GEETANJALI GROUP OF COLLEGES

UNIT 1: INTRODUCTION, PROCESS AND THREAD, PROCESS SCHEDULING

PART – 1

TOPICS:

- **Meaning of OS**
- **Functions of OS**
- **Features of OS**
- **OS Types**

❖ Meaning of OS

- ✓ An operating system (OS) serves as the foundation for all software programs on a computer. It acts as the primary centre for managing hardware and software.
- ✓ The OS serves as a bridge between the user and the computer's hardware.
- ✓ The operating system is the first piece of software to run on a computer when it is booted.
- ✓ Its job is to coordinate the execution of all other software, mainly user applications.
- ✓ It also provides various common services that are needed by users and applications.
- ✓ The purpose of an operating system is to provide an environment in which a user may execute program.
- ✓ An operating system acts as a resource manager and allocates resources to specific programs and users as necessary for their task.
- ✓ The commonly required resources are Input/output devices, memory, file storage space, CPU time and so on.
- ✓ The operating system must ensure the correct operation of the computer system. The hardware must provide appropriate mechanisms to prevent user programs from interfering with the proper operation of the system.

❖ Functions of an Operating System:

- ✓ **Process Management:**
  - o The OS manages processes by creating, scheduling, and terminating processes. It allocates resources (CPU time, memory) to processes and ensures they run smoothly without interfering with one another.
- ✓ **Memory Management:**
  - o The OS handles the computer's memory, ensuring that processes get the necessary memory space while preventing memory

conflicts. It includes functions like allocating and deallocating memory and managing virtual memory.

- ✓ **File System Management:**
    - The OS organizes, stores, retrieves, and manages files. It controls how data is stored on hard drives or other storage devices and provides a way to name, store, and access files.
- ✓ **Device Management:**
    - The OS controls and manages input and output devices (e.g., printers, monitors, keyboards) via device drivers. It provides an interface for these devices to interact with the system.
- ✓ **Security and Access Control:**
    - The OS ensures the system's security by managing user authentication, setting access permissions, and protecting data from unauthorized access or malware.
- ✓ **User Interface (UI):**
    - The OS provides an interface for users to interact with the system, such as a graphical user interface (GUI) or command-line interface (CLI). This allows users to run programs, access files, and perform system tasks.

❖ **Key Features of an Operating System:**

- ✓ **Multitasking:**
    - The ability to run multiple processes or applications simultaneously, sharing CPU time among them.
- ✓ **Multithreading:**
    - The ability for a process to create multiple threads that can execute concurrently, improving the efficiency of programs by performing multiple operations at once.
- ✓ **Networking:**
    - OSs provide networking features that allow computers to communicate over local networks or the internet. This includes managing network connections, protocols, and services.
- ✓ **Virtual Memory:**
    - Virtual memory allows the OS to use hard disk space as an extension of RAM, enabling programs to use more memory than is physically available.
- ✓ **Error Detection and Handling:**
    - The OS detects and responds to system errors (e.g., hardware failure, software bugs) and provides mechanisms to recover or alert users.

- ✓ **Resource Allocation:**
  - The OS efficiently allocates resources like CPU time, memory, and storage to various processes to ensure fair use and optimal system performance.
- ✓ **File System Organization:**
  - The OS provides a way to organize files using directories and supports different file systems (e.g., NTFS, FAT32, ext4).
- ✓ **Command Line Interface (CLI) / Graphical User Interface (GUI):**
  - OSs typically offer a CLI (e.g., Windows Command Prompt, Linux terminal) or GUI (e.g., Windows, macOS, Linux desktop environments) for user interaction.
- ✓ **System Utilities:**
  - The OS includes utilities and tools for system maintenance, such as disk clean up, antivirus programs, and task managers.
- ✓ **Power Management:**
  - The OS manages power usage on portable devices and desktops, adjusting power consumption for efficiency and to preserve battery life on laptops and mobile devices.

❖ OS Types (User & Features Point of View)

1. Batch Operating System

- ✓ When executing programs, batch operating systems do so in batches. Because devices with batch operating systems submit tasks to an operator, this type of system is ideal for time-consuming tasks with large files.
- ✓ **This type of system is widely used for payroll, data entry, and similar operations.**
- ✓ While beneficial for large tasks, you might have difficulty debugging batch operating systems when errors arise, and implementing a strong batch operating system can be resource-intensive.

2. Time-sharing Operating System

- ✓ Time-sharing operating systems switch between multiple tasks, each with a time limit.
- ✓ **This type of system benefits users from multiple locations by enabling them to use it simultaneously and providing quick responses to user input.**
- ✓ Time-sharing systems switch quickly from user to user, giving the insight that the entire system is dedicated to you when in reality, it’s supporting multiple users.

- ✓ While the limited idle time of the processor leads to fast task completion, system failure can cause universal failure across all tasks. Multics and Unix are two common operating systems with this style.

3. **Distributed Operating System**
   - ✓ Distributed operating systems connect multiple machines together through a shared network.
   - ✓ Each computer has its own memory and operating system but can communicate with other devices through the distributed system.
   - ✓ **This allows data sharing to happen quickly for system users, limits data processing delays, and reduces the load placed on the host machine.**
   - ✓ Telecommunications networks and airline reservation controls use this type of system because it allows for fast data exchange, is resilient against system failures, and can scale easily.

4. **Network Operating System**
   - ✓ **Network operating systems connect computers and other devices through a shared network, known as a local area network (LAN) Through this network, you can to access the same documents and share files and physical devices.**
   - ✓ Because of their advantages, such as remote server access, most users benefit from this type of operating system.
   - ✓ **Some of the most popular operating systems, such as Microsoft Windows, Linux, and macOS X, are all network operating systems.**
   - ✓ Because devices connect to a central server, it's easy for users to update their devices, share files, and handle security issues. However, the centralized systems can be expensive and require routine maintenance to ensure they support the devices on the network.

5. **Real-time Operating System**
   - ✓ Real-time operating systems perform tasks close to real-time, following to strict timing requirements.
   - ✓ **This type of operating system is best when precision is highly important, making it the operating system of choice for air traffic control systems, weapon control systems, robots, and medical imaging.**
   - ✓ To prioritize timing, less focus is put on queued tasks, and more on the current task.
   - ✓ This lowers the ability to run tasks simultaneously and can limit the prioritization of tasks in certain cases.

6. **Multi-programming System**
   - ✓ Multi-programming OS takes a step forward in efficiency by allowing multiple programs to be loaded into memory simultaneously.
   - ✓ **While one program is waiting for input/output operations or other delays, the CPU can execute another program.** This approach optimises CPU usage and minimises idle time, resulting in improved overall system performance.

7. **Multi-Processing System**
   - ✓ Multi-processing OS take advantage of multiple central processing units (CPUs) within a single computer.
   - ✓ **These systems can distribute tasks across multiple CPUs, allowing for true parallel processing.**
   - ✓ This approach significantly enhances computing power and is commonly found in servers, high-performance workstations, and supercomputers.

8. **Multi-Tasking OS**
   - ✓ Multi-tasking OS is designed for personal computers and workstations, allowing users to run multiple applications simultaneously.
   - ✓ **Each application runs as a separate process, and the OS manages the allocation of CPU time to each process.**
   - ✓ Users can switch between applications seamlessly, making efficient use of system resources.

9. **Single-User, Single-Tasking OS**
   - ✓ These operating systems are designed to support only a single user and allow them to perform one task at a time. **Examples include MS-DOS (Microsoft Disk Operating System) and early versions of Apple's Macintosh operating system.**

10. **Single-user, Multi-Tasking OS**
    - ✓ These operating systems enable a single user to perform multiple tasks simultaneously or in parallel.
    - ✓ They allow for efficient multitasking by dividing the processor's time among different tasks.
    - ✓ **Examples include Microsoft Windows, macOS, and Linux distributions like Ubuntu.**

11. **Embedded OS**
    - ✓ Embedded operating systems are lightweight and optimised for running on devices with limited resources, such as smartphones, IoT devices.

PART – 2

TOPICS:

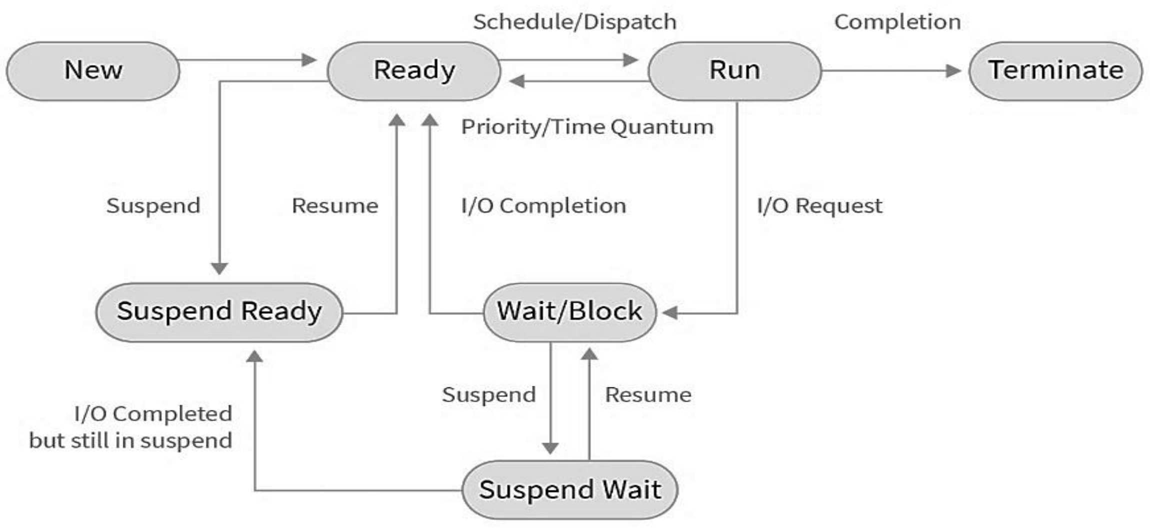
- **Process Definition**
- **Process States**
- **Process State Transitions**
- **Process Control Block**
- **Context Switching**
- **Threads**
  - **Concept of multithreads**
  - **Benefits of threads**
  - **Types of threads**

❖ Process

- ✓ The program under execution is called Process. Process is an active entity.
- ✓ Process goes through different states throughout the life cycle during the process execution, which is known as process states.
- ✓ **All the information associated with the process is stored in the Process Control Block (PCB).**

❖ Process States in Operating System

- ✓ From start to finish, the process goes through a number of stages. A minimum of five states is required.
- ✓ Even though the process could be in one of these states during execution, the names of the states are not standardised. Throughout its life cycle, each process goes through various stages.
- ✓ Whenever the process creation is taking place process is in a new state and when the process gets terminated it is in the terminated state or completed state.



- ✓ The states of the process are stored in the Process Control Block(PCB). PCB is a special data structure that stores information about the process.

List of States of the Process

1. **New State**
    - a. This is the first state of the process life cycle. **When process creation is taking place, the process is in a new state.**

2. **Ready State**
    - **a.** When the process creation is completed, the process comes into a ready state. **During this state, the process is loaded into the main memory and will be placed in the queue of processes which are waiting for the CPU allocation.**
    - **b.** When the process is in the creation process is in a new state and when the process gets created process is in the ready state.

3. **Running State**
    - **a. Whenever the CPU is allocated to the process from the ready queue, the process state changes to Running.**

4. **Block or Wait State**
    - **a.** When the process is executing the instructions, the process might require carrying out a few tasks that might not require CPU.
    - **b.** If the process requires performing Input-Output task or **the process needs some resources that are already acquired by other processes, during such conditions process is brought back into the main memory, and the state is changed to Blocking or Waiting for the state.**
    - **c.** Process is placed in the queue of processes that are in waiting or block state in the main memory.

5. **Terminated or Completed**
    - **a.** When the entire set of instructions is executed and the process is completed. The process is changed to a terminated or completed state.
    - **b. During this state the PCB of the process is also deleted.**

Note – 01:

A process must pass through at least four states.

A process must go through a minimum of four states to be considered complete.

The new state, run state, ready state, and terminate state are the four states.

Note – 02:

Present in Memory	State
Secondary Memory	New state
Main Memory	Ready state
Main Memory	Run state
Main Memory	Wait state
Secondary Memory	Suspend wait state
Secondary Memory	Suspend ready state

❖ Process Control Block (PCB)

- ✓ A Process Control Block (PCB) is a data structure used by the operating system to store and manage information about a specific process.
- ✓ The Process Control Block is also known as **Task Control Block**. A data structure (a table) that contains information on a process is known as a process control block, or PCB. **A PCB is required for every process or software that runs.**
- ✓ The operating system creates a process control block for a specific task when a user wishes to launch it. **Its alternate name is Task Control Block (TCB).**
- ✓ It basically acts as the breathing system for any information that might differ from process to process because it contains several bits of information related to a single process.

Why is the PCB important in an operating system?

The PCB is crucial because it allows the operating system to:

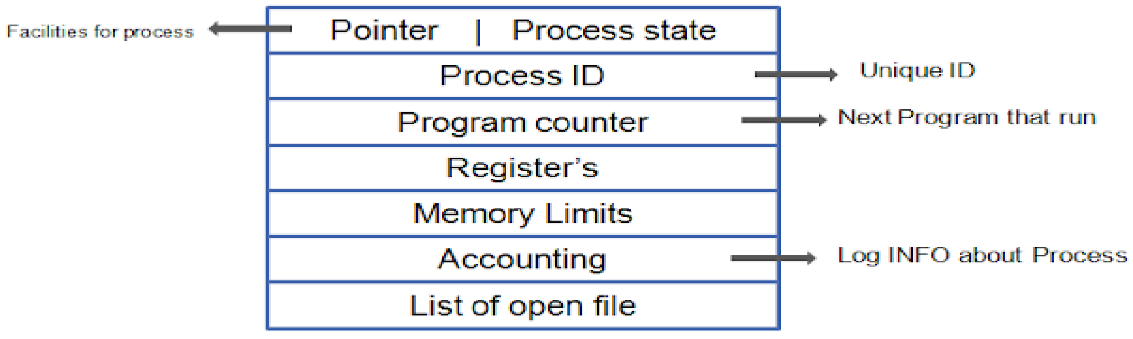
- ✓ Keep track of multiple processes
- ✓ Manage process scheduling
- ✓ Allocate resources to processes
- ✓ Switch between processes quickly (context switching)

What information does a PCB typically contain?

A PCB usually includes:

- ✓ **Process ID (PID):** A unique number to identify the process
- ✓ **Process State:** Whether it's running, ready, waiting, etc.
- ✓ **Program Counter:** Address of the next instruction to be executed
- ✓ **CPU Registers:** Contents of various processor registers
- ✓ **CPU Scheduling Information:** Priority level, scheduling queue pointers
- ✓ **Memory Management Information:** Memory allocated to the process

- ✓ **I/O Status Information:** List of I/O devices allocated to the process
- ✓ **Accounting Information:** CPU time used, time limits, account numbers



PCB Diagram

- ✓ **Pointer:** it used to switch in between the process.
- ✓ **Register:** every process has some registers to achieve special functionality. Many type of Register are used like, Tag, Flag, Index & stack register are used.
- ✓ **List of open file:** how many files are open (use) for this process & how many resources are used.

❖ Context Switching

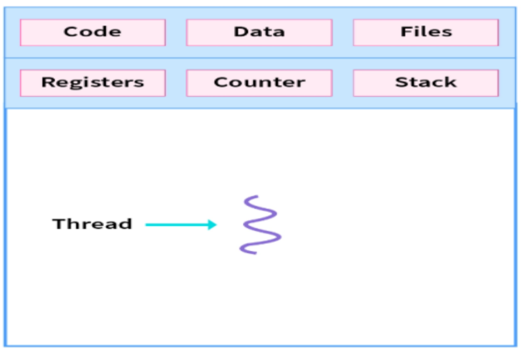
- ✓ Context switching refers to a **method used by the OS to switch processes from a given state to another one for the execution of its function using the CPUs present in the system.**
- ✓ A context switch is an operation that a computer's central processing unit (CPU) carries out when **alternating between processes or threads while ensuring that the processes do not conflict.**
- ✓ Effective context switching makes it possible to support a multitasking environment.
- ✓ **Each time the CPU makes a switch, the system temporarily d the currently executing task, saves its state (context) to a process control block (PCB), and then executes the next task in the queue.**
- ✓ Context switching is necessary in operating systems for the following reasons:

  - **Multitasking:**
    - Context switching allows multiple processes to run concurrently on a single CPU. This is essential for modern operating systems, which need to support a wide range of applications, such as web browsers, email clients, and video games.

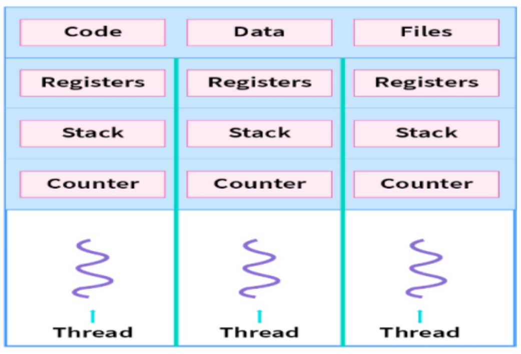
- **Efficient resource management:**
  - Context switching allows the operating system to efficiently manage the CPU and other resources. For example, **if one process is blocked waiting for I/O, the operating system can switch to another process that can run while waiting for the I/O to complete.**
- **Fairness:**
  - Context switching ensures that each process gets a fair share of the CPU's time. **Without context switching, a single process could control the CPU and prevent other processes from running.**

❖ **Threads**

- ✓ A thread **refers to a single sequential flow of activities being executed in a process**; it is also known as the thread of execution or the thread of control.
- ✓ Thread is a sequential flow of tasks within a process.
- ✓ Threads are **used to increase the performance of the applications**.
- ✓ A thread refers to an execution unit in the process that has its own programme counter, stack, as well as a set of registers.
- ✓ **Eg: While playing a movie on a device the audio and video are controlled by different threads in the background.**
- ✓ Types of threads per process
  - There are two sorts of process based on the number of threads:
    - Single thread process: Only one thread in an entire process
    - Multi-thread process: Multiple threads within an entire process.



Single-threaded process



Multithreaded process

Concept of multithreads

- ✓ In Multithreading, the **idea is to divide a single process into multiple threads instead of creating a whole new process.**

- ✓ The term multithreading means a process and a thread. Process means a program that is being executed.
- ✓ **Processes are further divided into independent units also known as threads**, also known as collections of threads.
- ✓ It is a process that is small and light-weighted residing inside a process.
- ✓ Multithreading divides the task of application into separate individual threads.
- ✓ It has a segment that divides the code into a small set of tasks that are lightweight and give less load to CPU memory.
- ✓ **Examples of Multithreading Operating Systems**
  - Multithreading is widely used by applications. Some of the applications are processing transactions like online bank transfers, recharge, etc.
  - For instance, in the banking system, many users perform day-to-day activities using bank servers like transfers, payments, deposits, `opening a new account, etc. All these activities are performed instantly without having to wait for another user to finish.

Benefits of Threads

- ➢ **Improved Performance and Responsiveness:**
  - By executing multiple threads concurrently, the program can perform multiple operations in parallel. This is especially useful in multi-core processors where threads can be run on different cores simultaneously, improving overall performance.
  - In GUI applications, **threads can ensure that the user interface remains responsive while other tasks (such as data loading or calculations) run in the background.**

- ➢ **Better Resource Utilization:**
  - Threads allow efficient use of CPU resources, especially in systems with multiple processors or cores. **Instead of waiting for a process to complete, threads can be scheduled to run on different cores.**

- ➢ **Scalability:**
  - With multithreading, **an application can scale efficiently as the number of cores in the system increases**, making it suitable for high-performance computing.

