
6. **Shell Scripts** **[08 Hrs]**
   - Shell programming - Ordinary and environment variables, The .profile. Read and read only commands, Command line arguments, exit and exit status of a command, Logical operators for conditional execution, The test command and its shortcut.
   - The if, while, for and case control statements, The set and shift commands and handling positional parameters, The here (<<) document and trap command, Simple shell program examples.
   - File inodes and the inode structure, File links – Hard and Soft Links. Filters, Head and Tail Commands.
   - Cut and paste commands, The sort command and its usage with different options.

■■■

Contents ...

■■■

1... Introduction

Objectives...

- To study Basic Fundamentals of Operating System.
- To understand Concept of Linux Operating System.
- To learn Basic Commands of Linux Operating System.

1.1 INTRODUCTION TO OPERATING SYSTEMS

- **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 a program which manages all the computer hardware's.
- It provides the base for application program and acts as an intermediary between a user and the computer hardware.
- The operating system has two objectives such as:
  - Firstly, an operating system controls the computer's hardware.
  - The second objective is to provide an interactive interface to the user and interpret commands. So that it can communicate with the hardware.
- The operating system is very important part of almost every computer system.

1.2 DIFFERENT SERVICES PROVIDED BY OPERATING SYSTEM TO USERS

- An operating system is an interface between the user of a computer and the computer hardware. It is a collection of software that manages computer hardware resources and offers common services for programs of the computer.
- An operating system provides completely different kind of services to each the users and to the programs likewise. It also provides application programs an environment to execute it freely. It provides users the services run various programs in a very convenient manner.

- Here is a list of common services offered by an almost all operating systems:
  1. **User Interface:** Usually Operating system comes in three forms or types. Depending on the interface their types have been further subdivided. These are:
     - Command line interface
     - Batch based interface
     - Graphical User Interface
  2. **Program Execution:** The operating system must have the capability to load a program into memory and execute that program. Furthermore, the program must be able to end its execution, either normally or abnormally / forcefully.
  3. **File system manipulation:** Programs need has to be read and then write them as files and directories. File handling portion of operating system also allows users to create and delete files by specific name along with extension, search for a given file and / or list file information. Some programs comprise of permissions management for allowing or denying access to files or directories based on file ownership.
  4. **Input / Output Operations:** A program which is currently executing may require I/O, which may involve file or other I/O device. For efficiency and protection, users cannot directly govern the I/O devices. So, the OS provide a means to do Input / Output operation which means read or write operation with any file.
  5. **Communication:** Process needs to swap over information with other process. Processes executing on same computer system or on different computer systems can communicate using operating system support. Communication between two processes can be done using shared memory or via message passing.
  6. **Resource Allocation:** When multiple jobs running concurrently, resources must need to be allocated to each of them. Resources can be CPU cycles, main memory storage, file storage and I/O devices. CPU scheduling routines are used here to establish how best the CPU can be used.
  7. **Error Detection:** Errors may occur within CPU, memory hardware, I/O devices and in the user program. For each type of error, the OS takes adequate action for ensuring correct and consistent computing.
  8. **Accounting:** This service of the operating system keeps track of which users are using how much and what kinds of computer resources have been used for accounting or simply to accumulate usage statistics.
  9. **Security and protection:** Protection includes in ensuring all access to system resources in a controlled manner. For making a system secure, the user needs to authenticate him or her to the system before using (usually via login ID and password).

1.3 INTRODUCE THE CONCEPT OF PROCESS

- A process refers to a program in execution; it's a running instance of a program. It is made up of the program instruction, data read from files, other programs or input from a system user.

- A program by its self is not a process. A program is a passive entity such as content of a file or instruction code of particular files whereas process is an active entity. Which specify next instruction to execute and set if associated resources.
- There are two different types of process

1. **Foreground Processes:**
   - These are introduced and controlled through a terminal session. In other words, there must be a client associated with the framework to begin such procedures; they haven't began naturally as a component of the framework functions/ services.
   - By default, every process that you start runs in the foreground. It gets its input from the keyboard and sends its output to the screen.
   - This happen with the ls command. If you wish to list all the files in your current directory,you can use the following command –
     $ls ch*.doc
   - This would display all the files, the names of which start with ch and end with .doc:

     ch01-1.doc ch010.doc ch02.doc ch03-2.doc
     ch04-1.doc ch040.doc ch05.doc ch06-2.doc
     ch01-2.doc ch02-1.doc
   - The process runs in the foreground, the output is directed to my screen, and if the ls command wants any input (which it does not), it waits for it from the keyboard.
   - While a program is running in the foreground and is time-consuming, no other commands can be run (start any other processes) because the prompt would not be available until the program finishes processing and comes out.

2. **Background Processes:**
   - A background process runs without being connected to your keyboard. If the background process requires any keyboard input, it waits.
   - The advantage of running a process in the background is that you can run other commands; you do not have to wait until it completes to start another!
   - The simplest way to start a background process is to add an ampersand (&) at the end of the command.

     $ls ch*.doc &
   - This displays all those files the names of which start with ch and end with .doc –

     ch01-1.doc ch010.doc ch02.doc ch03-2.doc
     ch04-1.doc ch040.doc ch05.doc ch06-2.doc
     ch01-2.doc ch02-1.doc
   - Here, if the ls command wants any input (which it does not), it goes into a stop state until we move it into the foreground and give it the data from the keyboard.

- That first line contains information about the background process - the job number and the process ID. You need to know the job number to manipulate it between the background and the foreground.
- Press the Enter key and you will see the following –

```
[1] + Done ls ch*.doc &amp;
$
```

- The first line tells you that the ls command background process finishes successfully. The second is a prompt for another command.

1.3.1 Process States

- A process which is Executed by the Process have various States, the State of the process is also called as the Status of the process. The Status includes whether the Process has Executed or Whether the process is Waiting for Some input and output from the user and whether the Process is Waiting for the CPU to Run the Program after the Completion of the Process:

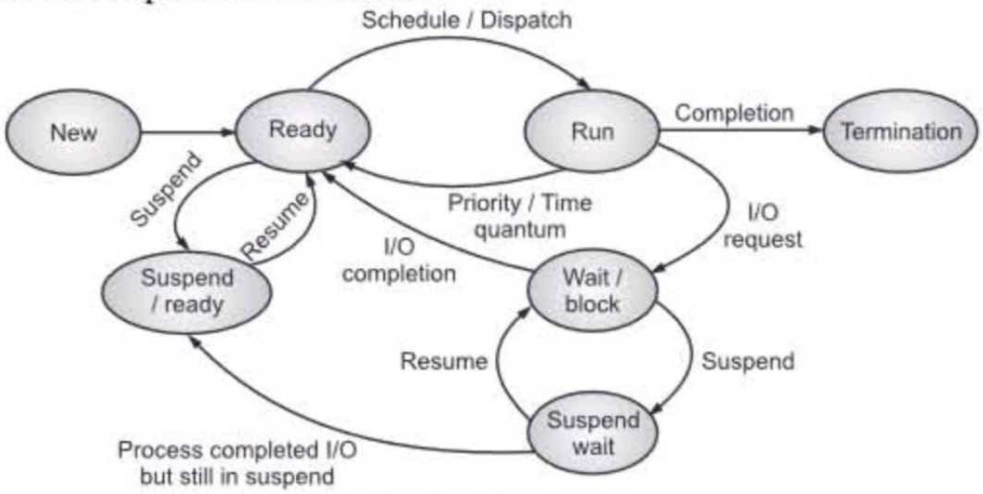


Fig. 1.1: Process states

- **New (Create):** In this step, the process is about to be created but not yet created, it is the program which is present in secondary memory that will be picked up by OS to create the process.
- **Ready:** New -> Ready to run. After the creation of a process, the process enters the ready state i.e. the process is loaded into the main memory. The process here is ready to run and is waiting to get the CPU time for its execution. Processes that are ready for execution by the CPU are maintained in a queue for ready processes.
- **Run:** The process is chosen by CPU for execution and the instructions within the process are executed by any one of the available CPU cores.
- **Blocked or wait:** Whenever the process requests access to I/O or needs input from the user or needs access to a critical region(the lock for which is already acquired) it enters the blocked or wait state. The process continues to wait in the main memory and does not require CPU. Once the I/O operation is completed the process goes to the ready state.

- **Terminated or completed:** Process is killed as well as PCB is deleted.
- **Suspend ready:** Process that was initially in the ready state but were swapped out of main memory (refer Virtual Memory topic) and placed onto external storage by scheduler are said to be in suspend ready state. The process will transition back to ready state whenever the process is again brought onto the main memory.
- **Suspend wait or suspend blocked:** Similar to suspend ready but uses the process which was performing I/O operation and lack of main memory caused them to move to secondary memory. When work is finished it may go to suspend ready.

1.3.2 Process Control Block

- Process Control Block is a data structure that contains information of the process related to it. The process control block is also known as a task control block, entry of the process table, etc.
- It is very important for process management as the data structuring for processes is done in terms of the PCB. It also defines the current state of the operating system.
- Process Control Block (PCB) in an operating system, is the data structure used to store the process information. It is also called
  - Task controlling block.
  - Process table

Structure of the Process Control Block

- The process control stores many data items that are needed for efficient process management. Some of these data items are explained with the help of the given diagram:

Fig. 1.2: Process Control Block (PCB)

The following are the data items:

- **Process State:** This specifies the process state i.e. new, ready, running, waiting or terminated.
- **Process Number:** This shows the number of the particular process.

- **Program Counter:** This contains the address of the next instruction that needs to be executed in the process.
- **Registers:** This specifies the registers that are used by the process. They may include accumulators, index registers, stack pointers, general purpose registers etc.
- **List of Open Files:** These are the different files that are associated with the process
- **CPU Scheduling Information:** The process priority, pointers to scheduling queues etc. is the CPU scheduling information that is contained in the PCB. This may also include any other scheduling parameters.
- **Memory Management Information:** The memory management information includes the page tables or the segment tables depending on the memory system used. It also contains the value of the base registers, limit registers etc.
- **I/O Status Information:** This information includes the list of I/O devices used by the process, the list of files etc.
- **Accounting information:** The time limits, account numbers, amount of CPU used, process numbers etc. are all a part of the PCB accounting information.
- **Location of the Process Control Block:** The process control block is kept in a memory area that is protected from the normal user access. This is done because it contains important process information. Some of the operating systems place the PCB at the beginning of the kernel stack for the process as it is a safe location.

1.3.3 User Interface

- Almost all operating systems have a user interface (UI).
- User interface can take several forms. One is a command-line interface (CLI) or command interpreter, that allows users to directly enter commands to be performed by the operating system. Another is a batch interface, it allows users to interface with the operating system via a graphical user interface, or GUI.

1.3.3.1 Command Interpreters/Command Line Interface

- Commands issued at the command line on any computer system are read and executed by a program known as a command interpreter.
- The Command Line Interface (CLI), is a non-graphical, text-based interface to the computer system, where the user types in a command and the computer then successfully executes it. The Terminal is the platform or the IDE that provides the command line interface (CLI) environment to the user.
- The CLI terminal accepts the commands that the user types and passes to a shell. The shell then receives and interprets what the user has typed into the instructions that can be executed by the OS (Operating System). If the output is produced by the specific command, then this text is displayed in the terminal. If any of the problems with the commands are found, then some error message is displayed.
- Some operating systems include the command interpreter in the kernel. Other operating system, such as Windows and UNIX, treat the command interpreter as a special program that is running when a user first logs on (on interactive systems).

- A command interpreter does the following:
  - Reads the commands and any options and arguments that you provide.
  - Translates or expands any special characters such as the * and ? used as wildcard characters on Linux and Unix systems .
  - Locates the command that you want to execute on your system.
  - Executes that command with the appropriate options and arguments and displays any output of that command.
- On Linux and Unix systems, a command interpreter is known as a shell. Linux and Unix systems offer many different shells, each of which has its own special features.

Examples Command Line Interface (CLI):

- MS-DOS (Microsoft Disk Operating System),
- CP/M (Control Program for Microcomputers)
- Apple DOS (Apple Disk Operating System)

1.3.3.2 Graphical user interface

- GUI is an interface that allows users to interact with different electronic devices using icons and other visual indicators. The graphical user interfaces were created because command line interfaces were quite complicated and it was difficult to learn all the commands in it.
- In today's times, graphical user interfaces are used in many devices such as mobiles, MP3 players, gaming devices, smart phones etc.
- The term was created in the 1970s to distinguish graphical interfaces from text-based ones, such as command line interfaces. However, today nearly all digital interfaces are GUIs.

Elements in graphical user interface:

1. **Window:**

- This is the element that displays the information on the screen. It is very easy to manipulate a window. It can be opened or closed with the click of an icon. Moreover, it can be moved to any area by dragging it around. In a multitasking environment, multiple windows can be open at the same time, all of them performing different tasks.
- There are multiple types of windows in a graphical user interface, such as container window, browser window, text terminal window, child window, message window etc.

2. **Menu:**

- A menu contains a list a choices and it allows users to select one from them. A menu bar is displayed horizontally across the screen such as pull down menu. When any option is clicked in this menu, then the pull down menu appears.
- Another type of menu is the context menu that appears only when the user performs a specific action. An example of this is pressing the right mouse button. When this is done, a menu will appear under the cursor.

3. **Icons:**

- Files, programs, web pages etc. can be represented using a small picture in a graphical user interface. This picture is known as an icon. Using an icon is a fast way to open documents, run programs etc. because clicking on them yields instant access.

4. **Controls:**

- Information in an application can be directly read or influences using the graphical control elements. These are also known as widgets. Normally, widgets are used to display lists of similar items, navigate the system using links, tabs etc. and manipulating data using check boxes, radio boxes etc.

5. **Tabs:**

- A tab is associated with a view pane. It usually contains a text label or a graphical icon. Tabs are sometimes related to widgets and multiple tabs allow users to switch between different widgets. Tabs are used in various web browsers such as Internet Explorer, Firefox, Opera, Safari etc. Multiple web pages can be opened in a web browser and users can switch between them using tabs.

1.3.4 System Calls

- A system call is a way for programs to interact with the operating system.
- A computer program makes a system call when it makes a request to the operating systems kernel.
- System call provides the services of the operating system to the user programs via Application Program Interface (API).
- In computing, a system call is the programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on.
- A system call is a way for programs to interact with the operating system. A computer program makes a system call when it makes a request to the operating system's kernel.
- System call provides the services of the operating system to the user programs via Application Program Interface(API). It provides an interface between a process and operating system to allow user-level processes to request services of the operating system.
- System calls are usually made when a process in user mode requires access to a resource. Then it requests the kernel to provide the resource via a system call.
- A figure representing the execution of the system call is given as follows:

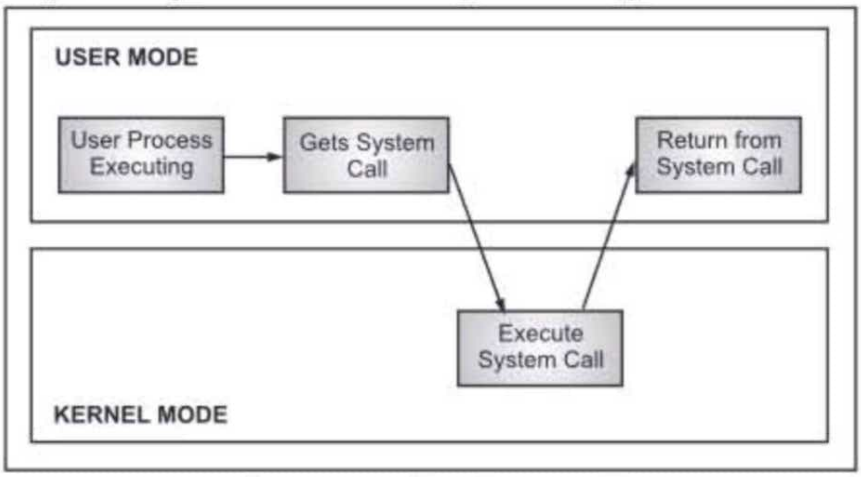


Fig. 1.3: Execution of system call

- As can be seen from this diagram, the processes execute normally in the user mode until a system call interrupts this. Then the system call is executed on a priority basis in the kernel mode. After the execution of the system call, the control returns to the user mode and execution of user processes can be resumed.

1.3.4.1 Types of System Calls

- There are mainly five types of system calls. These are explained in detail as follows:

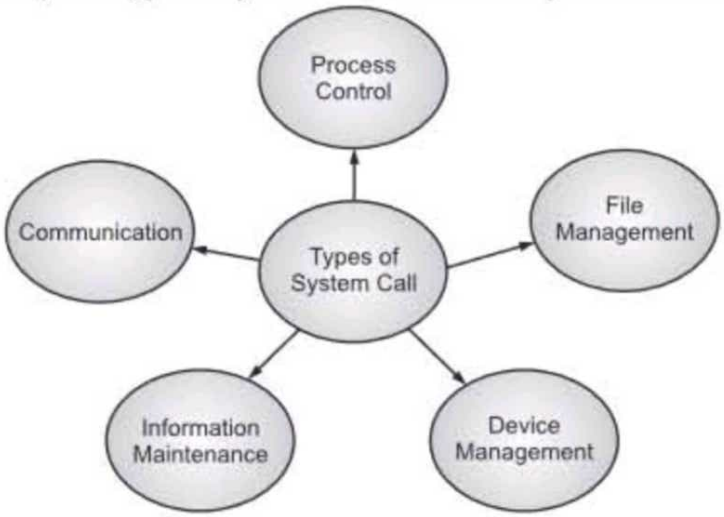


Fig. 1.4: Types of system call

1. **Process Control:** These system calls deal with processes such as process creation, process termination etc.
2. **File Management:** These system calls are responsible for file manipulation such as creating a file, reading a file, writing into a file etc.
3. **Device Management:** These system calls are responsible for device manipulation such as reading from device buffers, writing into device buffers etc.
4. **Information Maintenance:** These system calls handle information and its transfer between the operating system and the user program.
5. **Communication:** These system calls are useful for interprocess communication. They also deal with creating and deleting a communication connection.

1.4 INTRODUCTION TO LINUX OPERATING SYSTEM

- Linux is an operating system for PC computers and workstations that now features a fully functional graphical user interface(GUI) just like windows and the MAC.
- Linux is an operating system or a kernel. It is distributed under an open source license. Its functionality list is quite like UNIX.
- Linux is distinguished by its power and flexibility. A research team at AT&T's Bell Labs developed Unix in the late 1960s and early 1970s with a focus on creating an operating system that would be accessible and secure for multiple users.
- Linux is a family of open source Unix - like operating systems based on the Linux Kernel, on operating system Kernel first released on September 17, 1991 by Linux torvalds. Linux is typically packaged in a Linux distribution.

1.4.1 Features of Linux

- Following are some of the important features of Linux Operating System.
  - **Portable:** Linux operating system can work on different types of hardware's as well as Linux kernel supports the installation of any kind of hardware platform.
  - **Open Source:** Source code of LINUX operating system is freely available and, to enhance the ability of the LINUX operating system, many teams work in collaboration.
  - **Multiuser:** Linux operating system is a multiuser system, which means; multiple users can access the system resources like RAM, Memory or Application programs at the same time.
  - **Multiprogramming:** Linux operating system is a multiprogramming system, which means multiple applications can run at the same time.
  - **Hierarchical File System:** Linux operating system affords a standard file structure in which system files or user files are arranged.
  - **Shell:** Linux operating system offers a special interpreter program, that can be used to execute commands of the OS. It can be used to do several types of operations like call application programs, and so on.
  - **Security:** Linux operating system offers user security systems using authentication features like encryption of data or password protection or controlled access to particular files.

1.4.2 Architecture of the Linux

- Architectures are global and represent the concept of an entire system; they describe how disks are accessed, how memory is handled, how the boot process is defined.
- The following illustration shows the architecture of a Linux system:

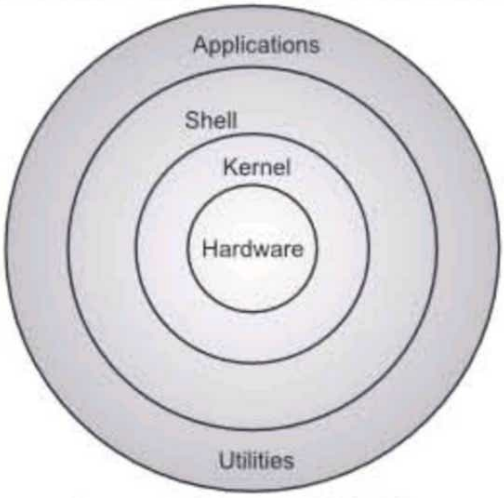


Fig. 1.5: Architecture of the linux

1. **Kernel** is the core part of the operating system, which is responsible for all the major activities of the LINUX operating system. This operating system consists of different modules and interacts directly with the underlying hardware. The

kernel offers the required abstraction to hide application programs or low-level hardware details to the system. The types of Kernels are as follows:
   - Monolithic Kernel
   - Micro kernels
   - Exo kernels
   - Hybrid kernels
2. **System libraries** are special functions, that are used to implement the functionality of the operating system and do not require code access rights of kernel modules.
3. **System Utility** programs are liable to do individual, and specialized-level tasks.
4. **Hardware** layer of the LINUX operating system consists of peripheral devices such as RAM, HDD, CPU, etc.
5. **Shell** is an interface between the user and the kernel, and it affords services of the kernel. It takes commands from the user and executes kernel's functions. The Shell is present in different types of operating systems, which are classified into two types: command line shells and graphical shells.
   The command line shells provide a command line interface, while the graphical line shells provide a graphical user interface. Though both shells perform operations, but the graphical user interface shells perform slower than the command line interface shells. Types of shells are classified into four:
   - Korn shell
   - Bourne shell
   - C shell
   - POSIX shell

1.4.3 Introduction to File System and Process Environment

1.4.3.1 File System

- A file system is the way in which files are named, stored, retrieved as well as updated on a storage disk or partition; the way files are organized on the disk.
- A file system is divided in two segments called: User Data and Metadata (file name, time it was created, modified time, it's size and location in the directory hierarchy etc). Linux system is capable of handling any number of storage devices.
- The entire Linux directory structure starting at the top (/) root directory.
- A specific type of data storage format, such as EXT3, EXT4, BTRFS, XFS, and so on. Linux supports almost 100 types of filesystems, including some very old ones as well as some of the newest. Each of these filesystem types uses its own metadata structures to define how the data is stored and accessed.
- A partition or logical volume formatted with a specific type of filesystem that can be mounted on a specified mount point on a Linux filesystem.
- The major file system types in Linux are
  1. **EXT3 and EXT4:** Ext3 and Ext4 stands for extended file system. Ext3 and Ext4 are used to create and access logical volume.

2. **VFAT:** VFAT is a filesystem equivalent to the windows FAT (File Allocation Table) file system, it stands for Virtual FAT. VFAT is used in external medias like Pendrive and all.
3. **Swap:** Swap is used to create a swap area in the hard disk, which can be used as a virtual memory. The total memory which a running application can see is the sum of physical memory (RAM) and the Virtual memory (Swap).

File Structure:

- A File Structure should be according to a required format that the operating system can understand.
  - A file has a certain defined structure according to its type.
  - A text file is a sequence of characters organized into lines.
  - A source file is a sequence of procedures and functions.
  - An object file is a sequence of bytes organized into blocks that are understandable by the machine.
  - When operating system defines different file structures, it also contains the code to support these file structure. Unix, MS-DOS support minimum number of file structure.

File Type:

- File type refers to the ability of the operating system to distinguish different types of file such as text files source files and binary files etc. Operating system like MS-DOS and UNIX have the following types of files:

1. **Ordinary Files:**
   - Ordinary files are the files that contain user information.
   - Ordinary files may have text, databases or executable program.
   - The user can apply various operations on such files like add, modify, delete or even remove the entire file.
2. **Directory Files:**
   - Directory files contain list of file names and other information related to these files.
3. **Special Files:**
   - Special files are also known as device files.
   - Special files represent physical device like disks, terminals, printers, networks, tape drive etc.

Special files are of two types:

- **Character special file**s: Data is handled character by character as in case of terminals or printers.
- **Block special files:** Data is handled in blocks as in the case of disks and tapes.

1.4.3.2 Process Environment

- A process is a program execution.
- A process has the five components process id, code, data, stack, and CPU state. The process uses the CPU when it is scheduled. It also uses other resources. These include system resources like memory and user-created resources like files.

- The OS view of a process consists of two parts:
  1. Code and data areas of the process, including its stack, and resources allocated
  2. Information concerning program execution.

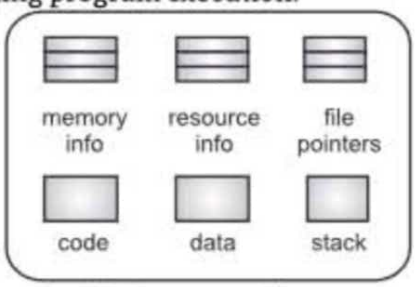


Fig. 1.6: Process environment

- It consists of a process environment and the process control block (PCB). The id of a process is used to access its process environment and PCB. This arrangement enables different OS modules to access process-related data conveniently and efficiently.
- Process environment Also called the process context, the process environment contains the address space of a process, i.e., its code, data, and stack, and all information necessary for accessing and controlling resources allocated to the process.
- The OS creates a process environment by allocating memory to the process, loading the process code in the allocated memory and setting up its data space. The OS also puts in information concerning access to resources allocated to the process, and its interaction with other processes and with the OS.
- Following table describes the components of the process environment. Contents of the process environment change during the execution of a process due to actions like file open and close, and dynamic creation and destruction of data by the process.
- Process environment consists of the code and data of the process and all information necessary for operation of the process.

Components of the Process Environment:

Sr. No.	Component	Description
1.	Code and data	Code of the program, including its functions and procedures, and its data, including the stack.
2.	Memory allocation Information	Information concerning memory areas allocated to the process. This information is used to implement memory accesses made by the process.
3.	Status of file processing activities	Pointers to files opened by the process, and current positions in the files
4.	Process interaction information	All information necessary to control interactions of the process with other processes, e.g. interprocess messages, signal handlers, ids of parent and child processes.
5.	Resource information	Information concerning resources allocated to a process.
6.	Miscellaneous information	Miscellaneous information needed for the interaction of a process with the OS.

1.5 WORKING WITH LINUX

Basics of Linux:

- Linux is a free version of UNIX. The free part is not meat in money terms but rather that the source code for Linux is freely available for inspection and modification.
- Linux is a multitask and multiuser operating system. An operating system is a collection of programs that run in a computer so that a person can easily access the hardware and all resources of the computers. The operating system is the big program that makes your computer life easy.
- A multitask operating system is capable of doing several tasks at the same time.
- A multiuser operating system is a computer system that allows multiple users that are on different computers to access single systems as resources simultaneously.

1.5.1 The Login Prompt

- A console is an all-text display mode that occupies the entire display monitor screen. A terminal window is a text-only window that emulates a console and which can be opened on a GUI screen
- Login is used when signing onto a system. If no argument is given, login prompts for the username. It can also be used to switch from one user to another at any time
- The login program is used to establish a new session with the system. It is normally invoked automatically by responding to the "login:" prompt on the user's terminal. Login may be special to the shell and may not be invoked as a sub-process. When called from a shell, login should be executed as exec login which will cause the user to exit from the current shell. Attempting to execute login from any shell but the login shell will produce an error message.
- The user is then prompted for a password, where appropriate. Echoing is disabled to prevent revealing the password. Only a small number of password failures are permitted before login exits and the communications link is break-off.
- If password aging has been enabled for your account, you may be prompted for a new password before proceeding. You will be forced to provide your old password and the new password before continuing; refer to our password for more information.
- Your user and group ID will be set according to their values in the /etc/passwd file. The value for $HOME, $SHELL, $PATH, $LOGNAME, and $MAIL are set according to the appropriate fields in the password entry. ulimit, umaskand nice values may also be set according to entries in the GECOS field.
- On some installations, the environment variable $TERM will be initialized to the terminal type on your line, as specified in /etc/ttytype.
- An initialization script may also be executed; check the documentation of your command interpreter for information on init scripts.
- A subsystem login is indicated by the presence of a "*" as the first character of the login shell. The given home directory will be used as the root of a new file system which the user is actually logged into.

Syntax:

```
login – Begin session on the system
login [-p] [ -h host ] [-H] [ -f username | username ]
```

- The user is then prompted for a password, where appropriate. Echoing is disabled to prevent revealing the password. Only a small number of password failures are permitted before **login** exits and the communications link is severed.
  - **-f** : Do not perform **authentication**; user is preauthenticated. In that case, **username** is mandatory.
  - **-h** : Name of the remote **host** for this login
  - **-p** : Preserve environment.
  - **-r** : Perform autologin protocol for rlogin

1.5.2 General Features of Commands/Command Structure

1.5.2.1 Command Line Features

- Following are some of the Command Line features:
  1. **Command History:** History option enables TL1 Agent to record all the commands which are executed in the history list. You can use UP-arrow or DOWN-arrow keys to traverse the history list. You can traverse the history list and modify or execute the command.
  2. **Command Completion:** Typing a letter (starting letter of the command) and pressing Tab key completes the TL1 command code. If more than one command code is registered starting with the typed letter, then all the matching ones are listed. Pressing the Tab key after entering the full command code displays the command syntax.
  3. **Command Line Editing:** Commands can be edited from command line using LEFT arrow key, RIGHT arrow key, and <DEL> key.
  4. **List of Commands**: Pressing "?" lists all the commands available for accessing the Agent. Pressing a command and "?" lists the matching command codes.
  5. **Complete Command Syntax**: Typing a command and pressing Tab key lists the complete command along with the usage for the command.
  6. **Escape Key:**

     Linux Commands:
     - File management and viewing
     - File system management
     - Help, Job and Process Management
     - Network Management
     - System Management
     - System management
     - User management
     - Printing and Programming
     - Document Prepartaio
     - Miscellaneous
- Typing a command and pressing ESC key clears the current line of text and returns to TL1 Prompt.

- Move up and down with PgUp and PgDn and move between the beginning and the end of a document with Home and End. End this viewing mode by pressing Q.
- Options without which the respective program cannot function are printed in italics.
- Further details, like file names, which must be passed to a command for correct functioning, are written in the Courier font.
- Specifications or parameters that are not required are placed in [brackets].
- Adjust possible specifications to your needs. It makes no sense to write ls file(s), if no file named file(s) actually exists. You can usually combine several parameters, for example, by writing ls -la instead of ls -l -a.

1.5.2.2 Command Structure

command [options] [parameters]:

- Multiple commands separated by; can be executed one after the other
- There can be more than one option and parameter for single command.
- For instance we used a cd .. command to go to previous directory.
- Dots ..are nothing more than parameters.
- Lets see what parameters we have with **ls** command.

ls -l:

- It displays list of files in long format. You can see that we get much more information by adding simple option -l.
- To display files in long format from different than working directory. Then we need to type in path to directory as parameter.

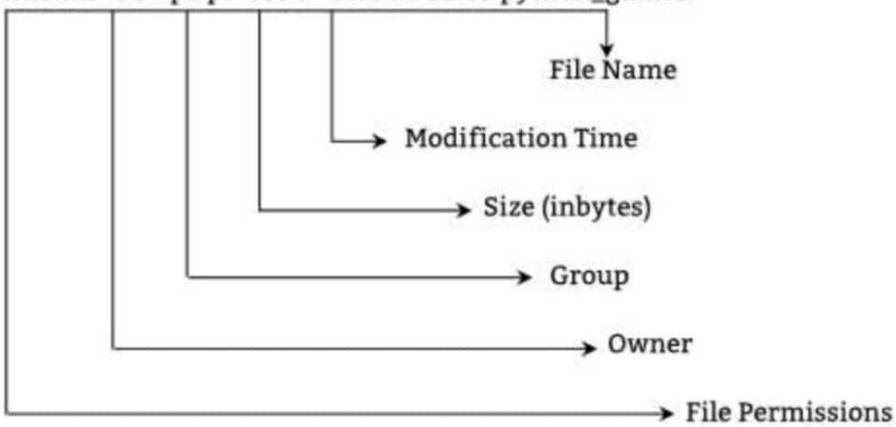
ls -l/usr:

- ls command can go with several options. Additionally we can include -a option which will show files where name starts with period (hidden). So we can write:

ls -l -a or ls - la

- Here is how to understand long file list:

```
drwxr-xr-x 2  pi pi 4096  May 25 20:47 Desktop
-rw-r--r-- 1  pi pi 5781  Feb  3 07:07 ocr_pi.png
drwxrwxr-x 2  pi pi 4096  Mar 10 12:20 python_games
```



- Couple handy default programs that can be helpful. First one is called less. It is used to view text file contents. Navigating is done with Page Up and Page down buttons.

Type for more controls:

less –help:

- Another handy program is called file. It helps to detect file type. It looks at file and tells what it contains. For instance we can run

  file ocr_pi.png

- It displays that file is PNG, 48×48 size and 8 bit color RGBA.
- This is by far not all that can be done with commands. Commands can be combined in to long command lines with pipes.

1.5.3 Command Arguments and Options

- Commands issued at the command line on any computer system are read and executed by a program known as a command interpreter.
- The Command Line Interface (CLI), is a non-graphical, text-based interface to the computer system, where the user types in a command and the computer then successfully executes it. The Terminal is the platform or the IDE that provides the command line interface (CLI) environment to the user.
- The CLI terminal accepts the commands that the user types and passes to a shell. The shell then receives and interprets what the user has typed into the instructions that can be executed by the OS (Operating System). If the output is produced by the specific command, then this text is displayed in the terminal. If any of the problems with the commands are found, then some error message is displayed.
- Some operating systems include the command interpreter in the kernel. Other operating system, such as Windows and UNIX, treat the command interpreter as a special program that is running when a user first logs on (on interactive systems).
- A command interpreter does the following:
  - Reads the commands and any options and arguments that you provide.
  - Translates or expands any special characters such as the * and ?used as wildcard characters on Linux and Unix systems .
  - Locates the command that you want to execute on your system.
  - Executes that command with the appropriate options and arguments and displays any output of that command.
- On Linux and Unix systems, a command interpreter is known as a shell. Linux and Unix systems offer many different shells, each of which has its own special features.

Examples Command Line Interface (CLI):

- MS-DOS (Microsoft Disk Operating System),
- CP/M (Control Program for Microcomputers) and
- Apple DOS (Apple Disk Operating System)

- Three main components of a command:
  1. **Command:** When entering a command, order matters. It's important to place the command name first - otherwise, the shell won't know how to process your submission.
  2. **Options:** Most commands come with certain options, which modify how the command is run. To apply an option, trail it immediately after your command with a hyphen (-).

     There is also a long option form of each short option that uses two hyphens (--). We'll stick to the short option for the sake of brevity.

     ```
     $ command-shortOptions
     $ command-longOptions
     ```
  3. **Argument:** A command is split into an array of strings named arguments. Argument 0 is (normally) the command name, argument 1, the first element following the command, and so on.

     Example:

     $ ls -la /tmp /var/tmp

     arg0 = ls

     arg1 = -la

     arg2 = /tmp

     arg3 = /var/tmp

1.6 UNDERSTANDING OF SOME BASIC COMMANDS SUCH AS ECHO, PRINTF, LS, WHO, DATE, PASSWD, CAL

- The command line is one of the most powerful features of Linux. There exists a sea of Linux command line tools, allowing you to do almost everything you can think of doing on your Linux PC.

1. **Echo:**

- The echo command displays whatever input text is given to it.

```
$ echo hello hi
hello hi
```

2. **LS:**

- The ls command lists contents of a directory in output.

```
$Ls progress
Capture.png hlist.o progress progress.hsizes.c
hlist.c LICENSE progress.1 progress.osizes.h
hlist.hMakefileprogress.c README.md sizes.o
```

3. **printf:**

- "printf" command in Linux is used to display the given string, number or any other format specifier on the terminal window. It works the same way as "printf" works in programming languages like C.

