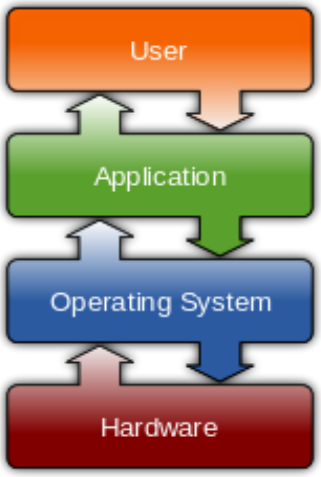
UNIT – 1
Introduction

Operating System



An operating system (OS) is system software that manages computer hardware and software resources and provides common services for computer programs. All computer programs, excluding firmware, require an operating system to function.

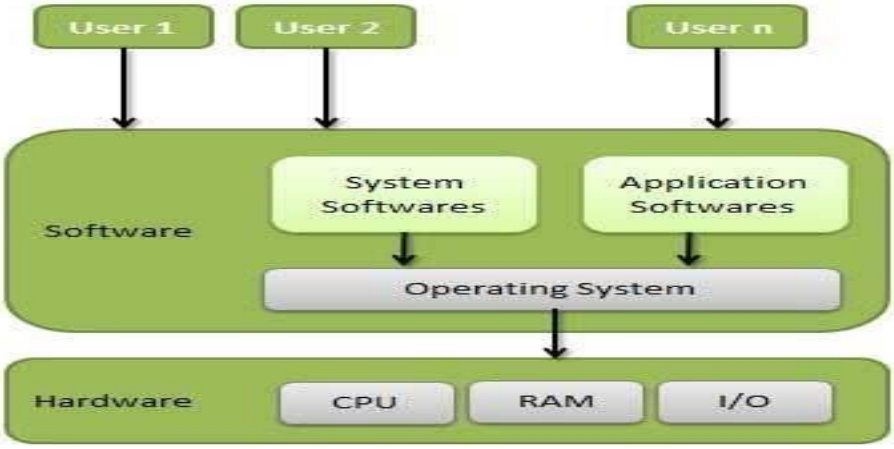
Time-sharing operating systems schedule tasks for efficient use of the system and may also include accounting software for cost allocation of processor time, mass storage, printing, and other resources.

For hardware functions such as input and output and memory allocation, the operating system acts as an intermediary between programs and the computer hardware, although the application code is usually executed directly by the hardware and frequently makes system calls to an OS function or is interrupted by it. Operating systems are found on many devices that contain a computer – from cellular phones and video game consoles to web servers and supercomputers.

The dominant desktop operating system is Microsoft Windows with a market share of around 82%. OS X by Apple Inc. is in second place (9.8%), and Linux is in third position (1.5%). In the mobile (smartphone and tablet combined) sector, based on Strategy Analytics Q3 2016 data, Android by Google is dominant with 87.5 percent or growth by 10.3 percent in one year and iOS by Apple is placed second with 12.1 percent or decrease by 5.2 percent in one year, while other operating system gets only 0.3 percent. Linux is dominant in the server and supercomputing sectors. Other specialized classes of operating systems, such as embedded and real-time systems, exist for many applications.

OPERATING SYSTEM CONCEPTS

Definition: An operating system is a program that acts as an interface between the user and the computer hardware and controls the execution of all kinds of programs.



Functions of Operating System.

Memory Management

Memory management refers to management of Primary Memory or Main Memory. Main memory is a large array of words or bytes where each word or byte has its own address.

Main memory provides a fast storage that can be accessed directly by the CPU. For a program to be executed, it must in the main memory. An Operating System does the following activities for memory management −

- Keeps tracks of primary memory, i.e., what part of it are in use by whom, what parts are not in uses?
- In multiprogramming, the OS decides which process will get memory when and how much.
- Allocates the memory when a process requests it to do so.
- De-allocates the memory when a process no longer needs it or has been terminated.

Processor Management

In multiprogramming environment, the OS decides which process gets the processor when and for how much time. This function is called process scheduling. An Operating System does the following activities for processor management −

- Keeps tracks of processor and status of process. The program responsible for this task is known as traffic controller.
- Allocates the processor (CPU) to a process.
- De-allocates processor when a process is no longer required.

Device Management

An Operating System manages device communication via their respective drivers. It does the following activities for device management −

- Keeps tracks of all devices. Program responsible for this task is known as the I/O controller.
- Decides which process gets the device when and for how much time.
- Allocates the device in the efficient way.
- De-allocates devices.

File Management

A file system is normally organized into directories for easy navigation and usage. These directories may contain files and other directions.

An Operating System does the following activities for file management −

- Keeps track of information, location, uses, status etc. The collective facilities are often known as file system.
- Decides who gets the resources.
- Allocates the resources.
- De-allocates the resources.

Other Important Activities

Following are some of the important activities that an Operating System performs −

- **Security** − By means of password and similar other techniques, it prevents unauthorized access to programs and data.
- **Control over system performanc**e − Recording delays between request for a service and response from the system.

- **Job accounting** – Keeping track of time and resources used by various jobs and users.
- **Error detecting aids** – Production of dumps, traces, error messages, and other debugging and error detecting aids.
- **Coordination between other software and users** – Coordination and assignment of compilers, interpreters, assemblers and other software to the various users of the computer systems.

Features of OS

- **Software and hardware management**

The operating system is the bridge between computer hardware and software and makes the communication between them possible. Also communication between different software's in the computer is also taken care by operating system.

- **Constant API**

Application Program Interface (API) is software that allows different applications that run on a computer to work on other computers also. But they should have same operating system. So it is very vital to have consistent API in the operating system.

- **Execution of programs**

Programs running in the computer are completely dependent on the operating system. But program execution is a tough process. The multitasking and multithreading features of the operating system are dependent upon the type of program execution feature of O.S.

- **Interruptions**

Interruption may happen at any time while using the computers. So the operating system should allow and handle many numbers of interrupts. Whenever an interruption occurs, the operating system should respond to it by saving and stopping the current execution and work on the new execution. This is the most hard-hitting process for the operating system.

- **Managing memory**

The operating system provides the memory for the programs that are executed at any moment. So the operating system should have good memory allocation facility to execute the programs smoothly. The prioritization and allocation of memory to the applications running should be taken care by the operating systems.

- **Networking**

Today computers are nothing without internet connection or some network connection. This is the age of networking. So if computers are connected to a network, the there should be definitely communication between one computer and another. So, the operating system is what makes it possible for one computer to communicate with other computers.

- **Security**

Security is the important feature that should be looked for in an operating system. An operating system in the computer takes care of all security issues of computer and data in it. Log in passwords, firewall settings, and every such aspect related to security depends on the ability of the operating system. Some of the computers in network may involve in file sharing, and other data sharing. So, it is important in such cases to have powerful secured operating systems.

Types of OS (USER POINT OF VIEW)

Operating systems are there from the very first computer generation and they keep evolving with time. We will discuss some of the important types of operating systems which are most commonly used.

Single User Operating System

In a single user operating system, a single user can access the computer system at a time. These types of operating systems are commonly found in home computers. There are two types of single user operating systems called single user, single task operating system and single user, multi-task operating system.

In a single user, single task operating system, a single user can perform only one task at a time. Palm OS for Palm handheld computers is an example for a single user, single task operating system. In a single user multitask system; a single user can perform multiple tasks at the same time. Microsoft Windows and Apple Mac OS allow a single user to work on multiple programs at the same time.

For example, a user can work on a word document and browser the World Wide Web simultaneously. Most modern personal computers and laptops are single user multi-tasking operating systems.

Multi User Operating System

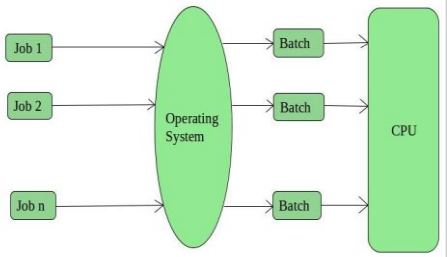
A multi user operating system allows multiple users to access the computer at the same time. The operating system manages the memory and resources among the various users according to the requirements. The task of one user will not affect the tasks of the other users. UNIX and Linux are two examples of multi user operating systems.

A time sharing operating system allows multiple users in different locations to use a particular computer system concurrently. In distributed operating system, the data processing task is divided among the processors accordingly. It is also a multiuser operating system.

Types of OS (FEATURES POINT OF VIEW)

Batch operating system (BOS)

The users of a batch operating system do not interact with the computer directly. Each user prepares his job on an off-line device like punch cards and submits it to the computer operator. To speed up processing, jobs with similar needs are batched together and run as a group. The programmers leave their programs with the operator and the operator then sorts the programs with similar requirements into batches.



The problems with Batch Systems are as follows

- Lack of interaction between the user and the job.
- CPU is often idle, because the speed of the mechanical I/O devices is slower than the CPU.
- Difficult to provide the desired priority.

Advantages:

- It is very difficult to guess or know the time required by any job to complete. Processors of the batch systems knows how long the job would be when it is in queue
- Multiple users can share the batch systems
- The idle time batch system is very less

- It is easy to manage large work repeatedly in batch systems

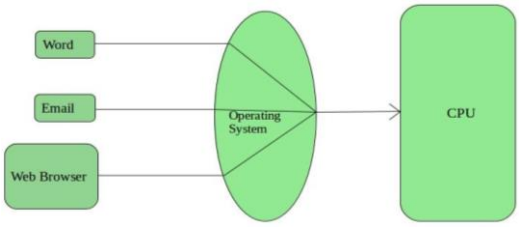
Disadvantages:

- The computer operators should be well known with batch systems
- Batch systems are hard to debug
- It is sometime costly
- The other jobs will have to wait for an unknown time if any job fails

Example: Payroll System, Bank Statements etc.

Time-sharing operating systems(TSOS)

Each task has given some time to execute, so that all the tasks work smoothly. Each user gets time of CPU as they use single system. These systems are also known as Multitasking Systems. The task can be from single user or from different users also. The time that each task gets to execute is called quantum. After this time interval is over OS switches over to next task.



Advantages:

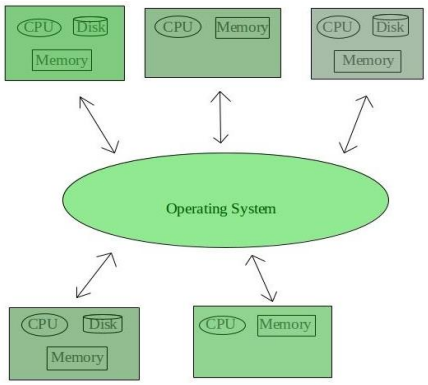
- Each task gets an equal opportunity
- Less chances of duplication of software
- CPU idle time can be reduced

Disadvantages:

- Reliability problem
- One must have to take care of security and integrity of user programs and data
- Data communication problem

Examples: Multics, UNIX etc.

Distributed Operating System (DOS)



These types of operating system are a recent advancement in the world of computer technology and are being widely accepted all-over the world and, that too, with a great pace. Various autonomous interconnected computers communicate each other using a shared communication network. Independent systems possess their own memory unit and CPU. These are referred as **loosely coupled systems** or distributed systems.

These systems processors differ in sizes and functions. The major benefit of working with these types of operating system is that it is always possible that one user can access the files or software which are not actually present on his system but on some other system connected within this network i.e., remote access is enabled within the devices connected in that network.

Advantages:

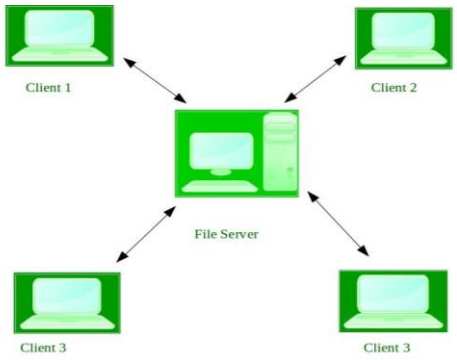
- Failure of one will not affect the other network communication, as all systems are independent from each other
- Electronic mail increases the data exchange speed
- Since resources are being shared, computation is highly fast and durable
- Load on host computer reduces
- These systems are easily scalable as many systems can be easily added to the network
- Delay in data processing reduces

Disadvantages:

- Failure of the main network will stop the entire communication
- To establish distributed systems the language which are used are not well defined yet
- These types of systems are not readily available as they are very expensive. Not only that the underlying software is highly complex and not understood well yet

Example: LOCUS etc.

Network Operating System (NOS)



This system run on a server and provides the capability to manage data, users, groups, security, applications, and other networking functions. These types of operating systems allow shared access of files, printers, security, applications, and other networking functions over a small private network. One more important aspect of Network Operating Systems is that all the users are well aware of the underlying configuration, of all other users within the network, their individual connections etc. and that's why these computers are popularly known as tightly coupled systems.

Advantages:

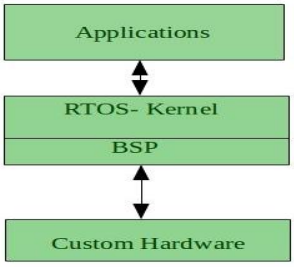
- Highly stable centralized servers
- Security concerns are handled through servers
- New technologies and hardware up-gradation are easily integrated to the system
- Server access are possible remotely from different locations and types of systems

Disadvantages:

- Servers are costly
- User has to depend on central location for most operations
- Maintenance and updates are required regularly

Examples: Microsoft Windows Server 2003, Microsoft Windows Server 2008, UNIX, Linux, Mac OS X, Novell NetWare, and BSD etc.

Real-Time Operating System (RTOS)



This type of OSs serves the real-time systems. The time interval required to process and respond to inputs is very small. This time interval is called **response time**.

Real-time systems are used when there are times requirements are very strict like missile systems, air traffic control systems, robots etc.

Two types of Real-Time Operating System which are as follows:

- **Hard Real-Time Systems:**
  These OSs are meant for the applications where time constraints are very strict and even the shortest possible delay is not acceptable. These systems are built for saving life like automatic parachutes or air bags which are required to be readily available in case of any accident. Virtual memory is almost never found in these systems.
- **Soft Real-Time Systems:**
  These OSs are for applications where for time-constraint is less strict.

Advantages:

- **Maximum Consumption:** Maximum utilization of devices and system, thus more output from all the resources
- **Task Shifting:** Time assigned for shifting tasks in these systems are very less. For example, in older systems it takes about 10 micro seconds in shifting one task to another and in latest systems it takes 3 micro seconds.
- **Focus on Application:** Focus on running applications and less importance to applications which are in queue.
- **Real time operating system in embedded system:** Since size of programs is small, RTOS can also be used in embedded systems like in transport and others.
- **Error Free:** These types of systems are error free.
- **Memory Allocation:** Memory allocation is best managed in these types of systems.

Disadvantages:

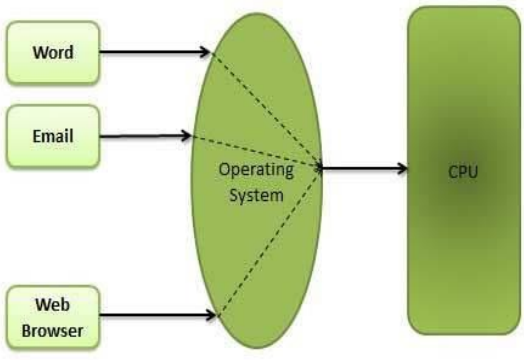
- **Limited Tasks:** Very few tasks run at the same time and their concentration is very less on few applications to avoid errors.
- **Use heavy system resources:** Sometimes the system resources are not so good and they are expensive as well.
- **Complex Algorithms:** The algorithms are very complex and difficult for the designer to write on.
- **Device driver and interrupt signals:** It needs specific device drivers and interrupts signals to response earliest to interrupts.
- **Thread Priority:** It is not good to set thread priority as these systems are very less prom to switching tasks.

Examples: Scientific experiments, medical imaging systems, industrial control systems, weapon systems, robots, air traffic control systems, etc.

Multitasking

Multitasking is when multiple jobs are executed by the CPU simultaneously by switching between them. Switches occur so frequently that the users may interact with each program while it is running. An OS does the following activities related to multitasking

- The user gives instructions to the operating system or to a program directly, and receives an immediate response.
- The OS handles multitasking in the way that it can handle multiple operations/executes multiple programs at a time.
- Multitasking Operating Systems are also known as Time-sharing systems.
- These Operating Systems were developed to provide interactive use of a computer system at a reasonable cost.



- A time-shared operating system uses the concept of CPU scheduling and multiprogramming to provide each user with a small portion of a time-shared CPU.
- Each user has at least one separate program in memory.
- A program that is loaded into memory and is executing is commonly referred to as a process.

- When a process executes, it typically executes for only a very short time before it either finishes or needs to perform I/O.
- Since interactive I/O typically runs at slower speeds, it may take a long time to complete. During this time, a CPU can be utilized by another process.
- The operating system allows the users to share the computer simultaneously. Since each action or command in a time-shared system tends to be short, only a little CPU time is needed for each user.
- As the system switches CPU rapidly from one user/program to the next, each user is given the impression that he/she has his/her own CPU, whereas actually one CPU is being shared among many users.

Multiprogramming

Sharing the processor, when two or more programs reside in memory at the same time, is referred as multiprogramming. Multiprogramming assumes a single shared processor. Multiprogramming increases CPU utilization by organizing jobs so that the CPU always has one to execute.

The following figure shows the memory layout for a multiprogramming system.

An OS does the following activities related to multiprogramming.

- The operating system keeps several jobs in memory at a time.
- This set of jobs is a subset of the jobs kept in the job pool.
- The operating system picks and begins to execute one of the jobs in the memory.
- Multiprogramming operating systems monitor the state of all active programs and system resources using memory management programs to ensure that the CPU is never idle, unless there are no jobs to process.

Advantages

- High and efficient CPU utilization.
- User feels that many programs are allotted CPU almost simultaneously.

Disadvantages

- CPU scheduling is required.
- To accommodate many jobs in memory, memory management is required.

Interactivity

Interactivity refers to the ability of users to interact with a computer system. An Operating system does the following activities related to interactivity –

- Provides the user an interface to interact with the system.
- Manages input devices to take inputs from the user. For example, keyboard.
- Manages output devices to show outputs to the user. For example, Monitor.

The response time of the OS needs to be short, since the user submits and waits for the result.

UNIT – 1
Process and Thread

Introduction of OS Process

A program/command when executed, a special instance is provided by the system to the process. This instance consists of all the services/resources that may be utilized by the process under execution.

- Whenever a command is issued in unix/linux, it creates/starts a new process. For example, pwd when issued which is used to list the current directory location the user is in, a process starts.
- Through a 5-digit ID number unix/linux keeps account of the processes, this number is call process id or pid. Each process in the system has a unique pid.
- Used up pid's can be used in again for a newer process since all the possible combinations are used.
- At any point of time, no two processes with the same pid exist in the system because it is the pid that Unix uses to track each process.

Initializing a process

A process can be run in two ways:

1. **Foreground Process:**
   Every process when started runs in foreground by default, receives input from the keyboard and sends output to the screen.
   When a command/process is running in the foreground and is taking a lot of time, no other processes can be run or started because the prompt would not be available until the program finishes processing and comes out.
2. **Background Process:**
   It runs in the background without keyboard input and waits till keyboard input is required. Thus, other processes can be done in parallel with the process running in background since they do not have to wait for the previous process to be completed.

PID (Process Identification Number)

PID is an UNIQUE identification number that is automatically assigned to each process when it is created on an operating system.

Processor

A processor fetches the instruction from the main memory, decodes it and executes one instruction at a time. It fetches the next instruction in a sequence.

Register

Registers are normally measured by the number of bits they can hold.

For example

An "8-bit register" or a "32-bit register". A processor often contains several kinds of registers, that can be classified according to their content or instructions that operate on them:

Program Counter (PC):

A program counter is a register in a computer processor that contains the address (location) of the instruction being executed at the current time. As each instruction gets fetched, the program counter increases its stored value by 1. After each instruction is

fetched, the program counter points to the next instruction in the sequence. When the computer restarts or is reset, the program counter normally reverts to 0.

Instruction Register (IR)

The instruction whose address is fetched by the program counter is loaded in the instruction register.

Memory Address Register (MAR)

If contains the address of the memory location to be accessed. It is directly connected with address

Memory Data Register (MDR)

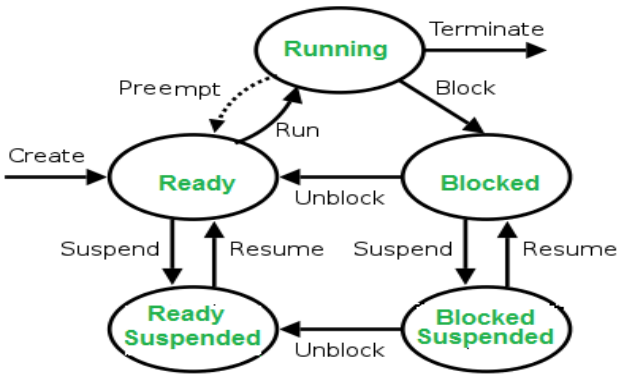
If data is to be brought from main memory, it is first loaded into the data register and then if can be passed to other register. It is directly connected to the external data bus.

What is Program?

A program is a collection of instructions that performs a specific task when executed by a computer. A computer requires programs to function, and typically executes the program's instructions in a central processing unit.

Process State Transition Diagram

The following typical process states are possible on computer systems of all kinds. In most of these states, processes are "stored" on main memory.



- **New (Create) –** In this step, process is about to be created but not yet created, it is the program which is present in secondary memory that will be picked up by OS to create the process.
- **Ready –** New -> Ready to run. After creation of a process, the process enters the ready state i.e., the process is loaded into the main memory. The process here is ready to run and is waiting to get the CPU time for its execution. Processes that are ready for execution by the CPU are maintained in a queue for ready processes.
- **Run –** The process is chosen by CPU for execution and the instructions within the process are executed by any one of the available CPU cores.
- **Blocked or wait –** Whenever the process requests access to I/O or needs an input from user or needs access to a critical region (the lock for which is already acquired) it enters the blocked or wait state. The process continues to wait in the main memory and does not require CPU. Once the I/O operation is completed the process goes to ready state.
- **Terminated or completed –** Process is killed as well as PCB is deleted.
- **Suspend ready –** Process that were initially in ready state but were swapped out of main memory (refer Virtual Memory topic) and placed onto external storage by scheduler are said to be in suspend ready state. The process will transition back to ready state whenever the process is again brought onto the main memory.

- **Suspend wait or suspend blocked –** Similar to suspend ready but uses the process which was performing I/O operation and lack of main memory caused them to move to secondary memory. When work is finished, it may go to suspend ready.

Process Control Block (PCB)

PCB stores many different items of data, all needed for correct and efficient process management. Though the details of these structures are obviously system-dependent, we can identify some very common parts, and classify them in three main categories:

1. Process identification data
2. Process state data
3. Process control data

The approach commonly followed to represent this information is to create and update status tables for each relevant entity, like memory, I/O devices, files and processes.

Memory tables may contain information about the allocation of main and secondary (virtual) memory for each process, authorization attributes for accessing memory areas shared among different processes, etc. I/O tables may have entries stating the availability of a device or its assignment to a process, the status of I/O operations being executed, the location of memory buffers used for them, etc.

File tables provide info about location and status of files. Finally, process tables store the data the OS needs to manage processes. At least part of the process control data structure is always maintained in main memory, though its exact location and configuration varies with the OS and the memory management technique it uses.

Process identification data always include a unique identifier for the process (almost invariably an integer number) and, in a multiuser-multitasking system, data like the identifier of the parent process, user identifier, user group identifier, etc. The process id is particularly relevant, since it is often used to cross-reference the OS tables defined above, e.g., allowing identifying which process is using which I/O devices, or memory areas.

Process state data are those pieces of information that define the status of a process when it is suspended, allowing the OS to restart it later and still execute correctly. This always includes the content of the CPU general-purpose registers, the CPU process status word, stack and frame pointers etc. During context switch, the running process is stopped and another process is given a chance to run. The kernel must stop the execution of the running process, copy out the values in hardware registers to its PCB, and update the hardware registers with the values from the PCB of the new process.

Process control information is used by the OS to manage the process itself. This includes:

- **The process scheduling state:** In terms of "ready", "suspended", etc., and other scheduling information as well, like a priority value, the amount of time elapsed since the process gained control of the CPU or since it was suspended. Also, in case of a suspended process, event identification data must be recorded for the event the process is waiting for.

- **Process structuring information:** process's children id's, or the id's of other processes related to the current one in some functional way, which may be represented as a queue, a ring or other data structures.
- **Inter process communication information:** various flags, signals and messages associated with the communication among independent processes may be stored in the PCB.
- **Process privileges,** in terms of allowed/disallowed access to system resources.
- **Process state:** State may enter into new, ready, running, waiting, dead depending on CPU scheduling.
- **Process No:** a unique identification number for each process in the operating system.
- **Program counter:** a pointer to the address of the next instruction to be executed for this process.
- **CPU registers:** indicates various register set of CPU where process need to be stored for execution for running state.
- **CPU scheduling information**: indicates the information of a process with which it uses the CPU time through scheduling.
- **Memory management information:** includes the information of page table, memory limits, Segment table depending on memory used by the operating system.
- **Accounting information:** includes the amount of CPU used for process exec
- **IO status information:** includes a list of I/O devices allocated to the process.

Context Switch

The Context switching is a technique or method used by the operating system to switch a process from one state to another to execute its function using CPUs in the system. When switching perform in the system, it stores the old running process's status in the form of registers and assigns the CPU to a new process to execute its tasks. While a new process is running in the system, the previous process must wait in a ready queue. The execution of the old process starts at that point where another process stopped it. It defines the characteristics of a multitasking operating system in which multiple processes shared the same CPU to perform multiple tasks without the need for additional processors in the system.

WHY USE?

A context switching helps to share a single CPU across all processes to complete its execution and store the system's tasks status. When the process reloads in the system, the execution of the process starts at the same point where there is conflicting.

Following are the reasons that describe the need for context switching in the Operating system.

1. The switching of one process to another process is not directly in the system. A context switching helps the operating system that switches between the multiple processes to use the CPU's resource to accomplish its tasks and store its context. We can resume the service of the process at the same point later. If we do not store the currently running process's data or context, the stored data may be lost while switching between processes.
2. If a high priority process falls into the ready queue, the currently running process will be shut down or stopped by a high priority process to complete its tasks in the system.

3. If any running process requires I/O resources in the system, the current process will be switched by another process to use the CPUs. And when the I/O requirement is met, the old process goes into a ready state to wait for its execution in the CPU. Context switching stores the state of the process to resume its tasks in an operating system. Otherwise, the process needs to restart its execution from the initials level.
4. If any interrupts occur while running a process in the operating system, the process status is saved as registers using context switching. After resolving the interrupts, the process switches from a wait state to a ready state to resume its execution at the same point later, where the operating system interrupted occurs.
5. A context switching allows a single CPU to handle multiple process requests simultaneously without the need for any additional processors.

Example

Suppose that multiple processes are stored in a **Process Control Block (PCB).** One process is running state to execute its task with the use of CPUs. As the process is running, another process arrives in the ready queue, which has a high priority of completing its task using CPU. Here we used context switching that switches the current process with the new process requiring the CPU to finish its tasks. While switching the process, a context switch saves the status of the old process in registers. When the process reloads into the CPU, it starts the execution of the process when the new process stops the old process. If we do not save the state of the process, we have to start its execution at the initial level. In this way, context switching helps the operating system to switch between the processes, store or reload the process when it requires executing its tasks.

Context switching triggers

Following are the three types of contexts switching triggers as follows.

1. Interrupts
2. Multitasking
3. Kernel/User switch

- **Interrupts**: A CPU requests for the data to read from a disk, and if there are any interrupts, the context switching automatic switches a part of the hardware that requires less time to handle the interrupts.
- **Multitasking**: A context switching is the characteristic of multitasking that allows the process to be switched from the CPU so that another process can be run. When switching the process, the old state is saved to resume the process's execution at the same point in the system.
- **Kernel/User Switch**: It is used in the operating systems when switching between the user mode, and the kernel/user mode is performed.

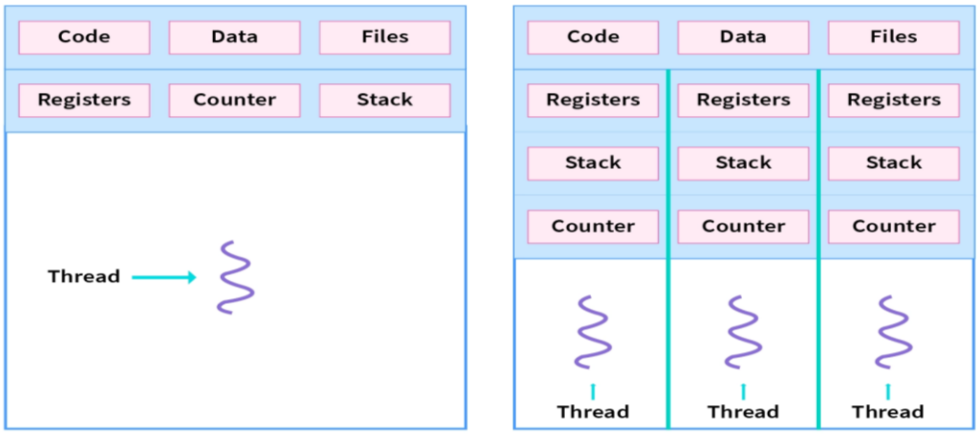
Thread

Thread is a sequential flow of tasks within a process. Threads in an operating system can be of the same or different types. Threads are used to increase the performance of the applications.

Each thread has its own program counter, stack, and set of registers. However, the threads of a single process might share the same code and data/file. Threads are also termed lightweight processes as they share common resources.

Eg: While playing a movie on a device the audio and video are controlled by different threads in the background.

The above diagram shows the difference between a single-threaded process and a multithreaded process and the resources that are shared among threads in a multithreaded process.



Single-threaded process Multithreaded process

Multithreading

In Multithreading, the idea is to divide a single process into multiple threads instead of creating a whole new process. Multithreading is done to achieve parallelism and to improve the performance of the applications as it is faster in many ways which were discussed above. The other advantages of multithreading are mentioned below.

- **Resource Sharing:** Threads of a single process share the same resources such as code, data/file.
- **Responsiveness:** Program responsiveness enables a program to run even if part of the program is blocked or executing a lengthy operation. Thus, increasing the responsiveness to the user.
- **Economy:** It is more economical to use threads as they share the resources of a single process. On the other hand, creating processes is expensive.

Benefits of Threads

Enhanced throughput of the system: When the process is split into many threads, and each thread is treated as a job, the number of jobs done in the unit time increases. That is why the throughput of the system also increases.

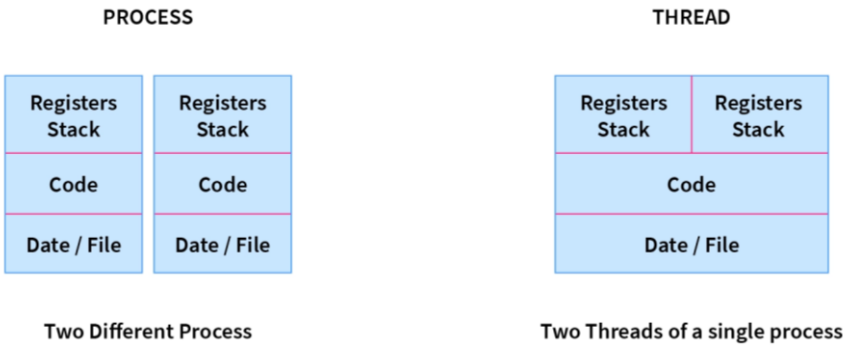
- **Effective Utilization of Multiprocessor system:** When you have more than one thread in one process, you can schedule more than one thread in more than one processor.
- **Faster context switch:** The context switching period between threads is less than the process context switching. The process context switch means more overhead for the CPU.
- **Responsiveness:** When the process is split into several threads, and when a thread completes its execution, that process can be responded to as soon as possible.
- **Communication:** Multiple-thread communication is simple because the threads share the same address space, while in process, we adopt just a few exclusive communication strategies for communication between two processes.

- **Resource sharing:** Resources can be shared between all threads within a process, such as code, data, and files. Note: The stack and register cannot be shared between threads. There is a stack and register for each thread.

Process v/s Thread

Process simply means any program in execution while the thread is a segment of a process. The main differences between process and thread are mentioned below:

Process	Thread
Processes use more resources and hence they are termed as heavyweight processes.	Threads share resources and hence they are termed as lightweight processes.
Creation and termination times of processes are slower.	Creation and termination times of threads are faster compared to processes.
Processes have their own code and data/file.	Threads share code and data/file within a process.
Communication between processes is slower.	Communication between threads is faster.
Context Switching in processes is slower.	Context switching in threads is faster.
Processes are independent of each other.	Threads, on the other hand, are interdependent. (i.e they can read, write or change another thread's data)
Eg: Opening two different browsers.	Eg: Opening two tabs in the same browser.



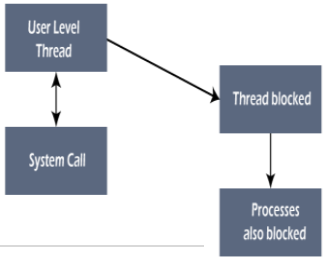
Types of Threads

In the operating system, there are two types of threads.

1. Kernel level thread.
2. User-level thread.

User-level thread

The operating system does not recognize the user-level thread. User threads can be easily implemented and it is implemented by the user. If a user performs a user-level thread blocking operation, the whole process is blocked. The kernel level thread does not know nothing about the user level thread. The kernel-level thread manages user-level threads



as if they are single-threaded processes? examples: Java thread, POSIX threads, etc.

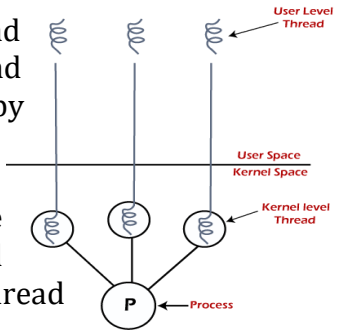
Advantages

- The user threads can be easily implemented than the kernel thread.
- User-level threads can be applied to such types of operating systems that do not support threads at the kernel-level.
- It is faster and efficient.
- Context switch time is shorter than the kernel-level threads.
- It does not require modifications of the operating system.
- User-level threads representation is very simple. The register, PC, stack, and mini thread control blocks are stored in the address space of the user-level process.
- It is simple to create, switch, and synchronize threads without the intervention of the process.

Disadvantages

- User-level threads lack coordination between the thread and the kernel.
- If a thread causes a page fault, the entire process is blocked.

Kernel level thread



The kernel thread recognizes the operating system. There is a thread control block and process control block in the system for each thread and process in the kernel-level thread. The kernel-level thread is implemented by the operating system. The kernel knows about all the threads and manages them. The kernel-level thread offers a system call to create and manage the threads from user-space. The implementation of kernel threads is more difficult than the user thread. Context switch time is longer in the kernel thread. If a kernel thread performs a blocking operation, the Banky thread execution can continue. Example: Window Solaris.

Advantages

The kernel-level thread is fully aware of all threads.

- The scheduler may decide to spend more CPU time in the process of threads being large numerical.
- The kernel-level thread is good for those applications that block the frequency.
- Disadvantages of Kernel-level threads
- The kernel thread manages and schedules all threads.
- The implementation of kernel threads is difficult than the user thread.
- The kernel-level thread is slower than user-level threads.

Components of Threads

Any thread has the following components.

- Program counter
- Register set
- Stack space

UNIT – 1
Process Scheduling

Schedulers

Schedulers are special system software which handles process scheduling in various ways. Their main task is to select the jobs to be submitted into the system and to decide which process to run. Schedulers are of three types

- Long-Term Scheduler
- Short-Term Scheduler
- Medium-Term Scheduler

- **Long Term Scheduler**

It is also called a **job scheduler**. A long-term scheduler determines which programs are admitted to the system for processing. It selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling.

The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound. It also controls the degree of multiprogramming. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.

On some systems, the long-term scheduler may not be available or minimal. Time-sharing operating systems have no long term scheduler. When a process changes the state from new to ready, then there is use of long-term scheduler.

- **Short Term Scheduler**

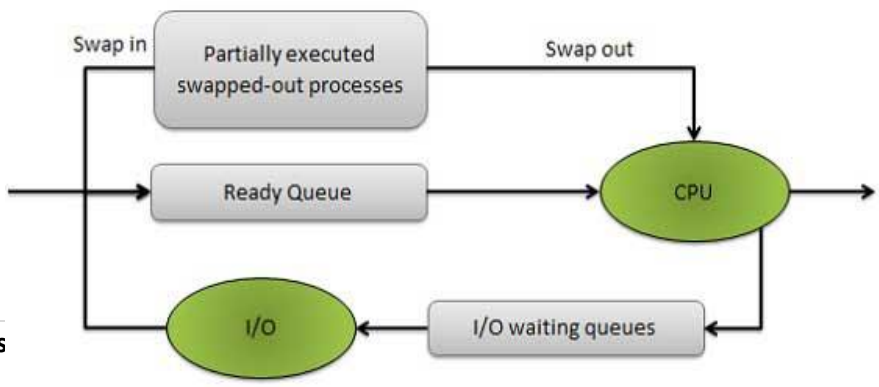
It is also called as **CPU scheduler**. Its main objective is to increase system performance in accordance with the chosen set of criteria. It is the change of ready state to running state of the process. CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them.

Short-term schedulers, also known as dispatchers, make the decision of which process to execute next. Short-term schedulers are faster than long-term schedulers.

- **Medium Term Scheduler**

Medium-term scheduling is a part of **swapping**. It removes the processes from the memory. It reduces the degree of multiprogramming. The medium-term scheduler is in-charge of handling the swapped out-processes.

A running process may become suspended if it makes an I/O request. Suspended processes cannot make any progress towards completion. In this condition, to remove the process from memory and make space for other processes, the suspended process is moved to the secondary storage. This process is called **swapping**, and the process is said to be swapped out or rolled out. Swapping may be necessary to improve the process mix.



Long-Term Scheduler	Short-Term Scheduler	Medium-Term Scheduler
It is a job scheduler	It is a CPU scheduler	It is a process swapping scheduler.
Speed is lesser than short term scheduler	Speed is fastest among other two	Speed is in between both short - and long - term scheduler.
It controls the degree of multiprogramming	It provides lesser control over degree of multiprogramming	It reduces the degree of multiprogramming.
It is almost absent or minimal in time sharing system	It is also minimal in time sharing system	It is a part of Time-sharing systems.
It selects processes from pool and loads them into memory for execution	It selects those processes which are ready to execute	It can re-introduce the process into memory and execution can be continued.

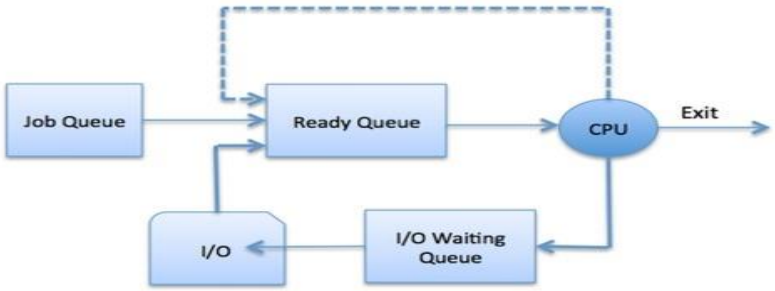
Process Scheduling Queues

The OS maintains all PCBs in Process Scheduling Queues. The OS maintains a separate queue for each of the process states and PCBs of all processes in the same execution state are placed in the same queue. When the state of a process is changed, its PCB is unlinked from its current queue and moved to its new state queue.

The Operating System maintains the following important process scheduling queues

- **Job queue:** This queue keeps all the processes in the system.
- **Ready queue:** This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.
- **Device queues:** The processes which are blocked due to unavailability of an I/O device constitute this queue.

The OS can use different policies to manage each queue (FIFO, Round Robin, Priority, etc.). The OS scheduler determines how to move processes between the ready and run queues which can only have one entry per processor core on the system; in the above diagram, it has been merged with the CPU.



Scheduling Criteria

Scheduling criteria is also called as scheduling methodology. Key to multiprogramming is scheduling. Different CPU scheduling algorithm have different properties. The criteria used for comparing these algorithms include the following:

- **CPU Utilization:** Keep the CPU as busy as possible. It ranges from 0 to 100%. In practice, it ranges from 40 to 90%.
- **Throughput:** Throughput is the rate at which processes are completed per unit of time.
- **Turnaround time:** This is the how long a process takes to execute a process. It is calculated as the time gap between the submission of a process and its completion.
- **Waiting time:** Waiting time is the sum of the time periods spent in waiting in the ready queue.
- **Response time:** Response time is the time it takes to start responding from submission time. It is calculated as the amount of time it takes from when a request was submitted until the first response is produced.
- **Fairness:** Each process should have a fair share of CPU.

Process Scheduling Policies

The act of determining which process in the ready state should be moved to the running state is known as Process Scheduling.

The prime aim of the process scheduling system is to keep the CPU busy all the time and to deliver minimum response time for all programs. For achieving this, the scheduler must apply appropriate rules for swapping processes IN and OUT of CPU.

Schedulers fell into one of the two general categories

- **Non-pre-emptive scheduling.** When the currently executing process gives up the CPU voluntarily.
- **Pre-emptive scheduling.** When the operating system decides to favor another process, pre-empting the currently executing process.

Non-pre-emptive scheduling

(1) First Come First Served (FCFS)

- First-Come-First-Served algorithm is the simplest scheduling algorithm is the simplest scheduling algorithm.
- Processes are dispatched according to their arrival time on the ready queue.
- Being a Non preemptive discipline, once a process has a CPU, it runs to completion.
- The FCFS scheduling is fair in the formal sense or human sense of fairness but it is unfair in the sense that long jobs make short jobs wait and unimportant jobs make important jobs wait.
- One of the major drawbacks of this scheme is that the average time is often quite long.

Process	P1	P2	P3	P4	P5	Total
Process Time	5	6	7	2	3	23
Waiting Time	0	5	11	18	20	54
Turn Around Time	5	11	18	20	23	77

Average waiting time = 54/5 = 10.8ms
Average Turnaround time = 77/5 = 15.4ms

(2) Shortest Job First (SJF) / Shortest Process Next (SPN)

- Shortest Job First other name of this algorithm is **Shortest Process Next.**
- Shortest-Job-First is a non-preemptive discipline in which waiting job (or process) with the smallest estimated time runs next.
- The SJF scheduling is especially appropriate for batch jobs for which the run times are known in advance.
- As SJF scheduling algorithm gives the minimum average time for a given set of processes, it is probably optimal.
- The SJF algorithm favors short jobs (or processors) at the expense of longer ones.
- The obvious problem with SJF scheme is that it requires precise knowledge of how long a job or process will run, and this information is not usually available.
- The best SJF algorithm can do is to rely on user estimates of run times.
- Like FCFS, SJF is non preemptive therefore, it is not useful in timesharing environment in which reasonable response time must be guaranteed.

Process	**P1**	**P2**	**P3**	**P4**	**P5**	**Total**
Process Time	5	6	7	2	3	23
Waiting Time	5	10	16	0	2	35
Turn Around Time	10	16	23	2	5	56
Process Sequence	**P4**	**P5**	**P1**	**P2**	**P3**	

Average waiting time = 35/5 = 7ms
Average Turnaround time = 56/5 = 11.2ms

Pre-emptive scheduling

(1) Shortest Remaining Time Next

Process	**P1**	**P2**	**P3**	**P4**	**P5**	**Total**
Process Time (ms)	9	4	5	7	3	28
Arrival Time (ms)	0	1	2	3	4	

- This is a preemptive version of the previous one, in which the scheduler always dispatches that ready process which has the shortest expected remaining time to completion.
- The dispatching decision is always made when a new process is submitted: the currently running process has obviously a smaller remaining time than all other ready ones; otherwise, it wouldn't have been dispatched.
- But a newly submitted job may have an even shorter one, in which case the currently running one would be blocked and put in ready state. This decreases the average turnaround time with respect to shortest process next (SPN).
- Initially at 0ms arrival time, we have only one process p1. We execute this process for 1ms, now remaining time for p1=8ms. At 1ms we have two processes p1 and p2. The burst time for p1=8ms and p2=4. Minimum is 4ms. So p2 will be executed for 1ms as the next process arrives after 1ms

ms	**p1**	**p2**	**p3**	**p4**	**p5**
0	9	**x**	**x**	**x**	**x**
1	8	4	**x**	**x**	**x**
2	8	3	5	x	**x**
3	8	2	5	7	**x**
4	8	1	5	7	3
5	8	0	5	7	3
8	8	x	5	7	0
13	8	**x**	**x**	7	**x**
20	8	**x**	**x**	0	**x**
28	0	**x**	**x**	**x**	**x**

- Now remaining time for p2=3ms. At 2ms we have 3 processes p1=8ms, p2=3ms and p3=5ms. Minimum=3ms, p2 will be executed for 1ms.