- **Concurrency:**
  - Threads allow concurrent execution, **which is useful for applications that require simultaneous operations, like web servers handling multiple client requests.**

Types of Threads

- **User Threads:**
  - These are threads that are created and **managed by the application itself**, not by the operating system (OS).
  - The application's threading library or framework is responsible for managing user threads, including creation, synchronization, and destruction.
  - **User threads are lightweight because the OS doesn't have to handle thread scheduling. They are easier to create and manage.**
  - The operating system doesn't have direct knowledge of these threads, which can limit the ability to optimize the use of system resources.

- **Kernel Threads:**
  - These are threads that are **managed directly by the operating system.** The OS handles the scheduling and management of kernel threads.
  - The OS kernel is aware of the kernel threads and schedules them for execution.
  - Since the OS manages kernel threads, it can optimize resource allocation and thread scheduling.
  - **Kernel threads can be heavier than user threads because they require more resources and have more overhead, as they involve the OS in managing them.**

PART – 3

TOPICS:

- **Types of Schedulers**
- **CPU Scheduling Algorithms**
    - **FCFS**
    - **SJN**
    - **Round Robin**
    - **Priority Base Non-Preemptive**
    - **Priority Base Preemptive**

❖ Types of Schedulers

- ✓ Process Scheduling handles the selection of a process for the processor on the basis of a scheduling algorithm and also the removal of a process from the processor.
- ✓ The process manager's activity is process scheduling, which involves removing the running process from the CPU and selecting another process based on a specific strategy.

- ✓ **Long term scheduler**
    - Long term scheduler is also known as job scheduler. **It chooses the processes from the pool (secondary memory) and keeps them in the ready queue maintained in the primary memory.**
    - Long Term scheduler mainly controls the degree of Multiprogramming.

- ✓ **Short term scheduler**
    - Short term scheduler **is also known as CPU scheduler**. **It selects one of the Jobs from the ready queue and dispatch to the CPU for the execution.**
    - **A scheduling algorithm is used to select which job is going to be dispatched for the execution.**

- ✓ **Medium term scheduler**
    - Medium term scheduler takes care of the swapped out processes. **If the running state processes needs some IO time for the completion, then there is a need to change its state from running to waiting.**
    - It removes the process from the running state to make room for the other processes.
    - **The medium term scheduler is responsible for suspending and resuming the processes.**

❖ CPU Scheduling Algorithms

- ✓ CPU scheduling is the task performed by the CPU that decides the way and order in which processes should be executed.
- ✓ **There are two types of CPU scheduling - Pre-emptive, and non-pre-emptive.**
- ✓ The criteria the CPU takes into consideration while "scheduling" these processes are **- CPU utilization, throughput, turnaround time, waiting time, and response time.**
- ✓ There are essential 4 conditions under which CPU scheduling decisions are taken:
    - **If a process is making the switch between the running state to the waiting state** (could be for an I/O request, or invocation of wait() for terminating one of its child processes)
    - **If a process is making the switch from the running state to the ready state** (on the occurrence of an interrupt, for example)
    - **If a process is making the switch between waiting and ready state** (e.g. when its I/O request completes)
    - **If a process terminates upon completion of execution.**

NOTES:

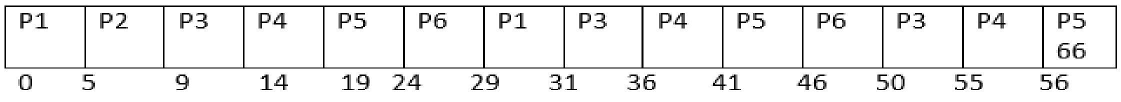
- ✓ **Arrival time (AT)**
    - The time when a process enters the ready queue to be executed
    - The time when a process arrives before it's ready to run
- ✓ **Burst time (BT)**
    - The total time a process needs to run on the CPU
    - The amount of CPU time needed to finish a process
- ✓ **Related CPU scheduling terms:**
    - **Completion time (CT):** The time when a process finishes running
    - **Turnaround time (TAT):** The time between when a process starts and when it finishes
    - **Waiting time (WT):** The time a process spends waiting in the ready queue before it gets the CPU
- ✓ **Calculating TAT and WT**
    - **TAT = CT - AT**
    - **WT = TAT – BT**
    - **BT + WT = TAT.**

PROCESS	ARRIVAL TIME	BURST TIME	COMPLETION TIME
P0	0	10	10
P1	1	6	16
P2	3	2	18
P3	5	4	22

- ✓ Turnaround time = Completion time - Arrival time
- ✓ Waiting time = Turnaround time - Burst time

PROCESS	ARRIVAL TIME	BURST TIME	COMPLETION TIME	TURN AROUND TIME	WAITING TIME
P0	0	10	10	10	0
P1	1	6	16	15	9
P2	3	2	18	15	13
P3	5	4	22	17	13

EX2



Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time
P1	0	7	31	31	24
P2	1	4	9	8	4
P3	2	15	55	53	38
P4	3	11	56	53	42
P5	4	20	66	62	42
P6	4	9	50	46	37

Average Completion Time

1. Average Completion Time = (31 +9 + 55 +56 +66 + 50) / 6
2. Average Completion Time = 267 / 6
3. Average Completion Time = 44.5

Average Waiting Time

1. Average Waiting Time = (5 + 26 + 5 + 42 + 42 + 37) / 6
2. Average Waiting Time = 157 / 6
3. Average Waiting Time = 26.16667

Average Turn Around Time

1. Average Turn Around Time = (31 + 8 + 53 + 53 + 62 + 46) / 6
2. Average Turn Around Time = 253 / 6
3. Average Turn Around Time = 42.16667

❖ FCFS

- ✓ **FCFS stands for First-Come-First-Serve**. FCSF is an OS scheduling algorithm that executes requests and processes automatically.
- ✓ **The OS stores these processes in the form of a queue in order of their arrival.**
- ✓ **Processes requesting the CPU first get the CPU allocation first** in this easy and simple algorithm with the help of a **FIFO queue**.
- ✓ FCFS is a simple and easy-to-implement CPU scheduling algorithm. In this algorithm, **those processes that request the CPU first acquire the allocation of the CPU first.**

- ✓ As soon as the process comes in the ready queue, its Process Control Block (PCB) is associated with the tail of the queue.
- ✓ **When the CPU becomes free from the first process, it should be allocated to the next process at the start of the queue.**
- ✓ Characteristics of First Come First Serve Method
    - This method is quite easy to use and implement.
    - This method **supports pre-emptive and non-preemptive** scheduling algorithms.
    - Jobs get executed on a first-come, first-serve basis always.
    - The usual waiting time is pretty high and It is poor in performance.
- ✓ **Advantages and Disadvantages of FCFS**
- ✓ **Advantages of FCFS:**
    1. The job that comes first is served first.
    2. It is the CPU scheduling algorithm's simplest form.
    3. It is quite easy to program.

- ✓ **Disadvantages of FCFS:**
    1. In FCFS, the Average Waiting Time is comparatively high.
    2. The processes with a shorter duration are at the back of the queue, they have to wait for the long processes to finish that are at the front of the queue.
    3. **It is not considered an ideal technique for time-sharing systems.**
    4. FCFS is not very efficient because of its simplicity.

❖ **SJN**

- ✓ **This is also known as shortest job first, or SJF.**
- ✓ **Best approach to minimize waiting time.**
- ✓ The processer should know in advance how much time process will take. Shortest Job Next (SJN) is an operating system (OS) scheduling algorithm that prioritizes processes with the shortest execution time. **It's also known as Shortest Job First (SJF) or Shortest Process Next (SPN).**
- ✓ **Key Features of SJN:**
    1. **Non-pre-emptive:** Once a process starts executing, it runs to completion unless it's finished. It doesn't pre-empt the current process in favour of shorter ones.
    2. **Shortest burst time:** SJN schedules the process with the shortest burst time (i.e., CPU time required for execution) first.
    3. **Minimizes average waiting time:** The goal is to reduce the average waiting time of processes in the ready queue.

Advantages

- ✓ SJN scheduling **minimizes the average waiting time for processes in the ready queue since shorter jobs** are given more priority reducing their waiting time and improving system performance in terms of response time and turnaround time.
- ✓ **Using the Shortest Job Next (SJN) scheduling algorithm can lead to better CPU resource utilization as it prioritizes shorter jobs**. This allows more processes to be completed faster.

Disadvantages

- ✓ longer-duration jobs may never have an opportunity to execute if there is a continuous arrival of shorter jobs.

How it Works:

- ➢ **Process Selection:** From the set of ready processes, **the process with the shortest CPU burst time is selected next for execution**.
- ➢ **Preemptive vs. Non-preemptive:** In non-preemptive SJN (SJF), once a process is running, it runs to completion. However, in a preemptive version (known as Shortest Remaining Time First, SRTF), the scheduler might preempt a currently running process if a new process arrives with a shorter burst time.

❖ **Round Robin**

- ✓ A round robin is an arrangement of choosing all elements in a group equally in some rational order, usually from the top to the bottom of a list and then starting again at the top of the list and so on.
- ✓ A simple way to think of round robin is that it is about "taking turns."
- ✓ **Each ready task has to run turn by turn in a cyclic queue for a limited time period in round-robin (RR).**
- ✓ CPU shifts to the next process after a fixed time interval known as time quantum or time-slice.
- ✓ Time slice is usually the minimum but differs from OS to OS.

Advantages

- ✓ **Every job gets a fair allocation of CPU.**
- ✓ No priority scheduling is involved.
- ✓ Total number of processes on the run queue helps assume the worst-case response time for a process.
- ✓ Context switching helps save states of preempted processes.
- ✓ Best performance in terms of average response time.

Disadvantages

- ✓ Spends more time on context switching.
- ✓ Performance depends on time quantum.
- ✓ Processes don’t have priorities.
- ✓ No special priority to more important tasks.

- ✓ **Preemptive scheduling allows a running process to be interrupted by a higher priority process, while non-preemptive scheduling requires a process to complete its current task before another process can take control.**

❖ **Priority Base Non-Preemptive**

- ✓ As the name suggests, the scheduling depends upon the priority of the processes rather than its burst time.
- ✓ So, the processes, in this case, must also have the priority number in its details on the basis of which the OS will schedule it.
- ✓ Non-Preemptive Priority Scheduling Algorithm Example
- ✓ For example, suppose we have 4 processes: P1, P2, P3 and P4 and they enter the CPU as follows:
- ✓ Note: Here, lower the priority number, higher is the priority.

Process ID	Arrival Time (milliseconds)	Burst Time (milliseconds)	Priority number
P1	0	4	3
P2	1	2	2
P3	2	3	4
P4	4	2	1

- ✓ As per the non-preemptive priority scheduling, the processes will be executed as follows:
- ✓ Gant Chart

$_0$ P1	$_1$ P1	$_2$ P1	$_3$ P1	$_4$ P4	$_5$ P4	$_6$ P2	$_7$ P2	$_8$ P3	$_9$ P3	$_{10}$ P3 $_{11}$

- ✓ Explanation
  - There is only P1 available at time 0, so it will be executed first irrespective of the priority, and it cannot be preempted in between before its completion.
  - When it is completed at 4th-time unit, we have all P2, P3, and P4 available. So, they are executed according to their priorities.

P ID	Arrival Time	Burst Time	Priority	Completion time (milliseconds)	Turn Around Time (milliseconds)	Waiting Time (milliseconds)
P1	0	4	3	4	4	0
P2	1	2	2	8	7	5
P3	2	3	4	11	9	6
P4	4	2	1	6	2	0

- ✓ Total Turn Around Time = 4 + 7 + 9 + 2 = 22 milliseconds
- ✓ Average Turn Around Time= Total Turn Around Time / Total No. of Processes

  = 22 / 4

= 5.5 milliseconds

- ✓ Total Waiting Time = 0 + 5 + 6 + 0
  = 11 milliseconds
- ✓ Average Waiting Time = Total Waiting Time / Total No. of Processes
  = 11 / 4
  = 2.75 milliseconds

❖ **Priority Base Preemptive**

- ✓ In the priority scheduling, the processes are scheduled on the basis of their priority, and not on the basis of their burst time.
- ✓ If the preemptive mode of this scheduling is being followed, then a process with a higher priority than the currently executing process can replace the executing process.
- ✓ This can be well explained with the help of the following example.
- ✓ **Preemptive Priority Scheduling Algorithm Example**
- ✓ Suppose, we have four processes: **P1**, **P2**, **P3** and **P4**, and they enter the CPU in the following manner:

Process ID	Arrival Time (milliseconds)	Burst Time (milliseconds)	Priority number
P1	0	4	3
P2	1	2	2
P3	2	3	4
P4	4	2	1

- ✓ As per the **preemptive priority scheduling**, the processes will be executed as follows:
- ✓ Gant Chart

0 P1	1 P2	2 P2	3 P1	4 P4	5 P4	6 P1	7 P1	8 P3	9 P3	10 P3 11

- ✓ Explanation
  - There is only **P1** available at time 0, so it will be executed first irrespective of the priority until some other process with a higher priority is encountered by the OS.
  - At the beginning of the 1st-time unit, we have **P2** which has a higher priority than **P1**, so it replaces **P1**.
  - At 2nd time unit, we have the process **P3**, but its priority is less than the executing process **P2**, so **P2** keeps executing without any replacement.
  - At 3rd time unit, our **P2** has been completed and till now we have processes **P1** and **P3** left. So, **P1** which has a higher priority than **P3** is executed.

- At 4th time unit, we have process **P4** which has the highest priority of them all. So, it replaces **P1**.
- Since now we have no more processes left to arrive, the processes will now run according to their priorities and will complete their execution.

P ID	Arrival Time	Burst Time	Priority	Completion time (milliseconds)	Turn Around Time (milliseconds)	Waiting Time (milliseconds)
P1	0	4	3	8	8	4
P2	1	2	2	3	2	0
P3	2	3	4	11	9	6
P4	4	2	1	6	2	0

✓ Total Turn-Around Time = 8 + 2 + 9 + 2= 21 milliseconds

✓ Average Turn-Around Time= Total Turn-Around Time / Total No. of Processes

= 21 / 4

= 5.25 milliseconds

✓ Total Waiting Time = 4 + 0 + 6 + 0 = 10 milliseconds

✓ Average Waiting Time = Total Waiting Time / Total No. of Processes

= 10 / 4

= 2.5 milliseconds

GEETANJALI GROUP OF COLLEGES

UNIT 2- DEADLOCKS & MEMORY MANAGEMENT

- **Deadlocks**
  - Deadlocks: Definition
  - Deadlock Prevention
  - Deadlock Avoidance
  - Deadlock Detection
- **Memory Management**
  - Physical Memory and Virtual Memory
  - Memory Allocation
  - Internal and External fragmentation
  - Contiguous Memory Allocation
  - Noncontiguous Memory Allocation
  - Virtual Memory Using Paging
  - Virtual Memory Using Segmentation

PART- 1- DEADLOCK

❖ **Deadlocks**

- Deadlock is a critical issue that can occur in operating systems when **multiple processes cannot proceed because each process is waiting for a resource that is being held by another process.**
- This situation creates a standstill where **no progress can be made until the deadlock is resolved.**
- In a deadlock, each process is stuck in a waiting state, unable to proceed with its execution.
- Deadlock usually occurs in systems where multiple processes **compete for limited resources such as CPU time, memory or input or output devices.**
- Deadlocks in operating systems can be classified based on the type of resources and the situations causing them. Here are the four main types of deadlock:
  1. **Resource Deadlock**
     - This occurs when processes compete for non-shareable resources and hold them while waiting for additional resources.
  2. **Communication Deadloc**k
     - This type arises when processes are waiting for messages or signals from each other to proceed, but none can send or receive due to cyclic dependencies.

3. **Thread Deadlock**
    - Occurs in multithreading environments when threads within a single process block each other by competing for shared resources.
4. **Circular Wait Deadlock**
    - This is a generic form where a cycle of processes exists, and each process is waiting for a resource held by the next process in the cycle.

Advantages of Deadlock in OS

- Deadlocks can be useful in certain scenarios, such as processes that perform a single burst of activity and do not require resource sharing.
- Deadlocks can provide simplicity and efficiency in systems where the correctness of data is more important than overall system performance.

Disadvantages of Deadlock in OS

- Deadlocks can cause system crashes, frozen applications, and unresponsive user interfaces, leading to a poor user experience.
- Deadlocks may result in delays in process initiation and overall system performance degradation.

- **Deadlock Prevention**
    - ✓ To prevent a deadlock before it occurs the system checks every transaction before execution. **This way if there is even a small chance for deadlock to occur, a process is not allowed to execute.**
    - ✓ Deadlock prevention is focused on removing the chance of Deadlocks by making sure that one of the required conditions for Deadlock is always avoided.
- **Deadlock Avoidance**
    - ✓ This method involves the system making real-time resource allocation decisions to maintain its safety.
    - ✓ **The Banker's algorithm is a method for deadlock avoidance.**
    - ✓ **It allocates resources to processes only if the system remains in a safe state.**
    - ✓ It checks if resource requests can be granted without causing a deadlock.
    - ✓ Example: Imagine a bank with a fixed amount of money (resources) and multiple customers (processes) making withdrawal requests.
    - ✓ The bank ensures that it always has enough resources to satisfy customer demands without causing insolvency (deadlock).

- **Deadlock Detection**

  The resource scheduler detects a deadlock occurrence and helps the OS to track all the resources' activities. Following methods can help resolve deadlock:
  - ✓ The OS **terminates all the processes involved in deadlock.** But this results in the destruction of all the progress made by processes.
  - ✓ The OS **preempts resources from one process and gives it to other processes** until the deadlock resolves.

PART – 2 - MEMORY MANAGEMENT

❖ **Memory Management**
- ✓ Memory management is how a computer allocates the main memory for an operating system (OS), applications, and any other operational processes.
- ✓ Memory management makes computer processes more efficient by allocating memory to processes that need more and releasing memory from applications or programs that may no longer use it. This allocation allows the central processing unit (CPU) to optimize data sending and instruct computer processes.

- **Physical Memory and Virtual Memory**
  - **Virtual memory**
    1. **Virtual memory is a memory management technique that allows a computer to use more memory than is physically available.**
    2. This is done by dividing the memory into pages, and then storing the pages in different locations, such as RAM and disk space.
    3. **The operating system keeps track of which pages are in memory and which pages are on disk.**
    4. When a program needs to access a page that is not in memory, the operating system will swap the page in from disk.
  - **Physical memory**
    1. **Physical memory is the actual memory that is installed on a computer. It is made up of random-access memory (RAM), which is the fastest type of memory.** Physical memory is used to store the operating system, programs, and data that are currently being used by the computer.
    2. The main difference between virtual memory and physical memory is **that virtual memory is a logical concept, while physical memory is a physical resource.**

Feature	Virtual Memory	Physical Memory
Definition	A memory management technique	The actual memory that is installed on a computer
Purpose	Allows a computer to use more memory than is physically available	Stores the operating system, programs, and data that are currently being used by the computer
Location	RAM and disk space	RAM
Access speed	Slower than physical memory	Faster than virtual memory

NOTE: Internal fragmentation refers to wasted space within an allocated memory block because the allocated size is larger than the requested size, while external fragmentation refers to wasted space between allocated blocks, making it difficult to find contiguous free space for new processes.

- **Memory Allocation**
  - ✓ **Memory allocation in an operating system refers to the process of assigning blocks of memory to different tasks (programs, processes, or data) that need memory during execution.**
  - ✓ Proper memory management ensures that each process has enough memory to execute, and that the system as a whole runs efficiently without running out of memory or encountering fragmentation issues.
  - ✓ There are **two main goals** in memory allocation:
    1. **Efficient utilization** of memory.
    2. **Isolation and protection** between processes, so one process doesn't accidentally overwrite another process’s memory.