Syntax:

```
$printf [-v var] format [ arguments]
```

Format Specifiers: The most commonly used printfspecifiers are %s, %b, %d, %x and %f.

Examples:

1. **%s specifier:** It is basically a string specifier for string output.
2. **%b specifier:** It is same as string specifier but it allows us to interpret escape sequences with an argument.
3. **%d specifier:** It is an integer specifier for showing the integral values.
4. **%x specifier:** It is used for output of lowercase hexadecimal values for integers and for padding the output.

4. who:

- who command is used to find out the following information:
  1. Time of last system boot.
  2. Current run level of the system.
  3. List of logged in users and more.

Description: The who command is used to get information about currently logged in user on to system.

Syntax: `$who [options] [filename]`

5. Date:

- **date** command is used to display the system date and time. date command is also used to set date and time of the system. By default the date command displays the date in the time zone on which linux operating system is configured. You must be the super-user (root) to change the date and time.

Syntax:

```
date [OPTION]... [+FORMAT]
date [-u|--utc|--universal] [MMDDhhmm[[CC]YY][.ss]]
```

(i) **date (no option):** With no options, the date command displays the current date and time, including the abbreviated day name, abbreviated month name, day of the month, the time separated by colons, the time zone name, and the year.

Command: `$date`

(ii) **-u Option:** Displays the time in GMT (Greenwich Mean Time)/UTC (Coordinated Universal Time) time zone.

Command: `$date -u`

(iii) **--date or -d Option:** Displays the given date string in the format of date. But this will not affect the system's actual date and time value. Rather it uses the date and time given in the form of string.

Syntax:

```
$date --date=" string "
```

Command:

```
$date --date="18/07/2019"
$date --date="July 18 2019"
```

Output:

Thu July 2 00:00:00 PST 2019
Thu July 2 00:00:00 PST 2019

6. **passwd:**

- The passwd command changes passwords for user accounts.

Syntax:

```
Passwd [ options] [ LOGIN]
sunita@ansh:~$ passwd
changing password for sunita.
(current)UNIX password:
Enter new UNIX password:
Retype new UNIX password:
Passwd: password updated successfully
sunita@ansh:~$
```

7. **cal:**

- Cal command is used for to display calendar of a specific month or a whole year.

Syntax:

```
cal [  [ month ] year ]
cal: This command is used for to shows current month calendar.
cal 07 2019: This command is used for to shows calendar of selected
current month and year.
cal 2019: This command is used for to show the whole calendar of the
year.
cal -3: This command is used to shows calendar of previous, current and
next month.
```

1.7 COMBINING COMMANDS

- Linux command chaining is the method of combining several different commands such that each of them can execute in succession based on the operator that separate them.
- The important part of chaining is the operators that you use to combine them. These operators determine how the commands execute.

Logical AND Operator (&&):

- The logical AND operator will execute the commands that follow only if the current command is successful.

```
bash$ cd ~/temp/ &&rm -fr *
```

- The **rm** command will execute if and only if the **cd** command is successful. If for some reason, the **cd** command fails, this will make sure that the files in another directory (the current working directory) won't be accidentally deleted. You can chain several different commands using the AND operator.

```
Bash$ cd~/workspace/project/ &&svn up &&mvn clean install
Bash$ cd/usr/src/linux&& make && make modules_install&& install
```

- The above two examples show how you can automate a compilation processes for your projects. Using the && operator makes sure that every command in the chain succeeds. This is probably the most used operator, especially in the command line.

Logical OR Operator (||):

- The logical OR operator acts quite the opposite to the AND operator. The command that follows will only be executed if the preceding command fails or the exit status is non-zero.

```
Bash$ grep -nH apache /etc/passwd || grep -nH apache /etc/group
```

- The grep command first checks if there is a user named apache (searches the passwd file). If it cannot find a user, then it will search if there is a group named apache in the group file.

```
Bash$ find/usr/bin/ -iname"netstat" || find / sbin/ -iname "netstat" ||
find/bin/ -iname "netstat"
```

- The above command searches different folders one after an other, till it finds the file by that name.

Semi-Colon Operator (;):

- This is useful when you want to execute a bunch of commands in succession and does not care about the exit state of the commands. This works best when the commands does not depend on the previous commands.

```
Bash$svnup; mvn clean install; rm-fr/tmp/*
```

- The above commands will all execute no matter what. The **mvn** will clean and install even if the subversion (**svn**) fails to update successfully. Also, the **rm** command will clean up the /tmp/folder regardless of the exit state of **mvn**.

Pipe Operator (|):

- The **pipe** operator is useful when the second command wants to use the output of the first command as its input.

```
Bash$1s -1/path/|grep -I filename.txt
```

- The above command can probably be done in many different ways, but it shows you how the pipe operator works. The output of the command ls -l /path/ is used by **grep** as its input. So, the above command will print out all the files in the directory /path that match the name filename.txt.

Ampersand Operator (&):

- The ampersand operator at the end of the commands will run the command in the background. You can run multiple commands in the back ground too, if you append each of them with the ampersand operator

```
Bash$rm -fr~/documents/&shred-uz ~/importantfiles/ &rm -r/tmp/*.txt
```

- The above command will run the **rm** command to remove the files from the documents/folder, while shredding the files in the important files/folder. This is a useful method when you want to run multiple commands simultaneously or at the same time.

Redirection Operators (>, <, >>):

- The redirection operators can be used when you want to use a file or I/O stream other than the default standard input and standard output. You can use these operator to redirect either the standard input, standard output or both. Almost all commands will accept input using redirection operators.

```
Bash$ sort< inputfile.txt
```

- The above command will sort the contents of the file, named inputfile.txt. It will print out the sorted content into the standard output or the screen.

```
Bash$ 1s-1/path/>~/temp/filelist.txt
```

- The above command will save the output of the ls command into a file name filelist.txt. It is useful in couple of different scenarios: if you want to view the output at a later time or if the output is too long to fit into a screenful or so.

```
Bash$sort<inputfile.txt>outputfile.txt
```

- You can use both the input and output redirection operators together in the command. The above **sort** command will take the contents of inputfile.txt, sort them and then save them as outputfile.txt.
- The other redirection operator is **>>**, which works exactly like the **>** operator but will append the output to the file rather than overwriting the output file stream.
- The above six operators are probably the ones that you will regularly use from command line. As I mentioned, there are more that you can use, but they are usually useful when writing much more advanced shell scripts rather than having to type it out in the command line.
- The above set are the most commonly used operators. However, there are several more that you probably should know. These set of operators are more of conditional in nature, or are used for grouping. They are very useful when writing long and complicated set of commands. They are much more commonly found in shell scripts.
  - **Logical NOT (!)**: This is an operator that is more useful when you want to negate an expression with in a command.
  - **AND-OR (&& — ||)**: This combination of AND and OR operators can be used to create an if-then-else condition.
  - **Concatenation (\)**: This is useful when you have to split the command between multiple lines.
  - **Precedence ()**: This allows several commands to be grouped, and all of the grouped commands are executed in a sub-shell.
  - **Combination ({})**: This can be used to group commands into different sets and the commands are executed within the same shell.

Summary

- Linux is an operating system that works just like Windows and Mac OSX.
- Linux Operating System (OS) is an interface between computer user and computer hardware. An operating system is software which performs all the basic tasks like file management, memory management, process management, handling input and output, and controlling peripheral devices such as disk drives and printers.
- A process is waiting for an event to occur or for a system resource.
- In this chapter gives insight into the most important commands of Linux system. Along with the individual commands, parameters are listed and, where appropriate, a typical sample application is introduced. To learn more about the various commands, it is usually possible to get additional information with the man program
- An argument, also called a command line argument, is a file name or other data that is provided to a command in order for the command to use it as an input
- Command line arguments (also known as positional parameters) are the arguments specified at the command prompt with a command or script to be executed. The locations at the command prompt of the arguments as well as the location of the command, or the script itself, are stored in corresponding variables.

Check Your Understanding

Multiple Choice Questions with Answer: **(10)**

1. System structure of Linux is _____.
   (a) Microsoft Windows (b) UNIX
   (c) Window Vista (d) Monolithic Kernel
2. File name that handle interrupts in Linux is _____.
   (a) Access file (b) Control file
   (c) Interrupts file (d) Proc interrupts file
3. Process control block (PCB) is most important _____.
   (a) Data Access Type (b) Data Algorithm
   (c) Data Structure (d) Data Block
4. Process location tells about location of _____.
   (a) Processes (b) Data
   (c) Files (d) Memory
5. Process which terminates before the parent process exits is known as _____.
   (a) Orphan (b) Zombie
   (c) Child (d) None of the above
6. Which command is used to check Linux version?
   (a) uname –a (b) uname -n
   (c) uname –s (d) kernel
7. The main function of the command interpreter is _____.
   (a) To get and execute the next user-specified command
   (b) To provide the interface between the API and application program
   (c) To handle the files in operating system.
   (d) none of the mentioned

8. Which command is used to display the unix version _____.
   (a) uname –r (b) uname –n
   (c) uname –t (d) kernel
9. Which of the following command is used to create file in Linux?
   (a) touch (b) cat
   (c) echo (d) All of the above

Answers

1. (d)	2. (d)	3. (c)	4. (a)	5. (b)	6. (a)	7. (a)	8. (a)	9. (a)

Practice Questions

Short Answer Questions:

1. What are the different services provided by operating systems?
2. What are the major activities of operating systems with regard to process management?
3. What is a process?
4. What is file structure?
5. Define process control system call.

Long Answer Questions:

1. Write short notes on echo, printf, date, passwd and who.
2. Explain structure of process control block.
3. Explain features of linux.
4. Discuss the Simple Operating System Structure.
5. Explain the features of commands.
6. Write short note on login prompt?
7. What are the different services provided by operating system to user?
8. Explain different process states in detail.
9. Write short note on user interface?
10. Write short note on system call?
11. Write short note on process environment?

■■■

2... Commands and Processes

Objectives...

- To study Commands and Processes.
- To understand the Process of Operating System.
- To learn the various Internal and External Commands.
- To study man command.

2.1 INTRODUCTION

- A process is compiled source code that is currently running on the system.
- There are generally two types of processes that run on Linux- Foreground Processes and Background Processes.
- A command when executed, a special instance is provided by the system to the process. This instance consists of all the services/resources that may be utilized by the process under execution.
- Whenever a command is issued in Unix/Linux, it creates/starts a new process.

2.2 WHAT IS A COMMAND?

- A command is a specific order from a user to the computer's operating system or to an application to perform a service.
- A command is an instruction given by a user telling a computer to do something, such a run a single program or a group of linked programs. Commands are generally issued by typing them in at the command line.
- A command consists of a command name usually followed by one or more strings (i.e., sequences of characters) that comprise options and arguments. Each of these strings is separated by white space (which consists of one or more spaces or tabs).
- The general syntax for commands is:

```
command [options] [arguments]
```

- The square brackets indicate that the enclosed items are optional. Most commands have at least a few options and can accept (or require) arguments. However, there are some commands that do not accept arguments, and a very few with no options.

2.3 MEANING OF INTERNAL AND EXTERNAL COMMANDS

2.3.1 Internal Commands

- The commands that are directly executed by the shell are known as Internal Commands. No separate process is there to run these commands.
- Internal commands are commands that are already loaded in the system. They can be executed any time and are independent.
- Internal commands are the built in commands of the shell. Which means that when you execute an internal command, no process will be launched to execute the command. Therefore the speed of executing an internal command will be very high. Example – cd, pwd, echo etc.

2.3.2 External Commands

- External commands are powerful commands and can help fix problems, improve performance, or perform other actions.
- The commands that are executed by the kernel are knows as External Commands. Each command has its unique process id.
- External commands are those commands which are stored as separate binaries. Shell starts separate sub-process to execute them. Most external commands are stored in the form of binaries in /bin directory.
- To execute external command shell check $PATH variable. If command present in the location mentioned in $PATH variable shell will execute it, otherwise it will give error. Example – ls, mv, cat etc.

2.4 TYPE COMMAND

- The type command is a shell built-in that displays the kind of command the shell will execute, given a particular command name,
- It works like this:

```
type command
```

where command is the name of the command you want to examine. Here are some examples:

```
[me@linuxbox ~] $ type type
type is a shell built-in
```

- But in which directory is the command located? The easiest way of knowing the location of a command file is to use the type command:

```
$ type ls
ls is /bin/ls
```

- ls is located in the /bin directory and because /bin is also a component of the value of the PATH variable, the system locates it easily and executes it.

Type -t Command:

- It will show a single word or string. It tells a command or name is an alias, shell reserved word, function, built-in, or disk file(external command).
- In case command is not found, nothing will be printed as output. When we check exit status of command, it will be false.

Example:

```
root@tuxworld:~# type -t ls
alias
root@tuxworld:~#
root@tuxworld:~# type -t date
file
root@tuxworld:~#
root@tuxworld:~# type -t declare
builtin
root@tuxworld:~#
root@tuxworld:~# type -t rvm
function
root@tuxworld:~#
```

Type -p Command:

- The command of the disk file that would be executed. In other words, absolute path of command.

  Example:

```
root@tuxworld:~# type -p date
/bin/date
root@tuxworld:~#
```

Type -P Command:

- Force a PATH search for each command/NAME, even if it is an alias, built-in, or function, and returns the name of the disk file that would be executed.

  Example:

```
root@tuxworld:~# type -P date
/bin/date
root@tuxworld:~#
root@tuxworld:~# echo $PATH
/usr/local/rvm/gems/ruby-2.1.0/bin:/usr/local/rvm/gems/ruby-
2.1.0@global/bin:/usr/local/rvm/rubies/ruby-
2.1.0/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/
usr/local/rvm/bin
root@tuxworld:~#
```

Type -a Command:

- It display all locations have command or NAME. It includes aliases, builtins, and functions (only in case , if -p option is not used)

  Example:

```
root@tuxworld:~# type -a echo
echo is a shell builtin
echo is /usr/sbin/echo
echo is /bin/echo
root@tuxworld:~#
```

Type -f Command:

- The -f option, suppress shell function lookup.

  Example: `type -f rvm`

2.5 man COMMAND IN LINUX WITH EXAMPLES

- man command in Linux is used to display the user manual of any command that we can run on the terminal. It provides a detailed view of the command which includes NAME, SYNOPSIS, DESCRIPTION, OPTIONS, EXIT STATUS, RETURN VALUES, ERRORS, FILES, VERSIONS, EXAMPLES, AUTHORS.
- Most executable programs intended for command-line use provide a formal piece of documentation called a manual or man page. A special paging program called man is used to view them, like this:

  ```
  man program
  ```

  where program is the name of the command to view.
- Man pages vary somewhat in format but generally contain a title, a synopsis of the command's syntax, a description of the command's purpose, and a listing and description of each of the command's options. Man pages, however, do not usually include examples, and they are intended as a reference, not a tutorial. As an example, let's try viewing the man page for the ls command:

  ```
  [me@linuxbox ~]$ man ls
  ```
- Onmost Linux systems, man uses less to display the manual page, so all of the familiar less commands work while displaying the page.
- The "manual" that man displays is bro ken into sections and covers not only user commands but also system administration commands, programming interfaces, file formats, and more.

Every manual is divided into the following sections:

  - Executable programs or shell commands.
  - System calls (functions provided by the kernel).
  - Library calls (functions within program libraries).
  - Games.
  - Special files (usually found in /dev).
  - File formats and conventions eg /etc/passwd.
  - Miscellaneous (including macro packages and conventions), e.g. groff(7).
  - System administration commands (usually only for root).
  - Kernel routines [Non standard].

- Sometimes we need to look in a specific section of the manual to find what we are looking for. This is particularly true if we are looking for a file format that is also the name of a command. If we don't specify a section number, we will always get the first instance of a match, probably in section 1. To specify a section number, we use man like this:

  ```
  man sectionsearch term
  ```

  For example:

  ```
  (me@linuxbox ]$ man 5 passwd
  ```

  will display the man page describing the file fo r mat of the /etc/passwdfile.

2.5.1 The man Command with Keyword Option

- An option, also referred to as a flag or a switch, is a single-letter or full word that modifies the behavior of a command in some predetermined way.
- Options are used on the command line following the name of the command and before any arguments. They are separated from the name of the command and arguments by at least one space.

  Syntax: `$man [OPTION] [COMMAND NAME].`

Options and Examples:

1. **No Option:** It displays the whole manual of the command.

   Syntax: `$ man [COMMAND NAME]`

   Example: `$ man printf`

   In this example, manual pages of the command 'printf' are simply returned.
2. **Section-num**: Since a manual is divided into multiple sections so this option is used to display only a specific section of a manual.

   Syntax: `$ man [SECTION-NUM] [COMMAND NAME]`

   Example: `$ man 2 intro`

   In this example, the manual pages of command 'intro' are returned.
3. **-f option**: One may not be able to remember the sections in which a command is present. So this option gives the section in which the given command is present.

   Syntax: `$ man -f [COMMAND NAME]`

   Example: `$ man -f ls`

   In this example, the command 'ls' is returned with its section number.
4. **-a option**: This option helps us to display all the available intro manual pages in succession.

   Syntax: `$ man -a [COMMAND NAME]`

   Example: `$ man -a intro`

   In this example you can move through the manual pages(sections) i.e either reading(by pressing Enter) or skipping(by pressing ctrl+D) or exiting(by pressing ctrl+C).
5. **-k option**: This option searches the given command as a regular expression in all the manuals and it returns the manual pages with the section number in which it is found.

   Syntax: `$ man -k [COMMAND NAME]`

   Example: `$ man -k cd`

   The command 'cd' is searched in all the manual pages by considering it as a regular expression.
6. **-w option**: This option returns the location in which the manual page of a given command is present.

   Syntax: `$ man -w [COMMAND NAME]`

   Example: `$ man -w ls`

   The location of command 'ls' is returned.

7. **-I option**: It considers the command as case sensitive.

   Syntax: `$ man -I [COMMAND NAME]`

   Example: `$ man -I printf`

   The command 'printf' is taken as case-sensitive i.e 'printf' returns the manual pages but 'Printf' gives error.

2.5.2 The man Command with Keyword whatis

- The whatis program displays the name and a one-line description of a man page matching a specified keyword.

  The syntax for whatis is:

  ```
  whatis keyword(s)
  ```

  The whatis command is equivalent to man –f command.

  The man command, when used with its -f option, produces the same output as whatis. Thus, for example,

  ```
  man -f cat
  ```

  is equivalent to

  ```
  whatis cat
  ```

2.6 OPERATING SYSTEM PROCESSES - CONCEPT

- A process is the instance of a computer program that is being executed by one or many threads. It contains the program code and its activity. Depending on the operating system (OS), a process may be made up of multiple threads of execution that execute instructions concurrently.
- While a computer program is a passive collection of instructions, a process is the actual execution of those instructions. Several processes may be associated with the same program; for example, opening up several instances of the same program often results in more than one process being executed.

Definition:

- A process is defined as an entity which represents the basic unit of work to be implemented in the system.
- Each process is uniquely identified by a number called the process identifier (PID) that is allotted by the kernel when the process is born. Apart from the PID, a process also has the following attributes:
  - The real user-id of the user who created the process. The user is then said to be the owner of the process.
  - The real group-id of the owner of the process. Both user-id and group-id of the owner are stored in / etc / passwd.
  - The priority with which it runs. The Kernel's process scheduler uses this value to determine the process that has to run next.
  - The current directory from where the process was run.
- To put it in simple words, we write our computer program in text file, when we execute our program it becomes a process which performs all the tasks mentioned in the program.

- When a program is loaded into the memory and it becomes a process, it can be divided into four section stack, heap, text and data. Below is a simplified layout of a process inside main memory:

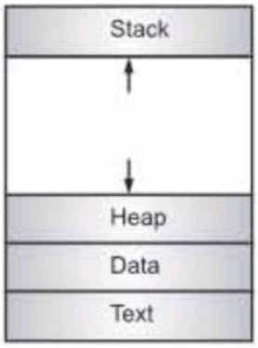


Fig. 2.1: Layout of a process inside main memory

1. **Stack**: The process Stack contains the temporary data such as method/function parameters, return address and local variables.
2. **Heap:** This is dynamically allocated memory to process during its run time.
3. **Text:** This includes the current activity represented by the value of Program Counter and the contents of the processor's registers.
4. **Data**: This section contains the global and static variables.

2.6.1 Mechanism of Process Creation

- As noted before, a process is just a program that is running. Like a file, a process is also a sequence of bytes interpreted as instructions to be run by the CPU. This is often an executable program which contains the binary code to be executed, along with data that would be required for the program to run. These data comprise the variables and arrays that you find in program code.
- The process image as viewed by the kernel runs in its own user addressspace – a protected space which can't be disturbed by other users. This address space bas a number of segments:
  - **Text Segment:** This contains the executable code of a program. Since several users may be using the same program, this area generally remains fixed.
  - **Data Segment:** All variables and arrays the program uses are held here. They include those variables and arrays that have values assigned to them during program execution.
  - **User Segment:** This contains all the process attributes that were discussed earlier the UIDs and GUIDs and the current directory. The kernel uses the information stored in this segment to manage all processes.
- There are three distinct phases in the creation of a process, using three important system calls: **fork(), exec()** and **wait()**. Knowledge of the role they play in process creation can often help you debug shell scripts and C programs. The three phases are discussed below:
  - **FORK:** A process in UNIX is created with the fork system call, which creates a copy of the process that invokes it. For example, when you enter a command at the prompt, the shell first creates a copy of itself. The image is practically identical to the calling process, except that the forked process gets a new PID. The forking mechanism is responsible for the multiplication of processes in the system.

- **EXEC:** The parent then overwrites the image it has just created with the copy of the program that has to be executed. This is done with the exec system call, and the parent is said to exec this process. No additional process is created here; the existing program's text and data areas are simply replaced (or overlaid) with the ones of the new program. This process has the same PID as the child that was just forked.
- **WAIT:** The parent then executes the wait system call to keep waiting for the child process to complete. When the child has completed execution, it sends a termination signal to the parent. The parent is then free to continue with its other functions.

2.6.2 Parent and Child Process

Parent Process:

- All the processes in operating system are created when a process executes the fork() system call except the startup process. The process that used the fork() system call is the parent process.
- In other words, a parent process is one that creates a child process. A parent process may have multiple child processes but a child process only one parent process.
- On the success of a fork() system call, the PID of the child process is returned to the parent process and 0 is returned to the child process. On the failure of a fork() system call, –1 is returned to the parent process and a child process is not created.

Child Process:

- A child process is a process created by a parent process in operating system using a fork() system call. A child process may also be called a subprocess or a subtask.
- A child process is created as its parent process's copy and inherits most of its attributes. If a child process has no parent process, it was created directly by the kernel.
- If a child process exits or is interrupted, then a SIGCHLD signal is send to the parent process.
- The environmental parameters of a process are generally made available to all its children, but changes made by a child to its own environment are not communicated to the parent. This means that values of variables defined or redefined in the child are not visible in the parent.

2.7 ps COMMAND WITH ITS OPTIONS

- The ps (process status) command is used to provide information about the currently running processes, including their process identification numbers (PIDs).
- A process, also referred to as a task, is an executing instance of a program. Every process is assigned a unique PID by the system.
- The basic syntax of ps is,

```
ps [options]
```

Example:

```
$ ps
PID        TTY        TIME        CMD
659        tty03      00:00:01    sh          Shell of user who invoked ps
684        tty03      00:00:00    ps
```

- PID is a Process ID of the running command (CMD).
- TTY is a place where the running command runs.
- TIME tell about how much time is used by CPU while running the command.
- CMD is a command that runs as a current process.

- When ps is used without any options, it sends to standard output, which is the display monitor by default, four items of information for at least two processes currently on the system: the shell and ps.
- A shell is a program that provides the traditional, text-only user interface in Unix-like operating systems for issuing commands and interacting with the system, and it is bash by default on Linux.

Option	Description
-a	Displays all processes on a terminal, with the exception of group leaders.
-c	Displays scheduler data.
-d	Displays all processes with the exception of session leaders.
-e	Displays all processes.
-f	Displays a full listing.
-glist	Displays data for the list of group leader IDs.
-j	Displays the process group ID and session ID.
-l	Displays a long listing.
-plist	Displays data for the list of process IDs.
-slist	Displays data for the list of session leader IDs.
-tlist	Displays data for the list of terminals.
-ulist	Displays data for the list of usernames.

2.8 at COMMAND

- The at command schedules a command to be run once at a particular time that you normally have permission to run.
- The at command can be anything from a simple reminder message, to a complex script. You start by running the at command at the command line, passing it the scheduled time as the option. It then places you at a special prompt, where you can type in the command (or series of commands) to be run at the scheduled time.
- When you're done, press Control-D on a new line, and your command will be placed in the queue.

- Commands used with at:
  - **at**: execute commands at specified time.
  - **atq**: lists the pending jobs of users.
  - **atrm**: delete jobs by their job number.

Specifying Time:

- at uses a very casual representation of time and date. It even knows some "commonly used" times you might not expect — it knows that "teatime" is traditionally at 4 PM, for instance.
- Here are some examples of times you can pass to at to schedule a command.

Examples of at Command:

Example 1: Schedule task at coming 10:00 AM.
$ at 10:00 AM

Example 2: Schedule task at 10:00 AM on coming Sunday.
$ at 10:00 AM Sun

Example 3: Schedule task at 10:00 AM on coming 25'th July.
$ at 10:00 AM July 25

Example 4: Schedule task at 10:00 AM on coming 22'nd June 2019.
$ at 10:00 AM 6/22/2019
$ at 10:00 AM 6.22.2019

Example 5: Schedule task at 10:00 AM on the same date at next month.
$ at 10:00 AM next month

Example 6: Schedule task at 10:00 AM tomorrow.
$ at 10:00 AM tomorrow

Example 7: Schedule task at 10:00 AM tomorrow.
$ at 10:00 AM tomorrow

Example 8: Schedule task to execute just after 1 hour.
$ at now + 1 hour

Example 9: Schedule task to execute just after 30 minutes.
$ at now + 30 minutes

Example 10: Schedule task to execute just after 1 and 2 weeks.
$ at now + 1 week
$ at now + 2 weeks

Example 11: Schedule task to execute just after 1 and 2 years.
$ at now + 1 year
$ at now + 2 years

Example 12: Schedule task to execute at midnight.
$ at midnight
The above job will execute on next 12:00 AM

2.9 THE NICE COMMAND

- Processes are usually executed with equal priority. This is not always desirable since high-priority jobs must be completed earliest LINUX offers the nice command, which is used with the and operator to reduce the priority of jobs. More important jobs can the n have greater access to the system resources (being "nice" to your neighbours).
- nice runs command COMMAND with an adjusted "niceness", which affects process scheduling. A process with a lower niceness value is given higher priority and more CPU time.
- A process with a higher niceness value (a "nicer" process) is given a lower priority and less CPU time, freeing up resources for processes that are more demanding. Niceness values range from -20 (most favorable to the process) to 19.
- **Syntax:** `nice [OPTION] [COMMAND [ARG]...]`
- To check the existing scheduling priority before changing it. If it's about the process you're about to run, then you should know the default scheduling priority is always 0.
- For example, we executed the following process:

  ```
  ./test-new
  ```
- The priority confirmation using the following command:

  ```
  ps -lu sunitapatil | grep test-new
  ```

  Here, 'sunitapatil' is the user who owns the 'test-new' process.
- How the nice command works? It's easy - just use the tool in the following way:

  ```
  nice -PRIORITY COMMAND
  ```
- For example, if I want the scheduling priority to be 10, here's how I can do that:

  ```
  nice -10 ./test-new
  ```

Usage of nice command:

- Now let's assume the case that system has only 1GB of RAM and it's working really slow, i.e. programs running on it(processes) are not responding quickly, in that case if you want to kill some of the processes, you need to start a terminal, if you start your bash shell normally, it will also produce lag but you can avoid this by starting the bash shell with high priority.

  For example: `nice -n -5 bash`

2.10 BACKGROUND PROCESSES

- A background process is the process that runs in the background, without user involvement. Typical tasks for these processes include logging, system monitoring, scheduling, and user notification.
- For example, Apache or Nginx web server always runs in the background to serve you images and dynamic content.
- The background process usually is a child process created by a control process for processing computing task. After the creation, the child process will run on its own course for performing the task independent of the control process, therefore, the control process is free of performing other designated task.
- A multitasking system lets a user do more than one job at a time. Since there can be only one job in the foreground, the rest of the jobs have to run in the background.

2.10.1 Foreground(fg) and Background(bg) Commands

- This will run the **wc** command in the foreground. The **fg** and **bg** commands can also be used with the job number, job name or a string as arguments, prefixed by the % symbol.

1. Fg Command:

- A foreground process is one that occupies your shell (terminal window), meaning that any new commands that are typed have no effect until the previous command is finished.

  fg = continues a stopped job by running it in the foreground.

  Syntax:

  ```
  fg [job_spec]
  ```

  job_spec may be:

  - **%n:** Refer to job number n.
  - **%str:** Refer to a job which was started by a command beginning with str.
  - **%?str:** Refer to a job which was started by a command containing str.
  - **%% or %+:** Refer to the current job. fg and bg will operate on this job if no job_spec is given.
  - **%-:** Refer to the previous job.

  Options for fg command:

  - **fg [JOB_SPEC]:** This command is used to put the mentioned job running in background to foreground.
  - **fg –help:** It displays help information.

    Examples:

    ```
    fg
    ```

    Typing fg will resume the most recently suspended or backgrounded job.

    ```
    fg 1
    ```

    Brings the job with the id 1 into the foreground, resuming it if it was suspended.

2. Bg Command:

- **bg** command in linux is used to place foreground jobs in background.
- Normally user can run a job in background, by adding & at end of the command (ex: sleep 10 &).

  Syntax:

  ```
  bg [job_spec ...]
  ```

  job_spec may be:

  - **%n:** Refer to job number n.
  - **%str:** Refer to a job which was started by a command beginning with str.
  - **%?str:** Refer to a job which was started by a command containing str.
  - **%% or %+:** Refer to the current job. fg and bg will operate on this job if no job_spec is given.
  - **%-:** Refer to the previous job.

Options for bg command:

- **bg [JOB_SPEC]**: This command is used to put the mentioned job in background.
- **bg –help**: This command displays help information.

Examples: In this example, let's assume we are in the bash shell.

If you initiate a process at the command line, and you want to return to the command line prompt before the program is finished executing to do something else (e.g., check your mail, edit a text file, whatever) you can press Control-Z and the job will stop.

If you then run the command:

```
bg %1
```

The stopped job will resume operation, but remain in the background. It will not receive any input from the terminal while it's in the background, but it will keep running, and you can continue to use the shell from the command line.

2.10.2 kill Command

- kill command in Linux (located in /bin/kill), is a built-in command which is used to terminate processes manually.
- The kill command is used to "kill" processes. This allows us to terminate programs that need killing.
- kill command sends a signal to a process which terminates the process. If the user doesn't specify any signal which is to be sent along with kill command then default TERM signal is sent that terminates the process.

1. **kill -l:** To display all the available signals you can use below command option:

   Syntax: `$kill -l`

   Signals can be specified in three ways:

   By number (e.g. -5)

   With SIG prefix (e.g. -SIGkill)

   Without SIG prefix (e.g. -kill)

   ```
   $ps
   ```

2. **kill pid:** To show how to use a PID with the kill command.

   Syntax: `$kill pid`

3. **kill -s:** To show how to send signal to processes.

   Syntax: `kill {-signal | -s signal} pid`

4. **kill -L:** This command is used to list available signals in a table format.

   Syntax: `kill {-l | --list[=signal] | -L | --table}`

2.10.3 Find Command

- The find command in UNIX is a command line utility for walking a file hierarchy. It can be used to find files and directories and perform subsequent operations on them. It supports searching by file, folder, name, creation date, modification date, owner and permissions.

  Syntax: $ find [where to start searching from] [expression determines what to find] [-options] [what to find]

Options:

- **-exec CMD:** The file being searched which meets the above criteria and returns 0 for as its exit status for successful command execution.
- **-ok CMD:** It works same as -exec except the user is prompted first.
- **-inumN:** Search for files with inode number 'N'.
- **-links N:** Search for files with 'N' links.
- **-name demo:** Search for files that are specified by 'demo'.
- **-newer file:** Search for files that were modified/created after 'file'.
- **-perm octal:** Search for the file if permission is 'octal'.
- **-print:** Display the path name of the files found by using the rest of the criteria.
- **-empty:** Search for empty files and directories.
- **-size +N/-N:** Search for files of 'N' blocks; 'N' followed by 'c'can be used to measure size in characters; '+N' means size > 'N' blocks and '-N' means size < 'N' blocks.
- **-user name:** Search for files owned by user name or ID 'name'.
- **\(expr \):** True if 'expr' is true; used for grouping criteria combined with OR or AND.
- **! expr:** True if 'expr' is false.

For Example:

1. **Search a file with specific name.**

   ```
   $ find ./GFG -name sample.txt
   ```

   It will search for sample.txt in GFG directory.

2. **Search a file with pattern.**

   ```
   $ find ./GFG -name *.txt
   ```

   It will give all files which have '.txt' at the end.

3. **How to find and delete a file with confirmation.**

   ```
   $ find ./GFG -name sample.txt -exec rm -i {} \;
   ```

   When this command is entered, a prompt will come for confirmation, if you want to delete sample.txt or not. if you enter 'Y/y' it will delete the file.

4. **Search for empty files and directories.**

   ```
   $ find ./GFG -empty
   ```

   This command find all empty folders and files in the entered directory or sub-directories

5. **Search for file with entered permissions.**

   ```
   $ find ./GFG -perm 664
   ```

   This command find all the files in the GFG directory or sub-directory with the given permissions.

6. **Search text within multiple files.**

   ```
   $ find ./ -type f -name "*.txt" -exec grep 'Geek' {} \;
   ```

   This command print lines which have 'Geek' in them and '-type f' specifies the input type is a file.

Summary

- Internal command are built in the COMMAND.COM files.
- Internal Commands: Commands which are built into the shell. External Commands: Commands which aren't built into the shell.
- man command in Linux is used to display the user manual of any command that we can run on the terminal.
- nice command execute a program/process with modified scheduling priority, the renice command allows you to change the scheduling priority of an already running process.
- A command is a specific order from a user to the computer's operating system or to an application to perform a service.
- type command is used for displaying information about command type.
- A parent process is one that creates a child process. A parent process may have multiple child processes but a child process only one parent process.
- A child process is a process created by a parent process in operating system using a fork() system call. A child process may also be called a subprocess or a subtask.
- The at command schedules a command to be run once at a particular time that you normally have permission to run.
- A background process is the process that runs in the background, without user involvement. Typical tasks for these processes include logging, system monitoring, scheduling, and user notification.
- A foreground process is one that occupies your shell (terminal window), meaning that any new commands that are typed have no effect until the previous command is finished.
- kill command in Linux (located in /bin/kill), is a built-in command which is used to terminate processes manually.

Check Your Understanding

1. The command 'umask -S' _____.
   (a) prints the current mask using symbolic notation
   (b) prints the current mask using octal numbers
   (c) sets the mask to 000
   (d) sets the mask to 777
2. Which option of the kill command sends the given signal name to the specified process?
   (a) -l (b) -n
   (c) -s (d) -a
3. The 'logout' built in command is used to _____.
   (a) shutdown the computer (b) logoff of the computer
   (c) logout the current user (d) to exit the current shell
4. _____ command will bring the background jobs to the foreground.
   (a) bg (b) fg
   (c) ctrl-Z (d) kill

5. Which command will push the current foreground job to the background?
   (a) bg (b) fg
   (c) ctrl-Z (d) kill
6. The command fg %1 will bring the first background job to the foreground.
   (a) True (b) False
7. What does the following command do?
   $ kill %2
   (a) kills job number 2 (b) kills the second background job
   (c) invalid command (d) kill all foreground & background jobs
8. POSIX shell provides job control facilities like bg or fg.
   (a) True (b) False
9. Which one of the following forms used with at command is invalid?
   (a) at noon (b) at now +2 years
   (c) at 3:07 + 1 day (d) at morning
10. We can use the _____ symbol with at command to redirect our output to a specified file.
    (a) > (b) <
    (c) >> (d) %

Answers

1. (a)	2. (c)	3. (d)	4. (b)	5. (a)	6. (a)	7. (b)	8. (b)	9. (d)	10. (a)

Practice Questions

Short Answer Questions:

1. Define background process.
2. Define Internal and External command?
3. Define parent and child processes.
4. What is the use of find command.
5. Explain nice command.