- At 3ms, there will be four processes p1=8ms, p2=3ms and p3=5ms. Minimum is 3ms, so p2 will be executed for 1ms
- At 4ms, p2=1 which is minimum, it will be executed and it will be finished. At 5ms there are 4 process p1=8ms, p3=5ms, p4=7ms and p5=3ms.

Minimum is p5=3ms. It will be executed for next 3ms. At 8ms p3=5ms is minimum it will be executed for next 5ms. At 13ms p4=7ms is minimum so it will be executed for next 7ms. At 20ms, p1 process will be executed for next 8ms.Waiting time = Total waiting time – Number of milliseconds the process has already executed – arrival time

	Total waiting time	Already executed	Arrival time	Waiting time
P1	20	1	0	19ms
P2	4	3	1	0ms
P3	8	0	2	6ms
P4	13	0	3	10ms
P5	5	0	4	1ms
Total waiting time = 36ms Average waiting time = 36/5 = 7.2ms Turnaround time = Total Turnaround time – arrival time				

	Total turnaround time	Arrival time	Waiting time	Total turnaround time = 64ms Average turnaround time = 64/5 = 12.8ms
P1	28	0	28ms	
P2	5	1	4ms	
P3	13	2	11ms	
P4	20	3	17ms	
P5	8	4	4ms	

(2) Round Robin

- Round-robin (RR) is one of the simplest scheduling algorithms for processes in an operating system.
- As the term is generally used, time slices are assigned to each process in equal portions and in circular order, handling all processes without priority (also known as cyclic executive).
- Round-robin scheduling is simple, easy to implement, and starvation-free. Round-robin scheduling can also be applied to other scheduling problems, such as data packet scheduling in computer networks.
- The name of the algorithm comes from the round-robin principle known from other fields, where each person takes an equal share of something in turn.

Process	P1	P2	P3	P4	P5	Total
Process Time (ms)	8	6	2	7	9	32

- Suppose Time quantum (Time interval) is of 4ms. So each will be executed for 4ms in FCFS order.

ms	p1	p2	p3	p4	p5
0	8	6	2	7	9
4	4	6	2	7	9
8	4	2	2	7	9
10	4	2	**0**	7	9
14	4	2	**x**	3	9
18	4	2	**x**	3	5
22	**0**	2	**x**	3	5
24	**x**	**0**	**x**	3	5
27	**x**	**x**	**x**	**0**	5
31	**x**	**x**	**x**	**x**	1
32	**x**	**x**	**x**	**x**	**0**

Priority Policy (Preemptive and Non-Preemptive)

- Priority scheduling is a more general case of SJF, in which each job is assigned a priority and the job with the highest priority gets scheduled first. (SJF uses the inverse of the next expected burst time as its priority - The smaller the expected burst, the higher the priority.)
- Note that in practice, priorities are implemented using integers within a fixed range, but there is no agreed-upon convention as to whether "high" priorities use large numbers or small numbers. This book uses low number for high priorities, with 0 being the highest possible priority.
- For example, the following Gantt chart is based upon this process burst times and priorities, and yields an average waiting time of 8.2 ms:

Process	P1	P2	P3	P4	P5	Total
Process Time	10	1	2	1	5	19
Waiting Time	6	0	16	18	1	41
Turn Around Time	16	1	18	19	6	60
Priority	**3**	**1**	**4**	**5**	**2**	
Process Sequence	**P3**	**P1**	**P4**	**P5**	**P2**	

- Priorities can be assigned either internally or externally. Internal priorities are assigned by the OS using criteria such as average burst time, ratio of CPU to I/O activity, system resource use, and other factors available to the kernel. External priorities are assigned by users, based on the importance of the job, fees paid, politics, etc.
- Priority scheduling can be either preemptive or non-preemptive.
- Priority scheduling can suffer from a major problem known as indefinite blocking, or starvation, in which a low-priority task can wait forever because there are always some other jobs around that have higher priority.

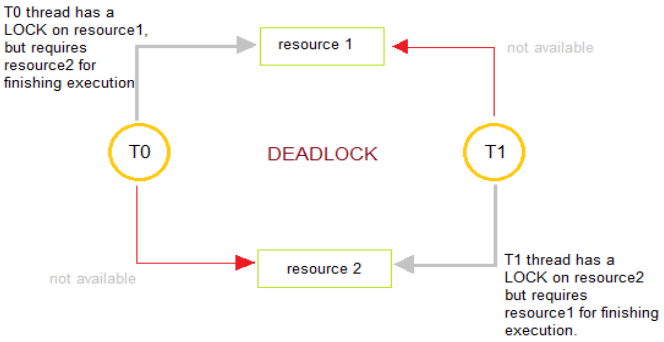
- If this problem is allowed to occur, then processes will either run eventually when the system load lightens (at say 2:00 a.m.), or will eventually get lost when the system is shut down or crashes. (There are rumors of jobs that have been stuck for years.)
- One common solution to this problem is aging, in which priorities of jobs increase the longer they wait. Under this scheme a low-priority job will eventually get its priority raised high enough that it gets run.

UNIT – 2
Dead Locks

Operating systems manage and control all the tasks running on a computer. As you can imagine, conflicts can arise with so many tasks simultaneously. This is where the term "deadlock" comes into play. Deadlocks can be caused by user actions or by software malfunctions. In this post, we'll explore what causes deadlock in the operating system and how to prevent them.

What is Deadlock?

Deadlocks are a set of blocked processes each holding a resource and waiting to acquire a resource held by another process.



In the above figure, process T0 has resource1, it requires resource2 in order to finish its execution. Similarly, process T1 has resource2 and it also needs to acquire resource1 to finish its execution. In this way, T0 and T1 are in a deadlock because each of them needs the resource of others to complete their execution but neither of them is willing to give up their resources.

In General, a process must request a resource before using it and it must release the resource after using it. And any process can request as many resources as it requires in order to complete its designated task. And there is a condition that the number of resources requested may not exceed the total number of resources available in the system.

Basically, in the Normal mode of Operation utilization of resources by a process is in the following sequence:

Request: Firstly, the process requests the resource. In a case, if the request cannot be granted immediately (e.g.: resource is being used by any other process), then the requesting process must wait until it can acquire the resource.

Use: The Process can operate on the resource (e.g.: if the resource is a printer, then in that case process can print on the printer).

Release: The Process releases the resource.

When two or more processes try to access the critical section at the same time and they fail to access simultaneously or stuck while accessing the critical section then this condition is known as Deadlock.

Every process needs a few resources to finish running. The procedure makes a resource request. If the resource is available, the OS will grant it; otherwise, the process will wait. When the process is finished, it is released.

The deadlock has the following characteristics:

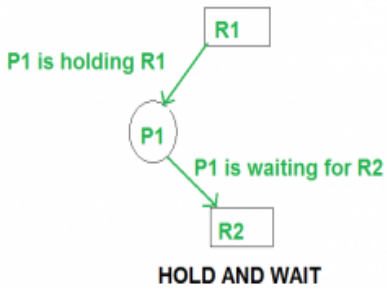
- Mutual Exclusion
- Hold and Wait
- No pre-emption
- Circular wait

Deadlock Detection

Deadlock detection in OS is a critical concept in computer science and operating systems that deals with the potential problem of deadlocks in concurrent systems. A deadlock occurs when two or more processes are unable to proceed with their execution because each process is waiting for a resource that is held by another process within the same system. This situation results in a standstill where no process can make progress, effectively halting the system's functionality.

We can prevent a Deadlock by eliminating any of the above four conditions.

- **Eliminate Mutual Exclusion:** It is not possible to dis-satisfy the mutual exclusion because some resources, such as the tape drive and printer, are inherently non-shareable.
- **Eliminate Hold and wait:** Allocate all required resources to the process before the start of its execution, this way holds and wait condition is eliminated but it will lead to low device utilization. for example, if a process requires a printer at a later time and we have allocated a printer before the start of its execution printer will remain blocked till it has completed its execution. The process will make a new request for resources after releasing the current set of resources. This solution may lead to starvation.
- **Eliminate No Pre-emption**: Preempt resources from the process when resources are required by other high-priority processes.
- **Eliminate Circular Wait**: Each resource will be assigned a numerical number. A process can request the resources to increase/decrease. order of numbering. For Example, if the P1 process is allocated R5 resources, now next time if P1 asks for R4, R3 lesser than R5 such a request will not be granted, only a request for resources more than R5 will be granted.



- **Detection and Recovery**: Another approach to dealing with deadlocks is to detect and recover from them when they occur. This can involve killing one or more of the processes involved in the deadlock or releasing some of the resources they hold.

Deadlock Prevention

Deadlock avoidance uses dynamic information, such as the current state of the system and the resource allocation, to ensure that a deadlock never occurs. Deadlock prevention uses static information, such as the maximum number of resources a process can request, to prevent a deadlock from occurring. Resource allocation algorithms such as the Banker's Algorithm can be used for deadlock prevention.

Deadlock prevention is a technique used in operating systems to avoid the situation where two or more processes are unable to proceed because each is waiting for one of the others to release a resource. This can be achieved by imposing a set of constraints on the way resources are requested and released by the processes.

Methods of deadlock prevention include:

- **Resource allocation:** By allocating resources in a way that ensures that a process can never enter a deadlock state, such as using the Banker's Algorithm, which checks for safe states before allocating resources.

- **Limit on resources:** By limiting the number of resources a process can request, a deadlock state can be prevented from occurring.
- **Timeout:** By setting a timeout for resource requests, a process will release a resource if it is unable to acquire it within a certain time period.
- **Priority-based resource allocation:** By allocating resources based on the priority of the process, a deadlock state can be prevented from occurring.
- **Pre-emptive resource allocation:** By pre-empting resources from a process that has been holding them for a long time, a deadlock state can be prevented from occurring.

These are some of the common methods used in operating systems to prevent deadlocks. The best method to use depends on the specific requirements of the system and the resources being used.

Deadlock avoidance

Deadlock avoidance is a technique used in operating systems to prevent the situation where two or more processes are unable to proceed because each is waiting for one of the others to release a resource. Unlike deadlock prevention, which uses static information to prevent a deadlock from occurring, deadlock avoidance uses dynamic information, such as the current state of the system and the resource allocation, to ensure that a deadlock never occurs. Some common methods of deadlock avoidance include:

- **Wait/Die:** In this method, a process that requests a resource that is not available will wait until the resource becomes available. If a process that holds the resource is also requesting a resource that is not available, the process that has been waiting the longest will be allowed to proceed.
- **Wound/Wait:** In this method, a process that requests a resource that is not available will be killed (wounded) if a process that holds the resource is also requesting a resource that is not available. The process that was killed will have to request the resource again later.
- **Resource allocation graph:** By representing the resources and the processes as nodes in a graph, the operating system can use algorithms to check for safe states before allocating resources.
- **Banker's Algorithm:** This algorithm is a variant of Resource Allocation Graph algorithm, it uses the available and maximum resource information for each process, the algorithm checks for safe state before allocating resources.
- **Time-stamp ordering:** In this method, a process is only allowed to request a resource if its time stamp is greater than the time stamp of the process that holds the resource.

These are some of the common methods used in operating systems for deadlock avoidance. The best method to use depends on the specific requirements of the system and the resources being used.

UNIT – 2
Memory Management

- An operating system manages the memory of computer so that it can be efficiently used.
- The task of operating system for the memory management are as follows
  - Protecting program including itself from any accidental erase of memory.
  - Handling programs that exceed the physical limitation of main memory.
  - Protecting user and programs from one another that are sharing of the single main memory among multiple user and multiple programs.
  - Providing memory protection, preventing programming interference with one another.
  - Users will be prevented from interference with the work of other users.
- In any computer systems normally, we have four kinds of memory available.
- From available memories three are physical memory and one is virtual memory

Physical Memory

Physical memory is the only memory that is directly accessible to the CPU. CPU reads the instructions stored in the physical memory and executes them continuously. The data that is operated will also be stored in physical memory in uniform manner.

CACHE MEMEORY

Cache memory, also called CPU memory, is random access memory (RAM) that a computer microprocessor can access more quickly than it can access regular RAM. This memory is typically integrated directly with the CPU chip or placed on a separate chip that has a separate bus interconnect with the CPU.

The basic purpose of cache memory is to store program instructions that are frequently re-referenced by software during operation. Fast access to these instructions increases the overall speed of the software program.

As the microprocessor processes data, it looks first in the cache memory; if it finds the instructions there (from a previous reading of data), it does not have to do a more time-consuming reading of data from larger memory or other data storage devices.

Most programs use very few resources once they have been opened and operated for a time, mainly because frequently re-referenced instructions tend to be cached. This explains why measurements of system performance in computers with slower processors but larger caches tend to be faster than measurements of system performance in computers with faster processors but more limited cache space.

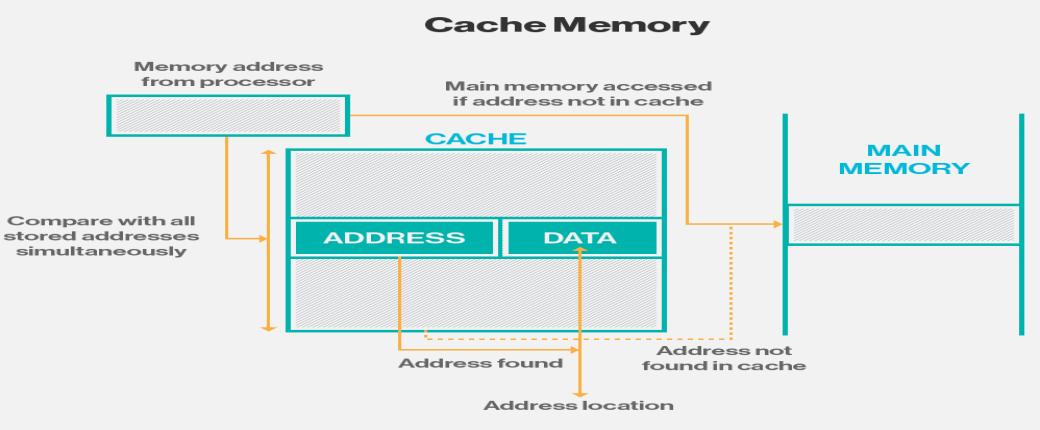
Multi-tier or multilevel caching has become popular in server and desktop architectures, with different levels providing greater efficiency through managed tiering. Simply put, the less frequently access is made to certain data or instructions, the lower down the cache level the data or instructions are written.

Cache memory levels

Cache memory is fast and expensive. Traditionally, it is categorized as "levels" that describe its closeness and accessibility to the microprocessor:

Level 1 (L1) cache is extremely fast but relatively small, and is usually embedded in the processor chip (CPU).
Level 2 (L2) cache is often more capacious than L1; it may be located on the CPU or on a separate chip or coprocessor with a high-speed alternative system bus interconnecting the cache to the CPU, so as not to be slowed by traffic on the main system bus.

Level 3 (L3) cache is typically specialized memory that works to improve the performance of L1 and L2. It can be significantly slower than L1 or L2, but is usually double the speed of RAM. In the case of multi core processors, each core may have its own dedicated L1 and L2 cache, but share a common L3 cache. When an instruction is referenced in the L3 cache, it is typically elevated to a higher tier cache



Buffer Cache

- The buffer cache contains data buffers that are used by the block device drivers.
- These buffers are of fixed sizes (for example 512 bytes) and contain blocks of information that have either been read from a block device or are being written to it. A block device is one that can only be accessed by reading and writing fixed sized blocks of data. All hard disks are block devices.
- The buffer cache is indexed via the device identifier and the desired block number and is used to quickly find a block of data. Block devices are only ever accessed via the buffer cache. If data can be found in the buffer cache, then it does not need to be read from the physical block device, for example a hard disk, and access to it is much faster.

Page Cache

- This is used to speed up access to images and data on disk.
- It is used to cache the logical contents of a file a page at a time and is accessed via the file and offset within the file. As pages are read into memory from disk, they are cached in the page cache.

Swap Cache

- Only modified (or dirty) pages are saved in the swap file.
- So long as these pages are not modified after they have been written to the swap file then the next time the page is swapped out there is no need to write it to the swap file as the page is already in the swap file. Instead, the page can simply be discarded. In a heavily swapping system this saves many unnecessary and costly disk operations.

Hardware Caches

- One commonly implemented hardware cache is in the processor; a cache of Page Table Entries. In this case, the processor does not always read the page table directly but instead caches translations for pages as it needs them. These are the Translation Look-aside Buffers and contain cached copies of the page table entries from one or more processes in the system.
- When the reference to the virtual address is made, the processor will attempt to find a matching TLB entry. If it finds one, it can directly translate the virtual address into a physical one and perform the correct operation on the data. If the processor cannot find a matching TLB entry then it must get the operating system to help. It does this by signaling the operating system that a TLB miss has occurred. A system specific mechanism is used to deliver that exception to the operating system code that can fix things up. The operating system generates a new TLB entry for the address mapping. When the exception has been cleared, the processor will make another attempt to translate the virtual address. This time it will work because there is now a valid entry in the TLB for that address.

The drawback of using caches, hardware or otherwise, is that in order to save effort Linux must use more time and space maintaining these caches and, if the caches become corrupted, the system will crash

Primary Memory

- It is the main memory of computer and also knows as RAM. It a volatile memory.
- OS's data and our important data will be stored in this memory.
- We can also say that waiting queue data will be stored in primary memory and ready queue data will be stored in cache memory.
- The speed of primary memory is less compared to cache memory.

Secondary Memory

- Secondary memory, also known as secondary storage, is the slower and cheaper form of memory.
- CPU does not access the secondary memory directly. The content in it must first be copied into the primary storage RAM for CPU to process.
- Secondary memory devices include hard drives, floppy disks, CDs, Pen drive, etc.

Fragmentation

Fragmentation is an unwanted problem in the operating system in which the processes are loaded and unloaded from memory, and free memory space is fragmented. Processes can't be assigned to memory blocks due to their small size, and the memory blocks stay unused.

Contiguous memory allocation allocates space to processes whenever the processes enter **RAM**. These **RAM** spaces are divided either by fixed partitioning or by dynamic partitioning. As the process is loaded and unloaded from memory, these areas are fragmented into small pieces of memory that cannot be allocated to coming processes.

What is Fragmentation?

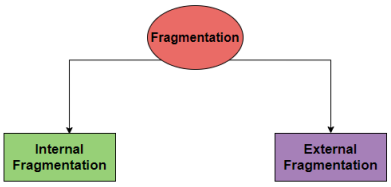
Fragmentation is an unwanted problem in the operating system in which the processes are loaded and unloaded from memory, and free memory space is fragmented. Processes can't be assigned to memory blocks due to their small size, and the memory blocks stay unused. It is also necessary to understand that as programs are loaded and deleted from memory, they generate free space or a hole in the memory. These small blocks cannot be allotted to new arriving processes, resulting in inefficient memory use.

The conditions of fragmentation depend on the memory allocation system. As the process is loaded and unloaded from memory, these areas are fragmented into small pieces of memory that cannot be allocated to incoming processes. It is called fragmentation.

Causes

User processes are loaded and unloaded from the main memory, and processes are kept in memory blocks in the main memory. Many spaces remain after process loading and swapping that another process cannot load due to their size. Main memory is available, but its space is insufficient to load another process because of the dynamical allocation of main memory processes.

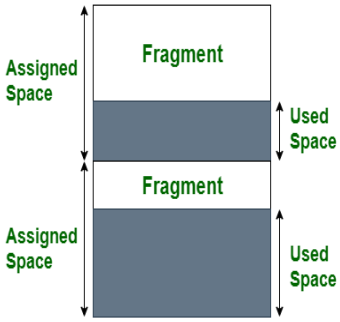
Types of Fragmentation



There are mainly two types of fragmentation in the operating system. These are as follows:

1. Internal Fragmentation:

Internal fragmentation happens when the memory is split into mounted-sized blocks. Whenever a method is requested for the memory, the mounted-sized block is allotted to the method. In the case where the memory allotted to the method is somewhat larger than the memory requested, then the difference between allotted and requested memory is called internal fragmentation. We fixed the sizes of the memory blocks, which has caused this issue. If we use dynamic partitioning to allot space to the process, this issue can be solved.

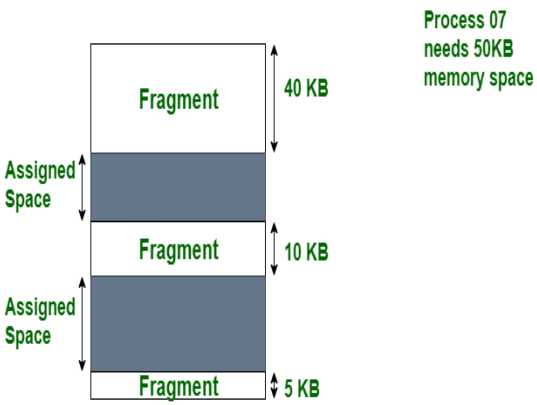


Internal Fragmentation

The above diagram clearly shows the internal fragmentation because the difference between memory allocated and required space or memory is called Internal fragmentation.

2. External Fragmentation:

External fragmentation happens when there's a sufficient quantity of area within the memory to satisfy the memory request of a method. However, the process's memory request cannot be fulfilled because the memory offered is in a non-contiguous manner. Whether you apply a first-fit or best-fit memory allocation strategy it'll cause external fragmentation.



In the above diagram, we can see that, there is enough space (55 KB) to run a process-07 (required 50 KB) but the memory (fragment) is not contiguous. Here, we use compaction, paging, or segmentation to use the free space to run a process.

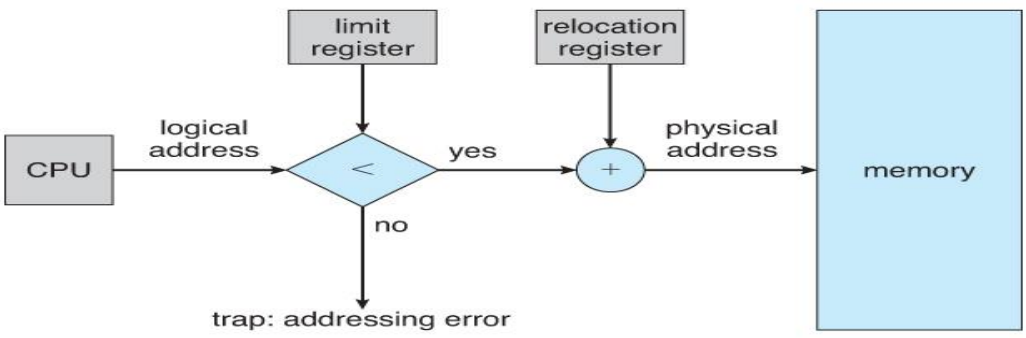
NO	Internal fragmentation	External fragmentation
1.	In internal fragmentation fixed-sized memory, blocks square measure appointed to process.	In external fragmentation, variable-sized memory blocks square measure appointed to the method.
2.	Internal fragmentation happens when the method or process is smaller than the memory.	External fragmentation happens when the method or process is removed.
3.	The solution of internal fragmentation is the best-fit block.	The solution to external fragmentation is compaction and paging.
4.	Internal fragmentation occurs when memory is divided into fixed-sized partitions.	External fragmentation occurs when memory is divided into variable size partitions based on the size of processes.
5.	The difference between memory allocated and required space or memory is called Internal fragmentation.	The unused spaces formed between non-contiguous memory fragments are too small to serve a new process, which is called External fragmentation.
6.	Internal fragmentation occurs with paging and fixed partitioning.	External fragmentation occurs with segmentation and dynamic partitioning.
7.	It occurs on the allocation of a process to a partition greater than the process's requirement. The leftover space causes degradation system performance.	It occurs on the allocation of a process to a partition greater which is exactly the same memory space as it is required.
8.	It occurs in worst fit memory allocation method.	It occurs in best fit and first fit memory allocation method.

Type of Memory Allocation

Contiguous memory allocation

- Contiguous memory allocation is one of the oldest memory allocation schemes.
- When a process needs to execute, memory is requested by the process. The size of the process is compared with the amount of contiguous main memory available to execute the process.
- If sufficient contiguous memory is found, the process is allocated memory to start its execution. Otherwise,
- It is added to a queue of waiting processes until sufficient free contiguous memory is available.

Memory Protection (Memory Mapping and Protection)



The system shown in below Figure allows protection against user programs accessing areas that they should not, allows programs to be relocated to different memory starting addresses as needed, and allows the memory space devoted to the OS to grow or shrink dynamically as needs change.

Memory Allocation

One method of allocating contiguous memory is to divide all available memory into equal sized partitions, and to assign each process to their own partition. This restricts both the number of simultaneous processes and the maximum size of each process, and is no longer used.

An alternate approach is to keep a list of **unused (free) memory blocks (holes)**, and to find a hole of a suitable size whenever a process needs to be loaded into memory. There are many different strategies for finding the "best" allocation of memory to processes, including the three most commonly discussed:

- **First fit** - Search the list of holes until one is found that is big enough to satisfy the request, and assign a portion of that hole to that process. Whatever fraction of the hole not needed by the request is left on the free list as a smaller hole. Subsequent requests may start looking either from the beginning of the list or from the point at which this search ended.

  Advantage : - Fastest algorithm because it searches as little as possible.

  Disadvantage: The remaining unused memory areas left after allocation become waste if it is too smaller. Thus, request for larger memory requirement cannot be accomplished.

- **Best fit** - Allocate the smallest hole that is big enough to satisfy the request. This saves large holes for other process requests that may need them later, but the resulting unused portions of holes may be too small to be of any use, and will therefore be wasted. Keeping the free list sorted can speed up the process of finding the right hole.

  Advantage

  Memory utilization is much better than first fit as it searches the smallest free partition first available.

  Disadvantage

  It is slower and may even tend to fill up memory with tiny useless holes.

- **Worst fit** - Allocate the largest hole available, thereby increasing the likelihood that the remaining portion will be usable for satisfying future requests.

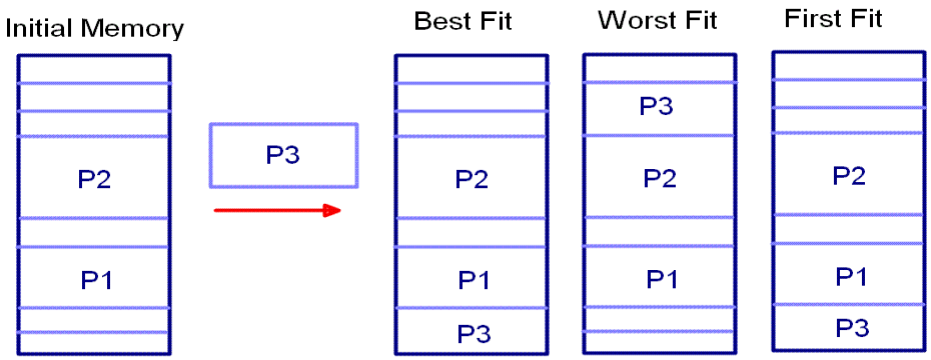
  Simulations show that either first or best fit are better than worst fit in terms of both time and storage utilization. First and best fits are about equal in terms of storage utilization, but first fit is faster.

  Advantage

  Reduces the rate of production of small gaps.

  Disadvantage

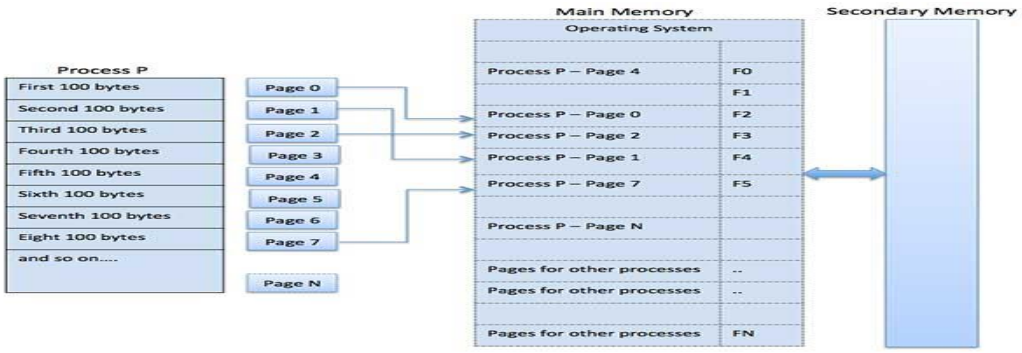
  If a process requiring larger memory arrives at a later stage, then it cannot be accommodated as the largest hole is already split and occupied.



Non-Contiguous memory allocation

- When a process needs to execute, memory is requested by the process.
- The size of the process is compared to the total main memory available to execute the process.
- If sufficient total memory is found (random or in sequence), the process is allocated memory to start its execution. Otherwise
- It is added to a queue of waiting processes until sufficient free memory is available.

Paging



A computer can address more memory than the amount physically installed on the system. This extra memory is actually called virtual memory and it is a section of a hard that's set up to emulate the computer's RAM. Paging technique plays an important role in implementing virtual memory.

Paging is a memory management technique in which process address space is broken into blocks of the same size called **pages** (size is power of 2, between 512 bytes and 8192 bytes). The size of the process is measured in the number of pages.

Similarly, main memory is divided into small fixed-sized blocks of (physical) memory called **frames** and the size of a frame is kept the same as that of a page to have Optimum utilization of the main memory and to avoid external fragmentation.

Address Translation

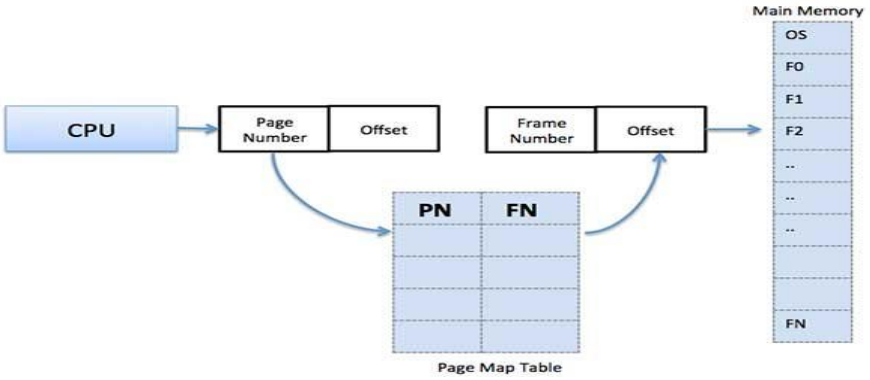
Page address is called logical address and represented by page number and the offset.

Logical Address = Page number + page offset

Frame address is called physical address and represented by a frame number and the offset.

Physical Address = Frame number + page offset

A page map table is used to keep track of the relation between a page of a process to a frame in physical memory.



When the system allocates a frame to any page, it translates this logical address into a physical address and creates entry into the page table to be used throughout execution of the program.

When a process is to be executed, its corresponding pages are loaded into any available memory frames. Suppose you have a program of 8Kb but your memory can accommodate only 5Kb at a given point in time, then the paging concept will come into picture. When a computer runs out of RAM, the operating system (OS) will move idle or unwanted pages of memory to secondary memory to free up RAM for other processes and brings them back when needed by the program.

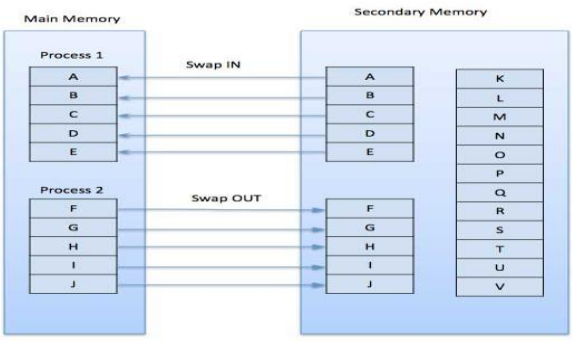
This process continues during the whole execution of the program where the OS keeps removing idle pages from the main memory and write them onto the secondary memory and bring them back when required by the program.

Advantages and Disadvantages of Paging

- Paging reduces external fragmentation, but still suffers from internal fragmentation.
- Paging is simple to implement and assumed as an efficient memory management technique.
- Due to equal size of the pages and frames, swapping becomes very easy.
- Page table requires extra memory space, so may not be good for a system having small RAM.

Demand Paging

A demand paging system is quite similar to a paging system with swapping where processes reside in secondary memory and pages are loaded only on demand, not in advance. When a context switch occurs, the operating system does not copy any of the old program's pages out to the disk or any of the new program's pages into the main memory Instead, it just begins executing the new program after loading the first page and fetches that program's pages as they are referenced.



While executing a program, if the program references a page which is not available in the main memory because it was swapped out a little ago, the processor treats this invalid memory reference as a page fault and transfers control from the program to the operating system to demand the page back into the memory.

Advantages

- Large virtual memory.
- More efficient use of memory.
- There is no limit on degree of multiprogramming.

Disadvantages

- Number of tables and the amount of processor overhead for handling page interrupts are greater than in the case of the simple paged management techniques.

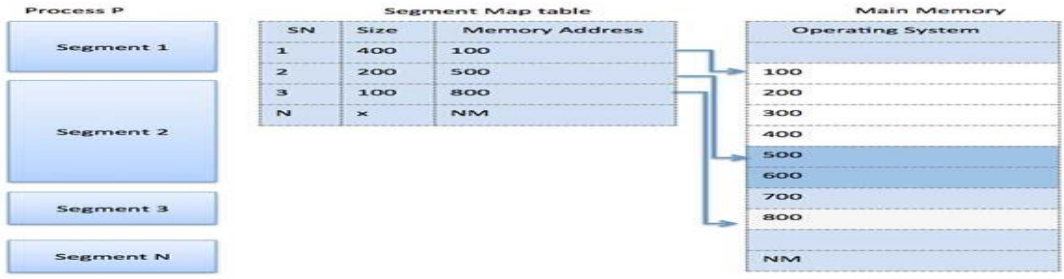
Segmentation

Segmentation is a memory management technique in which each job is divided into several segments of different sizes, one for each module that contains pieces that perform related functions. Each segment is actually a different logical address space of the program.

When a process is to be executed, its corresponding segmentation is loaded into non-contiguous memory though every segment is loaded into a contiguous block of available memory.

Segmentation memory management works very similar to paging but here segments are of variable-length where as in paging pages are of fixed size.

A program segment contains the program's main function, utility functions, data structures, and so on. The operating system maintains a segment map table for every process and a list of free memory blocks along with segment numbers, their size and corresponding memory locations in main memory. For each segment, the table stores the starting address of the segment and the length of the segment. A reference to a memory location includes a value that identifies a segment and an offset.



Segmentation without paging

Associated with each segment is information that indicates where the segment is located in memory— the segment base. When a program references a memory location the offset is added to the segment base to generate a physical memory address.

An implementation of virtual memory on a system using segmentation without paging requires that entire segments be swapped back and forth between main memory and secondary storage. When a segment is swapped in, the operating system has to allocate enough contiguous free memory to hold the entire segment. Often memory fragmentation results in there being not enough contiguous memory even though there may be enough in total.

Segmentation with paging

Instead of an actual memory location the segment information includes the address of a page table for the segment. When a program references a memory location the offset is translated to a memory address using the page table. A segment can be extended simply by allocating another memory page and adding it to the segment's page table.

An implementation of virtual memory on a system using segmentation with paging usually only moves individual pages back and forth between main memory and secondary storage, similar to a paged non-segmented system. Pages of the segment can be located anywhere in main memory and need not be contiguous. This usually results in a reduced amount of input/output between primary and secondary storage and reduced memory fragmentation

Virtual Memory

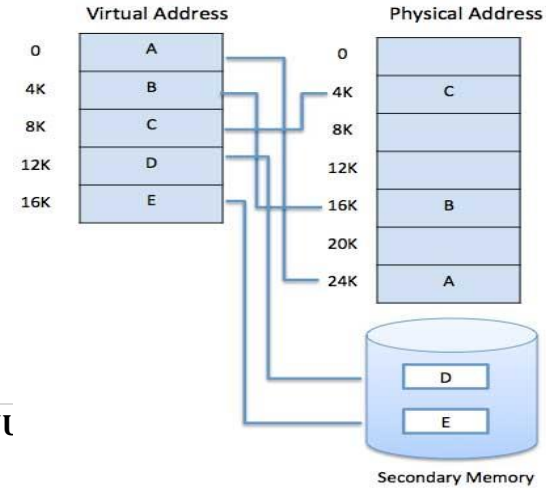
A computer can address more memory than the amount physically installed on the system. This extra memory is actually called **virtual memory** and it is a section of a hard disk that's set up to emulate the computer's RAM.

The main visible advantage of this scheme is that programs can be larger than physical memory. Virtual memory serves two purposes. First, it allows us to extend the use of physical memory by using disk. Second, it allows us to have memory protection, because each virtual address is translated to a physical address.

Following are the situations, when entire program is not required to be loaded fully in main memory.

- User written error handling routines are used only when an error occurred in the data or computation.
- Certain options and features of a program may be used rarely.
- Many tables are assigned a fixed amount of address space even though only a small amount of the table is actually used.
- The ability to execute a program that is only partially in memory would counter many benefits.
- Less number of I/O would be needed to load or swap each user program into memory.
- A program would no longer be constrained by the amount of physical memory that is available.
- Each user program could take less physical memory, more programs could be run the same time, with a corresponding increase in CPU utilization and throughput.

Modern microprocessors intended for general- purpose use, a memory management unit, or MMU, is built into the hardware. The MMU's job is to translate virtual addresses into physical addresses.



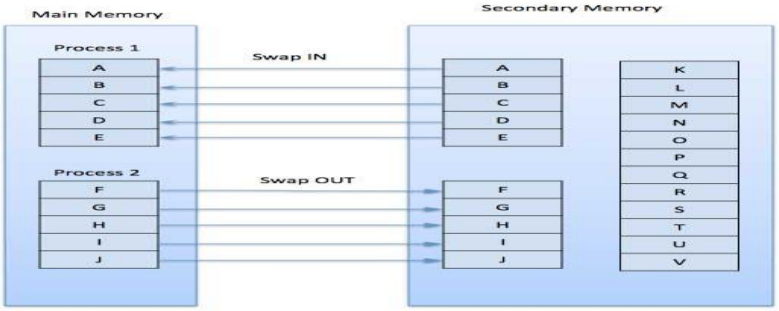
Virtual memory is commonly implemented by demand paging. It can also be implemented in

a segmentation system. Demand segmentation can also be used to provide virtual memory.

Virtual Memory with Paging

A paging system is same as swapping where processes reside in secondary memory and pages are loaded only on demand, not in advance. When a context switch occurs, the operating system does not copy any of the old program's pages out to the disk or any of the new program's pages into the main memory Instead, it just begins executing the new program after loading the first page and fetches that program's pages as they are referenced.

While executing a program, if the program references a page which is not available in the main memory because it was swapped out a little ago, the processor treats this invalid memory reference as a page fault and transfers control from the program to the operating system to demand the page back into the memory.



Advantages

- Large virtual memory.
- More efficient use of memory.
- There is no limit on degree of multiprogramming.

Disadvantages

- Number of tables and the amount of processor overhead for handling page interrupts are greater than in the case of the simple paged management techniques.

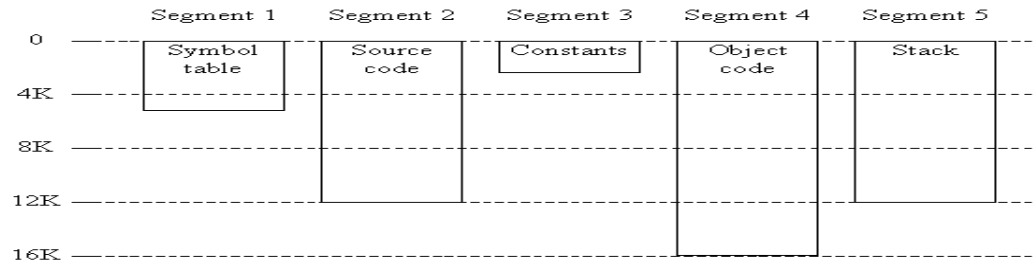
Virtual Memory with Segmentation

Another type of virtual memory is the segmented memory. With this kind of virtual memory, programs are built based on segments, which are defined by a programmer or a compiler. Segments have their own identifiers, determined length and independent address spaces. Segments contain sequences of data or instructions written under consecutive addresses. Segments have determined owners and access rights from the side of other users. In this respect, segments can be "private" for a given user or shared, i.e., available for accessing by other users. Segment parameters can be changed during program execution, to set dynamically the segment length and the rules for mutual access by many users. Segments are placed in a common virtual address space defined by their names and lengths. They can reside in the main memory or in the auxiliary store - usually disk memory. Segments requested by currently executed programs are automatically fetched to the main memory by the segmented memory control mechanism, which is implemented by the operating system.

Segmentation is the way to increase the address space of user programs but also a mechanism for conscious structured program organization with determined access rights and segment protection in a system used by many users.

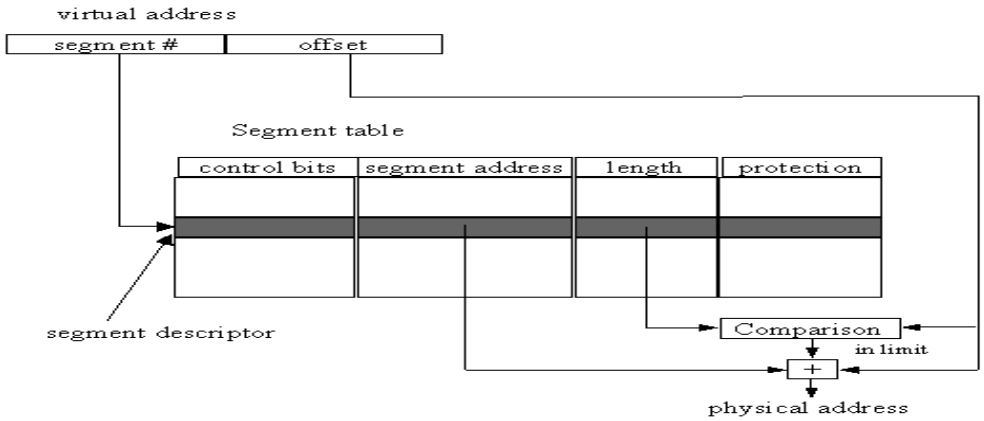
Examples of different segments in a program

A



Virtual address in segmentation is composed of two fields: a segment number and word (byte) displacement (offset) in the segment. Each segment has a descriptor stored in the segment table. In a descriptor, the parameters of a segment are determined such as control bits, segment address, segment length, protection bits. The control bits usually contain: the presence in the main memory bit, a segment type code, allowed access type code (read, write, and execution), size extension control. The protection bits contain the privilege code of the segment in the general data protection system. On each try to access a segment contents, the accessing program privilege level is compared to the privilege level of the segment and the access rules are checked. If the access rules are not fulfilled, access to the segment is blocked and the exception "segment access rules violation" is generated.

Segmented virtual address translation is performed by reading the segment descriptor from the segment table stored in the main memory. The descriptor determines if the segment resides in the main memory. If so, the physical data address is calculated by adding the offset given the virtual address to the base segment address read from the segment descriptor. Before that, the system checks if the given displacement does not overcome the segment length given in the descriptor. If it overcomes the segment length, the "segment length violation" exception is generated and program execution is broken. If the requested segment is out the main memory, the "missing segment" exception is generated. This exception is processed by the operating system which brings the requested segment from the auxiliary memory, updates the segment descriptor and activates the interrupted program. The scheme of virtual address translation with segmentation is shown in the figure below.