  - ✓ **Types of Memory Allocation**

    There are **two primary types of memory allocation** in operating systems:

    1. Contiguous Memory Allocation
    - ▪ In contiguous memory allocation, **each process is assigned a single contiguous block of memory.** This means that memory for each process is allocated in one continuous chunk, and the memory space for each process is directly next to each other in the memory.
    - ▪ **Advantages:**
      - **Simple and fast access:** It’s easy to access because it’s a single, continuous block of memory.
      - **Low overhead:** There's minimal overhead in managing the memory blocks.
    - ▪ **Disadvantages:**

- **Fragmentation:** It may lead to **external fragmentation**, where there are small gaps of unused memory scattered throughout the system.
- **Limited flexibility:** It may not efficiently use memory when processes require varying amounts of memory.

- **Example:** In early operating systems like MS-DOS, contiguous allocation was commonly used.

2. Non-Contiguous Memory Allocation

- In non-contiguous memory allocation, **memory for processes is allocated in scattered chunks across the system's memory space. The process can be divided into multiple segments, which might not be adjacent to each other.**
- **Types of Non-Contiguous Allocation:**
  - **Paging**: In paging, the process is divided into fixed-size blocks called **pages**, and physical memory is divided into blocks of the same size, called **frames**. The operating system maintains a page table to map the pages to frames.
    Advantages:
    - **No external fragmentation**: Since memory is divided into fixed-size blocks, it's easier to fit processes into the available memory.
    - **Efficient utilization of memory**: It reduces wastage of memory and can make use of available memory more efficiently.

    Disadvantages:
    - **Internal fragmentation**: There might be some unused space within the frames (if a process doesn't completely fill a frame).
    - **Overhead**: Maintaining a page table adds some complexity and overhead.
  - **Segmentation**: In segmentation, the process is divided into **variable-sized segments** based on logical divisions, like code, data, or stack. Each segment is stored in non-contiguous memory.
    Advantages:
    - **Logical division**: It allows a process to be divided in a way that reflects its logical structure.
    - **No internal fragmentation**: Segments are allocated according to their logical size.

    Disadvantages:

- **External fragmentation**: Like contiguous allocation, segmentation can suffer from external fragmentation if segments cannot be allocated due to gaps in memory.
- **Overhead**: Managing segments can involve additional complexity.

- **Example:** Modern operating systems like Linux and Windows use paging for memory management.

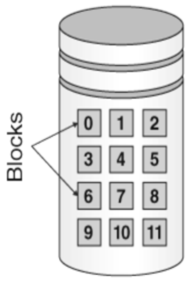
Other Memory Allocation Techniques

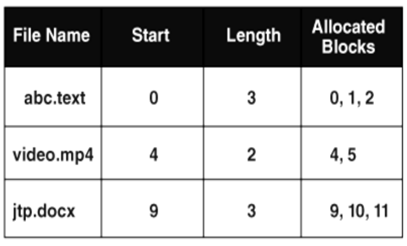
3. Dynamic Memory Allocation

- Dynamic memory allocation occurs at runtime, as opposed to static allocation which is determined at compile time. It allows memory to be allocated and freed as needed during program execution.
- **Heap memory** is commonly used for dynamic memory allocation in languages like C (via `malloc`, `free`), C++, and Java (via the **garbage collector**).
- **Advantages:**
  - **Flexible**: Memory is allocated only when required, and it can be resized.
  - **Efficient**: Useful in programs where the memory requirement may change during execution.
- **Disadvantages:**
  - **Memory leaks**: If memory is not properly freed, it can lead to memory leaks.
  - **Fragmentation**: If memory is allocated and deallocated frequently, it may lead to fragmentation within the heap.

Type	**Description**	**Advantages**	**Disadvantages**
Contiguous Allocation	Each process is allocated a contiguous block of memory.	Simple and fast access.	External fragmentation fixed size.
Non-Contiguous Allocation	Memory is allocated in scattered blocks (paging or segmentation).	No external fragmentation.	Internal fragmentatio (paging), externa fragmentation (segmentation).
Dynamic Allocation	Memory is allocated and freed during runtime (e.g., via heap).	Flexible memory usage.	Memory leaks fragmentation.

- **Contiguous Memory Allocation**
  - ✓ Contiguous memory allocation refers to a memory management technique in which whenever there occurs a request by a user process for the memory, one of the sections of the contiguous memory block would be given to that process, in accordance with its requirement.



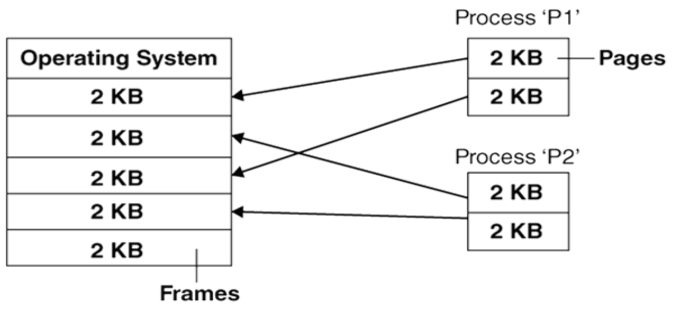


File Name	Start	Length	Allocated Blocks
abc.text	0	3	0, 1, 2
video.mp4	4	2	4, 5
jtp.docx	9	3	9, 10, 11

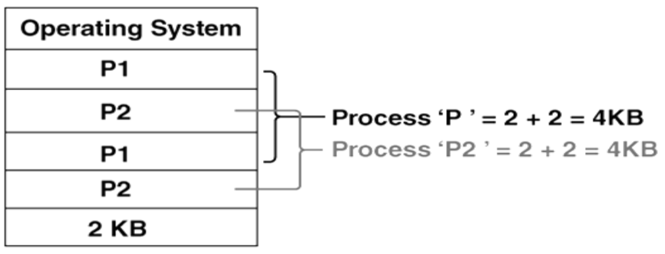
Directory

  - ✓ As you can see in the illustration shown above, three files are there in the directory.
  - ✓ The starting block, along with the length of each file, is mentioned in the table. Thus, we can see in this table that all the contiguous blocks get assigned to every file as per their need.
  - ✓ **Types of Partitions**
  - ✓ Contiguous memory allocation can be achieved when we divide the memory into the following types of partitions:
    - o **Fixed-Sized Partitions**
      - ▪ Another name for this is static partitioning. In this case, **the system gets divided into multiple fixed-sized partitions. In this type of scheme, every partition may consist of exactly one process.** This very process limits the extent at which multiprogramming would occur, since the total number of partitions decides the total number of processes. Read more on fixed-sized partitions here.
    - o **Variable-Sized Partitions**
      - ▪ **Dynamic partitioning is another name for this.** The scheme allocation in this type of partition is done dynamically. Here, **the size of every partition isn't declared initially. Only once we know the process size, will we know the size of the partitions.** But in this case, the size of the process and the partition is equal; thus, it helps in preventing internal fragmentation.

- **Noncontiguous Memory Allocation**
  - ✓ In the case of non-contiguous memory allocation, the processes would acquire the overall space in the memory.
  - ✓ **They are not present in a single place but at different locations as per the process requirement.**
  - ✓ This particular technique reduces overall memory wastage, which leads to external or internal fragmentation.
  - ✓ **Fragmentation and paging are the two ways that allow the physical address space of a process to be non-contiguous.**
  - ✓ In the case of non-contiguous allocation, **the OS needs to maintain the table that is known as the Page Table for every process that consists of the base address of each of the blocks acquired by the processes in memory space.** In this type of memory allocation, all the different parts of a given process get allocated to different places in the main memory.
  - ✓ Consider an empty main memory with each frame of 2 KB, along with two processes, P1 and P2, that are 2 KB each. The diagram given below will help you understand this in a better way:

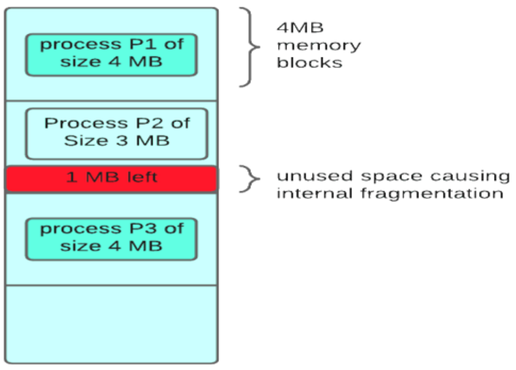


  - ✓ Resolving main memory:



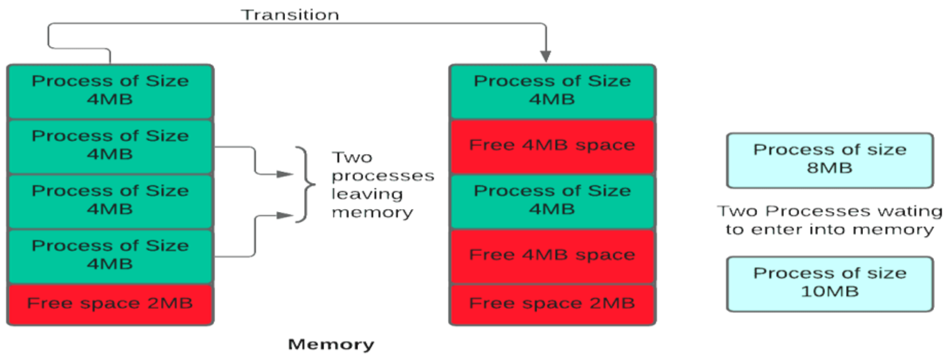
  - ✓ Thus, we can conclude that the process of paging allows the memory address space of the given process to be non-contiguous.
  - ✓ Thus, paging is comparatively flexible since only pages in a process are moved. It allows the residing of more processes in the main memory than in the contiguous allocation.

- **Internal and External fragmentation**
  - ✓ Fragmentation is an unwanted problem that arises due to memory allocation and deallocation.
  - ✓ **Internal Fragmentation**
    - Internal fragmentation occurs when the memory is distributed into fixed-sized blocks. If the memory allocated to the process is slightly larger than the memory demanded, then the difference between allocated and demanded memory is known as internal fragmentation.
    - The unused space is considered fragmented since it cannot be used for other processes.
    - For Example: Let's consider a fixed partitioning scheme where the memory blocks are of fixed sizes(say 4MB each). Now, suppose a process of size 3 MB comes and occupies a block of memory. So, the 1MB space in this block is free and can't be used to allocate it to other processes. This is called internal fragmentation. The diagram below shows memory division, where each memory block is size 4MB.



  - ✓ **External Fragmentation**
    - If there is enough space within the memory to meet the memory demand of a process, then in such a situation external fragmentation occurs.
    - The holes or vacant spaces created by leaving process mark empty spaces in the memory, and the non-contiguous spaces are not a good fit to accommodate a new upcoming process.
    - For example: Suppose there are many processes present in the main memory, and two processes each of size 4 MB leave the memory. A total of 8MB + 2MB = 10MB memory space is free now.
    - **Now a new process equal to the sum of total available space can't be loaded into the main memory as the unused space is not in a contiguous fashion**. This is known as external fragmentation.

- In the diagram given below, two processes of sizes, 8MB and 10MB try to enter into the memory, but neither of them can be loaded as no **contiguous** space of 8MB or 10MB is available.



Parameter	Internal Fragmentation	External Fragmentation
Definition,	The difference between the memory space needed and the assigned memory is considered as internal fragmentation.	When there are empty spaces among the non-contiguous memory blocks that cannot be assigned to any process, this problem is considered as external fragmentation.
Memory Block size	In internal fragmentation, the memory blocks are of fixed size.	In external fragmentation, the memory blocks are of varying sizes.
Occurrence	Internal fragmentation occurs when we divide physical memory into contiguous mounted-sized blocks and allocate memory for a process that may be larger than the amount of memory requested. As a result, the unused allocated space remains and cannot be used by other processes.	External fragmentation occurs when a process or processes are removed from the main memory and the free spaces created are too small to fit a new process.
Solution	The use of a dynamic partitioning scheme and best-fit block search are the solutions that can reduce internal fragmentation.	Compaction, paging, and segmentation are the solutions for external fragmentation.

- **Virtual Memory Using Paging**
  - ✓ Virtual memory is a memory management technique that **allows a computer to compensate for physical memory shortages by temporarily transferring data from random access memory (RAM) to disk storage.**

- ✓ This allows for larger programs to run on systems with less RAM than required. Virtual memory enables the system to create an illusion of a larger memory pool than what is physically available by swapping data in and out of the disk.
- ✓ Paging is one of the key techniques used to implement virtual memory. Here's how it works:

1. **Basic Concept of Paging:**
   - In paging, the **virtual memory is divided into fixed-size blocks called pages.**
   - **Physical memory (RAM) is divided into blocks of the same size, called page frames.**
   - When a program is executed, its pages are loaded into the available page frames in physical memory.

2. **How Paging Works:**
   - When a process is running, it accesses memory addresses through virtual addresses.
   - The page table maintains the mapping between the virtual pages and the physical frames in memory.
   - The virtual address generated by the CPU is divided into two parts:
     - **Page number**: This identifies which page of the process's virtual memory the address belongs to.
     - **Offset**: This specifies the exact location of the data within that page.
3. **Page Table:**
   - The page table is a critical component that maps virtual pages to physical page frames. Each process has its own page table.

4. **Page Fault:**
   - ✓ A **page fault** occurs when the CPU tries to access a page that is not currently in physical memory.

- **Virtual Memory Using Segmentation**

- ✓ Virtual Memory Using Segmentation is another memory management technique, distinct from paging, which **divides a program's memory into variable-sized segments, such as code, data, stack, and heap.**
- ✓ Unlike paging, which breaks memory into fixed-size blocks (pages), segmentation provides a more logical view of memory, making it easier to organize and protect different types of data.

1. **Segmentation:**
   - In segmentation, a program's memory is divided into **segments** of various lengths, depending on the nature of the data.

   - These segments could represent different parts of a program, such as:
     - **Code Segment**: Stores executable instructions (the actual code).
     - **Data Segment**: Stores global and static variables.
     - **Heap Segment**: Stores dynamic memory allocated during runtime (e.g., through malloc in C).
     - **Stack Segment**: Stores function call stacks, local variables, etc.
2. **Address Structure in Segmentation:**
   - **Logical Address**: In a system using segmentation, the logical address (virtual address) consists of two parts:
     - **Segment number**: Identifies the specific segment (e.g., code, data, heap).
     - **Offset**: Specifies the exact location within the segment.
3. **Segment Table:**
   - Each process has a segment table, which stores the base address and limit (length) for each segment. **The base address is where the segment starts in physical memory, and the limit ensures that memory access is within the bounds of the segment.**

How Segmentation Works:

1. **When a program is loaded into memory, each segment (e.g., code, stack, heap) is loaded into different areas of physical memory.**
2. The CPU generates a logical address when a program accesses a memory location. This logical address consists of:
   - **Segment number: Identifies the segment.**
   - **Offset: Identifies the position within the segment.**
3. The segment number is used to find the segment's base address in the segment table.
4. The offset is added to the base address to form the physical address in memory.
5. If the offset is beyond the limit (i.e., accessing memory outside the segment), a segmentation fault occurs, which leads to a runtime error or exception.

UNIT 3- GETTING STARTED WITH UNIX, UNIX SHELL COMMAND

Getting Started with Unix

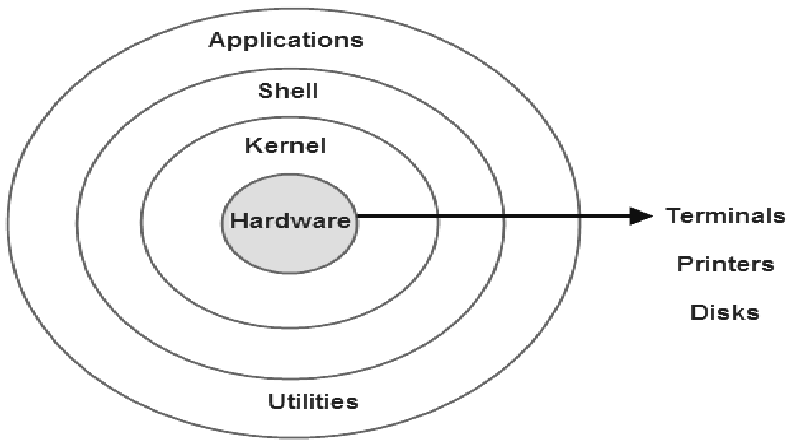
- Unix Architecture
- Unix Features
- Types of Shell (C, Bourn, Korn)
- Unix File System
- Types of Files
  - Ordinary Files
  - Directory Files
  - Device Files
- Unix File & Directory Permissions

Unix Shell Command

- Connecting Unix Shell : Telnet
- Login Commands passwd, logout, who, who am i, clear,uname
- File / Directory Related Command ls, cat, cd, pwd, mv, cp, ln, rm, rmdir, mkdir, chmod, chown, chgrp, find, more, less, head, tail, wc, touch, stat, alias, type
- Operators in Redirection & Piping , <>, |
- Finding Patterns in Files grep, fgrep, egrep
- Working with columns and fields cut, paste, join
- Tools for sorting: sort, uniq
- Comparing files: cmp, comm, diff
- Changing Information in Files: tr, sed
- Examining File Contents: od
- Tools for mathematical calculations: bc, factor
- Monitoring Input and Output: tee, script
- Tools for Displaying Date and Time: cal, date
- Communications: telnet, ping
- Process Related Commands: ps, sleep

❖ **Unix Architecture**

- ✓ Ken Thompson, Dennis Ritchie, and their team developed an Operating System called UNIX (Uniplexed Information Computing System) in 1970 in the AT&T laboratory. So, UNIX dates from 1970 and it is pretty old. However, most of us might not have used UNIX in our lives.
- ✓ The reason is pretty simple. UNIX was developed for the programmers/software developers to build software. It was not initially created for use by the common public.
- ✓ However, from UNIX, we got many other Operating Systems like Ubuntu, Solaris, etc.
- ✓ These operating systems are also used more commonly by the public, especially Ubuntu.



- ✓ The architecture of this operating system is four layered. It consists of Hardware, Kernel, System Call interface(shell) and application libraries/tools, utilities, etc…**The kernel controls the hardware of the computer and resides at the core of the architecture.**
- ✓ System calls acts as the interface between the kernel and other libraries. These libraries include general functions and built on top of the system calls.
- ✓ Shell is a special application that provides an interface to the other applications of the architecture.
- ✓ **Kernel**
  - For this operating system, Kernel is the **central core that interacts directly with the hardware of the system.** The main functions of Kernal are-
    - **Computer hardware such as memory, disc, printers, etc.. are controlled by the kernel.**
    - The kernel **schedules the processes, control and executes various user-defined tasks.**

    - Manages the data storage and control the computer accesses by several users.

- ✓ **Shell**
    - **It is the** interface between the user and the kernel. **Users can** interact with the shell using shell commands.
    - **Shell has two main responsibilities which include** interpreting the commands given by the users and execute them using the kernel, providing programming ability to the users to write shell commands **for a shell script to perform specific tasks.**
- ✓ **Commands**
    - **Some of the major categories of commands used by the Unix operating system are – 'sh' – shell commands providing a primary user interface, 'utilities' forming the core toolkit of Unix commands includes sub-categories such as system utilities supporting administrative tools and User utilities for environment management tools.**
    - **It also has commands for general purpose applications such as document-formatting and typesetting. Some Unix systems also include packages such as TeX and Ghostscript. It also supports inter-system communication as well as inter-user communication.**

❖ **Unix Features**

- ✓ **Multitasking Operating System:**
    - In UNIX, a user can run multiple tasks/processes simultaneously. For example, a user can run a text editor and can also open a web browser at the same time.
    - However, it is important to note that "multitasking" is not what actually happens behind the scenes.
    - The UNIX operating system can only execute one process at a time.
- ✓ **Multiuser Operating System:**
    - In UNIX, multiple users can run their own tasks/processes simultaneously. Hence, UNIX is called a multiuser OS. However, the case is the same as multitasking.
    - Each user executes his/her own process for quantum units of time and then the process gets context switched and the other user's process starts executing.
    - Then, after some time, the first user's process will again be executed from where it left. This happens really fast and creates an illusion of a multiuser operating system.
- ✓ **Small Commands and Pipelining:**
    - The UNIX programmer-friendly software as it relies more on the commands rather than GUI (Graphical User Interface).