Long Answer Questions:

1. Explain type command in detail.
2. Explain Foreground(fg) and Background(bg) commands.
3. Explain find command with suitable example.
4. Discuss the fork() and exec().
5. Explain man command with keyword option and whatis.
6. Write short note on manual pages.
7. Explain ps command with its options.
8. Explain at command in detail.
9. Write short note on kill command.
10. Explain find command in detail.

■■■

3...

File System

Objectives...

- To learn file types – ordinary, directory, device files, pipes.
- To understand Linux File Structure, and Absolute and Relative path.
- To understand organization of files in Linux.
- To study how to use $Home variable, $PATH Variable.
- To study file related commands such as cat, mv, rm, cp, wc and od commands and ls command with options.
- To understand relative and absolute permissions changing methods with chmod command

3.1 INTRODUCTION

- A file system is the methods and data structures that an operating system uses to keep track of files on a disk or partition; that is, the way the files are organized on the disk. The word is also used to refer to a partition or disk that is used to store the files or the type of file system.
- Linux systems have thousands of files, if you write a program or text; you add one more file to the system. Files grow rapidly, and if they are not organizing them properly it will be difficult to find them. So Linux organizes files into directories.
- It allows users to access files which are not belonging to them, as well as it provides security features with files so that outsiders cannot interface the files.
- Most Linux file systems have similar general structure, although the exact details slightly vary.
- An inode contains all information about a file, except its name. The name is stored in directory, together with the number of the inode. The inode contains the numbers of several data blocks, which are used to store the data in the file.
- Linux chooses to have a single hierarchical directory structure. Everything starts from the root directory, represented by /, and then expands into sub directories instead of having drives like windows environment.
- If you install a program in Windows, it usually store most of its files in its own directory structure. For instance c:\Program Files\program – name. In Linux, programs put their documentation into usrshare/doc/program-name. Files on Linux are case sensitive. This means that File1 is different from FILE.

3.2 LINUX FILES – NAMING FILES

- In Linux system, everything is a file and if it is not a file, it is a process. A file doesn't include only text files, images and compiled programs but also include partitions, hardware device drivers and directories.
- A File is container for storing information. In file we can store sequence of characters. Linux files doesnt contain eof(end of file) marks. All files attributes are kept in a separate area of the disk.
- A File is the basic structure used to store information on the Linux system. A file can contain any kind of information that can be represented as a sequence of bytes.
- A file can store manuscripts and other word processing documents, instructions or programs for the computer system itself. A file name uniquely identifies the file. Linux system keeps track of where the file is located and maintains other relevant information about the file.

3.2.1 Basic File Types

- A file name can be almost of any sequence of characters. The Linux system places few restrictions on how to name the file. Use any ASCII character in a file name upto 255 characters except the null character or the slash(/), which has special meaning in Linux file system. The slash acts as a separator between directories and files.
- Mostly, the file name is divided into two parts. First part is filename and second part optionally followed by a period and extension. Lets try to display files using ls command.

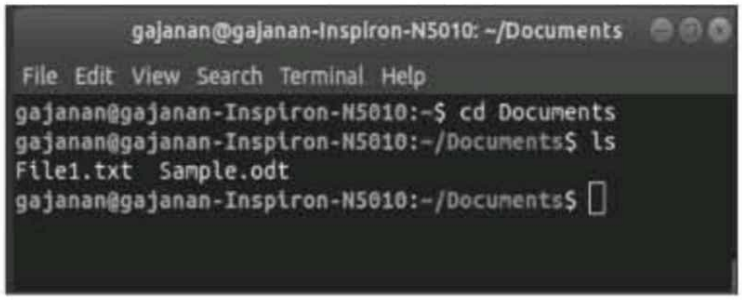


Fig. 3.1

- Although everything is treated as a file by Linux, it is necessary to divide these files into three categories. These are:

1. **Ordinary or Regular Files**.

- A large majority of the files found on Linux systems are ordinary files .
- Ordinary files contain ASCII (human-readable) text, executable program binaries, program data, and more.

2. **Directory Files:**

- It is shown in blue color. It contains their names and a number associated with each name. They contain several files grouped into it.

- Directory does not contain any data, but it keeps details of files and its sub directories it contain.
- A Linux file system is organized with number of directories and sub directories. Also we can create new directories. We group all files in directories and we can have same name of the file in two different directories.
- These types of files contain regular files/folders/special files stored on a physical device. And this type of files will be in blue color.
- **Example**: ls -l / | grep "^d" this command will display all directories. Also you can use mkdir command to create directories. And rmdir command to remove empty directories.
- If you have 10 files in a directory, there will be 10 entries in the directory. Each entry has two components:
  1. The Filename.
  2. A unique identification number for the file or directory (called the inode number).
- When an ordinary file is created or removed, its corresponding directory file is automatically updated by the kernel with the relevant information about the file.

3. Device Files (Special Files):

- All devices and peripherals are represented by files. To read or write a device you have to perform these operations on its file.
- Device file is used to represent a real physical device such as a printer, tape drive or terminal. Device or special files are used for device Input/Output(I/O) on Linux systems.
- They appear in a file system just like an ordinary file or a directory. On LINUX systems there are two flavours of special files for each device, character special files and block special files.
- When a character special file is used for device Input/Output(I/O), data is transferred one character at a time. This type of access is called raw device access.
- When a block special file is used for device Input/Output(I/O), data is transferred in large fixed-size blocks. This type of access is called block device access.
- Device filenames are generally found inside a single directory structure, /dev.

  In long-format output of ls -l, character special files are marked by the "c" symbol.

  In long-format output of ls -l, block special files are marked by the "b" symbol.

4. Pipes:

- A pipe is a form of redirection (transfer of standard output to some other destination) that is used in Linux operating systems to send the output of one command/program/process to another command/program/process for further processing.
- The 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.

  Syntax: `command1 | command2 | command3| ..... | commandN`

  Example: `$ls -l | more`

3.3 ORGANIZATION OF FILES

- A file system is the methods and data structures that an operating system uses to keep track of files on a disk or partition; that is, the way the files are organized on the disk.
- Linux chooses to have single hierarchical directory structure. Everything starts from the root directory, represented by /, and then expands into sub -directories instead of having so called drives.
- In windows environment, one may put one's files almost anywhere; on C drive, D drive etc. In Linux it sorts directories descending from the root directory / according to their importance to the boot process.
- Files in Linux System are organized into multi-level hierarchy structure known as a directory tree. At the very top of the file system is a directory called "root" which is represented by a "/". All other files are "descendants" of root.
- The directories are used to organize files. It contains not only files, but other directories also. This allows creating hierarchy. A hierarchical system of directory, based on a single root directory, in which all directories except the root directory are sub directory.
- The directory can contain all three types of files, ordinary file, directories and special files.

3.3.1 Standard Directories

- You will find a similar set of system directories on all Linux operating systems. These system directories are located directly below the root directory, and are essential to the startup and continuous operation of the system.
- **The following is a list of these key directories and their contents:**
  - **/bin:** The /bin directory contains programs needed for using and managing the system. Some of the frequently used commands in this directory are date (displays today's date), ls (lists the contents of a directory), and cp (makes a copy of a file). The name bin is used for this directory because executable programs in Linux are called binary files.
  - **/lib:** This is where libraries are kept. You will notice that many times when installing Linux software packages, additional libraries are also automatically downloaded, these are files need for your programs on Linux to work.

- **/dev:** This directory contains system device files. A device file is a special object in the file system that provides an interface to a particular device. Examples of devices having device files in /dev are disk drives, tape drives, or CDROM drives.
- **/etc:** System specific configuration files, and files essential for system startup are located in the /etc directory. In the past, administrative executable programs were also stored in this directory but have been moved to the /sbin directory.
- **/home:** The /home directory is where the home directories for all users of the system are stored. For example, if username is gajanan, the path to his home directory would be /home/gajanan. User gajanan would store his personal files and programs in this directory.
- **/mnt:** This directory is where temporary file systems are mounted. It may contain subdirectories like cdrom, floppy, and disk. Once a cdrom has been mounted on the system (made available for reading its contents), the path to the files located on the cdrom would be /mnt/cdrom.
- **/opt:** The /opt directory contains software files that are not installed when the operating system is installed. This directory usually contains products provided by third-party software vendors.
- **/sbin:** Programs for administering a system are located in the /sbin directory. Some examples are fdisk (used to partition a disk), fsck (used to check the integrity of a file system), and shutdown (used for stopping a system). An easy way to remember the contents of this directory is to equate sbin with "system binaries."
- **/tmp:** As the name implies, this directory is used for holding temporary files. This directory is commonly referred to as a scratch directory, and can be used by all system users. You should not store any important files in this directory because its contents may be deleted at any time. Your important files should be kept in your home directory.
- **/usr:** The /usr directory contains programs and files related to the users of a system. The data in /usr is typically read-only, and may be shared with other computer systems on a network. You will notice that the directory structure under /usr is somewhat similar to the directory structure under the root directory (/).
- **/media:** This is the place where external devices such as optical drive and USB drives can be mounted.
- **/var:** Files with varying content are stored in the /var directory. This includes system log files, mail system files, and print spooling system files.

3.3.2 Parent Child Relationship

- All files in linux are related to one another. The file system in Linux is a collection of all of these related files (ordinary, directory and device files) organized in a hierarchical structure.

- This file system also adopted by DOS and Windows. The implicit feature of every Linux file system is that there is top, which serves as a reference point for all types. This top is called as a Root which is actually a directory. It is conceptually different from the user-id root used by the system administrator to log in.
- As we have discussed files are stored in directories so they contain parent child relationship. In LINUX file system there is a top, called root which is actually a directory. The root directory has number of sub directories under it. They can have again more sub directories in it or files.
- Every file, apart from root, must have a parent, and it should be possible to trace the ultimate parentage of a file to root. So it is obvious that in parent child relationships the parent is always a directory and child can be a directory or a File.

```
$ pwd
/home/gajanan/MyDocs/MyExamples
```

- As you can see absolute path where in user gajanan there is MyDocs directory which contains MyExamples directory which can contain again directory or files in it.
- Thus home directory is parent for gajanan and grandparent from MyDocs. Directories are represented as parents where childs can be represented as files.

Have a look at following hierarchical representation of file system:

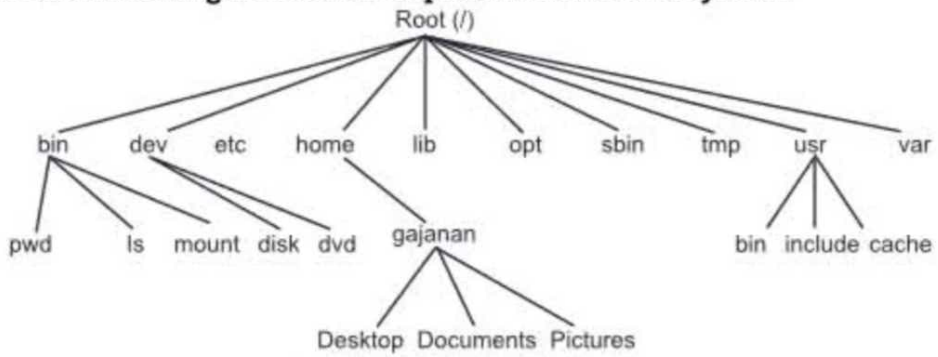


Fig. 3.2: Hierarchical representation of file system

- In this figure, the root contains sub directories home which contains all user directories and files. The directory in which the user is placed when user log in is called user home directory. In every login session, user start in his home directory and move up and down the directory tree.

3.3.3 The HOME Directory and The HOME Variable

Home Directories:

- When you log on to the system, LINUX automatically place you in home directory. It is created by system when a user account is opened.
- The /home directory is a place where by default all user home directories are created.
- If you log in with your user name then it will place you in directory with your username. These directories are a kind of personal place(Working space) for all the users other than root.

- There will be a separate folder for each user in /home directory. For example if you have a user called abc, then his default home directory is /home/abc. We can change this default folder when creating user in Linux. Our abc user can do whatever he wants in /home/abc folder where he have full rights on the files he created and owned in that folder.
- The shell variable HOME knows your home directory. HOME variable displays default path of users HOME directory.

```
$ echo $HOME
/home/gajanan
```

- The above one is absolute path name which is sequence of directories separated by slashes. An absolute path name shows a file's location with reference to the top. i.e. root.
- The slashes are the delimiters for the file and directory names, except the first slash it is synonym for the root. The directory gajanan is placed two levels below root.
- There are several easy ways for a user to return to its home directory regardless of its current directory (i.e., the directory in which it is currently working in).
- The simplest of these is to use the cd (i.e., change directory) command without any options or arguments (i.e., input files), i.e., by merely typing the following and then pressing the ENTER key:

```
$cd
```

- The tilde is used to represent users' home directories on Linux-like operating systems,

```
$cd ~
```

3.3.4 The PATH Variable, Manipulating the PATH

- The PATH is an environment variable. It is a colon delimited list of directories that your shell searches through when you enter a command.
- All executables are kept in different directories on the Linux and Linux like operating systems.
- To find out what your current path setting, type the following command at shell prompt. Open the Terminal and then enter:

```
echo "$PATH"
```

```
gajanan@gd-4184:~/MyDocs/dir2$ echo "$PATH"
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/usr/
lib/jvm/java-12-oracle/bin:/usr/lib/jvm/java-12-oracle/db/bin
gajanan@gd-4184:~/MyDocs/dir2$
```

- It contains set of paths where the system searches for an executable file. Each user on a system can have a different PATH variable.
- When an operating system is installed, one default PATH variable is created for the root (i.e., administrative) account and another default is created that will be applied to all ordinary user accounts as they are added to the system.
- The PATH variable for the root user contains more directories than for ordinary users because it includes directories, such as /sbin and /usr/sbin, that contain programs that are normally used only by that user.

- PATH variables can be changed relatively easily. They can be changed just for the current login session, or they can be changed permanently (i.e., so that the changes will persist through future sessions).
- It is a simple matter to add a directory to a user's PATH variable (and thereby add it to the user's default search path). It can be accomplished for the current session by using the following command, in which directory is the full path of the directory to be entered:

  ```
  PATH="directory:$PATH"
  ```

  For example, to add the directory /usr/sbin, the following would be used:

  ```
  PATH="/usr/sbin:$PATH"
  ```
- An alternative is to employ the export command, which is used to change aspects of the environment. Thus, the above absolute path could be added with the following two commands in sequence,

  ```
  PATH=$PATH:/usr/sbin
  export PATH
  ```

  or its single-line equivalent,

  ```
  export PATH=$PATH:/usr/sbin
  ```

3.3.5 Relative and Absolute Path Names

- A path is a unique location for file or a folder in a file system. A path to a file is combination of / and alpha numeric characters.

1. **Absolute path name:**

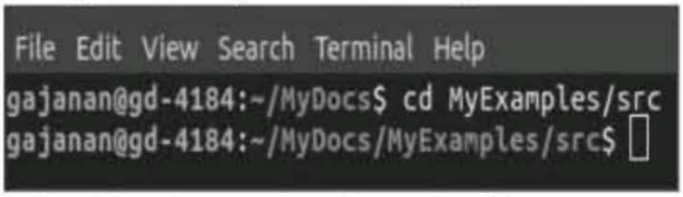
- Absolute path starts from the root directory root (/) and goes up to the actual object (file or directory). It contains the names of all directories that come in the middle of the directory root and the actual object.
- Lets take an example, suppose a user named gajanan creates a directory named test in his home directory. What will be the absolute path of this directory.
- To write the absolute path of this directory, we have to start writing the path from the directory root. The directory root directory is written as forward slash (/) after root directory, we have to write the name of the directory in which user's home directory is located. By default, Linux places user's home directory in the directory named home. Usually this directory is created just under the directory root.
- In our example, absolute path of the directory test will be root home/gajanan/root

Absolute path	Linux File System
/	root
/home	home directory
/home/gajanan	user's directory
/home /gajanan/test	file or directory

- In above picture the path home/gajanan/MyDocs/files/new is absolute path. As you can see we have been placed in to the directory called new.

2. Relative path:

- Relative path is defined as the path related to the present working directly(pwd). It starts at your current directory and never starts with a /.
- Mostly we are using relative path for moving from one directory to another. Any file name (path) begins with "/" is an absolute path. All others are relative path.
- All Linux programs can handle relative or absolute path or mixture of both. Suppose i am working in a directory called MyDocs and i have to move to directory src containing MyExamples directory in it. So relative path would be.



- Relative path starts from the current directory and goes up to the actual object. Relative path depends upon the current directory if we change the current directory, relative path also changes.
- Unlike the absolute path, all slashes in relative path represent the directory separator.
- **Examples for the absolute and relative paths for some operations:**
  1. Present location is abcxyz, and i want to deleted file /abc/xyz/dir1/sample.txt file-size
     - Using relative path
       `rm dir1/sample.txt`
     - Using absolute path
       `rm abc/xyz/dir1/sample.txt`
  2. Suppose my present location is /Documents/sample to /Documents
     - Using relative path
       `cd ..`
     - Using absolute path
       `cd /Documents`
  3. My present location is /var/ftp/ and I want to change the location to /var/log.
     - Using relative path:
       `cd ../log`
     - Using Absolute path:
       `cd /var/log`
  4. My present location is /etc/lvm and I want to change my location to /opt/eclipse.
     - Using relative path:
       `cd ../../opt/eclipse`
     - Using Absolute path:
       `cd /opt/eclipse`

3.4 DIRECTORY COMMANDS

1. pwd Command:

- The pwd command is a command line utility for printing the current working directory.
- It will print the full system path of the current working directory to standard output.
- By default the pwd command ignores symlinks, although the full physical path of a current directory can be shown with an option.
- The pwd command is normally a shell built-in meaning it is part of the code that runs the shell rather than an external executable.

2. cd command:

- It is used to change the directory. It is very important command to move from one directory to another also from current directory to user or root.

Syntax: `cd directoryname`

Example:

(a) We can change directory from current directory to specified directory.

- Using relative path.

  `$cd usr/local`

- Using absolute path.

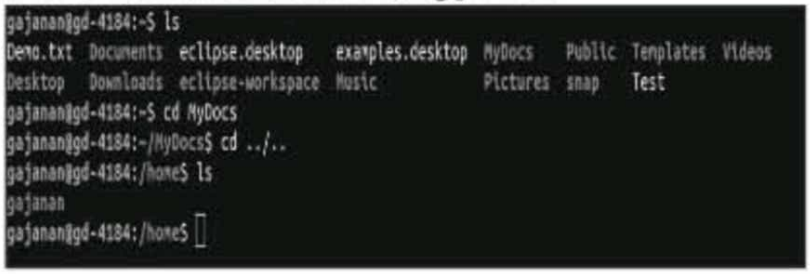
  `$ cd /usr/local/lib`

(b) To switch to parent directory we can use cd.. command.

As you can see we shifted to parent directory Mydocs from linux directory.

(c) Also we can move two directories up from current directory using following command

`$cd .. / ..` Have a look at following picture.



(d) To move to use's home directory use following command.

```
$ cd ~
```

(e) to go to the root directory type the following command

```
$cd /
```

3. **mkdir command:**

- Directories can be created with mkdir command. We can create multiple directories using mkdir command.

  Syntax: `$mkdir [options] directoryname`

  Example: `$mkdir dir1`

  also, `$mkdir dir2 dir3 dir4`

  it will create three directories at same time.

  Options:

  1. **--v:version:** it displays the version of mkdir command
  2. **–help:** it displays the help related information.
  3. **-v or --verbose**: It displays a message for every directory created.

  Example: `$ mkdir -v dir4`

  mkdir: created directory 'dir4'

4. **-p:**

- A flag which enables the command to create parent directories as necessary. If the directories exist, no error is specified.

  Syntax: `$mkdir -p [directories]`

  Suppose you execute the following command:

  ```
  mkdir -p first/second/third
  ```

  If the first and second directories do not exist, due to the -p option, mkdir will create these directories for us. If we do not specify the -p option, and request the creation of directories.

5. **rmdir command:**

- rmdir command is used remove empty directories from the filesystem in Linux.
- The rmdir command removes each and every directory specified in the command line only if these directories are empty. So if the specified directory has some directories or files in it then this cannot be removed by rmdir command.

  Syntax: `rmdir [-p] [-v | -verbose] [-ignore-fail-on-non-empty] directories …`

  Example: `rmdir first`

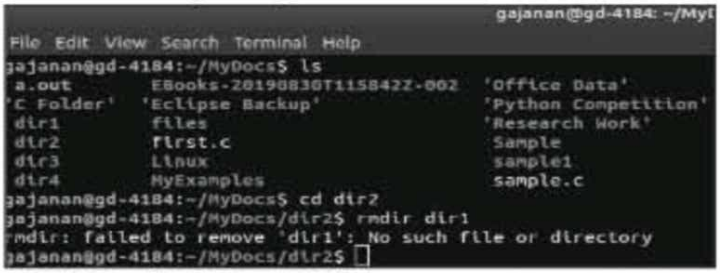
  this will remove directory first. Similarly we can delete more than one directory.

  Example: `rmdir second third fourth`

  This will delete three directories.

- Also you should remember that you can't remove a subdirectory unless you are placed in a directory which is hierarchically above the on you have chosen to remove.

- Let's have a look at following example:

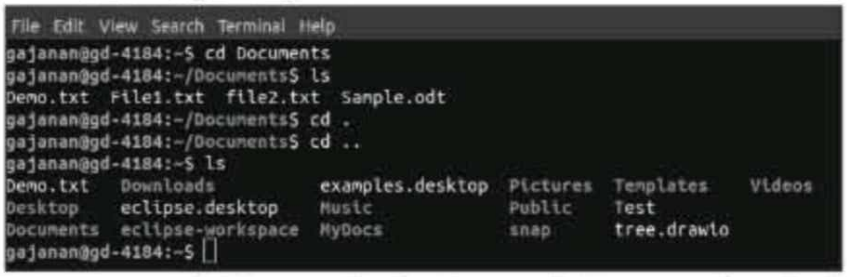


- We are placed into dir2 and we are trying to remove dir1 which is parent directory so it will not remove dir1. You have to move to parent directory then it will remove it.
- **Options:**
  1. **rmdir -p:** In this option each of the directory argument is treated as a pathname of which all components will be removed, if they are already empty, starting from the last component.
  2. **rmdir -v, –verbose:** This option displays verbose information for every directory being processed.
  3. **rmdir –ignore-fail-on-non-empty:** This option does not report a failure which occurs solely because a directory is non-empty. Normally, when rmdir is being instructed to remove a non-empty directory, it simply reports an error. This option consists of all those error messages.
  4. **rmdir –version:** This option is used to display the version information.

3.5 THE DOT(.) AND DOUBLE DOTS(..) NOTATIONS TO REPRESENT PRESENT AND PARENT DIRECTORIES AND THEIR USAGE IN RELATIVE PATH NAMES

- In Linux every directory contains two dots; single dot and double dots. When a directory is created, both the single dot and the double dots are also automatically created in it. By default these dots are hidden and do not show in the output of the command ls. To view these dots, we have to use the option a with the command ls.
- The single dot refers to the directory itself and the double dots refers to its parent directory or the directory that contains it. Shell allows us to access the current directory and the parent directory by using the single dot and the double dots respectively.
- Relative path also uses these dots to represent the current directory and the parent directory respectively. With the use of these dots, we can build the relative path of any file or directory from the current directory.
- Single dot (.) is used to represent the present directory,
- Double dot (..) represent parent directory.

- Have a look at following example:



- As you can see we are in Documents directory and when we enter cd .. it moves one level up in the directory structure. This method is compact and more useful when ascending the hierarchy. The command cd .. translate like " change your directory to the parent of the current directory".
- We can use any number of combinations of .. separated by /. However, when a / is used with .. it acquires a different meaning; instead of moving down a level, it moves one level up, for moving to home directory we can provide command cd /home. Also you can use relative pathname:

```
$pwd
/home/gajanan/newdir
$cd .. / ..
$pwd
$home
```

- As you can see with .. / .. it will allow you to move two levels up in the directory structure. As we were in gajanan/newdir when we provided .. / .. command it has redirected us to home directory.
- Now again, suppose you are currently working in Desktop directory.

```
$ pwd
/home/gajanan/Desktop
```

- You want to change your directory to home/gajanan/Desktop/Sample. Let's see both the absolute and relative path concepts to do this:

Using Relative path:

```
$ cd Sample
```

Using Absolute Path

```
$ cd /home/gajanan/Desktop/Sample
```

in Relative path we are just changing the directory with cd command and moving to the next directory. In Absolute path we are giving the complete path from home directory.

3.6 FILE RELATED COMMANDS

1. cat Command:

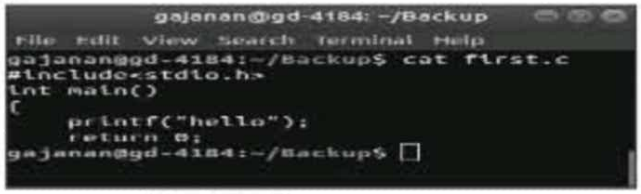
- The cat (short for "**concatenate**") command is one of the most frequently used command in Linux/Linux like operating systems.
- cat command allows us to create single or multiple files, view contain of file, concatenate files and redirect output in terminal or files.

General Syntax:

```
$cat [OPTION] [FILE]...
```

to display contents of file use following command,

```
$cat first.c
```



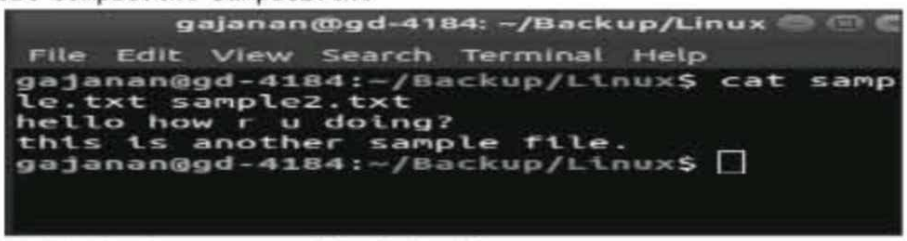
it will display the contents of the file first.c

Create a file with cat command:

```
$cat > sample.txt
```

- This command will create a file sample.txt immediately after it you can enter the text and press ctrl+z to save the contents.
- Displaying multiple files with cat

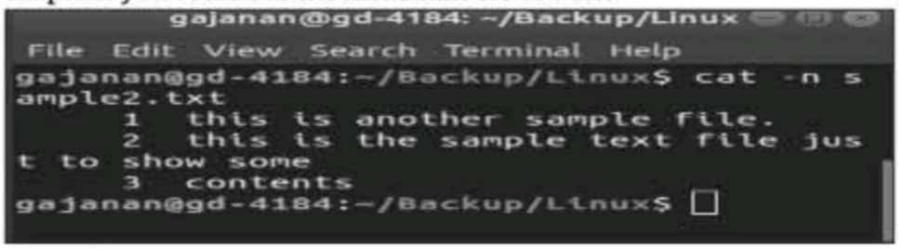
```
$cat sample.txt sample2.txt
```



- This will display the contents of both the files.

Displaying line numbers in a file:

- With **-n** option you could see the line numbers of a file



2. mv command:

- **mv** stands for **move**. mv is used to move one or more files or directories from one place to another in file system like Linux. It has two distinct functions:
  (i) It rename a file or folder.
  (ii) It moves group of files to different directory.
- No additional space is consumed on a disk during renaming. This command normally works silently means no prompt for confirmation.

Syntax: `Mv [option] source destination`

Example: `$mv file1.txt file2.txt`

- If the destination file doesn't exist, it will be created. In the above command mv simply replaces the source filename in the directory with the destination filename(new name). If the destination file exist, then it will be overwrite and the source file will be deleted. By default, mv doesn't prompt for overwriting the existing file.
- **Options:**

(i) **-i (Interactive):** the -i option makes the command ask the user for confirmation before moving a file that would overwrite an existing file, you have to press **y** for confirm moving, any other key leaves the file as it is. This option doesn't work if the file doesn't exist, it simply rename it or move it to new location.

3. rm Command:

- **rm** stands for **remove.** rm command is used to remove objects such as files, directories, symbolic links and so on from the file system.
- To be more precise, rm removes references to objects from the filesystem, where those objects might have had multiple references (for example, a file with two different names). By default, it does not remove directories.

  Syntax: `rm [option] FILE`

  Example: `rm sample1.txt`

  This will delete sample1.txt file
- **Deleting multiple files:** we can delete multiple files using rm command

  See the following code:

- **Options:**

  (i) -i (Interactive Deletion):
  - Like in cp -i option makes the command ask the user for confirmation before removing each file, you have to press **y** for confirm deletion, any other key leaves the file un-deleted.

    `rm -i samplefile.txt`

    rm: remove regular file 'samplefile.txt'?

    If you press y then this file will get delete.

  (ii) -f (Force Deletion):
  - **rm** prompts for confirmation removal if a file is **write protected**. The **-f** option overrides this minor protection and removes the file forcefully.

    `rm file2.txt`

4. cp command:

- **cp** stands for **copy**. This command is used to copy files or group of files or directory.
- cp is the command entered in a Linux shell to copy a file from one place to another, possibly on a different filesystem. The original file remains unchanged, and the new file may have the same or a different name.

Syntax:

```
cp [OPTION] Source Destination
cp [OPTION] Source Directory
cp [OPTION] Source-1 Source-2 Source-3 Source-n Directory
cp [OPTION] Source dest
cp [OPTION] Source Directory
```

Where,

- In the first and second syntax you copy SOURCE file to DEST file or DIRECTORY.
- In the third syntax you copy multiple SOURCE(s) (files) to DIRECTORY.
- To make a copy of a file called file1.txt in the current directory as file2.txt then enter:

```
$ cp file1.txt file2.txt
```

- You can copy multiple files simultaneously into another directory.

```
$ cp MULTIPLI.c file1.txt file2.txt new
```

Note: For this case last argument must be a directory name. For the above command to work, new must exist because cp command won't create it.

Two directory names:

- If the command contains two directory names, cp copies all files of the source directory to the destination directory, creating any files or directories needed. This mode of operation requires an additional option, typically R, to indicate the recursive copying of directories.

```
cp -R Src_directory Dest_directory
```

- In the above command, cp behaviour depends upon whether Dest_directoryis exist or not. If the Dest_directory doesn't exist, cp creates it and copies content of Src_directory recursively as it is. But if Dest_directory exists then copy of Src_directory becomes sub-directory under Dest_directory.

Cp command Options:

- There are many options of cp command, we will discuss some options below:

1. -i(interactive):

- i stand for Interactive copying. With this option system first warns the user before overwriting the destination file. cp prompts for a response,

```
$ cp -i file1.txt file2.txt
cp: overwrite 'file2.txt'? y
```

2. **-b(backup):**

- With this option cp command creates the backup of the destination file in the same folder with the different name and in different format.

```
$ ls
file1.txt file2.txt
$ cp -b file1.txt file2.txt
$ ls
file1.txt file2.txt file2.txt~
```

3. **-f(force):**

- If the system is unable to open destination file for writing operation because the user doesn't have writing permission for this file then by using -f option with cp command, destination file is deleted first and then copying of content is done from source to destination file.

```
$ cp sample1.txt sample2.txt
cp: cannot create regular file 'sample2.txt': Permission denied
```

- With -f option, command executed successfully

```
$ cp -f sample1.txt sample2.txt
```

4. **-r or -R: Copying:**

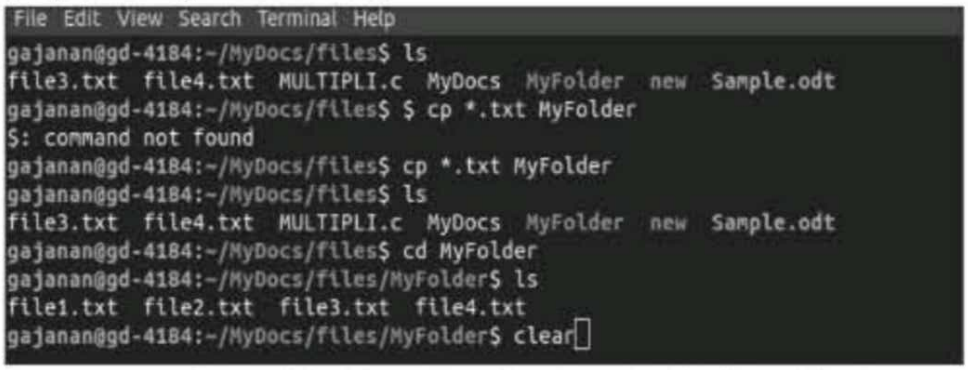
- directory structure. With this option cp command shows its recursive behaviour by copying the entire directory structure recursively. Suppose we want to copy MyDoc directory containing many files, directories into NewDir directory(not exist).

```
$ cp -r MyDoc NewDir
```

5. **Copying using * wildcard:**

- The star wildcard represents anything i.e. all files and directories. Suppose we have many text document in a directory and wants to copy it another directory, it takes lots of time if we copy files 1 by 1 or command becomes too long if specify all these file names as the argument, but by using * wildcard it becomes simple.

```
$ cp *.txt MyFolder
```



As you can see all *.txt files from files directory to MyFolder directory using cp * command

6. **wc Command**:

- This command stands for Word Count. This command is mainly used for counting purposes.
- It is used to find number of lines, word count, byte and character count.By default it displays four columnar outputs.
- First column shows number of lines present in a file specified, second column shows number of words present in the file,
- Third column shows number of characters present in file and fourth column itself is the file name which is given as argument.

  Example: $ cat textfile.txt

  this is sample text file where char, words will count.

  this is second line of text.

  ```
  $ wc textfile.txt
  2 16 82 textfile.txt
  ```

  wc counts 2 lines, 16 words and 82 characters. The file name also shown in the fourth column.
- The wc command offers three options to make a specific count.
- The -l option counts only the number of lines, while -w counts number of words and -c counts number of characters.

  ```
  $ wc -l textfile.txt
  2 textfile.txt
  $ wc -w textfile.txt
  16 textfile.txt
  $ wc -c textfile.txt
  82 textfile.txt
  ```

  wc can be used with multiple files, it produces a line for each file as well as count.

  ```
  $ wc file1.txt file2.txt textfile.txt
  2 12 75 file1.txt
  3 17 112 file2.txt
  2 16 82 textfile.txt
  7 45 269 total
  ```

7. **od command:**

- The **od** command writes an unambiguous representation, using octal bytes by default, of FILE to standard output. If more than one FILE is specified, od concatenates them in the listed order to form the input. With no FILE, or when FILE is a dash ("-"), od reads from standard input.

  Syntax: `od [OPTION]...[FILE]...`

Examples with options of od command:

(i) -b option:

```
$ od -b textfile.txt
0000000 164 150 151 163 040 151 163 040 163 141 155 160 154 145 164 145
0000020 170 164 040 146 151 154 145 040 011 167 150 145 162 145 040 143
0000040 150 141 162 054 167 157 162 144 163 040 167 151 154 154 040 143
0000060 157 165 156 164 056 012 164 150 151 163 040 151 163 040 163 145
0000100 143 157 156 144 040 012 154 151 156 145 040 157 146 040 164 145
0000120 170 164 056 040 012
0000125
```

Display the contents of file.txt in octal format (one byte per integer).