Virtual address translation scheme with segmentation

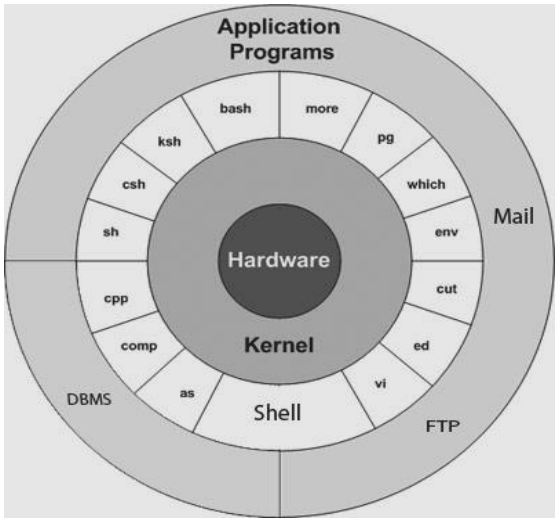
Paging	Segmentation
A page is a contiguous range of memory addresses which is mapped to physical memory.	A segment is an independent address space. Each segment has addresses in a range from 0 to maximum value.
It has only one linear address space.	It has many address spaces.
Programmer does not know that it is implemented	Programmer knows that it is implemented.
Procedures and data cannot be separated	Procedures and data can be separated
Procedures cannot be shared between users	Procedures can be shared between users
Procedures and data cannot be protected separately	Procedures and data can be protected separately
Compilation cannot be done separately	Compilation can be done separately
A page is a physical unit	A segment is a logical unit
A page is of fixed size	A segment is of arbitrary size.

UNIT – 3

Getting Started With UNIX

UNIX Architecture:

The main concept that unites all versions of UNIX is the following four basics:



- **Kernel:** The kernel is the heart of the operating system. It interacts with hardware and most of the tasks like memory management, tash scheduling and file management.
- **Shell:** The shell is the utility that processes your requests. When you type in a command at your terminal, the shell interprets the command and calls the program that you want. The shell uses standard syntax for all commands. C Shell, Bourne Shell and Korn Shell are most famous shells which are available with most of the UNIX variants.
- **Commands and Utilities:** There are various command and utilities which you would use in your day-to-day activities. **cp, mv, cat** and **grep** etc. are few examples of commands and utilities. There are over 250 standard commands plus numerous others provided through 3rd party software. All the commands come along with various optional options.
- **Files and Directories:** All data in UNIX is organized into files. All files are organized into directories. These directories are organized into a tree-like structure called the file system.

UNIX Features

- **Multiuser**
  - The main part of Unix install in the host(server) computer which is known as server
  - The number of terminals connected to the host machine
  - The number of terminal is depended upon number of ports available on the controller part.
  - The terminal can be of different type which are as follow

  Dumb terminal
    - The terminal consist of input device and with output device having no memory or hard disk off is or on
    - For example:-ATM machine

  Online terminal
    - A computer having its own input-output device with a hard disk, processer and ram are known as online terminal
    - Same as the dumb terminal a request will be same to the host machine and will receive output from it
    - Special software is installed on the user computer to make pc work like a dumb terminal is known as terminal evaluation

- Vtarm and xtalk are the two popular software which are used to connect the host machine

Dial up terminal

- This terminal will use a telephone line to connect with the host machine
- Modem is a require on the terminal side as well as host side to make a communication channel

- **Multitasking**
  - It is the main important feature of a Unix operating system. which is capable of carrying more than one task at a time
  - It will allow to user to write down a program while a same another program is begging executed
- **Communication**
  - Unix has expellant provirus of the communication with each & every user
  - A communication may be done between two computers or more than two computers in the network
  - The user can easily share date, program, message etc through network
- **Security**
  - The Unix allow to share data between the user but has three inherited provision
  - The 1st is user name & password for login
  - It provides three basic permission (read, write, executed). Which will define (decide) who can access a particular file, who can modify a file and who can execute the program
  - The 3rd provision is the file encryption which will encode the file in to the unreadable format. So that no one can understand the information even they access it
  - The above three provision will be implemented(to be consider) for each & every file of the Unix system and of user
- **Portability**
  - One of the main reasons of the universal popularity of UNIX system is that it can be ported to the all most any computer system.

Types of Shell

Bourne shell

Bourne shell was developed by Stephen Bourne at AT & T bell labs and first released in 1977 in the version 7 UNIX distributed to college and universities. It was used in earlier versions of UNIX. It is the most primitive shell, its later version called Bash Shell is having extended feature than the Bourne Shell.

The Bourne shell or sh was the default UNIX Shell of UNIX version 7 and replaced the original UNIX Shell. Bourne Shell introduced the feature of using file descriptor 2 for error message and keeping error message separate from data.

Features of Bourne Shell include file name expression using meta – character, command substitution, pipes, positional parameter , built in shell command and shell variables. It works as a command interpreter as well as programming language. Other shells like Korn shell and bash shell have the backward compatibility with the Bourne Shell. All the script which is written in Bourne Shell could be executed in these Shells as well. Bourne shell is known as the UNIVESAL Shell. Currently the Linux system uses the bash as a direct descendent of Bourne Shell.

Smt. J. J. Kundalia Commerce College, Rajkot

(Computer Science Department)

C Shell

C Shell was developed by William Joy at University of California, Berkeley, it can be used to interpret user command and also as a programming language to create shell script.

There are three standard streams supported by C Shell, standard input, standard error, they are connected to the terminal. It supports other features like redirected, command substituted, pipes and job control.

C Shell allows you to store values into variables. These variables are broadly classified as user defined variables and pre defined variables. The pre defined variables include the shell variables and environmental variables

C Shell executes any commands in six sequence steps.

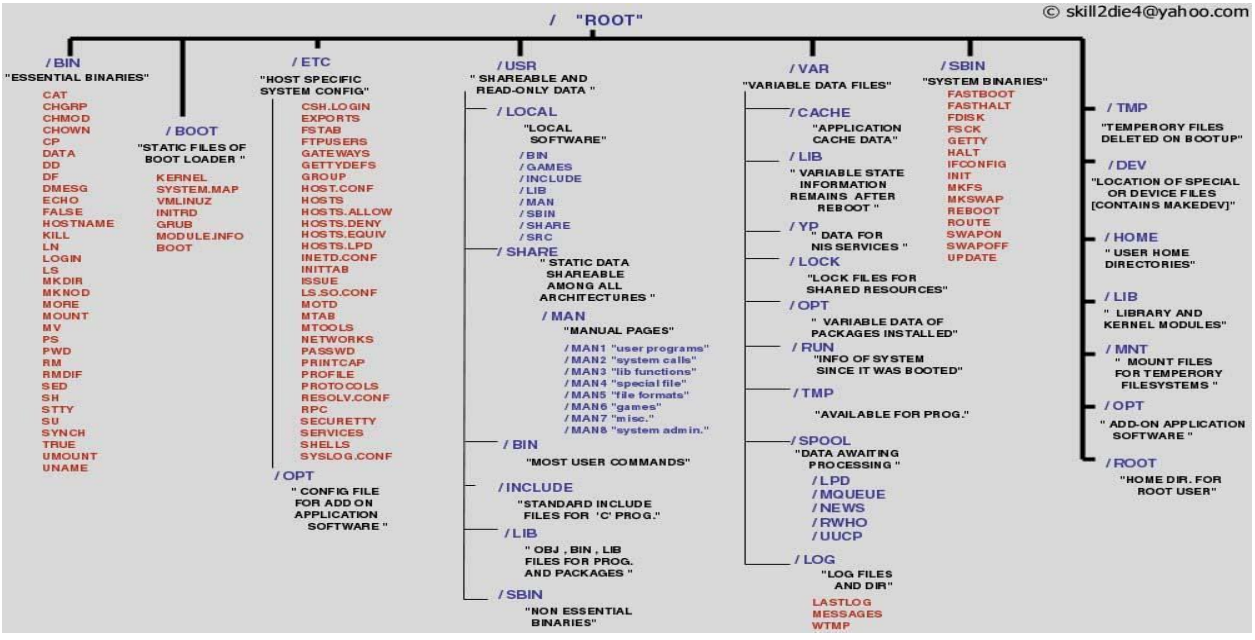
1. It parses the command into word
2. It looks for the variable names starting with $ sign.
3. It looks for the command substituted using back quotes.
4. It checks the command for the redirected files and verifies the files.
5. If wildcard characters are found, shell replaces it with the original file names.
6. In the last step, it uses the PATH variable to locate the directory containing the actual command.

Korn Shell

The Korn Shell was developed by David G. Korn at AT & T Bell Laboratories. The major advantages of ksh over other shell are: it allows users to use vi or emacs- style editing commands on your command lines. It provides the functionally of several UNIX command, including test, expr , echo and patten matching , which are integrated into the shell itself. Utilities like grep and awk have been added to the standard set of the filename wildcards and to the shell variables facility. Advanced I/O features, including in the Korn shell. New optional and variables that give different ways to customize the environment are also included. Efforts are made to increase the speed of shell code executed.

Korn Shell allows the use of function in the script; this function can be placed in the file. It also provides the extensive command history capability consisting of various combinations of command, environmental variables and files. The command can be recalled and re - executed without typing, It again. It also uses the same steps to execute the command as in the C Shell.

File Structure of UNIX



/bin	Contains the executable programs that are part of the Linux operating system. Many Linux commands such as cat, cp, ls, more, and tar are located in /bin. Example ls, cat, cp.
/dev	All the devices like input devices, sound card, modems are stored. It is a virtual directory that contains devices files. Example : /dev/udp, /dev/urandom, /dev/sda1
/etc	Contains config folder of entire operating system. All the global setting like ssh, telnet, and smtp/pop3 mail servers. Also contains system's password file like group lists, user skeletons, and cron jobs. Example: /etc/resolv.conf, /etc/logrotate.conf
/home	Default directory for users to store the personal files. Example /home/saugat, /home/sachit
/sbin	contains binary executables typtically used by system admnistrator only available to root. Mostly used for system maintenance purpose Commands such as mount, shutdown, umount, reside here Example: /sbin/halt/ /sbin/ip6tables
/usr	contains shareable and read only data contains binaries, libraries, documentation and source code for second level program
/usr/bin	Contains executable files for many Linux commands. It is not part of the core Linux operating system.
/usr/include	Contains header files for C and C++ programming languages
/usr/lib	Contains libraries for C and C++ programming languages.
/usr/local	Contains local files. It has a similar directories as /usr contains.
/usr/sbin	Contains administrative commands
/usr/share	Contains files that are shared, like, default configuration files, images, documentation, etc.
/usr/src	Contains the source code for the Linux kernel.
/var	Includes user specific files such as mail message, database of installed programs, log files etc.
/var/cache	Storage area for cached data for applications.
/var/lib	Contains information related to the current state of applications. Programs modify this when they run.
/var/lock	Contains lock files which are checked by applications so that a resource can be used by one application only.
/var/log	Contains log files for different applications.
/var/mail	Contains users emails
/var/opt	Contains variable data for packages stored in /opt directory.
/var/run	Contains data describing the system since it was booted.
/var/spool	Contains data that is waiting for some kind of processing.
/var/tmp	Contains temporary files preserved between system reboots
/tmp	All the temporary files are store here. The files under this directory are deleted when system is rebooted. For example: when new program is installed it use /tmp/ to put files during installation that won't be needed after the program is installed.
/mnt	Default location for mouting devices like cdrooms, floppy disk dries, USB

	memory sticks etc. Example : /mnt/cdroom
/proc	contains information about system process virtual file system that contains information about file system. Example /proc/cpuinfo, /proc/swaps
/lib	share libraries are stored(perl, python, C, etc.) /lib/ are also a kernel modules Example: ld-2.11.1.so, libncurses.so.5.7
/opt	Config file for add on Application software are found here. Third party application should be installed in this directory.
/root	Home directory of system administrator.'root'. Root user has write privilege under this directory
/boot	Contains everything required for boot process. Stores data that is used before the kernel begins executing user-mode program. Example: /boot/boot.b, /boot/chain.b, /boot/config-kernel-version

Types of Files

By default UNIX have only 3 types of files. They are

1. Regular files
2. Directory files
3. Special Files (This category is having 5 sub types in it.)

So in practical we have total 7 types (1+1+5) of files in Linux/Unix. And in Solaris we have 8 types. And you can see the file type indication at leftmost part of "ls -l" command. Here are those files type.

1. Regular File (-)
2. Directory Files(d)
3. Special or Device Files
   a) Block file (b)
   b) Character device file (c)
   c) Named pipe file or just a pipe file (p)
   d) Symbolic link file (l)
   e) Socket file (s)

For your information there is one more file type called door File (D) which is present in Sun Solaris as mention earlier. A door is a special file for inter-process communication between a client and server (so total 8 types in UNIX machines). We will learn about different types of files as below sequence for every file type.

Ordinary or Regular Files

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

Directories

A directory is a binary file used to track and locate other files and directories. The binary format is used so that directories containing large numbers of filenames can be search quickly.

Device (Special)/ Block Files

Device or special files are used for device I/O on UNIX and Linux systems. They appear in a file system just like an ordinary file or a directory.

On UNIX systems there are two flavors of special files for each device, character special files and block special files. Linux systems only provide one special file for each device.

When a character special file is used for device I/O, data is transferred one character at a time. This type of access is called raw device access.

When a block special file is used for device I/O, data is transferred in large fixed-size blocks. This type of access is called block device access.

Links

A link is a tool used for having multiple filenames that reference a single file on a physical disk. They appear in a file system just like an ordinary file or a directory.

Like special files, links also come in two different flavors. There are hard links and symbolic links.

Hard links do not actually link to the original file. Instead they maintain their own copy of the original file's attributes (i.e. location on disk, file access permissions, etc.). If the original file is deleted, its data can still be accessed using the hard link.

On the other hand, symbolic links contain a pointer, or pathname, to the original file. If the original file is deleted, its data can no longer be accessed using the symbolic link, and the link is then considered to be a stale link.

Named Pipes

Named pipes are tools that allow two or more system processes to communicate with each other using a file that acts as a pipe between them. This type of communication is known as inter process communication or IPC for short.

Sockets

Sockets are also tools used for inter process communication. The difference between sockets and pipes is that sockets will facilitate communication between processes running on different systems, or over the network.

With so many different types of files, it's often wise to identify a file's type before performing any operation with it. The **ls -l** command and the **file** command are useful for determining file types.

Consider the long listing of the **livefirelabs1** file:

-rw-rw-r-- 1 student1 student1 0 Jun 27 18:55 livefirelabs1

The first character of the first field indicates the file type. In this example, the first character is a - (hyphen) indicating that **livefirelabs1** is an ordinary or regular file. Consider the long listing of the **live1** file:

rwxrwxrwx 1 student1 student1 13 Jun 27 17:57 live1 -> livefirelabs1

The first character of the first field is the letter **l** indicating **live1** is a symbolic link. The following is a table listing what characters represent what types of files:

Ordinary or Regular File

d - Directory
c - Character special file
b - Block special file
l - Symbolic link
p - Named pipe
s - Socket

The **file** command is also helpful for determining file types. The syntax for this command is: $ **file** filename.

File Type	First Character in File Listing	Description
Regular file	**-**	Normal files such as text, data, or executable files
Directory	**d**	Files that are lists of other files
Link	**l**	A shortcut that points to the location of the actual file
Special file	**c**	Mechanism used for input and output, such as files in /dev
Socket	**s**	A special file that provides inter-process networking protected by the file system's access control
Pipe	**P**	A special file that allows processes to communicate with each other without using network socket semantics

File and Directory Permissions

there are already a lot of good security features built into Linux-based systems, one very important potential vulnerability can exist when local access is granted - - that is file permission-based issues resulting from a user not assigning the correct permissions to files and directories. So based upon the need for proper permissions, I will go over the ways to assign permissions and show you some examples where modification may be necessary.

Basic File Permissions

Permission Groups

Each file and directory has three user based permission groups:

Type	Description
Owner	The Owner permissions apply only the owner of the file or directory; they will not impact the actions of other users.
Group	The Group permissions apply only to the group that has been assigned to the file or directory; they will not affect the actions of other users.
all users	The All Users permissions apply to all other users on the system; this is the permission group that you want to watch the most.

Permission Types

Type	Description
Read	The Read permission refers to a user's capability to read the contents of the file.
Write	The Write permissions refer to a user's capability to write or modify a file or directory.
Execute	The Execute permission affects a user's capability to execute a file or view the contents of a directory.

Viewing the Permissions

You can view the permissions by checking the file or directory permissions in your favorite GUI File Manager (which I will not cover here) or by reviewing the output of the \"ls -l\" command while in the terminal and while working in the directory which contains the file or folder.

1. The '-' file type indicates that this is a regular file
2. User rights/Permissions
    A. The first character that I marked with an underscore is the special permission flag that can vary.
    B. The following set of three characters (rwx) is for the owner permissions.
    C. The second set of three characters (rwx) is for the Group permissions.
    D. The third set of three characters (rwx) is for the All Users permissions.
3. Following that grouping since the integer/number displays the number of hard links to the file.
4. The file is owned by user 'walbert' and group 'support'.

UNIT – 3
UNIX Shell Command

Connecting Unix Shell

Telnet

Telnet is a protocol, which is used on internet or Local Area Networks (LAN) to connect with UNIX remote machine. Lot of times when user needs to connect with the remote UNIX terminal through network. Telnet utility is one of the important utility, which is used to connect with remote machine and work on that machine remotely.

- TELNET command is used to communicate with another host with using TELNET protocol.
- Just like the FTP command if telnet enters in to its command mode. The prompt of telnet is

Syntax: telnet [host [port]]

$telnet 10.20.208.110

Options	Description
-4	Force IPv4 address resolution.
-6	Force IPv6 address resolution.
-8	Request 8-bit operation. This causes an attempt to negotiate the **TELNET BINARY** option for both input and output. By default **telnet** is not "8-bit clean" (it does not recognize 8-bit character encodings such as Unicode).
-E	Disables the escape character functionality; that is, sets the escape character to "no character".
-L	Specifies an 8-bit data path on output. This causes the **TELNET BINARY** option to be negotiated on just output.
-a	Attempt automatic login. Currently, this sends the username via the **USER** variable of the **ENVIRON** option if supported by the remote system. The username is retrieved via the **getlogin** system call.
-b ***address***	Use bind on the local socket to bind it to a specific local address.
-d	Sets the initial value of the debug toggle to **TRUE**.
-r	Emulate **rlogin**. In this mode, the default escape character is a tilde. Also, the interpretation of the escape character is changed: an escape character followed by a dot causes **telnet** to disconnect from the remote host. A **^Z** (Control-Z) instead of a dot suspends **telnet**, and a **^]** (Control-Close Bracket, the default **telnet** escape character) generates a normal **telnet** prompt. These codes are accepted only at the beginning of a line.
-S ***tos***	Sets the IP type-of-service (TOS) option for the **telnet** connection to the value *tos*.
-e***escapechar***	Sets the escape character to *escapechar*. If no character is supplied, no escape character will be used. Entering the escape character while connected causes **telnet** to drop to command mode.
-l ***user***	Specify *user* as the user to log in as on the remote system. By sending the specified name as the **USER**environment variable, so it requires that the remote system support the **TELNET ENVIRON** option. This option implies the **-a** option, and may also be used with the **open** command.
-n ***tracefile***	Opens *tracefile* for recording trace information. See the **set tracefile**

	command below.
host	Specifies a host to contact over the network.
port	Specifies a port number or service name to contact. If not specified, the telnet port (**23**) is used.

Login Command

1. Passwd

passwd command is used to change the password of system users. If the passwd command is executed by non-root user then it will ask for the current password and then set the new password of a user who invoked the command. When this command is executed by super user (Administrator) or root then it can reset the password for any user including root without knowing the current password.

Options	DESCRIPTION
-a, --all	This option can be used only with -S and causes show status for all users.
-d, --delete	Delete a user's password (make it empty). This is a quick way to disable a password for an account. It will set the named account passwordless.
-e, --expire	Immediately expire an account's password. This in effect can force a user to change his/her password at the user's next login.
-h, --help	Display help message and exit.
-g, --noheadings	Do not print a header line.
-h, --help	Display help text and exit.
-i, --inactive INACTIVE	This option is used to disable an account after the password has been expired for a number of days. After a user account has had an expired password for INACTIVE days, the user may no longer sign on to the account.
-k, --keep-tokens	Indicate password change should be performed only for expired authentication tokens (passwords). The user wishes to keep their non-expired tokens as before.
-l, --lock	Lock the password of the named account. This option disables a password by changing it to a value which matches no possible encrypted value (it adds a '!' at the beginning of the password). Note that this does not disable the account. The user may still be able to login using another authentication token (e.g. an SSH key). To disable the account, administrators should use usermod --expiredate 1 (this set the account's expire date to Jan 2, 1970). Users with a locked password are not allowed to change their password.
-n, --mindays MIN_DAYS	Set the minimum number of days between password changes to MIN_DAYS. A value of zero for this field indicates that the user may change his/her password at any time.
-q, --quiet	Quiet mode.
-r, --repository REPOSITORY	change password in REPOSITORY repository
-R, --root CHROOT_DIR	Apply changes in the CHROOT_DIR directory and use the configuration files from the CHROOT_DIR directory.
-S, --status	Display account status information. The status information consists of 7 fields. The first field is the user's login name. The second field indicates if the user account has a locked password (L), has no password (NP), or has a usable password (P). The third field gives the date of the last password change. The next four fields are the minimum age, maximum age, warning period, and inactivity period for the password. These ages are expressed in days.

-u, --unlock	Unlock the password of the named account. This option re-enables a password by changing the password back to its previous value (to the value before using the -l option).
-w, --warndays WARN_DAYS	Set the number of days of warning before a password change is required. The WARN_DAYS option is the number of days prior to the password expiring that a user will be warned that his/her password is about to expire.
-x, --maxdays MAX_DAYS	Set the maximum number of days a password remains valid. After MAX_DAYS, the password is required to be changed.

$ passwd

output:

$ passwd
Changing password for ubuntu.
(current) UNIX password:
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully

2. LOGOUT

logout command allows you to programmatically logout from your session. Causes the session manager to take the requested action immediately. Logging out of UNIX may be achieved simply by typing logout, or <ctrl-D> or exit

$ logout

output:

no output on screen, current user session will be logged out.

Option	Description
-a, **--all**	same as **-b -d --login -p -r -t -T -u**
-b, **--boot**	time of last system boot
-d, **--dead**	print dead processes
-H, **--heading**	print line of column headings
-l, **--login**	print system login processes
--lookup	attempt to cannibalize hostnames via DNS
-m	only hostname and user associated with stdin
-p, **--process**	print active processes spawned by init
-q, **--count**	all login names and number of users logged on
-r, **--runlevel**	print current run level
-s, **--short**	print only name, line, and time (default)
-t, **--time**	print last system clock change
-T, **-w**, **--mesg**	add user's message status as +, - or ?
-u, **--users**	list users logged in
--message	same as **–T**
--writable	same as **-T**
--help	display this help and exit
--version	output version information and exit

3. Who am i

Who am i prints the effective user ID. This command prints the username associated with the current effective user ID. Running who am i is the same as running the id command with the options -un.

Syntax: who am i [OPTION]

Option	Description
--help	display this help and exit
--version	output version information and exit

4. clear

clear is a standard Unix computer operating system command that is used to clear the terminal screen. This command first looks for a terminal type in the environment and after that, it figures out the *terminfo* database for how to clear the screen. And this command will ignore any command-line parameters that may be present. Also, the *clear* command doesn't take any argument and it is almost similar to *cls* command on a number of other operating systems.

Syntax:$ clear

5.Uname

Uname stands for UNIX name. It is a utility to check the system information of your Linux computer. The uname command is commonly used to check Linux kernel version, kernel release details, and hardware architecture (32-bit or 64-bit). The information from uname comes useful for troubleshooting (identifying potential mismatches), system upgrades (knowing the current kernel version and architecture can be pivotal), and scripting.

When used without any options, the uname command displays only the operating system name. However, you can use various options to display specific system information, such as -a for all information, -n for network node hostname, -r displays the kernel release version, etc.

The basic syntax of the uname command: uname [option]

Where option indicates a specific type of system information you want to retrieve.

Note: The uname command does not require any special privileges. Any user on a Linux or Unix-like system can run uname to obtain basic system information

The following table list some of the useful options of uname with its description.

Options	Description
-a	Display all available system information.
-n	Display only the network node hostname.
-s	Display the kernel name.
-r	Display the Kernel release version.
-v	Display the kernel version build information.
-m	Display the hardware platform name of the system.
-p	Display the processor type. (This option may not be supported on all platforms.)
-i	Display the hardware platform. (This option might not be available on all systems.)
-o	Display the operating system name.

File / Directory Related Commands

1. LS COMMAND

"ls" command is used to list directory contents.

Syntex: - ls [OPTION]... [FILE]..

- It is a command to list files in UNIX and Unix-like operating systems.
- It is specified by POSIX and the Single UNIX Specification.
- It is works same as DIR command of DOS operating System
- The command can display files and directory of working present directory or any other directory.
- It is one of the most frequently used commands in UNIX.
- Various options are available to list directory and file which are as under

2. Cat Command

The cat command (short for "**concatenate** ") is one of the most frequently used command in Linux/Unix.

- **Cat** command allows us to create single or multiple files, view contain of file, concatenate files and redirect output in terminal or files.
- It is a standard UNIX program used to concatenate and display files.
- Cat command display file contents to a screen.
- Cat command concatenate FILE(s), or standard input, to standard output. With no

Options	Description
Ls	List Files using ls with no option
Ls –l	Display All Information About Files/Directories
Ls –a	View Hidden Files
Ls –lh	List Files with Human Readable Format with option
Ls –F	List Files and Directories with '/' Character at the end
Ls –r	List Files in Reverse Order
Ls –R	Recursively list Sub-Directories
Ls-ltr	Reverse Output Order
Ls – lS	Sort Files by File Size
Ls – i	Display Inode number of File or Directory
Ls –-version	Shows version of ls command
Ls –help	Show Help Page
Ls –l / tmp	List Directory Information
Ls – n	Display UID and GID of Files
Alias ls = "ls –l"	ls command and its Aliases
Unalias ls	Remove ls command and its Aliases
ls – 1	Display One File Per Line

FILE, or when FILE is -,

- It reads standard input. Also, you can use cat command for quickly creating a file.
- The cat command can read and write data from standard input and output devices.
- It has three main functions related to manipulating text files: creating them, displaying them, and combining them.

The cat command is used for:

- Display text file on screen **$ cat filename**
- Create a new text file (>) **$cat > filename**
- Read text file
- Modifying file (>>) **$ cat >> filename**
- File concatenation

Options	Description
-A, **--show-all**	Equivalent to **-vET**.
-b, **--number-nonblank**	Number non-empty output lines. This option overrides **-n**.
-e	Equivalent to **-vE**.
-E, **--show-ends**	Display "**$**" at end of each line.
-n, **--number**	Number all output lines.
-s, **--squeeze-blank**	Suppress repeated empty output lines.
-t	Equivalent to **-vT**.
-T, **--show-tabs**	Display TAB characters as **^I**.
-v, **--show-nonprinting**	Use **^** and **M-** notation, except for **LFD** and **TAB**.
--help	Display a help message, and exit.
--version	Output version information, and exit.

3. Cd Command

To change directory - change the current working directory to a specific Folder.

Syntax: cd [Options] [Directory]

If directory is given, changes the shell's working directory to directory. If not, changes to HOME (shell variable). If the shell variable CDPATH exists, it is used as a search path. If directory begins with a slash, CDPATH is not used. If directory is '-', this will change to the previous directory location (equivalent to $OLDPWD). The return status is zero if the directory is successfully changed, non-zero otherwise.

Options	Description
-P	Do not follow symbolic links
-L	Follow symbolic links (default)

Options	Description
cd directory_name	This command is used to navigate to a directory with white spaces. Instead of using double quotes we can use single quotes then also this command will work.
cd /	This command is used to change directory to the root directory, The root directory is the first directory in your filesystem hierarchy.
cd dir_1/dir_2/dir_3	This command is used to move inside a directory from a directory
cd ~ / cd	This command is used to change directory to the home directory.
cd ..	This command is used to move to the parent directory of current directory, or the directory one level up from the current directory. ".." represents parent directory.

4. PWD command

pwd‘ stands for **‘Print Working Directory**‘. As the name states, command ‘pwd‘ prints the current working directory or simply the directory user is, at present. It prints the current directory name with the complete path starting from root (/). This command is built in shell command and is available on most of the shell – bash, Bourne shell, ksh,zsh, etc.

Options	Description
-L (logical)	Use PWD from environment, even if it contains symbolic links
-P (physical)	Avoid all symbolic links
–help	Display this help and exit
–version	Output version information and exit

If both ‘-L‘ and ‘-P‘ options are used, option ‘L‘ is taken into priority. If no option is specified at the prompt, pwd will avoid all symlinks, i.e., take option ‘-P‘ into account.

Exit status of command pwd:

Options	Description
0	Success
Non-zero	Failure

5. MV Command

mv stands for **move**. mv is used to move one or more files or directories from one place to another in file system like UNIX. It has two distinct functions:

- It rename a file or folder.
- It moves group of files to different directory.

No additional space is consumed on a disk during renaming. This command normally **works silently** means no prompt for confirmation.

Syntax: mv [options] source dest

option	description
-f	force move by overwriting destination file without prompt
-i	interactive prompt before overwrite
-u	update - move when source is newer than destination
-v	verbose - print source and destination files
man mv	help manual

6. CP command

cp stands for copy. This command is used to copy files or group of files or directory. It creates an exact image of a file on a disk with different file name. cp command require at least two filenames in its arguments.

Syntax: cp [options]... Source Dest
cp [options]... Source... Directory

Options	Description
-a	same as -dpR
-b	make backup before removal
-de	preserve links
-f	remove existing destinations, never prompt
-i	prompt before overwrite
-l	link files instead of copying
-p	preserve file attributes if possible
-P	append source path to DIRECTORY
-r	copy recursively, non-directories as files

--sparse=WHEN	control creation of sparse files
-R	copy directories recursively
-s	make symbolic links instead of copying
-S	override the usual backup suffix
-u	copy only when the SOURCE file is newer than the destination file or when the destination file is missing
-v	explain what is being done
-V	override the usual version control
-x	stay on this file system
--help	display this help and exit
--version	Output version information and exit.

7. ln command

The ln command is used to create links between files. Before going into the application of the ln command in detail, please refer the below link for a clear understanding of the hard link and soft link in Linux.

Syntax:

ln [OPTION]... [-T] TARGET LINK_NAME (1st form)
ln [OPTION]... TARGET... DIRECTORY (2nd form)
ln [OPTION]... -t DIRECTORY TARGET... (3rd form)

Basically, ln command is used to create hard links and soft links for files in Linux. Let's discuss all the three forms one by one.

- **1st Form:** This form is simple, the source file with destination link name you have to specify:
- **2nd Form:** Here, we have to give source file and directory as a link, simply the source file will be copied to the directory which you specify. See the example below.
- **3rd Form:** Here, we will specify the directory name and a file to be linked to the directory. It is mostly same to 2nd form.

Options	Description
-backup[=CONTROL]	Use this option to additionally create a backup of each existing destination file. The style of backup is optionally defined by the value of CONTROL.
-b	This functions like --backup, but you cannot specify the CONTROL; the default style (simple) is used.
-d, -F, --directory	This option allows the super user to attempt to hard link directories (although it will probably fail due to system restrictions, even for the superuser).
-f, --force	If the destination file or files already exist, overwrite them.
-i, --interactive	Prompt the user before overwriting destination files.
-L, --logical	Dereference TARGETs that are symbolic links. In other words, if you are trying to create a link (or a symlink) to a symlink, link to what it links to, not to the symlink itself..
-n, --no-dereference	Treat LINKNAME as a normal file if it is a symbolic link to a directory.
-P, --physical	Make hard links directly to symbolic links, rather than dereferencing them.
-r, --relative	Create symbolic links relative to link location.
-s, --symbolic	Make symbolic links instead of hard links.
-S, --suffix=SUFFIX	Use the file suffix SUFFIX rather than the default suffix "~".
-t, --target-	Specify the DIRECTORY in which to create the links.

directory=DIRECTORY	
-T, --no-target-directory	Always treat LINKNAME as a normal file.
-v, --verbose	Operate verbosely; print the name of each linked file.
--help	Display a help message, and exit.
--version	Display version information, and exit.

8. rm command

rm command is used to remove objects such as files, directories, symbolic links and so on from the file system like UNIX. To be more precise, rm removes references to objects from the file system, where those objects might have had multiple references

By default, it does not remove directories.

This command normally works silently and you should be very careful while running **rm** command because once you delete the files then you are not able to recover the contents of files and directories.

Syntax: rm [OPTION]... FILE...

Options	Description
-f, **--force**	Ignore nonexistant files, and never prompt before removing.
-i	Prompt before every removal.
-I	Prompt once before removing more than three files, or when removing recursively. This option is less intrusive than **-i**, but still gives protection against most mistakes.
--interactive[=*WHEN*]	Prompt according to *WHEN*: **never**, **once** (**-I**), or **always** (**-i**). If *WHEN* is not specified, then prompt always.
--one-file-system	When removing a hierarchy recursively, skip any directory that is on a file system different from that of the corresponding command line argument
--no-preserve-root	Do not treat "/" (the root directory) in any special way.
--preserve-root	Do not remove "/" (the root directory), which is the default behavior.
-r, **-R**, **--recursive**	Remove directories and their contents recursively.
-d,**--dir**	Remove empty directories. This option permits you to remove a directory without specifying **-r**/**-R**/**--recursive**, provided that the directory is empty. In other words, **rm -d** is equivalent to using **rmdir**.
-v, **--verbose**	Verbose mode; explain at all times what is being done.
--help	Display a help message, and exit.
--version	Display version information, and exit.

9. Rmdir

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

Syntax: rmdir [-p] [-v | --verbose] [--ignore-fail-on-non-empty] directory ...

Options	Description
-p	Each directory argument is treated as a pathname of which all components will be removed, if they are empty, starting with the last component.
-v, --verbose	Display verbose information for every directory processed.

--ignore-fail-on-non-empty	Do not report a failure which occurs solely because a directory is non-empty. Normally, when rmdir is instructed to remove a non-empty directory, it reports an error. This option suppresses those error messages.
--help	Display a help message, and exit.
--version	Output version information and exit.

10. Mkdir

mkdir command in Linux allows the user to create directories (also referred to as folders in some operating systems). This command can create multiple directories at once as well as set the permissions for the directories. It is important to note that the user executing this command must have enough permissions to create a directory in the parent directory, or he/she may recieve a 'permission denied' error.

Syntax: mkdir [options...] [directories ...]

Options	Description
directory	The name of the directory to be created. If the specified *directory* does not already exist, **mkdir** creates it. More than one *directory* may be specified. A specified *directory* can include path information. For instance, if the current directory is **/home/hope**, and you'd like to create the directory **/home/hope/Documents/writing**, you can use the command **mkdir Documents/writing**. If the **Documents** folder does not already exist, you should specify the **-p** option to create it automatically, otherwise the command will fail.
-m=*mode*, **--mode=***mode*	You can use the **-m** option to set a file mode (permissions, etc.) for the created directories. The syntax of *mode* is the same as with the **chmod** command.
-p, **--parents**	Create parent directories as necessary. When this option is specified, no error is reported if a *directory* already exists.
-v, **--verbose**	Verbose output. Print a message for each created directory.
-Z=*context*, **--context=***context*	If you are using SELinux, this option sets the security context of each created directory to *context*. For detailed information about security contexts, consult your SELinux documentation.
--help	Display a help message, and exit.
--version	Display version information, and exit.

11. Chmod

The chmod command is used to change the access mode of a file. The name is an abbreviation of change mode Throughout this section, user refers to the owner of the file, as a reminder that the symbolic form of the command uses "u".

Syntax: chmod [options] mode[,mode] file1 [file2 ...]

Usual implemented options include:

-R Recursive, i.e. include objects in subdirectories.

-v verbose, show objects changed (unchanged objects are not shown).

-m mode, sets the file mode (chmod), not a r=rwx -umask.

If a symbolic link is specified, the target object is affected. File modes directly associated with symbolic links themselves are typically not used. To view the file mode, the ls or stat commands may be used:

$ ls -l findPhoneNumbers.sh

-rwxr-xr-- 1 dgerman staff 823 Dec 16 15:03 findPhoneNumbers.sh

$ stat -c %a findPhoneNumbers.sh
754

The r, w, and x specify the read, write, and execute access. The first character of the ls display denotes the object type; a hyphen represents a plain file. This script can be read, written to, and executed by the user ; read and executed by members of the staff group; and only read by any other users.

Octal modes

The main parts of the chmod permissions:
For example: drwxrwx---

The characters to the right of the "d" define permissions for each class:

- The three leftmost(first) characters, rwx, define permissions for the user class
- The middle three characters, rwx, define permissions for the Group class.
- The last three characters, ---, define permissions for the Others class.

In this example, users who are not the owner of the file and who are not members of the Group (and, thus, are in the Others class) have no permission to access the file.

Numerical permissions

The chmod numerical format accepts up to four octal digits. The three rightmost digits define permissions for the file user, the Group, and Others. The optional leading digit, when 4 digits are given, specifies the special setuid, setgid, and sticky flags. Each digit of the three rightmost digits represents a binary value, which controls the "read", "write" and "execute" permissions respectively. A value of 1 means a class is allowed that action, while a 0 means it is disallowed.

#	Permission	rwx	Binary
7	read, write and execute	rwx	111
6	read and write	rw-	110
5	read and execute	r-x	101
4	read only	r--	100
3	write and execute	-wx	011
2	write only	-w-	010
1	execute only	--x	001
0	none	---	000

For example, 754 would allow:

- "read", "write", and "execute" for the user class, as the binary value of 7 is 111.
- "read" and "execute" for the Group class, as the binary value of 5 is 101.
- Only "read" for the Others class, as the binary value of 4 is 100.

Numeric example

Change permissions to permit members of the programmers group to update a file.

$ ls -l sharedFile
-rw-r--r-- 1 jsmith programmers 57 Jul 3 10:13 sharedFile
$ chmod 664 sharedFile
$ ls -l sharedFile
-rw-rw-r-- 1 jsmith programmers 57 Jul 3 10:13 sharedFile

Since the setuid, setgid and sticky bits are not specified, this is equivalent to:

$ chmod 0664 sharedFile

Symbolic modes[edit]

The chmod command also accepts a finer-grained symbolic notation,which allows modifying specific modes while leaving other modes untouched. The symbolic mode is composed of three components, which are combined to form a single string of text:

$ chmod [references][operator][modes] file ...

Classes of users are used to distinguish to whom the permissions apply. If no classes are specified “all” is implied. The classes are represented by one or more of the following letters:

Reference	Class	Description
u	user	file owner
g	group	members of the file's group
o	others	users who are neither the file's owner nor members of the file's group
a	all	all three of the above, same as ugo

The chmod program uses an operator to specify how the modes of a file should be adjusted. The following operators are accepted:

Operator	Description
+	adds the specified modes to the specified classes
-	removes the specified modes from the specified classes
=	the modes specified are to be made the exact modes for the specified classes

The modes indicate which permissions are to be granted or removed from the specified classes. There are three basic modes which correspond to the basic permissions:

Mode	Name	Description
r	read	read a file or list a directory's contents
w	write	write to a file or directory
x	execute	execute a file or recurse a directory tree
X	special execute	which is not a permission in itself but rather can be used instead of x. It applies execute permissions to directories regardless of their current permissions and applies execute permissions to a file which already has at least one execute permission bit already set (either User, Group' or Others). It is only really useful when used with + and usually in combination with the -R flag for giving Group or Others access to a big directory tree without setting execute permission on normal files (such as text files), which would normally happen if you just used chmod -R a+rx ., whereas with X you can do chmod -R a+rX . instead

Multiple changes can be specified by separating multiple symbolic modes with commas (without spaces). If a user is not specified, chmod will check the umask and the effect will be as if "a" was specified except bits that are set in the umask are not affected.

Symbolic examples

Add write permission (w) to the Group's (g) access modes of a directory, allowing users in the same group to add files:

$ ls -ld shared_dir # show access modes before chmod
drwxr-xr-x 2 teamleader usguys 96 Apr 8 12:53 shared_dir
$ chmod g+w shared_dir
$ ls -ld shared_dir # show access modes after chmod
drwxrwxr-x 2 teamleader usguys 96 Apr 8 12:53 shared_dir

Remove write permissions (w) for all classes (a), preventing anyone from writing to the file:

$ ls -l ourBestReferenceFile

-rw-rw-r-- 2 teamleader usguys 96 Apr 8 12:53 ourBestReferenceFile
$ chmod a-w ourBestReferenceFile
$ ls -l ourBestReferenceFile
-r--r--r-- 2 teamleader usguys 96 Apr 8 12:53 ourBestReferenceFile

Set the permissions for the user and the Group (ug) to read and execute (rx) only (no write permission) on referenceLib, preventing anyone to add files.

$ ls -ld referenceLib
drwxr----- 2 teamleader usguys 96 Apr 8 12:53 referenceLib
$ chmod ug=rx referenceLib
$ ls -ld referenceLib
dr-xr-x--- 2 teamleader usguys 96 Apr 8 12:53 referenceLib

12. chown

The chown command is most commonly used by Unix/Linux system administrators who need to fix a permissions problem with a file or directory, or many files and many directories.

Syntax:

- **chown [Options]... NewOwner File...**
- **chown [Options]... :Group File...**
- **chown [Options]... --reference=RFILE File...**

if used, NewOwner specifies the new owner and/or group as follows(with no embedded white space):

chown [OWNER] [[:.] [GROUP]]

Following are the examples of how the owner/group can be specified: If only an OWNER (a user name or numeric user id) is given, that user is made the owner of each given file, and the files' group is not changed.

chown OWNER

If the OWNER is followed by a colon or dot and a GROUP (a group name or numeric group id), with no spaces between them, the group ownership of the files is changed as well (to GROUP).

chown OWNER.GROUP
chown OWNER:GROUP

If a colon or dot but no group name follows OWNER, that user is made the owner of the files and the group of the files is changed to OWNER's login group.

chown OWNER.
chown OWNER:

If the colon or dot and following GROUP are given, but the owner is omitted, only the group of the files is changed; in this case, 'chown' performs the same function as 'chgrp'.

chown .GROUP
chown :GROUP

OPTIONS

Tag	Description
-c –changes	Verbosely describe the action for each File whose ownership actually changes.
--dereference	Do not act on symbolic links themselves but rather on what they point to.
-f –silent –quiet	Do not print error messages about files whose ownership cannot be changed.

-h --no-dereference	Act on symbolic links themselves instead of what they point to. This is the default. This mode relies on the 'lchown' system call. On systems that do not provide the 'lchown' system call, 'chown' fails when a file specified on the command line is a symbolic link. By default, no diagnostic is issued for symbolic links encountered during a recursive traversal, but see '--verbose'.
--reference=FILE	Use the user and group of the reference FILE instead of an explicit NewOwner value.
-R –recursive	Recursively change ownership of directories and their contents.
-v –verbose	Verbosely describe the action (or non-action) taken for every FILE. If a symbolic link is encountered during a recursive traversal on a system without the 'lchown' system call, and '--no-dereference' is in effect, then issue a diagnostic saying neither the symbolic link nor its referent is being changed.

EXAMPLES

- Change the owner of file.
  $ chown user1 sample.txt
- Change the group of file.
  $ chown :mygroup file.txt
- Change both the owner and group of file in single command.
  $ chown user1:mygroup file.txt
  Print

13. Find

find command in UNIX is a command line utility for walking a file hierarchy. It can be used to find files and directories and perform subsequent operations on them. It supports searching by file, folder, name, creation date, modification date, owner and permissions. By using the '-exec' other UNIX commands can be executed on files or folders found.

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

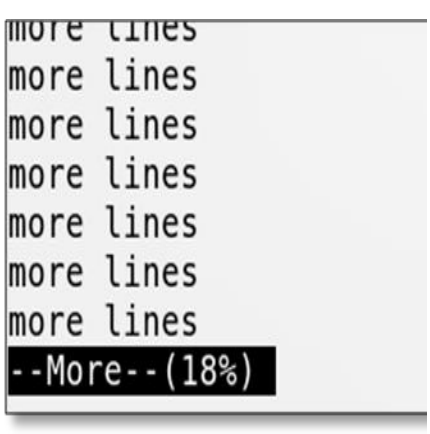
Option	Description
-exec CMD:	The file being searched which meets the above criteria and returns 0 for as its exit status for successful command execution.
-ok CMD	It works same as -exec except the user is prompted first.
-inum N	Search for files with inode number 'N'.
-links N	Search for files with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

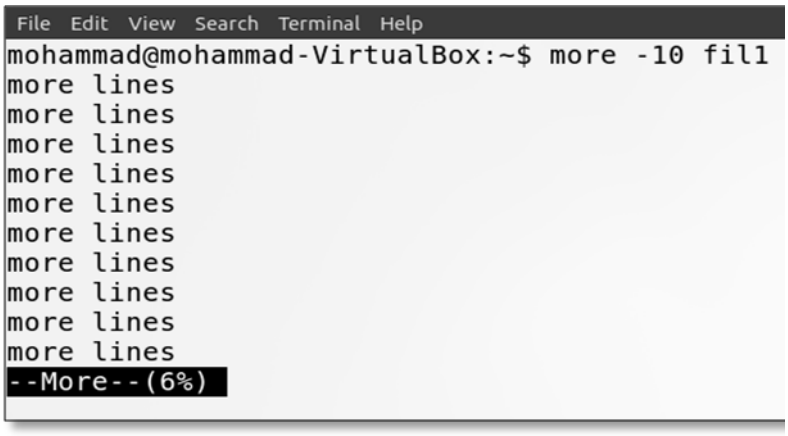
! expr	True if 'expr' is false.