- For example, the ls command is used to list all the files and directories inside the current directory in which we are working.
- Another example is the wc command, which is used to count words and bytes for files.
- Pipelining of commands is a concept of connecting multiple small commands using a pipe "|" symbol to perform complex instructions. For example, the above 2 commands can be pipelined as "$ ls | wc". This command can count the number of files in the directory.

✓ **The Powerful Unix Toolkit:**

- UNIX has a toolkit, that is, it has multiple applications that help in performing multiple tasks. For example, UNIX has compilers and interpreters, text manipulation tools, system administration tools, etc.

❖ **Types of Shell (C, Bourn, Korn)**

✓ Whenever a user logs in to the system or opens a console window, the kernel runs a new shell instance. **The kernel is the heart of any operating system.**

✓ It is responsible for the control management, and execution of processes, and to ensure proper utilization of system resources.

✓ **A shell is a program that acts as an interface between a user and the kernel.**

✓ Through a shell, we can execute programs and utilities on the kernel.

✓ Hence, at its core, **a shell is a program used to execute other programs on our system.**

1. The Bourne Shell (sh)

- Developed at AT&T Bell Labs **by Steve Bourne**, the Bourne shell is regarded as the **first UNIX shell ever**.
- It is denoted as sh. It gained popularity due to its compact nature and high speeds of operation.
- **This is what made it the default shell for Solaris OS.**
- However, the Bourne shell has some major drawbacks.
  - It **doesn't have in-built functionality to handle logical and arithmetic operations.**
  - Also, unlike most different types of shells in Linux, the Bourne shell **cannot recall previously used commands.**
- The complete path-name for the Bourne shell is /bin/sh and /sbin/sh. By default, it uses the prompt # for the root user and $ for the non-root users.

2. The GNU Bourne-Again Shell (bash)

- More popularly known as **the Bash shell**, the GNU Bourne-Again shell was **designed to be compatible with the Bourne shell.**
- It includes useful features from different types of shells in Linux such as Korn shell and C shell.
- It **allows us to automatically recall previously used commands and edit them with help of arrow keys,** unlike the Bourne shell.
- **The complete path-name for the GNU Bourne-Again shell is /bin/bash.**
- By default, it uses the prompt *bash-VersionNumber#* for the root user and *bash-VersionNumber$* for the non-root users.

3. The C Shell (csh)

- The C shell was **created at the University of California by Bill Joy**.
- It is **denoted as csh**.
- It was developed to **include useful programming features like in-built support for arithmetic operations and a syntax similar to the C programming language.**
- Further, it combined command history which was missing in different types of shells in Linux like the Bourne shell.
- The complete path-name for the C shell is /bin/csh. By default, it uses the prompt *hostname#* for the root user and *hostname%* for the non-root users.

4. The Korn Shell (ksh)

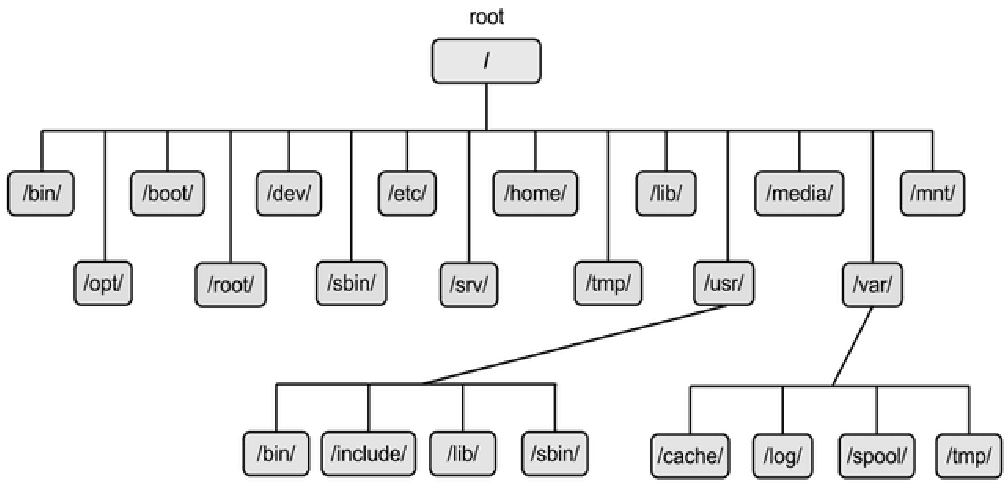
- The Korn shell was **developed at AT&T Bell Labs by David Korn**, **to improve the Bourne shell**.
- It is **denoted as ksh.**
- The Korn shell is essentially a **superset of the Bourne shell.**
- Besides supporting everything that would be supported by the Bourne shell, it provides users with new functionalities.
- **It allows in-built support for arithmetic operations while offering interactive features which are similar to the C shell.**
- The Korn shell runs scripts made for the Bourne shell, while offering string, array and function manipulation similar to the C programming language. It also supports scripts which were written for the C shell.
- The complete path-name for the Korn shell is /bin/ksh. By default, it uses the prompt *#* for the root user and *$* for the non-root users.

5. The Z Shell (zsh)

- The Z Shell or zsh is a sh shell extension with tons of improvements for customization. If you want a modern shell that has all the features a much more, the zsh shell is what you're looking for.
- **Some noteworthy features of the z shell include:**
  - Generate filenames based on given conditions
  - Plugins and theming support
  - Index of built-in functions
  - Command completion

❖ **Unix File System**

- ✓ Unix File System is **defined as the framework that organizes and stores a large amount of data that can be handled easily**.
- ✓ It consists of elements like a file, a collection of related data that can be viewed logically as a stream of bytes (or characters).
- ✓ **The file system consists of two main components which are files and directories.**
- ✓ The entire file system follows a hierarchy in which directories act as a special file containing multiple files.
- ✓ The highest level directory in the entire hierarchical structure is known as the root.
- ✓ **The root is denoted as ' / '.** There can be many sub-directories under this directory.
- ✓ The following diagram explains the hierarchy:



- ✓ Some directories and their descriptions are as follows:
  - **/bin:** short for binaries, this is the directory where many commonly used **executable commands reside**
  - **/dev**: contains **device-specific files**
  - **/etc**: contains **system configuration files**
  - **/home**: contains **user directories and files**
  - **/lib**: contains **all library files**

- **/mnt**: contains **device files related to mounted devices**
- **/proc**: contains **files related to system processes**
- **/root**: the root users' home directory (note this is different than ‘ / ‘)
- **/sbin**: **binary files of the system** reside here.
- **/tmp**: **storage for temporary files** that are periodically removed from the file system
- **/var**: It is a short form for ‘variable’, a place for files that may often change

❖ **Types of Files**

- **Ordinary Files**
  - ✓ An ordinary file is a file on the system that contains data, text, or program instructions.
  - ✓ Used to store your information, such as some text you have written or an image you have drawn. This is the type of file that you usually work with.

- **Directory Files**
  - ✓ Directories **store both special and ordinary files**. For users familiar with Windows or Mac OS, UNIX directories are equivalent to folders.
  - ✓ A directory file contains an entry for every file and subdirectory that it houses.
  - ✓ 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)**

- **Device Files**
  - ✓ A device file in Linux is an interface for various devices in the form of files.
  - ✓ **Device files are special files that exist as an interface for your system to interact with hardware.**
  - ✓ **They are located in the /dev directory, which is a temporary virtual filesystem that is created by Linux on boot.**
  - ✓ Device files are also known as device special files. Device files are employed to provide the operating system and users an interface to the devices that they represent.

- ❖ **Unix File & Directory Permissions**
  - ✓ UNIX commands allow you to set permissions. File by file, allowing you to control who can read a file, write to a file, or view a file on a Web page.
  - ✓ Unix file and directory permissions are a way of controlling who can read, write, or execute a file or directory on a Unix-like operating system (such as Linux or macOS).
  - ✓ **Key Concepts:**

    1. **Users and Groups:**
       - **Owner:** The user who created the file or directory, or the user assigned as the file's owner.
       - **Group:** A set of users who are assigned a group, which can be used to manage file access permissions for multiple users.
       - **Others:** All users who are neither the owner nor part of the group associated with the file or directory.
    2. **Types of Permissions:**
       - **Read (r):** Allows the user to open and read the file's contents or list the files in a directory.
       - **Write (w):** Allows the user to modify the file's contents or add/remove files in a directory.
       - **Execute (x):** Allows the user to run the file as a program or script, or navigate into the directory.
  - ✓ **Permission Notation:**
    - Permissions are typically shown in the following format:
      -rwxr-xr--
      - **The first character represents the file type** (e.g., - for a regular file, d for a directory).
      - **The next three characters represent the owner's permissions** (read, write, execute).
      - **The next three characters represent the group's permissions** (read, write, execute).
      - **The final three characters represent others' permissions** (read, write, execute).

      For example:
      - **-rwxr-xr--:** The owner has read, write, and execute permissions, the group has read and execute permissions, and others have read-only permissions.
  - ✓ **Permission Codes:**
    Each permission can be represented by a number, known as octal representation:
    - r (read) = 4
    - w (write) = 2
    - x (execute) = 1
    - No permission = 0

You can sum these values to get the permission code:

- rwx (read, write, execute) = 4 + 2 + 1 = 7
- rw- (read, write, no execute) = 4 + 2 + 0 = 6
- r-- (read, no write or execute) = 4 + 0 + 0 = 4

Example permission rwxr-xr-- translates to:

- Owner: rwx = 7
- Group: r-x = 5
- Others: r-- = 4

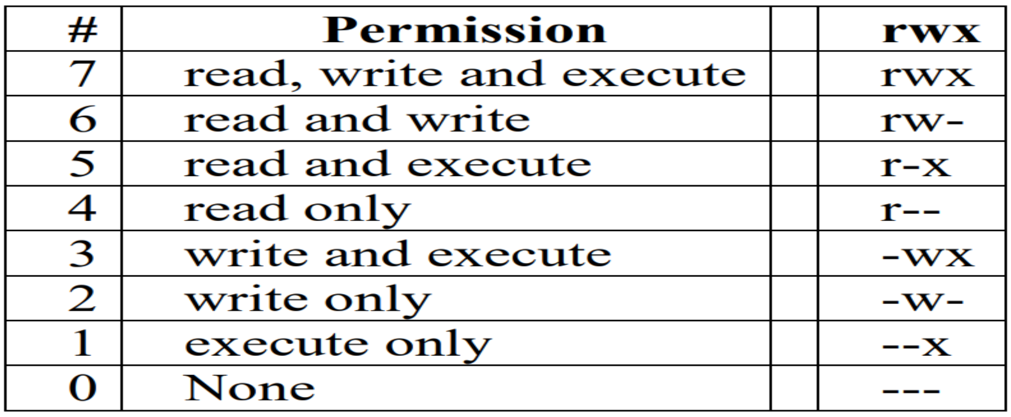
So the octal representation is 754.

✓ **Changing Permissions: chmod command**

- To change file or directory permissions, you use the chmod command.
  - **Octal mode:** You specify the permissions using numeric values.
    - Example: chmod 755 file.txt (Owner has full permissions, group and others have read and execute).
    - Example: chmod 644 file.txt (Owner has read and write, group and others have read-only).

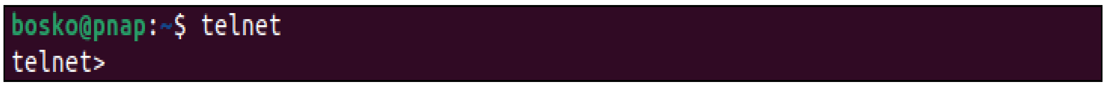
✓ **Changing Ownership: chown command**

- The chown command is used to change the owner and/or group of a file or directory.
- Example:
  - **chown user file.txt (Changes the owner of the file to user).**
  - **chown user:group file.txt (Changes the owner to user and the group to group).**
- **Changing Group: chgrp command**
- The chgrp command is used to change the group of a file or directory.
- Example:
  - **chgrp group file.txt (Changes the group of the file to group).**



#	Permission		rwx
7	read, write and execute		rwx
6	read and write		rw-
5	read and execute		r-x
4	read only		r--
3	write and execute		-wx
2	write only		-w-
1	execute only		--x
0	None		---

- **Connecting Unix Shell: Telnet**
    - The telnet command (short for teletype network) utilizes the telnet network protocol to connect to remote machines and manage them through a TCP/IP network. The protocol uses port 23 to establish a connection and provides a way to manage remote systems using the CLI.
    - **How to Use telnet in Linux**
        1. A vital prerequisite for using telnet is to have it installed on both the local and remote machine, and that the default port 23 is allowed through the firewall on the remote machine.
        2. The syntax for the telnet command is:
            i. telnet [options] [remote_server_address] [port]
            ii. The [options] and [port] parameters are optional. For [remote_server_address], telnet accepts symbolic and numeric addresses.
            iii. Running the command without options or without specifying an address opens the telnet interactive mode:



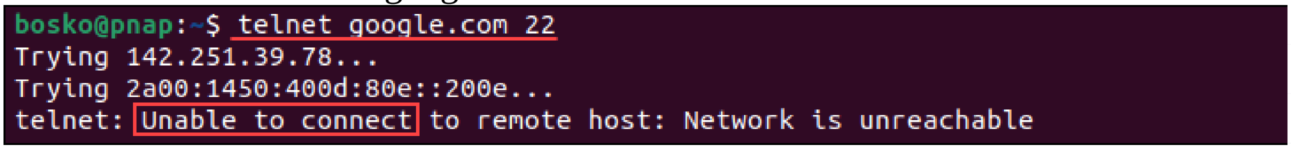
```
bosko@pnap:~$ telnet
telnet>
```

3. Use the interactive shell to connect to remote servers, print the connection status, etc.
4. To end a session and exit telnet, run:
    i. quit

Check for Open Ports

Check if a certain port is answering any calls by specifying the port number in the command. Doing so allows you to see what's going on in a network and if a port is responsive or not.

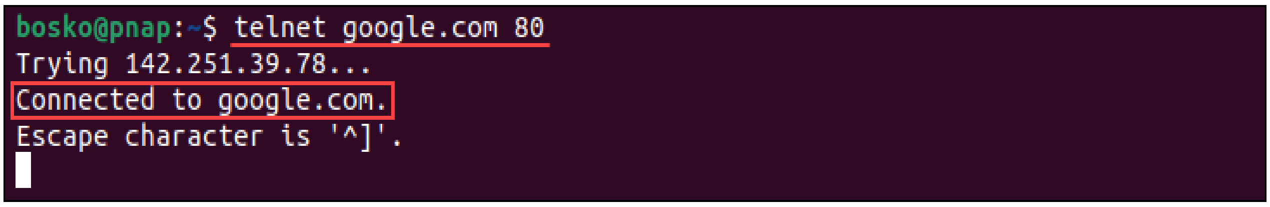
1. Use the following syntax:
    - telnet [server_address] [port]
2. Not specifying a port number defaults to port 23. For example, to check if port 22 is open on a server, run:
    - telnet google.com 22



```
bosko@pnap:~$ telnet google.com 22
Trying 142.251.39.78...
Trying 2a00:1450:400d:80e::200e...
telnet: Unable to connect to remote host: Network is unreachable
```

The connection breaks, which means that the specified port is not open.

3. However, trying port 80 produces a different result:
    - telnet google.com 80



```
bosko@pnap:~$ telnet google.com 80
Trying 142.251.39.78...
Connected to google.com.
Escape character is '^]'.
```

The output shows that the port is open since the connection was established.

- **Login Commands passwd, logout, who, who am i, clear,uname**

1. **Passwd**
   - The passwd command is used to change password.
   - Administrator can change password of any user using this command but the ordinary user can change their password.
   - **General rules for entering a valid password**
     - It must contain at least one numeric or special character within first eight characters.
     - It must contain two alphabetic characters
     - The length of password must be at least 3 characters and not more than 255 characters.
     - It must not be same as old password. It must differ by at least 3 positions.
     - It must not be same as login id
     - These passwords are case sensitive.
   - **Options**
     - -d, ---delete Delete a user's password (make it empty).
     - -l, ---lock Lock the named account. This option disables an account by changing the password
     - -u, ---unlock Unlock the account.
     - Syntax : $passwd
     - Example : $passwd
       - Output :
         - old password :
         - new password :
         - confirm password :
2. **logout / exit**
   - Def : This command terminate the session and quite the current login.
     1. Syntax : $logout/$exit
     2. Example : $logout/$exit
   - Output : Output may be either login screen or login command depend upon OS.

3. **who**
   - Def. : This command displays all the users who are currently log into the system. It returns the username, the terminal (Device id) and login

time of the user.
- Syntax : $who
- Example : $who

4. **who am I**
   - Def : Instead of getting information about all user who are currently login, if you want to get information about yourself then who am I is used.
   - Who command with the arguments "am I" gives the details of the user who executes this command.
   - Syntax : $who am i
   - Example : $who am i

5. **Clear**
   - Def : This command clears the terminal screen. It takes no arguments or ignores any command line arguments. It works similar to that of cls command in Dos.
   - Syntax : $clear
   - Example : $clear

6. **uname**
   - uname (short for unix name) is a computer program in Unix and Unix-like computer operating systems that prints the name, version and other details about the current machine and the operating system running on it.
   - Syntax : $uname [options]
   - Example : $ uname –a
     - -a Prints all system information.
     - -s Prints the Kernel name.
     - -n Prints the network node hostname.
     - -r Prints the Kernel release number.
     - -v Prints the Kernel version.
     - -m Outputs the machine's architecture type.
     - -p Prints the CPU type.
     - -i Prints hardware platform type.
     - -o Prints the operating system name.

- **File / Directory Related Command**

  - **Ls**
    - ls command is used to obtain the list of all the files in the current directory.

- It provides the complete list of filenames in the current directory arranged in ASCII collating sequence (numbers first, uppercase and then lowercase).
- It lists the directories and files in the current directory.
- file and directory are stored in ascending order.
- syntax: $ ls [options]
- example: $ ls
- **Option:**
  - **$ls –l:** This command list out file and directory with other information like file permission creation date, time, size owner etc.
  - **$ls –a:** This command list out files and directory including hidden file and directory In unix hidden directories and file name start with "."
  - **$ls –l f1:** This command will list out the information for the file f1 Syntax: $ls –l file name Example: $ls –l f1
  - **$ls –r:** This command will represent files in reverse order.

- **Cat**
  - This command is used to display the contents of a file on the terminal and also to create file.
  - Cat command is used to create new file.
  - This command is also useful to view or concate one or more file.
  - Syntax : cat > [filename].
  - Example:
  - To create a file using cat, enter the command cat, followed by the > and the file name.