(ii) -x option:

```
$ od -x textfile.txt
0000000 6874 7369 6920 2073 6173 706d 656c 6574
0000020 7478 6620 6c69 2065 6877 7265 2065 6863
0000040 7261 772c 726f 7364 7720 6c69 206c 6f63
0000060 6e75 2e74 740a 6968 2073 7369 7320 6365
0000100 6e6f 2064 696c 656e 6f20 2066 6574 7478
0000120 202e 000a
0000123
```

it will display the contents of file in hexadecimal format.

(iii) -c Option:

It displays the contents of input in character format.

```
$ od -c textfile.txt
0000000 t h i s i s s a m p l e t e
0000020 x t f i l e w h e r e c h
0000040 a r , w o r d s w i l l c o
0000060 u n t . \n t h i s i s s e c
0000100 o n d l i n e o f t e x t
0000120 . \n
0000123
```

Following option accepts the input from command line.

```
$ od -c -
good evening to all
0000000 g o o d e v e n i n g t o a l l.
```

(iv) -bc option:

```
$ od -bc textfile.txt
0000000 164 150 151 163 040 151 163 040 163 141 155 160 154 145 164 145
t h i s i s s a m p l e t e
0000020 170 164 040 146 151 154 145 040 167 150 145 162 145 040 143 150
x t f i l e w h e r e c h
0000040 141 162 054 167 157 162 144 163 040 167 151 154 154 040 143 157
a r , w o r d s w i l l c o
0000060 165 156 164 056 012 164 150 151 163 040 151 163 040 163 145 143
u n t . \n t h i s i s s e c
0000100 157 156 144 040 154 151 156 145 040 157 146 040 164 145 170 164
   o n d l i n e o f t e x t
0000120 056 040 012
. \n
0000123
```

Each line now replaced with two. The octal representation is shown line. The printable characters and escape sequences are shown in second line.

3.6.1 File Attributes and Permission and knowing them

- As we are creating number of files in Linux we should know what are different file attributes and what kind of file permissions are given to the files. As you may have problems with handling file or directory. As your file can be modified or deleted by others. We should know about file permissions.
- ls – l command is used to long listings of files we will use this to list files and see what are different file attributes like its permissions, size and ownerships details. ls look up the file's inode to fetch its attributes.

```
File  Edit  View  Search  Terminal  Help
gajanan@gd-4184:~/Documents$ ls -l
total 24
-rw-r--r-- 1 gajanan gajanan    4 Sep 11 08:55 Demo.txt
---------- 1 gajanan gajanan    4 Sep  3 09:48 File1.txt
-rw-r--r-- 1 gajanan gajanan   15 Sep 11 08:54 file2.txt
drwxr-xr-x 2 gajanan gajanan 4096 Sep 20 10:33 newdir1
-rw-r--r-- 1 gajanan gajanan 7715 Sep  3 09:47 Sample.odt
```

- This list is preceded by the words total 24, which indicates total 24 blocks are occupied by these files on disks, each block consisting of 512 bytes.

3.6.1.1 File Attributes

- The first columns indicate the type and permissions associated with the file. The first character in this is mostly a – which indicates the file is ordinary file. If it is a directory file so d will appear at beginning. So in the image newdir1 is directory and Demo.txt,File1.txt,File2.txt is Text files.
- As you can see a series of characters in the first column having values rwx. In Linux system, a file can have three types of permissions, read, write and execute.

- **Links:** The second column indicates the number of links associated with the file. This is actually number of filenames maintained by the system of that file. A link count greater than one indicates that the file has more than one name. It doesn't mean that it has two copies of files.
- **Ownership:** When you create a file, you automatically become its owner. The third column shows gajanan as the owner of all of these files. The owner has full authority to tamper with a file's contents and permissions- a privilege not available with others except the root user. Similarly, you can create, modify or remove files in a directory if you are the owner of directory.
- Group ownership: when opening a user account, the system administrator also assigns the user to some group. The fourth column represents the group ownership of file.
- Fifth column shows the size of file in bytes. It only contains number of count of characters.
- Sixth column shows the last modification time of the file mentioning date and time of the file.
- Seventh column indicates actual name of the file with the given extension.

3.6.1.2 File Permissions

- Linux has very simple system of assigning permissions to file.
- All three owners (users,owners,group,others) in the Linux system have three types of permissions defined. Nine characters denote the three types of permissions.
  - **Read(r):** The read permission allows you to open and read the content of a file. But you can't do any editing or modification in the file.
  - **Write (w):** The write permission allows you to edit, remove or rename a file. For instance, if a file is present in a directory, and write permission is set on the file but not on the directory, then you can edit the content of the file but can't remove, or rename it.
  - **Execute (x):**In Unix type system, you can't run or execute a program unless execute permission is set. But in Windows, there is no such permission available.
- Permissions are listed below:

Permission	On a file	On a directory
r (read)	Read file contents (cat)	Read directory contents (ls)
W (write)	Change file contents (vi)	Creates files in directory (touch)
x (execute)	Execute a file	Enter the directory

- When you are the User owner, then the user owner permission applies to you. Other permissions are not relevant to you. When you are Group the group permissions are applied to you. And when you are others, other permissions are applied to you.
- Permissions in the command line are displayed as follows:

```
rwx r-x r - -
```

- Lets try to understand the characters here each group is a category, the first group (rwx) is the owner and it contains three permissions read, write, extcute. Now the user gajanan has read, write and execute permissions on file.
- The second group(r-x) has a hyphen in the middle slot, which indicates absence of write permission by the group owner of file. This group owner is gajanan, so all users belonging to the group gajanan have read and execute permissions on file.
- The Third group(r--) it has the read permission to three others and two hyphens absent indicates that the others can't write and executs the files. As you can set different permissions for the three categories of users as owner, group and others. It's very important to understand these permissions because it can violate the security permissions of files.
- Note in case of absence of permission it is denoted by – (hyphen) absence of permission for specific group.
- When in the command line, the permissions are edited by using the command chmod. You can assign the permissions explicitly or by using a binary reference.

3.6.2 The ls Command with Options

- **ls** is a Linux shell command that lists directory contents of files and directories. Some practical examples of ls command are shown below.

1. **-x option:**

- When you have multiple files it's better to display with multiple columns. Modern versions of ls do by default. -x option is used to produce a multicolumnar output.

  Example: `$ ls -x`

  ```
  Desktop Documents Downloads eclipse. Desktop eclipse-workspace
  ```

  Examples: `Desktop Music MyDocs Pictures Public snap Templates Test Videos`

2. **-a option:**

- It will give you the whole list of a directory including the hidden files also. In Linux, hidden files start with a dot (.) and can't be seen in the regular directory.

  Example: `$ ls -a`

  ```
  . Downloads Music .swt
  .. .eclipse MyDocs Templates
  .bash_history eclipse.desktop Pictures Test
  .bash_logout eclipse-workspace .pki .thunderbird
  .bashrc examples. Desktop .profile .tooling
  .cache .gnupg Public Videos
  .config .ICEauthority snap .vscode
  Desktop .local .ssh .vscode-cpptools
  Documents .mozilla .sudo_as_admin_successful
  ```

3. -l option:

- The ls command will only display the files. But if you want your files to be displayed in a long list format, then you can use ls -l command.

  Example:

```
$ ls -l
total 68
drwxr-xr-x 2 gajanan gajanan 4096 Aug 29 19:12 Desktop
drwxr-xr-x 2 gajanan gajanan 4096 Sep 3 12:18 Documents
drwxr-xr-x 2 gajanan gajanan 4096 Sep 4 09:28 Downloads
-rw-r--r-- 1 gajanan gajanan 238 Aug 30 11:05 eclipse.desktop
drwxrwxr-x 5 gajanan gajanan 4096 Aug 30 11:50 eclipse-workspace
-rw-r--r-- 1 gajanan gajanan 8980 Aug 29 18:58 examples.desktop
drwxr-xr-x 2 gajanan gajanan 4096 Aug 29 19:12 Music
drwxr-xr-x 15 gajanan gajanan 4096 Sep 3 20:02 MyDocs
drwxr-xr-x 2 gajanan gajanan 4096 Aug 29 19:12 Pictures
drwxr-xr-x 2 gajanan gajanan 4096 Aug 29 19:12 Public
drwxr-xr-x 4 gajanan gajanan 4096 Aug 30 19:24 snap
drwxr-xr-x 2 gajanan gajanan 4096 Aug 29 19:12 Templates
-rw-r--r-- 1 gajanan gajanan 8192 Sep 1 11:28 Test
drwxr-xr-x 2 gajanan gajanan 4096 Aug 29 19:12 Videos
```

  Here, as you can see the list in long list format.

Columns above indicate specific things:

- Column 1 indicates information regarding file permission.
- Column 2 indicates the number of links to the file.
- Column 3 & 4 indicates the owner and group information.
- Column 5 indicates size of the file in bytes.
- Column 6 shows the date and time on which the file was recently modified.
- Column 7 shows the file or directory name.

4. ls -l --block-size=[SIZE]:

- If you want to display the file size of your list in a particular format or size, then you can use this command. Just put the size in place of [SIZE] as per your requirement.

  Syntax: `ls -l -- block-size=[SIZE]`

  Example: `ls -l -- block-size =M`

  Here, all file size has listed in Megabyte.

You can replace [SIZE] with the following measures:

- K = Kilobyte
- M = Megabyte
- G = Gigabyte
- T = Terabyte

Example: `$ ls -l --block-size=M`

```
total 1M
drwxr-xr-x 2 gajanan gajanan 1M Aug 29 19:12 Desktop
drwxr-xr-x 2 gajanan gajanan 1M Sep 3 12:18 Documents
drwxr-xr-x 2 gajanan gajanan 1M Sep 4 09:28 Downloads
-rw-r--r-- 1 gajanan gajanan 1M Aug 30 11:05 eclipse.desktop
drwxrwxr-x 5 gajanan gajanan 1M Aug 30 11:50 eclipse-workspace
-rw-r--r-- 1 gajanan gajanan 1M Aug 29 18:58 examples.desktop
drwxr-xr-x 2 gajanan gajanan 1M Aug 29 19:12 Music
drwxr-xr-x 15 gajanan gajanan 1M Sep 3 20:02 MyDocs
drwxr-xr-x 2 gajanan gajanan 1M Aug 29 19:12 Pictures
drwxr-xr-x 2 gajanan gajanan 1M Aug 29 19:12 Public
drwxr-xr-x 4 gajanan gajanan 1M Aug 30 19:24 snap
drwxr-xr-x 2 gajanan gajanan 1M Aug 29 19:12 Templates
-rw-r--r-- 1 gajanan gajanan 1M Sep 1 11:28 Test
drwxr-xr-x 2 gajanan gajanan 1M Aug 29 19:12 Videos
```

5. **ls -d */ Option:**

- If you only want to display the sub-directories excluding all other files, you can use this command.

  Example: `$ls -d */`

  ```
  Desktop/ Downloads/ Music/ Pictures/ snap/ Videos/
  Documents/ eclipse-workspace/ MyDocs/ Public/ Templates/
  ```

- The above result only shows sub-directories excluding all the other files.

6. **ls -g Option:**

- If you don't want to display the owner information in your list, then you can exclude this column with the help of this command.

  Example: `$ls -g`

  ```
  total 68
  drwxr-xr-x 2 gajanan 4096 Aug 29 19:12 Desktop
  drwxr-xr-x 2 gajanan 4096 Sep 3 12:18 Documents
  drwxr-xr-x 2 gajanan 4096 Sep 4 09:28 Downloads
  -rw-r--r-- 1 gajanan 238 Aug 30 11:05 eclipse.desktop
  drwxrwxr-x 5 gajanan 4096 Aug 30 11:50 eclipse-workspace
  -rw-r--r-- 1 gajanan 8980 Aug 29 18:58 examples.desktop
  drwxr-xr-x 2 gajanan 4096 Aug 29 19:12 Music
  drwxr-xr-x 15 gajanan 4096 Sep 3 20:02 MyDocs
  drwxr-xr-x 2 gajanan 4096 Aug 29 19:12 Pictures
  drwxr-xr-x 2 gajanan 4096 Aug 29 19:12 Public
  drwxr-xr-x 4 gajanan 4096 Aug 30 19:24 snap
  drwxr-xr-x 2 gajanan 4096 Aug 29 19:12 Templates
  -rw-r--r-- 1 gajanan 8192 Sep 1 11:28 Test
  drwxr-xr-x 2 gajanan 4096 Aug 29 19:12 Videos
  ```

7. **ls -lG Option:**

- If you don't want to display the group information in your list then you can exclude this column with the help of this command.

  Example: `$ ls -lG`

8. **-r option:**

- It will display all list elements in the reverse order.

  Example: `$ ls -r`

```
Videos snap MyDocs eclipse-workspace Documents
Test Public Music eclipse.desktop Desktop
Templates Pictures examples.desktop Downloads
```

9. **ls ~ Option:**

- Linux ls ~ command shows the contents of the home directory. Let us see the example of ls ~ command.

  Example: `$ ls ~`

```
Desktop eclipse.desktop Music Public Test
Documents eclipse-workspace MyDocs snap Videos
Downloads examples.desktop Pictures Templates
```

10. **ls ../ Option:**

- This command displays the contents of parent directory.

  Example: `gajanan@gd-4184:~/Documents$ ls ../`

```
Desktop eclipse.desktop Music Public Test
Documents eclipse-workspace MyDocs snap Videos
Downloads examples.desktop Pictures Templates
```

- Currently we are in Documents directory and showing the contents of its parent directory.

11. **ls –version option:** It checks the version of ls command

  Example: `$ ls ../`

```
Desktop eclipse.desktop Music Public Test
Documents eclipse-workspace MyDocs snap Videos
Downloads examples.desktop Pictures Templates
gajanan@gd-4184:~/Documents$ ls --version
ls (GNU coreutils) 8.28
Copyright (C) 2017 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later
<http://gnu.org/licenses/gpl.html>.
```

- This is free software: you are free to change and redistribute it.
- There is NO WARRANTY, to the extent permitted by law.
- Written by Richard M. Stallman and David MacKenzie.

12. ls -lh Option:

- This command will show you the file sizes in human readable format. Size of the file is very difficult to read when displayed in terms of byte. The (ls -lh)command will give you the data in terms of Mb, Gb, Tb, etc.

  Example: `$ ls -lh`

```
total 68K
drwxr-xr-x 2 gajanan gajanan 4.0K Aug 29 19:12 Desktop
drwxr-xr-x 2 gajanan gajanan 4.0K Sep 3 12:18 Documents
drwxr-xr-x 2 gajanan gajanan 4.0K Sep 4 09:28 Downloads
-rw-r--r-- 1 gajanan gajanan 238 Aug 30 11:05 eclipse.desktop
drwxrwxr-x 5 gajanan gajanan 4.0K Aug 30 11:50 eclipse-workspace
-rw-r--r-- 1 gajanan gajanan 8.8K Aug 29 18:58 examples.desktop
drwxr-xr-x 2 gajanan gajanan 4.0K Aug 29 19:12 Music
drwxr-xr-x 15 gajanan gajanan 4.0K Sep 3 20:02 MyDocs
drwxr-xr-x 2 gajanan gajanan 4.0K Aug 29 19:12 Pictures
drwxr-xr-x 2 gajanan gajanan 4.0K Aug 29 19:12 Public
drwxr-xr-x 4 gajanan gajanan 4.0K Aug 30 19:24 snap
drwxr-xr-x 2 gajanan gajanan 4.0K Aug 29 19:12 Templates
-rw-r--r-- 1 gajanan gajanan 8.0K Sep 1 11:28 Test
drwxr-xr-x 2 gajanan gajanan 4.0K Aug 29 19:12 Videos
```

13. ls -Fx Option:

- To identify directories and executable files, the -F option should be used. Combining both options with -x produces a multicolumnar output.

  Example: `$ ls -Fx`

```
Desktop/ Documents/ Downloads/ eclipse.desktop eclipse-workspace/
examples.desktop Music/ MyDocs/ Pictures/ Public/
snap/ Templates/ Test Videos/
here / indicates the directories and * indicates that the file contain
executable code.
```

14. ls -t option:

- It sorts filenames by last modification time.

  Example: `$ ls -t`

```
Downloads Test eclipse.desktop Pictures Videos
MyDocs snap Desktop Public examples.desktop
Documents eclipse-workspace Music Templates
```

15. ls -lt Option:

- This option sorts listing by last modification time.

Example: `$ ls -lt`

```
total 68
drwxr-xr-x 2 gajanan gajanan 4096 Sep 4 09:28 Downloads
drwxr-xr-x 15 gajanan gajanan 4096 Sep 3 20:02 MyDocs
drwxr-xr-x 2 gajanan gajanan 4096 Sep 3 12:18 Documents
-rw-r--r-- 1 gajanan gajanan 8192 Sep 1 11:28 Test
drwxr-xr-x 4 gajanan gajanan 4096 Aug 30 19:24 snap
drwxrwxr-x 5 gajanan gajanan 4096 Aug 30 11:50 eclipse-workspace
-rw-r--r-- 1 gajanan gajanan 238 Aug 30 11:05 eclipse.desktop
drwxr-xr-x 2 gajanan gajanan 4096 Aug 29 19:12 Desktop
drwxr-xr-x 2 gajanan gajanan 4096 Aug 29 19:12 Music
drwxr-xr-x 2 gajanan gajanan 4096 Aug 29 19:12 Pictures
drwxr-xr-x 2 gajanan gajanan 4096 Aug 29 19:12 Public
drwxr-xr-x 2 gajanan gajanan 4096 Aug 29 19:12 Templates
drwxr-xr-x 2 gajanan gajanan 4096 Aug 29 19:12 Videos
-rw-r--r-- 1 gajanan gajanan 8980 Aug 29 18:58 examples.desktop
```

3.6.3 Changing File Permissions

- Linux is a multi user operating system, so it has security to prevent people from accessing each others confidential files.
- Lets take an example with ls -l command to show long list of files.

```
$ ls -l
total 68
drwxr-xr-x 2 gajanan gajanan 4096 Aug 29 19:12 Desktop
drwxr-xr-x 2 gajanan gajanan 4096 Sep 3 12:18 Documents
drwxr-xr-x 2 gajanan gajanan 4096 Sep 4 09:28 Downloads
-rw-r--r-- 1 gajanan gajanan 238 Aug 30 11:05 eclipse.desktop
drwxrwxr-x 5 gajanan gajanan 4096 Aug 30 11:50 eclipse-workspace
-rw-r--r-- 1 gajanan gajanan 8980 Aug 29 18:58 examples.desktop
drwxr-xr-x 2 gajanan gajanan 4096 Aug 29 19:12 Music
drwxr-xr-x 15 gajanan gajanan 4096 Sep 3 20:02 MyDocs
```

- We will describe them as follows:
  1. The first character will almost always be either a '-', which means it's a file, or a 'd', which means it's a directory.
  2. The next nine characters (rw-r–r–) show the security; we'll talk about them later.

3. The next column shows the owner of the file. In this case it is me, my userID is "gajanan"
4. The next column shows the group owner of the file. In my case I want to give the "gajanan" group of people special access to these files.
5. The next column shows the size of the file in bytes.
6. The next column shows the date and time the file was last modified.
7. And the final column gives the filename.

- All the three owners (user owner, group, others) in the Linux system have three types of permissions defined. Nine characters denote the three types of permissions.
  1. **Read (r):** The read permission allows you to open and read the content of a file. But you can't do any editing or modification in the file.
  2. **Write (w):** The write permission allows you to edit, remove or rename a file. For instance, if a file is present in a directory, and write permission is set on the file but not on the directory, then you can edit the content of the file but can't remove, or rename it.
  3. **Execute (x):** In Linux type system, you can't run or execute a program unless execute permission is set. But in Windows, there is no such permission available.

3.6.4 The Relative and Absolute Permissions Changing Methods

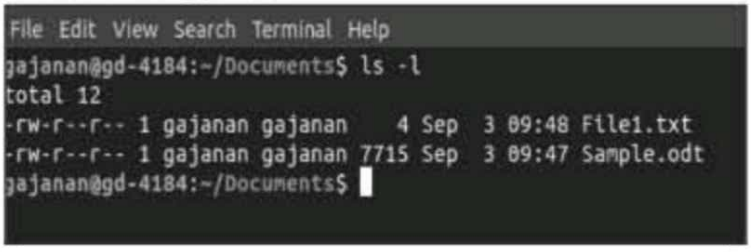
Relative Permissions:

- When changing permissions in relative manner, chmod changes only permissions specified in command line and leaves the other permissions unchanged.

  Syntax: `chmod category operation permission filename.`
- Where category belongs to user, group or others, the operation is assign or removes permission and the type of permission is read, write or execute.
- chmod takes its arguments as an expression, which combines the category, the operation and the type of permission by making use of suitable abbreviations. The abbreviations used for the three elements of the expressions are:

Category	Operation	Permission
u -user	+ assigns permission	r read permission
g – group	- removes permission	W writes permission
o - others	= it assigns permission x	X executes permission

- Let us look at the following picture.



- Let us try to understand the nine characters which are displayed through ls -l command.

```
rwx    rwx    rwx
user   group  other
```

- The file types can be shown while listing as follows:

-	Ordinary file
d	Directory file
d	Character file
b	Block special file.
S	semaphore
l	Symbolic link

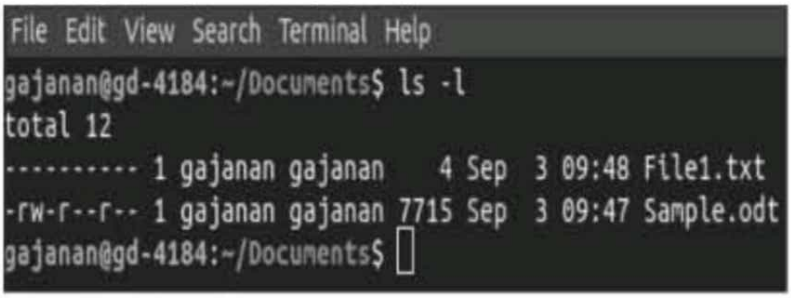
- Where r stands for read permission.w stands for write permission. x stands for execute permission.
- The command which we use to change the permissions is chmod command with arguments u, g and o. Whereas u for user, g for group and o for others.
- After this we can use + for adding permission, – for removing permission and = for assigning permission. Then we have to specify which permission we have to change r,w and x also we can use them with combination. And we also have to provide the file name where we are changing permissions.
- For example, we are changing permission on file called file1.txt .so we will use chmod command with chmod o+x File1.txt.

```
File  Edit  View  Search  Terminal  Help
gajanan@gd-4184:~/Documents$ chmod o+x File1.txt
gajanan@gd-4184:~/Documents$ ls -l
total 12
-rw-r--r-x 1 gajanan gajanan    4 Sep  3 09:48 File1.txt
-rw-r--r-- 1 gajanan gajanan 7715 Sep  3 09:47 Sample.odt
gajanan@gd-4184:~/Documents$
```

- You can also change multiple permissions at once. For example, if you want to take all permissions away from everyone, you would type:

```
chmod ugo-rwx File1.txt
```

- The code above revokes all the read(r), write(w) and execute(x) permission from all user(u), group(g) and others(o) for the file xyz.txt which results to this



- Another example can be this:

```
chmod ug+rw,o-x File1.txt
```

- The code above adds read(r) and write(w) permission to both user(u) and group(g) and revoke execute(x) permission from others(o) for the file abc.mp4.
- Something likes this:

```
chmod ug=rx,o+r Sample.txt
```

assigns read(r) and execute(x) permission to both user(u) and group(g) and add read permission to others for the file abc.c.

Absolute form:

- The other way to use the chmod command is the absolute form, in which you specify a set of three numbers that together determine all the access classes and types.
- Rather than being able to change only particular attributes, you must specify the entire state of the file's permissions.
- The three numbers are specified in the order: user (or owner), group, and other. Each number is the sum of values that specify read, write, and execute access: Octal bits are used for read,write and execute permissions. Which has base 8. for read permission 4 (100) bits, write permission requires 2 (010) bits, and execute requires 1 (001) bits.
- **Look at the following table:**

Binary	Octal	Permissions	Significance
000	0	---	No permissions
001	1	--x	Execute
010	2	-w-	Write
011	3	-wx	Write execute
100	4	r--	Read
101	5	r-x	Read and execute
110	6	rw	Read and write
111	7	rws	Read,write,execute

Example: `$chmod 777 Sample.odt`

with this command we are specifying read, write and execute permissions to the user, group and others to a file sample.odt file

```
File  Edit  View  Search  Terminal  Help
gajanan@gd-4184:~/MyDocs/files$ ls -l
total 20
-rw-r--r-- 1 gajanan gajanan  847 Aug 25 18:27 MULTIPLI.c
-rw-r--r-- 1 gajanan gajanan    0 Sep  3 20:15 MyDocs
drwxr-xr-x 2 gajanan gajanan 4096 Sep  3 20:40 MyFolder
drwxr-xr-x 2 gajanan gajanan 4096 Sep  3 20:18 new
-rw-r--r-- 1 gajanan gajanan 7715 Sep  3 09:47 Sample.odt
gajanan@gd-4184:~/MyDocs/files$ chmod 777 Sample.odt
gajanan@gd-4184:~/MyDocs/files$ ls
MULTIPLI.c  MyDocs  MyFolder  new  Sample.odt
gajanan@gd-4184:~/MyDocs/files$ ls -l
total 20
-rw-r--r-- 1 gajanan gajanan  847 Aug 25 18:27 MULTIPLI.c
-rw-r--r-- 1 gajanan gajanan    0 Sep  3 20:15 MyDocs
drwxr-xr-x 2 gajanan gajanan 4096 Sep  3 20:40 MyFolder
drwxr-xr-x 2 gajanan gajanan 4096 Sep  3 20:18 new
-rwxrwxrwx 1 gajanan gajanan 7715 Sep  3 09:47 Sample.odt
gajanan@gd-4184:~/MyDocs/files$
```

- The 7 indicates read, write and execute permissions (4+2+1). to provide read only permissions we can provide 7 4 4 where user can read, write and execute, group and other can read only.

  Example: `$ chmod 744 Sample.odt`

```
gajanan@gd-4184:~/MyDocs/files$ ls
MULTIPLI.c MyDocs MyFolder new Sample.odt
gajanan@gd-4184:~/MyDocs/files$ ls -l
total 20
-rw-r--r-- 1 gajanan gajanan 847 Aug 25 18:27 MULTIPLI.c
-rw-r--r-- 1 gajanan gajanan 0 Sep 3 20:15 MyDocs
drwxr-xr-x 2 gajanan gajanan 4096 Sep 3 20:40 MyFolder
drwxr-xr-x 2 gajanan gajanan 4096 Sep 3 20:18 new
-rwxr--r-- 1 gajanan gajanan 7715 Sep 3 09:47 Sample.odt
```

Summary

- In Linux files are assumed of three types ordinary files, directories and special files or they are called device files.
- Directories maintain the filename and its associated inode numbers. A device file contains no data but the kernel uses the attributes of the file to operate the device.
- Linux contains Hierarchical structure of directories where the parent is root (/) which contains directories and directories contains file. Where they have parent child relationship.
- HOME variable it is the home or login directory where the user is placed initially.
- PATH it contains the set of paths where the system searches for an executable file.

- Pwd command is used to tell the present working directory.
- An absolute path is defined as the specifying the location of a file or directory from the root directory(/).
- Relative path is defined as the path related to the present working directly(pwd). It starts at your current directory and never starts with a /.
- mkdir command used to create user directories. rmdir is used to remove user directories.
- . and .. represents the files location relative to the current and parent directory respectively.
- cat command allows us to create single or multiple files, view contain of file, concatenate files and redirect output in terminal or files.
- mv it renames a file or moves a group of files to a different directory.
- rm deletes the specified file it can be used with wild cards * and ? As in DOS, to delete all files of a specified type.
- cp creates an exact copy of a file with a different name. Wc command counts words, lines and characters or bytes. Cd is used to switch to specified directory, fails if user is not having permission to access directory.
- ls command displays the contents of current directory. It can be used with several options as -l, -d etc. A file's owner uses chmod to alter file permissions. The permissions can be relative when used with + or – symbols, or absolute when used with octal numbers. The octal numbers 7 includes read (4), write (2) and execute permissions (1) bit. For security reasons the regular file shouldn't normally have writer permissions for group and others. A files access rights are also influenced by the permissions of its directory which usually has read and execute permissions for all. Chown is used to change ownership.

Check Your Understanding

1. What is the command used to remove files.
   (a) dm (b) rm
   (c) rm (d) delete
2. What is the command used to count total number of lines, words and characters contained in file?
   (a) countw (b) wcount
   (c) wc (d) countp
3. What is the command used to remove directory.
   (a) rdir (b) rmdir
   (c) rd (d) remove
4. Which directory contains configuration files in Linux?
   (a) /etc (b) /bin
   (c) /dev (d) /root
5. Command to create a file in Linux
   (a) cat (b) echo
   (c) touch (d) all of above

6. What is the shortest command to take you to home directory?
   (a) cd ~ (b) cd $HOME
   (c) cd (d) ..
7. Which command is used to list all the files in your current directory (including hidden).
   (a) ls -l (b) ls -t
   (c) ls -a (d) ls -i
8. In Linux everything is stored as ____
   (a) File (b) directory
   (c) executable (d) none of the above
9. Which command is used to copy contents one file to another?
   (a) copy (b) cp
   (c) cv (d) mkdir
10. Which command is used to show present working directory in Linux.
    (a) pwd (b) pws
    (c) ls (d) cat

Answers

1. (c)	2. (c)	3. (b)	4. (a)	5. (d)	6. (a)	7. (c)	8. (a)	9. (b)	10. (a)

Practice Questions

Short Answer Questions:

1. Define absolute and relative path.
2. What is file? List the basic file types.
3. What is difference between dot(.) and double dot(..).
4. What is use of HOME variable?
5. What is use of cd command in operating system?

Long Answer Questions:

1. Explain three kinds of permissions in Linux.
2. Explain the Linux 'cd' command options along with the description?
3. Explain ls command with options?
4. Explain Linux directory commands.
5. What is File? Explain different categories of File.
6. Explain the following commands with example:
   (i) pwd (ii) mkdir (iii) cd (iv) rmdir
7. Explain absolute methods of changing permissions by giving examples.
8. Explain briefly Absolute and relative path names with examples.
9. Briefly Describe:
   (i) HOME (ii) PATH (iii) wc (iv) pwd

10. Assuming the file current permissions are rwx —r r -x. Specify chmod expression required to change the following using absolute and relative method of assigning permissions: (i) rwxrwxr-x (ii) r-x r-x –x (iii) r-- r-- w--
11. Which command is used for listing file attributes? Explain the significance of each field in output.
12. Explain the usage of cat commands with examples.
13. What is the use of chmod command? Differentiate absolute and symbolic modes with examples.
14. Explain directory structure of Linux.
15. Using cat command, create a file named 'names.txt' containing at least ten names and addresses of your friends(firstname, surname, streetname, cityname). Type the following commands and give the output.

Command	Output
`wc -ls names.txt`	
`mkdir dir1 dir2`	
`cp names.txt dir1`	
`cp names.txt list`	
`rmdir dir2`	
`cd dir2`	
`rm names.txt`	
`cd`	
`pwd`	
`ls -l`	
`mv list list.txt`	

16. Give the commands to perform the following actions and give the output:
    (i) Display the home directory followed by path.
    (ii) Write the contents of directory of file.
    (iii) Create a file named manualcp containing manual for cp command.
    (iv) Create a file containing word count of each and every file in the current directory plus a total at the end.
    (v) Create a single file containing the data from all.txt files in the current directory.

■■■

4...

Using Shells and Vi Editor

Objectives...

- To understand Shell? And Different types of shells.
- To understand the shells interpretive cycle, Wild cards and file name generation, Removing the special meanings of wild cards.
- To study Pipe, The grep, egrep commands.
- To study Vi editor, Different ways of invoking and quitting vi, Different modes of vi, Input mode commands, Command mode commands, The ex mode commands.

4.1 INTRODUCTION

- The GNU/Linux shell is a special interactive utility. It provides a way for users to start programs, manage files on the file system, and manage processes running on the Linux system. The core of the shell is the command prompt. The command prompt is the interactive part of the shell. It allows you to enter text commands, and then it interprets the commands and executes them in the kernel.

4.2 WHAT IS SHELL?

- A shell is special user program which provide an interface to user to use operating system services. Shell accept human readable commands from user and convert them into something which kernel can understand.
- It is a command language interpreter that execute commands read from input devices such as keyboards or from files. The shell gets started when the user logs in or start the terminal.
- Shell is a unique and multifaceted program. It is also a process that creates an environment for you to work. It allows users to execute commands by typing them manually, or automatically in program called shell script.
- it is also a programming language of its own with complete programming language constructs such as conditional execution, loops, variables, functions and many more.
- There can be command line shell and Graphical shells. Command line Shell can be accessed by user using a command line interface. A special program called Terminal in linux/macOS 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.

- Graphical shells 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. Window OS or Ubuntu OS can be considered as good example which provide GUI to user for interacting with program.
- To find what are available shells on your system type the following command,

  $cat /etc/shells

4.2.1 Different types of shells

- There are different types of shells provided by linux operating system as follows:

1. BASH = Bourne Again Shell:

- Steve Bourne written the Bourne shell. Bash shell is a command language interpreter, that executes commands, the commands are read from standard input device like keyboard, file.
- It is most widely used shell in Linux systems. It is used as default login shell in Linux systems and in macOS. It is shown with $ prompt. If you want to see that type of shell you are using just type following command.

Example: We have written following script using Bash shell

```
$gedit Hello_world.sh
#!/bin/bash
# My first Bash Script
echo "Hello World"
echo $SHELL
echo 'date'
```

- We have executed shell script using ./Hello_world.sh and all the commands we have written in Hello_world.sh has been executed the result is displayed as follows:

2. C shell:

- The C Shell (or csh) is a Linux shell basically it is a command processor typically run in a text window, allowing the users to type commands. the % character is the default prompt.

- The C Shell also reads commands from files, called a script. Like all Linux shells it supports filename, wildcarding, piping, command substitutions, variables and control structures for condition testing and iteration.
- The C shell's syntax and usage are very similar to the C programming language. It provides lackings in Bourne shell.

3. **KSH (Korn Shell):**
   - Ksh stands for Korn shell and was designed and developed by David G. Korn.
   - It is a complete, powerful, high-level programming language and also an interactive command language just like many other Linux/GNU Linux shells.
   - K shell has features of both B and C shell along with additional feature.

4.3 THE SHELLS INTERPRETIVE CYCLE

- The shell is a type of program called an interpreter. An interpreter operates in a simple loop: It accepts a command, interprets the command, executes the command, and then waits for another command. The shell displays a "prompt," to notify you that it is ready to accept your command.

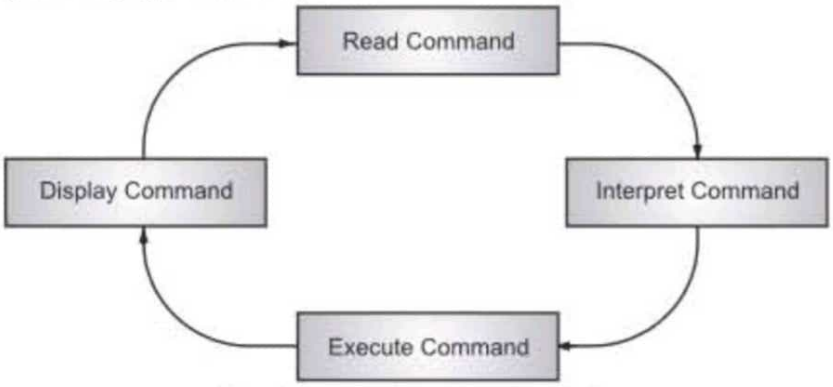


Fig. 4.1: Shells Interpretive Cycle

Following activities are performed by the shell in interpretive cycle:

1. The shell issues the prompt and waits for you to enter the command.
2. After command is entered, shell scans the command line for the metacharacters and expands abbreviations like * (wild card character) to recreate simplified command line.
3. It then passes on the command line to the execution.
4. The shell waits for command to execute. It cannot do anything while the command is running.
5. After command is executed, the prompt reappears and shell waits to accept next command. So this cycle goes on.