14. More

- If the output of any file is not fitted on screen means not able to display in a single screen then the output will be automatically scroll up to its last page
- At that time more command is very helpful to display the file in pausing mode.
- The information will be scrolled as per user direction, if the enter key is pressed a single line will be scrolled, if the spacebar is pressed then a page will be scrolled
- The more command is having following options

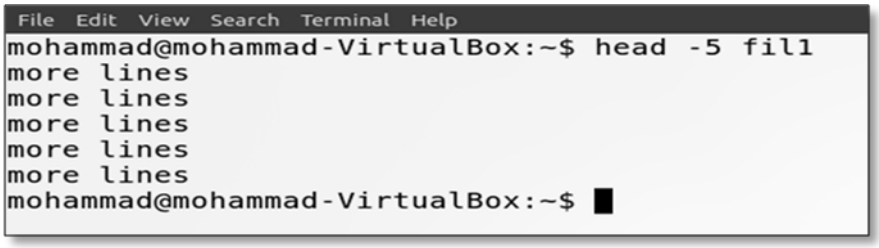
-num	Is to set number of lines for page
+num	To start output for give number of line of file





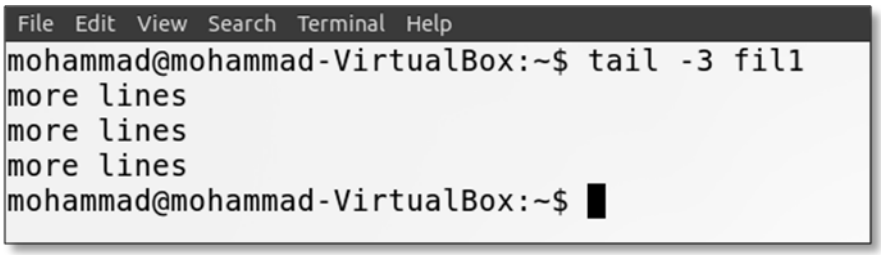
15. head

- The command is use to display a specified number of line from the starting of a file



16. tail

- The command is used to display a specified number of line from the last line of the file



17. wc

- It is use to count words, lines and characters of file(s)

- There are three options available

-l	To count number of lines
-c	To count number of characters
-w	To count number of words

```
File  Edit  View  Search  Terminal  Help
mohammad@mohammad-VirtualBox:~$ wc fil1
 148  296 1628 fil1
mohammad@mohammad-VirtualBox:~$ wc -l fil1
148 fil1
mohammad@mohammad-VirtualBox:~$ wc -c fil1
1628 fil1
mohammad@mohammad-VirtualBox:~$ wc -w fil1
296 fil1
mohammad@mohammad-VirtualBox:~$
```

```
File  Edit  View  Search  Terminal  Help
mohammad@mohammad-VirtualBox:~$ wc fil1 fil2
 148  296 1628 fil1
   4    8   44 fil2
 152  304 1672 total
mohammad@mohammad-VirtualBox:~$
```

18. Stat command

The **stat** is a command which gives information about the file and filesystem. Stat command gives information such as the size of the file, access permissions and the user ID and group ID, birth time access time of the file. Stat command has another feature, by which it can also provide the file system information. This is the best tool to use when we want the information of any file.

syntax: stat --options filenames

Stat can accept one or more filenames as an input to the stat command. Now let's see one example

stat /etc/resolv.conf

The information we get from stat

Following is the information we get about the file when we run the stat command.

- **File:** The name of the provided file. If the provided file is a symlink, then the name will be different.
- **Size:** The size of a given file in Bytes.
- **Blocks:** Total number of allocated blocks to the file to store on the hard disk.
- **IO Block:** The size of every allocated block in bytes.
- **File type:** The file may be of the following types: Regular files, special files, directories, or symbolic links.
- **Device:** Device number in hexadecimal format.

- **Inode:** Inode number of the file.
- **Links:** Number of hard links of the file.
- **Access:** Access permissions in the numeric and symbolic methods.
- **Context:** The field stores SELinux security context.
- **Access:** The last time at which the file was accessed.
- **Modify:** The last time at which file was modified.
- **Change:** The last time the at which file's attribute or content was changed.
- **Birth:** The time at which the file was created.

Options	Description
-f, --filesystem	display filesystem status instead of file status
-c, --format=FORMAT	use the specified FORMAT instead of the default
-L, --dereference	follow links
-Z, --context	print the SELinux security context
-t, --terse	print the information in terse form
--help	display this help and exit
--version	output version information and exit

The valid format sequences for files (without --filesystem):

%A	Access rights in human readable form
%a	Access rights in octal
%B	The size in bytes of each block reported by '%b'
%b	Number of blocks allocated (see %B)
%C	SELinux security context string
%D	Device number in hex
%d	Device number in decimal
%F	File type
%f	Raw mode in hex
%G	Group name of owner
%g	Group ID of owner
%h	Number of hard links
%i	Inode number
%N	Quoted File name with dereference if symbolic link
%n	File name
%o	IO block size
%s	Total size, in bytes
%T	Minor device type in hex
%t	Major device type in hex
%U	Username of owner
%u	User ID of owner
%X	Time of last access as seconds since Epoch
%x	Time of last access
%Y	Time of last modification as seconds since Epoch
%y	Time of last modification
%Z	Time of last change as seconds since Epoch
%z	Time of last change

Valid format sequences for file systems:

%a	Free blocks available to non-superuser
%b	Total data blocks in file system
%c	Total file nodes in file system
%C	SELinux security context string
%d	Free file nodes in file system
%f	Free blocks in file system
%i	File System id in hex
%l	Maximum length of file names
%n	File name
%s	Optimal transfer block size
%T	Type in human readable form
%t	Type in hex

Examples
stat index.htm
Reports the status of file index.htm, displaying results similar to the following output:
File: `index.htm'
Size: 17137 Blocks: 40 IO Block: 8192 regular file
Device: 8h/8d Inode: 23161443 Links: 1
Access: (0644/-rw-r--r--)
Uid: (17433/comphope) Gid: (32/ www)
Access: 2007-04-03 09:20:18.000000000 -0600
Modify: 2007-04-01 23:13:05.000000000 -0600
Change: 2007-04-02
16:36:21.000000000 -0600

19. Alias

Depending on the type of work you do on your Linux system, you may need to enter the same long and complicated commands frequently. The **alias** command lets you create shortcuts for these commands, making them easier to remember and use.

an alias is a shortcut that references a command. An alias replaces a string that invokes a command in the Linux shell with another user-defined string.

Aliases are mostly used to replace long commands, improving efficiency and avoiding potential spelling errors. Aliases can also replace commands with additional options, making them easier to use.

Linux Alias Syntax : alias [option] [name]='[value]'

The different elements of the **alias** command syntax are:

- **alias**: Invokes the **alias** command.
- **[option]**: Allows the command to list all current aliases.
- **[name]**: Defines the new shortcut that references a command. A name is a user-defined string, excluding special characters and **'alias'** and **'unalias'**, which cannot be used as names.
- **[value]**: Specifies the command the alias references. Commands can also include options, arguments, and variables. A value can also be a path to a script you want to execute.

Creating Permanent Aliases in Linux

To keep **aliases** between sessions, you can save them in your user's shell configuration profile file. This can be:

- Bash – **~/.bashrc**
- ZSH – **~/.zshrc**
- Fish – **~/.config/fish/config.fish**

The syntax you should use is practically the same as creating a temporary alias. The only difference comes from the fact that you will be saving it in a file this time. So for example, in bash, you can open a **.bashrc** file with your favorite editor like this.

20. Type

The **type** command is used to describe how its argument would be translated if used as commands. It is also used to find out whether it is built-in or external binary file.

Syntax: type [Options] command names

Option	Deception
–a	display all locations containing an executable named NAME; includes aliases, built-ins, and functions, if and only if the '-p' option is not also used
-f	suppress shell function lookup
-P	force a PATH search for each NAME, even if it is an alias, built-in, or function, and returns the name of the disk file that would be executed
-p	returns either the name of the disk file that would be executed or nothing if `type -t NAME' would not return `file'.
-t	output a single word which is one of `alias', `keyword', `function', `built-in', `file' or `', if NAME is an alias, shell reserved word, shell function, shell built-in, disk file, or not found, respectively

Operators in Redirection & PIPING

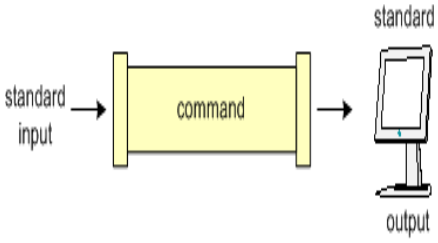
What is Redirection?

Redirection is a feature in Linux such that when executing a command, you can change the standard input/output devices. The basic workflow of any Linux command is that it takes an input and gives an output.

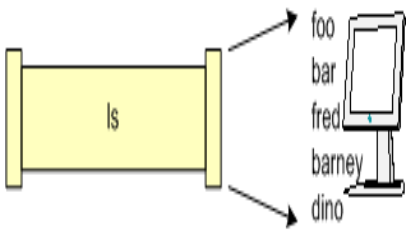
- The standard input (stdin) device is the keyboard.
- The standard output (stdout) device is the screen.

With redirection, the above standard input/output can be changed.

Commands are entered at the keyboard and output resulting from these commands is displayed on the computer screen. Thus, input (by default) comes from the terminal and the resulting output (stream) is displayed on (or directed to) the monitor. Commands typically get their input from a source referred to as standard input (stdin) and typically display their output to a destination referred to as standard output (stdout) as pictured below:



As depicted in the diagram above, input flows (by default) as a stream of bytes from standard input along a channel, is then manipulated (or generated) by the command, and command output is then directed to the standard output. The ls command can then be described as follows; there is really no input (other than the



command itself) and the ls command produces output which flows to the destination of stdout (the terminal screen), as below:

The notations of standard input and standard output are actually implemented in Unix as files (as are most things) and referenced by integer file descriptors (or channel descriptors). The file descriptor for standard input is 0 (zero) and the file descriptor for standard output is 1. These are not seen in ordinary use since these are the default values. Hear we will learn

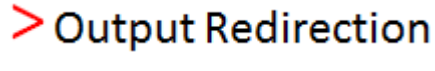
- Output Redirection
- Input redirection
- File Descriptors (FD)
- Error Redirection
- Why Error Redirection?

1. Output Redirection

The '>' symbol is used for output (STDOUT) redirection.

Example: ls -al > listings

Here the output of command ls -al is re-directed to file "listings" instead of your screen.




```
home@VirtualBox:~$ ls -al > listings
home@VirtualBox:~$ cat listings
total 324
drwxr-xr-x 26 home home  4096 2012-09-10 10:42 .
drwxr-xr-x  3 root root  4096 2012-09-01 19:43 ..
-rw-rw-r--  1 home home     0 2012-09-10 09:25 abc
```

Use the correct file name while redirecting command output to a file. If there is an existing file with the same name, the redirected command will delete the contents of that file and then it may be overwritten."

If you do not want a file to be overwritten but want to add more content to an existing file, then you should use '**>>**' operator.

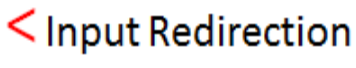
```
home@VirtualBox:~$ cat sample
Hang on for the best Linux Lessons.
home@VirtualBox:~$ echo Thanks for reading >> sample
home@VirtualBox:~$ cat sample
Hang on for the best Linux Lessons.
Thanks for reading
```

You can redirect standard output, to not just files

2. Input redirection

The '**<**' symbol is used for input(STDIN) redirection

Example: The mail program in Linux can help you send emails from the Terminal.



You can type the contents of the email using the standard device keyboard. But if you want to attach a File to email you can use the input re-direction operator in the following format.

Mail -s "Subject" to-address < Filename

This would attach the file with the email, and it would be sent to the recipient. The above examples were simple. Let's look at some advance re-direction techniques which make use of File Descriptors.

3. File Descriptors (FD)

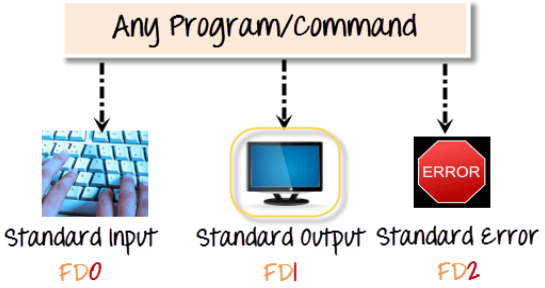
In Linux/Unix, everything is a file. Regular file, Directories, and even Devices are files. Every File has an associated number called File Descriptor (FD).

Your screen also has a File Descriptor. When a program is executed the output is sent to File Descriptor of the screen, and you see program output on your monitor. If the output is sent to File Descriptor of the printer, the program output would have been printed.

4. Error Redirection

Whenever you execute a program/command at the terminal, 3 files are always open, viz., standard input, standard output, standard error.

These files are always present whenever a program is run. As explained before a file descriptor, is associated with each of these files.



File	File Descriptor
Standard Input STDIN	**0**
Standard Output STDOUT	**1**
Standard Error STDERR	**2**

By default, error stream is displayed on the screen. Error redirection is routing the errors to a file other than the screen.

5. Why Error Redirection?

Error re-direction is one of the very popular features of Unix/Linux. Frequent UNIX users will reckon that many commands give you massive amounts of errors.

- For instance, while searching for files, one typically gets permission denied errors. These errors usually do not help the person searching for a particular file.
- While executing shell scripts, you often do NOT want error messages cluttering up the normal program output.

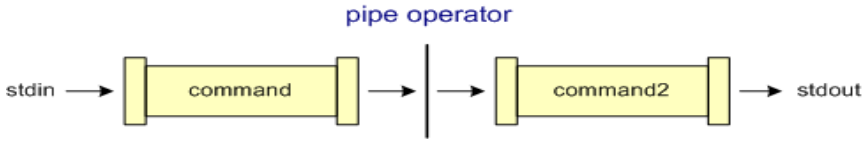
Redirection Operator	Resulting Operation
Command 1 > file	stdout written to file, overwriting if file exists
command 1 >> file	stdout written to file, appending if file exists
command 1 < file	input read from file
command 2 > file	stderr written to file, overwriting if file exsits
command 2 >> file	stderr written to file, appending if file exists
command > file 2>&1	stdout written to file, stderr written to same file descriptor

Pipe Operator

What is a Pipe in Linux?

The Pipe is a command in Linux that lets you use two or more commands such that output of one command serves as input to the next. In short, the output of each process directly as input to the next one likes a pipeline. The symbol '|' denotes a pipe.

Pipes help you mash-up two or more commands at the same time and run them consecutively. You can use powerful commands which can perform complex tasks in a moment.



We can look at an example of pipes using the who and the wc commands. who command will list each user logged into a machine and wc command counts characters, words and lines. Thus if we connect the standard output from the who command to the standard input of the wc (using the -l (ell) option), we can count the number of users on the system:

$ who | wc -l

In the first part of this example, each of the four lines from the who command will be "piped" into the wc command, where the -l (ell) option will enable the wc command to count the number of lines.

While this example only uses two commands connected through a single pipe operator, many commands can be connected via multiple pipe operators.

Advance Tools

Metacharacters

Metacharacters are a group of characters that have special meanings to the UNIX operating system. Metacharacters can make many tasks easier by allowing you to redirect information from one command to another or to a file, string multiple commands together on one line, or have other effects on the commands they are issued in. The following table lists some of the metacharacters for the Rutgers default shell (the T shell).

character	Description
(space)	UNIX interprets a space as a separator not as a character.
*	A wild card character that matches any group of characters of any length, allowing a user to specify a large group of items with a short string. For example, to specify all the files that start with 'abc', you use **abc***.
?	A wild card character that matches any single character. Thus **ls ???** lists files in the current directory whose names are only three characters long, while **ls ???.*** lists those files with a three letter main name and any extension.
[..]	A set of characters that can be matched. Thus **ls [a-c]*.???** lists all files that begin with a, b, or c and have a three letter extension and **lpr [ad]*** prints all files that begin with a or d.
$	Indicates that the following text is the name of a shell (environment) variable whose value is to be used.
|	Separates commands to form a pipe (see redirection in "Intermediate Use Of The UNIX Operating System").
<	Redirect the standard input (see redirection in "Intermediate Use Of The UNIX Operating System").

>	Redirect the standard output (see redirection in "Intermediate Use Of The UNIX Operating System") to replace current contents.
>>	Redirect the standard output (see redirection in "Intermediate Use Of The UNIX Operating System") to append to current contents.
>&	Redirect the standard output and standard error (see redirection in "Intermediate Use Of The UNIX Operating System") to replace current contents.
>>&	Redirect the standard output an standard error (see redirection in "Intermediate Use Of The UNIX Operating System") to append to current contents.
%	Introduces a job name (see multitasking in "Intermediate Use Of The UNIX Operating System").
&	Place a process into the background (see multitasking in "Intermediate Use Of The UNIX Operating System").
()	Encloses a sequence of commands or pipes to be executed as a single command.
!	Precedes a history substitution (see "man history")
;	Separates sequences of commands (or pipes) that are on one line.
&&	Separates two sequences of commands or pipes the second of which is executed only if the first succeeds.
||	Separates two sequences of commands or pipes the second of which is executed only if the first fails.
\	Used to "quote" the following metacharacter so it is treated as a plain character, as in *.

Finding pattern in Files

1. GREP

The name *grep* means "Global regular expression print", but you can think of the grep command as a "search" command for Unix and Linux systems: It's used to search for text strings and *regular expressions* within one or more files.

Syntax: grep [options] pattern [files]

Options	Description
-c	This prints only a count of the lines that match a pattern
-h	Display the matched lines, but do not display the filenames.
-i	Ignores, case for matching
-l	Displays list of a filenames only
-n	Display the matched lines and their line numbers.
-v	This prints out all the lines that do not matches the pattern
-e exp	Specifies expression with this option. Can use multiple times
-f file	Takes patterns from file, one per line.
-E	Treats pattern as an extended regular expression (ERE)
-w	Match whole word
-o	Print only the matched parts of a matching line with each such part on a separate output line

Example	Output
$cat > file.txt unix is great os. unix is opensource. unix is free os. learn operating system. Unix linux which one you choose. uNix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.	unix is great os. unix is opensource. unix is free os. Unix linux which one you choose. uNix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.

$grep -i "UNix" file.txt	

2. FGREP

fgrep searches for fixed-character strings in a file or files. "Fixed-character" means the string is interpreted literally meta characters do not exist, and therefore regular expressions cannot be used.

fgrep is useful when you need to search for strings which contain lots of regular expression meta characters, such as "$", "^", etc. By specifying that your search string contains fixed characters, you don't need to escape each of them with a backslash.

If your string contains newlines, each line will be considered an individual fixed-character string to be matched in the search.

Running fgrep is the same as running grep with the -F option.

Syntax: fgrep [options] [-e pattern_list] [pattern] [file]

Option	Description
-b	Precede each line by the block number on which it was found. This can be useful in locating block numbers by context (first block is 0).
-c	Print only a count of the lines that contain the pattern.
-h	Suppress printing of files when searching multiple files.
-i	Ignore upper/lower case distinction during comparisons.
-l	Print the names of files with matching lines once, separated by new-lines. Does not repeat the names of files when the pattern is found more than once.
-n	Precede each line by its line number in the file (first line is 1).
-s	Work silently, that is, display nothing except error messages. This is useful for checking the error status.
-v	Print all lines except those that contain the pattern.
-x	Print only lines matched entirely.
-e *pattern_list*	Search for a string in *pattern-list* (useful when the string begins with a "-").
-f *pattern-file*	Take the list of patterns from *pattern-file*.
pattern	Specify a *pattern* to be used during the search for input.
file	A path name of a *file* to be searched for the patterns. If no file operands are specified, the standard input will be used.

Return Value

This command returns the following exit values:

Item	Description
0	A match was found.
1	No match was found.
>1	A syntax error was found or a file was inaccessible (even if matches were found)

3. EGREP

The egrep command searches an input file (standard input by default) for lines matching a pattern specified by the Patternparameter. These patterns are full regular expressions as in the ed command (except for the \ (backslash) and \\ (double backslash)). The following rules also apply to the egrep command:

- A regular expression followed by a + (plus sign) matches one or more occurrences of the regular expression.
- A regular expression followed by a?(question mark) matches zero or one occurrence of the regular expression.
- Multiple regular expressions separated by a | (vertical bar) or by a new-line character match strings that are matched by any of the regular expressions.
- A regular expression may be enclosed in ( ) (parentheses) for grouping.

The new-line character will not be matched by the regular expressions. The order of precedence for operators is [,], *, ?, +, concatenation, | and the new-line character.

Syntax: -egrep [options] PATTERN [FILE...]

Option	Description
-b	Precedes each line by the block number on which it was found. Use this flag to help find disk block numbers by context. The -b flag cannot be used with input from stdin or pipes.
-c	Displays only a count of matching lines.
-e *Pattern*	Specifies a *Pattern*. This works like a simple *Pattern* but is useful when the *Pattern* begins with a - (minus sign).
-f *StringFile*	Specifies a file that contains strings.
-h	Suppresses file names when multiple files are being processed.
-i	Ignores the case of letters when making comparisons.
-l	Lists just the names of files (once) with matching lines. Each file name is separated by a new-line character. If standard input is searched, a path name of "(Standard Input)" is returned.
-n	Precedes each line with its relative line number in the file.
-p [*Separator*]	Displays the entire paragraph containing matched lines. Paragraphs are delimited by paragraph separators, as specified by the *Separator* parameter, which are patterns in the same form as the search pattern. Lines containing the paragraph separators are used only as separators; they are never included in the output. The default paragraph separator is a blank line.
-q	Suppresses all output to standard output, regardless of matching lines. Exits with a 0 status if an input line is selected.
-s	Displays only error messages. This is useful for checking status.
-u	Causes output to be unbuffered.
-v	Displays all lines except those that match the specified pattern.
-w	Does a word search.
-x	Displays lines that match the specified pattern exactly with no additional characters.
-y	Ignores the case of letters when making comparisons.

Working With Columns and Files

1. CUT

The cut command in UNIX is a command for cutting out the sections from each line of files and writing the result to standard output. It can be used to cut parts of a line by **byte position, character and field**. Basically the cut command slices a line and extracts the text. It is necessary to specify option with command otherwise it gives error. If more than one file name is provided then data from each file is **not precedes** by its file name.

Syntax: - cut OPTION... [FILE]...

Option	Description
-b, **--bytes=***LIST*	Select only the bytes from each line as specified in *LIST*. *LIST* specifies a byte, a set of bytes, or a range of bytes;
-c, **--characters=***LIST*	Select only the characters from each line as specified in *LIST*. *LIST* specifies a character, a set of characters, or a range of characters
-d, **--delimiter=***DELIM*	use character *DELIM* instead of a tab for the field delimiter.
-f, **--fields=***LIST*	Select only these fields on each line; also print any line that contains no delimiter character, unless the **-s** option is specified. *LIST* specifies a field, a set of fields, or a range of fields
-n	This option is ignored, but is included for compatibility reasons.
--complement	complement the set of selected bytes, characters or fields.
-s, **--only-delimited**	do not print lines not containing delimiters.
--output-delimiter=*STRING*	use *STRING* as the output delimiter string. The default is to use the input delimiter.
--help	Display a help message and exit.
--version	output version information and exit.

2. PASTE

Paste command is one of the useful commands in Unix or Linux operating system. It is used to join files horizontally (parallel merging) by outputting lines consisting of lines from each file specified, separated by **tab** as delimiter, to the standard output. When no file is specified, or put dash ("-") instead of file name, paste reads from standard input and gives output as it is until a interrupt command **[Ctrl-c]** is given.

Syntax: paste [OPTION]... [FILES]...

Option	Description
-d, **--delimiters=***LIST*	Reuse characters from *LIST* instead of tabs.
-s, **--serial**	Paste one file at a time instead of in parallel.
--help	Display a help message, and exit.
--version	Display version information, and exit.

3. JOIN

The join command in UNIX is a command line utility for joining lines of two files on a common field.

Suppose you have two files and there is a need to combine these two files in a way that the output makes even more sense. For example, there could be a file containing names and the other containing ID's and the requirement is to combine both files in such a way that the names and corresponding ID's appear in the same line. Join command is the tool for it. join command is used to join the two files based on a key field present in both the files. The input file can be separated by white space or any delimiter.

Syntax: $join [OPTION] FILE1 FILE2

Option	Description
-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.
-j *FIELD*	Equivalent to "**-1** *FIELD* **-2** *FIELD*".
-o *FORMAT*	Obey *FORMAT* while constructing output line.
-t *CHAR*	Use *CHAR* as input and output field separator.
-v *FILENUM*	Like **-a** *FILENUM*, but suppress joined output lines.
-1 *FIELD*	Join on this *FIELD* of file **1**.
-2 *FIELD*	Join on this *FIELD* of file **2**.
--check-order	Check that the input is correctly sorted, even if all input lines are pairable.
--nocheck-order	Do not check that the input is correctly sorted.
--header	Treat the first line in each file as field headers, print them without trying to pair them.
--help	Display a help message and exit.
--version	Display version information and exit.

Tools for Sorting

1. Sort

SORT command is used to sort a file, arranging the records in a particular order. By default, the sort command sorts file assuming the contents are ASCII. Using options in sort command, it can also be used to sort numerically.

1. SORT command sorts the contents of a text file, line by line.
2. SORT is a standard command line program that prints the lines of its input or concatenation of all files listed in its argument list in sorted order.
3. The SORT command is a command line utility for sorting lines of text files. It supports sorting alphabetically, in reverse order, by number, by month and can also remove duplicates.
4. The SORT command can also sort by items not at the beginning of the line, ignore case sensitivity and return whether a file is sorted or not. Sorting is done based on one or more sort keys extracted from each line of input.
5. By default, the entire input is taken as sort key. Blank space is the default field separator.

The sort command follows these features as stated below:

1. Lines starting with a number will appear before lines starting with a letter.
2. Lines starting with a letter that appears earlier in the alphabet will appear before lines starting with a letter that appears later in the alphabet.
3. Lines starting with a lowercase letter will appear before lines starting with the same letter in uppercase.

Syntax: uniq [OPTION]... [INPUT [OUTPUT]]

Option	Description
-b	Ignores leading blanks.
-d	Considers only blanks and alphanumeric characters.
-f	Fold lower case to upper case characters.
-g	Compares according to general numerical value.
-i	Considers only printable characters.
-M	Compares (unknown) < 'JAN' < ... < 'DEC'.
-h	Compare human readable numbers (e.g., 2K 1G).
-n	Compares according to string numerical value.

-R	Shuffles, but groups identical keys.
-r	Reverses the result of comparisons.

2. UNIQ

The uniq command in Linux is a command line utility that reports or filters out the repeated lines in a file.

In simple words, uniq is the tool that helps to detect the adjacent duplicate lines and also deletes the duplicate lines. uniq filters out the adjacent matching lines from the input file(that is required as an argument) and writes the filtered data to the output file.

Syntax : $uniq [OPTION] [INPUT[OUTPUT]]

The syntax of this is quite easy to understand. Here, INPUT refers to the input file in which repeated lines need to be filtered out and if INPUT isn't specified then uniq reads from the standard input. OUTPUT refers to the output file in which you can store the filtered output generated by uniq command and as in case of INPUT if OUTPUT isn't specified then uniq writes to the standard output.

Option	Description
-c, **--count**	Prefix lines with a number representing how many times they occurred.
-d, **--repeated**	Only print duplicated lines.
-D, **--all-repeated**[=*delimit-method*]	Print all duplicate lines. *delimit-method* may be one of the following: **none** — Do not delimit duplicate lines at all. This is the default. **prepend** — Insert a blank line before each set of duplicated lines. **separate** — Insert a blank line between each set of duplicated lines. The **-D** option is the same as specifying **--all-repeated=none**.
-f *N*, **--skip-fields=***N*	Avoid comparing the first *N* fields of a line before determining uniqueness. A field is a group of characters, delimited by whitespace. This option is useful, for instance, if your document's lines are numbered, and you want to compare everything in the line except the line number. If the option **-f 1** were specified, the adjacent lines **1 This is a line.** **2 This is a line.** would be considered identical. If no **-f** options were specified, they would be considered unique.
-i, **--ignore-case**	Normally, comparisons are case-sensitive. This option performs case-insensitive comparisons instead.
-s *N*, **--skip-chars=***N*	Avoid comparing the first *N* characters of each line when determining uniqueness. This is like the **-f** option, but it skips individual characters rather than fields.
-u, **--unique**	Only print unique lines.
-z, **--zero-terminated**	End lines with 0 byte (NULL), instead of a newline.
-w, **--check-chars=***N*	Compare no more than *N* characters in lines.
--help	Display a help message and exit.
--version	Output version information and exit.

Comparing Files

1. cmp

cmp command in Linux/UNIX is used to compare the two files byte by byte and helps you to find out whether the two files are identical or not.

When cmp is used for comparison between two files, it reports the location of the first mismatch to the screen if difference is found and if no difference is found i.e the files compared are identical.

cmp displays no message and simply returns the prompt if the the files compared are identical.

Syntax: cmp [OPTION]... FILE1 [FILE2 [SKIP1 [SKIP2]]]

The optional SKIP1 and SKIP2 specify the number of bytes to skip at the beginning of each file (zero by default).

The syntax of cmp command is quite simple to understand. If we are comparing two files then obviously we will need their names as arguments (i.e as FILE1 & FILE2 in syntax). In addition to this, the optional SKIP1 and SKIP2 specify the number of bytes to skip at the beginning of each file which is zero by default and OPTION refers to the options compatible with this command about which we will discuss later on.

SKIP values may be followed by the following multiplicative suffixes:

kB	kilobytes	1000
K	kibibytes	1024
MB	megabytes	1,000,000
M	mebibytes	1,048,576
GB	gigabytes	1,000,000,000
G	gibibytes	1,073,741,824

Option	Description
-b, **--print-bytes**	Print differing bytes.
-i, **--ignore-initial=***SKIP*	Skip first *SKIP* bytes of both files.
-i, **--ignore-initial=***SKIP1***:***SKIP2*	Skip first *SKIP1* bytes of *FILE1* and first *SKIP2* bytes of *FILE2*.
-l, **--verbose**	Output byte numbers and differing byte values.
-n, **--bytes**=*LIMIT*	Compare at most *LIMIT* bytes.
-s, **--quiet**, **--silent**	Suppress all normal output.
--help	Display a help message and exit.
-v, **--version**	Output version information and exit.

2. Comm

comm compare two sorted files line by line and write to standard output; the lines that are common and the lines that are unique.

Suppose you have two lists of people and you are asked to find out the names available in one and not in the other or even those common to both. Comm is the command that will help you to achieve this. It requires two sorted files which it compares line by line.

Syntax : $comm [OPTION]... FILE1 FILE2

- As using comm, we are trying to compare two files therefore the syntax of comm command needs two filenames as arguments.
- With no OPTION used, comm produces three-column output where first column contains lines unique to FILE1, second column contains lines unique to FILE2 and third and last column contains lines common to both the files.

- comm command only works right if you are comparing two files which are already sorted.

Option	Description
-1	suppress column 1 (lines unique to **FILE1**)
-2	suppress column 2 (lines unique to **FILE2**)
-3	suppress column 3 (lines that appear in both files)
--check-order	check that the input is correctly sorted, even if all input lines are pairable
--nocheck -order	do not check that the input is correctly sorted
--output -delimiter=*STR*	separate columns with string *STR*
--help	display a help message, and exit.
--version	output version information, and exit.

3. Diff

Diff stands for difference. This command is used to display the differences in the files by comparing the files line by line. Unlike its fellow members, cmp and comm, it tells us which lines in one file have is to be changed to make the two files identical.

The important thing to remember is that diff uses certain special symbols and instructions that are required to make two files identical. It tells you the instructions on how to change the first file to make it match the second file

Option	Description
c	Print the differing characters. Display control characters as a '^' followed by a letter of the alphabet and precede characters that have the high bit set with 'M-' (which stands for "meta").
--ignore-initial=BYTES	Ignore any differences in the the first BYTES bytes of the input files. Treat files with fewer than BYTES bytes as if they are empty.
-l	Print the (decimal) offsets and (octal) values of all differing bytes.
--print-chars	Print the differing characters. Display control characters as a '^' followed by a letter of the alphabet and precede characters that have the high bit set with 'M-' (which stands for "meta").
--quiet -s --silent	Do not print anything; only return an exit status indicating whether the files differ.
--verbose	Print the (decimal) offsets and (octal) values of all differing bytes.
-v --version	Output the version number of 'cmp'. The file name '-' is always the standard input. 'cmp' also uses the standard input if one file name is omitted. An exit status of 0 means no differences were found, 1 means some differences were found, and 2 means trouble.

Changing Information in Files

1. Tr

The tr command in UNIX is a command line utility for translating or deleting characters. It supports a range of transformations including uppercase to lowercase, squeezing repeating characters, deleting specific characters and basic find and replace. It can be used with UNIX pipes to support more complex translation. tr stands for translate.

Syntax :$ tr [OPTION] SET1 [SET2]

Option	Description

-C	Complement the set of characters in *string1*, that is "**-C ab**" includes every character except for '**a**' and '**b**'.
-c	Same as **-C** but complement the set of values in *string1*.
-d	Delete characters in *string1* from the input.
-s	Squeeze multiple occurrences of the characters listed in the last operand (either *string1* or *string2*) in the input into a single instance of the character. This occurs after all deletion and translation is completed.
-u	Guarantee that any output is unbuffered.

2. SED

SED command in UNIX is stands for stream editor and it can perform lot's of function on file like, searching, find and replace, insertion or deletion. Though most common use of SED command in UNIX is for substitution or for find and replace. By using SED you can edit files even without opening it, which is much quicker way to find and replace something in file, than first opening that file in VI Editor and then changing it.

- SED is a powerful text stream editor. Can do insertion, deletion, search and replace (substitution).
- SED command in UNIX supports regular expression which allows it perform complex pattern matching.

Syntax: SED OPTIONS... [SCRIPT] [INPUTFILE...]

Option	Description
-n, --quiet, --silent	Suppress automatic printing of pattern space.
-e script, --expression=script	Add the script script to the commands to be executed.
-f script-file, --file=script-file	Add the contents of script-file to the commands to be executed.
--follow-symlinks	Follow symlinks when processing in place.
-i[SUFFIX], --in-place[=SUFFIX]	Edit files in place (this makes a backup with file extension SUFFIX, if SUFFIX is supplied).
-l N, --line-length=N	Specify the desired line-wrap length, N, for the "l" command.
--POSIX	Disable all GNU extensions.
-r, --regexp-extended	Use extended regular expressions in the script.
-s, --separate	Consider files as separate rather than as a single continuous long stream.
-u, --unbuffered	Load minimal amounts of data from the input files and flush the output buffers more often.
--help	Display a help message, and exit.
--version	Output version information and exit.

Examining File Contents

1. od

od command in Linux is used to convert the content of input in different formats with octal format as the default format. This command is especially useful when debugging Linux scripts for unwanted changes or characters. If more than one file is specified, od command concatenates them in the listed order to form the input.It can display output in a variety of other formats, including hexadecimal, decimal, and ASCII. It

is useful for visualizing data that is not in a human-readable format, like the executable code of a program.

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

Tag	Description
-A, **--address-radix**=*RADIX*	decide how file offsets are printed
-j, **--skip-bytes**=*BYTES*	skip BYTES input bytes first
-N, **--read-bytes**=*BYTES*	limit dump to BYTES input bytes
-S, **--strings**[=*BYTES*]	output strings of at least BYTES graphic chars
-t, **--format**=*TYPE*	select output format or formats
-v, **--output-duplicates**	do not use * to mark line suppression
-w, **--width**[=*BYTES*]	output BYTES bytes per output line
--traditional	accept arguments in traditional form
--help	display this help and exit
--version	output version information and exit

Traditional format specifications may be intermixed; they accumulate:

Tag	Description
-a	same as **-t** a, select named characters
-b	same as **-t** o1, select octal bytes
-c	same as **-t** c, select ASCII characters or backslash escapes
-d	same as **-t** u2, select unsigned decimal 2-byte units
-f	same as **-t** fF, select floats
-i	same as **-t** dI, select decimal ints
-l	same as **-t** dL, select decimal longs
-o	same as **-t** o2, select octal 2-byte units
-s	same as **-t** d2, select decimal 2-byte units
-x	same as **-t** x2, select hexadecimal 2-byte units
If first and second call formats both apply, the second format is assumed if the last operand begins with + or (if there are 2 operands) a digit. An OFFSET operand means **-j** OFFSET. LABEL is the pseudo-address at first byte printed, incremented when dump is progressing. For OFFSET and LABEL, a 0x or 0X prefix indicates hexadecimal; suffixes may be . for octal and b for multiply by 512.	
TYPE is made up of one or more of these specifications:	
a	named character
c	ASCII character or backslash escape
d[SIZE]	signed decimal, SIZE bytes per integer
f[SIZE]	floating point, SIZE bytes per integer
o[SIZE]	octal, SIZE bytes per integer
u[SIZE]	unsigned decimal, SIZE bytes per integer
x[SIZE]	hexadecimal, SIZE bytes per integer

Tool for mathematical Calculation

1. bc

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.

Syntax: bc [-hlwsqv] [long-options] [file ...]

Tag	Description
--help	Display help
-h, --help	Print the usage and exit.
file	A file containing the calculations/functions to perform. This can be piped from standard input.
-i, --interactive	Force interactive mode.
-l, --mathlib	Define the standard math library.
-w, --warn	Give warnings for extensions to POSIX bc.
-s, --standard	Process exactly the POSIX bc language.
-q, --quiet	Do not print the normal GNU bc welcome.
-v, --version	Print the version number and copyright and quit.

2. Factor

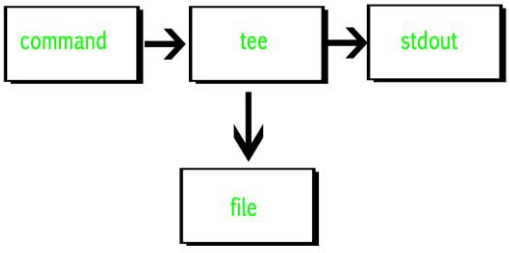
The factor command in Linux is used to print the prime factors of the given numbers, either given from command line or read from standard input. The numbers given through standard input may be delimited by tabs, spaces or newlines.

Syntax: factor [NUMBER]

Monitoring input and output

1. Tee

tee command reads the standard input and writes it to both the standard output and one or more files. The command is named after the T-splitter used in plumbing. It basically breaks the output of a program so that it can be both displayed and saved in a file. It does both the tasks simultaneously, copies the result into the specified files or variables and also display the result.



Syntax: tee [OPTION]... [FILE]...

Option	Description
-a, **--append**	Append to the given FILEs. Do not overwrite.
-i, **--ignore-interrupts**	Ignore interrupt signals.

2. Script

script command in Linux is used to make typescript or record all the terminal activities. After executing the *script* command it starts recording everything printed on the screen including the inputs and outputs until exit. By default, all the terminal

information is saved in the file *typescript* , if no argument is given. *script* is mostly used when we want to capture the output of a command or a set of command while installing a program or the logs generated on the terminal while compiling an opensource codes, etc. *script* command uses *two files* i.e. one for the terminal output and other for the timing information.

Syntax: script [options] [file]

Option	Description
-a, --append	Append the output to file or typescript, retaining the prior contents.
-c, --command co mmand	Run the command rather than an interactive shell. This option makes it easy for script to capture the output of a program that behaves differently when its standard output is not a tty.
-e, --return	Return the exit code of the child process. Uses the same format as bash termination on signal termination: exit code is 128+n.
-f, --flush	Flush output after each write. This option is nice for telecooperation: one person does "mkfifo foo; script -f foo", and another can supervise real-time what is being done using "cat foo".
--force	Allow the default output destination, i.e. the typescript file, to be a hard or symbolic link. The command will follow a symbolic link.
-q, --quiet	Operate without displaying any output.
-t, --timing[=file]	Output timing data to standard error, or to file when given. This data contains two fields, separated by a space. The first field indicates how much time elapsed since the previous output. The second field indicates how many characters were output this time. This information can be used to replay typescripts with realistic typing and output delays.

Tool for displaying Data and Time

1. Cal

If a user wants a quick view of calendar in Linux terminal, cal is the command for you. By default, cal command shows current month calendar as output. cal command is a calendar command in Linux which is used to see the calendar of a specific month or a whole year.

Syntax: cal [month] [year] [-m month] [-y year] [-h] [-3] [-1] [-A num] [-B num] [-d YYYY-MM] [-j] [-N]

Option	Description
-h	Don't highlight today's date.
-m month	Specify a month to display. The month specifier can be a full month name (e.g., Februiary), a month abbreviation of at least three letters (e.g., Feb), or a number (e.g., 2). If you specify a number, followed by the letter f or p, the month of the following or previous year, respectively, will be displayed. For instance, -m 2f displays February of next year.
-y year	Specify a year to display. For example, -y 1970 displays the entire calendar of the year 1970.
-3	Display last month, this month, and next month.
-1	Display only this month. This is the default.
-A num	Display num months occurring after any months already specified. For example, -

	3 -A 3 displays last month, this month, and four months after this one; and -y 1970 -A 2 displays every month in 1970, and the first two months of 1971.
-B num	Display num months occurring before any months already specified. For example, -3 -B 2 displays the previous three months, this month, and next month.
-d YYYY-MM	Operate as if the current month is number MM of year YYYY.
-j	Display a Julian calendar, instead of the default Gregorian calendar. All days are numbered from January 1, rather than from the beginning of the month.
-N	Behave as if you ran ncal, using its options and display output.

2. DATE

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

Syntax: date [-u|--utc|--universal] [MMDDhhmm[[CC]YY][.ss]]

Option	Description
-d, **--date=***STRING*	Display time described by string *STRING*, as opposed to the default, which is '**now**'.
-f, **--file=***DATEFILE*	Like **--date**, but processed once for each line of file *DATEFILE*.
-I[*TIMESPEC*], **--iso-8601**[=*TIMESPEC*]	Output date/time in ISO 8601 format. For values of *TIMESPEC*, use '**date**' for date only (the default), '**hours**', '**minutes**', '**seconds**', or '**ns**' for date and time to the indicated precision.
-r, **--reference=***FILE*	Display the last modification time of file *FILE*.
-R, **--rfc-2822**	Output date and time in RFC 2822 format. Example: **Mon, 07 Aug 2006 12:34:56 -0600**
--rfc-3339=*TIMESPEC*	Output date and time in RFC 3339 format. *TIMESPEC* can be set to '**date**', '**seconds**', or '**ns**' for date and time to the indicated precision. Date and time components are separated by a single space, for example: **2006-08-07 12:34:56-06:00**
-s, **--set=***STRING*	Set time described by string *STRING*.
-u, **--utc**, **--universal**	Print or set Coordinated Universal Time.
--help	Display a help message and exit.
--version	Display version information and exit.

Date Format

FORMAT is a sequence of characters which specifies how output will appear. It comprises some combination of the following sequences:

%%	A literal percent sign ("**%**").
%a	The abbreviated weekday name (e.g., **Sun**).
%A	The full weekday name (e.g., **Sunday**).
%b	The abbreviated month name (e.g., **Jan**).
%B	Locale's full month name (e.g., **January**).
%c	The date and time (e.g., **Thu Mar 3 23:05:25 2005**).