    -
      ```
      $ cat>txt
      hiii
      this is first file.
      create by cat command <ctrl+d>
      ```
  - For viewing multiple file data using single command use following
    - $cat f1 f2 f3
- **Cd(changing the directory)**
  - You can move around in the file system by using cd (change directory) command. It changes the current directory to the directory specified as the argument.
  - The user can move from one directory to other by changing the directory.
  - It changes the current directory to the specified directory.
  - Syntax: $cd directoryname
  - Example: cd bca/bscit

- **pwd(present working directory)**
  - This command is used to locate present working directory of the user.
  - It doesn’t have any option or argument list.
  - It displays the absolute path of the current directory.
    - Syntax: $pwd
- **Mv**
  - mv command is used to move files and directories.
  - If the destination file doesn’t exist, it will be created.
  - It is used to rename file or move group of file to different directory.
  - It doesn’t create copy of the file but it rename only. It is also use for renaming the directory.
  - Syntax: $mv oldfilename newfilename
  - $ mvsource dest
  - Example: 1) $mv file1 file2
  - **The mv command can be used to:**
    - Move (rename) one file to another file.
    - Move one or more files to a directory.
    - Move an entire directory structure.
    - If target filename already exists, it will be overwritten.
- **Cp**
  - cp is a shell command to copy files and directories.
  - This command is used to copy single file or group of the file.
  - It creates the **exact copy of the source file with a different name of the destination file.**
- **Ln**
  - This command is used to link two files.
  - When a file is copied, both the original and copy occupy separate space in the disk. UNIX allows a file to have more than one name, and yet maintain a single copy in the disk.
  - Changes in one file will also be reflected in the other. The file is said to have more than one links.
  - A file can have as many names as you want to give it, but the only thing that is common to all of them is that they all have the same i-node number.
  - This command allows one file to have multiple file names but there is only one copy of the file on the disk.
  - Syntax: $ln target_filename link_filename
  - Example: $ln file1 lnfile1

- **Rm**
  - It is used to delete the link of file or directory; it does not mean that the file physically deleted.
  - If all the links are deleted and you want to delete the file, then it asks for confirmation for delete file if you have permission to delete file then file is physically removed.
  - Syntax : $rm filename
  - Example: $rm file1
  - It can delete more than one file with a single instruction.
  - $ rm u1 u2 u3
- **Rmdir**
  - The rmdir command removes empty directories from your file system.
  - It is used to delete the link of file or directory; it does not mean that the file physically deleted.
  - If all the links are deleted and you want to delete the file, then it asks for confirmation for delete file if you have permission to delete file then file is physically removed.
  - If any specified directory is not empty, rmdir will not remove it, and will proceed to try and remove any other directories you specified.
  - If the directory contains the file then it can't be remove and give error message "directory is not empty" to the user.
  - It can also remove multiple directories using single command.
  - Syntax: $rmdir directoryname
  - Example $rmdir dira (single directory)
  - $rmdir dira dirb (multiple directory)
- **Mkdir**
  - It is use to create directory and its sub directory using a single command.
  - Syntax: mkdir [options] directoryname
  - Example $mkdir –p dira/dirb/dirc (it contain directory inside directory)
  - If system can't make directory This can happen due to these reasons:
    - The directory may already exist.
    - The user doesn't have the permission to create a directory.
- **Find**
  - It examines the directory tree to look for file matching the criteria and then takes action on the selected file.
  - It evaluates the given expression from left-to-right
  - It examines time the file in directories specified in the path list then it matches each file for one or more selection criteria and then it takes some action on those selected files.
  - Syntax: find directoryname (directory must present in your system)

  - Syntax: find –name filename (for searching particular file)
  - Example: $find bca Example: $find –name file1

- **More**
  - It is for paging output. If a file is too large for its content to fit in one screen, it will scroll off your screen when you view it with cat.
  - This sometimes happen so fast that, before you hit<ctrl- > to stop it, quite a bit of output would have scrolled off.
  - more allows a user to view a file, one screen at a time.
  - $ more f1
- **Less**
  - Less and pg are similar to more command. And less command is a Linux command, it is also supported by UNIX, because in Linux less command is more powerful than more command. $ less file1
- **Head**
  - head makes it easy to output the first part of files.
  - head, by default, prints the first 10 lines of each FILE to standard output. With more than one FILE, it precedes each set of output with a header identifying the file name.
  - If no FILE is specified, or when FILE is specified as a dash ("-"), head reads from standard input.
  - head myfile.txt
  - head -n 5 myfile.txt myfile2.txt #Displays only the first 5 lines of both files. (use –c for byte)
  - head -c 6M myfile.txt #Displays the first six megabytes.
- **Tail**
  - The tail command displays the end of the file. Like head, when use without an option it will display last ten lines of the file.
  - Syntax tail [options] [file]
  - $ tail f1.txt
- **Wc**
  - The wc command is for counting.
  - When it used without option it will count the number of characters, words and lines and display that count.
  - Syntax: wc [options] [file]
  - Example: $ wc f1
  - 30 60 276
  - Where 30 is the number of lines, 60 is the number of words, and 276 is the number of characters.
  - When it use with –c option, it will count only characters from the given file.
  - $ wc -c f1
  - $ wc -w f1 (count only words)

- **Touch**
  - The touch command can be used to create a new (empty) file or to update the last access or modification date/time on an existing file.
  - The command is used primarily when the script is checking for last date or time a function was performed.
    - -a, change the access time only
    - -c, if the file does not exist, do not create it
    - -d, update the access and modification times
    - -m, change the modification time only
- **Stat**
  - The stat command is used to print out the status of Linux files, directories and file systems.
  - Syntax: stat [OPTION]... FILE...
  - Example: stat file.txt
  - stat displays the following file information:
  - File - The name of the file.
  - Size - The size of the file in bytes.
  - Blocks - The number of allocated blocks the file takes.
  - IO Block - The size in bytes of every block.
  - File type - (ex. regular file, directory, symbolic link.)
  - Device - Device number in hex and decimal.
- **Alias**
  - Linux 'alias' command replaces one string from the shell with another string. It is a shell built-in command.
  - It converts a complicated command into a simpler command or in other words, it creates a shortcut by replacing it with the simpler one.
  - Syntax: alias newName=command name (To create alias for commands)
  - Example: alias c =clear
- **Type:** The type command is used to display information about the command type.

- **Operators in Redirection & Piping , <>, |**
  - Input and output in the Linux environment is distributed across three *streams*. These streams are:
    - **standard input** (*stdin*)
    - **standard output** (*stdout*)
    - **standard error** (*stderr*)
  - The streams are also numbered:
    - **stdin** (**0**)
    - **stdout** (**1**)
    - **stderr** (**2**)

- During standard interactions between the user and the terminal, standard input comes from the user's keyboard.
- Standard output and standard error are displayed on the user's terminal as text. Collectively, the three streams are referred to as the standard streams.

Stream Redirection

- Linux includes redirection commands for each stream. These can be used to write standard output or standard error to a file.
- If you write to a file that does not exist, a new file with that name will be created prior to writing.
- Commands with a single bracket *overwrite* the destination's existing contents.
    - **Overwrite**
        - > - standard output
        - < - standard input
        - **2**> - standard error
- Commands with a double bracket *do not* overwrite the destination's existing contents.
    - **Append**
        - >> - standard output
        - << - standard input
        - **2**>> - standard error

Pipes

- Pipes are used to redirect a stream from one program to another.
- When a program's standard output is sent to another through a pipe, the first program's output will be used as input to the second, rather than being printed to the terminal.
- Only the data returned by the second program will be displayed.
- **The Linux *pipe* is represented by a vertical bar: |**
- Here is an example of a command using a pipe:

  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.

- **Finding Patterns in Files grep, fgrep, egrep**
    - Grep is a powerful utility available by default on UNIX-based systems.
    - **The name stands for Global Regular Expression Print.**

- By using the grep command, you can customize how the tool searches for a pattern or multiple patterns in this case.
- You can grep multiple strings in different files and directories.
- The tool prints all lines that contain the words you specify as a search pattern.

How to Grep Multiple Patterns – Syntax

- The basic grep syntax when searching multiple patterns in a file includes using the grep command followed by strings and the name of the file or its path.
- The patterns need to be enclosed using single quotes and separated by the pipe symbol. Use the backslash before pipe | for regular expressions.
  - **grep 'pattern1\|pattern2' fileName_or_filePathCopy**
  - Search for a pattern in a file:
    - **grep "pattern" filename**
  - Ignore case while searching:
    - **grep -i "pattern" filename**
  - Display line numbers of matched lines:
    - **grep -n "pattern" filename**

- **egrap(Extended Global Regular Expression Print)**
  - egrep is a variant of grep that allows the use of extended regular expressions (ERE).
  - This means you can use more advanced pattern matching features, like +, ?, and | for alternation, without escaping them.
  - The latest way to use grep is with the -E option. This option treats the pattern you used as an extended regular expression.
    - **grep -E 'pattern1|pattern2' fileName_or_filePathCopy**
  - **The deprecated version of extended grep is egrep.**
    - **egrep 'pattern1|pattern2' fileName_or_filePathCopy**

- **fgrep (Fixed-string Global Regular Expression Print)**
  - **fgrep** searches files for one or more pattern arguments.
  - It does not use regular expressions; instead, it uses fixed string comparisons to find matching lines of text in the input.
  - **fgrep is the same as grep -F.**
  - Search for a literal string in a file (no regular expressions):
    - **fgrep "apple" filename**
  - Search for a string across multiple files:
    - **fgrep "apple" file1 file2**

What is the Difference Between grep, grep -E, and egrep?

- The egrep command is an outdated version of extended grep. It does the same function as grep -E.
- The difference between grep and extended grep is that extended grep includes meta characters that were added later.
- These characters are the parenthesis (), curly brackets {}, and question mark. The pipe character | is also treated as a meta character in extended grep.

- **Working with columns and fields cut, paste, join**

1. Cut

✓ The cut command is a command-line utility that extracts specific sections of a specified **file** or piped data and prints the result to standard output.

✓ The command cuts parts of a line based on fields, delimiters, **byte** positions, and character

✓ **Syntax**

cut [option] [file]

Option	Description
-f (--fields=LIST)	Specifies which fields to extract from the input based on a delimiter.
-b (--bytes=LIST)	Instructs which bytes are to be extracted from each input line.
-c (--characters=LIST)	Specifies which characters are to be extracted from each input line.

Examples

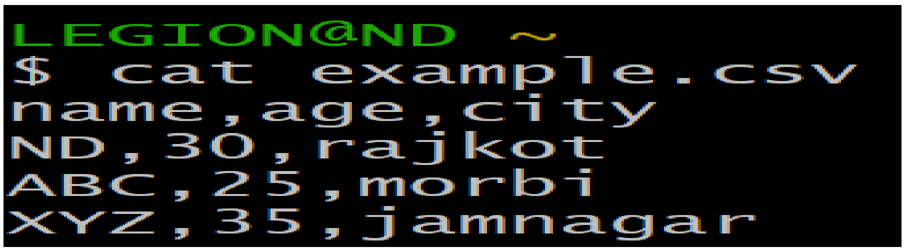
echo -e "name,age,city\nND,30,RAJKOT\nABC,25,MORBI\nXYZ,35,JAMNAGAR" > example.csv

✓ The command has no output but creates a file example.csv with the following content:

```
name,age,city
ND,30,RAJKOT
ABC,25,MORBI
XYZ,35,JAMNAGAR
```

✓ Confirm the file contents with the cat command:

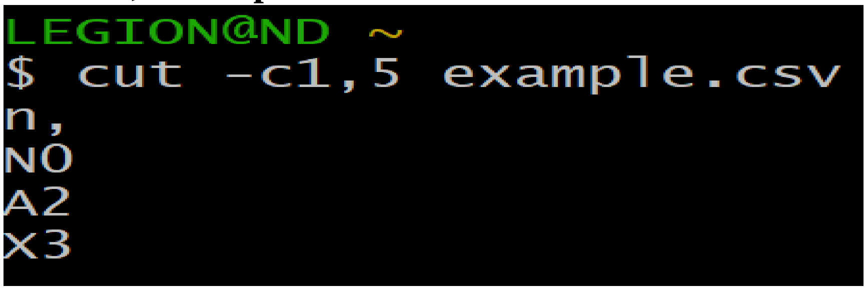
```
cat example.csv
```



```
LEGION@ND ~
$ cat example.csv
name,age,city
ND,30,rajkot
ABC,25,morbi
XYZ,35,jamnagar
```

- ✓ Below are the most common cut command use case examples.
  - cut -c [LIST] [file]
- ✓ The [LIST] argument specifies the characters to be extracted from each line of [file].
- ✓ For example, if you want to extract the first and fifth characters from each line of the example.csv file, run:

  cut -c1,5 example.csv



```
LEGION@ND ~
$ cut -c1,5 example.csv
n,
NO
A2
X3
```

2. Paste

- ✓ paste is a command that allows you to merge lines of files horizontally. It outputs lines consisting of the sequentially corresponding lines of each file specified as an argument, separated by tabs.
- ✓ The general syntax for the paste command is as follows:
  - paste [OPTION].. [FILE]...
- ✓ If no input files are provided or when - is given as argument, paste uses the standard input.
  - echo -e "Apple\nBanana\nCherry" > file1.txt
  - echo -e "Red\nYellow\nPurple" > file2.txt
  - echo -e "Fruit\nFruit\nFruit" > file3.txt
- ✓ Now, we can use the paste command to combine these files:
- ✓ Example output:

```
Apple   Red     Fruit
Banana  Yellow  Fruit
Cherry  Purple  Fruit
```

3. Join

- ✓ The join command reads the files specified by the File1 and File2 parameters, joins lines in the files according to the flags, and writes the results to standard output.
- ✓ **The File1 and File2 parameters must be text files.**
- ✓ Usage :
  - join [OPTION]… FILE1 FILE2

-a FILENUM also print unpairable lines from file FILENUM, where FILENUM
is 1 or 2, corresponding to FILE1 or FILE2
-e EMPTY replace missing input fields with EMPTY
-i, –ignore-case ignore differences in case when comparing fields

```
$ cat testfile1
        1 India
        2 US
        3 Ireland
        4 UK
        5 Canada

$ cat testfile2
        1 NewDelhi
        2 Washington
        3 Dublin
        4 London
        5 Toronto

$ join testfile1 testfile2
        1 India NewDelhi
        2 US Washington
        3 Ireland Dublin
        4 UK London
        5 Canada Toronto
```

- **Tools for sorting: sort, uniq**
  - ✓ The Linux utilities sort and uniq are useful for ordering and manipulating data in text files and as part of shell scripting.
  - ✓ The sort command takes a list of items and sorts them alphabetically and numerically.
  - ✓ The uniq command takes a list of items and removes adjacent duplicate lines.
    - o **sort [options] [file]**
  - ✓ Here are some common options you can use with the sort command:

    -n: Sort numerically
    -r: Sort in reverse order
    -k <field>: Sort based on a specific field
    -t <delimiter>: Use a custom field delimiter
  - ✓ For example, let's say you have a file named data.txt with the following content:
    - o banana
    - o apple
    - o orange
    - o banana
  - ✓ To sort the file in ascending order, you can run the following command:
    - o sort data.txt

- ✓ This will output the sorted file:
  - apple
  - banana
  - banana
  - orange
- ✓ If you want to sort the file in descending order, you can use the -r option:
  - sort -r data.txt
- ✓ This will output the file sorted in reverse order:
  - orange
  - banana
  - banana
  - apple

- ✓ **Uniqing a file means removing duplicate lines from the file. You can use the uniq command to achieve this. The basic syntax is as follows:**
  - **uniq [options] [file]**
- ✓ Here are some common options you can use with the uniq command:
  - -c: Count the number of occurrences of each unique line
  - -d: Only display lines that have duplicates
  - -u: Only display unique lines
- ✓ Let's continue with the data.txt file from the previous example:
  - banana
  - apple
  - orange
  - banana
- ✓ To remove the duplicate lines, you can run the following command:
  - uniq data.txt
- ✓ This will output the file with unique lines:
  - banana
  - apple
  - orange
- ✓ If you want to see the count of each unique line, you can use the -c option:
  - uniq -c data.txt
- ✓ This will output the file with the count of each unique line:
  - 2 banana
  - 1 apple
  - 1 orange

- **Comparing files: cmp, comm, diff**
- **Cmp**
  - cmp is used to compare two files byte by byte.
  - If a difference is found, it reports the byte and line number where the first difference is found. If no differences are found, by default, cmp returns no output.
  - Syntax: $ cmp filename1 filename2
  - Example:

    ```
    $ cat > p1
    This is file1
    $ cat > p2
    This is file2
    $ cmp p1 p2
    p1 p2 is differ: byte 13, line1
    ```
- **Comm**
  - Compare two sorted files line-by-line.
  - Compare sorted files FILE1 and FILE2 line-by-line.
  - With no options, comm produces three-column output. Column one contains lines unique to FILE1, column two contains lines unique to FILE2, and column three contains lines common to both files.
  - Each of these columns can be suppressed individually with options.
  - Syntax :comm [OPTION]... FILE1 FILE2
  - Example

    ```
    $cat c1
    This is c1 file
    Line 2
    This is example for comm
    $cat c2
    This is c2 file
    Line 2
    This is example for comm
    $comm c1 c2
    ```

    Option:

    ```
    comm –1 a1 a2
    Suppress line uniq to file1
    comm –2 a1 a2
    Suppress line uniq to file2
    comm -3 a1 a2
    Suppress line that appear in both file
    ```

- **Diff**
  - diff analyzes two files and prints the lines that are different.
  - The diff is the third command that can be used to display the file differences.
  - **It is more powerful than cmp command.**
  - **It displays the differences between two files while cmp command displays only first location of mismatch.**
  - It gives location of mismatch as well as suggests changes to make two files identical. It also tells that which line in particular file have to be changed.
  - Syntax : $ diff filename1 filename2
  - Example

```
$cat c1
This is c1 file
Line 2
This is example for comm
$cat c2
This is c2 file
Line 2
This is example for comm
$diff c1 c2
```

- **Changing Information in Files: tr, sed**
  - ➢ **Tr command**
    - ✓ The tr command in Linux is a powerful utility for translating or deleting characters in a text file or stream.
    - ✓ It can be used to perform various tasks, such as removing unwanted characters, converting cases, and replacing characters.
      - **Syntax: tr [options] set1 [set2]**
    - ✓ The set1 and set2 are character sets to be translated or deleted. The options are used to modify the behavior of the tr command.

Sequence	Interpretation
\NNN	Characters with the NNN octal value (1 to 3 octal digits).
\\	Backslash.
\b	Backspace.
\f	Form feed.
\n	Newline character.
\r	Return character.
\t	Horizontal tab.
\v	Vertical tab.
[CHAR*]	Copies CHAR* in SET2 up to the length of SET1.
[CHAR*REPEAT]	Repeats copies of CHAR. Repeats octal if starting with 0.

[:alnum:]	All letters and digits.
[:alpha:]	All letters.
[:blank:]	Horizontal whitespaces.
[:digit:]	All digits.
[:lower:]	All lowercase characters.
[:space:]	Horizontal or vertical whitespace characters.
[:upper:]	All uppercase letters.

Convert Lowercase to Uppercase

```
cat testing.url
        www.testing.com
tr 'a-z' 'A-Z' < testing.url
        WWW.TESTING.COM
$ tr '[:lower:]' '[:upper:]' < testing.url WWW.TESTING.COM
```

Basic Find and Replace

```
cat env.txt
        $JAVA-HOME and $MAVEN-HOME are system variables.
cat env.txt | tr '-' '_'
        $JAVA_HOME and $MAVEN_HOME are system
variables.
```

Delete Specific Characters

```
$ echo "A a B b C c" | tr -d 'a-z'
        A B C
```

- **Sed command**
  - ✓ sed is a stream editor, and it can be used to edit text files, with its most common use being to replace occurrences of words in a files.
  - ✓ The sed (Stream Editor) command in Linux is a powerful utility for text manipulation.
  - ✓ It enables users to perform various operations such as searching, replacing, inserting, and deleting text in files.
  - ✓ The sed command is a stream editor that processes text in a line-by-line fashion.
  - ✓ This allows you to modify file content without directly opening the file in a text editor.
  - ✓ It is widely used in shell scripting and system administration to automate text-processing tasks.
  - ✓ Key Features of sed
    - Pattern matching and replacement
    - In-place file editing
    - Text filtering and manipulation

- Support for regular expressions
- Multiline operations

Commands

A command defines the actions that it has to complete with the source file and considering the specified options.