4.4 WILD CARDS AND FILE NAME GENERATION

- When you have number of files named in a series (for example chap01, chap02) of filenames with common character (user1,userfile2,etc) you can use wildcards (also called as metacharacters) to specify many files at once.

- These special characters are *(asterisk),?(question mark) and [] (square brackets). When they are used in a file name given as an argument to a command:
  1. *: An asterisk is replaced by any number of characters in a filename. For example, file* would match file1, file2, file3, etc. if those files were in the same directory. You can use this to save typing for a single filename (for example, al* for alphabet.txt) or to name many files at once (as in ae*).
  2. ?: A question mark is replaced by any single character (so h?p matches hop and hip, but not help).
  3. []: These are surrounded by a choice of characters you'd like to match. Any one of the characters between the brackets will be matched. For example, [Cc]hapter would match either Chapter or chapter, but [ch]apter would match either capter or hapter. Use a hyphen (-) to separate a range of consecutive characters. For example, chap[13] would match chap1, chap2, or chap3.
- You can use them with any command such as ls command or rm command to list or remove files matching a given criteria.
- Following are some examples of wild card characters with ls command.

(i) * **(asterisk):** This command matches all files with names starting with 1 (which is the prefix) and ending with one or more occurrences of any character. We can also write file*, fil* or fi*

```
$ ls -l f*
```

```
gajanan@gd-4184:~/MyDocs/files$ ls -l f*
-rw-r--r-- 1 gajanan gajanan 88 Sep 28 12:08 file2.txt
-rw-r--r-- 1 gajanan gajanan 14 Sep 11 08:58 file3.txt
-rw-r--r-- 1 gajanan gajanan 20 Sep 18 09:51 file4.txt
gajanan@gd-4184:~/MyDocs/files$
```

If we will use * with echo command this will display all the files present in the directory.

```
gajanan@gd-4184:~/MyDocs/files$ echo *
dummyfile.txt file2.txt file3.txt file4.txt Linuxfile.txt MULTIPLI.c MyDocs MyFo
lder new Sample.odt testvi.txt
gajanan@gd-4184:~/MyDocs/files$
```

(ii) ? **(question mark):** It matches a single character one. Rm fil?* this will delete all files named file1, file2, file3 etc.

```
gajanan@gd-4184:~/MyDocs/files$ ls
dummyfile.txt  file2.txt  file4.txt      MULTIPLI.c  MyFolder  Sample.odt
file1.txt      file3.txt  Linuxfile.txt  MyDocs      new       testvi.txt
gajanan@gd-4184:~/MyDocs/files$ rm fil?*
gajanan@gd-4184:~/MyDocs/files$ ls
dummyfile.txt  MULTIPLI.c  MyFolder  Sample.odt
Linuxfile.txt  MyDocs      new       testvi.txt
gajanan@gd-4184:~/MyDocs/files$
```

(iii) [] open and close brackets: With linux shell that supports regular expressions that open and close brackets wildcard match a single character in a range. For example [a-z] matches any character through a to z. If we will write [1-9] it will match all the characters which contains 1 to 9 numbers.
[!ijk] – it will match a single character that is not an i, j or k.
[!x-z] – it will match a single character that is not within the ASCII range of the characters x and z.

```
gajanan@gd-4184:~/MyDocs/files$ ls -l fi[a-z]e1.txt
-rw-r--r-- 1 gajanan gajanan 20 Sep 28 17:14 file1.txt
gajanan@gd-4184:~/MyDocs/files$ ls -l file[1-9].txt
-rw-r--r-- 1 gajanan gajanan 20 Sep 28 17:14 file1.txt
-rw-r--r-- 1 gajanan gajanan 20 Sep 28 17:14 file2.txt
-rw-r--r-- 1 gajanan gajanan 20 Sep 28 17:15 file3.txt
-rw-r--r-- 1 gajanan gajanan 20 Sep 28 17:15 file4.txt
gajanan@gd-4184:~/MyDocs/files$
```

You can combine wildcards to build a complex filename matching criteria as described in the following examples.

```
$ls??le*
```

This command will match all filenames prefixed with any two characters followed by `fi` but ending with one or more occurrence of any character.

```
gajanan@gd-4184:~/MyDocs/files$ ls ??le*
file1.txt  file2.txt  file3.txt  file4.txt
gajanan@gd-4184:~/MyDocs/files$
```

4.5 REMOVING THE SPECIAL MEANINGS OF WILD CARDS

- The characters <, >, |, and & are four examples of special characters that have particular meanings to the shell. The wildcards we saw earlier in this chapter (*,?, and [...]) are also special characters.

Quoting:

- Quoting is used to disable the special meaning of the special characters. There are many shell metacharacters which have specific meanings. But when you need to represent those characters then it will require to remove the special meaning of those characters and it is done by quoting the character. You can do this task by using three ways. These are escape characters, single quotes and double quotes

Single quote

- All special characters between these quotes lose their special meaning.

Double quote

- Most special characters between these quotes lose their special meaning with these exceptions,

```
$, \$, \, \\
```

- Lets take an example if we use command,

```
$echo "$USER"
```

it will not protect it because double quotes does not protect $. where as command $ echo '$user' will take it everything as literal. And it will protect characters.

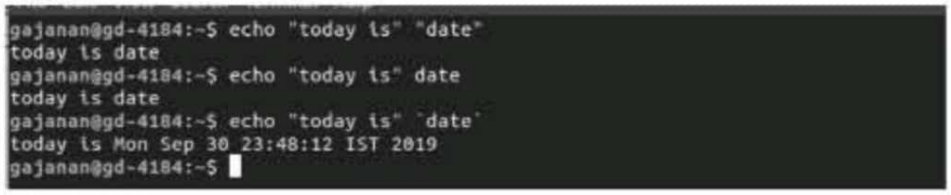
```
gajanan@gd-4184:~$ echo "$USER"
gajanan
gajanan@gd-4184:~$ echo '$USER'
$USER
gajanan@gd-4184:~$
```

Backslash

- Any character immediately following the backslash loses its special meaning.
- Example: suppose we are going to use command.
  `$mkdir one two`
- It will create two directories called one and two. Because space is the default delimiter. So if i want to create a directory called one two then i have to write command with double quotes or Back slash.

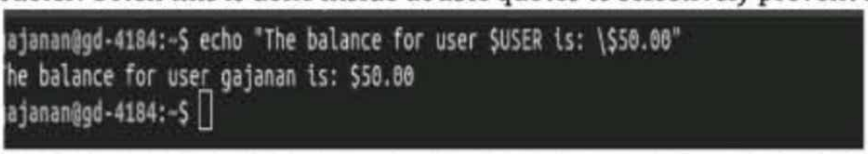
Back quote:

- Anything in between back quotes would be treated as a command and would be executed.



Escape characters:

- Bash escape character is defined by non-quoted backslash (\). It preserves the literal value of the character followed by this symbol. Normally, $ symbol is used in bash to represent any defined variable. But if you use escape in front of $ symbol then the meaning of $ will be ignored and it will print the variable name instead of the value. Run the following commands to show the effects of escape character (\).
- For example Sometimes you only want to quote a single character. To do this, you can precede a character with a backslash, which in this context is called the escape character. Often this is done inside double quotes to selectively prevent an expansion.



- It is also common to use escaping to eliminate the special meaning of a character in a filename. For example, it is possible to use characters in filenames that normally have special meaning to the shell. These would include "$", "!", "&", " ", and others. To include a special character in a filename you can to this:
  `$mv file1\&filename file2_filename`
- To allow a backslash character to appear, escape it by typing "\\". Note that within single quotes, the backslash loses its special meaning and is treated as an ordinary character.

4.6 THREE STANDARD FILES AND REDIRECTION

4.6.1 Three Standard Files

- **Streams:** input and output in linux environment is distributed across the streams. The streams can also be numbered with the file descriptors stdin(0), stdout(1) and stderr(2).
  1. **Standard Input(stdin):** The file (or stream) representing input, which is connected to the keyboard. The standard input stream typically carries data from a user to a program. Programs that expect standard input usually receive input from a device, such as a keyboard. Standard input is terminated by reaching EOF (end-of-file).
  2. **Standard Output(stdout):** The file (or stream) representing output, which is connected to the displays. standard output displays the output. It is used by all commands to display the output. The default output is displayed on the screen. It is denoted by number 1. it is also known as stdout called as standard output.

     The terminal, the default destination a file using the redirection symbols > and >> as input to another program using a pipeline.
  3. **Standard Error(stderr):** The file (or stream) representing error messages. It is also connected to display. stderr also known as standard error, it is used to write all the system errors. the default file descriptor where the process can write error messages. In Linux operating system stderr is defined by the POSIX standard. Its default file descriptor number is 2.

4.6.2 Redirection

- It 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.
- In GNU/Linux everything is a file, including the hardware. The common return values are 0 for input that is keyboard. 1 for output that is screen and 2 for Error is again screen.
- A redirector opens above streams to talk to a user or program. Default standard input is the keyboard. < is the input redirection symbol.

  Syntax: `command< filename`
- Before a command is executed, its input and output may be redirected using a special notation interpreted by the shell. Redirection allows command's file handles to be duplicated, opened, closed, made to refer to different files, and can change the files the command reads from and writes to.
- Redirection may also be used to modify file handles in the current shell execution environment.
- The following redirection operators may precede or appear anywhere within a simple command or may follow a command. Redirections are processed in the order they appear, from left to right.
- Each redirection that may be preceded by a file descriptor number may instead be preceded by a word of the form {varname}. In this case, for each redirection operator except >&- and <&-, the shell will allocate a file descriptor greater than 10 and assign

it to {varname}. If >&- or <&- is preceded by {varname}, the value of varname defines the file descriptor to close. If {varname} is supplied, the redirection persists beyond the scope of the command, allowing the shell programmer to manage the file descriptor himself.

- In the following descriptions, if the file descriptor number is omitted, and the first character of the redirection operator is '<', the redirection refers to the standard input (file descriptor 0). If the first character of the redirection operator is '>', the redirection refers to the standard output (file descriptor 1).

Redirecting input:

- Redirection of input causes the file whose name results from the expansion of word to be opened for reading on file descriptor n, or the standard input (file descriptor 0) if n is not specified.
- The general format for redirecting input is:

```
command < filename
```

Example: lets take see command of sorting file contents using input file descriptor,

```
gajanan@gd-4184:~/MyDocs/files$ cat file2.txt
5
9
4
3
2
8
1
6
gajanan@gd-4184:~/MyDocs/files$ sort <file2.txt
1
2
3
4
5
6
8
9
gajanan@gd-4184:~/MyDocs/files$
```

Redirecting output:

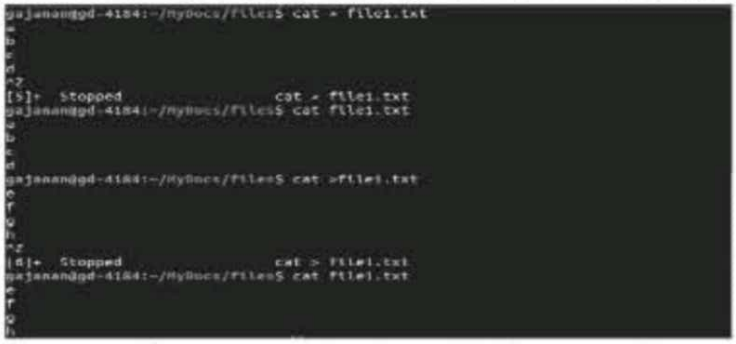
- > it is the redirection symbol.
- The general format for redirecting output is:

```
command > filename
```

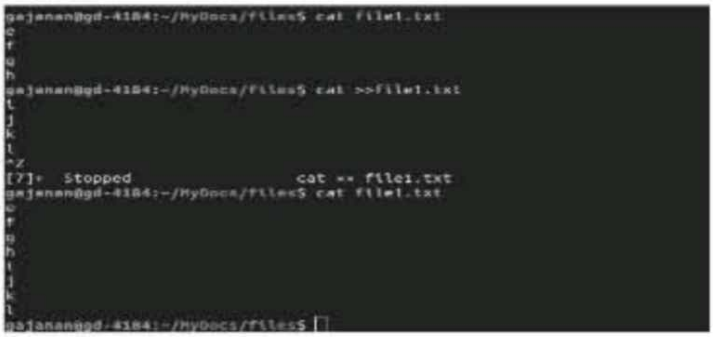
- Lets take an example of ls command. Ls is nothing but the information of files present in the directory. When we enter command ls > ls_file.txt. All the file names are redirected to the file ls_file.txt as you can see the output.

```
gajanan@gd-4184:~/MyDocs/files$ ls > ls_file.txt
gajanan@gd-4184:~/MyDocs/files$ ls
dummyfile.txt  file3.txt      ls_file.txt  MyFolder    testvi.txt
file1.txt      file4.txt      MULTIPLI.c   new
file2.txt      Linuxfile.txt  MyDocs       Sample.odt
gajanan@gd-4184:~/MyDocs/files$ cat ls_file.txt
dummyfile.txt
file1.txt
file2.txt
file3.txt
file4.txt
Linuxfile.txt
ls_file.txt
MULTIPLI.c
MyDocs
MyFolder
new
Sample.odt
testvi.txt
gajanan@gd-4184:~/MyDocs/files$
```

Example: Using cat command we will create file1.txt. We will type some contents into the file and save it. Now when we will again create same file called file1.txt and we enter some contents the old contents will get lost. We can see in picture.



- Now we will use >> redirection with the same file name file1.txt and we will add contents into it. The old contents of file plus newly added contents will remain into the file.



Redirecting Error:

- The default standard error output is visible on the screen or monitor. 2> symbol is the error redirection symbol.
- It has the following syntax:

```
command 2> error.txt
```

Example: Now we want to redirect the result of rm command. So we will write command $rm /tmp/file1.txt . If the file is not present we will get error saying no such file or directory is present. Now we will redirect the error message to a file called error.txt. If we will see the file contents you will get same error message.

```
gajanan@gd-4184:~/MyDocs/files$ rm /tmp/file1.txt
rm: cannot remove '/tmp/file1.txt': No such file or directory
gajanan@gd-4184:~/MyDocs/files$ rm /tmp/file1.txt 2>error.txt
gajanan@gd-4184:~/MyDocs/files$ rm /tmp/file1.txt
rm: cannot remove '/tmp/file1.txt': No such file or directory
gajanan@gd-4184:~/MyDocs/files$ cat error.txt
rm: cannot remove '/tmp/file1.txt': No such file or directory
gajanan@gd-4184:~/MyDocs/files$
```

Appending Redirected output:

- Redirection of output in this fashion causes the file whose name results from the expansion of word to be opened for appending on file descriptor n, or the standard output (file descriptor 1) if n is not specified. If the file does not exist it is created.
- The general format for appending output is:

  `[n]>>word`

4.7 CONNECTING COMMANDS: PIPE, THE GREP, EGREP COMMANDS

1. Pipe:

- Pipes are used to redirect a stream from one program to another. It creates a chain of commands.
- It connects the output of one command to the input of the next command. Pipes are unidirectional i.e. data flows from left to right through the pipeline.
- **Syntax:** `command1|command2|command3|.....|commandN.`
- The Linux pipe is represented by a vertical bar.

  `*|*`

Example: We will see an example of cat command with wc pipe.

$ cat file1.txt|wc

This command will show first number of words,number of lines and number of characters in the file1.txt

```
gajanan@gd-4184:~/MyDocs/files$ cat file1.txt|wc
      8       8      16
gajanan@gd-4184:~/MyDocs/files$
```

Example: Lets take another example of 1s with less.

`$ls|less`

- This takes the output of ls, which displays the contents of your current directory, and pipes it to the less program. less displays the data sent to it one line at a time.
- ls normally displays directory contents across multiple rows. When you run it through less, each entry is placed on a new line.
- Though the functionality of the pipe may appear to be similar to that of > and >> (standard output redirect), the distinction is that pipes redirect data from one command to another, while > and >> are used to redirect exclusively to files.

`ls -l|more:`

```
gajanan@gd-4184:~$ ls -l|more
total 88
-rw-r--r--  1 gajanan gajanan     4 Sep  1 09:48 Demo.txt
drwxr-xr-x  2 gajanan gajanan  4096 Sep 27 13:51 Desktop
drwxr-xr-x  3 gajanan gajanan  4096 Sep 25 15:34 Documents
drwxr-xr-x  7 gajanan gajanan  4096 Sep 26 15:42 Downloads
-rw-r--r--  1 gajanan gajanan   238 Aug 30 11:05 eclipse.desktop
drwxrwxr-x  5 gajanan gajanan  4096 Aug 30 11:50 eclipse-workspace
-rw-r--r--  1 gajanan gajanan  8980 Aug 29 18:58 examples.desktop
-rw-r--r--  1 gajanan gajanan   847 Aug 25 18:27 MULTIPLI.c
drwxr-xr-x  2 gajanan gajanan  4096 Aug 29 19:12 Music
drwxr-xr-x 17 gajanan gajanan  4096 Sep 22 22:48 MyDocs
drwxr-xr-x  2 gajanan gajanan  4096 Sep 28 17:28 Pictures
drwxr-xr-x  2 gajanan gajanan  4096 Aug 29 19:12 Public
-rw-r--r--  1 gajanan gajanan   185 Sep 23 22:33 sample1.txt
-rw-r--r--  1 gajanan gajanan    45 Sep 26 09:51 sample.txt
drwxr-xr-x  6 gajanan gajanan  4096 Sep 11 18:04 snap
drwxr-xr-x  2 gajanan gajanan  4096 Aug 29 19:12 Templates
-rw-r--r--  1 gajanan gajanan  8192 Sep  1 11:28 test
-rw-rw-r--  1 gajanan gajanan  2025 Sep 11 18:35 tree.drawio
drwxr-xr-x  2 gajanan gajanan  4096 Aug 29 19:12 Videos
gajanan@gd-4184:~$
```

Example: Sorting data in a file.

```
$sort names.txt|uniq
```

It will sort all the data in the names.txt file and will show unique records.

```
gajanan@gd-4184:~/MyDocs/files$ sort names.txt|uniq
amol
gajanan
madhav
ramesh
sunil
suresh
vikas
gajanan@gd-4184:~/MyDocs/files$
```

2. The Grep command:

- The grep command stands for Global Regular Expression Print. Grep command scans its input for a pattern and displays the lines containing the pattern. It displays the line numbers or file names where the pattern occurs.

The general syntax of grep command:

```
grep option pattern filenames(s)
```

Options for grep Command:

1. **-i Option:** The -i option enables to search for a string case insensitively in the give file. It matches the words like "LINUX", "Linux", "linux".

   ```
   command: $grep -i Linux Linuxfile.txt.
   ```

   This command finds the word Linux in a file called Linuxfile.txt and it has been highlighted in red color

```
gajanan@gd-4184:~/MyDocs/files$ cat Linuxfile.txt
A Brief History of Linux. Linux is free os.It is one of the most popular
operating system worldwide because of its large support base and distribution..
Linux is a freely distributable version of Unix,originally developed
by Linux Torvalds, who began work on Linux in 1991 as a student at the
Linux is very easy to learn.LINUX Operating system or Unix operating Systen

gajanan@gd-4184:~/MyDocs/files$ grep -i Linux Linuxfile.txt
A Brief History of Linux. Linux is free os.It is one of the most popular
Linux is a freely distributable version of Unix,originally developed
by Linux Torvalds, who began work on Linux in 1991 as a student at the
Linux is very easy to learn.LINUX Operating system or Unix operating Systen
gajanan@gd-4184:~/MyDocs/files$
```

2. **-c option:** We can find the number of lines that matches the given string/pattern.

   ```
   command: grep -c Linux Linuxfile.txt
   ```

   Using this command we will find the word Linux how many times it is occured in the file Linuxfile.txt.

   Output:

```
gajanan@gd-4184:~/MyDocs/files$ grep -c Linux Linuxfile.txt
4
gajanan@gd-4184:~/MyDocs/files$
```

3. **-l option**: We can just display the files that contains the given string/pattern

   command: $grep -l "Linux" *

```
gajanan@gd-4184:~/MyDocs/files$ grep -l "Linux" *
Linuxfile.txt
grep: MyFolder: Is a directory
grep: new: Is a directory
gajanan@gd-4184:~/MyDocs/files$
```

4. **-w option:** 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. It is somewhat similar with – i option of grep command.

   command: $grep -w Linux Linuxfile.txt

   Output:

```
gajanan@gd-4184:~/MyDocs/files$ grep -w Linux Linuxfile.txt
A Brief History of Linux. Linux is free os.It is one of the most popular
Linux is a freely distributable version of Unix,originally developed
by Linux Torvalds, who began work on Linux in 1991 as a student at the
Linux is very easy to learn.LINUX Operating system or Unix operating System
gajanan@gd-4184:~/MyDocs/files$
```

5. **-o option:** 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.

   command: $ grep -o "Linux" Linuxfile.txt

   Output:

```
gajanan@gd-4184:~/MyDocs/files$ grep -o Linux Linuxfile.txt
Linux
Linux
Linux
Linux
Linux
Linux
gajanan@gd-4184:~/MyDocs/files$
```

6. **-n option**: Show line number while displaying the output using grep -n:

   command: $grep -n Linux Linuxfile.txt

   Output: As you can see the line numbers have been added with the matching word Linux.

```
gajanan@gd-4184:~/MyDocs/files$ grep -n Linux Linuxfile.txt
1:A Brief History of Linux. Linux is free os.It is one of the most popular
3:Linux is a freely distributable version of Unix,originally developed
4:by Linux Torvalds, who began work on Linux in 1991 as a student at the
5:Linux is very easy to learn.LINUX Operating system or Unix operating System
gajanan@gd-4184:~/MyDocs/files$
```

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

   command: $grep -v Linux Linuxfile.txt

Output:

```
gajanan@gd-4184:~/MyDocs/files$ cat Linuxfile.txt
A Brief History of Linux. Linux is free os.It is one of the most popular
operating system worldwide because of its large support base and distribution..
Linux is a freely distributable version of Unix,originally developed
by Linux Torvalds, who began work on Linux in 1991 as a student at the
Linux is very easy to learn.LINUX Operating system or Unix operating System

gajanan@gd-4184:~/MyDocs/files$ grep -v Linux Linuxfile.txt
operating system worldwide because of its large support base and distribution..

gajanan@gd-4184:~/MyDocs/files$
```

8. **^ option:** 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.

   command: $grep ^Linux Linuxfile.txt

Output:

```
gajanan@gd-4184:~/MyDocs/files$ grep ^Linux Linuxfile.txt
Linux is a freely distributable version of Unix,originally developed
Linux is very easy to learn.LINUX Operating system or Unix operating System
gajanan@gd-4184:~/MyDocs/files$
```

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.

   command: $grep os Linuxfile.txt

Output:

```
gajanan@gd-4184:~/MyDocs/files$ grep os Linuxfile.txt
A Brief History of Linux. Linux is free os.It is one of the most popular
gajanan@gd-4184:~/MyDocs/files$
```

10. **-b option:** with this command precede each matching line with a block number.

    command: $grep -b Linux Linuxfile.txt

Output:

```
gajanan@gd-4184:~/MyDocs/files$ grep -b Linux Linuxfile.txt
0:A Brief History of Linux. Linux is free os.It is one of the most popular
153:Linux is a freely distributable version of Unix,originally developed
222:by Linux Torvalds, who began work on Linux in 1991 as a student at the
293:Linux is very easy to learn.LINUX Operating system or Unix operating System
gajanan@gd-4184:~/MyDocs/files$
```

3. egrep Command:

- It stands for "Extended Global Regular Expression Print. It is a program which scans a specified file line by line, returning lines that contain a pattern matching a given regular expression.

- It works the same way as grep does. It treats the pattern as an extended regular expression and prints out the lines that match the pattern. If there are several files with the matching pattern, it also displays the file names for each line.
- The egrep command used mainly due to the fact that it is faster than the grep command. The egrep command treats the meta-characters as they are and do not require to be escaped as the case with grep. This allows reducing the overhead of replacing these characters while pattern matching making egrep faster than grep.

Syntax: `egrep [options] 'pattern' files.`

Options for egrep command:

1. **-a option and - - text option:** Print NUM lines of trailing context after matching lines. Places a line containing -- between contiguous groups of matches.

```
gajanan@gd-4184:~/MyDocs/files$ egrep -a large Linuxfile.txt
operating system worldwide because of its large support base and distribution..
gajanan@gd-4184:~/MyDocs/files$ egrep --text large Linuxfile.txt
operating system worldwide because of its large support base and distribution..
gajanan@gd-4184:~/MyDocs/files$
```

2. **-c or - - count option:** Suppress normal output; instead print a count of matching lines for each input file.

```
gajanan@gd-4184:~/MyDocs/files$ egrep -c large Linuxfile.txt
1
gajanan@gd-4184:~/MyDocs/files$ egrep --count large Linuxfile.txt
1
gajanan@gd-4184:~/MyDocs/files$
```

3. **-V or –version option:** it provides the grep version.

```
gajanan@gd-4184:~/MyDocs/files$ egrep -V
grep (GNU grep) 3.1
Copyright (C) 2017 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Written by Mike Haertel and others, see <http://git.sv.gnu.org/cgit/grep.git/tre
e/AUTHORS>.
gajanan@gd-4184:~/MyDocs/files$
```

4. **-H option:** it prints the file name with every match.

```
File  Edit  View  Search  Terminal  Help
gajanan@gd-4184:~/MyDocs/files$ egrep -H Linux Linuxfile.txt
Linuxfile.txt:A Brief History of Linux. Linux is free os.It is one of the most p
opular
Linuxfile.txt:Linux is a freely distributable version of Unix,originally develop
ed
Linuxfile.txt:by Linux Torvalds, who began work on Linux in 1991 as a student at
 the
Linuxfile.txt:Linux is very easy to learn.LINUX Operating system or Unix operati
ng System
gajanan@gd-4184:~/MyDocs/files$
```

5. **– i option:** it is not case sensitive. It will display the result matching with word in lower case and upper case.

6. **-I option:** it is case sensitive it will display only that characters which are matching exactly with the given string in command.

```
gajanan@gd-4184:~/MyDocs/files$ egrep -i operating Linuxfile.txt
operating system worldwide because of its large support base and distribution..
Linux is very easy to learn.LINUX Operating system or Unix operating System
gajanan@gd-4184:~/MyDocs/files$ egrep -I operating Linuxfile.txt
operating system worldwide because of its large support base and distribution..
Linux is very easy to learn.LINUX Operating system or Unix operating System
gajanan@gd-4184:~/MyDocs/files$
```

7. **-n option or –line-number option:** it prefix each line of output with the line number within its input file.

```
gajanan@gd-4184:~/MyDocs/files$ egrep -n operating Linuxfile.txt
2:operating system worldwide because of its large support base and distribution.
.
5:Linux is very easy to learn.LINUX Operating system or Unix operating System
gajanan@gd-4184:~/MyDocs/files$ egrep --line-number operating Linuxfile.txt
2:operating system worldwide because of its large support base and distribution.
.
5:Linux is very easy to learn.LINUX Operating system or Unix operating System
gajanan@gd-4184:~/MyDocs/files$
```

8. **w or –word option:** Select only those lines containing matches that form whole words. The test is that the matching substring must either be at the beginning of the line, or preceded by a non-word constituent character. Similarly, it must be either at the end of the line or followed by a non-word constituent character.
9. **- -help option:** it outputs you a brief help message.

```
gajanan@gd-4184:~/MyDocs/files$ egrep --help
Usage: grep [OPTION]... PATTERN [FILE]...
Search for PATTERN in each FILE.
Example: grep -i 'hello world' menu.h main.c

Pattern selection and interpretation:
  -E, --extended-regexp     PATTERN is an extended regular expression
  -F, --fixed-strings       PATTERN is a set of newline-separated strings
  -G, --basic-regexp        PATTERN is a basic regular expression (default)
  -P, --perl-regexp         PATTERN is a Perl regular expression
  -e, --regexp=PATTERN      use PATTERN for matching
  -f, --file=FILE           obtain PATTERN from FILE
  -i, --ignore-case         ignore case distinctions
  -w, --word-regexp         force PATTERN to match only whole words
  -x, --line-regexp         force PATTERN to match only whole lines
  -z, --null-data           a data line ends in 0 byte, not newline

Miscellaneous:
  -s, --no-messages         suppress error messages
  -v, --invert-match        select non-matching lines
```

4.8 Vi EDITOR

4.8.1 Introduction to the Vi Editor

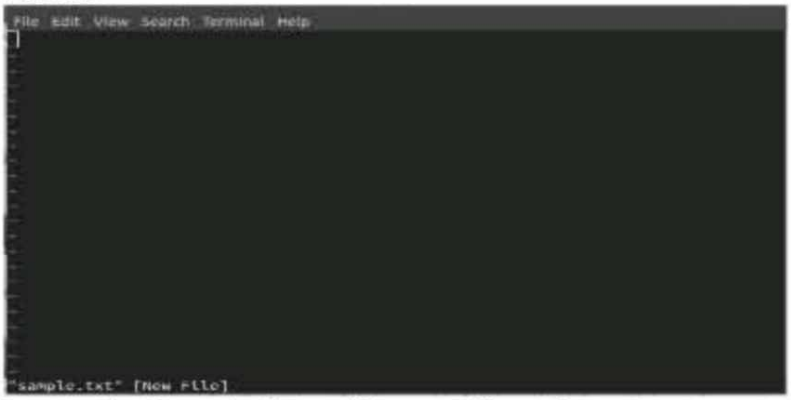
- Linux has number of editors that can process the contents of files, whether it contains data, source or sentences. Vi displays part of file on terminal screen. Vi editor is the most powerful and standard text editor on the system. Vi stands for (visual editor).
- Bill Joy created vi editor for the BSD System. Now this program is standard on linux systems. Vi uses number of internal commands to navigate to any point in a text file

and edit the texts. It also allows you to copy and move text within a file. An improved version of the vi editor which is called the **VIM** has also been made available now. Here, VIM stands for **Vi IM**proved.

- It uses all of the regular keyboard keys for regular commands that is called command mode. You must be in a special insert mode before you can type actual text on the screen.
- Don't use [Caps Lock] as vi commands are case sensitive. The command in small case and upper case is different.
- **Syntax to open vi editor:**

```
vi  <option> <file name>
```

- If you enter command on terminal like vi sample.txt. You will appear on vi editor with following screen.



- If the file is not exist then vi editor will create file at the end you can see name of file with qualifier [New File]. The cursor is positioned at the top and all lines will show this symbol ~.
- To write something on vi editor you should enter into input mode. Now press the key marked i, so you will enter into command mode and you are ready to input the text.
- When you entered the text the cursor will be positioned to the last character of the last line. This is called as current line.Now you can press escape [Esc] key to go back to command mode.
- Actually, the file is not saved yet it is in temporary storage called buffer. To save the file you must write:w command this will write the contents to the file.

4.8.2 Different ways of Invoking and Quitting Vi

Starting Vi Editor:

- There are various ways to start vi editor we will see various commands which will allow us to open vi editor and we can add, make changes into the text as well as after saving our work we can quit the vi editor. So lets have a look at following commands.

Starting the Vi Editor:

1. **$ vi filename:** Created a new file if it is already does not exist, otherwise opens an existing file.
2. **$ vi -R filename:** Opens an existing file in read only mode.
3. **$ view filename:** It also opens an existing file in view or read only mode.
4. **vi + 50 file name:** This will open the file from 50 line number.

Quitting Vi Editor:

1. Exiting Vi Editor

- When you have finished your editing, you need to inform VI that you wish to "quit" the editor and return to your shell. This is done by typing ":q" (quit). VI will respond by updating the file information line with the name of your file in quotes followed by the current number of lines and characters. After exiting the editor, the Linux prompt will again appear on your screen.

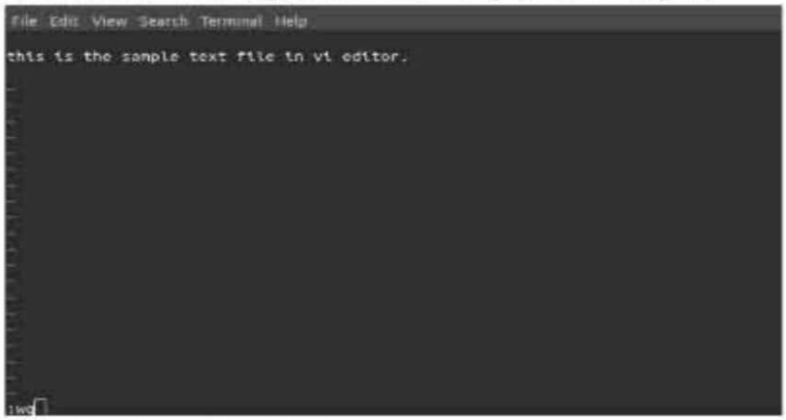
  :q - you have to press [Esc] key and then press: the cursor will appear now press:q it will exit the vi editor.

2. Exiting the Editor -- Retaining Changes

- VI requires that the buffer be empty of newly edited material when you type ":q". Thus if you have made any alterations to the file since the last time you typed ":w", vi will not know how you want these changes handled. The editor will print the statement:

  No write since last change (:quit! overrides)

- At the bottom of the screen. If you decide you want to retain these changes, you must type "**:w**" before issuing a new "**:q**". Most system users get in the habit of combining these two commands into a single command "**:wq**" (write and quit).



3. Exiting the Editor--Discarding Changes

- Sometimes when working on a file, it is desirable to leave the editor without saving the modifications. This is accomplished by typing "**:quit!**". When you use this command, **vi** will immediately discard all alterations made to your file since the last "**:w**". If you have not used a "**:w**" since opening the file, all changes since the beginning of the editing session will be abandoned.
- The action of the "**:quit!**" command is illustrated below.

  :q! - To quit the vi editor without saving any changes you have made:

  (i) if you are currently in insert or append mode,press esc.

  (ii) press colon(:) the cursor should appear at the lower left corner of the screen besides a colon prompt.

  (iii) enter the command q! This will quit the vi editor, and all changes you have made to the document will be lost

Other ways to exit Vi editor:

- Normally you should remember the above three commands. But there are more shortcuts to quit vi editor. These are as follows:
  - **Esc + :x +(Enter):** This command will save and exit.
  - **Esc + :qa + (Enter):** This will quit all open files.
  - **Esc + Shift (ZZ):** It will save and quit.
  - **Esc + Shift (ZQ):** Exit without saving.

4.9 DIFFERENT MODES OF Vi

- Vi operates in various modes, which are used to accomplish various tasks. When you first start vi, you will be placed in command mode. From this point you can issue various commands to manipulate text, to move around in the file, save, quit. Editing in the text is done with the input mode commands.
- Vi editor uses three modes as follows:
  1. Command mode
  2. Input mode
  3. Ex mode

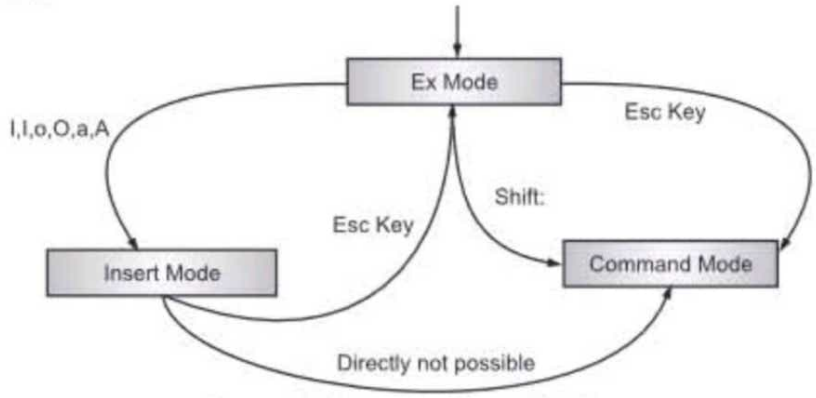


Fig. 4.2: Different modes of vi editor

4.9.1 The Command Mode Commands

- The default mode of the editor where every key pressed is interpreted as a command mode to run on text, in this mode you can cut, copy, paste text.
- This mode also saves the changes you have made to the file. Unnecessary pressing of {Esc] in this mode will sounds beep but also confirms that you are in this mode.
- When vi starts up, it is in Command Mode. This mode is where vi interprets any characters we type as commands and thus does not display them in the window. This mode allows us to move through a file, and to delete, copy, or paste a piece of text. To enter into Command Mode from any other mode, it requires pressing the [Esc] key. If we press [Esc] when we are already in Command Mode, then vi will beep or flash the screen.