%C	The current century; like **%Y**, except omit last two digits (e.g., **20**).
%d	Day of month (e.g., **01**).
%D	Date; same as **%m/%d/%y**.
%e	Day of month, space padded; same as **%_d**.
%F	Full date; same as **%Y-%m-%d**.
%g	Last two digits of year of ISO week number (see **%G**).
%G	Year of ISO week number (see **%V**); normally useful only with **%V**.
%h	Same as **%b**.
%H	Hour (**00**..**23**).
%I	Hour (**01**..**12**).
%j	Day of year (**001**..**366**).
%k	Hour, space padded (**0**..**23**); same as **%_H**.
%l	Hour, space padded (**1**..**12**); same as **%_I**.
%m	Month (**01**..**12**).
%M	Minute (**00**..**59**).
%n	A newline.
%N	Nanoseconds (**000000000**..**999999999**).
%p	Locale's equivalent of either **AM** or **PM**; blank if not known.
%P	Like **%p**, but lower case.
%r	Locale's 12-hour clock time (e.g., **11:11:04 PM**).
%R	24-hour hour and minute; same as **%H:%M**.
%s	Seconds since 1970-01-01 00:00:00 UTC.
%S	Second (**00**..**60**).
%t	A tab.
%T	Time; same as **%H:%M:%S**.
%u	Day of week (**1**..**7**); 1 is **Monday**.
%U	Week number of year, with Sunday as first day of week (**00**..**53**).
%V	ISO week number, with Monday as first day of week (**01**..**53**).
%w	Day of week (**0**..**6**); 0 is **Sunday**.
%W	Week number of year, with Monday as first day of week (**00**..**53**).
%x	Locale's date representation (e.g., **12/31/99**).
%X	Locale's time representation (e.g., **23:13:48**).
%y	Last two digits of year (**00**..**99**).
%Y	Year.
%z	+hhmm numeric time zone (e.g., **-0400**).
%:z	+hh:mm numeric time zone (e.g., **-04:00**).
%::z	+hh:mm:ss numeric time zone (e.g., **-04:00:00**).
%:::z	Numeric time zone with "**:**" to necessary precision (e.g., **-04**, **+05:30**).
%Z	Alphabetic time zone abbreviation (e.g., EDT).

By default, date pads numeric fields with zeroes. The following optional flags may follow '**%**':

-	(Hyphen) do not pad the field.
_	Pad with spaces.
0	Pad with zeros.
^	Use upper case if possible.
#	Use opposite case if possible.

After any flags comes an optional field width, as a decimal number; then an optional modifier, which is either **E** to use the locale's alternate representations if available, or **O** to use the locale's alternate numeric symbols if available.

Communications Commands

1. Wall

wall displays the contents of file or, by default, its standard input, on the terminals of all currently logged-in users. The command will cut any lines that are over 79 characters to new lines. Short lines are white space padded to have 79 characters. The command will always put carriage return and new line at the end of each line.

Only the super-user can write on the terminals of users who have chosen to deny messages or are using a program which automatically denies messages.

Reading from a file is refused when the invoker is not super-user and the program is suid(set-user-ID) or sgid(set-group-ID).

Syntax: wall [-n] [-t timeout] [message | file]

Option	Description
-n	This option will suppress the banner.
-t	This option will abandon the write attempt to the terminals after timeout seconds. This timeout needs to be a positive integer. The by default value is 300 seconds, which is a legacy from the time when peoples ran terminals over modem lines.
-V	This option display version information and exit.
-h	This option will display help message and exit

2. Write

The write utility allows you to communicate with other users by copying lines from your terminal to theirs. When you run the write command, the user you are writing to gets a message of the format:

Message from yourname@yourhost on yourtty at hh:mm ...

Any further lines you enter will be copied to the specified user's terminal. If the other user wants to reply, they must run write as well. When you are done, type an end-of-file or interrupt character. The other user will see the message 'EOF' indicating that the conversation is over.

You can prevent people (other than the super-user) from writing to you with the mesg command.

If the user you want to write to is logged in on more than one terminal, you can specify which terminal to write to by specifying the terminal name as the second operand to the write command. Alternatively, you can let write select one of the terminals and it will pick the one with the shortest idle time. So, if the user is logged in at work and also dialed up from home, the message will go to the right place.

The traditional protocol for writing to someone is that the string '-o', either at the end of a line or on a line by itself, means that it is the other person's turn to talk. The string 'oo' means that the person believes the conversation to be over.

Syntax: write user [tty]

Option	Description
user	The user to write to.
tty	The specific terminal to write to, if the user is logged in to more than one session.

3. mail

The Mail command in unix or linux system is used to send emails to the users, to read the received emails, to delete the emails etc. Mail command will come in handy especially when writing automated scripts. For example, you have written an automated script for taking weekly backup of oracle database. How to know the status of backup, whether it is succeeded or not? In this case, sending an email from the automated script at the end of the backup will be helpful in knowing the status.

Syntax : mail [options] to-address [-- sendmail-options]

Option	Description
-v	Verbose mode. The details of delivery are displayed on the user's terminal.
-i	Ignore tty interrupt signals. This is particularly useful when using mail on noisy phone lines.
-I	Forces mail to run in interactive mode even when input isn't a terminal. In particular, the '~' special character when sending mail is only active in interactive mode.
-n	Inhibits reading /etc/mail.rc upon startup.
-N	Inhibits the initial display of message headers when reading mail or editing a mail folder.
-s	Specify subject on command line (only the first argument after the -s flag is used as a subject; be careful to quote subjects containing spaces.)
-c	Send carbon copies to list of users.
-b	Send blind carbon copies to list. List should be a comma-separated list of names.
-f	Read in the contents of your mbox (or the specified file) for processing; when you quit, mail writes undeleted messages back to this file.
-u	Is equivalent to: mail -f /var/spool/mail/user

4. News

- The command will help to see the message to user by administrator, even when they are not online.
- The message will be stored in /usr/news.

5. Finger

- When command is used without any arguments it will display list of all logged users .
- Similar to who command, but output will be different, in finger it will display user detail also
- In output first field will be user login name, second field show the user's full name, the third field show the terminal of the user, fourth field will show the idle time (time since they are doing any work), and The last field shows office address.
- The command can display single user detail if user name give as argument, which is not possible in who command.

6. Mesg

- Users can disallow anyone to send them messages via the mesg utility. Therefore, before you start attempting to send messages, it's a good idea to check whether

messages are allowed. For yourself, you can simply enter the mesg command as follows:

- The mesg utility is invoked by a users to control write access others have to the terminal device associated with the standard error output.
- Traditionally, write access is allowed by default. However, as users become more conscious of various security risks, there is a trend to remove write access by default, at least for the primary login shell. To make sure your ttys are set the way you want them to be set, mesg should be executed in your login scripts.

Options	Description
n	Disallows messages
y	Permits messages to be displayed
If no arguments are given, mesg displays the present message status to the standard error output. The mesg utility exits with one of the following values:	
\0	Messages are allowed.
\1	Messages are not allowed.
1	An error has occurred.

7. 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. Fast ping low latency means faster connection. Ping uses **ICMP(Internet Control Message Protocol)** to send an **ICMP echo message** to the specified host if that host is available then it sends **ICMP reply message**. Ping is generally measured in millisecond every modern operating system has this ping pre-installed.

The basic syntax of the ping command contains ping pursued by the hostname, a website name, or exact IP address.

Syntax: **ping [options] hostname or IP address**

So, we can type inside our terminal for checking whether the remote host is up:

- **from:** It tells the target and its IP address.
- **Important:** The IP address might be different for any website depending on our geographical location.
- **ttl=52:** It tells the value, i.e., Time to Live from 1-255. Also, it indicates network number hops a packet could take before any router removes it.
- **icmp_seq=1:** It tells the all ICMP packet's sequence number. It increases by a single number for all subsequent echo requests.
- **time=7.68 ms:** It tells the Time that it took any packet for reaching the target and come back to the origin. It expressed in ms (milliseconds).

ping "localhost" for checking Local Network

If we find issues reaching a remote machine or a website, we can ping the localhost to ensure that we have a network connection. We can use anyone of the following ways for checking the interface of the local network:

- **ping 0:** It is one of the quickest option to ping a localhost. The terminal will resolve determine the IP address and gives a response once we enter this command.

- **ping localhost:** We can use the ping localhost name. This name will refer to our system and when we enter this command, we will say **"ping this system"**.
- **ping 127.0.0.1:** A few people prefer entering the IP address to ping the localhost.

The following are some generally used ping commands:

Option	Description
a	It produces a sound if the peer could be reached.
b	It permits ping the IP address of a broadcast.
B	It prevents the ping from changing the probe source address.
c	It limits the number of transferred ping requests.
d	It sets an option, i.e., SO DEBUG over the used socket.
f	It floods the network by transferring several packets per second.
i	It describes the interval among the successive transmission of the packet. One second is the default value.
I	It sets the IP address of the source to the described IP address of the interface. This option is needed if pinging the lack address of IPv6 link. We can use the name of the device or IP address.
l	It specifies several packets to transfer without delaying a response.
q	It shows IP addresses in the output of the ping instead of hostnames.
T	It fixes the Time To Live.
v	It gives verbose output.
V	It shows the version of the ping and exits to a newer command prompt line

Process Related Commands

Ps

Linux/unix is a multitasking and multi-user systems. So, it allows multiple processes to operate simultaneously without interfering with each other. Process is one of the important fundamental concept of the Linux OS. A process is an executing instance of a program and carries out different tasks within the operating system.

Linux/unix provides us a utility called **ps** for viewing information related with the processes on a system which stands as abbreviation for **"Process Status".** ps command is used to list the currently running processes and their PIDs along with some other information depends on different options. It reads the process information from the virtual files in **/proc** file-system. /proc contains virtual files, this is the reason it's referred as a virtual file system.

Stntax : ps [options]

```
[root@root ~]# ps
  PID TTY        TIME CMD
12330 pts/0    00:00:00 bash
21621 pts/0    00:00:00 ps
```

Result contains four columns of information. Where,

- PID – the unique process ID
- TTY – terminal type that the user is logged into
- TIME – amount of CPU in minutes and seconds that the process has been running
- CMD – name of the command that launched the process.

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

Command to run process in background

1. NICE

Nice command lets you run a command at a priority lower than the command's normal priority. The nice command Runs a command at a lower or higher priority. The Command parameter is the name of any executable file on the system. If you do not specify an Increment value the nice command defaults to an increment of 10. You must have root user authority to run a command at a higher priority. The priority of a process is often called its nice value. The nice value can range from -20(most favorable scheduling) to 19 (least favorable), with 19 being the lowest priority.

For example, if a command normally runs at a priority of 10, specifying an increment of 5 runs the command at a lower priority, 15, and the command runs slower. The nice command does not return an error message if you attempt to increase a command's priority without the appropriate authority. Instead, the command's priority is not changed, and the system starts the command as it normally would.

The nice value is used by the system to calculate the current priority of a running process. Use the ps command with the -l flag to view a command's nice value. The nice value appears under the NI heading in the ps command output.

Syntax : nice [- Increment| -n Increment]

Option	Description
-Increment	Increments a command's priority up or down. You can specify a positive or negative number. Positive increment values reduce priority. Negative increment values increase priority. Only users with root authority can specify a negative increment. If you specify an increment value that would cause the nice value to exceed the range of -20 to 19, the nice value is set to the value of the limit that was exceeded. This flag is equivalent to the -n Increment flag.
-n Increment	This flag is equivalent to the -Increment flag.

2. KILL

The Kill command in Unix/Linux operating system is used to send a signal to the specified process or group. If we don't specify any signal, then the kill command passes the SIGTERM signal. We mostly use the kill command for terminating or killing a process. However we can also use the kill command for running a stopped process.

syntax

- kill [option]

Option	Description
-signal, -s signal	The name, abbreviated name, or number of the signal to be sent, preceded by a dash. For example, -SIGTERM, -TERM, or -15. To view a list of available signals, use the -l or -L options (see below), or refer to our list of Linux signals.
pid	A numeric process ID. If you're not sure what the PID is of a process, use the ps command to list it, e.g., ps -aux.
-l, --list[=signal]	List available signal names. With -l or --list, lists all signal names. With --list=signal, translates a number into its signal name.
-L, --table	List available signal names and numbers in a table.

All available UNIX signals have different names, and are mapped to certain numbers as described below:

Number	Name (short name)	Description	Used for
0	SIGNULL (NULL)	Null	Check access to pid
1	SIGHUP (HUP)	Hangup	Terminate; can be trapped
2	SIGINT (INT)	Interrupt	Terminate; can be trapped
3	SIGQUIT (QUIT)	Quit	Terminate with core dump; can be trapped
9	SIGKILL (KILL)	Kill	Forced termination; cannot be trapped
15	SIGTERM (TERM)	Terminate	Terminate; can be trapped
24	SIGSTOP (STOP)	Stop	Pause the process; cannot be trapped. This is default if signal not provided to kill command.
25	SIGTSTP (STP)	Terminal	Stop/pause the process; can be trapped
26	SIGCONT (CONT)	Continue	Run a stopped process

3. At / Batch

- at is a UNIX command which allows a script to be executed once at a specified time and/or date.
- at command should be followed by time and then date on which it is to be executed.
    - If no date is specified, today is assumed.
    - If no date is specified and the time is less than now, tomorrow is assumed.
    - If no month is specified, the current month is assumed.
    - If a month earlier than the current month is given, next year is assumed.
- Using CTRL-D will exit the prompt and offer you a job reference number. When an at job has run, a message will be written to the UNIX mailbox of the account name running the job.
- Your queued at jobs can be viewed using the atq command at the UNIX prompt. This will reveal the job reference number, owners and the date and time that jobs are due to run.

- To cancel a job use the command: atrm {job reference}at the UNIX prompt. You must have appropriate permissions to do this.
- When using these commands, the root user account has the privilege to view or cancel all at jobs .

4. Cron / Crontab

- On most systems, you must get permission from the system administrator before you can submit job requests to cron.
- On some shared systems, because there is only one crontab file, only the administrator has access to the crontab command.
- To schedule one-time only tasks with cron, at or batch command.
- For commands that need to be executed repeatedly (e.g., hourly, daily, or weekly), you can use the crontab command. The crontab command creates a crontab file containing commands and instructions for the cron to execute.
- Each entry in a crontab file consists of six fields, specifying in the following order: minute(s) hour(s) day(s) month(s) weekday(s) command(s)
- The fields are separated by spaces or tabs. The first five are integer patterns and the sixth is the command to execute. The following table briefly describes each of the fields:

Field	Value	Description
minute	0-59	The exact minute that the command sequence executes
hour	0-23	The hour of the day that the command sequence executes
day	1-31	The day of the month that the command sequence executes
month	1-12	The month of the year that the command sequence executes
weekday	0-6	The day of the week that the command sequence executes (Sunday = 0, Monday = 1, Tuesday = 2, and so forth)
command	Special	The complete sequence of commands to execute. The command string must conform to Bourne shell syntax. Commands, executables (such as scripts), or combinations are acceptable.

5. wait

Wait waits for the process identified by process ID pid (or the job specified by job ID jobid), and reports its termination status. If an ID is not given, wait waits for all currently active child processes, and the return status is zero. If the ID is a job specification, wait waits for all processes in the job's pipeline.

Syntax: wait [pid] [jobid]

Option	Description
pid	The unsigned decimal integer process ID of a command, for which the utility is to wait for the termination.
jobid	A job control job ID that identifies a background process group to wait for. The job control job ID notation is applicable only for invocations of wait in the current shell execution environment, and only on systems supporting the job control option.

6. Sleep

The sleep command pauses for an amount of time defined by NUMBER. SUFFIX may be "s" for seconds (the default), "m" for minutes, "h" for hours, or "d" for days.

Some implementations require that NUMBER be an integer, but modern Linux implementations allow NUMBER to also be a floating-point value. If more than one NUMBER is specified, sleep delays for the sum of their values.

Syntax: sleep NUMBER[SUFFIX]...

Where SUFFIX may be:

- s for seconds **(the default)**
- m for minutes.
- h for hours.
- d for days.

Example

sleep 10

7. TOP

The top command displays all the running process within the environment of your system. It helps in monitoring system usage and performances. It is mainly used to detect load on the server by system administrators.

The top command stands for table of processes. It is a task manager program, detected in several Unix-like operating systems, that shows information about memory and CPU utilization.

Overview of Top Command

The program generates an ordered list of active processes chosen by user-specified format and periodically updates it. Default ordering can be done by CPU usage, and the top CPU consumers are only shown. The top command displays how much memory and processing power is being utilized, as well as other details of the active processes.

A few top versions permit extensive customization of display, like sorting methods or choice of columns. The command is helpful for system administrators because it displays which processes and users are utilizing the most system resources at a time.

Implementations of Top Command

There are various different top versions available. The classical Unix version was specified by William LeFebvre and copyrighted in 1984 originally. It's hosted on SourceForge, and the 3.7 version was revealed in 2008. The Linux release of top is an element of the procps-ng tool group. Originally, it was specified by Roger Binns but after that taken over by others shortly. The roughly equivalent function is prstat on Solaris.

Microsoft Windows contains the graphical Task Manager utility and tasklist command. IBM AIX contains an updating active processes list as a component of the topas_nmon and topas commands.

In Linux, the load average numbers are known as the sum of the total processes waiting inside the run-queue plus the total count executing currently. The number is not relative but absolute. Hence, unlike utilization, it can be unbounded. The instant variations of the total processes are damped using an exponential decay formula calculated with fixed point math.

A program, i.e., ps, is the same as the top command but rather generates a process snapshot taken during invocation. The n (total iterations) option of the top command can generate a similar result, making the program execute the specified iterations and exit after showing its result.

Options of Top Command

Options	Description
-a	This option is used to solve the processes according to the allocated memory.

-b	It begins the top command in batch mode, which can be helpful in sending results from the top command to other files and programs.
-c	It begins the top command with the last remembered state reversed 'c'.
-d	It shows the delay between the screen updates and then overrides the associated value in the startup default or personal configuration file of one.
-h	It shows the usage prompt and library version, then quit.
-H	It begins a top command with the last remembered state reversed 'H'.
-i	It begins a top command with the last remembered state reversed i.
-m	It reports USED (rss process sum and swap count) rather than VIRT.
-M	It shows memory units and displays floating point values within the memory summary.
-n	It describes the maximum number of frames or iterations; the top command should generate before completion.
-p	It only monitors processes with process IDs.
-s	It is far better managed from the system configuration file.
-S	All processes are listed using the CPU time that it and its dead children have utilized when ***"Cumulative mode"*** is active.
-U	It only monitors processes with an effective username or UID matching that provided one. It matches saved, effective, real file system UIDs.
-u	It only monitors processes with an effective username or UID matching that provided one.
-v	It shows the usage prompt and library version, then quit.

Columns and Fields of Top Command

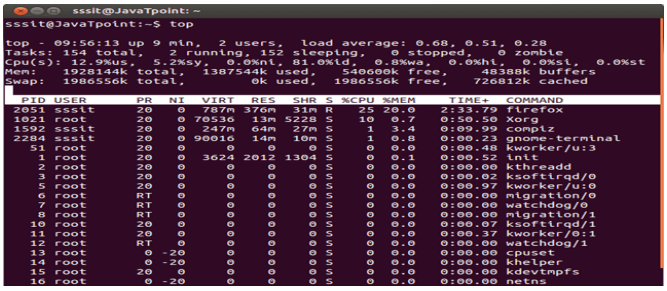
Some of the important fields or columns available in the top command are explained below:

- **PID:** It stands for Process Id or unique process Id of the task, which wraps periodically, never rebooting at zero.
- **RUSER:** It stands for the Real User Name of the task's owner.
- **PPID:** It stands for Parent Process Pid. It is the process ID of the parent of a task.
- **UID:** It is the effective user Id of the owner of the task.
- **USER:** It is the effective user name of the owner of the task.
- **GROUP:** It is the effective group name of the owner of the task.
- **TTY:** It is the controlling terminal name.
- **PR:** It shows the task's priority.
- **NI:** It is the task's nice value. A negative NI defines higher priority, and a positive NI defines lower priority.
- **P:** A number indicating the last utilized processor.
- **TIME:** It shows the CPU time that the task has utilized since it began.

Syntax:

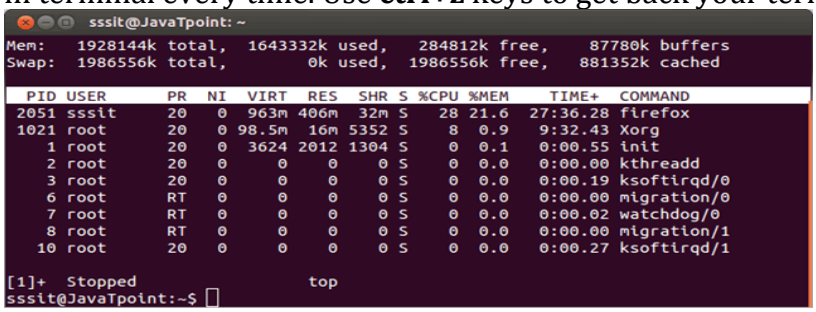
1. top

   Look at the above snapshot, its output is explained here,



Line1	**Line2**	**Line3**
• Time • how long system is running • how many users are logged in • and load average	• Total number of tasks • number of running tasks • number of sleeping tasks • number of stopped tasks • and number of zombie tasks	• It shows CPU usage in percentage for • users • system • low priority processes • idle processes • io wait • hardware interrupts • software interrupts • steal time
Line4	**Line5**	**Table explanation**
• It shows memory usage in kilobytes for • total memory • used memory • free memory • buffered memory	• It shows swap memory usage in kilobytes for • total memory • used memory • free memory • cached memory	• proces ID • user • priority • nice user • virtual memory • resident memory • shareable memory • CPU used percentage • memory used percentage • time a process has run • command

If you want you can **hide/show** these header lines by pressing some keys.
For example,
press **l** - to show/hide Line1. Top line
press **t** - to show/hide Line3. CPU information
press **m** - to show/hide Line4 and 5. Memory information
Keeping top command running in background
You can keep top command running in the background continuously without typing top in terminal every time. Use **ctrl+z** keys to get back your terminal.

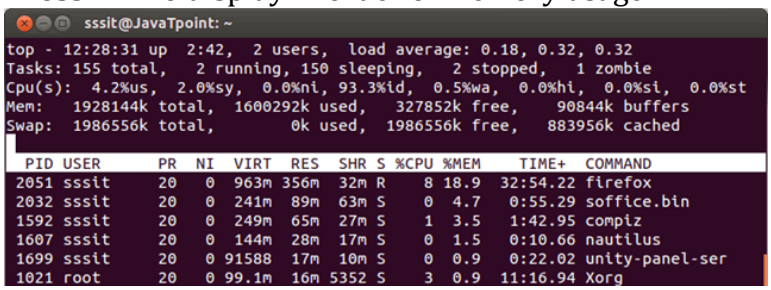


Look at the above snapshot, after pressing ctrl+z keys top command has stopped and we got our terminal back.To bring back top command in terminal type **fg** in terminal.

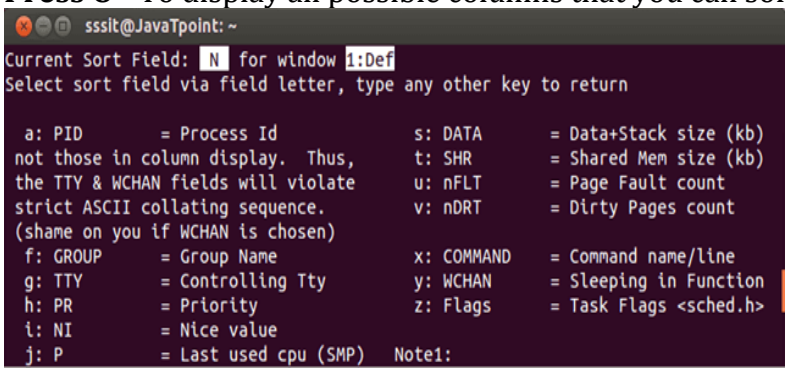
Sorting top output

By default, top command always display output in the order of CPU usage.

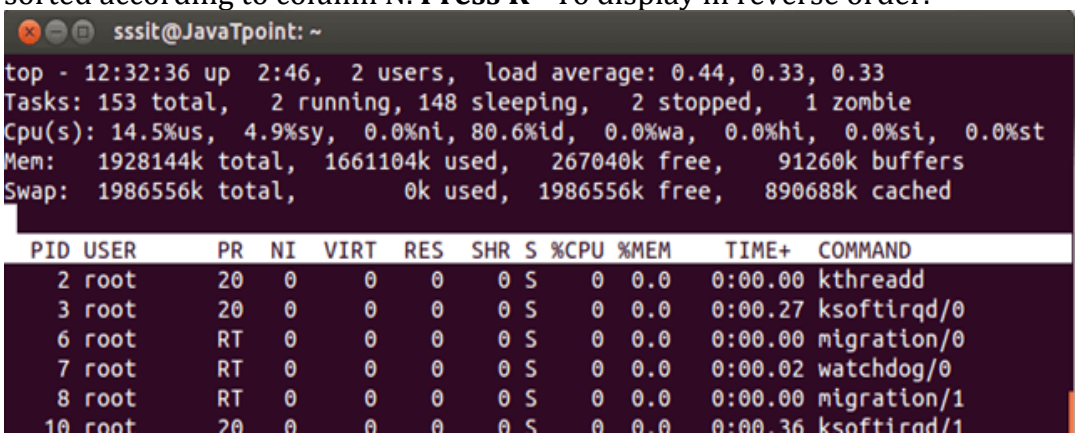
Press M - To display in order of memory usage.



Press O - To display all possible columns that you can sort.

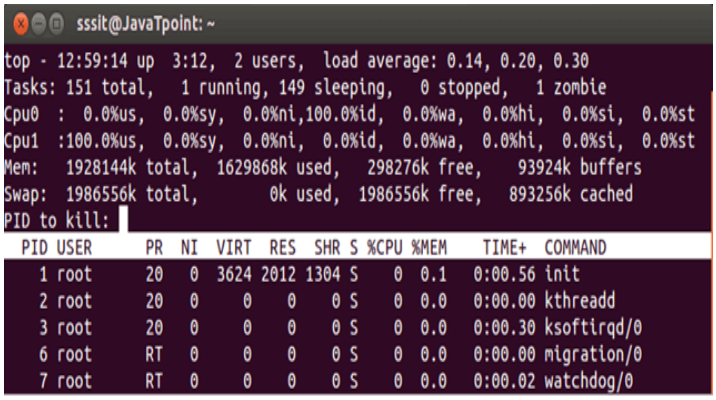


Look at the above snapshot, all the columns are assigned an alphabetic letter. To sort by column type the respective alphabet and output will be sorted according to that column. In the first line, current sort field is shown that is N which means currently it is sorted according to column N. **Press R** - To display in reverse order.



Killing a task without exiting from top

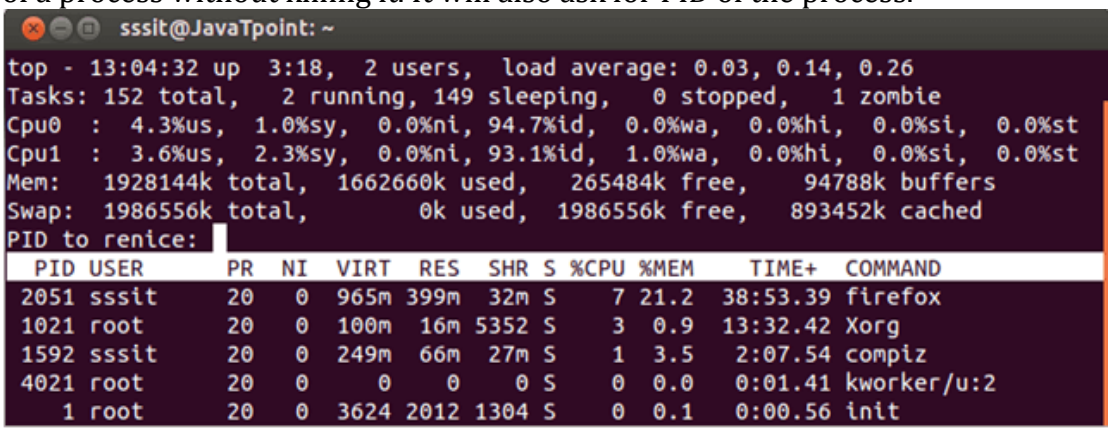
A task can be stopped without exiting from top command by pressing **k** key. It will ask for task's PID number, if you'll have authority to kill that task, then task will be removed. Otherwise, your command will fail.



Look at the above snapshot, after pressing k, we got a message asking for PID of task to be killed.

Renice a task

Renice is done to change the scheduling order. By pressing **r,** you can change the priority of a process without killing it. It will also ask for PID of the process.



Look at the above snapshot, after pressing r, we got a message asking for PID of task to be reniced.

Display processes for selected user

In top command output you can display all the processes for a particular user only by two options. One through command line and other without existing top.

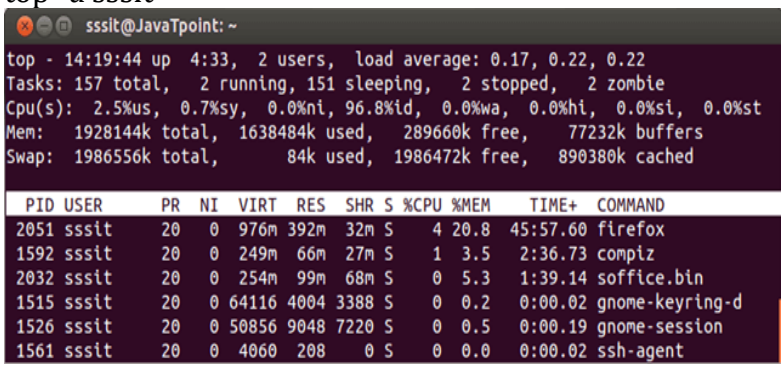
In command line, use the following command

Syntax:

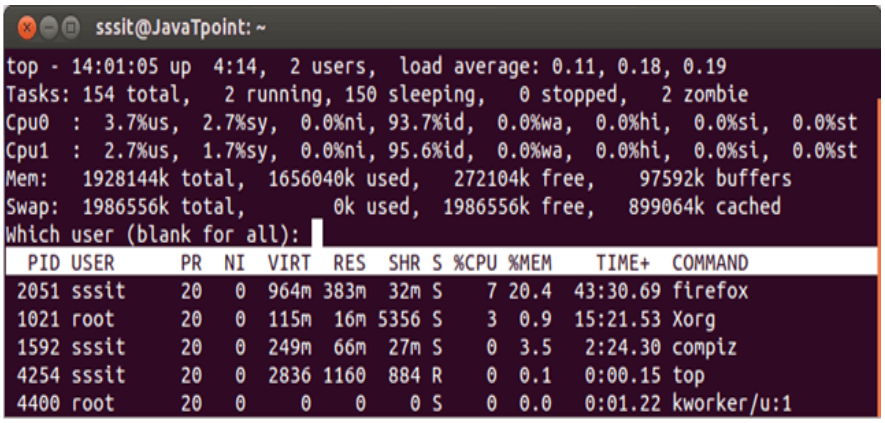
top -u **<userName>**

Example:

top -u sssit

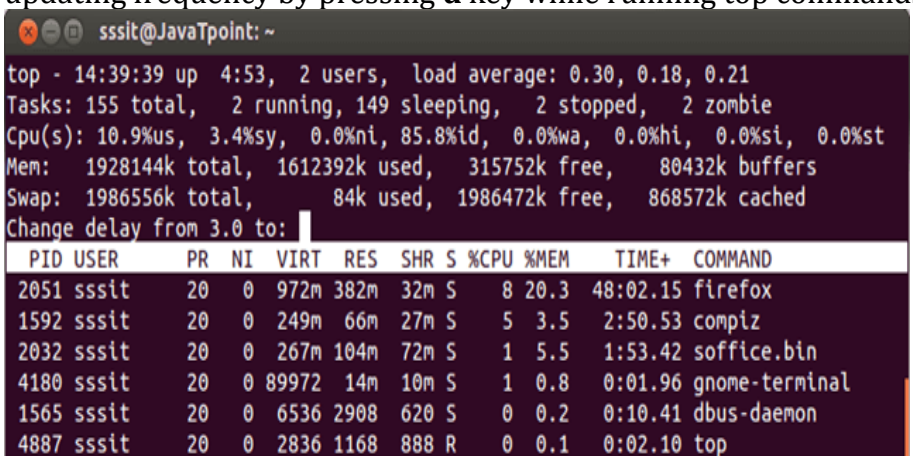


Look at the above snapshot, it displays all the processes only for user sssit. When top command is running, press u, it will ask for username. Type the username and press enter.



Look at the above snapshot, after pressing u, it is asking for username.

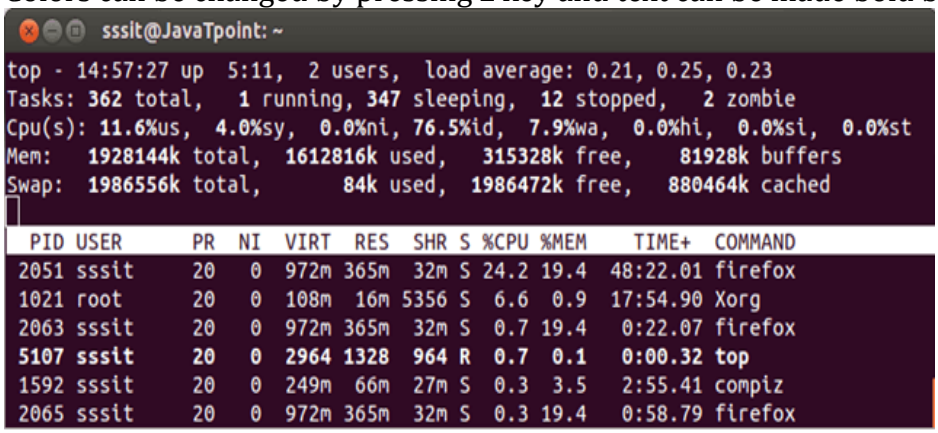
Updating top output By default, top output is updated after every 3 seconds. When you want to update it in between 3 seconds press **space bar.** You can also change updating frequency by pressing **d** key while running top command.



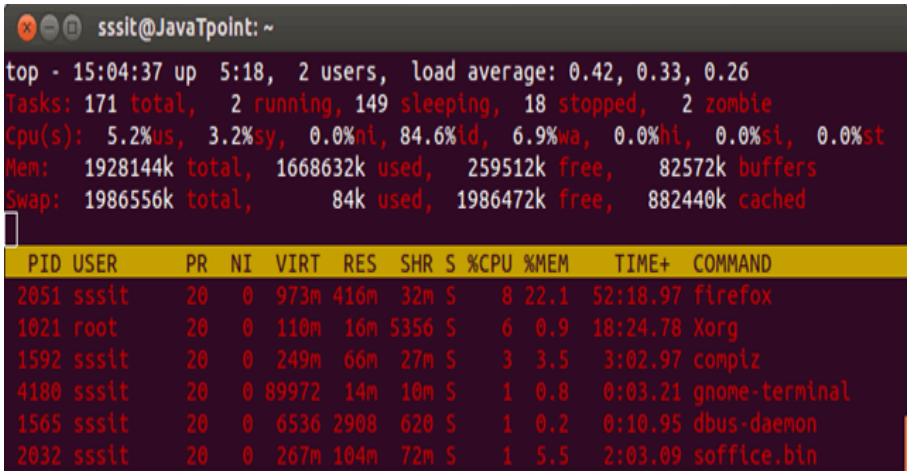
Look at the above snapshot, after pressing d key, it is asking for time for which it will be frequently updated.

Changing colors

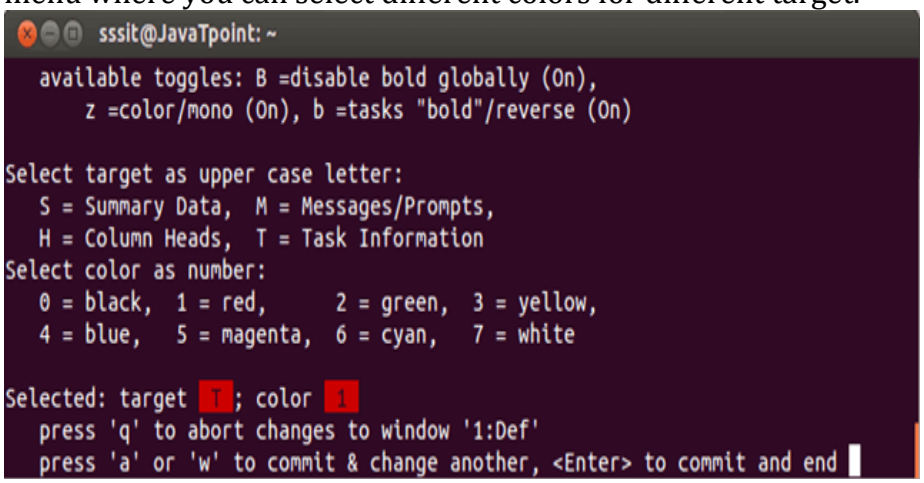
Colors can be changed by pressing **z** key and text can be made bold by pressing **b** key.



Look at the above snapshot, by pressing b all running processes are highlighted in white. To change color press **z** (small z) key.

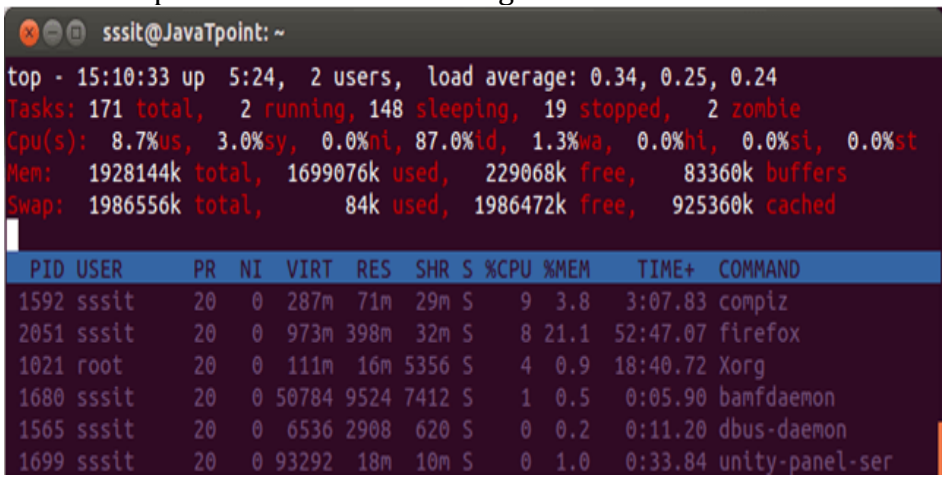


Look at the above snapshot, our output is colored after pressing z. Now, if you want to change the colors for different areas, press **Z** (capital Z). it will take you to the menu where you can select different colors for different target.



Suppose we want to apply blue color in column heading and magenta color in the task information.

Then we'll press 4 with H for heading and 5 with T for task information.



Look at the above snapshot, colors have been changed for their respective target.
Quitting after certain iterations The top command continuously displays output until you'll quit by pressing q. But you can define certain number of iterations after which top command will automatically quit from the terminal.

Syntax:

top -n **<number>**

Example:

top -n 2

With above example, it will show 2 iterations and exit automatical.

8. Jobs

Jobs command is used to list the jobs that you are running in the background and in the foreground. If the prompt is returned with no information no jobs are present. All shells are not capable of running this command. This command is only available in the csh, bash, tcsh, and ksh shells.

Syntax : jobs [JOB]

Options

- JOB Job name or number.
- **-l** Lists process IDs in addition to the normal information.
- **-n** List only processes that have changed status since the last notification.
- **-p** Lists process IDs only.
- **-r** Restrict output to running jobs.
- **-s** Restrict output to stopped jobs.

jobs command examples

To display the status of jobs in the current shell:

Command : $ jobs

Output:

```
[1]   7893 Running          gpass &
[2]   7904 Running          gnome-calculator &
[3]-  7955 Running          gedit fetch-stock-prices.py &
[4]+  7958 Stopped          ping cyberciti.biz
```

To display the process ID or jobs for the job whose name begins with "p,": Command

$ jobs -p %p OR **$ jobs %p**

Output:

```
[4]-  Stopped          ping cyberciti.biz
```

The character % introduces a job specification. In this example, you are using the string whose name begins with suspended command such as %ping.

Pass the -p option to jobs command to display PIDs only:

Command $ jobs -p

Output:

```
7895
7906
7910
7946
```

Pass the -r option to jobs command to display only running jobs only: Command : $ jobs -r

Output:

```
[1]   Running          gpass &
[2]   Running          gnome-calculator &
[3]-  Running          gedit fetch-stock-prices.py &
```

Concept of Mounting a File Systems

1. Mount Command

All files in a **Linux** filesystem are arranged in form of a big tree rooted at '**/**'.These files can be spread out on various devices based on your partition table, initially your parent directory is mounted(i.e attached) to this tree at '**/**', others can be mounted manually using GUI interface(if available) or using **mount** command. **mount** command is used to mount the filesystem found on a device to big tree structure (**Linux** filesystem) rooted at '**/**'. Conversely, another command **umount** can be used to detach these devices from the Tree.

Syntax:

- mount -t type device dir
- mount [-l|-h|-V]
- mount -a [-fFnrsvw] [-t fstype] [-O optlist]
- mount [-fnrsvw] [-o options] device|dir
- mount [-fnrsvw] [-t fstype] [-o options] device dir

This command tells the **Kernel** to attach the filesystem found at **device** to the **dir**. If you leave the **dir** part of syntax it looks for a **mount point** in **/etc/fstab**.

You can use **–source** or **–target** to avoid ambivalent interpretation.

- **mount --target /mountpoint**

/etc/fstab usually contains information about which device is need to be mounted where.

Most of the devices are indicated by files like **/dev/sda4**, etc. But it can be different for certain filesystems. Please refer below for more information.

- **man mount**

It is important to note that we are only discussing the standard form of **mount** command given as syntax. Different forms are somewhat discussed because it has certain limitations on different kernels.

Option	Description
l	Lists all the file systems mounted yet.
h	Displays options for command.
V	Displays the version information
a	Mounts all devices described at **/etc/fstab**
t	Type of filesystem device uses.
T	Describes an alternative fstab file.
r	Read-only mode mounted.

Concept of Demounting a File Systems

1. umount Command

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

- umount [-hV]
- umount -a [-dflnrv] [-t vfstype] [-O options]

- umount [-dflnrv] dir | device [...]

that a file system cannot be unmounted when it is 'busy' - for example, when there are open files on it, or when some process has its working directory there, or when a swap file on it is in use. The offending process could even be umount itself - it opens libc, and libc in its turn may open for example locale files.

Tag	Description
-V	Print version and exit.
-h	Print help message and exit.
-v	Verbose mode.
-n	Unmount without writing in /etc/mtab.
-r	In case unmounting fails, try to remount read-only.
-d	In case the unmounted device was a loop device, also free this loop device.
-i	Don't call the /sbin/umount.<filesystem> helper even if it exists. By default /sbin/umount.<filesystem> helper is called if one exists.
-a	All of the file systems described in /etc/mtab are unmounted. (With umount version 2.7 and later: the proc filesystem is not unmounted.)
-t vfstype	Indicate that the actions should only be taken on file systems of the specified type. More than one type may be specified in a comma separated list. The list of file system types can be prefixed with no to specify the file system types on which no action should be taken.
-O options	Indicate that the actions should only be taken on file systems with the specified options in /etc/fstab. More than one option type may be specified in a comma separated list. Each option can be prefixed with no to specify options for which no action should be taken.
-f	Force unmount (in case of an unreachable NFS system). (Requires kernel 2.1.116 or later.)
-l	Lazy unmount. Detach the filesystem from the filesystem hierarchy now, and cleanup all references to the filesystem as soon as it is not busy anymore. (Requires kernel 2.4.11 or later.)

UNIT – 4
Text Editing with VI Editor

Introduction

There are many ways to edit files in UNIX and for me one of the best ways is using screen-oriented text editor VI. This editor enables you to edit lines in context with other lines in the file. Now a day you would find an improved version of VI editor which is called VIM. Here VIM stands for Vi Improved.

The VI is generally considered the de facto standard in UNIX editors because:

- It's usually available on all the flavors of UNIX system.
- Its implementations are very similar across the board.
- It requires very few resources.
- It is more user friendly than any other editors like ed or ex.

You can use VI editor to edit an existing file or to create a new file from scratch. You can also use this editor to just read a text file.

Starting the vi Editor:

There are following way you can start using vi editor:

Command	Description
vi filename	Creates a new file if it already does not exist, otherwise opens existing file.
vi -R filename	Opens an existing file in read only mode.
view filename	Opens an existing file in read only mode.