Command	Description
a	append: Adds one or more other lines to the selected lines.
c	change: Replaces selected lines with new content.
d	delete: Deletes the selected lines.
g	get: Copies the content from the hold spaces into the patten space.
G	GetNewline: Pastes the content from the hold spaces into the pattern space.
h	hold: Copies the content from the pattern space into the hold space.
H	HoldNewLine: Pastes the content from the pattern spaces into the hold space.
i	insert: Inserts one or more lines in front of the selected lines.
p	print: Displays the selected lines.
q	quit: Ends Linux SED.
w	write: Writes lines into the text file.
!	Negation: Applies the command to lines that do not apply to the input.

- **Examining File Contents: od**
- **$od [octal dump]:**
  - Many files contain nonprinting characters and most UNIX commands don‘t display it properly. The od command display the ASCII octal value of its input.
  - $ od means octal dump. It displays the contain as a**octal number**.
  - This can be useful when the output file contains **non printable character**
  - To make this character visible you have to use od command that display **ASCII** octal value of file contain by default
  - **For performing od use following steps:**

```
$ cat > f1
This is file1
Ctrl + d
```

Example: $ od f1

Options:

$ od –b:

Display the octal value of each character in the file

Example: $ od -b f1

$ od –a :

It will display name character

Example: $ od –a f1

$ od –c :

It will display ASCII character or back slash

Example: $od -c f1

$ od –d:

Display signed decimal

Example: $od -d f1

$ od –f:

It will display floating point

Example: $od -f f1

$ od –o(it is default):

It will display octal

Example: $od -o f1

$ od –x:

Display hexa decimal

Example: $od -x f1

- **Tools for mathematical calculations: bc, factor**
- **$bc [basic calculator]:**
  - ✓ bc command is used for command line calculator. It is similar to basic calculator by using which we can do basic mathematical calculations.
  - ✓ Arithmetic operations are the most basic in any kind of programming language. Linux or Unix operating system provides the bc command and expr command for doing arithmetic calculations.
  - ✓ You can use these commands in bash or shell script also for evaluating arithmetic expressions.
  - ✓ **The bc command supports the following features:**
    - o Arithmetic operators, Increment or Decrement operators, Assignment operators, Comparison or Relational operators, Logical or Boolean operators, Math functions, Conditional statements, Iterative statements.
    - o The bc(Basic Calculation) command is for mathematical calculation. It will do basic calculation specified by you. To perform calculation, you can take input from keyboard or from file.

  - When you invoke bc command the input has to be given by keyboard, each line terminated by pressing <Enter> . After you have finished your work, use <ctrl+d>
- ✓ **Syntax:** $ bc [-c] [-l] [file] (c considers compile only and l consider log value)
- ✓ **Example:**
  - bc (some system information may represents)
    then
    12+5 <enter>
    5*5 <enter>
    50-20 <enter>
    Ctrl + d

++var : Pre increment operator, variable is increased first and then result of variable is stored.
var++ : Post increment operator, result of the variable is used first and then variable is incremented.
– – var : Pre decrement operator, variable is decreased first and then result of variable is stored.
var – – : Post decrement operator, result of the variable is used first and then variable is decremented.

- **factor**
  - ✓ It is another math related command.
  - ✓ When factor is invoked, it waits for a number to be typed in.
  - ✓ When you input the number it will factories the number and prints its prime factors. Then it waits for another command.
  - ✓ It exits if it encounters zero or any non-numeric character. This command print the prime factor of each number.
  - ✓ This command is one of the simplest command [no argument.
  - ✓ **Syntax:** $factor [number]
  - ✓ **Example:** $ factor 12

- **Monitoring Input and Output:**
- **Tee**
  - ✓ UNIX provides feature by which you can save the standard output in a file, as well as display on the terminal. It is possible by tee command.
  - ✓ Tee can be used standard input and standard output, which means that it can be placed anywhere in a pipeline.
  - ✓ Syntax: tee [option] [files]
  - ✓ Example :
  - $ ls –l | tee filelist
  - $ cat filelist

- **Script**
  - ✓ This command provides one of the important features of the UNIX. All commands, their output and the error messages are stored in the file.
  - ✓ If you are doing some important work, and want to keep log of all your activities, than you should invoke this command immediately after you log in.
  - ✓ $ script
    - o script started, output file is typescript.
  - ✓ The prompt returns and all your work now recorded in the file typescript. You can terminate this session by entering exit
    - o $ exit
  - ✓ You can now this file with the cat command. Script overwrites any previous typescript that may exist.
  - ✓ **Example**

```
$ script st.txt
$ cal
$ date
$ exit
$ cat st.txt
```

- **Tools for Displaying Date and Time:**
- **Cal**
  - ✓ The cal command displays a simple, formatted calendar in your terminal.
  - ✓ $ cal command display simple calendar if no argument are supply by default it display the calendar of the current month.
  - ✓ Syntax: $cal
  - ✓ Example: $cal
  - ✓ $cal 02 2015:
- **Date**
  - ✓ You can display the current date with the date command, which shows the date and time to the nearest second.
  - ✓ $ date
  - ✓ You can display only month by +%m option and month name by +%d option and only year with +%y option.

- **Communications:**
- **telnet**
  - ✓ Communication command between client and server.
  - ✓ telnet is used to connect remote system using telnet protocol.

- ✓ when this command is invoke it enters in command mode and display telnet prompt. in this mode it accept the command to be execute

Command	Description
Open	Connect to remote computer
Close	Close the connection
Display	It show the parameter
Mode	It is use to change character mode to line mode
Status	It display the status information

- **ping**
  - ✓ PING (Packet Internet Groper) command is used to check the network connectivity between host and server/host.
  - ✓ This command takes as input the IP address or the URL and sends a data packet to the specified address with the message “PING” and get a response from the server/host this time is recorded which is called latency.
  - ✓ ICMP tool (Internet Control Message Protocol) tests the connectivity between two nodes.
    1. **$ ping www.google.com OR $ ping 172.217.27.196**

- **Process Related Commands: Command to run process in background**
- **Ps**
  1. Process is an instance of a running program. A process is said to be born when the program starts execution and remains alive as long as the program is active.
  2. Processes have some attributes which have stored in process table.
  3. Two important attributes are process id (pid) -unique identification number of the process and parent process id (ppid) –unique identification number of the parent process.
  4. The ps (process status) command displayed the status of the process. It display some process attributes. It displays the information of process running on the user terminal.
  5. The output of this command is depend on the version of the UNIX system
  6. The PID field gives you the process id. The TT field gives terminal name on which process executes.
  7. The TIME shows the how much cpu time process has used. The command field lists the command argument stored in the process itself (process name).

   i. -f option: The ps command with –f option will give the full information including user name, parent process id and starting time of the process.
   ii. -u option: The ps command with –u option gives the information for the process of particular user.
   iii. -a option: The ps command with –a option gives information about only user process. It will not show the status of system process.

8. This command is useful for display the information about active processes.
9. Ps command takes snapshot of these processes and display the process identification(PID), terminal associated with this process(TTY) and CPU counting time in hh:mm:ss format.
10. **Syntax: $ps**

- **Sleep**
  1. sleep is used to suspend execution for an interval of time.
  2. The sleep utility shall suspend execution for at least the internal number of second specified by the time.
  3. The time specified in second
  4. **$ sleep 10;echo "   after 10 second"**

GEETANJALI GROUP OF COLLEGES

UNIT 4- TEXT EDITOR WITH VI AND NANO EDITOR, SHELL PROGRAMMING

PART 1: TEXT EDITOR WITH VI AND NANO EDITOR

- Introduction of VI Editor
- Modes in VI
- Switching mode in VI
- Cursor movement
- Screen control commands
- Entering text, cut, copy, paste in VI editor
- Introduction of NANO editor

PART 2: SHELL PROGRAMMING

- Shell Keywords
- Shell Variable
- System Variables
  - PS2, PATH, HOME, LOGNAME, MAIL, IFS, SHELL, TERM, MAILCHECK
- User Variable
  - Set, unset and echo command with shell variables
- Positional Parameters
- Interactive shell script using read and echo
- Decision Statement
  - If then fi
  - If then else fi
  - If then elif else fi
  - Case esac
- Test command
- Logical Operators
- Looping Statements
  - For loop
  - While loop
  - Until loop
  - Break, continue command
- Array
- Function
- Various shell script examples

PART 1: TEXT EDITOR WITH VI AND NANO EDITOR

- **Introduction of VI Editor**
  - vi is a screen-oriented text editor originally created for the Unix operating system.
  - We have learned to create a file using the cat command.
  - The cat command although it allows us to create a file is not a good option when creating a shell script.
  - We need a good text editor to perform such operations.
  - Text editor like nano, VI or vim, ed and other are generally best suited for creating text file.
  - Gedit is a graphical editor available with GNOME desktop environment.
  - Kwrite is graphical editor available with KDE desktop environment.
  - We will use the vim or VI edit which us VISUAL DISPLAY editor to write the shell script.
  - The editor is available with almost all UNIX AND LINUX flavors.
  - The VIM is a text editor written by BRAM MOOLENAAR and released publicly in 1991.
  - It is an enhanced version of the VI editor distributed with most UNIX system.
  - VIM is a highly configurable text editor built to enable efficient text editing.
  - The VIM editor can be used from both a command line interface and as a standalone application in a graphical user interface.
  - To work with the vim editor, we will have to initiate it first.
  - Open a new terminal window. We can open the vim editor using two ways, first type vi at the prompt and press enter key or type vi followed by file name and press enter key.
  - **Starting vi**
    - To start using vi, at the Unix prompt type vi followed by a file name. If you wish to edit an existing file, type in its name; if you are creating a new file, type in the name you wish to give to the new file.
    - $vi filename
    - Then hit Return. You will see a screen similar to the one below which shows blank lines with tildes and the name and status of the file.
    - Vifirstfile
      ```
      ~
      ~
      ~
      ~
      firstfile: new file: line 1
      ```

- You are presented with a full empty screen, each line beginning with a ~(tilde). This indicates that they are non-existent lines. The last line is reserved for some commands.

Modes in VI

- There are three modes of operation in vi: Command Mode, Input Mode, and Line Mode
- **The Command Mode:**
  - When we first start editing a file using the VIM editor, the editor will be opened in a command mode.
  - We can issue many commands that allow us to insert, append, delete text, or search and navigate within our file.
  - Note that when using command mode we can't insert text immediately. We first need to issue an **insert(i), append(a) or open(o) command** to insert the text in the file.
  - An extension to command mode is a visual mode.
  - Visual mode is flexible and easy way to select a piece of text from the file.
  - It is the only way to select a block of text that needs to be modified.

Command	**Usage**
v	Switch to the visual mode (allows us to manipulate characters)
V	Switch to the Visual Mode(allows us to manipulate lines)
Ctrl + v	Switch to the block-visual mode(allows us to manipulate rectangular blocks of text)

- **The Insert Mode**
  - When we issue an insert, append or open command, we will be in the insert mode.
  - The current text editor shows us the current mode of operation.
  - Once in an insert mode we can type text into our file or navigate within the file.
  - We can toggle between the command mode and the insert mode by pressing ESC key.
  - This operating will be performed often using the VIM editor.

Command	**Usage**
a	To insert text after the current cursor position.
i	To insert text before the current cursor position.
A	To append text at the end of the current line.
I	To insert text from the beginning of a line.
o	To insert in a new line above the current cursor position.
O	To insert in a new line below the current cursor position.

- **The Last Line Mode:**
    - The last line mode normally is used to perform operations line quitting the VIM session or saving a file.
    - To go to the last line mode we first need to be in the command mode.
    - From command mode we can go last line mode by pressing the COLON (:) KEY.
    - After pressing this key, we will see a colon character at the beginning of the last line of our editor window with a cursor blinking near it.
    - This indicates that the editor is ready for accepting a "last line command".
    - It is possible to toggle back to the command mode from the last line mode by pressing the ESC key TWICE or pressing BACKSPACE key until the initial ":" character is gone alone with all the characters that we had typed or by simple the ENTER key.

- **Saving the file:**
    - Once we have written the contents shown we need to save this file.
    - To save the file we need to switch to the last line mode from the insert mode.
    - Press the ESC key and type colon (:), you will notice that colon is displayed in the bottom of the screen.
    - Type wq (Write and quit) and press the enter key. o The VIM editor will now be closed and we will be able to see the shell prompt

Command	Description
:w	To save file remain in editing
:wq	To save file and quite the vi editor
:x	Same as above
:q	Quit editing mode and no changes
:q!	Quit editing mode without saving changes
:saveas file name	Save with file name continue with editing

➢ **Switching mode in VI**

- While you start VI editor at that time you will be at vi mode that will ready to accept defined command on that particular key but not any input.
- If you want to input text into the file you will have to go to input mode for that you will have to
- press "i".
- If you will press "i" at VI mode you will be at input mode here you can input any data into the file from this mode if you want to switch to vi mode then you will have to press "Esc" key.
- If you will press "Esc" key at input mode you will be able to switch to vi mode.

- If you want to switch to command mode from the VI mode then you will have to press ":" or "/" that will support you to switch you from VI mode to command mode.
- If you want to switch to command mode from the input mode then you will have to first come to the VI mode then you will be able to switch to command mode.
- To switch from input mode to command mode then you first will have to press "Esc" key and then after you will have to press ":" or "/" key that will support you to switch from input mode to command mode via VI mode.

➢ **Cursor movement**

- Whatever the command you are giving that will work on VI mode only.
- To move around in the file, use the arrow keys: the up arrow will move the cursor up one line, the down arrow down one line.
- The right arrow will cause the cursor to move one position to the right.
- From the end of a line, it will go to the beginning of the next line.
- For terminals with no functional arrow keys, four keys will move the cursor around:

h	left arrow
j	down arrow
k	up arrow
l	right arrow

➢ **Screen control commands**

- Whatever the command you are giving that will work on vi mode only.

ctrl + f	This will move the user forward one screen in the file.
ctrl+b	This will move the user backward one screen in the file.
G	This will move the cursor to the end of the file. Note, again that vi, like any other UNIX utility, is always case sensitive.
Num	This will bring the cursor to line defined line number.

➢ **Entering text, cut, copy, paste in VI editor**

- Whatever the command you are giving that will work on vi mode only.

x	That will delete on character.
dd	That will delete lines beginning at the current line.
yy	That will copy lines beginning at the current line.
p	That will paste copied line.
u	This is very useful command. It cancels the effect of the previously executed command.
Q	When in vi, typing will bring the user into command mode. As the : is typed, it will be displayed n the last line of the screen, and vi will wait for a command to be typed. q is such a command. This will exit vi, if no changes have been made since the last write-to- file command :q! will exit, even if the

	buffer has not been written to the file.
w	Will cause the contents of the buffer to be written to the file :wq will write the buffer to the file, and exit the calling shell.
I	This will support you to switch from VI mode to input mode.
:	This will support you to switch from VI mode to command mode.
esc	This will support you to switch from input mode to VI mode.
/	This will support you to find any particular pattern from the file.

➢ **Introduction of NANO editor**

- GNU nano is a friendly and convenient text editor like vi and emac. It offers many other extra features like word searching, replacing, jump to a line or column, filename tab completion, auto-indentation, etc.
- Nano is a simple yet powerful command line-based text editor, very popular among beginner Linux users for its simple-to-use interface.
- At the top, GNU nano version is shown at the left and in the middle filename is shown being edited (currently no file is being edited hence New Buffer is written).
- At the end of the screen, keyboard commands are given. Command written as ^G means press ctrl + g key and command M-R means press alt + r.
- There is no use of uppercase letter in any of the keyboard command in nano editor. You can use lowercase letter with ctrl and alt keys.
- Note:For MAC users escape key is used instead of alt key.
- Press ctrl + g for the help menu. You'll get all information about nano editor in this menu.
- **Creating or editing a file**
  - To create a new file or edit an existing one, at your Unix shell prompt, type:
    - **nano filename**
  - replace filename with the name of the file you want to create or edit. for example, to create a file and name it indiana.txt, enter:
    - **nano file1.txt**
  - If the file already exists, Nano opens it for you to edit. If it doesn't exist yet, Nano creates it and places you in an editing buffer.

PART 2: SHELL PROGRAMMING

- **Shell Keywords**
  - keywords cannot be used as variable names because of it is a reserved word with containing reserved meaning.

echo	read	set	unset
readonly	shift	export	if
fi	else	while	do
done	for	until	case
esac	break	continue	exit
return	trap	wait	eval
exec	ulimit	umask	

- **Shell Variable**
  - A variable is a location for storing a value which can be a filename, text, number or any other data. It is usually referred to with its Symbolic name which is given to it while creation. The value thus stored can be displayed, deleted, edited and re-saved.
  - A shell variable is a special variable that is set by the shell and is required by the shell in order to function correctly. Some of these variables are environment variables whereas others are local variables.

- **System Variables**
  - The UNIX OS has defined system variables for its own use. They are used to customize the working environment. So, it is also called environment variable.
  - Generally, they contain some default value, but user can set its own values in this variable. Users can also read the value already set in this variable.
  - To see the default value of all the system variables, use set command.
  - The following are some system variables
    - PS1 Primary prompt string, default '$'
    - PS2 Secondary Prompt string, default '>'
    - PATH Define the path which the shell must search in order to execute any command or file.
    - HOME Stores the default working directory of the user. cd command gives the home directory.
    - LOGNAME Stores the login name of the user.
    - IFS Internal field separator which is a space, tab or new line.
    - SHELL Defines the name of your default working shell.
    - TERM Defines the name of the terminal on which you are working.
    - MAIL Defines the file where the mail of the user is stored.

        - MAILCHECK Defines the duration after which the shell checks whether the user has received any mail. By default its value is 600 second

- **User Variable**
    - Shells also allow you to create your own variables for use within scripts (local variables) and to pass between scripts (global variables).
    - User variables are traditionally created using lower-case characters.
    - To create a variable merely choose a lower-case name for the variable and give it a value using an equal (=) sign.
    - Make certain that there are no spaces on either side of the equal sign.
    - You can use the unset command to nullify the value of a previous set variable. Here are some examples.
        - **Syntax**

```
variable_name = value
a = 10
echo $a
company ='new Technology'
echo $company;
```

    - **set variable:**
        - set variable is used to set value of any variable. It is used directly on the prompt or in the shell script.
            - **$ set name**
            - **name=abc**
        - Here, name is the variable name and abc is its value in string. If the value is a list of string, then use parenthesis.
            - **$ set name**
            - **name= (val1, val2, val3)**
    - **unset variable:**
        - To delete or unset variable, the unset variable is used. After unsetting the variable, the variable and its value both erased.
            - **$ unset name**
        - After executing above command the variable name and its value assigned to it are erased from shell‟s memory. Now, you cannot use the name variable.
        - To use the value of variable, use $ sign with variable name
- **Positional Parameters**
    - On many occasion, we need to pass information to a program. A very convenient way of doing this is by specifying arguments at the command line. But how does the shell script know what has been passed to it?
    - For this, the shell uses something called „Positional Parameters‟. This can be considered as variables defined by the shell. They are nine in number, named $1 through $9.