- VI receives instructions on what action is required, such as:
  - Bringing text to the screen: scrolling, searching, using goto, moving the cursor, transferring to text input mode text insertion: append, insert, open, change, and replace reading in another file.
  - Deleting (lines, words, or characters), changing (lines, words, or characters), copying or yanking (lines, words, or characters), undoing previous command action, using named buffers, joining lines.
- It is important for the new user to realize that when the majority of VI commands are keyed in, the editor does not echo the command back on the screen; rather, it simply executes the response the command requires.
- You can have several cursor movements in command mode as follows:

1. Moving among Words and Lines

- While these four keys (or your direction keys) can move you just about anywhere you want to go in your file, there are some shortcut keys that you can use to move a little more quickly through a document. To move more quickly among words, you might use the following:

w	Moves the cursor forward one word.
b	Moves the cursor backward one word (if in the middle of a word, b will move you to the beginning of the current word).
e	Moves to the end of a word.
h	To move left one character( ) ←
j	To move down one character( ) ↓
k	To move up one character( ) ↑
l	To move right one character ( ) →
G	To move to end of file.
nG or n	Cursor goes to the specified (n) line (ex. 10G goes to line 10).
$	Move cursor to the end of current line.
0 (zero)	Move cursor to the beginning of current line.

2. Screen Movement

- To move the cursor to a line within your current screen use the following keys:

H	Moves the cursor to the top line of the screen.
M	Moves the cursor to the middle line of the screen.
L	Moves the cursor to the last line of the screen.

- To scroll through the file and see other screens use:

ctrl-f	Scrolls down one screen.
ctrl-b	Scrolls up one screen.
ctrl-u	Scrolls up a half a screen.
ctrl-d	Scrolls down a half a screen.

Two other useful commands for moving quickly from one end to the other of a document are G to move to the end of the file and 1G to move to the beginning of the file. If you precede G with a number, you can move to a specific line in the document (e.g. 15G would move you to line 15).

3. Deleting (or Cutting) Characters, Words, and Lines

- To delete a character, first place your cursor on that character. Then, you may use any of the following commands:

x	Deletes the character under the cursor.
X	Deletes the character to the left of your cursor.
dw	Deletes from the character selected to the end of the word.
dd	Deletes all the current line.
D	Deletes from the current character to the end of the line.

Preceding the command with a number will delete multiple characters. For example, 10x will delete the character selected and the next 9 characters; 10X will delete the 10 characters to the left of the currently selected character. The command 5dw will delete 5 words, while 4dd deletes four lines.

4. Copying (yanking) Characters:

yw	This command copies a word into a buffer.
yy	This command copies a current line from the cursor into a buffer.
5yy	It will copy 5 lines into buffer. Have a look at following picture. When 5yy is entered it has copied 5 lines.

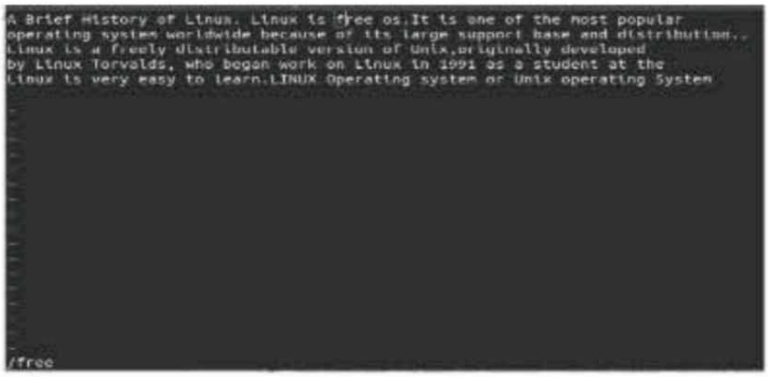
```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,j,k,a[2][2],b[2][2],c[2][2],sum;
printf("Enter the element of first matrix");
for(i=0;i<2;i++)
{
for(j=0;j<2;j++)
{
scanf("%d",&a[i][j]);
}
}
printf("Enter the element of second matrix");
for(i=0;i<2;i++)
{
for(j=0;j<2;j++)
{
scanf("%d",&b[i][j]);
}
}
printf("\nthe element of first matrix=\n");
for(i=0;i<2;i++)
5 lines yanked
```

5. Pasting Characters

p	This command will paste the contents which are copied where the cursor is currently placed. The alphabet P is also used to paste the character.

6. Searching for a pattern:

search-text	It is used for searching a given text with forward slash / and cursor will point to the matched word in a file
?search-text	This option searches for given text from bottom to top.
n	Repeat search in same direction along which previous search was made.
N	Repeat search in direction opposite to that along which previous search was made.



7. Undoing last editing Instructions:

- In command mode, to undo last changes made, we use,

u	This command will undo the changes you have made.
U	This command will restore a line. Linux lets you undo and redo multiple editing instructions. u behaves differently here; repetitive use of this key progressively undoes your previous actions. Further 10u reverses your 10 editing actions. The functions of U remains same here. You can press Ctrl r for redoing your undone actions
J	This command Join next line down to the end of the current line.

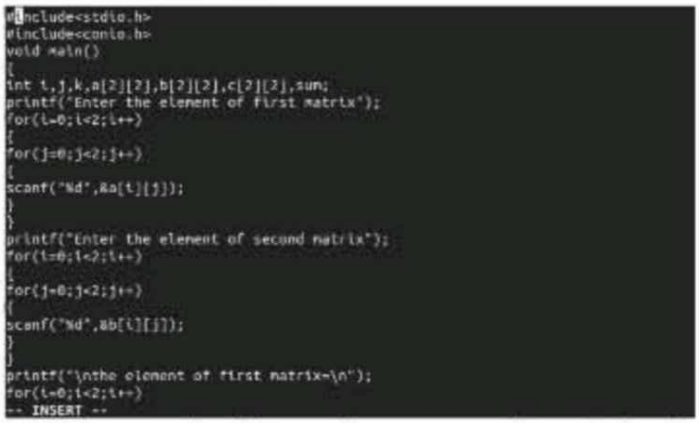
8. Repeating the last command:

- In vi editor the .(dot) key is used for repeating last command. The principal is use the actual command only once and then repeat at other places with the dot command.
- Lets take example to delete 5 lines with 5dd, then to repeat this operation elsewhere, all you have to do it position the cursor at the desired location and then press .(dot) operator. This will repeat the last editing instruction so it will delete 5 lines as last operation.

4.9.2 Input Mode Commands

- Every key pressed after switching to this mode actually shows up as text. This mode is used for inserting the text into the file.
- You can switch to the input mode from the command mode by pressing i on the keyboard.
- Once you are in input mode you are allowed to make changes into the file.
- Remember that you can't move around with the cursor keys in this mode. When you're done entering text, press Esc to go back to command mode. Most editing sessions include moving back and forth between command mode and text input mode.

Input mode commands:



```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,j,k,a[2][2],b[2][2],c[2][2],sum;
printf("Enter the element of first matrix");
for(i=0;i<2;i++)
{
for(j=0;j<2;j++)
{
scanf("%d",&a[i][j]);
}
}
printf("Enter the element of second matrix");
for(i=0;i<2;i++)
{
for(j=0;j<2;j++)
{
scanf("%d",&b[i][j]);
}
}
printf("\nthe element of first matrix=\n");
for(i=0;i<2;i++)
-- INSERT --
```

- As you can see in picture when we press i, a or A it will come in input mode.
- Sometimes the cursor can be a substantial distance from the next editing position on the screen. Large vertical jumps may be made with the use of the mnemonic commands: "H" for high or home, "M" for middle, or "L" for low or last screen line. When you type "H""M" or "L" the cursor will immediately reposition to the first character space on the highest, middle, or lowest screen line.
- After using these three commands for a while, you may decide to become more precise in moving the cursor by adding a number to the command. For example, "3H" would move the cursor to the third line from the top of the screen. The command "3L" would likewise move the cursor to the third line from the bottom of the screen. As you may guess, the middle is always the middle and you can not fancy up the "M" command.
- **Following are more options for input mode commands:**

i	This command inserts text to left of cursor and the existing text is shifted to right.
a	This command appends text to right of cursor and existing text is shifted to right.
I	This command inserts text at the beginning of the line and existing text is shifted to right.
A	This command appends text at end of line.
o	This is used to open line below.
O	This command is used to open line above.
R	This replaces text from cursor to right and existing text is overwritten.
s	This replaces single character under cursor with any number of characters.
S	It replaces entire line.

4.9.3 The Ex mode Commands

- The ex mode is an extension of command mode. To get into it, press Esc and then: (the colon). The cursor will go to the bottom of the screen at a colon prompt. Write your file by entering:w and quit by entering:q. You can combine these to save and exit by entering:wq. However, if you're finished with your file, it's generally more convenient to type Shift-z-z from command mode.

- It is also called last line mode. The cursor will jump to the last line of the screen and vi will wait for a command, this mode is used to handle files (like saving) and perform the substitution. When we press: (colon) in the command mode we enter ito this mode.
- So when we want to save the file contents by saving our file contents or without saving we want to exit the vi editor we should use following commands which are handled by ex Mode.
  1. **Saving the work(:w):** This command is used to write the file contents from buffer to disk.
     :w (Enter) after this cursor will again will blink on the first line if you want to add more contents in to the file press 'i' or 'a' for entering text.

     With this mode you can optionally use the name of file. In this case the contents are written to another file.
  2. **Saving and Quitting: (:x and:wq):** The command:x stands for exit if we want to save and quit editor and return back to shell we have to press:x [Enter].
     Another way is to use:wq command this will also save and quit the editor. After adding contents press [Esc] and then use:wq [Enter].
     Also we can use shift + zz this will also use to save and exit the editor.
  3. **Discarding changes (or quit without saving):** We can also abort the changes we have made and quit the editing mode of vi editor without saving the contents of the file. To do this use:q stands for quit the editing mode and return back to the shell. Suppose we have edited the file contents and want to exit the editor. It will give error saying that file contents have been changed to save this we can use:w.

If we have modified contents of the file and we want to discard the changes we have made. So to forcefully quit the editing mode press:q! It will ignore all changes made by user and will quit editor.

4. **Writing selected lines:** It works like save as option in windows to save specific contents of file into another file. Here 5th line will get add into a file called new newfile.txt.

   :5 w newfile.txt

If we you command view newfile.txt we will get following result.

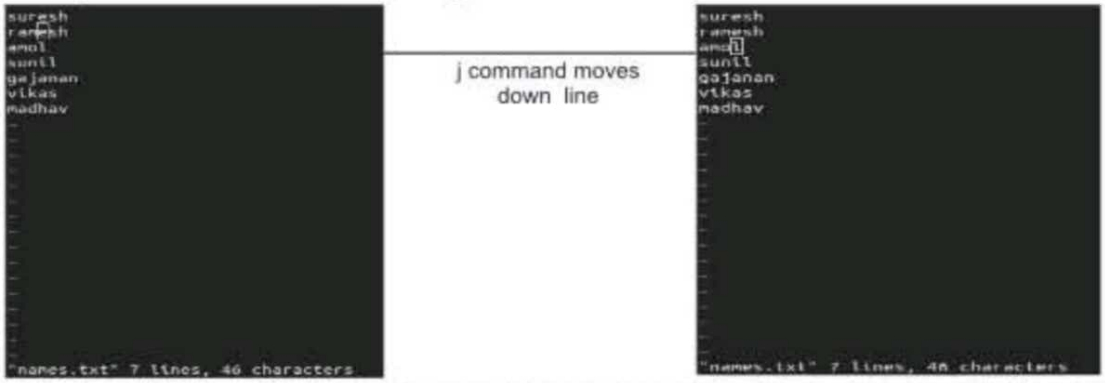
4.10 ILLUSTRATIVE EXAMPLES OF NAVIGATION COMMANDS

- You can enter the following commands in command mode for navigation purpose. You can cancel an incomplete command by pressing [Esc] key.

Sr. No.	Item	Description
1.	Left Arrow or h or Ctrl+H	Move cursor one character left.
2.	Down arrow or j or Ctrl J	Moves cursor down one line.
3.	Up Arrow or k or Ctrl P	It moves cursor one line up
4.	Right Arrow or l	Moves the cursor one character right.

- Suppose we have created a file called namex.txt and we will open this file in vi editor and we will have look at various navigation operations on it.

1. **j -command:** It is used to navigate to down line in vi editor.



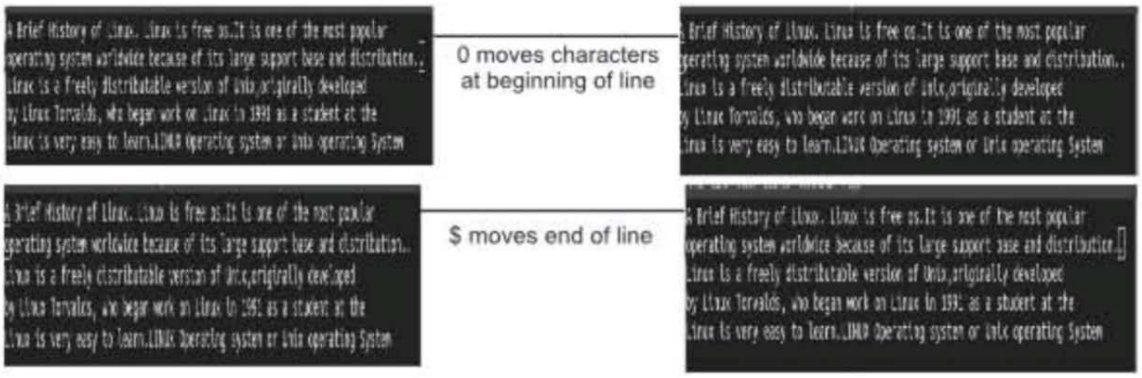
2. **l:** This command in command mode is used to move characters in right direction.

3. **k:** This command move in up direction in vi editor.

4. **h:** This command moves characters in left direction.

Moving within a Line by Character Position:

Item	Description
^	Moves the cursor to the first nonblank character.
0	Moves the cursor to the beginning of the line.
$	Moves the cursor to the end of the line.
fx	Moves the cursor to the next x character.
Fx	Moves the cursor to the last x character.
tx	Moves the cursor to one column before the next x character.
Tx	Moves the cursor to one column after the last x character.
;	Repeats the last f, F, t, or T subcommand.
,	Repeats the last f, F, t, or T subcommand in the opposite direction.
Number|	Moves the cursor to the specified column.



Moving to Words:

Item	Description
w	Moves the cursor to the next small word.
b	Moves the cursor to the previous small word.
e	Moves the cursor to the next end of a small word.
W	Moves the cursor to the next big word.
B	Moves the cursor to the previous big word.
E	Moves the cursor to the next end of a big word.

Moving by Line Position:

Item	Description
H	Moves the cursor to the top line on the screen.
L	Moves the cursor to the last line on the screen.
M	Moves the cursor to the middle line on the screen.
+	Moves the cursor to the next line at its first nonblank character.
–	Moves the cursor to the previous line at its first nonblank character.
Enter	Moves the cursor to the next line at its first nonblank character.

Moving to Sentences, Paragraphs, or Sections

Item	Description
(	Places the cursor at the beginning of the previous sentence, or the previous s-expression if you are in LISP mode.
)	Places the cursor at the beginning of the next sentence, or the next s-expression if you are in LISP mode.
{	Places the cursor at the beginning of the previous paragraph, or at the next list if you are in LISP mode.
}	Places the cursor at the beginning of the next paragraph, at the next section if you are in C mode, or at the next list if you are in LISP mode.
]]	Places the cursor at the next section, or function if you are in LISP mode.
[[	Places the cursor at the previous section, or function if you are in LISP mode.

Moving by Redrawing the Screen:

Item	Description
z	Redraws the screen with the current line at the top of the screen.
z-	Redraws the screen with the current line at the bottom of the screen.
z.	Redraws the screen with the current line at the center of the screen.
/Pattern/z-	Redraws the screen with the line containing the character string, specified by the Pattern parameter, at the

Paging and Scrolling:

Item	Description
Ctrl-U	Scrolls up one-half screen.
Ctrl-D	Scrolls down one-half screen.
Ctrl-F	Scrolls forward one screen.
Ctrl-B	Scrolls backward one screen.
Ctrl-E	Scrolls the window down one line.
Ctrl-Y	Scrolls the window up one line.
z+	Pages up.
z^	Pages down.

Summary

- Shell provides you with an interface to the Unix system. It gathers input from you and executes programs based on that input. There are three types of shells Bash shell, C shell, Korn shell.
- There are various wild card characters used like [], it matches a range (?) it matches a single character , (*) it can match any character for removing special meaning of wild cards can be escaped with \ called escaping. quotes like (' ') single quotes and (" ") double quotes this is called quoting.
- Shell provides three different types of standard files standard input, standard output, standard error.
- 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.
- Grep command is used for searching text in editor with various options. Egrep command is variation of grep.
- Vi editor is the visual editor of linux. It is more powerful and flexible editor used with terminal. There are three types of modes like input mode, command mode, ex mode.
- Command mode is used to enter various commands. Input mode is used to enter the text and Ex mode is the last line mode used for file handling and substitution. There are various ways of starting and quitting vi editor like vi filename will start vi editor and :q, :w, :x :q! , :wq, shift +zz will quit the vi editor.
- We can easily navigate in vi editor to directions using h,j,k,l or we can also move using words by w.
- copying is done by yy and pasting the data is done by p character. Undoing is performed by u or U.
- We can also search for various texts using /textname and ?textname.

Check Your Understanding

1. Which command is used to close the vi editor?
   (a) q (b) wq
   (c) both q and wq (d) none of the Above
2. In vi editor, the key combination CTRL+f _____.
   (a) moves screen down one page (b) moves screen up one page
   (c) moves screen up one line (d) moves screen down one line
3. Which vi editor command copies the current line of the file?
   (a) yy (b) yw
   (c) yc (d) none of the Above
4. Which command searches the string in file opened in vi editor?
   (a) / or? (b) f or F
   (c) t or T (d) none of the Above
5. For navigation purposes, the mode should be _____ mode.
   (a) command (b) input
   (c) insert (d) ex

6. Which of the following keys are used to moving the cursor up and down?
   (a) k (b) h
   (c) l (d) k and j
7. Which of the following keys is used to moving cursor leftwards along a line?
   (a) k (b) h
   (c) l (d) j
8. Which one of the following keys are used to moving cursor rightwards along a line?
   (a) k (b) h
   (c) l (d) j
9. Using 'w' command we can _____
   (a) move back to the beginning of the word
   (b) move forward to the end of the word
   (c) move forward to the beginning of the word
   (d) move back to the end of the word
10. Which key is used for deleting text?
   (a) d (b) y
   (c) k (d) f
11. Which command is used for putting deleted lines or part of lines at a different location?
   (a) p and P (b) x
   (c) dd (d) y
12. Which of the following syntax correct for, Display the 'file' contents page wise.
   (a) cat file|tail (b) cat file|head
   (c) cat file|more (d) None of the above
13. What are the vaid wild cards in shell.
   (a) ? (b) *
   (c) [] (d) {}
14. which of the following is not the mode of vi editor.
   (a) input mode (b) output mode
   (c) ex mode (d) command mode

Answers

1. (c)	2. (a)	3. (a)	4. (a)	5. (a)	6. (d)	7. (b)	8. (c)	9. (c)	10. (d)
			11. (a)	12. (c)	13. (c)	14. (b)			

Practice Questions

Short Answer Questions:

1. What is shell?
2. What is the use of grep command?
3. Explain different wild card characters.
4. What is escape character.
5. What is the use of pipe character?

Long Answer Questions:

1. What is shell? Explain different types of shell?
2. What is Vi Editor? Explain with its different modes?
3. Differentiate between command mode and input mode?
4. Explain what is shell interpretive cycle?
5. List various wild cards used in shell?
6. List various navigation commands in linux.
7. Explain various ways of invoking and quitting vi editor?
8. Explain the following commands:
   (i) grep
   (ii) egrep
9. What is pipe? Explain with example?
10. Write short note on removing the special meaning of wild cards.

■■■

5...

Security and Networking

Objectives...

- To understanding Linux security.
- To study uses of root and Linux Introduction To Users.
- To learn how password is stored and can be used.
- To understand how SSH works.
- To understanding the concept of networking and different protocols.

5.1 INTRODUCTION

- Security should be one of the foremost thoughts at all stages of setting up your Linux computer. To implement a good security policy on a machine, it requires a good knowledge of the fundamentals of Linux as well as some of the applications and protocols that are used.

5.2 UNDERSTANDING LINUX SECURITY

Why do we need security?

- Security refers to providing a protection system to computer system.
- A secure operating system provides security mechanisms that ensure that the system's security goals are enforced despite the threats faced by the system.
- Systems that provide a high degree of assurance in enforcement have been called Secure Systems, or even more frequently "Trusted" Systems. However, it is also true that no system of modern complexity is completely secure.
- Computer system must be protected against unauthorized access, malicious access to system memory, viruses, worms etc
- A security goal defines the operations that can be executed by a system while still preventing unauthorized access.
- Operating system security (OS security) is the process of ensuring OS integrity, confidentiality and availability.

- Security goals describe how the system implements accesses to system resources that satisfy the following:
  1. Secrecy
  2. Integrity
  3. Availability

Security requirements:

- The main security requirements are:
  - **Authorization:** Authorization allows to access for those users, which are related to the system.
  - **Authentication:** Authentication refers to varying each user of the system. Operating system generally authenticate users using username/password, user card, finger print, eye retina pattern, signature.
  - **Privacy/Confidentiality:** Ensure that personal information is not being accessed by unauthorized parties.
  - **Integrity:** Ensuring that the data has not been tampered with.
  - **Non-repudiation:** Confirmation that data is received.
  - **Availability:** Ensure that the system can perform it's required function.

5.2.1 Uses of ROOT

What is the ROOT?

- UNIX system provides a special login name for the exclusive use of the administrator, it is called root.
- It's a password is generally set at the time of installation of the system and has to be used for logging in:

```
Login: root
Password: ********
#_
```

- The prompt of root is #, unlike the $ or % is used by non-privileged users. Once you log in as root, you are placed in root's home directory.
- ROOT is the user name or account that by default has access to all commands and files on a Linux .It is also referred to as the root account, root user and the superuser.
- It is the directory in which all other directories, including their subdirectories, and files reside. The root directory is designated by a forward slash (/).

5.3 LINUX INTRODUCTION TO USERS

- How to identify a system's user account with commands like **who, who am i, etc**.
- If more than one person use a single system, then everyone may have their own user account. Here, it will be helpful to know the user account details.
- It also tells how to create a second user account and run program on that with the help of **su and sudo command.**

Command	Purpose	Syntax
Whoami	It tells you about the system's username.	whoami
Who	The who command gives the information about the users logged on to the system.	who
who am i	This command displays the information about the current user only.	Who am i
W	This command tells about the users who are logged in and what are they doing.	w
Su	• The su command allows you to run a shell as another user. • Any user can acquire superuser status with su command if she knows the root password.	su <username>:
su to root	You can change the user to root when you know the root password.	su root
su as root	The root user can become any existing user without knowing that user's password. Otherwise, password is needed.	su - sssit
su-	If any user name is not mentioned then by default, it will assume root as the target user	su–

5.3.1 pseudo Command

- sudo (**S**uper **U**ser **DO**) command allows a user with proper permissions to execute a command as another user, such as the superuser.
- To Run Program as another User, we use sudo command.
- If you prefix **"sudo"** with any command, it will run that command with elevated privileges or in other words allow a user with proper permissions to execute a command as another user, such as the superuser.
- The option of sudo lets us have multiple administrators. These users who can use the sudo command need to have an entry in the sudoers file located at **"/etc/sudoers"**.
- The sudo command allows a user to start programs with the credentials of another user.
- With sudo command we're allowed to create new users on the system without becoming root or without knowing the root password.
- By default, sudo requires that users authenticate themselves with a password which is the user's password, not the root password itself.

Syntax:

```
sudo -V | -h | -l | -L | -v | -k | -K | -s | [ -H ] [-P ] [-S ] [ -b ] |
[ -p prompt ] [ -c class|- ] [ -a auth_type ] [-r role ] [-t type ]
[ -u username|#uid ] command
```

Options for seudo command:

Option	Description
-V	The -V (version) option causes sudo to print the version number and exit. If the invoking user is already root, the -V option will print out a list of the defaults sudo was compiled with as well as the machine's local network addresses.
-l	The -l (list) option will print out the commands allowed (and forbidden) the user on the current host.
-L	The -L (list defaults) option will list out the parameters that may be set in a Defaults line along with a short description for each. This option is useful in conjunction with grep.
-h	The -h (help) option causes sudo to print a usage message and exit.
-v	If given the -v (validate) option, sudo will update the user's timestamp, prompting for the user's password if necessary. This extends the sudo timeout for another 5 minutes (or whatever the timeout is set to in sudoers) but does not run a command.
-k	The -k (kill) option to sudo invalidates the user's timestamp by setting the time on it to the epoch. The next time sudo is run a password will be required. This option does not require a password and was added to allow a user to revoke sudo permissions from a .logout file.
-K	The -K (sure kill) option to sudo removes the user's timestamp entirely. Likewise, this option does not require a password.
-b	The -b (background) option tells sudo to run the given command in the background. Note that if you use the -b option you cannot use shell job control to manipulate the process.
-p	The -p (prompt) option allows you to override the default password prompt and use a custom one. The following percent ('%') escapes are supported: **%u:** is expanded to the invoking user's login name; **%U:** is expanded to the login name of the user the command will be run as (which defaults to root); **%h:** is expanded to the local hostname without the domain name; **%H:** is expanded to the local hostname including the domain name %% (two consecutive % characters) are collapsed into a single % character.
-c	The -c (class) option causes sudo to run the specified command with resources limited by the specified login class. The class argument can be either a class name as defined in /etc/login.conf, or a single '-' character. Specifying a class of - indicates that the command should be run restricted by the default login capabilities for the user running the command. If the class argument specifies an existing user class, the command must be run as root, or the sudo command must be run from a shell that is already root. This option is only available on systems with BSD login classes where sudo has been configured with the --with-logincap option.

contd. ...

-a	The -a (authentication type) option causes sudo to use the specified authentication type when validating the user, as allowed by /etc/login.conf. The system administrator may specify a list of sudo-specific authentication methods by adding an "auth-sudo" entry in /etc/login.conf. This option is only available on systems that support BSD authentication where sudo has been configured with the --with-bsdauth option.
-u	The -u (user) option causes sudo to run the specified command as a user other than root. To specify a uid instead of a username, use #uid.
-s	The -s (shell) option runs the shell specified by the SHELL environment variable if it is set or the shell as specified in the file passwd.
-H	The -H (HOME) option sets the HOME environment variable to the home directory of the target user (root by default) as specified in passwd. By default, sudo does not modify HOME.
-P	• The -P (preserve group vector) option causes sudo to preserve the user's group vector unaltered. • By default, sudo will initialize the group vector to the list of groups of the target user. • The real and effective group IDs, however, are still set to match the target user.
-r	The -r (role) option causes the new security context to have the role specified by ROLE.
-t	• The -t (type) option causes the new security context to have the type (domain) specified by TYPE. • If no type is specified, the default type is derived from the specified role.
-S	The -S (stdin) option causes sudo to read the password from standard input instead of the terminal device.
--	• The -- flag indicates that sudo should stop processing command line arguments. • It is most useful in conjunction with the -s flag.

5.4 WORKING WITH PASSWORDS

- A secret string used by a user for authentication during login is a password.
- The code is not flashed on the terminal, but is stored in an encrypted manner in /etc/shadow.
- Also features a command with a similar name (passwd) to change the password.
- passwd command is used to change your password.

Syntax: `passwd [options]`

Options for passwd:

Options	Description
-a	Show password attributes for all entries.
-l	Locks password entry for name.
-d	Deletes password for name. The login name will not be prompted for password
-f	Force the user to change password at the next login by expiring the password for name
-g	the password for the named group is changed
-r	It is used with the –g option to remove the current password from the named group

Example: `passwd`
Entering just passwd would allow you to change the password. After entering passwd you will receive the following three prompts:

```
Current Password:******
New Password:**********
Confirm New Password:**********
```

5.4.1 How Passwords are Stored in LINUX?

- Traditionally passwords are stored in the /etc/passwd file in an encrypted format.
- However, because of advances in processor speed, encrypted passwords are now almost universally stored in separate shadow password files.
- Hence passwords are stored in three file:

(a) The /etc/passwd:

1. It is the password file that stores each user account.
2. All fields are separated by a colon (:) symbol. It contains one entry per line for each user listed in /etc/passwd file.

(b) The /etc/shadow:

1. It contain the password information for the user account and optional aging information.
2. The encrypted passwords and other information such as password expiry information (the password aging information).

(c) The /etc/group:

1. It is a text file that defines the groups on the system.
2. There is one entry per line.

- The order is as follows:

1. **Username**: It is your login name.
2. **Password**: It is your encrypted password. The password should be minimum 8-12 characters long including special characters, digits, lower case alphabetic and more.
3. **Last password change (last changed)**: Days since Jan 1, 1970 that password was last changed.

4. **Minimum**: The minimum number of days required between password changes i.e. the number of days left before the user is allowed to change his/her password.
5. **Maximum**: The maximum number of days the password is valid (after that user is forced to change his/her password).
6. **Warn**: The number of days before password is to expire that user is warned that his/her password must be changed.
7. **Inactive**: The number of days after password expires that account is disabled.
8. **Expire**: days since Jan 1, 1970 that account is disabled i.e. an absolute date specifying when the login may no longer be used.

Linux can change password for other user account:

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

```
# passwd
```

OR

```
$ sudopasswd Amit
```

Output:

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

Where,

Amit – is username or account name.

5.5 BYPASSING USER AUTHENTICATION

- While most applications require authentication for gaining access to private information or to execute tasks, not every authentication method is able to provide adequate security.
- Negligence, ignorance, or simple understatement of security threats often result in authentication schemes that can be bypassed by simply skipping the login page and directly calling an internal page that is supposed to be accessed only after authentication has been performed.
- In addition to this, it is often possible to bypass authentication measures by tampering with requests and tricking the application into thinking that we're already authenticated.
- This can be accomplished either by modifying the given URL parameter or by manipulating the form or by counterfeiting sessions.
- Authentication plays a critical role in the security of web applications.
- When a user provides his login name and password to authenticate and prove his identity, the application assigns the user specific privileges to the system, based on the identity established by the supplied credentials.

5.6 UNDERSTANDING SSH: THE SECURE SHELL

- Secure Shell (SSH) is a suite of tools for logging into and executing commands on a remote machine. To allow remote access to the machine is via ssh.
- This service requires that the open ssh-server package be installed.

What is Secure Shell?

- **SSH**, also known as **Secure Shell** or **Secure Socket Shell**, is a network protocol that provides administrators with a secure way to access a remote computer.

- SSH establishes a cryptographically secured connection between two parties(client and server), authenticating each side to the other, and passing commands and output back and forth.
- SSH, the Secure Shell is a powerful, software-based approach to network security that provides a secure channel for data transmission through a network.
- This connection can also be used for terminal access, file transfers, and for tunneling other applications.

5.6.1 Features

- Transmission is secure.
- Transmission can be compressed.
- No login password required.
- Allow you to edit files.
- View the contents of directories.
- Custom based applications.
- Create user accounts.
- Change permissions.
- Anything can be done from command prompt can be done remotely and securely

5.6.2 Components of Secure Shell

1. **SSHD Server:** A program that allows incoming SSH connections to a machine, handling authentication, authorization.
2. **Clients:** A program that connects to SSH servers and makes requests for service
3. **Session:** An ongoing connection between a client and a server. It begins after the client successfully authenticates to a server and ends when the connection terminates.

5.6.3 SSH Architecture

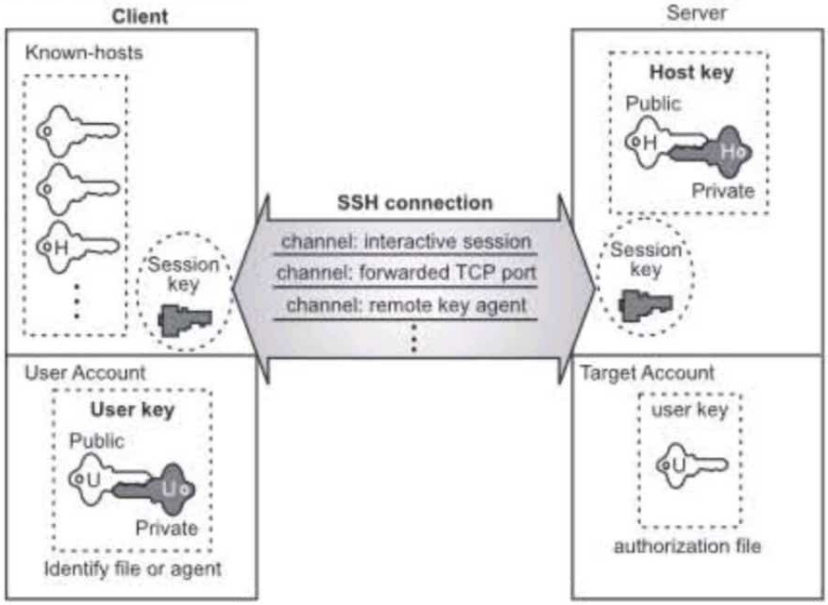


Fig. 5.1: SSH Architecture

How SSH Works?

- SSH protocol uses symmetric encryption, asymmetric encryption and hashing in order to secure transmission of information. The SSH connection between the client and the server happens in three stages:

1. Verification of Server:

- The client initiates a SSH connection with the server. Server listens to default port 22(this port can be changed) for SSH connections. At this point, the server identity is verified.
- There are two cases:
  (i) If the client is accessing the server for first time, client is asked to authenticate server manually by verifying public key of server. Public key of server can be found using **sshkeyscan** command or can be found at different places (WHERE?GOOGLE!). Once the key is verified, the server is added in **known_hosts** file in ~/.ssh directory on client machine. The **known_hosts** file contains the information about all the verified servers by the client.
  (ii) If the client is not accessing the server for the first time, the server's identity is matched with previously recorded information in **known_hosts** file for verification.

2. Generation of Session Key:

(i) After the server is verified, both the parties negotiate a session key using a version of something called the **Diffie-Hellman** algorithm.
(ii) This algorithm is designed in such a way that both the parties contribute equally in generation of session key.
(iii) The generated session key is shared symmetric key i.e. the same key is used for encryption and decryption.

3. Authentication of the client:

(i) The final stage involves authentication of the client.
(ii) Authentication is done using **SSH key pair**.
(iii) As the name suggests, SSH key pair is nothing but a pair of two key to serve two different purposes. One is public key that is used to encrypt data and can be freely shared. The other one is private key that is used to decrypt data and is never shared with anyone.