The example 'vi testfile' to create a new file testfile if it already does not exist in the current working directory: As a result you would see a screen something like as follows:

You will notice a tilde (~) on each line following the cursor. A tilde represents an unused line. If a line does not begin with a tilde and appears to be blank, there is a space, tab, newline, or some other non-viewable character present.

So now you have opened one file to start with. Before proceeding further let us understanding few minor but important concepts explained below.

Operation Modes:

While working with VI editor you would come across following two modes:

- **Command mode**: This mode enables you to perform administrative tasks such as saving files, executing commands, moving the cursor, cutting (yanking) and pasting lines or words, and finding and replacing. In this mode, whatever you type is interpreted as a command.

- **Insert mode:** This mode enables you to insert text into the file. Everything that's typed in this mode is interpreted as input and finally it is put in the file.

The VI always starts in command mode. To enter text, you must be in insert mode. To come in insert mode, you simply type 'i'. To get out of insert mode, press the **Esc key**, which will put you back into command mode.

If you are not sure which mode, you are in, press the Esc key twice, and then you'll be in command mode. You open a file using VI editor and start type some characters and then come in command mode to understand the difference.

Getting Out of VI:

- The command to quit out of vi is :q. Once in command mode, type colon, and 'q', followed by return (Enter key).
- If your file has been modified in any way, the editor will warn you of this, and not let you quit. To ignore this message, the command to quit out of vi without saving is : **q!**. This lets you exit vi without saving any of the changes.
- The command to save the contents of the editor is **:w**. You can combine the above command with the quit command, or **:wq** and return.
- The easiest way to save your changes and exit out of vi is the **ZZ** command. When you are in command mode, type ZZ and it will do the equivalent of: wq**.**
- You can specify a different file name to save to by specifying the name after the**:w**. For example, if you wanted to save the file you were working as another filename called filename2, you would type :w filename2 and return. Try it once.

Moving within a File:

To move around within a file without affecting your text, you must be in command mode (press Esc twice). Here are some of the commands you can use to move around one character at a time:

Command	Description
k	Moves the cursor up one line.
j	Moves the cursor down one line.
h	Moves the cursor to the left one character position.
l	Moves the cursor to the right one character position.

There are following two important points to be noted:

- The vi is case-sensitive, so you need to pay special attention to capitalization when using commands.
- Most commands in vi can be prefaced by the number of times you want the action to occur.
- For example, 2j moves cursor two lines down the cursor location. There are many other ways to move within a file in vi.

Remember that you must be in command mode (press Esc twice). Here are some more commands you can use to move around the file:

Command	Description
0 or |	Positions cursor at beginning of line.
$	Positions cursor at end of line.
w	Positions cursor to the next word.
b	Positions cursor to previous word.
(	Positions cursor to beginning of current sentence.
)	Positions cursor to beginning of next sentence.

E	Move to the end of Blank delimited word
1G	Move to the first line of the file
G	Move to the last line of the file
nG	Move to nth line of the file
:n	Move to nth line of the file
H	Move to top of screen
nH	Moves to nth line from the top of the screen
M	Move to middle of screen
L	Move to botton of screen
nL	Moves to nth line from the bottom of the screen

Editing Files:

To edit the file, you need to be in the insert mode. There are many ways to enter insert mode from the command mode:

Command	Description
i	Inserts text before current cursor location.
I	Inserts text at beginning of current line.
a	Inserts text after current cursor location.
A	Inserts text at end of current line.
o	Creates a new line for text entry below cursor location.
O	Creates a new line for text entry above cursor location.
~	Change case of individual character

Deleting Characters:

Here is the list of important commands which can be used to delete characters and lines in an opened file:

Command	Description
x	Deletes the character under the cursor location.
X	Deletes the character before the cursor location.
dw	Deletes from the current cursor location to the next word.
4dw	Delete 4 words
d^	Deletes from current cursor position to the beginning of the line.
d$	Deletes from current cursor position to the end of the line.
D	Deletes from the cursor position to the end of the current line.
dd	Deletes the line the cursor is on.
3dd	Delete 3 lines.
C	Delete contents of a line after the cursor and insert new text. Press ESC key to end insertion.

- As mentioned above, most commands in VI can be prefaced by the number of times you want the action to occur.
- For example, 2x deletes two characters under the cursor location and 2dd deletes two lines the cursor is on.
- I would highly recommend exercising all the above commands properly before proceeding further.

Change Commands:

You also have the capability to change characters, words, or lines in vi without deleting them. Here are the relevant commands:

Command	Description
cc	Removes contents of the line, leaving you in insert mode.
cw	Changes the word the cursor is on from the cursor to the lowercase w end of the word.
r	Replaces the character under the cursor. vi returns to command mode after the replacement is entered.
R	Overwrites multiple characters beginning with the character currently under the cursor. You must use Esc to stop the overwriting.
s	Replaces the current character with the character you type. Afterward, you are left in insert mode.
S	Deletes the line the cursor is on and replaces with new text. After the new text is entered, vi remains in insert mode.
u	Undo last change
U	Undo all changes to the entire line

Copy and Past Commands:

You can copy lines or words from one place and then you can past them at another place using following commands:

Command	Description
yy	Copies the current line.
yw	Copies the current word from the character the lowercase w cursor is on until the end of the word.
p	Puts the copied text after the cursor.
P	Puts the yanked text before the cursor.

Saving and Closing the file

You should be in the command mode to exit the editor and save changes to the file.

Command	Description
Shift+zz	Save the file and quit
:w	Save the file but keep it open
:q	Quit without saving
:wq	Save the file and quit
ESC	Terminate insert mode

IMPORTANT

- You must be in command mode to use commands. (Press Esc twice at any time to ensure that you are in command mode.)
- You must be careful to use the proper case (capitalization) for all commands.
- You must be in insert mode to enter text.

Introduction of nano editor

Nano is a user-friendly, simple and **WYSIWYG (What You See Is What You Get)** text editor, which improves the features and user-friendliness of UW Pico text editor. Unlike vim editor or any other command-line editor, it doesn't have any mode. It has an easy GUI (Graphical User Interface) which allows users to interact directly with the text in spite of switching between the modes as in vim editor.

Installing Nano Text Editor

Nano is generally by default available in many Linux distributions but in case, it is not installed you may install the same using the following commands.

$sudo apt update

In case of Debian/Ubuntu

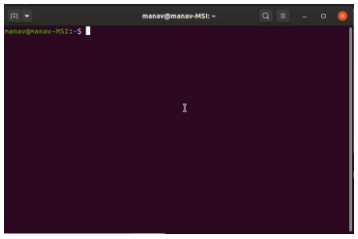
$sudo apt install nano

In case of CentOS/Fedora

$yum install nano

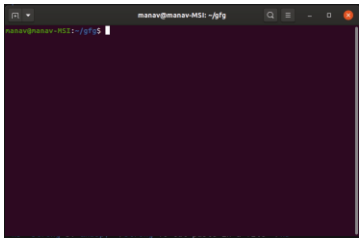
Working with Nano Text Editor

1. To create and open a new file.: **$nano new_filename**



The above command will open a new file with new_filename as shown in the output. In case the file already exists, it will open the same and in case the file is not there in the current directory it will create a new one. At the bottom of the window, there is a list of shortcut keys for nano.

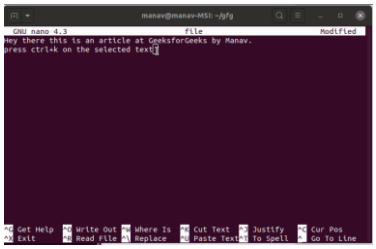
2. To save a file: **press Ctrl+o**



It will ask you for the filename. In case, you want to save the changes to a new file or want to create a new file then change the name else keep the name same.

As soon as you will press enter key, then In case, you have changed the name of the file then it will save the file with a new name and if not then it will save the changes to the current file.

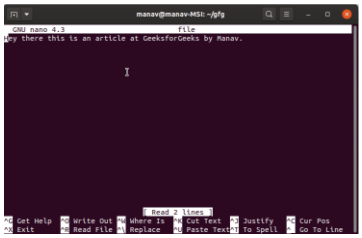
3. To cut paste in a file. ***Ctrl+o*** is used to cut and ***Ctrl+u*** is used to paste the text.



- *To cut and paste a whole line*. Move to the line which you want to cut then press ***Ctrl+k*.** Now the line is moved to clipboard, To paste it, go to the position where you want to paste and then press ***Ctrl+u***

- *To cut and paste the selected text*. Select the text which you want to cut then press ***Ctrl+k*.** Now the text is moved to clipboard. To paste it, go to the position where you want to paste and then press ***Ctrl+u*.**

4. To search a word in a file. ***Ctrl+w*** is used.



Press Ctrl+w

It will ask for a word to search for.

Enter the word

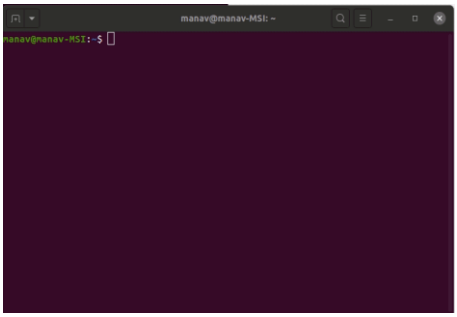
It will search for the word and will place the cursor in the first letter of the first occurrence of the word.

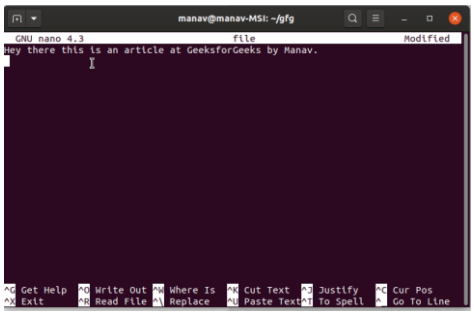
5. To enable spell check in nano. First, install the spell check package.

$sudo apt install spell

It will then ask for the password then enter the password. Then press y and then press enter.

- To do spell check first press **Ctrl+t**
- Now it will ask you to replace the incorrect words
- Enter the word to replace with there
- As soon as you will press the enter key





UNIT – 4
Shell Programming

Shell Keywords

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

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

What is a Variable?

In computer science, 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.

Variables play an important role in computer programming because they enable programmers to write flexible programs. As they are related to the Operating system that we work on, it is important to know some of them and how we can influence them.

Variable Types

When a shell is running, three main types of variables are present –

- **Environment/System Variables** – An environment variable is a variable that is available to any child process of the shell. Some programs need environment variables in order to function correctly. Usually a shell script defines only those environment variables that are needed by the programs that it runs.
- **Shell Variables** – 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.
- **Local Variables** – A local variable is a variable that is present within the current instance of the shell. It is not available to programs that are started by the shell. They are set at command prompt.

Shell Variables

The following table shows a number of special variables that you can use in your shell scripts

Variable	Description
$0	The filename of the current script.
$n	These variables correspond to the arguments with which a script was invoked. Here **n** is a positive decimal number corresponding to the position of an argument (the first argument is $1, the second argument is $2, and so on).

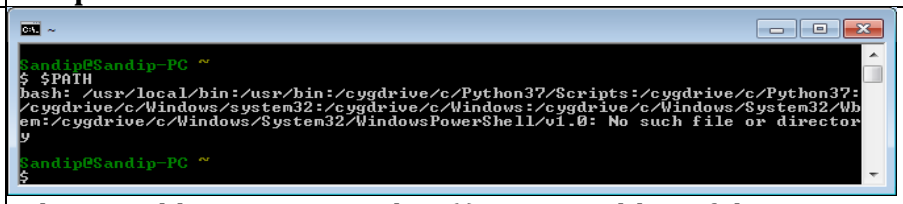
$#	The number of arguments supplied to a script.
$*	All the arguments are double quoted. If a script receives two arguments, $* is equivalent to $1 $2.
$@	All the arguments are individually double quoted. If a script receives two arguments, $@ is equivalent to $1 $2.
$?	The exit status of the last command executed.
$$	The process number of the current shell. For shell scripts, this is the process ID under which they are executing.
$!	The process number of the last background command.

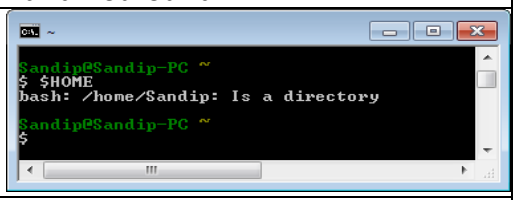
What are Environment / System variables?

Environment variables are dynamic values which affect the processes or programs on a computer. They exist in every operating system, but types may vary. Environment variables can be created, edited, saved, and deleted and give information about the system behavior.

Environment variables can change the way a software/programs behave E.g. $LANG environment variable stores the value of the language that the user understands. This value is read by an application such that a Chinese user is shown a Mandarin interface while an American user is shown an English interface.

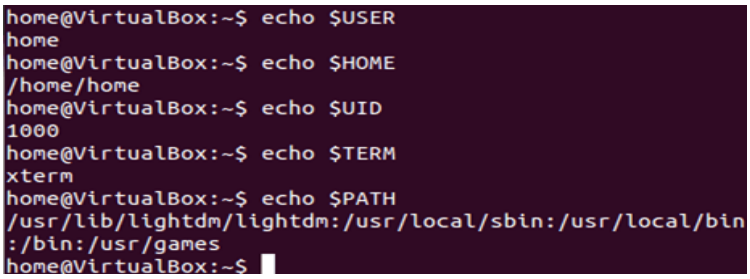
Let's study some common environment variables

Variable	Description
PS2	PS2 (**Prompt String 2**) is one of the prompts available in Linux/Unix. The other prompts are PS1, PS3 and PS4. This is very much useful for entering a large command in multiple lines and when you execute incomplete command, this prompt will come into picture. Check what your default PS2 prompt by executing below command: **echo $PS2** **Output: >** If you see the prompt changed from '>' to"-->'. This will be very handy and more informative when dealing with a command which spreads on multiple lines. **PS2="--->"** Set the above PS2 prompt and check it yourself with following data **echo $PS2** **Output: -->**
PATH	 This variable contains a colon (:)-separated list of directories in which your system looks for executable files. When you enter a command on terminal, the shell looks for the command in different directories mentioned in the

	$PATH variable. If the command is found, it executes. Otherwise, it returns with an error 'command not found'.
HOME	Indicates the home directory of the current user: the default argument for the cd built-in command. 
LOGNAME/ USER	Contains your username. It's set automatically when you log in.
TERM	Default terminal emulator
SHELL	Shell being used by the user
IFS	The **Internal Field Separator** to separate input on the command line. By default, this is a space.
MAIL	The path to the current user's mailbox.
MAILCHECK	The MAILCHECK environment variable contains the minimum number of seconds that must elapse before dbx checks for incoming mail. The check is performed just before printing the prompt, if $MAILCHECK seconds have elapsed since the last check. If $MAILCHECK is null or is zero, the check is performed before each prompt. If $MAILCHECK is unset, or is set to a non-numeric value, checking for mail is disabled. The default is 600 seconds

Accessing Variable values

In order to determine value of a variable, use the command Variables are-Case Sensitive. Make sure that you type the variable name in the right letter case otherwise you may not get the desired results.



The 'env' command displays all the environment variables.

USER VARIABLE

1. SET

The set command assigns a value to a variable **(or multiple values to multiple variables)**. Without any options, all set variables are shown.

If a value has spaces in it, it should be contained in quotes. If a list of values is desired, parentheses () should be used for the assignment, while individual access is done via square brackets [].

A single ***set*** command can be used to set many variables, but such a use is not recommended.

Examples:

```
set value = 7
echo $value
set new_val = "some number, like seven"
echo $new_val
```

2. UNSET

The unset command to remove a variable from your shell environment.

Examples:

```
set value = 7
echo $value
unset value
echo $value
```

3. ECHO

The echo command is used to displays the given text/message on the screen/ terminal window. This is often used to show the user what is happening or to prompt for a response.

Examples:

echo "BCA SEM 4 Semester"

POSITIONAL PARAMETERS

A positional parameter is a variable within a shell program; its value is set from an argument specified on the command line that invokes the program. Positional parameters are numbered and are referred to with a proceeding ``$'': $1, $2, $3, and so on

Positional Parameter Name	Returns true (0) if:
$0	The name of the script
$1	The first argument to the script
$2	The second argument to the script
$n	The n-th argument to the script
$#	The number of arguments to the script
$@	A list of all arguments to the script
$*	A list of all arguments to the script
${#N}	The length of the value of positional parameter N (Korn shell only)

NOTE:

$@ and $* have the same meaning. This is true if they are not enclosed in double quotes"".

Examples:

```
echo "The total no of args are: $#"
echo "The script name is : $0"
echo "The first argument is : $1"
echo "The second argument is: $2"
echo "The All arg is : $*"
```

```
[Sandip.Sandip-PC] ➤
sh positional.sh a b c d e f g h i j

The total no of args are: 10
The script name is : positional.sh
The first argument is : a
The second argument is: b
The All arg is : a b c d e f g h i j
```

Interactive shell script using read and echo

1. echo

echo command in Linux is used to display line of text/string that are passed as an argument . This is a built in command that is mostly used in shell scripts and batch files to output status text to the screen or a file

Syntax: echo [option] [string]

Tag	Description
-n	Do not output a trailing newline.
-e	Enable interpretation of backslash escape sequences (see below for a list of these).
-E	Disable interpretation of backslash escape sequences (this is the default).
--help	Display a help message and exit.
--version	Output version information and exit.
****	A literal backslash character ("\").
\a	An alert (The BELL character).
\b	Backspace
\c	Produce no further output after this.
\e	The escape character; equivalent to pressing the escape key.
\f	A form feed.
\n	A newline.
\r	A carriage return
\t	A horizontal tab.
\v	A vertical tab.
\0NNN	byte with octal value NNN (which can be 1 to 3 digits).
\xHH	byte with hexadecimal value HH (which can be either 1 or 2 digits)

2. read

read is a built in command of the Bash shell, which reads a line of text from standard input and splits it into words. These words can then be used as the input for other commands.

Syntax: read [-ers] [-a array] [-d delim] [-i text] [-n nchars] [-N nchars][-p prompt] [-t timeout] [-u fd] [name ...] [name2 ...]

Option	Description
-a *array*	Store the words in an indexed array named *array*. Numbering of array elements starts at zero.
-d *delim*	Set the delimiter character to *delim*. This character will signal the end of the line. If **-d** is not used, the default line delimiter is a newline.
-e	Get a line of input from an interactive shell. The user manually inputs characters until the line delimiter is reached.
-i *text*	When used in conjunction with **-e** (and only if **-s** is not used), *text* is inserted as the initial text of the input line. The user is permitted to edit *text* on the input line.
-n *nchars*	Stop reading after an integer number *nchars* characters have been read, if the line delimiter has not been reached.

-N *nchars*	Ignore the line delimiter. Stop reading only after *nchars* characters have been read, EOF is reached, or read times out (see **-t**).
-p *prompt*	Print the string *prompt*, without a newline, before beginning to read.
-r	Use "raw input". Specifically, this option causes **read** to interpret backslashes literally, rather than interpreting them as escape characters.
-s	Do not echo keystrokes when **read** is taking input from the terminal.
-t *timeout*	Time out, and return failure, if a complete line of input is not read within *timeout* seconds. If the timeout value is zero, **read** will not read any data, but will return success if input was available to read. If *timeout* is not specified, the value of the shell variable **TMOUT** is used instead, if it exists. The value of *timeout* can be a fractional number, e.g., **3.5**.
-u *fd*	Read from the file descriptor *fd* instead of standard input. The file descriptor should be a small integer. For information about opening a custom file descriptor,

Shell Basic Operators

There are various operators supported by each shell. We will discuss in detail about Bourne shell (default shell). We will now discuss the following operators

- Arithmetic Operators
- Relational Operators
- Boolean Operators
- String Operators
- File Test Operators

Arithmetic Operators

The following arithmetic operators are supported by Bourne Shell. Assume variable a holds 10 and variable b holds 20 then

Operator	Description	Example
+ (Addition)	Adds values on either side of the operator	`expr $a + $b` will give 30
- (Subtraction)	Subtracts right hand operand from left hand operand	`expr $a - $b` will give -10
* (Multiplication)	Multiplies values on either side of the operator	`expr $a \* $b` will give 200
/ (Division)	Divides left hand operand by right hand operand	`expr $b / $a` will give 2
% (Modulus)	Divides left hand operand by right hand operand and returns remainder	`expr $b % $a` will give 0
= (Assignment)	Assigns right operand in left operand	a = $b would assign value of b into a
== (Equality)	Compares two numbers, if both are same then returns true.	[$a == $b] would return false.
!= (Not Equality)	Compares two numbers, if both are different then returns true.	[$a != $b] would return true.

It is very important to understand that all the conditional expressions should be inside square braces with spaces around them, for example [$a == $b]is correct whereas, [$a==$b] is incorrect. All the arithmetical calculations are done using long integers

Relational Operators

Bourne Shell supports the following relational operators that are specific to numeric values. These operators do not work for string values unless their value is numeric.

For example, following operators will work to check a relation between 10 and 20 as well as in between "10" and "20" but not in between "ten" and "twenty".

Assume variable a holds 10 and variable b holds 20 then −

Operator	Description	Example
-eq	Checks if the value of two operands is equal or not; if yes, then the condition becomes true.	[$a -eq $b] is not true.
-ne	Checks if the value of two operands is equal or not; if values are not equal, then the condition becomes true.	[$a -ne $b] is true.
-gt	Checks if the value of left operand is greater than the value of right operand; if yes, then the condition becomes true.	[$a -gt $b] is not true.
-lt	Checks if the value of left operand is less than the value of right operand; if yes, then the condition becomes true.	[$a -lt $b] is true.
-ge	Checks if the value of left operand is greater than or equal to the value of right operand; if yes, then the condition becomes true.	[$a -ge $b] is not true.
-le	Checks if the value of left operand is less than or equal to the value of right operand; if yes, then the condition becomes true.	[$a -le $b] is true.

It is very important to understand that all the conditional expressions should be placed inside square braces with spaces around them. For example, [$a <= $b] is correct whereas, [$a <= $b] is incorrect.

Boolean Operators

The following Boolean operators are supported by the Bourne Shell. Assume variable a holds 10 and variable b holds 20 then −

Operator	Description	Example
!	This is logical negation. This inverts a true condition into false and vice versa.	[! false] is true.
-o	This is logical OR. If one of the operands is true, then the condition becomes true.	[$a -lt 20 -o $b -gt 100] is true.
-a	This is logical AND. If both the operands are true, then the condition becomes true otherwise false.	[$a -lt 20 -a $b -gt 100] is false.

String Operators

The following string operators are supported by Bourne Shell. Assume variable a holds "abc" and variable b holds "efg"

Operator	Description	Example
=	Checks if the value of two operands is equal or not; if yes, then the condition becomes true.	[$a = $b] is not true.

!=	Checks if the value of two operands is equal or not; if values are not equal then the condition becomes true.	[$a != $b] is true.
-z	Checks if the given string operand size is zero; if it is zero length, then it returns true.	[-z $a] is not true.
-n	Checks if the given string operand size is non-zero; if it is nonzero length, then it returns true.	[-n $a] is not false.
str	Checks if str is not the empty string; if it is empty, then it returns false.	[$a] is not false.

Decision Statement

We will understand shell decision-making in Unix. 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. We will now understand two decision-making statements here

- The if...else statement
- The case... esac statement

The if...else statements

If else statements are useful decision-making statements which can be used to select an option from a given set of options.

Unix Shell supports following forms of **if...else** statement

- if...fi statement
- if...else...fi statement
- if...elif...else...fi statement
- if..then..else..if..then..fi..fi..(Nested if)

Most of the if statements check relations using relational operators discussed in the previous chapter.

if... fi statement (Simple if)

Simple if is used for decision making in shell script. if the given condition is true then it will execute the set of code that you have allocated to that block.

Syntax	Example
if [condition] **then** Execute the statements **fi**	echo "Enter Number:-" read no if [$no -eq 1] then echo "Number 1 " fi

if...else...fi statement

if.. else .. fi statement is used for decision making in shell script where the given condition is true then it will execute the set of code that you have allocated to that block otherwise you can execute the rest of code for the false condition.

Syntax	Example

if [condition] then Execute Statement if Condition is True elif Execute Statement if Condition is False fi	#Check Number is Positive or Not echo "Enter Number:" read no if [$no -gt 0] then echo "Number is Positive" elif echo "Number is Negative" fi

if...elif...else...fi statement

it is possible to create compound conditional statements by using one or more else if(elif) clause. if the 1st condition is false, then subsequent elif statements are checked. when an elif condition is found to be true, the statements following that associated parts are executed.

Syntax	Example
if [condition - 1] then Execute Statement if Condition 1 elif [condition -2] Execute Statement if Condition 2 elif [condition -3] Execute Statement if Condition 3 elif Else Condition fi	echo "Enter Student Mark:-" read mark if [$mark -gt 70] then echo "Distinction" elif [$mark -gt 60] then echo "First Class" elif [$mark -gt 50] then echo "Second Class" elif [$mark -gt 40] then echo "Pass Class" elif echo "Fail" fi

if..then..else..if..then..fi..fi..(Nested if)

if statement and else statement can be nested in bash shell programming the keyword "fi" indicates the end of the inner if statement and all if statement should end with "fi".

Syntax	Example
if [condition] then if [condition] then Execute Statement elif Execute Statement fi elif	#Nested if Example echo "Enter Your number :" read no if [$no –ne 0] then if [$no -gt 0] then echo "Positive Value" elif

Execute Statement fi	echo "Negative Value" fi elif echo "Zero Value" fi

The case...esac Statement

You can use multiple **if...elif** statements to perform a multi-way branch. However, this is not always the best solution, especially when all of the branches depend on the value of a single variable.

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 which has been described in detail here

- case...esac statement

The case...esac statement in the UNIX shell is very similar to the switch...case statement we have in other programming languages like C or C++ etc.

Syntax	Example
case $[variable_name] in value1) Statement 1 ;; value2) Statement 2 ;; value3) Statement 3 ;; value4) Statement 4 ;; valueN) Statement N ;; *) Default Statement ;; esac	#Case Statement Example echo "Enter Number:" read no case $no in 1) echo "ONE" ;; 2) echo "TWO ;; 3) echo "Three" ;; default echo “input between 1 to 3” esac

Test Command

The test command checks for various properties of files, strings and integers. It produces no output (except error messages) but returns the result of the test as the exit status

The command line includes a Boolean expression. The test command can test for various conditions, and then returns 0 for true and 1 for false. Usually, test is used in if...then or while loops.

Syntax:

test *expression*	**test [** *expression* **]**	**test [[** *expression* **]]**

Example

```
Sandip@Sandip-PC ~$ a=10
Sandip@Sandip-PC ~$ b=20
Sandip@Sandip-PC ~$ test $a -le $b
Sandip@Sandip-PC ~$ echo $?
0
Sandip@Sandip-PC ~$ test $a -eq $b
Sandip@Sandip-PC ~$ echo $?
1
```

Looping Statement

A loop is a powerful programming tool that enables you to execute a set of commands repeatedly we will examine the following types of loops available to shell programmers

- The while loop
- The for loop
- The until loop

While Loop

The while loop enables you to execute a set of commands repeatedly until some condition occurs. It is usually used when you need to manipulate the value of a variable repeatedly.

Syntax	Example	Output
while command **do** **Statement(s) to be executed if command is true** **done**	clear i=1 while [$i -le 5] do echo $i i=$((i+1)) done	1 2 3 4 5

Here the Shell command is evaluated. If the resulting value is true, given statement(s) are executed. If command is false then no statement will be executed and the program will jump to the next line after the done statement.

FOR Loop

The for loop operates on lists of items. It repeats a set of commands for every item in a list.

Syntax	Example	Output
for var **in** word1 word2 ... wordN **do** **Statement(s) to be executed for every word.** **done**	for i in {1..5} do echo $i done	1 2 3 4 5
	for var in 1 2 3 4 5 do echo $var done	
	for FILE in $HOME/.bash* do echo $FILE done	/root/.bash_history /root/.bash_logout /root/.bash_profile /root/.bashrc

Here var is the name of a variable and word1 to wordN are sequences of characters separated by spaces (words). Each time the for loop executes, the value of the variable var is set to the next word in the list of words, word1 to wordN. And also display all the files starting with .bash and available in your home. We will execute this script from my root

UNTILL Loop

The while loop is perfect for a situation where you need to execute a set of commands while some condition is true. Sometimes you need to execute a set of commands until a condition is true.

Syntax	Example	Output
until command **do** **Statement(s) to be executed** **until command is true** **done**	i=0 until [! $i -lt 5] do echo $a i=$((i+1)) done	1 2 3 4 5

Here the Shell command is evaluated. If the resulting value is false, given statement(s) are executed. If the command is true then no statement will be executed and the program jumps to the next line after the done statement.

Break Statement

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.

The break command can also be used to exit from a nested loop using this format Here n specifies the nth enclosing loop to the exit from.

Syntax	Example	Output
break or **break n**	a=0 while [$a -lt 10] do echo $a if [$a -eq 5]	0 1 2 3 4

	then break fi a=$(($a + 1)) done	5

Continue statement

Continue is a command which is used to skip the current iteration in for, while and until loop. It takes one more parameter [N], if N is mentioned then it continues from the nth enclosing loop.

Syntax	Example	Output
continue or **continue [N]**	a=0 while [$a -lt 10] do echo $a if [$a -eq 5] then continue fi a=$(($a + 1)) done	0 1 2 3 4 6 7 8 9

Array

A shell variable is capable enough to hold a single value. These variables are called scalar variables.

Shell supports a different type of variable called an array variable. This can hold multiple values at the same time. Arrays provide a method of grouping a set of variables. Instead of creating a new name for each variable that is required, you can use a single array variable that stores all the other variables.All the naming rules discussed

Arrays in Shell Scripting

An array is a structured arrangement of similar data elements. Within shell scripting, an array is a variable that holds multiple values, whether they are of the same type or different types. It's important to note that in shell scripting, everything is treated as a string. Arrays adhere to a zero-based index, which signifies that indexing initiates from 0.

How to Declare Array in Shell Scripting?

Arrays can be declared in a shell script using various approaches:

1. Indirect Declaration

In this method, you assign a value to a specific index of the array variable. There's no need to declare the array beforehand.

ARRAYNAME[INDEXNR]=value

2. Explicit Declaration

With explicit declaration, you first declare the array and then assign values to it.

declare -a ARRAYNAME

3. Compound Assignment

This method involves declaring the array along with its initial set of values. You can later add additional values to the array.

ARRAYNAME=(value1 value2 .... valueN)

Alternatively, you can use index numbers to assign values explicitly:

ARRAYNAME=([1]=10 [2]=20 [3]=30)

Printing Array Values in Shell Script:

To display array elements, you have several options:

Here is a `array_test.sh`script explaining multiple options. (You can create script with any name)

```
#!/bin/bash
# To declare a static Array
arr= ("BCA" "COMPUTER" "BBA" "MANAGMENT" "BCOM" "ACCOUNT")
echo "All elements of the array:"          # To print all elements of the array
echo "${arr[@]}"
echo "${arr[*]}"

# To print the first element
echo "The first element:"
echo "${arr[0]}"

# To print a selected index element
selected_index=3
echo "Selected index element at index $selected_index:"
echo "${arr[$selected_index]}"

# To print elements from a particular index
echo "Elements from a particular index:"
echo "${arr[@]:2}" # Prints elements starting from index 2
echo "${arr[*]:2}" # Prints elements starting from index 2

# To print elements in a range
echo "Elements in a range:"
echo "${arr[@]:1:3}" # Prints elements from index 1 to 3
echo "${arr[*]:1:3}" # Prints elements from index 1 to 3
```

Explanation:

1. **Array Declaration:** An array named arr is declared, containing six elements. These elements are strings: "BCA", "COMPUTER", "BBA", "MANAGMENT", "BCOM", and "ACCOUNT".
2. **Printing All Elements:**
   - echo "All elements of the array:": A message is printed to indicate that all elements of the array are being displayed.
   - ${arr [@]}: This syntax is used to print each element of the array separately. It displays all elements in the array.
   - ${arr [*]}: Similar to the previous line, this syntax prints all elements of the array as a single string.
3. **Printing the First Element:**
   - echo "The first element:": A message is printed to indicate that the first element of the array is being displayed.

- ${arr [0]}: This syntax retrieves and displays the first element of the array. In Bash, array indexing starts at 0.

4. **Printing a Selected Index Element:**
   - selected_index=3: A variable named selected_index is assigned the value 3. This variable represents the desired index in the array.
   - echo "Selected index element at index $selected_index:": A message is printed to indicate the selected index.
   - ${arr[$selected_index]}: Using the value stored in selected_index, this syntax retrieves and displays the element at the specified index (index 3 in this case).
5. **Printing Elements from a Particular Index:**
   - echo "Elements from a particular index:": A message is printed to indicate that elements from a specific index are being displayed.
   - ${arr [@]:2}: This syntax extracts and displays all elements starting from index 2 in the array. It prints each element separately.
   - ${arr [*]:2}: Similar to the previous line, this syntax displays the elements as a single string, starting from index 2.
6. **Printing Elements in a Range:**
   - echo "Elements in a range:": A message is printed to indicate that elements within a specified range are being displayed.
   - ${arr[@]:1:3}: This syntax extracts and displays elements starting from index 1 up to index 3 (inclusive). It prints each element separately.
   - ${arr[*]:1:3}: Similar to the previous line, this syntax displays the extracted elements as a single string.

Function

A shell is a command-line interpreter, and shell scripts commonly execute file manipulation, program execution, and text output. In this article, we are going to read about functions and their types in shell scripting.

Definition

A function is a collection of statements that execute a specified task. Its main goal is to break down a complicated procedure into simpler subroutines that can subsequently be used to accomplish the more complex routine.

Function syntax

```
function functionName()
{
        # some code goes here...
}
```

Where functionName is the name of the function. After the name we have the opening and closing parenthesis (). **The body of the function starts from { and ends at }**. Inside the opening and closing curly brackets {} we have the body of the function. Another way of creating a function is given below.

```
functionName()
{
  # body of the function
}
```

Function name:-We follow the given rules when naming functions.

- Name must start with letters (a-z or A-Z) or underscore _

- After that we can use letters (a-z and A-Z), underscore _ and digits (0-9)
- Don't put space in the function name
- Don't use special keywords like echo, printf etc to name your function

Example #01: Write a Shell Script to print "Hello World" using function

In the following example we are creating greetings function.

CODE	Output
#!/bin/sh **# create a function** function greetings() { echo "Hello World" } **# call the function** greetings	$ sh example01.sh Hello World

Passing argument to a function

We can also pass arguments to a function in shell script and access them using variables like $1, $2, $3... where, $1 points at the first argument, $2 points at the second argument and so on.

Example #02: Write a Shell Script to create a function that accepts user name and displays a greetings message

In the following example we are creating greeting's function which accepts an argument and prints a greetings message.

CODE	Output:
#!/bin/sh # create a function greetings() { echo "Hello $1" } # call the function greetings "Smt. JJKCC"	$ sh example02.sh Hello Smt. JJKCC

Example #03: Write a Shell Script to print greetings message using the name entered by the user

In this example we will first take the name from the user and save it in variable name.

CODE	Output:
#!/bin/sh # function greetings() { echo "Hello $1" } # take user name printf "Enter your name: " read name # call the function greetings "$name"	$ sh example03.sh Enter your name: Smt. JJKCC Hello Smt. JJKCC

Return Code: We use the return command to return value from a function.

Example #04: Write a Shell Script to create a function that takes a number as argument and finds the square. On success it returns 0 otherwise, 1

For this we will create square function which will take a number as argument.

CODE	Output:
#!/bin/sh # function square() { # if argument missing if [-z "$1"] then return 1 # return error code fi # assign argument to variable n n="$1" # find square result=`expr "$n * $n" \| bc -l` return 0 # return success code } # take user input printf "Enter a number: " read num # call the function square "$num" # save the returned code from the above function call returnCode=$? if [$returnCode -eq 0] then # display the result printf "Square of %d = %d\n" "$num" "$result" elif [$returnCode -eq 1] then printf "Error Code: $returnCode Error: Number missing!\n" else printf "Error Code: $returnCode Error: Unknown!\n" fi	$ sh example04.sh Enter a number: 5 Square of 5 = 25 $ sh example04.sh Enter a number: Error Code: 1 Error: Number missing!

Value returned by the function is accessed using $? so, we use it to save the returned value in variable $returnCode. So, result variable is like a global variable even though it is inside the square function. This is different from what we know from other programming languages like C, C++, Java etc. were a variable defined inside a function can't be accessed from outside directly.

Example #05: Write a Shell Script to compute Simple Interest based on user input

CODE	Output:
#!/bin/sh # simple interest function simpleInterest () { p="$1" r="$2" t="$3" si=`expr "($p * $r * $t)/100" | bc -l` } # take user input printf "Enter Principal: " read pr printf "Enter Rate [0-100]: " read rt printf "Enter Time [in years]: " read tm # call function simpleInterest $pr $rt $tm printf "Simple Interest: %.3f\n" "$si"	Output: $ sh example05.sh Enter Principal: 100 Enter Rate [0-100]: 10 Enter Time [in years]: 2 Simple Interest: 20.000

UNIT –5

Getting Started With Linux

History of Linux

Linux is the first truly free Unix-like operating system. The underlying GNU Project was launched in 1983 **by *Richard Stallman*** originally to develop a Unix-compatible operating system called GNU, intended to be entirely free software. Many programs and utilities were contributed by developers around the world, and by 1991 most of the components of the system were ready. Still missing was the kernel.

Linus Torvalds invented Linux itself. In 1991, To rivals was a student at the University of Helsinki in Finland where he had been using Minix, a non-free Unix-like system, and began writing his own kernel. He started by developing device drivers and hard-drive access, and by September had a basic design that he called Version 0.01. This kernel, which is called Linux, was afterwards combined with the GNU system to produce a complete free operating system.

On October 5th, 1991, Torvalds sent a posting to the comp. OS minix newsgroup announcing the release of Version 0.02, a basic version that still needed Minix to operate, but which attracted considerable interest nevertheless. The kernel was then rapidly improved by Torvalds and a growing number of volunteers communicating over the *Internet*, and by December 19th a functional, stand-alone Unix-like Linux system was released as Version 0.11.

On January 5, 1992, Linux Version 0.12 was released, an improved, stable kernel. The next release was called Version 0.95, to reflect the fact that it was becoming a full-featured system. After that Linux became an underground phenomenon, with a growing group of distributed programmers that continue to debug, develop, and enhance the source code baseline to this day.

Torvalds released Version 0.11 under a freeware license of his own devising, but then released Version 0.12 under the well established GNU General Public License. More and more free software was created for Linux over the next several years.

Linux continued to be improved through the 1990's, and started to be used in large-scale applications like *web hosting*, networking, and database serving, proving ready for production use. Version 2.2, a major update to the Linux kernel, was officially released in January 1999. By the year 2000, most computer companies supported Linux in one way or another, recognizing a common standard that could finally reunify the fractured world of the *Unix Wars*. The next major release was V2.4 in January 2001, providing (among other improvements) compatibility with the upcoming generations of Intel's 64-bit Itanium computer processors.

Although Torvalds continued to function as the Linux kernel release manager, he avoided work at any of the many companies involved with Linux in order to avoid showing favoritism to any particular organization, and instead went to work for a company called Transmeta and helped develop mobile computing solutions, and made his home at the Open Source Development Labs (OSDL), which merged into The Linux Foundation.

Concept of GNU

The GNU Linux project was created for the development of a Unix-like operating system that comes with source code that can be copied, modified, and redistributed.

Richard Stallman announced the GNU Linux project in 1983 and, with others, formed the Free Software Foundation in 1985.

According to the GNU Linux project, there is no independent GNU operating system. Furthermore, they claim that there is no independent Linux operating system either. The OS known as Linux is based on the Linux kernel but all other components are GNU. As such, many believe that the OS should be known as GNU/Linux or GNU Linux.

Concept of GPL

General **P**ublic **L**icense, the license that accompanies some open source software that details how the software and its accompany source code can be freely copied, distributed and modified. The most widespread use of *GPL* is in reference to the GNU GPL, which is commonly abbreviated simply as *GPL* when it is understood that the term refers to the GNU GPL. One of the basic tenets of the GPL is that anyone who acquires the material must make it available to anyone else under the same licensing agreement.

The GPL does not cover activities other than the copying, distributing and modifying of the source code.

Open Source & Freeware

Open Source

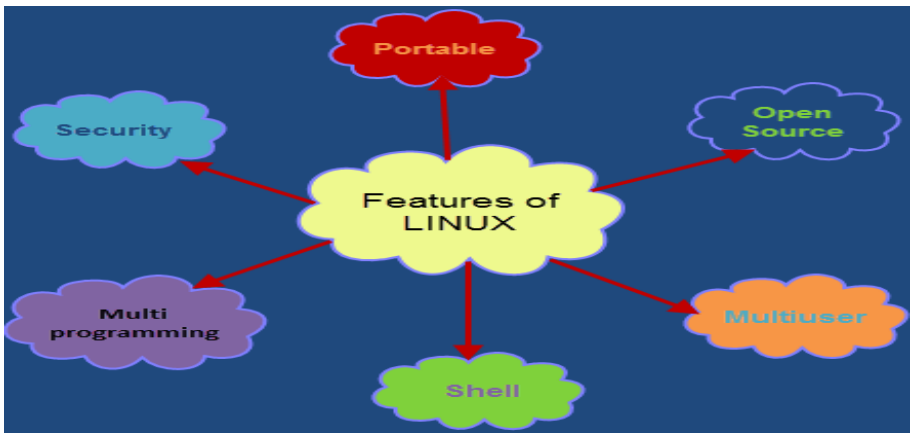
Open-source software (OSS) is computer software with its source code made available with a license in which the copyright holder provides the rights to study, change, and distribute the software to anyone and for any purpose. Open-source software may be developed in a collaborative public manner. According to scientists who studied it, open-source software is a prominent example of open collaboration.

The open-source software development, or collaborative development from multiple independent sources, generates an increasingly more diverse scope of design perspective than any one company is capable of developing and sustaining long term.

Freeware

Freeware is proprietary software that is available for use at no monetary cost. In other words, freeware may be used without payment but may usually not be modified, re-distributed or reverse-engineered without the author's permission. Two historic examples of freeware include Skype and Adobe Acrobat Reader.

Freeware, although itself free of charge, may be intended to benefit its owner, e.g. by encouraging sales of a more capable version ("Freemium" or Shareware business model). The source code of freeware is typically not available, unlike free software and open source software which are also often distributed free of charge.