- It supports command line arguments. User can provide arguments at the command prompt while executing the shell script. Such arguments can be provided along with the script name.
- Consider that you want to copy one file into another and give the filename at command line. It takes two arguments as a filename
    - $ sh copyfile.sh f1 f2

- **Interactive shell script using read and echo**
    - The echo command is used to print the string and read command is used to accept input from the user. The read command is used to read input values in variables.
    - The syntax of read command is: read variable
    - When script encounters this command, it pauses at the point to take input from the keyboard. Any value entered by the user is stored in the variable specified with the read command.
    - Let's take an example:

```
echo “Enter your name”
read nm
echo “Hello”
$nm
```

It will give following

output:

```
Enter your name
ND
Hello ND
```

- **Decision Statement**
    - While writing a shell script, there may be a situation when you need to adopt one path out of the given two paths. So, you need to make use of **conditional statements** that allow your program to make correct decisions and perform the right actions.
    - Unix Shell supports **conditional statements** which are used to perform different actions based on different conditions.
    - Unix shell provides decision making using if then else and case structure.
    - **If – then – fi**

Syntax:

```
If [condition]
Then
execute command if condition is true
…
fi
```

Example:

```
NAME="Rohit"
if [ $NAME = "Rohit" ]
```

```
then
        echo "True - my name is Rohit"
fi
```

- **If – else – fi**
    - If given condition is true, then command1 is executed else command2 is executed.
    - **Syntax:**
      ```
      If [condition]
      Then
              execute command1 if condition is true
      else
              execute command2 if condition is false
      fi
      ```
    - **Example:**
      ```
      NAME="Virat"
      if [ $NAME = "Rohit" ]
      then
              echo "True - my name is Rohit"
      else
              echo "False"
              echo "My name is $NAME"
      fi
      ```
- **if...elif...else...fi**
    - The if...elif...fi statement is the one level advance form of control statement that allows Shell to make correct decision out of several conditions.
    - **Syntax:**
      ```
      if [ condition 1 ]
      then
              Statement to be executed if condition 1 is true
      elif [ condition 2 ]
      then
              Statement to be executed if condition 2 is true
      Else
              Statement to be executed if no expression is true
      fi
      ```
    - **Example:**
      ```
      NAME="Rohit"
      if [ $NAME = "Virat" ]
      then
              echo "Virat Kohli"
      elif [ $NAME = "Rohit" ]
      then
      ```

```
        echo "Rohit Sharma"
else
        echo "This is Dhoni"
fi
```

- **The case...esac Statement**
  - Unix Shell supports case...esac statement which handles exactly this situation, and it does so more efficiently than repeated if...elif statements.
  - There is only one form of case...esac statement
  - **Syntax:**

```
case "$variable"
in
        "$condition1" )
                command... ;;
        "$condition2" )
                command... ;;
esac
```

- **Test command**
  - test is used as part of the conditional execution of shell commands.
  - The test command is used to evaluate conditional expressions with integers, strings and file attributes. The syntax of test command is: test expression
  - Here, expression is the condition to be evaluated. The test command is used with the if condition,
  - while loop, until loop etc.. to check the condition

  - **Integer related test Operators:**
  - Assume variable a holds 10 and variable b holds 20 then

INTEGER1 -eq INTEGER2	INTEGER1 is equal to INTEGER2	[$a -eq $b] is not true
INTEGER1 -ge INTEGER2	INTEGER1 is greater than or equal to INTEGER2	[$a -ge $b] is not true.
INTEGER1 -gt INTEGER2	INTEGER1 is greater than INTEGER2	[$a -gt $b] is not true.
INTEGER1 -le INTEGER2	INTEGER1 is less than or equal to INTEGER2	[$a -le $b] is true.
INTEGER1 -lt INTEGER2	INTEGER1 is less than INTEGER2	[$a -lt $b] is true.
INTEGER1 -ne INTEGER2	INTEGER1 is not equal to INTEGER2	[$a -ne $b] is true.

- **String related test operator**
- Assume variable a holds "abc" and variable b holds "efg" then

-n STRING	the length of STRING is nonzero	[-n $a] is not false
STRING	equivalent to -n STRING	[$a] is not false.
-z STRING	the length of STRING is zero	[-z $a] is not true.
STRING1 = STRING2	the strings are equal	[$a = $b] is not true.
STRING1 != STRING2	the strings are not equal	[$a != $b] is true.

- **File related test operator**
- Assume a variable file holds an existing file name "test" the size of which is 100 bytes and has read, write and execute permission on

Operator	Description	Example
-b file	Checks if file is a block special file; if yes, then the condition becomes true.	[-b $file] is false.
-c file	Checks if file is a character special file; if yes, then the condition becomes true.	[-c $file] is false.
-d file	Checks if file is a directory; if yes, then the condition becomes true	[-d $file] is not true.
-f file	Checks if file is an ordinary file as opposed to a directory or special file; if yes, then the condition becomes true.	[-f $file] is true.
-g file	Checks if file has its set group ID (SGID) bit set; if yes, then the condition becomes true.	[-g $file] is false.
-r file	Checks if file is readable; if yes, then the condition becomes true	[-r $file] is true
-w file	Checks if file is writable; if yes, then the condition becomes true.	[-w $file] is true.
-x file	Checks if file is executable; if yes, then the condition becomes true.	[-x $file] is true.

- **Logical Operators**
  Assume variable a holds 10 and variable b holds 20 then

(EXPRESSION)	EXPRESSION is true	
! EXPRESSION	EXPRESSION is false	[! false] is true.

EXPRESSION1 - a EXPRESSION2	both EXPRESSION1 and EXPRES SION2 are true	[$a -lt 20 -a $b -gt 100] is false
EXPRESSION1 - o EXPRESSION2	either EXPRESSION1 or EXPRES SION2 is true	[$a -lt 20 -o $b -gt 100] is true.

- **Looping Statements**
  - **For loop**
    - The for loop moves through a specified list of values until the list is exhausted.
    - Syntax of for loop using in and list of values is shown below. This for loop contains a number of variables in the list and will execute for each item in the list.
    - For example, if there are 10 variables in the list, then loop will execute ten times and value will be stored in varname.

    -
      ```
      for VARIABLE in list
      do
              command 1
              command 2
              command N
      done
      ```
    - **Example**

    -
      ```
      for i in 1 2 3 4 5
      do
              echo "Welcome $i times"
      done
      ```
  - **While loop**
    - There is a condition in while. And commands are executed till the condition is valid. Once condition becomes false, loop terminates.

    -
      ```
      while [ condition ]
      do
              command 1
              command 2
              command N
      done
      ```
  - **Until loop**
    - Until loop always executes at least once. Loop while executes till it returns a zero value and until loop executes till it returns non-zero value.
    - **Syntax:**

    -
      ```
      until [condition]
      do
      ```

```
        command 1
        command 2
        command 3
    done
```

- **Break**
  - The break statement is used to terminate the execution of the entire loop, after completing the execution of all of the lines of code up to the break statement. It then steps down to the code following the end of the loop.
  - Syntax: Break
  - The break command can also be used to exit from a nested loop using this format: break n
- **continue command**
  - The continue statement is similar to the break command, except that it causes the current iteration of the loop to exit, rather than the entire loop.
  - This statement is useful when an error has occurred but you want to try to execute the next iteration of the loop.
  - **Syntax: continue**
  - Like with the break statement, an integer argument can be given to the continue command to skip commands from nested loops.

- **Array**
  - An array is a collection of similar data elements stored at contiguous memory locations.
  - Variables store single data elements. Arrays can store a virtually unlimited number of data elements. When working with a large amount of data, variables can prove to be very inefficient and it’s very helpful to get hands-on with arrays.
  - There are two types of arrays that we can work with, in shell scripts.
  - **Indexed Arrays -** Store elements with an index starting from 0.
  - **Associative Arrays -** Store elements in key-value pairs.
    - **Indexed Arrays**
      - We can use any variable as an indexed array without declaring it.
      - To explicitly declare a variable as a Bash Array, use the keyword 'declare' and the syntax can be defined as:
      - declare -a ARRAY_NAME
      - where, ARRAY_NAME indicates the name that we would assign to the array.
      - ARRAY_NAME[index_1]=value_1
      - ARRAY_NAME[index_2]=value_2
      - ARRAY_NAME[index_n]=value_n

- ❖ **Associative Arrays**
  - We can use the keyword 'declare' and the -A (uppercase) option to declare the associative arrays. The syntax can be defined as:
  - declare -A ARRAY_NAME
  - A general method to create an associative array can be defined in the following form:
  - declare -A ARRAY_NAME
  - ARRAY_NAME[index_abc]=value_abc
  - ARRAY_NAME[index_xyz]=value_xyz
  - where index_ is used to define any string.
  - We can also write the above form in the following way:
  - declare -A ARRAY_NAME
  - ARRAY_NAME=

```
( [one]="first item"
[two]="second item"
[third]="third value" )
```

- **Accessing Array Elements**
  - The array elements that are being indexed individually. We can access all the elements by specifying the array index as shown below:
    - myArray[e1]="hello sybca student"
    - echo ${myArray[e1]}
  - Now check out the indexed array example.
    - myArray=(1 2 3 4 5)
    - echo ${myArray[2]}

- **Function**
  - Using functions to perform repetitive tasks is an excellent way to create code reuse.
  - Functions are popular for the following reasons:
    - Help to reuse a piece of code.
    - Improve the readability of the program.
    - Make the program modular.
    - Make maintenance easier.
  - Syntax:

```
function_name () {
        list of commands }
```

  - In order to call a function, we need to write its name and pass the required parameters in a space separate technique:
    - $ function_name $arg1 $arg2 $arg3

UNIT 5- GETTING STARTED WITH LINUX, LINUX BOOTING, LINUX ADMIN (UBUNTU)

PART 1: GETTING STARTED WITH LINUX

- History of LINUX
- GNU, GPL Concept
- Open Source & Freeware
- Structure and Features of LINUX
- Installation and Configuration of LINUX
    - Using with Ubuntu
- Startup, Shutdown & boot loaders of LINUX

PART 2: LINUX BOOTING

- LINUX booting process
    - LILO Configuration
    - GRBU Configuration

PART 3: LINUX ADMIN (UBUNTU)

- Creating LINUX User Account and Password
- Installing and Managing Samba Server
- Installing and Managing Apache Server
- Configure Ubuntu's Built-in Firewall
- Working with WINE

PART 1: GETTING STARTED WITH LINUX

➢ **History of LINUX**

- Linux is an open-source operating system.
- The first Linux version 0.02 and released on Oct 5 1991, consist of Linux kernel and 3 utilities.
- Linux was developed as a freely distributable version of UNIX.
- One of the early UNIX work a likes was Minix, written by Andy Tanenbaum.
- Although Minix didn't have a full range of features, it provided a small operating system that could be used on PC machines.
- To expand on Minix, a number of users started developing an enhanced operating system that would take advantage of the 80386 CPU's architecture.
- It supports centralized software installation from pre-existing repositories.
- Its resource requirements are low.
- It is often top of mind when developers are building application ecosystems and tooling for servers, leading to high levels of compatibility.
- It sustains necessary modifications to operating system behaviors.

➢ **GNU, GPL Concept**

- GNU stands for Gnu’s Not Unix, and it is pronounced as ―g-noo‖. It is a recursive acronym, and it stands for ―” Gnu’s Not Unix”.
- GNU is a free and open-source operating system that was started in 1984 by Richard Stallman GNU is based on the Unix operating system, but it has been greatly modified over the years.
- One of GNU’s main goals is to give users the freedom to run, study, share (copy), and modify the software.
- GNU is also designed to be portable so that it can be used on many different types of computers.
- Linux is licensed under the GNU General Public License (the GPL or copyleft), which is reproduced here to clear up some of the confusion about Linux's copyright status.
- Linux is not shareware, nor is it in the public domain. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software to make sure the software is free for all its users.
- This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it.

- **Open Source & Freeware**
  - Open source means the software code is available with licensed in which the copyright holder provides the rights to study, modify, and redistribute the software to anyone without any cost and with any purpose.
  - Freeware means software is available without cost.
  - Freeware generally available with some restricted rights.
  - There is not proper definition for freeware and open source, but it can be identified in following terms.
  - Anyone can join the open-source project, where in freeware a closed development group is developing the software.
  - In open-source code is available so new version can be created from the existing software, where in free ware code is private.

- **Structure and Features of LINUX**
- Linux OS has following components:

1) **Kernel**
   - Kernel is the core of the operating system. It establishes communication between devices and software.
   - **Device management:** A system has many devices connected to it like CPU, memory device, sound cards, graphic cards, etc. It also manages communication between all the devices.
   - **Memory management:** Kernel keeps a track of used and unused memory and make sure that processes shouldn't manipulate data of each other using virtual memory address.
   - **Process management:** In process management kernel assign enough time and gives priorities to processes before handling CPU to other process. It also deals with security and ownership information.
2) **System Libraries**
   - System libraries are special programs that helps in accessing the kernel's features. A kernel has to be triggered to perform a task and this triggering is done by the applications.
3) **System Tools**
   - Linux OS has a set of utility tools which are usually simple commands. It is software which GNU project has written and publish under their open source license so that software is freely available to everyone.
4) **Development Tools**
   - With the above three components your OS is running and working. But to update your system you have additional tools and libraries. These additional tools and libraries are written by the programmers and are called tool chain.

Linux Features

- **Multiuser capability:** Multiple users can access the same system resources like memory, hard disk, etc. But they have to use different terminals to operate.
- **Multitasking:** More than one function can be performed simultaneously by dividing the CPU time intelligently.
- **Portability:** Portability doesn't mean it is smaller in file size or can be carried in pen drives or memory cards. It means that it supports different types of hardware.
- **Security:** It provides security in three ways namely authenticating (by assigning password and login ID), authorization (by assigning permission to read, write and execute) and encryption (converts file into an unreadable format).
- **Live CD/USB:** Almost all Linux provide live CD/USB so that users can run/try it without installing it.
- **Application support:** It has its own software repository from where users can download and install many applications.
- **File System:** Provides hierarchical file system in which files and directories are arranged.
- **Open Source:** Linux code is freely available to all and is a community based development project.

➢ Installation and Configuration of LINUX

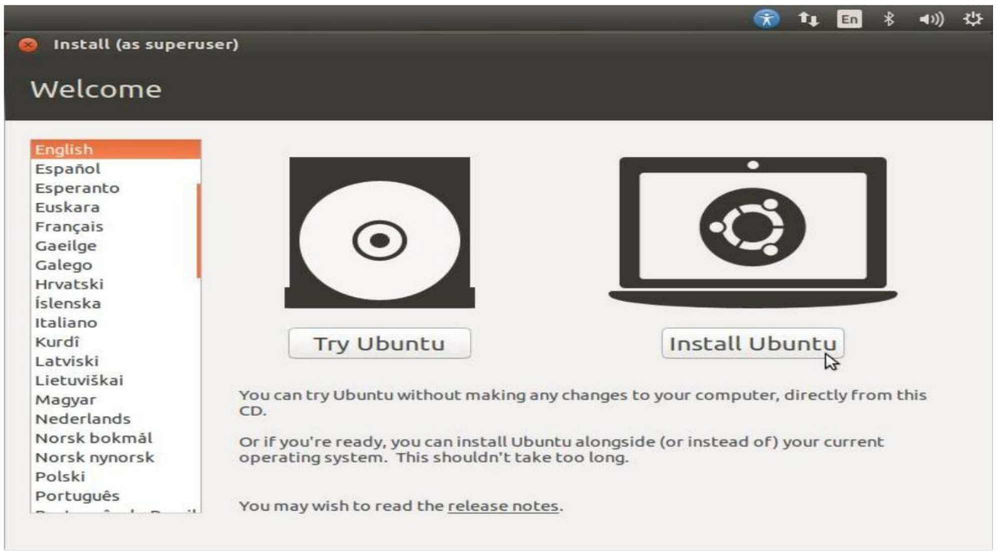
Step:1 Download the ISO file using the following links

https://www.ubuntu.com/download/

Once the iso file has been downloaded, burn it into DVD or USB drive and make it bootable.

Step:2 Boot your system with Bootable DVD / USB drive.

To start the installation, click on —" Install Ubuntu"



You can try Ubuntu without making any changes to your computer, directly from this CD.

Or if you're ready, you can install Ubuntu alongside (or instead of) your current operating system. This shouldn't take too long.

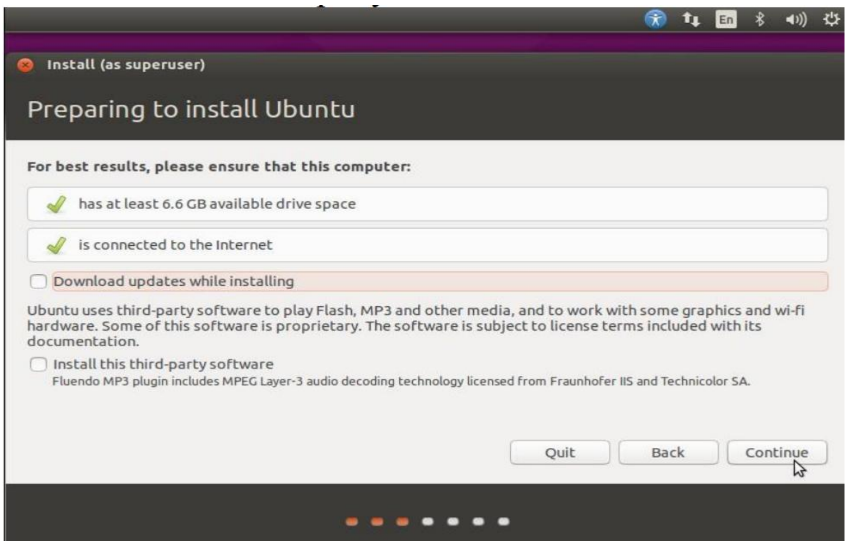
You may wish to read the release notes.

Step:3 Check Install Prerequisite

Make sure you have at least 6.6 GB free disk space and if you want to download updates and third party tools at the installation, check both the boxes and make sure system is connected to Internet.

Download updates while installing

In stall this third-party software



Install (as superuser)

Preparing to install Ubuntu

For best results, please ensure that this computer:

✓ has at least 6.6 GB available drive space

✓ is connected to the Internet

☐ Download updates while installing

Ubuntu uses third-party software to play Flash, MP3 and other media, and to work with some graphics and wi-fi hardware. Some of this software is proprietary. The software is subject to license terms included with its documentation.

☐ Install this third-party software
Fluendo MP3 plugin includes MPEG Layer-3 audio decoding technology licensed from Fraunhofer IIS and Technicolor SA.

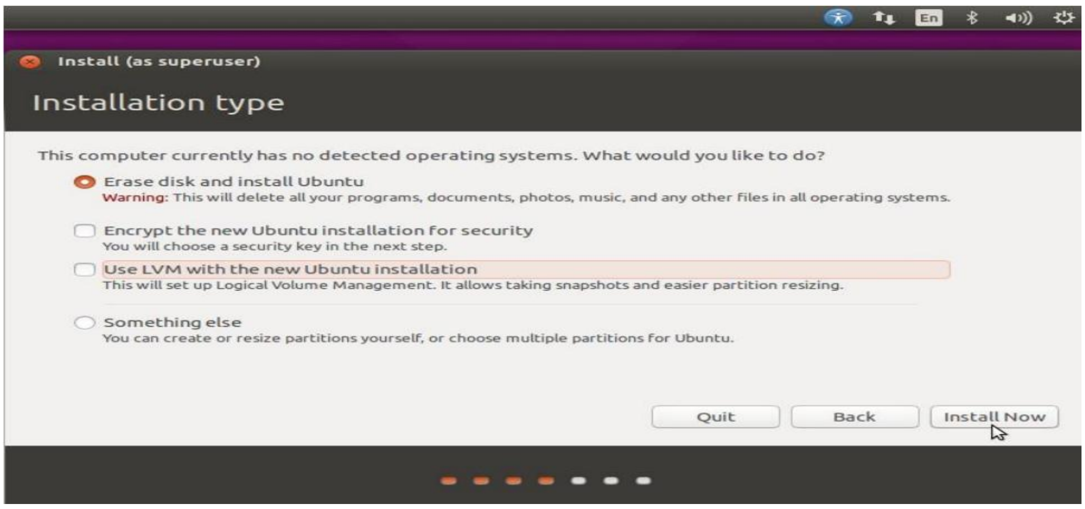
Click on Continue….

Step:4 Select the Installation Type

If your system has new hard drive and if nothing installed on it, then you can choose the first option **“Erase disk & Install Ubuntu”.**

If you want **LVM (Logical Volume Management)** based file system, then use third option **“Use LVM with new Ubuntu Installation”.**

If you want to create customize partition table, the use last option **“Some else”.**



Install (as superuser)

Installation type

This computer currently has no detected operating systems. What would you like to do?

● Erase disk and install Ubuntu
Warning: This will delete all your programs, documents, photos, music, and any other files in all operating systems.