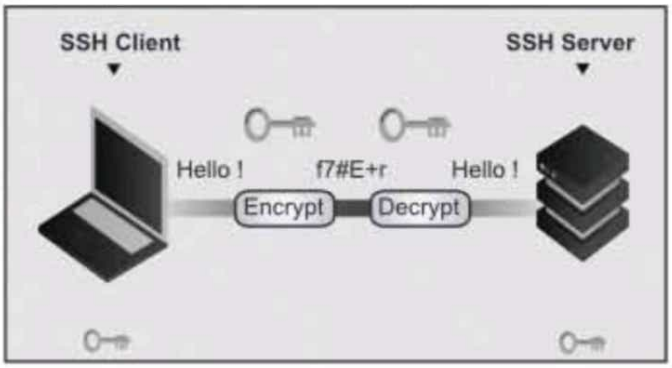


Fig. 5.2: Established symmetric encryption

- After symmetric encryption has been established, the authentication of the client happens as follows:
  1. The client begins by sending an ID for the key pair it would like to authenticate with to the server.
  2. The server checks the authorized_keys file of the account that the client is attempting to log into for the key ID.
  3. If a public key with matching ID is found in the file, the server generates a random number and uses the public key to encrypt the number and sends this encrypted message.
  4. If the client has the correct private key, it will decrypt the message to obtain the random number that was generated by the server.
  5. The client combines the obtained random number with the shared session key and calculates the MD5 hash of this value.
  6. The client then sends this MD5 hash back to the server as an answer to the encrypted number message.
  7. The server uses the same shared session key and the original number that it sent to the client to calculate the MD5 value on its own. It compares its own calculation to the one that the client sent back. If these two values match, it proves that the client was in possession of the private key and the client is authenticated.
- Asymmetry of the keys allows authentication of the client because client can only decrypt the messages if it has the correct associated private key.

5.7 NETWORKING

5.7.1 Basic introduction to Networking

- A network is a transmission system that connects two or more applications running of different computer.
- A network can be defined as a group of computers and other devices connected in some ways so as to be able to exchange data.

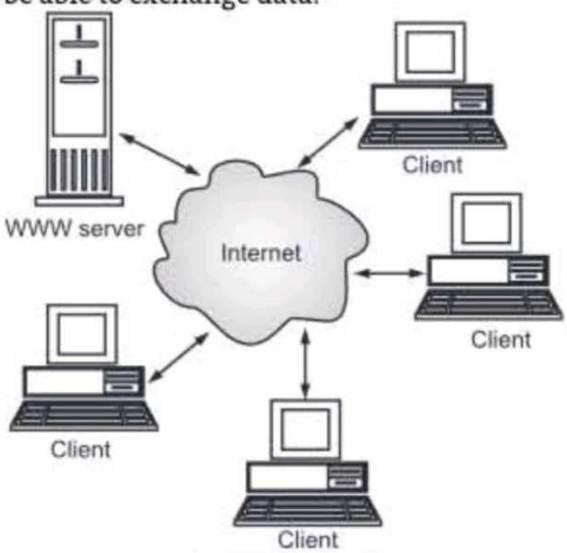


Fig. 5.3: Network

- Each of the devices on the network can be thought of as a node; each node has a unique address.
- Addresses are numeric quantities that are easy for computers to work with, but not for humans to remember.
- Example: 204.160.241.98
- Some networks also provide names that humans can more easily remember than numbers. Example: www.javasoft.com, corresponding to the above numeric address.

5.7.2 Network Protocol

- Protocols mean set of rules.
- Protocol is a set of mutually accepted and implemented rules at both ends of the communications channel for the proper exchange of information.
- It is a formal description of message formats and the rules two or more machines has follow to exchange messages.
- The key elements of a protocol are syntax, semantics and timing:

  Syntax: Syntax refers to the structure or format of the data, meaning the order in which they are presented.

  Semantics: Semantics refers to the meaning of each section of bits.

  Timing: Timing refers to when data should be sent and how fast it can be sent.

5.7.2.1 Different Types of Protocols

1. **TCP:**

- Transmission control protocol is used for communication over a network. In TCP data is broken down into small packets and then sent to the destination. However, IP is making sure packets are transmitted to the right address.

2. **Internet Protocol (IP):**

- IP is also working with TCP. It is an addressing Protocol. IP addresses packets route them and show different nodes and network Unless it reaches its right destination. The IP protocol is developed in 1970.

3. **FTP:**

- FTP is short for File Transfer Protocol. It is an application layer protocol.
- FTP uses TCP to create a virtual connection for control information and then create a separate TCP connection for data transfers.
- The key function of FTP are:
  (i) to remote sharing of files (computer programs and/or data).
  (ii) to encourage indirect or implicit (via program) use of remote computers.
  (iii) to shield a user from variations in file storage systems among hosts.
  (iv) to transfer data reliably and efficiently.

  Purpose:
  - FTP is used for exchanging files over the internet.
  - FTP enables the users to upload and download the files from the internet.

  Working:
  - FTP establishes two TCP connections between the client and the server. One connection is used for transferring data. Other connection is used for transferring control information.

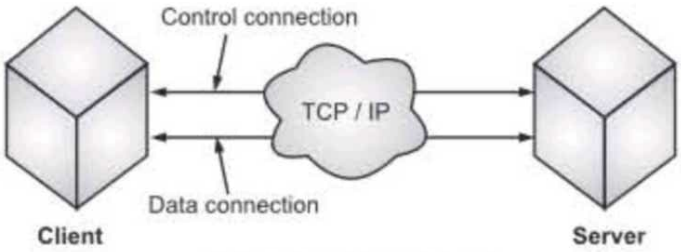


Fig. 5.4: Working of FTP

Some important points about FTP:

- FTP uses TCP at the transport layer.
- FTP uses port number 21 for control connection.
- FTP uses port number 20 for data connection.
- FTP uses persistent TCP connections for control connection.
- FTP uses non-persistent connections for data connection.
- FTP is a connection oriented protocol.
- FTP is an out-of-band protocol as data and control information flow over different connections.
- Emails can't be sent using FTP.
- FTP can transfer one file at a time.

4. **HTTP:**
   - HTTP is short for Hyper Text Transfer Protocol.
   - It is an application layer protocol.
   - HTTP allows an open ended set of methods to be used to indicate the purpose of a request, it builds on the discipline of reference provided by the Uniform Resource Identifier (URI) as location (URL) or name (URN) for indicating the resources on which a method is to be applied.
   - HTTP protocol is the request/response protocol.

   Purpose:
   - It is mainly used for the retrieval of data from websites throughout the internet.
   - It works on the top of TCP/IP suite of protocols.

   Working:
   - HTTP uses a client-server model where- Web browser is the client. Client communicates with the web server hosting the website.

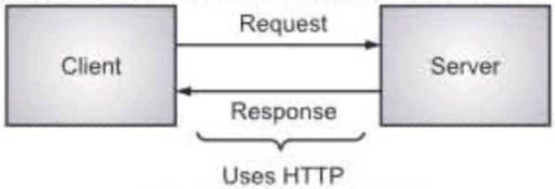


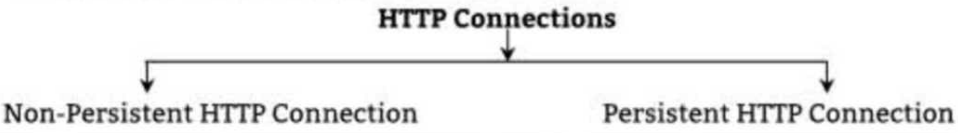
Fig. 5.5: Working of HTTP

- Whenever a client requests some information to the website server, the browser sends a request message to the HTTP server for the requested objects.

- Then:
  - HTTP opens a connection between the client and server through TCP.
  - HTTP sends a request to the server which collects the requested data.
  - HTTP sends the response with the objects back to the client.
  - HTTP closes the connection.

HTTP Connections:

- HTTP connections can be of two types:
  1. Non-persistent HTTP connection
  2. Persistent HTTP connection



Sr. No.	Non-persistent HTTP connection	Persistent HTTP connection
1.	Non-persistent HTTP connection is one that is used for serving exactly one request and sending one response.	Persistent HTTP connection is one that can be used for serving multiple requests.
2.	HTTP server closes the TCP connection automatically after sending a HTTP response.	HTTP server closes the TCP connection only when it is not used for a certain configurable amount of time.
3.	A new separate TCP connection is used for each object.	A single TCP connection is used for sending multiple objects one after the other.
4.	HTTP 1.0 supports non-persistent connections by default.	HTTP 1.1 supports persistent connections by default.

Some Important points about HTTP:

- HTTP uses TCP at the transport layer.
- HTTP uses port number 80.
- HTTP 1.0 is non-persistent and HTTP 1.1 is persistent.
- HTTP 1.0 is a connectionless protocol.
- HTTP is an in-band protocol.
- HTTP is a stateless protocol.

5. Telnet:

- Telnet is an established with some rules which are used to connect to another computer. Telnet is mainly used for the remote login process. The computer which is requesting for a connection that is a local computer and which is accepting the connection that is a remote computer. If you give a command in a local computer that command is executed in the remote computer. Telnet is also based on client and server model.

5.8 IP ADDRESS

- IP address is a unique logical address assigned to a machine over the network. An IP address exhibits the following properties:
  - IP address is the unique address assigned to each host present on Internet.
  - IP address is 32 bits 4bytes long.
  - IP address consists of two components: network component and host component.
  - Each of the 4 bytes is represented by a number from 0 to 255, separated with dots. For example 137.170.4.124

5.8.1 IP Addressing Scheme

- An important part of all networking protocols is the addressing scheme.
- Without being able to locate the individual machines (or hosts as they are called) then it would not be possible for any communication between the hosts.
- There will be more than one addressing scheme in use but the most important of these is the Internet Protocol (referred to as IP), this is significant as it provides the addressing for each end of the connection.
- The other addressing schemes are effectively hidden from the user at layers two or below and are automatically handled by the networking hardware.
- The current version of IP is called IP version 4 but will be replaced by IPV6 in future.
- The addresses used in the Internet Protocol consist of four octets and is 32 bits long. The address is stored in a format known as dotted decimal.

  Syntax: xxx.xxx.xxx.xxx, where xxx is a number between 0 and 255.

  Example: 192.168.3.27
- Users refer to the computer using it's host name.
- The IP address is obtained from the host name using the "Domain Name System" (DNS).
- There is no actual relationship between the hostname and the IP address instead this uses a lookup table.
- There are 5 different classes however only 3 are commonly used.
  1. **Class A:** These are for large organizations. The network portion is 8 bits long and begins with binary 0. There are 126 possible networks each with up to 16.7 million hosts.
  2. **Class B:** These are for medium sized organizations. The network portion is 16 bits long and starts with binary 10. There are 16 thousand networks each with up to 65 thousand hosts. In reality the definition of a medium sized organisation would be a very large company.
  3. **Class C:** These are for smaller organizations. The network portion is 24 bits long and begins with binary 110. There are 200 thousand possible networks each with up to 254 hosts
  4. **Class D:** These are allocated for multicast although is rarely used. The addresses begin with binary 1110.
  5. **Class E:** These are experimental. The addresses begin with binary 1111.

IP Address class ranges

Class Type	Range	Starts with bit string
A	0 to 127	0
B	128 to 191	10
C	192 to 223	110
D	224 to 239	1110
E	240 to 255	11110

5.9 DNS

- DNS is short for Domain Name Service or Domain Name System.
- A client-server application that maps host names into their corresponding IP addresses is known as DNS.
- Mapping host names into their corresponding IP addresses is called name resolution or name translation or name mapping or Address Resolution.
- Domain Name System helps to resolve the host name to an address. It uses a hierarchical naming scheme and distributed database of IP addresses and associated names

Why we need to use names instead of IP numbers?

- IP addresses are difficult to remember.
- IP addresses can change.

Purpose:

- DNS is a host name to IP Address translation service.

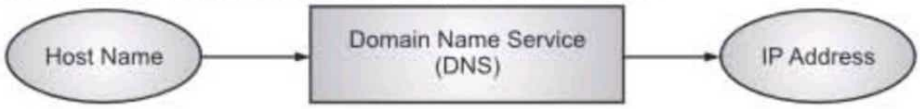


Fig. 5.6: Domain Name Service Translation

Need:

- The need for Domain Name Service arises due to the following reasons-

Point-01:

- IP Addresses are not static and may change dynamically.
- So, a mapping is required which maps the domain names to the IP Addresses of their web servers.

Point-02:

- IP Addresses are a complex series of numbers.
- So, it is difficult to remember IP Addresses directly while it is easy to remember names.

Process of DNS Resolution:

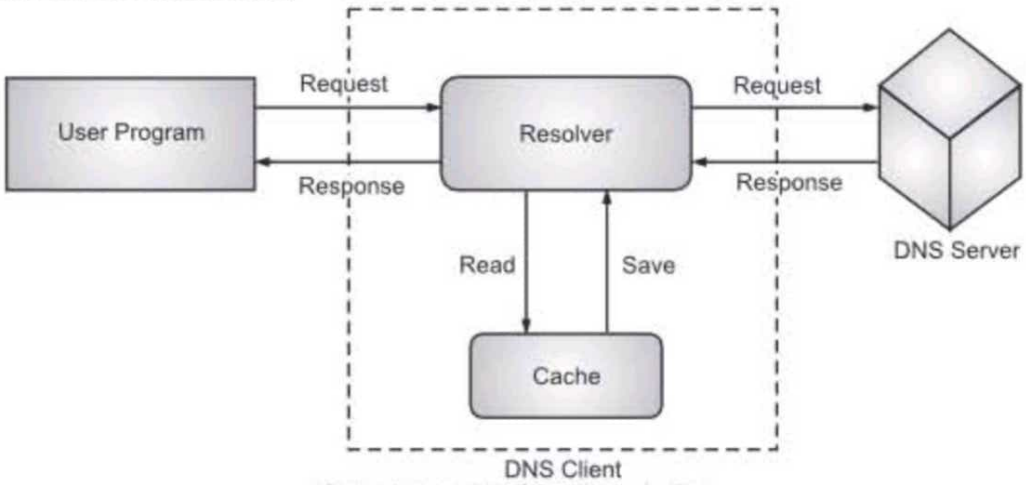


Fig. 5.7: Process of DNS Resolution

- The steps involved in DNS Resolution are:

  Step-01:
  - A user program sends a name query to a library procedure called the resolver.

  Step-02:
  - Resolver looks up the local domain name cache for a match.
  - If a match is found, it sends the corresponding IP Address back.
  - If no match is found, it sends a query to the local DNS server.

  Step-03:
  - DNS server looks up the name.
  - If a match is found, it returns the corresponding IP Address to the resolver.
  - If no match is found, the local DNS server sends a query to a higher level DNS server.
  - This process is continued until a result is returned.

  Step-04:
  - After receiving a response, the DNS client returns the resolution result to the application.

Features of Domain Name System (DNS):

- Domain Name System (DNS) provides name resolution services to the client computers.
- Domain Name System (DNS) provides IP address to hostname mapping.
- Linux DNS Servers can operate in three modes and these modes are, Caching only DNS server, Primary DNS server and Slave (secondary) DNS server.
- Replication of DNS database between servers

Summary

- Operating system security (OS security) is the process of ensuring OS integrity, confidentiality and availability.
- UNIX system provides a special login name for the exclusive use of the administrator, it is called root.
- Sudo (Super User DO) command allows a user with proper permissions to execute a command as another user, such as the superuser.

- A secret string used by a user for authentication during login is a password. Traditionally passwords are stored in the /etc/passwd file in an encrypted format. But now almost universally stored in separate shadow password files.
- SSH, also known as Secure Shell or Secure Socket Shell, is a network protocol that provides administrators with a secure way to access a remote computer.
- Protocol is a set of mutually accepted and implemented rules at both ends of the communications channel for the proper exchange of information.
- DNS(Domain Name Sever) converts the names our web browser address to the IP address of web servers hosting those sites.

Check Your Understanding

1. Which of the following comes under secured Linux based OS?
   (a) Ubuntu (b) Fedora
   (c) Kubuntu (d) Tails
2. Using the _____ account of a UNIX system, one can carry out administrative functions.
   (a) root (b) administrative
   (c) user (d) client
3. What is the login name of system administrator?
   (a) root (b) su
   (c) master (d) admin
4. What is the prompt for system administrator?
   (a) $ (b) %
   (c) # (d) &
5. When we log in as root user we are placed in ____
   (a) /bin (b) /root
   (c) /system (d) /admin
6. Which command is used for acquiring superuser status?
   (a) pu (b) su
   (c) admn (d) super
7. Which of the following is used for creating user's environment?
   (a) su (b) su –
   (c) -su (d) su – –
8. _____ command is used by the superuser for changing root's password.
   (a) pd (b) password
   (c) pswd (d) pwd
9. All the user information is stored in _____
   (a) etc/passwd (b) bin/passwd
   (c) bin/users/passwd (d) etc/shadow
10. Which of the following is not a field stored in /etc/passwd?
    (a) username (b) password
    (c) UID, GID (d) encrypted password
11. A DNS client is called _____
    (a) DNS updater (b) DNS resolver
    (c) DNS handler (d) none of the mentioned
12. DNS database contains _____
    (a) name server records (b) hostname-to-address records
    (c) hostname aliases (d) all of the mentioned

13. Which of the following devices translate hostnames into IP address?
    (a) DNS server (b) HUB
    (c) HTTP (d) Firewall
14. Expansion of FTP is
    (a) Fine Transfer Protocol (b) File Transfer Protocol
    (c) First Transfer Protocol (d) None of the mentioned
15. FTP is built on _____ architecture
    (a) Client-server (b) P2P
    (c) Both of the mentioned (d) None of the mentioned
16. FTP uses _____ parallel TCP connections to transfer a file
    (a) 1 (b) 2
    (c) 3 (d) 4
17. HTTP allows which response?
    (a) Multiplexing (b) Serial
    (c) Coherent (d) Binary
18. HTTP expands?
    (a) HyperText Transfer Protocol (b) HyperTerminal Transfer Protocol
    (c) HyperText Terminal Protocol (d) HyperTerminal Text Protocol

Answers

1. (d)	2. (a)	3. (a)	4. (c)	5. (b)	6. (b)	7. (b)	8. (c)	9. (a)	10. (d)
	11. (b)	12. (d)	13. (a)	14. (b)	15. (a)	16. (b)	17. (a)	18. (a)	

Practice Questions

Short Answer Questions:

1. Why do we need security?
2. What is root?
3. How passwords are stored in LINUX?
4. What are the Characteristics of FTP?
5. What is protocol?
6. What is network security?
7. Define the used of sudo command.

Long Answer Questions:

1. Explain in brief how SSH works?
2. Explain network protocol with its different types.
3. Explain the working of HTTP protocol.
4. Explain IP addressing scheme.
5. What is DNS? Explain needs of DNS in brief.
6. Write short note on sudo command.
7. Write short note on bypassing user authentication.
8. Explain working with passwords.
9. Short note on sudo command.
10. Write short note on HTTP protocol.
11. Write short note on FTP protocol.

■■■

6... Shell Scripts

Objectives...

- To learn how to use and script the Bash shell programming.
- To study master core Bash language constructs such as variables, loops, conditionals, and functions.
- To learn how the shell relates to the keyboard, the screen, the operating system, and users' programs.
- To understand when and why command-line interfaces should be used instead of graphical interfaces.

6.1 INTRODUCTION

- In this chapter we will look at one of the major component of LINUX architecture- The Shell.
- A shell is a program that interprets commands and acts as an interface to the LINUX operating system.
- A LINUX shell is command interprets a user command which are directly entered by user or which can be read from a file called as shell programming or shell scripting.
- Shell scripts are interpreted by the operating system and not complied.
- There are other Scripting Language like PHP, Python, Perl, Ruby etc, which provides more functions and helps you to achieve the result easily.

Advantages of Shell Scripts:

1. The command and syntax are exactly the same as those directly entered in command line, so programmer do not need to switch to entirely different syntax.
2. Writing shell scripts are much quicker.
3. Quick start.
4. Interactive debugging etc.

Disadvantages of Shell Scripts:

1. Prone to costly errors, a single mistake can change the command which might be harmful.
2. Slow execution speed.
3. Design flaws within the language syntax or implementation.
4. Not well suited for large and complex task.
5. Provide minimal data structure unlike other scripting languages etc.

6.2 SHELL PROGRAMMING

- All shell statements and LINUX commands can be entered in the command line itself. However, when a group of command has to execute regularly, it is better to store in a file. All such files are called shell scripts, shell program, or shell procedures. The extension is .sh is used for shell program.
- 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
- Before we run the script, it is essential to make the script executable first. After that invoke the script name to run the script. For making the script executable, we have to use chmod +x script_name. chmod +x.
- "#!" 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.

Example:

```
#first
ls
who
pwd
```

- The first line begins with '#' . This is the symbol, which marks the beginning of a comment. First is name of the file in which shell script is type.
- The shell is told first to give list of all files in the current directory.
- Then print list of all users who have logged in and finally print the working directory.
- To execute the shell script we type first at the shell prompt and press the enter key.

6.3 VARIABLES

1. Shell variables are an internal part of shell programming.
2. They provide the ability to store and manipulate information within a shell program.
3. The variables you use are completely under your control.
4. Creation and destroy of any number of variables as needed to solve the problem.

6.3.1 Variable Types

- Two main types of variables are present:

(a) System Variable/Environment Variables:

1. These are the variables which are created and maintained by Operating System(Linux) itself. Generally these variables are defined in CAPITAL LETTERS.

2. These variables are dynamic values which affect the processes or programs on a computer. They exist in every operating system, but types may vary.
3. Environment variables that store information within the shell can be expanded using the dollar sign ($) metacharacter.
4. Environment variables can be created, edited, saved, and deleted and give information about the system behavior.
5. Environment variables can change the way a software/programs behave.

- Here is a list of common environment variables in Linux:
  - **USER**: Your current username.
  - **SHELL**: The path to the current command shell (for example, /bin/bash).
  - **PWD**: The current working directory.
  - **HOSTNAME**: The hostname of the computer.
  - **HOME**: Your home directory.
  - **MAIL**: The location of the user's mail spool. Usually /var/spool/mail/USER.
  - **LANG**: Your current language.
  - **TZ**: Your time zone.
  - **PS1**: The default prompt in bash.
  - **TERM**: The current terminal type (for example, xterm).
  - **DISPLAY**: The display used by X. This variable is usually set to: 0.0, which means the first display on the current computer.
  - **HISTFILESIZE**: The maximum number of lines contained in the history file.
  - **EDITOR**: The user's preferred text editor.
  - **MANPATH**: The list of directories to search for manual pages.
  - **OSTYPE**: The type of operating system.

```
#!/bin/sh
echo "$PATH"
echo "$HOME"
echo "$BASH_VERSION"
echo "$HOSTNAME"
```

(b) User Defined Variables/Ordinary Variables:

1. These variables are defined by users. A shell script allows us to set and use our own variables within the script. Setting variables allows you to temporarily store data and use it throughout the script, making the shell script more like a real computer program.
2. User variables can be any text string of up to 20 letters, digits, or an underscore character. User variables are case sensitive, so the variable var1 is different from the variable var1. This little rule often gets novice script programmers in trouble.
3. Values are assigned to user variables using an equal sign. No spaces can appear between the variable, the equal sign, and the value. Here are a few examples of assigning values to user variables:

6.3.2 Defining Variables

- Variables are defined as follows:

```
variable_name=variable_value
```

For example:

```
#!/bin/sh
NAME="Archana"
echo $NAME
```

Output:

```
Archana
```

6.3.3 Deleting Variables

- The following syntax can be used to remove a variable from the system.

```
Unset variable_name
```

Command	Description
`echo $VARIABLE`	To display value of a variable.
`Env`	Displays all environment variables.
`VARIABLE_NAME= variable_value`	Create a new variable.
`unset`	Remove a variable.
`export Variable=value`	To set value of an environment variable.

6.3.4 Reading Variables From Standard Input

- The read command reads one line of input from the terminal and assigns it to variables give as arguments.

  Syntax: `read var1 var2 var3...`

  - **Action:** reads a line of input from standard input.
  - Assign first word to var1, second word to var2,...
  - The last variable gets any excess words on the line.

6.4 WHAT IS .PROFILE FILE READ ONLY COMMANDS?

- We can put definitions of commonly used functions in the .profile file so that they are available whenever you login. Or you can store these functions in a file and execute that file when required in the current shell.
- A profile file is a start-up file of an LINUX user, like the autoexec.bat file of DOS. When a LINUX user tries to login to his account, the operating system executes a lot of system files to set up the user account before returning the prompt to the user.
- In addition to the system settings, the user might wish to have some specific settings for his own account. To achieve this in LINUX, at the end of the login process, the operating system executes a file at the user level, if present. This file is called profile file.

- The name of the profile file varies depending on the default shell of the user. The profile file, if present, should always be in the home directory of the user. The following are the profile files of the commonly used shells:

Shell	Profile File
Ksh	.profile
Bourne	.profile
Bash	.bash_profile
Tcsh	.login
Csh	.login

- The specific settings which an Linux user usually does is:
  - Setting of any environment variable.
  - Setting of any alias.(Though it is always recommended to keep the aliases in a separate file).
  - Setting of PATH variable or any other path variables.
- Any changes made to the profile file will reflect by either of the following ways:
  1. Login to the user account again.
  2. Source the profile file. Sourcing the profile file means running the profile file at the command prompt.
- The command to run the profile file, say .profile is:

```
. $HOME/.profile
```

- In case of sourcing the profile file in tcsh or csh, the command is:

```
source $HOME/.login
```

- .profile file controls the following by default:
  1. Shells to open.
  2. Prompt appearance.
  3. Keyboard Sound.
  4. The .profile file contains your individual profile that overrides the variables set in the /etc/profile file.

6.5 COMMAND LINE ARGUMENTS

- A Command line Arguments are passed after the name of a program in command line operating systems like DOS or Linux and are passed into the program from the operating system.
- Shell scripts also accept command line arguments.
- Command line arguments are useful for passing input to script at runtime which has its own advantage. This article will help you to pass command line arguments in a shell script.
- To pass a command line argument we can simply write them after script name separated with space. All command line parameters can be access by their position number using $.

Example: `sh myScript.sh 12 Archana tecadmin.net`

- **sh:** Linux shell.
- **myScript.sh:** Linux shell script.
- **12:** First command line parameter accessible by $1.
- **Archana:** Second command line parameter accessible by $2.
- **tecadmin.net:** Third command line parameter accessible by $3.

6.5.1 Access Command Line Argument with Position Number

- So as above example command line parameters are accessible at $1, $2, $3.... $9, $10.... $100 etc. Maximum length of command line parameters are not defined by shell but it's defined by operating system and measured in Kilobytes.
  - **$*:** Store all command line arguments
  - **$@:** Store all command line arguments
  - **$#:** Store count of command line arguments
  - **$0:** Store name of script itself
  - **$1 :** Store first command line argument
  - **$2:** Store second command line argument
  - **$3:** Store third command line argument
  - **$9:** Store 9'th command line argument
  - **$10:** Store 10'th command line argument
  - ...
  - ...
  - **$99:** Store 99'th command line argument

Program 1: Create a shell script to Print all Argument with script name and total number of arguments passed. Create script file **myfirstscript.sh** using following content.

```
# myfirstscript.sh
#!/bin/bashscho Script Name:'$0"sdfknlgf
echo Script Name: "$0"
echo Total Number of Argument Passed: "$#"
echo Arguments List -
echo 1. $1
echo 2. $2
echo 3. $3
echo All Arguments are: "$*"
```

Output:

```
12 Archana tecadmin.net
Script Name: myfirstscript.sh
Total Number of Argument Passed: 3
Arguments List -
1. 12
2. Archana
3. tecadmin.net
All Arguments are: 12 Archana tecadmin.net
```

6.5.2 Exit and Exit Status of Command

(a) Exit:

- It is a command in Linux used to exit the shell where it is currently running. It takes one more parameters as [N]and exits the shell with a return of status N.
- To end a shell script and set its exit status, use the exit command. The exit command terminates a script, just as in a C program.

  Syntax: `exit[n]`

(b) Exit status:

- The $? Variable is used to represent the exit status of the previous command.
- Exit status is an integer number returned by every command upon its completion. As a value most commands return and exit status of 0,if they were successful, and 1 if they were unsuccessful.
- Each Linux command returns a status when it terminates normally or abnormally. You can use value of exit status in the shell script to display an error message or take some sort of action.
- A non-zero (1-255 values) exit status means command was failure.

```
echo$?
date # run date command
echo$?# print exit status
```

6.5.3 Test Command

- The test command checks file types and compares values.
- Test is a built-in command of the Bash shell that can test file attributes, and perform string and arithmetic comparisons.

  Syntax: `test EXPRESSION`
- It provides no output, but returns an exit status of 0 for "true" (test successful) and 1 for "false" (test failed).
- The test command is frequently used as part of a conditional expression. For instance, the following statement says, "If 4 is greater than 5, output yes, otherwise output no."
- UNIX provides a shorthand for test i.e. [ ]. We can use this pair of rectangular brackets enclosing the expression. Thus the following two are equal,

```
test $x -eq $y
[ $x -eq $y ]
```

Example:

1. test 100 -gt 99 && echo "Yes, that's true." || echo "No, that's false."

   This command will print the text "Yes, that's true." because 100 is greater than 99.
2. Test 100 -lt 99 && echo "Yes." || echo "No."

   This command will print the text "No." because 100 is not less than 99.

6.6 LOGICAL OPERATORS FOR CONDITIONAL EXECUTION

- There are various operators supported by shell:
  - Arithmetic Operators
  - Relational Operators
  - Boolean Operators
  - String Operators
  - File Test Operators

1. Arithmetic Operators:

- These operators are used to perform normal arithmetic's /mathematical operations. There are 7 arithmetic operators:
  - **Addition (+)**: Binary operation used to add two operands.
  - **Subtraction (-)**: Binary operation used to subtract two operands.
  - **Multiplication (*)**: Binary operation used to multiply two operands.
  - **Division (/)**: Binary operation used to divide two operands.
  - **Modulus (%)**: Binary operation used to find remainder of two operands.
  - **Increment Operator (++)**: Unary operator used to increase the value of operand by one.
  - **Decrement Operator (–)**: Unary operator used to decrease the value of a operand by one.

2. Relational Operators:

- Relational operators are those operators which defines the relation between two operands. They give either true or false depending upon the relation. They are of 6 types:
  - **Equal to:** -eq
  - **Not equal to:** -ne
  - **Greater than:** -gt
  - **Less than:** -lt
  - **Greater than or equal to:** -ge
  - **Less than or equal to:** -le
- Assume variable a holds 10 and variable b holds 20 then:

Operator	Description	Example
-eq	Checks if the value of two operands are equal or not; if yes, then the condition becomes true.	[$a -eq $b] is not true.
-ne	Checks if the value of two operands are equal or not; if values are not equal, then the condition becomes true.	[$a -ne $b] is true.

contd. ...

-gt	Checks if the value of left operand is greater than the value of right operand; if yes, then the condition becomes true.	[$a -gt $b] is not true.
-lt	Checks if the value of left operand is less than the value of right operand; if yes, then the condition becomes true.	[$a -lt $b] is true.
-ge	Checks if the value of left operand is greater than or equal to the value of right operand; if yes, then the condition becomes true.	[$a -ge $b] is not true.
-le	Checks if the value of left operand is less than or equal to the value of right operand; if yes, then the condition becomes true.	[$a -le $b] is true.

3. Logical Operators:

- They are also known as Boolean operators. These are used to perform logical operations. We use the logical operators to test more than one condition.
- They are of 3 types:
  - -a (read as AND)
  - -o (read as OR)
  - !(read as NOT)
  - Assume variable a holds 10 and variable b holds 20 then:

Operator	Description	Example
!	This is logical negation. This inverts a true condition into false and vice versa.	[! false] is true.
-o	This is logical **OR**. If one of the operands is true, then the condition becomes true.	[$a -lt 20 -o $b -gt 100] is true.
-a	This is logical **AND**. If both the operands are true, then the condition becomes true otherwise false.	[$a -lt 20 -a $b -gt 100] is false.

Logical AND (&&):

- This is a binary operator, which returns true if both the operands are true otherwise returns false.
- Logical AND in bash script is used with operator -a.

Program 2: We will check if the number is even and greater than 10.

```
#!/bin/sh
# take a number from the user
echo "Enter a number: "
read a
```

```
# check
if [ `expr $a % 2` == 0 -a $a -gt 10 ]
then
echo "$a is even and greater than 10."
else
echo "$a failed the test."
fi
```

Output:

```
Enter a number:
10
10 failed the test.
Enter a number:
20
20 is even and greater than 10.
```

- **Logical OR (||)**: This is a binary operator, which returns true is either of the operand is true or both the operands are true and returns false if none of them is false. Logical OR in bash script is used with operator -o.

Program 3: We will check if entered number is either odd or less than 10.

```
#!/bin/sh
# take a number from the user
echo "Enter a number: "
read a
# check
if [ `expr $a % 2` != 0 -o $a -lt 10 ]
then
echo "$a is either odd or less than 10."
else
echo "$a failed the test."
fi
```

Output:

```
Enter a number:
10
10 failed the test.

Enter a number:
9
9 is either odd or less than 10.
```

Equal to(==):

- We use the = equal operator to check if two strings are equal.

Program 4: Example we will check if entered strings are equal.

```
#!/bin/sh
# take two strings from user
echo "Enter first string:"
read str1
echo "Enter second string:"
read str2
# check
if [ "$str1" = "$str2" ]
then
echo "Strings are equal."
else
echo "Strings are not equal."
fi
```

Output:

```
Enter first string:
Hello
Enter second string:
hello
Strings are not equal.
```

```
Enter first string:
Hello
Enter second string:
Hello
Strings are equal.
```

Not Equal to (!):

- This is a unary operator which returns true if the operand is false and returns false if the operand is true.

Program 5: We will check if entered string is not equal to "Hello World".

```
#!/bin/sh
# take a string from user
echo "Enter string:"
readstr
s="Hello World"
# check
if [ "$str" != "$s" ]
then
echo "$str not equal to $s."
else
echo "$str equal to $s."
fi
```

Output:

```
Enter string:
Hello
Hello not equal to Hello World.

Enter string:
Hello World
Hello World equal to Hello World.
```

4. String operators:

- Assume variable a holds "pqr" and variable b holds "efg" then:
- **Examples:** Assume variable a holds 10 and variable b holds 20 then:

Operator	Description	Example
=	Checks if the value of two operands are equal or not; if yes, then the condition becomes true.	[$a = $b] is not true.
!=	Checks if the value of two operands are equal or not; if values are not equal then the condition becomes true.	[$a != $b] is true.
-z	Checks if the given string operand size is zero; if it is zero length, then it returns true.	[-z $a] is not true.
-n	Checks if the given string operand size is non-zero; if it is nonzero length, then it returns true.	[-n $a] is not false.
Str	Checks if **str** is not the empty string; if it is empty, then it returns false.	[$a] is not false.

-z to check size zero:

- We use -z to check if the size of the string is zero.

Program 6: we will check if the size of the entered string is zero.

```
#!/bin/sh
# take a string from user
echo "Enter string:"
readstr
# check
if [ -z "$str" ]
then
echo "String size equal to 0."
else
echo "String size not equal to 0."
fi
```

Output:

```
Enter string:
Hello
String size equal to 5.
```

-n to check size not zero:

- We use -n to check if the size of the string is not zero.

Program 7: We will check if the size of the entered string is not zero.

```
#!/bin/sh
# take a string from user
echo "Enter string:"
readstr
# check
if [ -n "$str" ]
then
echo "String size not equal to 0."
else
echo "String size equal to 0."
fi
```

Output:

```
Enter string:
Hello
String size not equal to 0.

Enter string:
String size equal to 0.
```

- ${#str} to find length of the string

Program 8: We will print the length of the entered string.

```
#!/bin/sh
# take a string from user
echo "Enter string:"
readstr
echo "Length of the entered string = ${#str}"
```

Output:

```
Enter string:
Length of the entered string = 0

Enter string:
Hello
Length of the entered string = 5
```

5. File Test Operator:

- These operators are used to test a particular property of a file.
  - **-b operator**: This operator check whether a file is a block special file or not. It returns true, if the file is a block special file otherwise false.
  - **-c operator**: This operator checks whether a file is a character special file or not. It returns true if it is a character special file otherwise false.