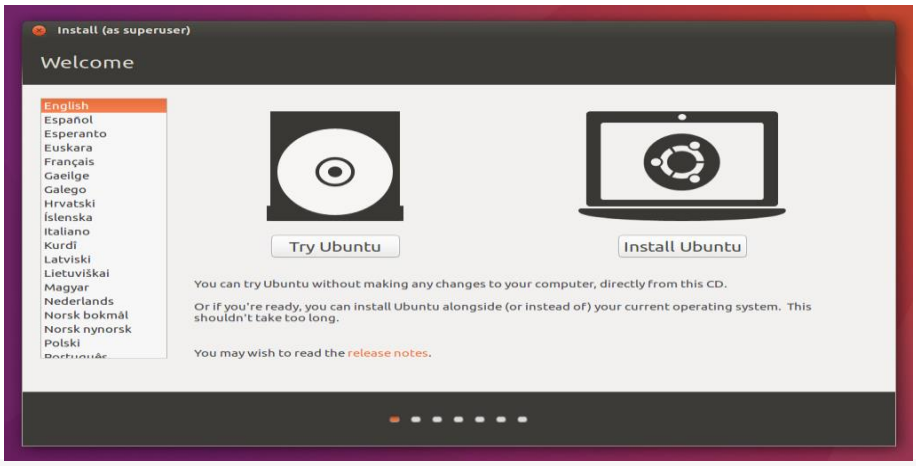
Features

- **Portable** – Portability means software's can works on different types of hardware in same way. Linux kernel and application programs support their installation on any kind of hardware platform.
- **Open Source** – Linux source code is freely available and it is community based development project. Multiple teams works in collaboration to enhance the capability of Linux operating system and it is continuously evolving.
- **Multi-User** – Linux is a multiuser system means multiple users can access system resources like memory/ ram/ application programs at same time.
- **Multiprogramming** – Linux is a multiprogramming system means multiple applications can run at same time.
- **Hierarchical File System** – Linux provides a standard file structure in which system files/ user files are arranged.
- **Shell** – Linux provides a special interpreter program which can be used to execute commands of the operating system. It can be used to do various types of operations, call application programs etc.
- **Security** – Linux provides user security using authentication features like password protection/ controlled access to specific files/ encryption of data.
- **Demand loads executables:** Linux only reads from disk those parts of a program that are actually used.
  - **Shared copy-**on-write pages among executables. This means that multiple processes can use the same memory to run in. When one tries to write to that memory, that page (4KB piece of memory) is copied somewhere else. Copy-on-write has two benefits: increasing speed and decreasing memory use.
  - **Virtual memory** using paging (not swapping whole processes) to disk: to a separate partition or a file in the file system, or both, with the possibility of adding more swapping areas during runtime (yes, they're still called swapping areas). A total of 16 of these 128 MB (2GB in recent kernels) swapping areas can be used at the same time, for a theoretical total of 2 GB of useable swap space. It is simple to increase this if necessary, by changing a few lines of source code.
  - **A unified memory** pool for user programs and disk cache, so that all free memory can be used for caching, and the cache can be reduced when running large programs.
  - **All source code** is available, including the whole kernel and all drivers, the development tools and all user programs; also, all of it is freely distributable. Plenty of commercial programs are being provided for Linux without source, but everything that has been free, including the entire base operating system, is still free.
  - **Multiple virtual consoles**: several independent login sessions through the console, you switch by pressing a hot-key combination (not dependent on video hardware). These are dynamically allocated; you can use up to 64.
  - **Supports several common file** systems, including minix, Xenix, and all the common system V file systems, and has an advanced file system of its own, which offers file systems of up to 4 TB, and names up to 255 characters long.
  - **Many networking protocols**: the base protocols available in the latest development kernels include TCP, IPv4, IPv6, AX.25, X.25, IPX, DDP

(AppleTalk), Netrom, and others. Stable network protocols included in the stable kernels currently include TCP, IPv4, IPX, DDP, and AX.25.

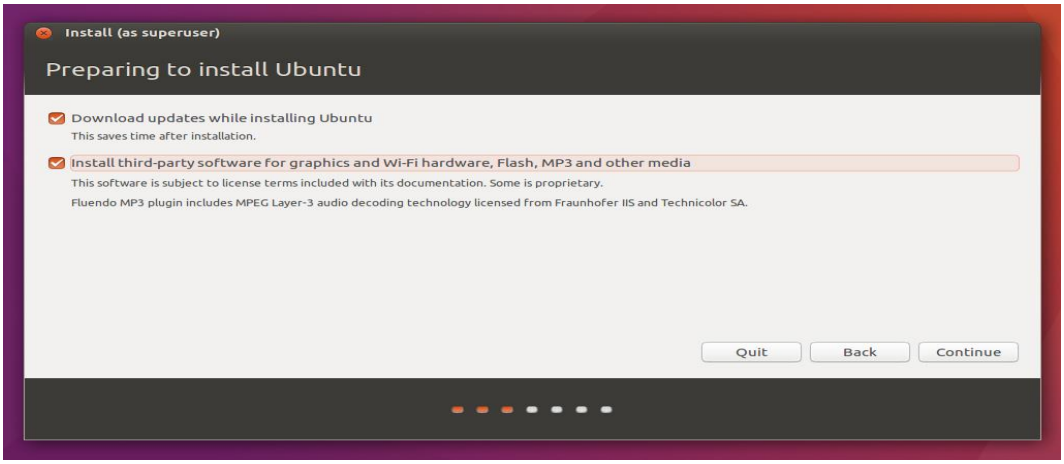
Installation Of Ubuntu

If your computer doesn't automatically do so, you might need to press the F12 key to bring up the boot menu, but be careful not to hold it down - that can cause an error message.



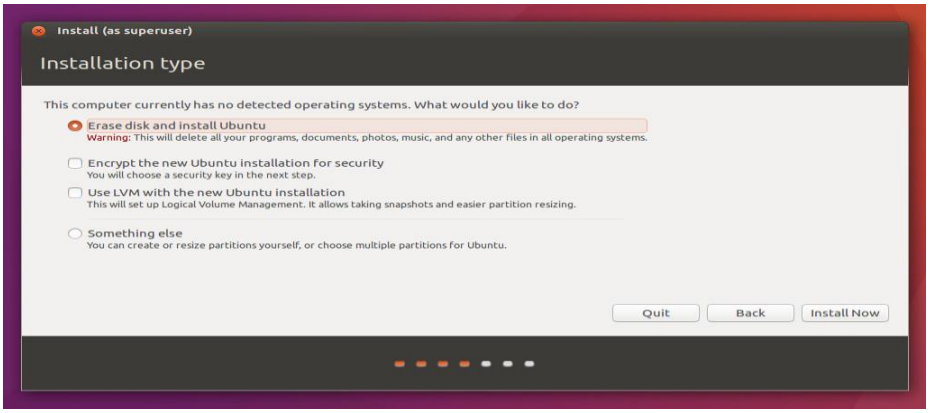
Prepare to install Ubuntu

- We recommend you plug your computer into a power source
- You should also make sure you have enough space on your computer to install Ubuntu
- We advise you to select Download updates while installing and Install this third-party software now
- You should also stay connected to the internet so you can get the latest updates while you install Ubuntu
- If you are not connected to the internet, you will be asked to select a wireless network, if available. We advise you to connect during the installation so we can ensure your machine is up to date



Allocate drive space

Use the checkboxes to choose whether you'd like to Install Ubuntu alongside another operating system, delete your existing operating system and replace it with Ubuntu, or — if you're an advanced user — choose the 'Something else' option

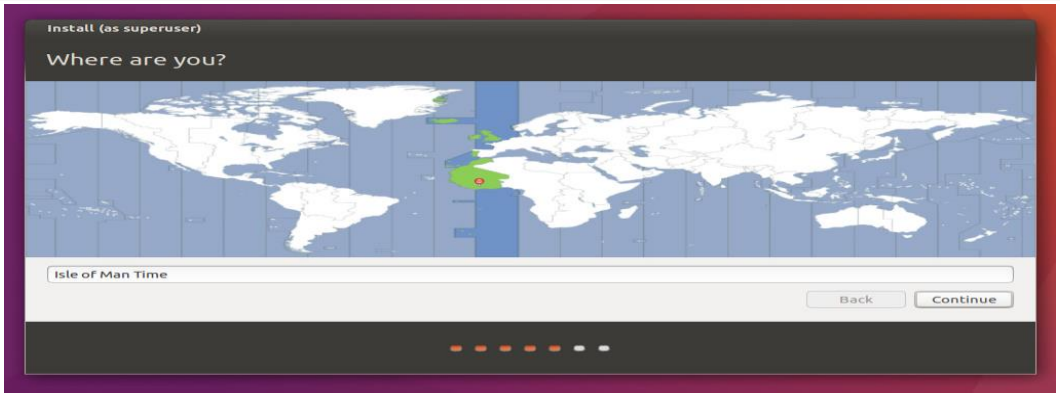


Begin the installation

Depending on your previous selections, you can now verify that you have chosen the way in which you would like to install Ubuntu. The installation process will begin when you click the Install Now button. Ubuntu needs about 4.5 GB to install, so add a few extra GB to allow for your files.

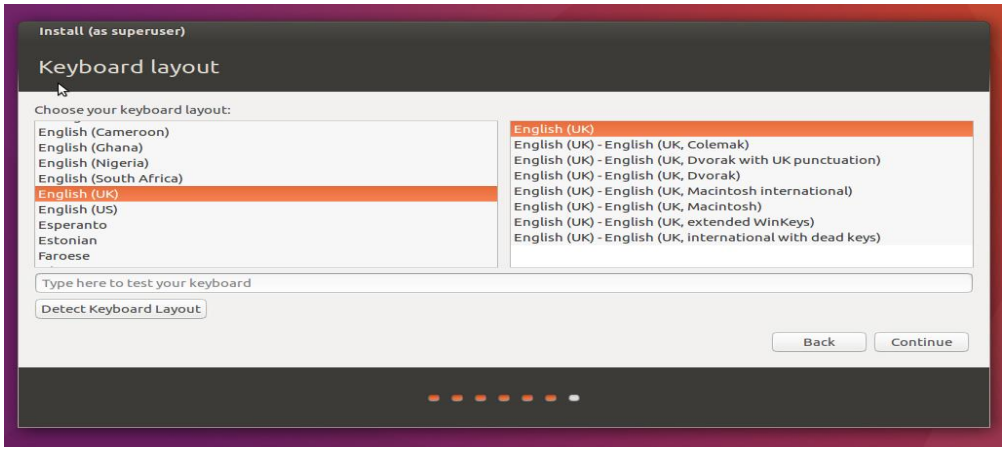
Select your location

If you are connected to the internet, this should be done automatically. Check your location is correct and click 'Forward' to proceed. If you're unsure of your time zone, type the name of the town you're in or click on the map and we'll help you find it.

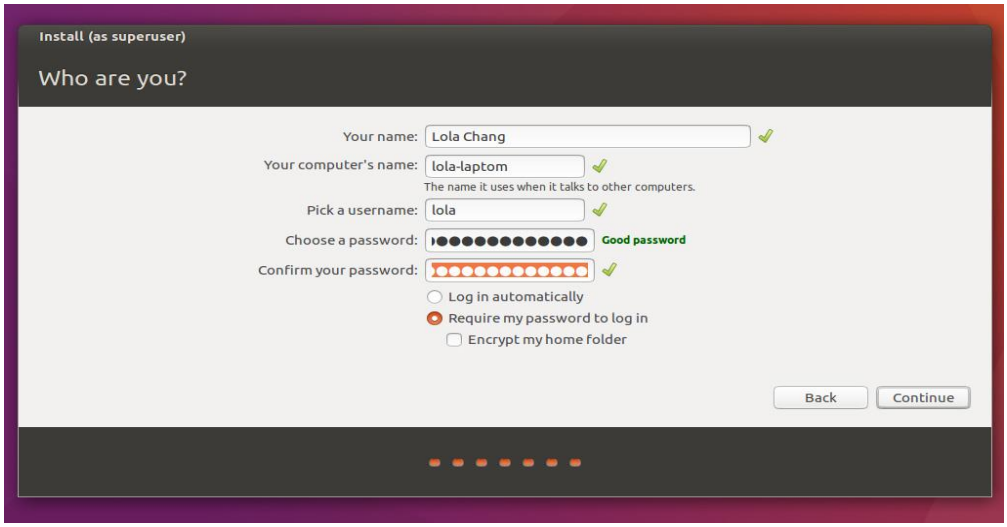


Select your preferred keyboard layout

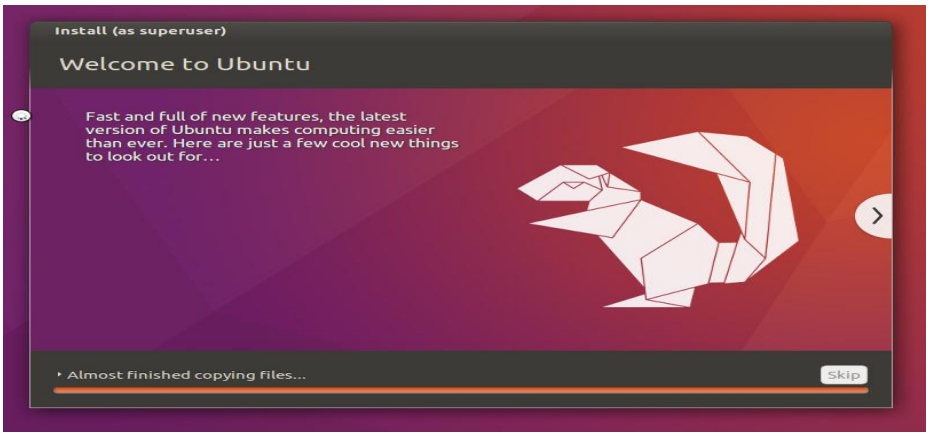
Click on the language option you need. If you're not sure, click the 'Detect Keyboard Layout' button for help.



Enter your login and password details



Learn more about Ubuntu while the system installs…
…or make a cup of tea!



Startup, Shutdown and Boot Loaders of Linux

Startup and shutdown scripts are used to manually start up or shut down Couch base Server. On Linux, Couch base Server is installed as a standalone application with support for running as a background (daemon) process during start-up through the use of a standard control script, which is located in /etc/init.d/couchbase-server .

The startup script is automatically installed during installation from one of the Linux packaged releases (Debian / Ubuntu or Red Hat/CentOS). By default, Couch base Server is configured to be started automatically at run levels 2, 3, 4, and 5, and explicitly shut down at run levels 0, 1 and 6.

To manually start Couch base Server using the startup/shutdown script:

```
>> sudo /etc/init.d/couchbase-server start
```

To manually stop Couch base Server using the startup/shutdown script:

```
>> sudo /etc/init.d/couchbase-server stop
```

UNIT – 5
Linux Booting

Linux Booting Process

1. **BIOS**
   - 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
   - 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.
   - Retrieves information from CMOS (Complementary Metal-Oxide Semiconductor) a battery-operated memory chip on the motherboard that stores time, date, and critical system information.
   - Once BIOS sees everything is fine it will begin searching for an operating system Boot Sector on a valid master boot sector on all available drives like hard disks CD-ROM drive etc.
   - Once BIOS finds a valid MBR it will give the instructions to boot and executes the first 512-byte boot sector that is the first sector ("Sector 0") of a partitioned data storage device such as hard disk or CD-ROM etc
2. **MBR**
   - Normally we use multi-level boot loader. Here MBR means I am referencing to DOS MBR.
   - Afer BIOS executes a valid DOS MBR, the DOS MBR will search for a valid primary partition marked as bootable on the hard disk.
   - if MBR finds a valid bootable primary partition then it executes the first 512-bytes of that partition which is second level MBR.
   - In linux we have two types of the above mentioned second level MBR known as LILO and GRUB
3. **GRUB/ LILO**
   - ❖ **LILO**
     - LILO is a Linux boot loader which is too big to fit into single sector of 512-bytes.
     - So it is divided into two parts: an installer and a runtime module.
     - The installer module places the runtime module on MBR. The runtime module has the info about all operating systems installed.
     - When the runtime module is executed, it selects the operating system to load and transfers the control to kernel.

- LILO does not understand file systems and boot images to be loaded and treats them as raw disk offsets

❖ **GRUB**

- GRUB MBR consists of 446 bytes of primary boot loader code and 64 bytes of the partition table.
- GRUB locates all the operating systems installed and gives a GUI to select the operating system need to be loaded.
- Once user selects the operating system GRUB will pass control to the karnel of that operating system.

4. Kernel

- Once GRUB or LILO transfers the control to Kernel, the Kernels does the following tasks
- Initializes devices and loads initrd module mounts root file system

5. Init

- The kernel, once it is loaded, finds init in sbin(/sbin/init) and executes it.
- Hence the first process which is started in linux is init process.
- This init process reads /etc/inittab file and sets the path, starts swapping, checks the file systems, and so on.
- It runs all the boot scripts(/etc/rc.d/*,/etc/rc.boot/*)
- starts the system on specified run level in the file /etc/inittab

6. Runlevel

- There are 7 run levels in which the linux OS runs and different run levels serves for different purpose. 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 – Unused
  5 – X11
  6 – Reboot
- We can set in which run-level we want to run our operating system by defining it on /etc/inittab file.
- Now as per our setting in /etc/inittab the Operating System the operating system boots up and finishes the boot-up process.
- Below are given some few important differences about LILO and GRUB

LILO	GRUB
LILO has no interactive command interface	GRUB has interactive command interface
LILO does not support booting from a network	GRUB does support booting from a network
If you change your LILO config file, you have to rewrite the LILO stage one boot loader to the MBR	GRUB automatically detects any change in config file and auto loads the OS
LILO supports only linux operating system	GRUB supports large number of OS

What is LILO?

LILO is Linux Loader is a boot loader for Linux. It is used to load Linux into the memory and start the Operating system. LILO can be configured to boot other operating systems as well. LILO is customizable, which means that if the default configuration is not correct, it can be changed. Config file for LILO is lilo.conf.

LILO Configuration

LILO (**LI**nux **LO**ader) is a boot loader for Linux and was the default boot loader for most Linux distributions in the years after the popularity of loading. Today, many distributions use GRUB as the default boot loader, but LILO and its variant ELILO are still in wide use. Further development of LILO was discontinued in December 2015 along with a request by ***Joachim Weidorn*** for potential developers.

LILO does not depend on a specific file system, and can boot an operating system (e.g. Linux kernel images) from floppy disks and hard disks. One of up to sixteen different images can be selected at boot time. Various parameters, such as the root device, can be set independently for each kernel. LILO can be placed in the master boot record (MBR) or the boot sector of a partition. In the latter case, the MBR must contain code to load LILO.

At system start, only the BIOS drivers are available for LILO to access hard disks. For this reason, a very old BIOS access area is limited to cylinders 0 to 1023 of the first two hard disks. For later BIOS, LILO can use 32-bit "**Logical Block Addressing" (LBA)** to access the entire capacity of the hard disks the BIOS has access to.

To Configure LILO File in /etc/lilo.conf File

As with many Linux utilities, 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. In fact, it may be a little too thorough for simple configuration. So here's the lowdown on your basic **lilo.conf** file.

The sample **lilo.conf** file shown below is for a typical dual-boot configuration, with Windows installed on the first partition and Linux on the second. You can probably use this as-is, except for the **image = line** and possibly the **root = line**, depending on where Linux was installed. Detailed explanation follows.

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=50
message=/boot/message
lba32
default=linux
image=/boot/vmlinuz-2.4.0-0.43.6
        label=linux
        initrd=/boot/initrd-2.4.0-0.43.6.img
        read-only
        root=/dev/hda5
other=/dev/hda1
        label=dos
```

- **boot=/dev/hda** - Instructs LILO to be installed on the first hard disk of the first IDE controller.
- **map=/boot/map** - Locates the map file. In normal use, this should not be modified.
- **install=/boot/boot.b** - Instructs LILO to install the specified file as the new boot sector. In normal use, this should not be altered. If the install line is missing, LILO assumes a default of /boot/boot.b as the file to be used.
- **prompt -** Instructs LILO to show you whatever is referenced in the message line. While it is not recommended that you remove the prompt line, if you do remove it, you can still access a prompt by holding down the [Shift] key while your machine starts to boot.
- **timeout=50** - Sets the amount of time that LILO waits for user input before proceeding with booting the default line entry. This is measured in tenths of a second, with 50 as the default.
- **message=/boot/message -** Refers to the screen that LILO displays to let you select the operating system or kernel to boot.
- **lba32 -** Describes the hard disk geometry to LILO. Another common entry here is linear. You should not change this line unless you are very aware of what you are doing. Otherwise, you could put your system in an unbootable state.
- **default=linux -** Refers to the default operating system for LILO to boot as seen in the options listed below this line. The name linux refers to the label line below in each of the boot options.
- **image=/boot/vmlinuz-2.4.0-0.43.6 -** Specifies which Linux kernel to boot with this particular boot option.
- **label=linux -** Names the operating system option in the LILO screen. In this case, it is also the name referred to by the default line.
- **initrd=/boot/initrd-2.4.0-0.43.6.img -** Refers to the *initial ram disk* image that is used at boot time to initialize and start the devices that makes booting the kernel possible. The initial ram disk is a collection of machine-specific drivers necessary to operate a SCSI card, hard drive, or any other device needed to load the kernel. You should never try to share initial ram disks between machines.
- **read-only -** Specifies that the root partition (refer to the root line below) is read-only and cannot be altered during the boot process.
- **root=/dev/hda5 -** Specifies which disk partition to use as the root partition.
- **other=/dev/hda1 -** Specifies the partition containing DOS.

GRUB Configuration

The configuration file (/boot/grub/grub.conf), which is used to create the list of operating systems to boot in GRUB's menu interface, essentially allows the user to select a pre-set group of commands to execute. The commands given in GRUB Commands can be used, as well as some special commands that are only available in the configuration file.

Configuration File Structure

The GRUB menu interface configuration file is /boot/grub/grub.conf. The commands to set the global preferences for the menu interface are placed at the top of the file, followed by stanzas for each operating kernel or operating system listed in the menu.

The following is a very basic GRUB menu configuration file designed to boot either Red Hat Enterprise Linux:

```
default=0
timeout=10
splashimage=(hd0,0)/grub/splash.xpm.gz

# section to load Linux
title Red Hat Enterprise Linux (2.4.21-1.ent)
    root (hd0,0)
    kernel /vmlinuz-2.4.21-1 ro root=/dev/sda2
    initrd /initrd-2.4.21-1.img

# section to load Windows
title Windows
    rootnoverify (hd0,0)
    chainloader +1
```

This file configures GRUB to build a menu with Red Hat Enterprise Linux as the default operating system and sets it to auto-boot after 10 seconds. Two sections are given, one for each operating system entry, with commands specific to the system disk partition table.

Configuring a GRUB menu configuration file to boot multiple operating systems is beyond the scope of this chapter. Consult Additional Resources for a list of additional resources.

Configuration File Directives

The following are directives commonly used in the GRUB menu configuration file:

- **hainloader *</path/to/file>*** — Loads the specified file as a chain loader. Replace *</path/to/file>* with the absolute path to the chain loader. If the file is located on the first sector of the specified partition, use the blocklist notation, +1.
- **color *<normal-color> <selected-color>*** — Allows specific colors to be used in the menu, where two colors are configured as the foreground and background. Use simple color names such as red/black.

  For example: **color red/black green/blue**
- **default=*<integer>*** — Replace *<integer>* with the default entry title number to be loaded if the menu interface times out.
- **fallback=*<integer>*** — Replace *<integer>* with the entry title number to try if the first attempt fails.
- **hiddenmenu** — Prevents the GRUB menu interface from being displayed, loading the default entry when the timeout period expires. The user can see the standard GRUB menu by pressing the [Esc] key.
- **initrd *</path/to/initrd>*** — Enables users to specify an initial RAM disk to use when booting. Replace *</path/to/initrd>* with the absolute path to the initial RAM disk.
- **kernel *</path/to/kernel> <option-1> <option-N>*** — Specifies the kernel file to load when booting the operating system. Replace *</path/to/kernel>* with an absolute path from the partition specified by the root directive. Multiple options can be passed to the kernel when it is loaded.
- **password=*<password>*** — Prevents a user who does not know the password from editing the entries for this menu option.

  Optionally, it is possible to specify an alternate menu configuration file after the password=*<password>* directive. In this case, GRUB restarts the second stage boot

loader and uses the specified alternate configuration file to build the menu. If an alternate menu configuration file is left out of the command, a user who knows the password is allowed to edit the current configuration file.

- **root (*<device-type><device-number>,<partition>*)** — Configures the root partition for GRUB, such as (hd0,0), and mounts the partition.
- **rootnoverify (*<device-type><device-number>,<partition>*)** — Configures the root partition for GRUB, just like the root command, but does not mount the partition.
- **timeout=*<integer>*** — Specifies the interval, in seconds, that GRUB waits before loading the entry designated in the default command.
- **splashimage=*<path-to-image>*** — Specifies the location of the splash screen image to be used when GRUB boots.
- **title group-title** — Specifies a title to be used with a particular group of commands used to load a kernel or operating system.

UNIT – 5

Linux Admin (Ubuntu)

USER CONFIGURATION

ADD USER

User management is a critical part of maintaining a secure system. Ineffective user and privilege management often lead many systems into being compromised. Therefore, it is important that you understand how you can protect your server through simple and effective user account management techniques.

Ubuntu developers made a conscientious decision to disable the administrative root account by default in all Ubuntu installations. This does not mean that the root account has been deleted or that it may not be accessed. It merely has been given a password which matches no possible encrypted value, therefore may not log in directly by itself.

Instead, users are encouraged to make use of a tool by the name of sudo to carry out system administrative duties. Sudo allows an authorized user to temporarily elevate their privileges using their own password instead of having to know the password belonging to the root account. This simple yet effective methodology provides accountability for all user actions, and gives the administrator granular control over which actions a user can perform with said privileges.

1. If for some reason you wish to enable the root account, simply give it a password: Configurations with root passwords are not supported.
   - sudo passwd

   Sudo will prompt you for your password, and then ask you to supply a new password for root as shown below:
   - [sudo] password for username: (enter your own password)
   - Enter new UNIX password: (enter a new password for root)
   - Retype new UNIX password: (repeat new password for root)
   - passwd: password updated successfully
2. To disable the root account password, use the following passwd syntax:
   - sudo passwd -l root

   However, to disable the root account itself, use the following command:
   - usermod --expiredate 1
3. You should read more on Sudo by reading the man page:
   - man sudo

By default, the initial user created by the Ubuntu installer is a member of the group "sudo" which is added to the file /etc/sudoers as an authorized sudo user. If you wish to give any other account full root access through sudo, simply add them to the sudo group

Adding and Deleting Users

The process for managing local users and groups is straightforward and differs very little from most other GNU/Linux operating systems. Ubuntu and other Debian based distributions encourage the use of the "adduser" package for account management.

1. To add a user account, use the following syntax, and follow the prompts to give the account a password and identifiable characteristics, such as a full name, phone number, etc.

- sudo adduser username

2. To delete a user account and its primary group, use the following syntax:
   - sudo deluser username

   Deleting an account does not remove their respective home folder. It is up to you whether or not you wish to delete the folder manually or keep it according to your desired retention policies.

   Remember, any user added later on with the same UID/GID as the previous owner will now have access to this folder if you have not taken the necessary precautions.

   You may want to change these UID/GID values to something more appropriate, such as the root account, and perhaps even relocate the folder to avoid future conflicts:
   - sudo chown -R root:root /home/username/
   - sudo mkdir /home/archived_users/
   - sudo mv /home/username /home/archived_users/
3. To temporarily lock or unlock a user account, use the following syntax, respectively:
   - sudo passwd -l username
   - sudo passwd -u username
4. To add or delete a personalized group, use the following syntax, respectively:
   - sudo addgroup groupname
   - sudo delgroup groupname
5. To add a user to a group, use the following syntax:
   - sudo adduser username groupname

User Profile Security

When a new user is created, the adduser utility creates a brand new home directory named /home/username. The default profile is modeled after the contents found in the directory of /etc/skel, which includes all profile basics.

If your server will be home to multiple users, you should pay close attention to the user home directory permissions to ensure confidentiality. By default, user home directories in Ubuntu are created with world read/execute permissions. This means that all users can browse and access the contents of other users home directories. This may not be suitable for your environment.

1. To verify your current user home directory permissions, use the following syntax:
   - ls -ld /home/username

   The following output shows that the directory /home/username has world-readable permissions:
   - drwxr-xr-x 2 username username 4096 2007-10-02 20:03 username
2. You can remove the world readable-permissions using the following syntax:
   - sudo chmod 0750 /home/username

   Some people tend to use the recursive option (-R) indiscriminately which modifies all child folders and files, but this is not necessary, and may yield other undesirable results. The parent directory alone is sufficient for preventing unauthorized access to anything below the parent.

   A much more efficient approach to the matter would be to modify the adduser global default permissions when creating user home folders. Simply edit the file /etc/adduser.conf and modify the DIR_MODE variable to something appropriate, so that all new home directories will receive the correct permissions.

- DIR_MODE=0750

3. After correcting the directory permissions using any of the previously mentioned techniques, verify the results using the following syntax:
   - ls -ld /home/username
4. The results below show that world-readable permissions have been removed:
   - drwxr-x--- 2 username username 4096 2007-10-02 20:03 username

Password Policy

A strong password policy is one of the most important aspects of your security posture. Many successful security breaches involve simple brute force and dictionary attacks against weak passwords. If you intend to offer any form of remote access involving your local password system, make sure you adequately address minimum password complexity requirements, maximum password lifetimes, and frequent audits of your authentication systems.

By default, Ubuntu requires a minimum password length of 6 characters, as well as some basic entropy checks. These values are controlled in the file /etc/pam.d/common-password, which is outlined below.

- password [success=1 default=ignore] pam_unix.so obscure sha512

If you would like to adjust the minimum length to 8 characters, change the appropriate variable to min=8. The modification is outlined below.

- password [success=1 default=ignore] pam_unix.so obscure sha512 minlen=8

Creating a User Account and Password

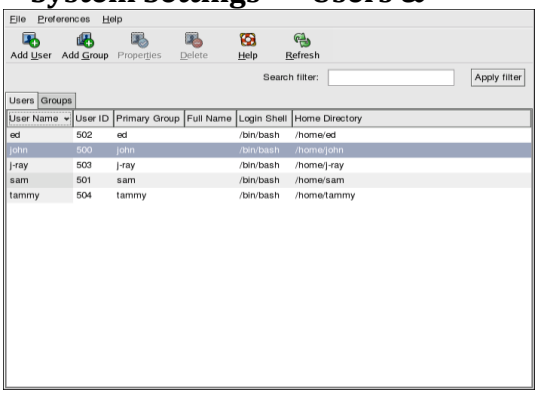
When you first started your Linux system after installation, you were given the opportunity to create one or more user accounts using the **Setup Agent**. If you did not create at least one account (not including the root account) you should do so now. You should avoid working in the root account for daily tasks.

There are two ways to create new and/or additional user accounts: using the graphical **User Manager** application or from a shell prompt.

To create a user account graphically using the **User Manager**:

1. Select **Applications** (the main menu on the panel) => **System Settings** => **Users & Groups** from the panel.

   You can also start the **User Manager** by typing redhat-config-users at a shell prompt.
2. If you are not logged in as root, you will be prompted for your root password.
3. The window shown in Figure__will appear. Click **Add User**.
4. In the **Create New User** dialog box, enter a username (this can be an abbreviation or nickname), the full name of the user for whom this account is being created, and a password (which you will enter a second time for verification). The name of this user's home directory and the name of the login shell should appear by default. For most users, you can accept the defaults for the other configuration options. Refer to the ***Linux System Administration Guide*** for details about additional options.
5. Click **OK**. The new user will appear in the user list, signaling that the user account creation is complete.



To create a user account from a shell prompt:

1. Open a shell prompt.
2. If you are not logged in as root, type the command su - and enter the root password.
3. Type **useradd** followed by a space and the username for the new account you are creating at the command line (for example, **useradd *jjkcc***). Press [Enter]. Often, usernames are variations on the user's name, such as **jjkcc** User account names can be anything from the user's name, initials, or birthplace to something more creative.
4. Type passwd followed by a space and the username again (for example, passwd jjkcc).
5. At the New password: prompt enter a password for the new user and press [Enter].
6. At the Retype new password: prompt, enter the same password to confirm your selection.

Step by Step

1. Click the icon at the far right of the menu bar and select System Settings.
2. Open User Accounts.
3. You need administrator privileges to add user accounts. Click Unlock in the top right corner and type your password.
4. In the list of accounts on the left, click the + button to add a new user account.
5. If you want the new user to have administrative access to the computer, select Administrator for the account type. Administrators can do things like add and delete users, install software and drivers, and change the date and time.
6. Enter the new user's full name. The username will be filled in automatically based on the full name. The default is probably OK, but you can change it if you like.
7. Click Create.
8. The account is initially disabled until you choose what to do about the user's password. Under Login Options click Account disabled next to Password. Select **Set a password** now from the Action drop-down list, and have the user type their password in the New password and Confirm password fields.
9. You can also click the button next to the **New password** field to select a randomly generated secure password. These passwords are hard for others to guess, but they can be hard to remember, so be careful.
10. Click

Installing and Managing Samba server

Samba is a software package which helps to share files between linux and windows, This post explains, how to share files between ubuntu and windows using samba. This is a very basic guide only for beginners.

Samba installation on ubuntu

Open terminal (CTRL + ALT + t) and follow the steps.

Step 1 » Update the respository .

krizna@leela:~$ sudo apt-get update

Step 2 » Install samba by typing the below command.

krizna@leela:~$ sudo apt-get install samba samba-common

Step 3 » Install GUI admin tool and it's dependencies.

krizna@leela:~$ sudo apt-get install python-glade2 system-config-sambanow

installation is over.

Samba configuration on ubuntu

Step 4 » Create a user with restricted shell access.

krizna@leela:~$ sudo useradd -s /bin/false myshare

Here **myshare** is the **username**.

Step 5 » create samba password for the user **myshare**.

krizna@leela:~$ sudo smbpasswd -a myshare

New SMB password:

Retype new SMB password:

Step 6 » Create a folder for sharing .

krizna@leela:~$ sudo mkdir /shareyou

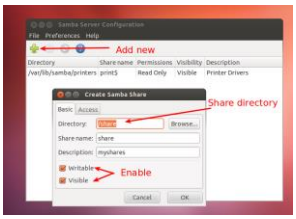
can also create in the user home directory .

Step 7 » Modify ownership of the share folder .

(Not required if the folder is created in home directory).

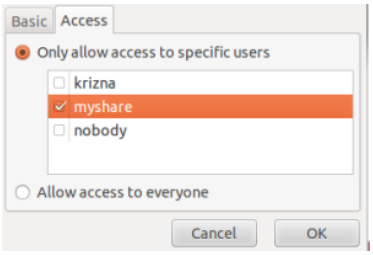
krizna@leela:~$ sudo chown -R myshare /share

Step 8 » Now open the GUI Samba admin tool

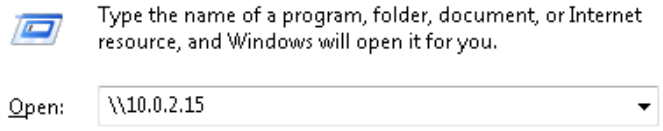
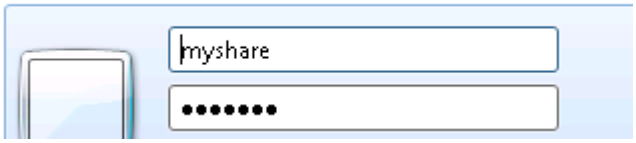
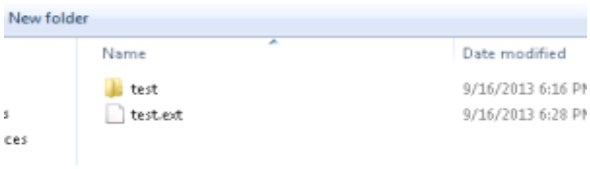


Step 9 » Click Add new share and enter the share details under basic tab like below image.

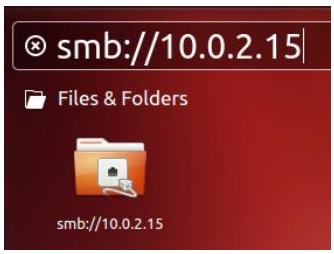
Step 10 » Add the username (**myshare**) for your share under access tab and click **ok**.

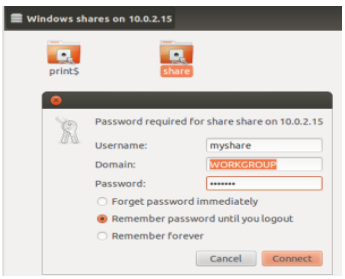


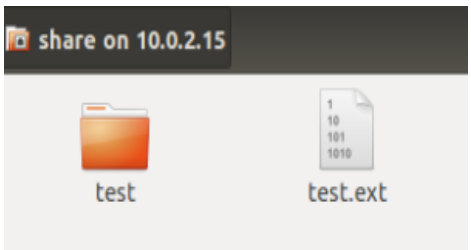
Step 11 » Now you can able to access you share from windows machines using ip address.

STEP -1	
STEP -2	
STEP -3	

Step 12 » From ubuntu







Installing and Managing Apache Server

The Apache HTTP server is the most widely-used web server in the world. It provides many powerful features including dynamically loadable modules, robust media support, and extensive integration with other popular software. Here, we'll explain how to install an Apache web server on your Ubuntu server.

Before you begin this guide, you should have a regular, non-root user with **sudo** privileges configured on your server. Additionally, you will need to enable a basic firewall to block non-essential ports. You can learn how to configure a regular user account and set up a firewall for your server by following our initial server setup guide for Ubuntu.

When you have an account available, log in as your non-root user to begin.

Step 1 - Installing Apache

Apache is available within Ubuntu's default software repositories, making it possible to install it using conventional package management tools.

Let's begin by updating the local package index to reflect the latest upstream changes:

- **sudo apt update**

Then, install the apache2 package:

- **sudo apt install apache2**

After confirming the installation, apt will install Apache and all required dependencies.

Step 2 - Adjusting the Firewall

Before testing Apache, it's necessary to modify the firewall settings to allow outside access to the default web ports. Assuming that you followed the instructions in the prerequisites, you should have a UFW firewall configured to restrict access to your server.

During installation, Apache registers itself with UFW to provide a few application profiles that can be used to enable or disable access to Apache through the firewall.

List the ufw application profiles by typing:

sudo ufw app list

You will see a list of the application profiles:

Output

Available applications:
Apache
Apache Full
Apache Secure
OpenSSH

As you can see, there are three profiles available for Apache:

- **Apache**: This profile opens only port 80 (normal, unencrypted web traffic)
- **Apache Full**: This profile opens both port 80 (normal, unencrypted web traffic) and port 443 (TLS/SSL encrypted traffic)
- **Apache Secure**: This profile opens only port 443 (TLS/SSL encrypted traffic)

It is recommended that you enable the most restrictive profile that will still allow the traffic you've configured. Since we haven't configured SSL for our server yet in this guide, we will only need to allow traffic on port 80:

sudo ufw allow 'Apache'

You can verify the change by typing:

sudo ufw status

You should see HTTP traffic allowed in the displayed output:

Output

```
Status: active
To                         Action      From
--                         ------      ----
OpenSSH                    ALLOW       Anywhere
Apache                     ALLOW       Anywhere
OpenSSH (v6)               ALLOW       Anywhere (v6)
Apache (v6)                ALLOW       Anywhere (v6)
```

As you can see, the profile has been activated to allow access to the web server.

Step 3 - Checking your Web Server

At the end of the installation process, Ubuntu starts Apache. The web server should already be up and running.

Check with the systemd init system to make sure the service is running by typing:

sudo systemctl status apache2

Output

```
apache2.service - The Apache HTTP Server
  Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor preset:
enabled)
 Drop-In: /lib/systemd/system/apache2.service.d
          └─apache2-systemd.conf
  Active: active (running) since Tue 2018-04-24 20:14:39 UTC; 9min ago
Main PID: 2583 (apache2)
   Tasks: 55 (limit: 1153)
  CGroup: /system.slice/apache2.service
          ├─2583 /usr/sbin/apache2 -k start
          ├─2585 /usr/sbin/apache2 -k start
          └─2586 /usr/sbin/apache2 -k start
```

As you can see from this output, the service appears to have started successfully. However, the best way to test this is to request a page from Apache.

You can access the default Apache landing page to confirm that the software is running properly through your IP address. If you do not know your server's IP address, you can get it a few different ways from the command line. Try typing this at your server's command prompt:

hostname -I

You will get back a few addresses separated by spaces. You can try each in your web browser to see if they work.

An alternative is typing this, which should give you your public IP address as seen from another location on the internet:

curl -4 icanhazip.com

When you have your server's IP address, enter it into your browser's address bar:http://your_server_ip You should see the default Ubuntu Apache web page:

This page indicates that Apache is working correctly. It also includes some basic information about important Apache files and directory locations.

Step 4 - Managing the Apache Process

Now that you have your web server up and running, let's go over some basic management commands.

To stop your web server, type:

sudo systemctl stop apache2

To start the web server when it is stopped, type:

sudo systemctl start apache2

To stop and then start the service again, type:

sudo systemctl restart apache2

If you are simply making configuration changes, Apache can often reload without dropping connections. To do this, use this command:

sudo systemctl reload apache2

By default, Apache is configured to start automatically when the server boots. If this is not what you want, disable this behavior by typing:

sudo systemctl disable apache2

To re-enable the service to start up at boot, type:

sudo systemctl enable apache2

Apache should now start automatically when the server boots again.

Step 5 - Setting Up Virtual Hosts (Recommended)

When using the Apache web server, you can use *virtual hosts* (similar to server blocks in Nginx) to encapsulate configuration details and host more than one domain from a single server. We will set up a domain called **example.com**, but you should **replace this with your own domain name**. To learn more about setting up a domain name with DigitalOcean,

Apache on Ubuntu has one server block enabled by default that is configured to serve documents from the **/var/www/html directory**. While this works well for a single site, it can become unwieldy if you are hosting multiple sites. Instead of modifying **/var/www/html,** let's create a directory structure within **/var/www** for our **example.com** site, leaving **/var/www/html** in place as the default directory to be served if a client request doesn't match any other sites.

Create the directory for **example.com** as follows, using the -p flag to create any necessary parent directories:

sudo mkdir -p /var/www/example.com/html

Next, assign ownership of the directory with the $USER environmental variable:

sudo chown -R $USER:$USER /var/www/example.com/html

The permissions of your web roots should be correct if you haven't modified your unmask value, but you can make sure by typing:

sudo chmod -R 755 /var/www/example.com

Next, create a sample index.html page using nano or your favorite editor:

nano /var/www/example.com/html/index.html

Inside, add the following sample HTML:

/var/www/example.com/html/index.html

```
<html>
  <head>
    <title>Welcome to Example.com!</title>
  </head>
```

```
  <body>
    <h1>Success!  The example.com server block is working!</h1>
  </body>
</html>
```

Save and close the file when you are finished.

In order for Apache to serve this content, it's necessary to create a virtual host file with the correct directives. Instead of modifying the default configuration file located at /etc/apache2/sites-available/000-default.conf directly, let's make a new one at /etc/apache2/sites-available/example.com.conf:

sudo nano /etc/apache2/sites-available/example.com.conf

Paste in the following configuration block, which is similar to the default, but updated for our new directory and domain name:

/etc/apache2/sites-available/example.com.conf

```
<VirtualHost *:80>
  ServerAdmin admin@example.com
  ServerName example.com
  ServerAlias www.example.com
  DocumentRoot /var/www/example.com/html
  ErrorLog ${APACHE_LOG_DIR}/error.log
  CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

The **<VirtualHost>** directive is used to group the settings used by apache for a specific virtual host. The first thing we saw defined in it, is the *:80 instruction. This indicates the ip address and port used by the virtual host.

Multiple virtual hosts can be defined in the same file or by following the "one virtual host definition per file" scheme. In both cases the first definition is considered to be the default, if no other virtual host is matched by the client request.

The **ServerAdmin** directive is optional, and it is used to specify the contact address that the web server will show in case of error messages. Normally we want to provide a valid email address as the argument of this directive, since the web server will use mailto: on it, to make contacting the administrator easier.

DocumentRoot is mandatory and it's essential for the virtual host configuration. The argument to this instruction must be a valid file-system path. The directory provided will be considered the root directory of the virtual host, and must not contain a trailing '/'. In this case, the document root directory its /var/www/html. If we take a look at its content, we see that it contains the index.html page used as the server welcome page we saw before.

The last two instructions is provided in this virtual-host are Error-Log and CustomLog. By using the first one, we set the file to which the server will log the occurring errors. The second instead is used to log the requests sent to the server in the specified format.

Notice that we've updated the DocumentRoot to our new directory and ServerAdmin to an email that the **example.com** site administrator can access. We've also added two directives: ServerName, which establishes the base domain that should match for this virtual host definition, and ServerAlias, which defines further names that should match as if they were the base name.

Save and close the file when you are finished.
Let's enable the file with the a2ensite tool:

sudo a2ensite example.com.conf

Disable the default site defined in 000-default.conf:

sudo a2dissite 000-default.conf

Next, let's test for configuration errors:

sudo apache2ctl configtest

You should see the following output:

Output

Syntax OK

Restart Apache to implement your changes:

sudo systemctl restart apache2

Apache should now be serving your domain name. You can test this by navigating to http://example.com, where you should see something like this:

Success! The example.com virtual host is working!

Step 6 – Getting Familiar with Important Apache Files and Directories

Now that you know how to manage the Apache service itself, you should take a few minutes to familiarize yourself with a few important directories and files.

Content

- /var/www/html: The actual web content, which by default only consists of the default Apache page you saw earlier, is served out of the /var/www/html directory. This can be changed by altering Apache configuration files.