☐ Encrypt the new Ubuntu installation for security
You will choose a security key in the next step.

☐ Use LVM with the new Ubuntu installation
This will set up Logical Volume Management. It allows taking snapshots and easier partition resizing.

○ Something else
You can create or resize partitions yourself, or choose multiple partitions for Ubuntu.

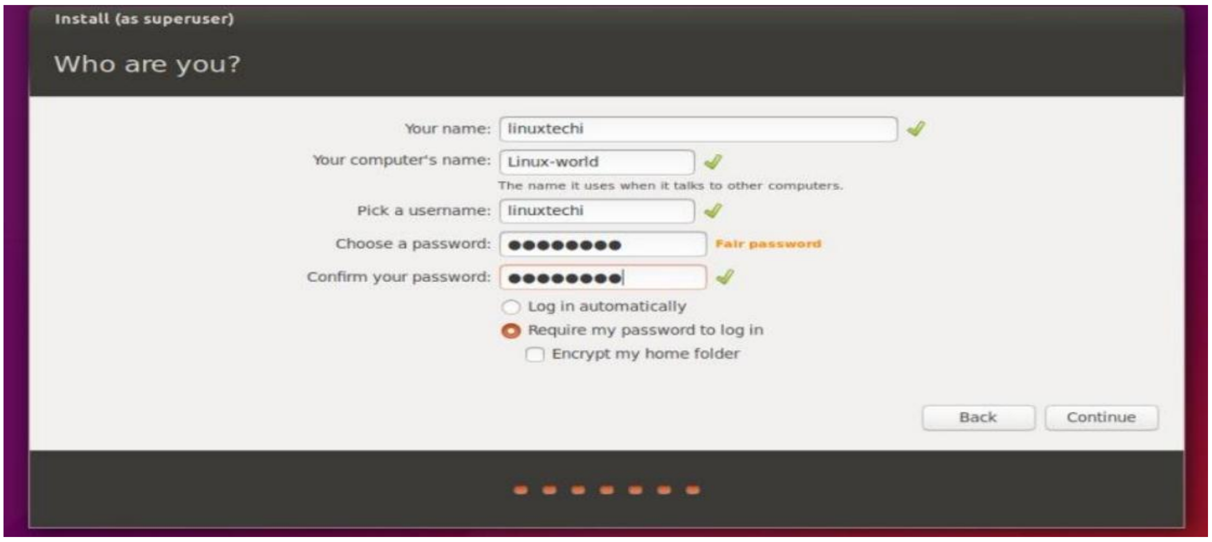
Click on Install Now….

Step:5 Select your respective Time Zone

Step:6 Select your respective Keyboard Layout

Click on Continue…

Step:7 Set the Hostname of your system and User credentials that will be used after installation.



Install (as superuser)

Who are you?

Your name: linuxtechi ✓

Your computer's name: Linux-world ✓

The name it uses when it talks to other computers.

Pick a username: linuxtechi ✓

Choose a password: ●●●●●●●● Fair password

Confirm your password: ●●●●●●●● ✓

Log in automatically [✓ Require my password to log in]

Encrypt my home folder

Click on Continue….

Step:8 Installation has started.

Once the installation is completed, it will ask to restart the Machine.

Step:9 Login Screen after reboot.

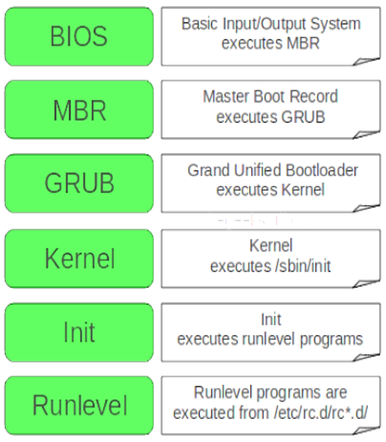
➢ Startup, Shutdown & boot loaders of LINUX

- Linux startup process is the multi-stage initialization process performed during booting a Linux installation.
- For each of these stages and components there are different variations and approaches; for example, GRUB(-GRand Unified Bootloader), LILO(LInux Loader).
- The BIOS performs startup tasks specific to the actual hardware platform.
- The boot loader often presents the user with a menu of possible boot options and has a default option, which is selected after some time passes.
- Once the selection is made, the boot loader loads the kernel into memory, supplies it with some parameters and gives it control.
- **Shutdown**
  - If you are going to initiate a shutdown of your system and there are other users logged in to your system, it is always polite to notify them first.
  - To do this, we would use the shutdown command. The shutdown command will accept times and a broadcast message:
    - shutdown [options] time [warning_message]
  - Shutdown and reboot immediately
    - $ shutdown -r now

PART 2: LINUX BOOTING

➢ LINUX booting process

- The Linux boot process is the name given to the startup procedures/order that your system goes through to load its operating system.
- **Boot Loader**
  - The Ubuntu installation routine writes a new boot sector (also known as boot loader).
  - The boot loader is a separate program called GRUB (Grand Unified Boot Loader).
  - If more than one operating system is installed, GRUB boot loader with list of all installed operating system appears.
  - The first option will be selected automatically within 10 seconds.
  - In the boot loader screen, all pre-installed operating systems are listed, apart from that an entry ending in “(recovery mode)”.


BIOS
Basic Input/Output System executes MBR
MBR
Master Boot Record executes GRUB
GRUB
Grand Unified Bootloader executes Kernel
Kernel
Kernel executes /sbin/init
Init
Init executes runlevel programs
Runlevel
Runlevel programs are executed from /etc/rc.d/rc*.d/

1. **BIOS:**
   i. When we power on BIOS performs a Power-On Self-Test (POST) for all of the different hardware components in the system to make sure everything is working properly.
   ii. Also it checks for whether the computer is being started from an off position (cold boot) or from a restart (warm boot) is stored at this location
2. **MBR:**
   i. MBR stands for Master Boot Record.
   ii. It is located in the 1st sector of the bootable disk.
   iii. MBR is less than 512 bytes in size.
   iv. So, in simple terms MBR loads and executes the GRUB boot loader.
3. **GRUB:**
   i. GRUB stands for Grand Unified Bootloader.
   ii. If you have multiple kernel images installed on your system, you can choose which one to be executed.

   iii. Once user selects the operating system GRUB will pass control to the karnel of that operating system.

4. **Kernel:**
   i. Mounts the root file system as specified in the “root=” in grub.conf
   ii. Kernel executes the /sbin/init program.
5. **Init**
   i. The kernel, once it is loaded, finds init in sbin(/sbin/init) and executes it.
   ii. Hence the first process which is started in linux is init process.
   iii. It runs all the boot scripts(/etc/rc.d/*,/etc/rc.boot/*)
6. **Runlevel**
   i. There are some run levels in which the linux OS runs and different run levels serves for different purpose.
   ii. The descriptions are given below.
      0 – halt
      1 – Single user mode
      2 – Multiuser, without NFS (The same as 3, if you don’t have networking)
      3 – Full multiuser mode
      4 – Reboot

➢ **LILO Configuration**

- LILO (Linux Loader) is a boot loader for Linux and was the default boot loader for most Linux distributions.
- LILO is the LInuxLOader, the most popular boot loader for Linux.
- It is used to load Linux into memory and start the operating system. On a machine with multiple operating systems, LILO can be configured to boot the other systems as well.
- LILO can be customized with a configuration file in the /etc directory.
- The lilo.conf file has its own man page which is quite thorough.
  - **boot=/dev/hda :** Tells LILO where to install the bootloader.
  - **map=/boot/map:** The map file is automatically generated by LILO
  - **install=/boot/boot.b:** Tells LILO what to use as the new boot sector
  - **compact:** Makes LILO read the hard drive faster.
  - **Prompt:** Tells LILO to prompt us at boot time to choose an operating system
  - **timeout=50:** Tells LILO how long to wait at the prompt before booting the default operating system.
  - **image=/boot/vmlinuz-2.0.36:** The name of a Linux kernel for LILO to boot
  - **label=linux:** The name that is used to identify this image at the LILO

    - **root=/dev/hda2:** Tells LILO where the root (/) file system is.
    - **read-only:** Tells LILO to instruct the Linux kernel to initially mount the root file system as read-only.

- **GRUB Configuration**
  - GRUB is a boot loader designed to boot a wide range of operating systems from a wide range of filesystems.
  - GRUB is becoming popular due to the increasing number of possible root filesystems that Linux can reside upon.
  - GRUB is documented in a GNU info file. The GRUB boot loader uses the configuration file /boot/grub/grub.conf.
  - **GRUB Configuration File**
    - default=0 timeout=10
        - splashimage=(hd0,0)/grub/splash.xpm.gz
    - The ‘default =’ option tells GRUB which image to boot by default after the timeout period.
    - The ‘splashimage’ option specifies the location of the image for use as the background for the GRUB GUI.

- **Creating LINUX User Account and Password**
- **Installing and Managing Samba Server**
  - Samba is a software package, released in 1992, gives network administrators flexibility and freedom in terms of setup, configuration and choice of systems and equipment.
  - Because of its ease and flexibility, Samba has become extremely popular and continues to do so.
  - Samba runs on UNIX platforms, but can communicate to Windows clients just like the native platform. It allows a Unix system to move into a Windows Network without causing any mix- up.
  - Windows users can easily access file and print services without having any problem.
  - **Installing and Managing Samba Server and Creating Samba User**

1. Install Samba

   ```
   sudo apt-get update
   sudo apt-get install samba
   ```

   - Creating Samba Test Directory and Files-
     - For this part of the procedure, you'll use the su - (switch user) command to work as root.
   - While logged on as root, create the new directory /smbdemo with the following command: **mkdir /smbdemo**
   - Change the permissions on the new directory to 770 with the following command: **chmod 770 /smbdemo**
   - Navigate to the new directory with the following command: **cd /smbdemo**
   - Add three empty files to the directory with the following command: touch file1 file2 file3

2. Set a password for your user in Samba

   ```
   sudo smbpasswd –a <username>
   ```

3. Create a directory to be shared

   ```
   mkdir /home/<user_name>/<foldername>
   ```

4. Make a safe backup copy of the original smb.conf file to your home folder, in case you make an error

   ```
   sudocp /etc/samba/smb.conf ~
   ```

5. Edit the file "/etc/samba/smb.conf"

   ```
   sudonano /etc/samba/smb.conf
   ```

   Once "smb.conf" has loaded, add this to the very end of the file:

   ```
   [<foldernamme>]
   path = /home/<user_name>/<flodername>
   valid users =<username>
   ```

```
read only = no
```

6. **Restart the samba:**

```
sudo service smbd restart
```

7. **Once Samba has restarted, use this command to check your smb.conf for any syntax errors:**

```
testparm
```

➢ **Installing and Managing Apache Server**

Apache is a freely available Web server that is distributed under an "open source" license. The name 'Apache' was chosen from respect for the Native American Indian tribe of Apache. It is used to serve more than half of all active websites. Developer of apache server is a Apache Software Foundation.

Installing Apache

- Getting apache onto your Ubuntu machine is easy. Using either the Synaptic Package Manager, Ubuntu Software Center, search and install the “apache2” module. Alternatively, you can open a terminal and type the following command:

  ```
  sudo apt-get install apache2
  ```

- Once the installation finished, open a browser and go to the URL “http://localhost”. If you see the word “It Works!”, then your installation of apache is successful.

Configuring Apache

- On Ubuntu, Apache keeps its main configuration files within the "/etc/apache2" folder:

  ```
  cd /etc/apache2
  ls –F
  ```

  apache2.conf: - main configuration file for the server
  conf-available: - contains additional server configurations
  conf-enabled
  envvars: - contains Apache's environment variables.
  Magic: - MIME types
  mods-available: - contains configurations for all installed modules
  mods-enabled
  ports.conf: - the TCP ports that Apache listens to
  sites-available: - contains configurations for your virtual hosts
  sites-enabled

Optimize LDAP Services

- ✓ LDAP (Lightweight Directory Access Protocol) is a protocol for managing information from a centralized location over the use of a file and directory hierarchy.
- ✓ It can be used to organize and store any kind of information.
- ✓ LDAP is mainly used for centralized authentication.

  Install LDAP
- ✓ The OpenLDAP server is in Ubuntu's default package "slapd", so we can install it easily with apt-get.
- ✓ We will also install some additional utilities.

```
sudo apt-get update
sudo apt-get install slapdldap-utils
```

LDAP Configuration Files

- All LDAP configuration files are kept in the /etc/openldap directory. These include slapd.conf, the LDAP server configuration file, and ldap.conf, the LDAP clients and tools configuration file. \
- To enable the LDAP server, you have to manually edit the slapd.conf file and change the domain value (dv) for the suffix and root dn entries to your own network's domain address. This is the network that will be serviced by the LDAP server. (suffix and root dn parameter in a config fie).
- For, clients you can either edit the ldap.conf file directly or use the System Setting Authentication tools, clicking the Configure LDAP button on either the User Information or Authentication panel.

Optimizing DNS Services

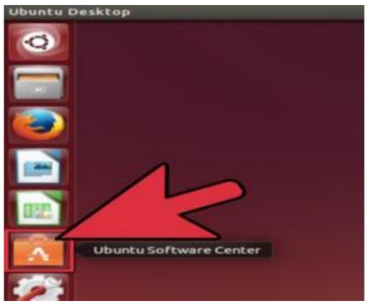
- Domain Name Service (DNS) is an Internet service that maps IP addresses and fully qualified domain names (FQDN) to one another.
- BIND stands for Berkley Internet Naming Daemon.
- BIND is the most common program used for maintaining a name server on Linux.
- Domain Names are hierarchical – there are levels.
  - Top Level Domain (TLD)
  - Second Level Domain (SLD).
- The combination of TLD and SLD forms a domain name. Thus, the general form of a domain name is SLD.TLD.

Configure Ubuntu's Built-in Firewall

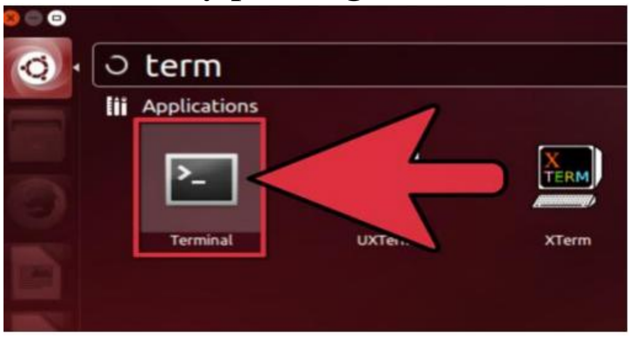
- Firewall is considered as the first method of defense in securing your cloud server.
- Ubuntu includes its own (built-in) Firewall known as UFW.
- UFW stands for "Uncomplicated Firewall".
- Its main goal is to provide an easy-to use interface.
- The firewall is disabled by default. To enable the firewall, run the following commands from a terminal.
  - **sudo ufw enable**
- To install Firewall:
  - **sudo aptitude install ufw Or**
  - **sudo apt-get install ufw**
- To check the status, use following command:
  - **sudoufw status Setup rules**
- You should first define some rules for the firewall for allowing and denying connection.
- To set the defaults used by UFW, use the following commands
  - **sudo ufw default deny incoming And**
  - **sudo ufw default allow outgoing.**
- If you want to deny the outgoing request also, use the following command:
  - **sudo ufw default deny outgoing**
- Enable the SSH connection, use this command:
  - **sudo ufw allow ssh**
  - **Deleting rules**
- There are two option to delete rules.
  - **sudo ufw delete allow ssh**
- If you need to reset your cloud server's rules to their default settings, you can do this :
  - **sudo ufw reset**

Working with WINE

- Wine is an open-source software application that allows Linux users to run Windows programs.
- This can be helpful for programmers, who have to test the compatibility of programs and scripts with Windows.
- Wine also includes a software library (Winelib) to allow developers to compile applications for Windows on Unix-like servers.
- Wine makes it possible to run Windows programs alongside any Unix-like operating system, particularly Linux.
- At its heart, Wine is an implementation of the Windows Application Programming Interface (API) library, acting as a bridge between the Windows program and Linux.
- **Installing Wine**
  1. Open the Software Center. This is Ubuntu's app store, and is the easiest way to install the most stable version of Wine and other software for Ubuntu. You'll need an active internet connection to install it.

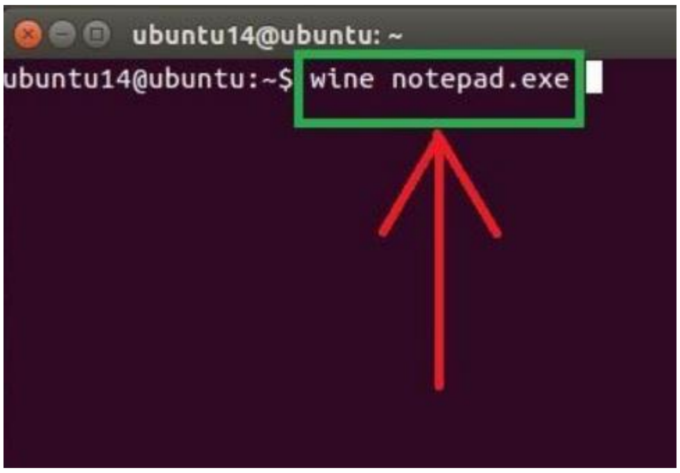


  2. Search for wine in the Software Center. The Wine program should be the first on the list of results.
  3. Click Install to begin installing the Wine software. This may take a few minutes
  4. Open the Terminal after Wine has finished installing. You will need to configure Wine before you can use it, which needs to be done through the Terminal.
     a. **You can open the Terminal from Applications → Accessories → Terminal, or by pressing Ctrl+Alt+T.**



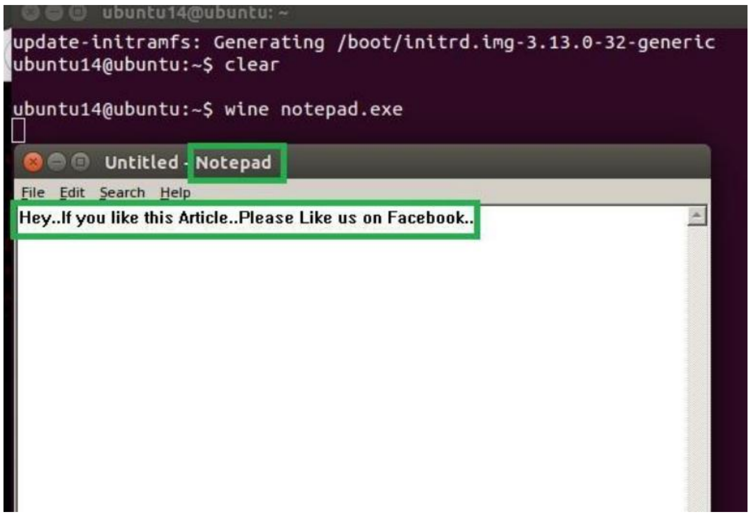
5. Type .winecfg and press ↲ Enter. This will create a folder on your computer that will act as the Windows "C:" drive, which will allow you to run programs. This folder is labeled .wine and is hidden in your Home directory. Wine configuration window will be flashed in the dialog box. For an example: you want to select the version of your Windows system, select it using the Windows Version option.

6. Close it and then move to the Terminal window. Suppose you want to test, if you are able to use the Windows applications or not, then type "wine notepad.exe", and press enter.



```
ubuntu14@ubuntu: ~
ubuntu14@ubuntu:~$ wine notepad.exe
```

7. Notepad screen is available, type whatever you like and then save it. In a similar way, you can use any application as per your requirement.



```
ubuntu14@ubuntu: ~
update-initramfs: Generating /boot/initrd.img-3.13.0-32-generic
ubuntu14@ubuntu:~$ clear

ubuntu14@ubuntu:~$ wine notepad.exe
```

Untitled - Notepad

Hey..If you like this Article..Please Like us on Facebook..