- **-d operator**: This operator checks if the given directory exists or not. If it exits then operators returns true otherwise false.
- **-e operator**: This operator checks whether the given file exits or not. If it exits this operator returns true otherwise false.
- **-r operator**: This operator checks whether the given file has read access or not. If it has read access then it returns true otherwise false.
- **-w operator**: This operator check whether the given file has write access or not. If it has write then it returns true otherwise false.
- **-x operator**: This operator check whether the given file has execute access or not. If it has execute access then it returns true otherwise false.
- **-s operator**: This operator checks the size of the given file. If the size of given file is greater than 0 then it returns true otherwise it false.

6.7 CONDITIONAL AND LOOPING STATEMENTS

- We use the If statement to make decisions whether to execute a block of code or not based on some condition.

6.7.1 Conditional Statements

1. **if statement:**

Syntax:

```
if [ condition ]
then
 # if block code
fi
```

Where, condition is some condition which if evaluates to true then the if block code is executed otherwise, it is ignored.

The fi marks the end of the if statement.

Program 9: Write a Shell Script to check if two numbers are equal.

```
#!/bin/sh
# take two numbers from the user
echo "Enter two numbers: "
read a b
# check
if [ $a == $b ]
then
echo "Numbers are equal."
fi
echo "End of script."
```

Output:

```
Enter two numbers:
10 20
End of script.

Enter two numbers:
10 10
Numbers are equal.
End of script.
```

---

2. if else statement:

Syntax:

```
if [ condition ]
then
 # if block code
else
 # else block code
fi
```

Where, condition is some condition which if evaluates to true then the if block code is executed otherwise, else block code is executed.

Program 10: Write a Shell Script to check if two numbers are equal or not using if else statement.

```
#!/bin/sh
# take two numbers from the user
echo "Enter two numbers: "
read a b
# check
if [ $a == $b ]
then
echo "Numbers are equal."
else
echo "Numbers are not equal."
fi
echo "End of script."
```

Output:

```
Enter two numbers:
10 20
Numbers are not equal.
End of script.
```

---

3. ifelif else statement

Syntax:

```
if [ condition ]
then
 # if block code
elif [ condition2 ]
then
 # elif block code
else
 # else block code
fi
```

Where, condition is some condition which if evaluates to true then the if block code is executed. If it is false then, we check the elif condition. If that too is false then we execute the else block if it is present.

Program 11: Write a Shell Script to check if a number is odd, even or zero.

```
#!/bin/sh
# take a numbers from the user
echo "Enter a number: "
read a
# check
if [ $a == 0 ]
then
echo "It's zero."
elif [ `expr $a % 2` == 0 ]
then
echo "It's even."
else
echo "It's odd."
fi
echo "End of script."
```

Output:

```
Enter a number:
0
It's zero.
End of script.
Enter a number:
10
It's even.
End of script.
```

4. **Case conditional statement:**

Similar to the if statement we use the case statement to make decisions and execute a block of code based on some match.

Syntax:

```
case word in
 pattern1)
 # block of code for pattern1
 ;;
 pattern2)
 # block of code for pattern2
 ;;
 *)
 # default block
 ;;
esac
```

- Where, word is some value that is matched with the pattern1, pattern2 and so on.
- If the word matches any pattern then the block of code belonging to that pattern is executed.
- The ;; marks the end of the block and takes us out of the case.
- The *) represents the default pattern. If no match is found then the code in the default pattern is executed.
- The default pattern *) is optional and can be omitted.
- The esac (reverse of case) marks the end of the case statement.

Program 12: Write a Shell Script to take integer value from user and print in its word form.

```
#!/bin/sh
# take a number from user
echo "Enter number:"
readnum
case $num in
 1)
echo "It's one!"
 ;;
 2)
echo "It's two!"
 ;;
 3)
echo "It's three!"
 ;;
 *)
echo "It's something else!"
 ;;
esac
echo "End of script."
```

Output:

```
$ sh num.sh
Enter number:
2
It's two!
End of script.
```

Program 13: Write a Shell Script to take user name and time of the day as input and display some greetings message.

```
#!/bin/sh
# take user name
echo "Enter your name:"
read name
```

```
# take time of the day
echo "Enter time of the day [Morning/Afternoon/Evening/Night]:"
read daytime
case $daytime in
 "Morning")
echo "Good Morning $name"
 ;;
 "Afternoon")
echo "Good Afternoon $name"
 ;;
 "Evening")
echo "Good Evening $name"
 ;;
 "Night")
echo "Good Night $name"
 ;;
esac
echo "End of script."
```

Output:

```
Enter your name:
ANISH
Enter time of the day [Morning/Afternoon/Evening/Night]:
Morning
Good Morning ANISH
End of script.
```

---

5. printf:

The printf command is used to print pre-formatted output. And it is similar to the printf() function of C programming language.

We can say that printf is a successor of echo command.

Syntax:

```
printf<format><arguments>
```

Where, format is the format string used on the arguments.

format is optional and can be omitted.

Program 14: To print name and age entered by the user.

```
#!/bin/sh
# take user name
printf "Enter name: "
read name
# take user age
printf "Enter age: "
```

```
read age
# display result
echo "---Result---"
printf "Name: %s\n" "$name"
printf "Age: %d\n" "$age"
```

Output:

```
Enter name: NIKITA
Enter age: 24
---Result---
Name: NIKITA
Score: 24
```

Format:

- Following are the format we can use for the arguments in printf command.

Format	Description
%d	For signed decimal numbers
%i	For signed decimal numbers
%u	For unsigned decimal numbers
%o	For unsigned octal numbers
%x	For unsigned hexadecimal numbers with lower case letters (a-f)
%X	For unsigned hexadecimal numbers with upper case letters (A-F)
%f	For floating point numbers
%s	For string
%%	For percent % symbol

Modifiers:

- We use modifiers with format to modify output.

Format	Description
N	This specifies the width of the field for output.
*	This is the placeholder for the width.
-	To left align output in the field. (Default: Right align)
0	Pad result with leading 0s.
+	To put + sign before positive numbers and - sign for negative numbers.

Program 15: To print 100 in decimal, octal and hexadecimal form.

```
#!/bin/sh
num=100
# output in decimal, octal, hex form
printf "Decimal: %d\n" "$num"
printf "Octal: %o\n" "$num"
printf "Hex: %X\n" "$num"
```

Output:

```
$ sh example.sh
Decimal: 100
Octal: 144
Hex: 64
```

Program 16: If we want to add leading 0 for octal numbers and 0x for hex then we will add # as shown below.

```
#!/bin/sh
num=100
printf "Decimal: %d\n" "$num"
printf "Octal: %#o\n" "$num"
printf "Hex: %#X\n" "$num"
```

Output:

```
$ sh example1.sh
Decimal: 100
Octal: 0144
Hex: 0X64
```

6.7.2 Looping Statements

- Loop is used to execute a block of code multiple times.

Brace expansion:

- We use the brace expansion {m..n} to generate string in shell script.

  Example:

```
{1..5} will give 1 2 3 4 5
{a..f} will give a b c d e f
{Z..T} will give Z Y X W V U T
{-5..5} will give -5 -4 -3 -2 -1 0 1 2 3 4 5
{A,B,C,D} will give A B C D
{A,B,C{1..3},D} will give A B C1 C2 C3 D
```

1. **for loop:**

   Syntax:

```
for variable in items
do
 # code for each item
done
```

Program 17: Write a Shell Script to print from 1 to 5 using for loop.

```
#!/bin/sh
for i in 1 2 3 4 5
do
echo $i
done
```

Output:

```
1
2
3
4
5
```

Program 18: Write a Shell Script to print from 1 to 5 using C style for loop.

```
#!/bin/sh
for(( i = 1; i<= 5; i++ ))
do
echo $i
done
```

Output:

```
1
2
3
4
5
```

---

2. While loop:

- The while loop is similar to a **for loop** and help in executing a block of code multiple times as long as some given condition is satisfied.

  Syntax:

  ```
  while [ condition ]
  do
   # body of while loop
  done
  ```

  Where condition is some condition which if satisfied results in the execution of the body of the loop.

  To come out of the while loop we make the condition fail.

  To quit the script we can use the exit command.

Program 19: Write a Shell Script to print from 1 to 5 using while loop.

```
#!/bin/sh
i=1
while [ $i -le 5 ]
do
echo $i
i=`expr $i + 1`
done
```

Output:

```
1
2
3
4
5
```

Program 20: Write a Shell Script to print the following pattern.

```
1
1 3
1 3 5
1 3 5 7
```

```
#!/bin/sh
# for the rows
r=1
while [ $r -le 4 ]
do
 # for the output
count=1
 # for the column
 c=1
while [ $c -le $r ]
do
   printf "$count "
count=`expr $count + 2`
c=`expr $c + 1`          //c=$(( $c + 1 ))
done
printf "\n"
 r=$(( $r + 1 ))
done
```

Output:

```
1
1 3
1 3 5
1 3 5 7
```

---

3. Until loop:

- The while loop is perfect for a situation where you need to execute a set of commands, while some condition is true. Sometimes you need to execute a set of commands until a condition is true.

  Syntax:

```
until [ condition ]
do
# body of loop
done
```

Program 21: Write shell script for until loop.

```
#!/bin/sh
a=0
until [ $a -lt 5 ]
do
echo $a
 a=`expr $a + 1`
done
```

Output:

```
1
2
3
4
5
```

---

4. Select Loop

- The select loop provides as easy way to create a numbered menu from which users can select options. It is useful when you need to ask the user to choose one or more options from a list.

  Syntax:

```
Select var in word1 word2…. wordn)
do
block of statements
done
```

Program 22: Write shell script for select loop.

```
select DRINK in tea coffee juice water appe all none
do
case $DRINK in tea|coffee|water|all)
echo" Go to canteen:"
 ;;
juice|appe)
echo" Available at home"
 ;;
none)
break
;;
 *)
echo"ERROR: Invalid selection"
 ;;
esac
done
```

Output:

```
Tea
Coffee
Water
Juice
Appe
All
None
#? Juice
Available at home
#? None
$
```

6.8 THE SET AND SHIFT COMMANDS AND HANDLING POSITIONAL PARAMETERS

1. Set Command:

- This command can be used to set the values of the positional parameters on the command line.

Example:

```
$ set how do you do
$ echo $1 $2
how do
```

Here, "how" was assigned to $1 and "do" was assigned to $2 and so on.

Now $1 will be 'morning' and so on to $8 being 'Linux' and $9 being 'tutorial'.

Program 23: Write shell script for set and shift commands.

```
Set Hello how are you
echo $1$2 $3
while[$# -gt 0]
do
echo $1
Shift
done
```

Output:

```
Hello how are
Hello
how
are
you
```

2. **Shift Command:**

- The shift command in LINUX is used to move the command line arguments to one position left.
- The shift command throws away the left-most variable (argument number 1) and reassigns values to the remaining variables. The value in $2 moves to $1, the value in $3 moves to $2, and so on.
- This command is used to shift the position of the positional parameters. i.e. $2 will be shifted to $1 all the way to the tenth parameter being shifted to $9.
- Note that if in case there are more than 9 parameters, this mechanism can be used to read beyond the 9^{th}.

  Example:
  - $ set hello good morning how do you do welcome to Linux tutorial.
  - Here, 'hello' is assigned to $1, 'good' to $2 and so on to 'to' being assigned to $9. Now the shift command can be used to shift the parameters 'N' places.

  Example:
  - $ shift 2
  - $echo $1
  - Now $1 will be 'morning' and so on to $8 being 'Linux' and $9 being 'tutorial'.

3. **Positional Parameters:**

- Positional parameters in a shell script are nothing but the command line arguments passed to a shell script.
- A positional parameter is a parameter denoted by one or more digits, other than the single digit 0.
- Positional parameters are assigned from the shell's arguments when it is invoked, and may be reassigned using the set built-in command.
- Positional parameter N may be referenced as ${N}, or as $N when N consists of a single digit.
- Positional parameters may not be assigned to with assignment statements. The set and shift built-in are used to set and unset them

Handling positional parameter

- $1 is the first commandline argument. If you run ./asdf.sh a b c d e, then $1 will be a, $2 will be b, etc. In shells with functions, $1 may serve as the first function parameter, and so forth.
- $0 Expands to the name of the shell or shell script. This is set at shell initialization. If Bash is invoked with a file of commands. $0 is set to the name of that file.

6.8.1 Here << Documents and Trap Command

<< Document:

- The text between the delimiters is used as the standard input of the command. Between the delimiters shell variable expansion and command substitution are performed.

  For example: `cat<< ENDSTRING`
- You are about to enter an interactive session which will modify the behavior of this system to suit your use.

- ENDSTRING will display the text between the delimiters to the screen.
- Here documents are useful for displaying known multi-line text to the screen without having to create a separate file to contain the text.

Trap Command:

- This command specifies the action to be taken when one or more specified signals are received. When the shell receives a specified signal, it executes the specified action.
- The trap command is a function of the shell that responds to hardware signals and other events.
- It defines and activates handlers to be run when the shell receives signals or other special conditions.

Syntax: `trap [option] [[ARG] SIGNAL_SPEC...]`

Options:

1. **–l:** print a list of signal names and their corresponding numbers.
2. **–p:** display the trap commands associated with each SIGNAL_SPEC.

- **ARG** is a command to be read and executed when the shell receives the signal(s) SIGNAL_SPEC. If ARG is absent (and a single SIGNAL_SPEC is supplied) or ARG is a dash ("-"), each specified signal is reset to its original value. If ARG is the null string, each SIGNAL_SPEC is ignored by the shell and by the commands it invokes.
- If a **SIGNAL_SPEC is EXIT (0),** ARG is executed upon exit from the shell.
- If a **SIGNAL_SPEC is DEBUG,** ARG is executed before every simple command.
- If a **SIGNAL_SPEC is RETURN,** ARG is executed each time a shell function or a script run by the "." or source built-in commands finishes executing.
- A **SIGNAL_SPEC of ERR** means to execute ARG each time a command's failure would cause the shell to exit when the -e option is enabled.
- If no arguments are supplied, trap prints the list of commands associated with each signal.

6.9 FILE INODES AND THE INODE STRUCTURE

6.9.1 File inodes

- Each LINUX file has a description that is stored in a structure called inode.
- Inode number also called index number.
- It consists following attributes:
  1. File types (executable, block special etc.).
  2. Permissions (read, write etc.).
  3. UID (Owner).
  4. GID (Group).
  5. FileSize.
  6. Time stamps including last access, last modification and last inode number change.
  7. File deletion time.
  8. Number of links (soft/hard).
  9. Location of ile on harddisk.
  10. Some other metadata about file.

Display File Data Information:

- You can display the inode data on a file or directory by using **stat** command. You need to indicate the name of the file.

```
# stat hello
```

Print the index number of files:

- The **ls command** is used to list file/folder information. The -i option with ls displays the inode number of each file. We can combine it with -l option to list information in detail

```
# ls -il
```

- Display filesystem inode space information
- By default, **df command** summarizes available and used disk space. You can instead receive a report on available and used inodes by passing the -i or --inodes option.

```
# df -i
```

- This information can be helpful if a partition has very many small files, which can deplete available inodes sooner than they deplete available disk space.
- List the contents of the filesystem superblock.
- You can use tune2fs -l command to displays all information related to inode.

```
# tune2fs -l /dev/sda6 | grep inode
```

- Manipulate the filesystem meta data.
- You can see the contents of an inode as it exists on an Ext4 file system with debugfs command.

6.9.2 Inode structure

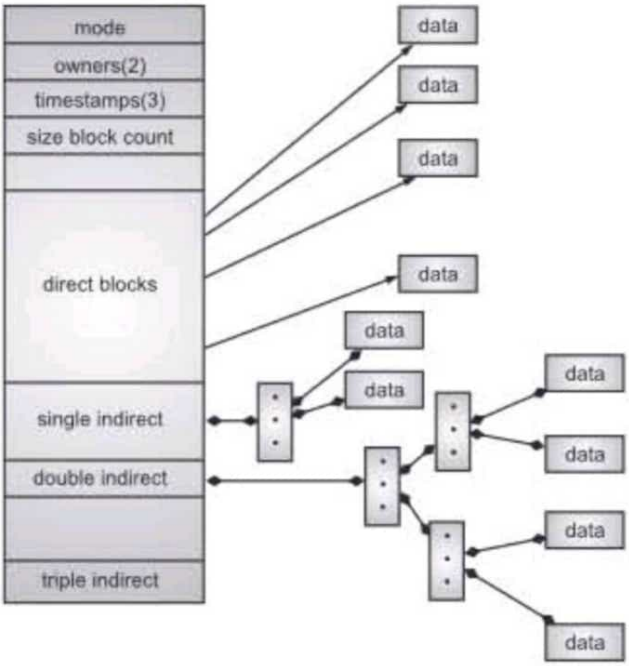


Fig. 6.1: inode structure

- **Mode:** This keeps information about two things, one is the permission information, the other is the type of inode, for example an inode can be of a file, directory or a block device etc.
- **Owner Info:** Access details like owner of the file, group of the file etc.
- **Size:** This location store the size of the file in terms of bytes.
- **Time Stamps:** it stores the inode creation time, modification time, etc.
- Now comes the important thing to understand about how a file is saved in a partition with the help of an inode.
- **Block Size:** Whenever a partition is formatted with a file system. It normally gets formatted with a default block size. If the block size is 4K, then for a file of 15K it will take 4 blocks(because 4K*4 16), and technically speaking you waste 1 K.
- **Direct block:** The blocks marked "direct" can refer to a single disk block that contains the real data. "single indirect" block contains the number of a disk block which in itself contains a list of block numbers that we can reference and they have the real data.
- Going on the same line, we have **"double indirect"** and **"triple indirect"** blocks. Lets now try to get an estimate of the max limit on the size of a file that Linux File System can handle.
- Assume one block is of 1 Kbytes and a block number is an integer of 4 bytes (32 bits). Thus a block can have 256 block numbers.

```
10 direct blocks => 10 K bytes
1 single indirect block with 256 block number entries => 256 Kbytes
1 double indirect block with 256 single indirect entries => 64 Mbytes
1 triple indirect block with 256 double indirect entries => 16 Gbytes
```

6.10 FILE LINKS

- A link is an association between a filename and an inode.
- A link in LINUX is a pointer to a file. Like pointers in any programming languages, links in LINUX are pointers pointing to a file or a directory.
- Creating links is a kind of shortcuts to access a file. Links allow more than one file name to refer to the same file, elsewhere.

There are two types of links:

- These links behave differently when the source of the link (what is being linked to) is moved or removed.
- Symbolic links are not updated (they merely contain a string which is the pathname of its target); hard links always refer to the source, even if moved or removed.

1. **Hard Links:**
   - Linking files by reference.
   - System maintains a count of the number of links.
   - Does not work across file systems.
   - ls -l command shows all the links with the link column shows number of links.

   Command to create a hard link is:

```
$ ln [original filename] [link name]
```

2. **Soft Links:**
   - Linking files by name
   - No counter is maintained
   - Work across file system
   - Each soft linked file contains a separate Inode value that points to the original file. As similar to hard links, any changes to the data in either file is reflected in the other.
   - ls -l command shows all links with first column value l? and the link points to original file.
   - Soft Link contains the path for original file and not the contents.
   - A soft link can link to a directory.

   Command to create a Soft link is:

   ```
   $ ln -s [original filename] [link name]
   ```

   - Directory entries are hard links as they directly link an inode to a filename. Symbolic links use the file (contents) as a pointer to another filename.

6.11 FILTERS

- In LINUX, a filter is a command that gets its input from the standard input stream, manipulates the input and then sends the result to the standard output stream.
- Some filters can receive data directly from a file.
- There are 12 more simple filters available. Three filters – **grep, sed** and **awk**– are so powerful. The following table summarizes the common filters:

Filter	Action
Cat	Passes all data from input to output.
Cmp	Compares two files.
Comm	Identifies common lines in two files.
Cut	Passes only specified columns.
Diff	Identifies differences between two files or between common files in two directories.
Head	Passes the number of specified lines at the beginning of the data.
Paste	Combines columns.
Sort	Arranges the data in sequence.
Tail	Passes the number of specified lines at the end of the data.
Tr	Translates one or more characters as specified.
Uniq	Deletes duplicate (repeated) lines.
Wc	Counts characters, words, or lines.
Grep	Passes only specified lines.
Sed	Passes edited lines.
Awk	Passes edited lines – parses lines.

6.12 HEAD AND TAIL COMMANDS

- LINUX provides two commands, **head** and **tail** to display the portions of files.

head Command

- While the cat command copies entire files, the head command copies a specified number of lines from the beginning of one or more files to the standard output stream. If no files are specified it gets the lines from standard input.

  Format: head options inputfiles . . .
- The option –N is used to specify the number of lines. If the number of lines is omitted, head assumes 10 lines. If the number of lines is larger than the total number of lines in the file, the total file is used.

  Example:

```
$ head -2 first.txt
Now is the time
To start with shell script
```

- When multiple files are included in one head command, head displays the name of file before its output.

  Example:

```
$ head -2 first.txt second.txt
first.txt
Now is the time.
To start with shell script.
second.txt
I love scripting language.
It is same as other script language.
```

tail Command:

- The tail command also outputs data, only this time from the end of the file.

  Format: `tail options inputfile`
- Although only file can be referenced (in most systems), it has several options as shown in the table below:

Option	Code
Count from beginning	+N
Count form end	-N
Count by lines	-l
Count by characters	-c
Count by blocks	-b
Reverse order	-r
Quiet	-q
verbose	-v
follow	-f

- **-n num:** Prints the last 'num' lines instead of last 10 lines. num is mandatory to be specified in command otherwise it displays an error. This command can also be written as without symbolizing 'n' character but '-' sign is mandatory.

```
$ tail -n 3 state.txt
```

OR

```
$ tail -3 state.txt
```

Tail command also comes with an **'+'** option which is not present in the head command. With this option tail command prints the data starting from specified line number of the file instead of end. For command: **tail +n file_name**, data will start printing from line number 'n' till the end of the file specified.

```
$ tail +25 state.txt
```

2. **-c num:** Prints the last 'num' bytes from the file specified. Newline count as a single character, so if tail prints out a newline, it will count it as a byte. In this option it is mandatory to write -c followed by positive or negative num depends upon the requirement. By +num, it display all the data after skipping num bytes from starting of the specified file and by -num, it display the last num bytes from the file specified.

Note: Without positive or negative sign before num, command will display the last num bytes from the file specified.

```
With negative num
$ tail -c -6 state.txt
 OR
$ tail -c 6 state.txt
```

3. **-q:** It is used if more than 1 file is given. Because of this command, data from each file is not precedes by its file name.

```
Without using -q option
$ tail state.txt capital.txt

With using -q option
$ tail -q state.txt capital.txt
```

4. **-f:** This option is mainly used by system administration to monitor the growth of the log files written by many Linux program as they are running. This option shows the last ten lines of a file and will update when new lines are added. As new lines are written to the log, the console will update with the new lines. The prompt doesn't return even after work is over so, we have to use the interrupt key to abort this command. In general, the applications writes error messages to log files. You can use the -f option to check for the error messages as and when they appear in the log file.

```
$ tail -f logfile
```

5. **-v:** By using this option, data from the specified file is always preceded by its file name.

```
$ tail -v state.txt
```

6. **–version:** This option is used to display the version of tail which is currently running on your system.

```
$ tail --version
```

6.13 CUT AND PASTE COMMANDS

- In LINUX cut command removes the columns of data from a file, and paste combines columns of data. Because the cut and paste commands work on columns, text files don't work well.
- The cut command in LINUX is a command for cutting out the sections from each line of files and writing the result to standard output.
- It can be used to cut parts of a line by byte position, character and field.
- Basically the cut command slices a line and extracts the text.
- It is necessary to specify option with command otherwise it- gives error.

  Syntax: `cut OPTION... [FILE]...`
- Consider two files **state.txt** and **capital.txt**

```
$ cat state.txt
Andhra Pradesh
Arunachal Pradesh
Assam
Bihar
Chhattisgarh
```

Options and their Description with examples:

1. **-b(byte):** To extract the specific bytes, you need to follow -b option with the list of byte numbers separated by comma. Range of bytes can also be specified using the hyphen(-). It is necessary to specify list of byte numbers otherwise it gives error. Tabs and backspaces are treated like as a character of 1 byte.

   List without ranges:

   $ cut -b 1,2,3 state.txt

   And
   Aru
   Ass
   Bih
   Chh

   List with ranges:

   $ cut -b 1-3,5-7 state.txt

   Andra
   Aruach
   Assm
   Bihr
   Chhtti

2. **-c (column):** To cut by character use the -c option. This selects the characters given to the -c option. This can be a list of numbers separated comma or a range of numbers separated by hyphen(-). Tabs and backspaces are treated as a character. It is necessary to specify list of character numbers otherwise it gives error with this option.

   Syntax: `$cut -c [(k)-(n)/(k),(n)/(n)] filename`

   Here, **k** denotes the starting position of the character and **n** denotes the ending position of the character in each line, if k and n are separated by "-" otherwise they are only the position of character in each line from the file taken as an input.

   $ cut -c 2,5,7 state.txt

   nr

   rah

   sm

   ir

   hti

   Above cut command prints second, fifth and seventh character from each line of the file.

   $ cut -c 1-7 state.txt

   Andhra

   Arunach

   Assam

   Bihar

   Chhatti

   Above cut command prints first seven characters of each line from the file.

3. **-f (field): -c** option is useful for fixed-length lines. Most Linux files doesn't have fixed-length lines. To extract the useful information you need to cut by fields rather than columns. List of the fields number specified must be separated by comma. Ranges are not described with -f option. **cut** uses **tab** as a default field delimiter but can also work with other delimiter by using **-d** option. **Note:** Space is not considered as delimiter in LINUX.

   Syntax: `$cut -d "delimiter" -f (field number) file.txt`

   Like in the file **state.txt** fields are separated by space if -d option is not used then it prints whole line:

   $ cut -f 1 state.txt

   Andhra Pradesh

   Arunachal Pradesh

   Assam

   Bihar

   Chhattisgarh

   If -d option is used then it considered space as a field separator or delimiter:

4. **–complement:** As the name suggests it complement the output. This option can be used in the combination with other options either with **-f** or with **-c**.
   Arunchal Pradesh
   Assa
   Biha
   Chhatisgarh
5. **–output-delimiter:** By default the output delimiter is same as input delimiter that we specify in the cut with **-d** option. To change the output delimiter use the option **–output-delimiter="delimiter"**.
   $ cut -d " " -f 1,2 state.txt --output-delimiter='%'
6. **–version:** This option is used to display the version of cut which is currently running on your system.

Option	Code	Results
Character	-c	Extracts fixed columns specified by column number
Field	-f	Extracts delimited columns
Delimiter	-d	Specifies delimiter if not tab (default)
Suppress	-s	Suppresses output if no delimiter in line.

Paste Command:

The paste command combines lines together.

It gets the input from two or more files.

Tospecifythattheinputiscomingfromthestandardinputstream,youuseahyphen(-)instead of a filename.

Syntax: `paste options file list`

The paste combines the first line of the first file with the first line of the second file and writes the combined line to the standard output stream. Between the columns, it writes a tab.

At the end of the last column, it writes a newline character.

It then combines the next two lines and writes them, continuing until all the lines have been written to standard stream.

Note: The cat and paste command are similar: The cat command combines files vertically (by lines). The paste command combines files horizontally (bycolumns).

6.14 SORT COMMAND AND ITS USAGE WITH DIFFERENT COMMANDS

- Sorts the contents of a text file, line by line.
- It used to rearrange the contents of text files line by line.
- The command is a filter command that sorts the input text and prints the result to stdout. By default, sorting is done line by line, starting from the first character.
- Numbers are sorted to be ahead of letters.
- Lowercase letters are sorted to be ahead of uppercase letters.

Linux Sort Command with Examples:

Sort Syntax: `sort [options] [files]`

6.14.1 Sort Options

- **Some of the options supported are:**

  sort -b: Ignore blanks at the start of the line.

  sort -r: Reverse the sorting order.

  sort -o: Specify the output file.

  sort -n: Use the numerical value to sort.

  sort -M: Sort as per the calendar month specified.

  sort -u: Suppress lines that repeat an earlier key.

  sort -k POS1, POS2: Specify a key to do the sorting. POS1 and POS2 are optional parameters and are used to indicate the starting field and the ending field indices. Without POS2, only the field specified by POS1 is used. Each POS is specified as "F.C" where F represents the field index, and C represents the character index from the start of the field.

  sort -t SEP: Use the provided separator to identify the fields.

  Examples: Assume the below initial contents of file1.txt for the following examples.

  01 Priya
  04 Shreya
  03 Tuhina
  02 Tushar

  Sort with default ordering:

  $ sort file1.txt
  01 Priya-
  02 Tushar
  03 Tuhina
  04 Shreya

  Sort in reverse ordering:

  $ sort -r file1.txt
  04 Shreya
  03 Tuhina
  02 Tushar
  01 Priya

6.15 SIMPLE SHELL PROGRAM EXAMPLES

1. **Write a shell script to ask your name, program name and enrollment number and print it on the screen.**

```
Echo "Enter your name:"
Read Name
Echo "Enter your program name:"
Read Prog
Echo "Enter your enrollment number:"
Read Enroll
Clear
Echo "Details you entered"
Echo Name: $Name
Echo Program Name: $Prog
Echo Enrolment Number: $Enroll
```

2. **Write a shell script to find the sum, the average and the product of the four integers entered.**

```
Echo "Enter four integers with space between"
Read a b c d
Sum =`expr $a + $b + $c + $d`
Avg =`expr $sum / 4`
Dec =`expr $sum % 4`
Dec =`expr \ ($dec \* 1000 \) / 4`
Product =`expr $a \* $b \* $c \* $d`
Echo Sum = $sum
Echo Average = $avg. $dec
Echo Product = $product
```

3. **Write a shell program to exchange the values of two variables.**

```
Echo "Enter value for a:"
Read a
Echo "Enter value for b:"
Read b
Clear
Echo "Values of variables before swapping"
Echo A = $a
Echo B = $b
Echo Values of variables after swapping
a = `expr $a + $b`
b = `expr $a - $b`
a = `expr $a - $b`
Echo A = $a
Echo B = $b
```

4. **Write a shell script to display the digits which are in odd position in a given 5 digit number.**

```
Echo "Enter a 5 digit number"
Read num
n = 1
while [ $n -le 5 ]
do
a = `Echo $num | cut -c $n`
Echo $a
n = `expr $n + 2`
done
```

5. **Write a shell program to reverse the digits of five digit integer.**

```
Echo "Enter a 5 digit number"
Read num
n = $num
rev=0
while [ $num -ne 0 ]
do
```

```
r = `expr $num % 10`
rev = `expr $rev \* 10 + $r`
num = `expr $num / 10`
done
Echo "Reverse of $n is $rev"
```

6. Write a shell script to find the largest among the 3 given numbers.

```
Echo "Enter 3 numbers with spaces in between"
Read a b c
1 = $a
if [ $b -gt $l ]
then
l = $b
fi
if [ $c -gt $l ]
then
l = $c
fi
Echo "Largest of $a $b $c is $l"
```

7. Write a shell program to concatenate two strings and find the length of the resultant string.

```
Echo "Enter first string:"
Read s1
Echo "Enter second string:"
Read s2
s3 = $s1$s2
len = `Echo $s3 | wc -c`
len = `expr $len - 1`
Echo "Concatenated string is $s3 of length $len "
```

8. Write a shell program to display the alternate digits in a given 7 digit number starting from the first digit.

```
Echo "Enter a 7 digit number"
Read num
n = 1
while [ $n -le 7 ]
do
a = `Echo $num | cut -c $n`
Echo $a
n = `expr $n + 2`
done
```

9. Write a shell program to check whether a given string is palindrome or not.

```
Echo "Enter a string to be entered:"
Read str
Echo
len = `Echo $str | wc -c`
len = `expr $len - 1`
```

```
i = 1
j = `expr $len / 2`
while test $i -le $j
do
k = `Echo $str | cut -c $i`
l = `Echo $str | cut -c $len`
if test $k != $l
then
Echo "String is not palindrome"
exit
fi
i = `expr $i + 1`
len = `expr $len - 1`
done
Echo "String is palindrome"
```

10. Write a shell script to find the smallest of three numbers Echo "Enter 3 numbers with spaces in between".

```
Read a b c
s = $a
if [ $b -lt $s ]
then
s = $b
fi
if [ $c -lt $s ]
then
s = $c
fi
Echo "Smallest of $a $b $c is $s"
```

11. Write a shell program to find factorial of given number.

```
Echo "Enter a number"
Read n
fact = 1
i = 1
while [ $i -le $n ]
do
fact = `expr $fact \* $i`
i = `expr $i + 1`
done
Echo "Factorial of $n is $fact"
```

Summary

- A LINUX shell is command interprets a user command which are directly entered by user or which can be read from a file called as shell programming or shell scripting.
- Command line arguments are useful for passing input to script at runtime which has its own advantage. This article will help you to pass command line arguments in a shell script.
- To pass a command line argument we can simply write them after script name separated with space. All command line parameters can be access by their position number using $.
- The exit command terminates a script, just as in a C program. Every command returns an exit status (sometimes referred to as a return status or exit code).
- The set and shift commands are used to set and unset positional parameters.
- Hard links
  - Linking files by reference
  - System maintains a count of the number of links
  - Does not work across file systems.
- Soft links
  - Linking files by name
  - No counter is maintained
  - Work across file system
- Head and tail to display the portions of files.
- Cut command removes the columns of data from a file, and paste combines columns of data.
- Sort command is used sorting the contents of the text files.

Check Your Understanding

1. What is a shell script?
   (a) group of commands
   (b) a file containing special symbols
   (c) a file containing a series of commands
   (d) group of functions
2. Shell scripts need to be saved with an extension .sh .
   (a) True (b) False
3. The first line in any shell script begins with a _____
   (a) & (b) !
   (c) $ (d) #
4. To run the script, we should make it executable first by using _____
   (a) chmod +x (b) chmod +r
   (c) chmod +w (d) chmod +rwx
5. Which command is used for making the scripts interactive?
   (a) ip (b) input
   (c) read (d) write
6. read command is shell's internal tool.
   (a) True (b) False

7. A single read statement can be used with one or more variables.
   (a) True (b) False
8. What are positional parameters?
   (a) special variables for assigning arguments from the command line
   (b) pattern matching parameters
   (c) special variables for reading user input
   (d) special variables and patterns
9. The first argument is read by the shell into the parameter _____
   (a) 1$ (b) $3
   (c) $$ (d) $1
10. The complete set of positional parameters is stored in _____ as a single string.
   (a) $n (b) $#
   (c) $* (d) $$

Answers

1. (c)	2. (b)	3. (d)	4. (a)	5. (c)	6. (a)	7. (a)	8. (a)	9. (d)	10. (d)

Practice Questions

Short Answer Questions:

1. Define Inode.
2. What is use of EXIT and EXIT STATUS command.
3. Define-Hard links, Soft links.
1. What is the use of sort command in Linux?
4. Explain Test command.

Long Answer Questions:

1. Explain shell script.
2. What are the different shell variables available in LINUX OS?
3. What is Filter? Explain any four filter commands.
4. Illustrate positional parameters with examples.
5. Explain control structures in Shell programming.
6. Write shell script to reverse a given number and check whether it is palindrome.
7. Write a shell script to count the number of vowels in a given.
8. Write the syntax of if-then-else-if statement with an example.
9. Explain different types of test used in shell script with an example.
10. Write short note on filters.

■■■