Server Configuration

- /etc/apache2: The Apache configuration directory. All of the Apache configuration files reside here.
- /etc/apache2/apache2.conf: The main Apache configuration file. This can be modified to make changes to the Apache global configuration. This file is responsible for loading many of the other files in the configuration directory.
- /etc/apache2/ports.conf: This file specifies the ports that Apache will listen on. By default, Apache listens on port 80 and additionally listens on port 443 when a module providing SSL capabilities is enabled.
- /etc/apache2/sites-available/: The directory where per-site virtual hosts can be stored. Apache will not use the configuration files found in this directory unless they are linked to the sites-enabled directory. Typically, all server block configuration is done in this directory, and then enabled by linking to the other directory with the a2ensite command.
- /etc/apache2/sites-enabled/: The directory where enabled per-site virtual hosts are stored. Typically, these are created by linking to configuration files found in the sites-available directory with the a2ensite. Apache reads the configuration files and links found in this directory when it starts or reloads to compile a complete configuration.
- /etc/apache2/conf-available/, /etc/apache2/conf-enabled/: These directories have the same relationship as the sites-available and sites-enabled directories, but are used to store configuration fragments that do not belong in a virtual host. Files in the conf-available directory can be enabled with the a2enconf command and disabled with the a2disconf command.

- /etc/apache2/mods-available/, /etc/apache2/mods-enabled/: These directories contain the available and enabled modules, respectively. Files in ending in .load contain fragments to load specific modules, while files ending in .conf contain the configuration for those modules. Modules can be enabled and disabled using the a2enmod and a2dismod command.

Server Logs

- /var/log/apache2/access.log: By default, every request to your web server is recorded in this log file unless Apache is configured to do otherwise.
- /var/log/apache2/error.log: By default, all errors are recorded in this file. The LogLeveldirective in the Apache configuration specifies how much detail the error logs will contain.

What is LDAP?

Lightweight **D**irectory **A**ccess **P**rotocol **(LDAP)** is a network protocol for accessing and manipulating information stored in a directory.

Services built on the LDAP protocol are used to serve a wide range of information. The protocol is well-suited to serving information that must be highly available and accessible, but does not change frequently. Common applications include:

1. Centralization of user and group information as part of Single Sign On (SSO).
2. Authenticate users in a web application.
3. Create a shared address directory for mail agents.
4. Authenticate users locally

Before starting this tutorial, you should have an Ubuntu server set up with Apache and PHP. You can follow our tutorial How To Install Linux, Apache, MySQL, PHP (LAMP) stack on Ubuntu, skipping Step 2 as we will not need the MySQL database server.

Additionally, since we will be entering passwords into the web interface, we should secure Apache with SSL encryption. Read How To Secure Apache with Let's Encrypt on Ubuntu to download and configure free SSL certificates. You will need a domain name to complete this step. We will use these same certificates to provide secure LDAP connections as well.

Steps to Setup Open LDAP

1. Download Open LDAP version 2.4.40 from its official website by following command:
   - **wget ftp: // ftp.openldap.org /pub / OpenLDAP / openldap-release / openldap-2.4.40.tgz**

2. Unzip downloaded file
   - **tar zxf openldap-2.4.40.tgz**

3. Configure Open LDAP

   Now Navigate to your unzipped openldap directory by executing "cd openldap" and then Run " ./configure "

   command in terminal. Most probably you will get an error as *"configure: error : BDB/HDB: BerkeleyDB not available"* .

   To fix this we need to install BerkeleyDB. To install Berkeley DB follow command below:
   - **sudo apt-get install libdb-dev**

4. Re-run ./configure command after successful installation of Berkeley DB as follows :
   - **./configure**
   - **Then execute command as follows**:
     - make depend
     - make
5. Now install software by following command :
   - **sudo make install**
6. Create Directory for your base dn which will be used to keep BDB files and indices for your domain.
   - **sudo mkdir /usr/local/var/openldap-data/your-domain.com**
7. OpenLDAP with a BDB backend requires a DB_CONFIG file for configuration and tuning of database. This file exists in / usr / local / etc / openldap / DB_CONFIG.example. Copy this file to your base directory and rename it as DB_CONFIG as follows.
   - **sudo cp /usr/local/etc/openldap/DB_CONFIG.example / usr / local / var / openldap-data / your-domain.com/**
   - **sudo mv /usr/local/var/openldap-data/your-domain.com / DB_CONFIG.example /usr/local/var/openldap-data/your-domain.com / DB_CONFIG**

NOTE : You can also change configuration of DB_CONFIG file for optimum performance and tuning of database.

In our set up, we have kept default configuration in DB_CONFIG file.

8. Execute "sudo nano /usr/local/etc/openldap/slapd.conf" and Edit slapd.conf file to specify connection information of your LDAP as follows :
   - **# Specify your LDAP connection information here**
   - **suffix "dc=your-domain,dc=com"**
   - **rootdn "cn=Manager,dc=your-domain,dc=com"**
   - **rootpw secret**

   Then change following line to specify the base directory of your LDAP where the BDB files containing the database and associated indices live. In our setup, our base directory is /usr/local/var/openldap-data/your-domain.com
   - **directory /usr/local/var/openldap-data/your-domain.com**
9. Start LDAP
   - **Goto /usr/local/libexec directory, and execute following**
     - **sudo ./slapd**
10. Check slapd is running or not by following command:
    - **ldapsearch -x -b '' -s base '(objectclass=*)' namingContexts**

    This command should return your base dn as follows:
    - **dn : namingContexts: dc=your-domain,dc=com**
11. Open slapd.conf file by executing "sudo nano /usr/local/etc/openldap/slapd.conf" and add following lines in it to add required schema in LDAP :
    - **include /usr/local/etc/openldap/schema/cosine.schema**
    - **include /usr/local/etc/openldap/schema/inetorgperson.schema**
12. Then Restart slapd as follows:
    - To stop LDAP use following command :
      - **sudo kill -INT `cat /usr/local/var/run/slapd.pid`**

- Then to start LDAP, Goto /usr/local/libexec directory, and execute following
  - **sudo ./slapd**

13. Now you are able to create Entries in LDAP from Apache Directory Studio.

- Open Apache directory studio and click on new connection. Enter Hostname=<your-host-name>, port = 389 & click Next.
- Enter Bind DN or user and Bind password as per your configuration in slapd.conf. file.
- In our example, it is cn=Manager, dc= your-domain.com, dc=com and secret respectively.
- Click on Finish.

Add Slapd as System Service

- If LDAP is installed by util commands then init script for slapd is automatically created at time of installation. But if you have installed slapd by downloading zip file, then you have to to add slapd as system service manually.
- To Manually add slapd as system service do as follows:

1. First create slapd service file as follows:
   - sudo touch /etc/init.d/slapd
   - cd /etc/init.d
   - sudo nano slapd
2. Download slapd service file attached at the end of this post. Copy contents of that file and paste it in your service file.
3. Open slapd service file and check if following paths are properly set or not for your LDAP.
   - **SLAPD=/usr/local/libexec/slapd**
   - **SLAPD_CONF=/usr/local/etc/openldap/slapd.conf**
4. Now give necessary permission to service file as follows:
   - sudo chmod +x /etc/init.d/slapd
5. Configure the service in init levels as follows:
   - **sudo ln -s /etc/init.d/slapd /etc/rc3.d/S90slapd**
   - **sudo ln -s /etc/init.d/slapd /etc/rc4.d/S90slapd**
   - **sudo ln -s /etc/init.d/slapd /etc/rc5.d/S90slapd**
   - **sudo ln -s /etc/init.d/slapd /etc/rc0.d/K10slapd**
   - **sudo ln -s /etc/init.d/slapd /etc/rc6.d/K10slapd**
6. To enable System Startup links run following command:
   - **sudo update-rc.d slapd defaults**
7. Restart slapd as follows:
   - **sudo service slapd restart**
8. Now LDAP is setup as your system service, so use following commands to start and stop and restart LDAP Server.
   - **sudo service slapd start**
   - **sudo service slapd stop**
   - **sudo service slapd restart**

Enable Separate Logging for Open LDAP

1. Execute "sudo nano /etc/rsyslog.d/50-default.conf" and edit following line as follows to enable Open LDAP logging. :
   .;auth,authpriv,local4.none -/var/log/syslog
   Add following line there to enable generation of Open LDAP logs in separate file:
   local4.* -/var/log/openldap/openldap
2. Create configuration file to manage Open LDAP log rotation by executing "sudo touch / etc / logrotate. d /openldap" and add following configuration in it . You can change this configuration as per your requirements :

```
/var/log/openldap/openldap
{
# Add extension of date with name of log file
dateext
# Specify format of date appended with name of the log file
dateformat -%Y-%m-%d.log
# Keep 365 days worth of backlogs
rotate 365
# Rotate log files daily
daily
# If the log file is missing, go on to the next one without issuing an error message.
missingok
# Postpone compression of the previous log file until next rotation of logs. This only works when used in combination with
# compress
delaycompress
# Compress old log files
compress
# Lines between postrotate and endscript are executed after the log file is rotated. Following line stands for gracefully restart
# of rsyslog service after log rotation.
postrotate
invoke-rc.d rsyslog reload > /dev/null
endscript
}
```

3. Open /etc/logrotate.conf file and uncomment following line to enable compression of all log files generated by rsyslog service :
   - compress
4. Then restart "rsyslog" and "slapd" service with following commands respectively :
   - sudo service rsyslog restart
   - sudo service slapd restart

What is DNS?

Here is an easy way to understand the DNS and how it works. If we need to access tecmint.com in browser, the system will look for tecmint.com. Here at the end of the .comthere will be a (.) so what is this ?.

The (.) represent the namespace Root server, there are total 13 root servers globally available. While we accessing tecmint.com it will ask to name server as per

operating system configuration. In Ubuntu, we used to configure the name-server in /etc/resolv.conf, while accessing tecmint.com my browser will ask to root name-servers, if the root name-server don't have my requested domain information it will cache my requested information and forward my request to (TLD) Top Level Domain name-server, even in TLD name-server my request is not available it will be cached and forwarded to Authoritative name-server.

While the domain registration, our domain registered will define which authoritative name-server should our domain use. So, authoritative name servers have our domain information's, while our request reach ANS it will reply for the query that tecmint.com have 111.111.222.1 at the same time it will be cached in Authoritative name-server and send the request back to browser. Every above step is done within milliseconds.

Hope you got what is DNS now, and how it works. Now let us set up a Caching DNS Server in Ubuntu Server LTS.

Step 1: Installing DNS Server

First, take a look at the my local DNS server information such as static IP address and hostname, which is used to for this article purpose.

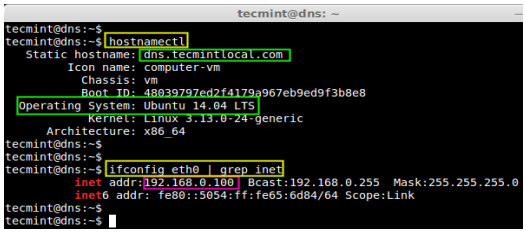
IP Address: 192.168.0.100

Hostname: dns.tecmintlocal.com

To verify that the above settings are correct, we can use 'hostnamectl' and 'ifconfig' commands.

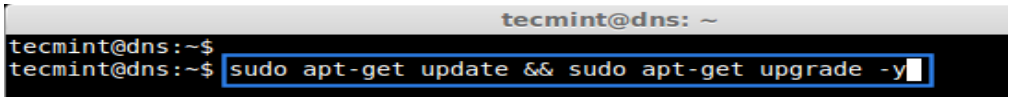
$ hostnamectl

$ ifconfig eth0 | grep inet



Verify System Hostname Next, we update the default repositories and do a system upgrade, before setting-up DNS cache server.

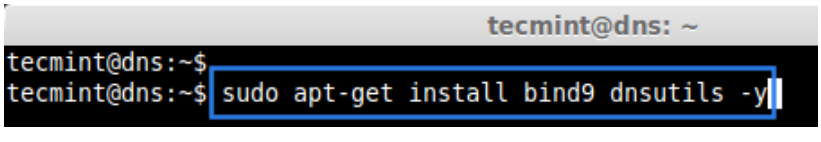
$ sudo apt-get update && sudo apt-get upgrade –y



Upgrade Ubuntu

Now, install the DNS Packages bind and dnsutils using the following command.

$ sudo apt-get install bind9 dnsutils -y

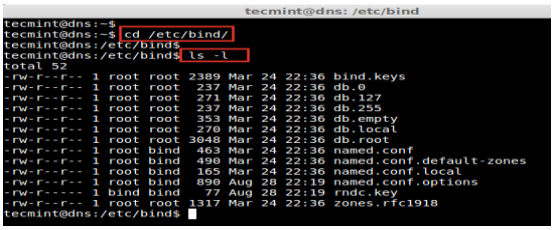


Install DNS Serve

Once, dns installed, move to the bind configuration directory, under /etc/bind.

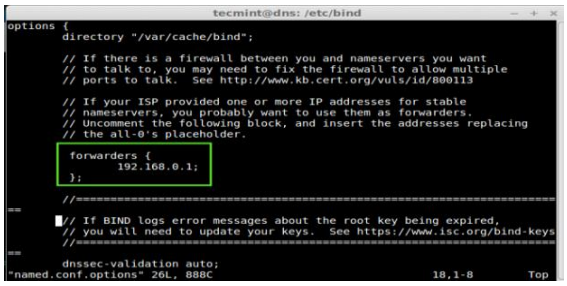
$ /etc/bind/

$ ls -l



Bind Configuration

Step 2: Setting DNS Cache Server



First of all, we setup and configure caching server here. Open and edit the file named.conf.options using vim editor.

$ sudo vim named.conf.options

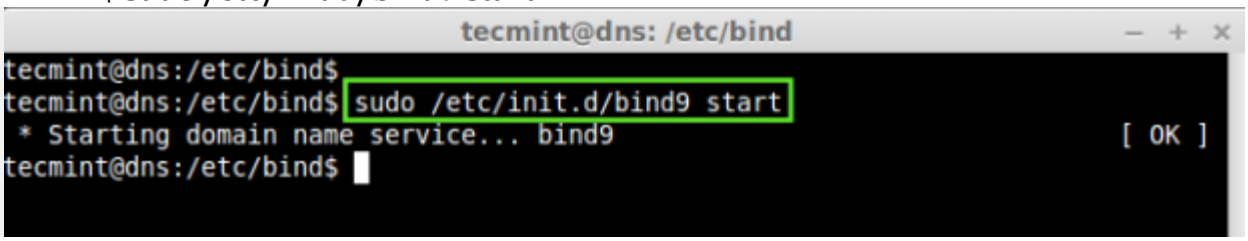
Now, here the word 'forwarders' is used to cache domain name requests. So, here we are going to use my router as the forwarder. Uncomment the // in-front of the line's as shown in the picture.

forwarders {
192.168.0.1;
};

Named Configuration

Save and exit the file using wq!. Now its time to start the bind server for a small testing.

$ sudo /etc/init.d/bind9 start

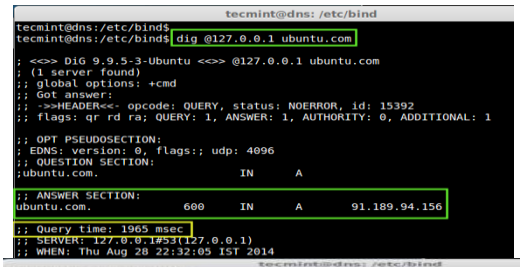


Start DNS Server

If we need to test whether caching works, we can use dig command and check whether the cache working or not.

For example purpose, we going to dig ubuntu.com now, at first, it will won't be cache, so it may may take some milliseconds, once it cached it will be in lightning speed.

$ dig @127.0.0.1 ubuntu.com



Query DNS Lookups

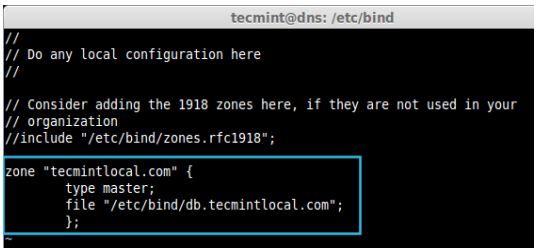
Here, we can see in the above image at first dig it took 1965 milliseconds for my query and shows which ipaddress is binded to ubuntu.com. Let us try for one more dig and see the Query time.

DNS Query Time

In the second try we got the query within 5 milliseconds. Hope you know what is caching server now. The above image shows that total 13 root servers are caching Ubuntu.com, because millions of peoples already accessed Ubuntu official site.

Step 3: Setting Master DNS Server

Create a MASTER DNS Server, Here I'm defining the domain name as tecmintlocal.com, edit the file named.conf.local using vim editor.



$ sudo vim /etc/bind/named.conf.local

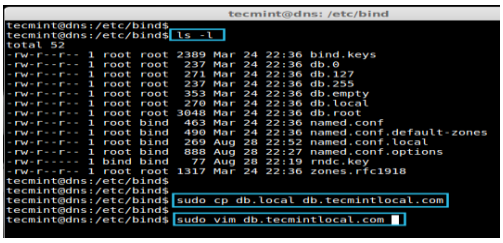
Enter the DNS-Master entry as shown below.

```
zone "tecmintlocal.com" {
    type master;
    file "/etc/bind/db.tecmintlocal.com";
    };
```

zone: Hosts details in Domain
type: Master DNS.
file: Location to store zone information.

Create Zone in DNS

Create the zone file db.tecmintlocal.com (Forward look-ups) from making a copy from db.local.



$ sudo cp db.local db.tecmintlocal.com

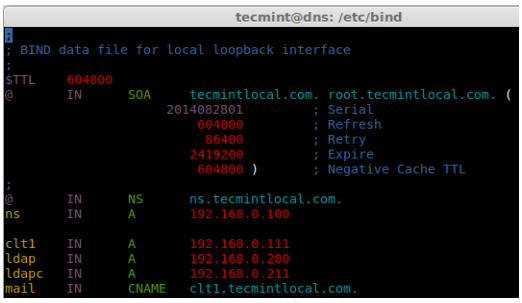
Now open and edit the copied zone file using vim editor.

$ sudo vim db.tecmintlocal.com

Copy Zone Files

Next, add the following example entry, which I have used for tutorial purpose. I use the same for other virtual machine setups too. Modify the below entry as per your requirement.

```
;
; BIND data file for local loopback interface
;`
$TTL    604800
@       IN      SOA     tecmintlocal.com. root.tecmintlocal.com. (
                              2014082801         ; Serial
                                604800         ; Refresh
                                 86400         ; Retry
                               2419200         ; Expire
                                604800 )       ; Negative Cache TTL
;
@       IN      NS      ns.tecmintlocal.com.
ns      IN      A       192.168.0.100
clt1    IN      A       192.168.0.111
ldap    IN      A       192.168.0.200
ldapc   IN      A       192.168.0.211
mail    IN      CNAME   clt1.tecmintlocal.com.
```

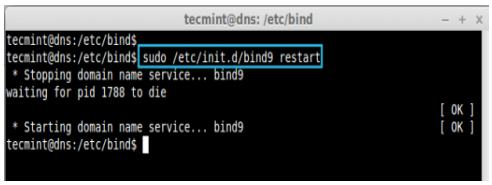


Save and exit the file using wq!..

Create Forward DNS Zone

Finally, restart the bind DNS service using below command.

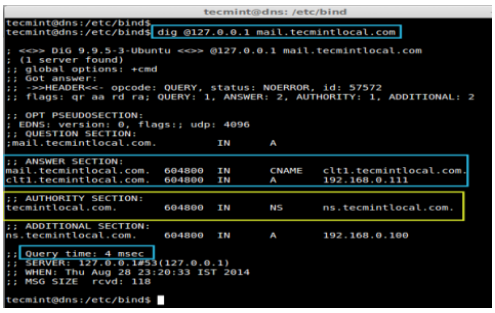
$ sudo service bind9 restart



Start DNS Service

We need to confirm, whether our above zone setup works. Let us check using dig command. Run the command as follows from localhost query.

$ dig @127.0.0.1 mail.tecmintlocal.com

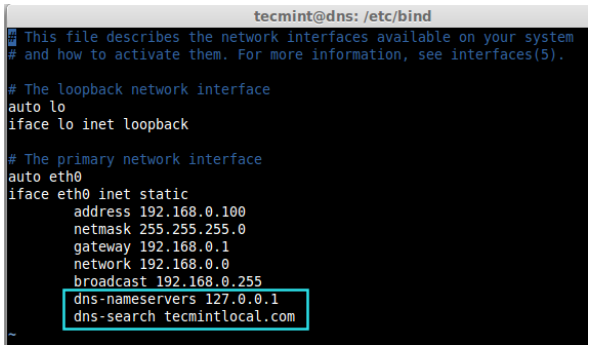


Verify DNS Zone

Let's ping and test the clt1.tecmintlocal.com, before that we need to change the dns-server entry to localhost in our dns server machine and restart the network to get effect.Open and edit the Network interface settings and enter the DNS entry.

$ sudo vim /etc/network/interfaces

Change the DNS entry in the interface as below.



auto lo
iface lo inet loopback
auto eth0
iface eth0 inet static
address 192.168.0.100
netmask 255.255.255.0
gateway 192.168.0.1
network 192.168.0.0
broadcast 192.168.0.255
dns-nameservers 127.0.0.1
dns-search tecmintlocal.com

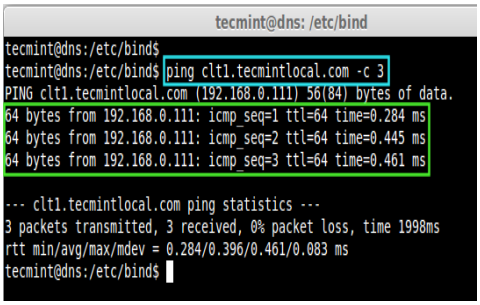
Add DNS Entry

After adding entry, restart the Network using following command.

$ sudo ifdown eth0 && sudo ifup eth0

If restarting network does not take effect, We must need a restart. Now Let's ping and check the clt1.tecmintlocal.com, while its replying, we need to get the ip address what we defined for host-name clt1.

$ ping clt1.tecmintlocal.com -c 3

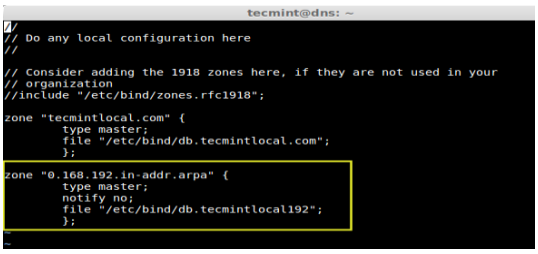


Ping Domain Setting Reverse DNS Lookups

Again open and edit the file named.conf.local.

$ sudo vim /etc/bind/named.conf.local

Now add the following reverse dns lookup entry as shown.



zone "0.168.192.in-addr.arpa" {
type master;
notify no;
file "/etc/bind/db.tecmintlocal192";

```
};
```

Create Reverse DNS

Save and exit the file using wq!. Now create a db.tecmintlocal192 file, as I have mentioned in the master file above for reverse look-up, copy the db.127 to db.tecmintlocal192 using following command.

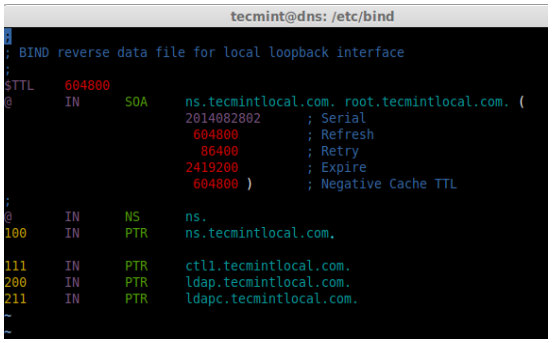
$ sudo cp db.127 db.tecmintlocal192

Now, open and edit a file db.tecmintlocal192 for setup the reverse look-up.

$ sudo vim db.tecmintlocal192

Enter the following entry as below, modify the below entry as per your requirement.

```
;
; BIND reverse data file for local loopback interface
;
$TTL    604800
@       IN      SOA     ns.tecmintlocal.com. root.tecmintlocal.com. (
                              2014082802      ; Serial
                               604800         ; Refresh
                                86400         ; Retry
                              2419200         ; Expire
                               604800 )       ; Negative Cache TTL
;
@       IN      NS      ns.
100     IN      PTR     ns.tecmintlocal.com.
111     IN      PTR     ctl1.tecmintlocal.com.
200     IN      PTR     ldap.tecmintlocal.com.
211     IN      PTR     ldapc.tecmintlocal.com.
```



Reverse DNS Entry

Restart the bind service using.

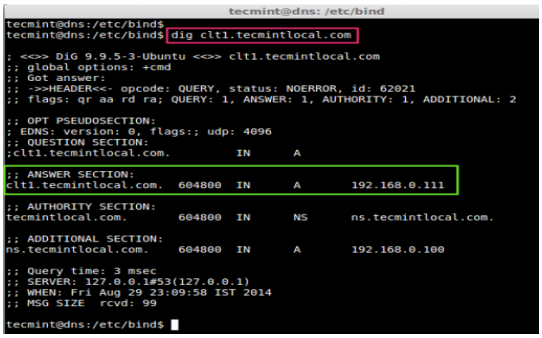
Now, verify the reserve look-up entry.

$ host 192.168.0.111

While we do a reverse look-up using an ip address as shown above, it want to reply with a name as above image shows.

Let's do a check using dig command too.

$ dig clt1.tecmintlocal.com



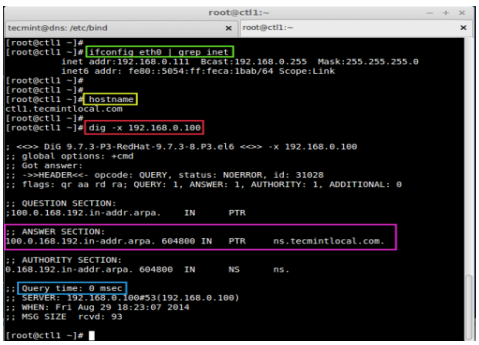
Reverse DNS Lookup

Here, we can see the Answer for our Query in Answer Section as domain-name clt1.tecmintlocal.com has the ip address 192.168.0.111.

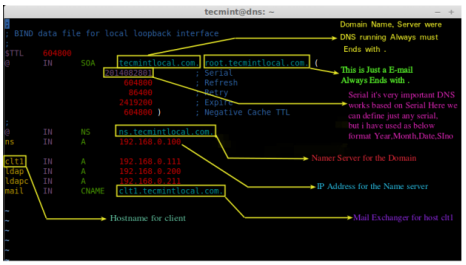
Step 4: Setting Client Machine

Just change the ip address and dns entry in client machine to Our local dns server 192.168.0.100, if so our client machine will get assigned host-name from local **DNS-server.**

Let us check the host-name of our client using following series of commands.



$ ifconfig eth0 | grep inet
$ hostname
$ dig -x 192.168.0.100



Verify Local DNS

Understanding zone file entry in dns, This image will give you a small explanation what we have defined in zone file entry.

OPTIMIZING FTP SERVICES

FTP, short for File Transfer Protocol, is a network protocol that was once widely used for moving files between a client and server. It has since been replaced by faster, more secure, and more convenient ways of delivering files. Many casual Internet users expect to download directly from their web browser with https, and command-line users are more likely to use secure protocols such as the scp or sFTP.

FTP is still used to support legacy applications and workflows with very specific needs. If you have a choice of what protocol to use, consider exploring the more modern options. When you do need FTP, however, vsftpd is an excellent choice. Optimized for security, performance, and stability, vsftpd offers strong protection against many security problems found in other FTP servers and is the default for many Linux distributions.

we'll show you how to configure vsftpd to allow a user to upload files to his or her home directory using FTP with login credentials secured by SSL/TLS.

To follow along with this tutorial you will need: Once you have an Ubuntu server in place, you're ready to begin.

Step 1 - Installing vsftpd

We'll start by updating our package list and installing the vsftpd daemon:

- **sudo apt-get update**
- **sudo apt-get install vsftpd**

When the installation is complete, we'll copy the configuration file so we can start with a blank configuration, saving the original as a backup.

- **sudo cp /etc/vsftpd.conf /etc/vsftpd.conf.orig**

With a backup of the configuration in place, we're ready to configure the firewall.

Step 2 - Opening the Firewall

We'll check the firewall status to see if it's enabled. If so, we'll ensure that FTP traffic is permitted so you won't run into firewall rules blocking you when it comes time to test.

- **sudo ufw status**

In this case, only SSH is allowed through:

Output
Status: active
To Action From
-- ------ ----
OpenSSH ALLOW Anywhere
OpenSSH (v6) ALLOW Anywhere (v6)

You may have other rules in place or no firewall rules at all. Since only ssh traffic is permitted in this case, we'll need to add rules for FTP traffic.

We'll need to open ports 20 and 21 for FTP, port 990 for later when we enable TLS, and ports 40000-50000 for the range of passive ports we plan to set in the configuration file:

- **sudo ufw allow 20/tcp**
- **sudo ufw allow 21/tcp**
- **sudo ufw allow 990/tcp**
- **sudo ufw allow 40000:50000/tcp**
- **sudo ufw status**

Now our firewall rules looks like:

Output

```
Status: active
To                         Action      From
-                          ------      ----
OpenSSH                    ALLOW       Anywhere
990/tcp                    ALLOW       Anywhere
20/tcp                     ALLOW       Anywhere
21/tcp                     ALLOW       Anywhere
40000:50000/tcp            ALLOW       Anywhere
OpenSSH (v6)               ALLOW       Anywhere (v6)
20/tcp (v6)                ALLOW       Anywhere (v6)
21/tcp (v6)                ALLOW       Anywhere (v6)
990/tcp (v6)               ALLOW       Anywhere (v6)
40000:50000/tcp (v6)       ALLOW       Anywhere (v6)
```

With vsftpd installed and the necessary ports open, we're ready to proceed to the next step.

Step 3 - Preparing the User Directory

we're going to create a user, but you may already have a user in need of FTP access. We'll take care to preserve an existing user's access to their data in the instructions that follow. Even so, we recommend you start with a new user until you've configured and tested your setup.

First, we'll add a test user:

- **sudo adduser sammy**

Assign a password when prompted and feel free to press "ENTER" through the other prompts.

FTP is generally more secure when users are restricted to a specific directory.vsftpd accomplishes this with chroot jails. When chroot is enabled for local users, they are restricted to their home directory by default. However, because of the way vsftpd secures the directory, it must not be writable by the user. This is fine for a new user who should only connect via FTP, but an existing user may need to write to their home folder if they also shell access.

In this example, rather than removing write privileges from the home directory, we're will create an ftp directory to serve as the chroot and a writable files directory to hold the actual files.

Create the ftp folder, set its ownership, and be sure to remove write permissions with the following commands:

- **sudo mkdir /home/sammy/ftp**
- **sudo chown nobody:nogroup /home/sammy/ftp**

- **sudo chmod a-w /home/sammy/ftp**

Let's verify the permissions:

- **sudo ls -la /home/sammy/ftp**

Output
total 8
4 dr-xr-xr-x 2 nobody nogroup 4096 Aug 24 21:29 .
4 drwxr-xr-x 3 sammy sammy 4096 Aug 24 21:29 ..
Next, we'll create the directory where files can be uploaded and assign ownership to the user:

- **sudo mkdir /home/sammy/ftp/files**
- **sudo chown sammy:sammy /home/sammy/ftp/files**

A permissions check on the files directory should return the following:

- **sudo ls -la /home/sammy/ftp**

Output
total 12
dr-xr-xr-x 3 nobody nogroup 4096 Aug 26 14:01 .
drwxr-xr-x 3 sammy sammy 4096 Aug 26 13:59 ..
drwxr-xr-x 2 sammy sammy 4096 Aug 26 14:01 files

Finally, we'll add a test.txt file to use when we test later on:

- **echo "vsftpd test file" | sudo tee /home/sammy/ftp/files/test.txt**

Now that we've secured the ftp directory and allowed the user access to the files directory, we'll turn our attention to configuration.

Step 4 - Configuring FTP Access

We're planning to allow a single user with a local shell account to connect with FTP. The two key settings for this are already set in vsftpd.conf. Start by opening the config file to verify that the settings in your configuration match those below:

- **sudo nano /etc/vsftpd.conf**
- **/etc/vsftpd.conf**

. . .
Allow anonymous FTP? (Disabled by default).
anonymous_enable=NO
#
Uncomment this to allow local users to log in.
local_enable=YES
. . .

Next we'll need to change some values in the file. In order to allow the user to upload files, we'll uncomment the write_enable setting so that we have:

/etc/vsftpd.conf
. . .
write_enable=YES
. . .

We'll also uncomment the chroot to prevent the FTP-connected user from accessing any files or commands outside the directory tree.

/etc/vsftpd.conf
. . .
chroot_local_user=YES
. . .

We'll add a user_sub_token in order to insert the username in our local_root directory path so our configuration will work for this user and any future users that might be added.

/etc/vsftpd.conf
user_sub_token=$USER
local_root=/home/$USER/ftp

We'll limit the range of ports that can be used for passive FTP to make sure enough connections are available:

/etc/vsftpd.conf
pasv_min_port=40000
pasv_max_port=50000

Note: We pre-opened the ports that we set here for the passive port range. If you change the values, be sure to update your firewall settings.

Since we're only planning to allow FTP access on a case-by-case basis, we'll set up the configuration so that access is given to a user only when they are explicitly added to a list rather than by default:

/etc/vsftpd.conf
userlist_enable=YES
userlist_file=/etc/vsftpd.userlist
userlist_deny=NO

userlist_deny toggles the logic. When it is set to "YES", users on the list are denied FTP access. When it is set to "NO", only users on the list are allowed access. When you're done making the change, save and exit the file.

Finally, we'll create and add our user to the file. We'll use the -a flag to append to file:

- **echo "sammy" | sudo tee -a /etc/vsftpd.userlist**

Double-check that it was added as you expected:

- **cat /etc/vsftpd.userlist**

Output
sammy
Restart the daemon to load the configuration changes:
sudo systemctl restart vsftpd

Now we're ready for testing.

Step 5 - Testing FTP Access

We've configured the server to allow only the user sammy to connect via FTP. Let's make sure that's the case.

Anonymous users should fail to connect: We disabled anonymous access. Here we'll test that by trying to connect anonymously. If we've done it properly, anonymous users should be denied permission:

- **ftp -p 203.0.113.0**

Output
Connected to 203.0.113.0.
220 (vsFTPd 3.0.3)

Name (203.0.113.0:default): anonymous
530 Permission denied.
ftp: Login failed.
ftp>
Close the connection:
bye

Users other than sammy should fail to connect: Next, we'll try connecting as our sudo user. They, too, should be denied access, and it should happen before they're allowed to enter their password.

- **ftp -p 203.0.113.0**

Output
Connected to 203.0.113.0.
220 (vsFTPd 3.0.3)
Name (203.0.113.0:default): sudo_user
530 Permission denied.
ftp: Login failed.
ftp>
Close the connection:
bye

sammy should be able to connect, as well as read and write files: Here, we'll make sure that our designated user *can*connect:

- **ftp -p 203.0.113.0**

Output
Connected to 203.0.113.0.
220 (vsFTPd 3.0.3)
Name (203.0.113.0:default): sammy
331 Please specify the password.
Password: your_user's_password
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>

We'll change into the files directory, then use the get command to transfer the test file we created earlier to our local machine:

- **cd files**
- **get test.txt**

Output
227 Entering Passive Mode (203,0,113,0,169,12).
150 Opening BINARY mode data connection for test.txt (16 bytes).
226 Transfer complete.
16 bytes received in 0.0101 seconds (1588 bytes/s)
ftp>

We'll turn right back around and try to upload the file with a new name to test write permissions:

- **put test.txt upload.txt**

Output

227 Entering Passive Mode (203,0,113,0,164,71).
150 Ok to send data.
226 Transfer complete.
16 bytes sent in 0.000894 seconds (17897 bytes/s)
Close the connection:
bye

Now that we've tested our configuration, we'll take steps to further secure our server.

Step 6 - Securing Transactions

Since FTP does *not* encrypt any data in transit, including user credentials, we'll enable TTL/SSL to provide that encryption. The first step is to create the SSL certificates for use with vsftpd.

We'll use openssl to create a new certificate and use the -days flag to make it valid for one year. In the same command, we'll add a private 2048-bit RSA key. Then by setting both the -keyout and -out flags to the same value, the private key and the certificate will be located in the same file.

We'll do this with the following command:

- **sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/ssl/private/vsftpd.pem -out /etc/ssl/private/vsftpd.pem**

You'll be prompted to provide address information for your certificate. Substitute your own information for the questions below:

Output

Generating a 2048 bit RSA private key
.....+++
.....+++
writing new private key to '/etc/ssl/private/vsftpd.pem'

You are about to be asked to enter information that will be incorporated into your certificate request.

What you are about to enter is what is called a Distinguished Name or a DN. There are quite a few fields but you can leave some blank For some fields there will be a default value,

If you enter '.', the field will be left blank.

Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:NY
Locality Name (eg, city) []:New York City
Organization Name (eg, company) [Internet Widgits Pty Ltd]:DigitalOcean
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []: your_IP_address
Email Address []:

Once you've created the certificates, open the vsftpd configuration file again:

- **sudo nano /etc/vsftpd.conf**

Toward the bottom of the file, you should two lines that begin with rsa_. Comment them out so they look like:

- **/etc/vsftpd.conf**
- **# rsa_cert_file=/etc/ssl/certs/ssl-cert-snakeoil.pem**

- **# rsa_private_key_file=/etc/ssl/private/ssl-cert-snakeoil.key**

Below them, add the following lines which point to the certificate and private key we just created:

- **/etc/vsftpd.conf**
- **rsa_cert_file=/etc/ssl/private/vsftpd.pem**
- **rsa_private_key_file=/etc/ssl/private/vsftpd.pem**

After that, we will force the use of SSL, which will prevent clients that can't deal with TLS from connecting. This is necessary in order to ensure all traffic is encrypted but may force your FTP user to change clients. Change ssl_enable to YES:

- **/etc/vsftpd.conf**
- **ssl_enable=YES**

After that, add the following lines to explicitly deny anonymous connections over SSL and to require SSL for both data transfer and logins:

- **/etc/vsftpd.conf**
- **allow_anon_ssl=NO**
- **force_local_data_ssl=YES**
- **force_local_logins_ssl=YES**

After this we'll configure the server to use TLS, the preferred successor to SSL by adding the following lines:

- **/etc/vsftpd.conf**
- **ssl_tlsv1=YES**
- **ssl_sslv2=NO**
- **ssl_sslv3=NO**

Finally, we will add two more options. First, we will not require SSL reuse because it can break many FTP clients. We will require "high" encryption cipher suites, which currently means key lengths equal to or greater than 128 bits:

- **/etc/vsftpd.conf**
- **require_ssl_reuse=NO**
- **ssl_ciphers=HIGH**

When you're done, save and close the file.

Now, we need to restart the server for the changes to take effect:

- **sudo systemctl restart vsftpd**

At this point, we will no longer be able to connect with an insecure command-line client. If we tried, we'd see something like:

- **ftp -p 203.0.113.0**
- **Connected to 203.0.113.0.**
- **220 (vsFTPd 3.0.3)**
- **Name (203.0.113.0:default): sammy**
- **530 Non-anonymous sessions must use encryption.**
- **ftp: Login failed.**
- **421 Service not available, remote server has closed connection**
- **ftp>**

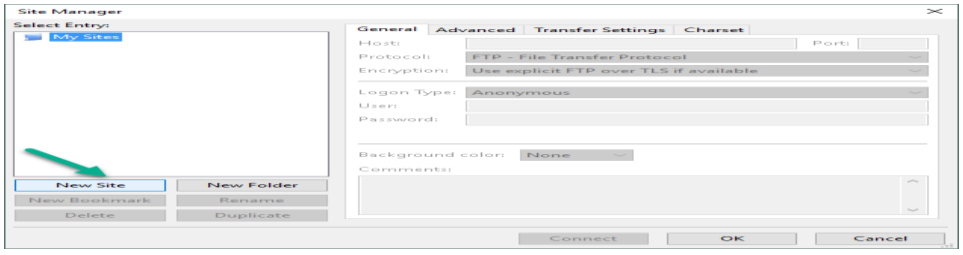
Next, we'll verify that we can connect using a client that supports TLS.

Step 7 - Testing TLS with FileZilla

Most modern FTP clients can be configured to use TLS encryption. We will demonstrate how to connect using FileZilla because of its cross platform support. Consult the documentation for other clients.

When you first open FileZilla, find the Site Manager icon just below the word File, the left-most icon on the top row. Click it:

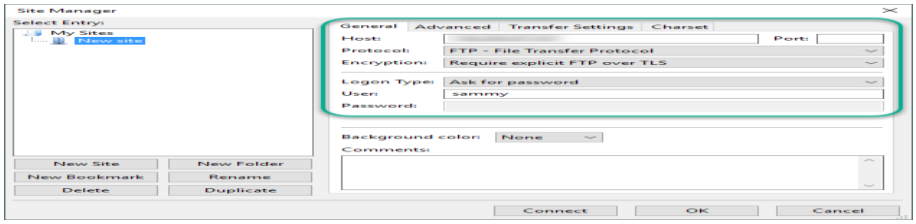
A new window will open. Click the "New Site" button in the bottom right corner:



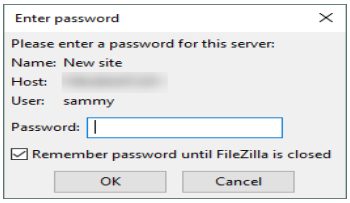
Under "My Sites" a new icon with the words "New site" will appear. You can name it now or return later and use the Rename button.

You must fill out the "Host" field with the name or IP address. Under the "Encryption" drop down menu, select "Require explicit FTP over TLS".

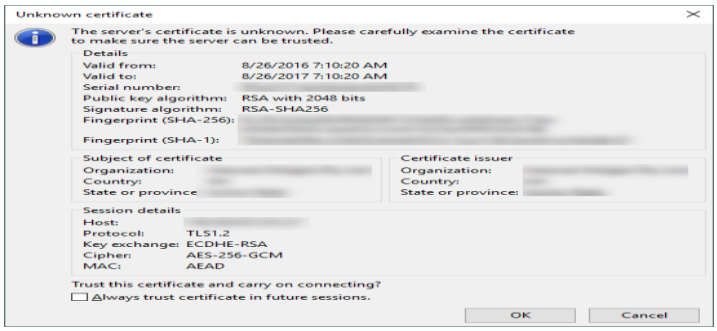
For "Logon Type", select "Ask for password". Fill in the FTP user you created in the "User" field:



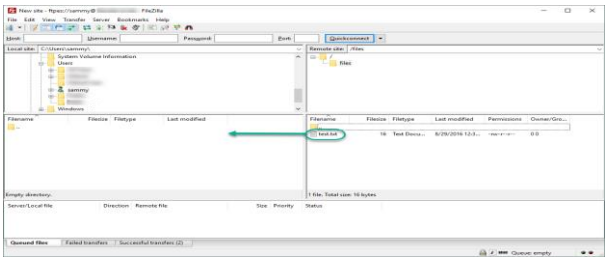
Click "Connect" at the bottom of the interface. You will be asked for the user's password:



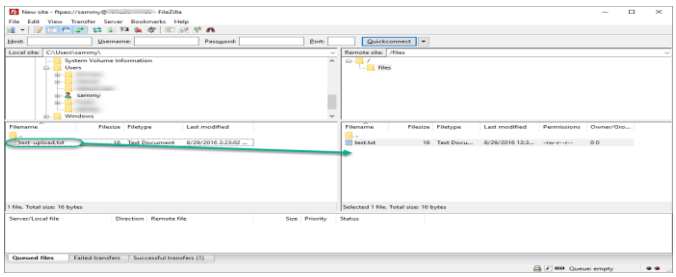
Click "OK" to connect. You should now be connected with your server with TLS/SSL encryption.



When you've accepted the certificate, double-click the files folder and drag upload.txt to the left to confirm that you're able to download files.



When you've done that, right-click on the local copy, rename it to upload-tls.txt` and drag it back to the server to confirm that you can upload files.



You've now confirmed that you can securely and successfully transfer files with SSL/TLS enabled.

Step 8 - Disabling Shell Access (Optional)

If you're unable to use TLS because of client requirements, you can gain some security by disabling the FTP user's ability to log in any other way. One relatively straightforward way to prevent it is by creating a custom shell. This will not provide any encryption, but it will limit the access of a compromised account to files accessible by FTP. First, open a file called ftponly in the bin directory:

- **sudo nano /bin/ftponly**

We'll add a message telling the user why they are unable to log in. Paste in the following:

- **#!/bin/sh**
- **echo "This account is limited to FTP access only."**
- **Change the permissions to make the file executable:**
- **sudo chmod a+x /bin/ftponly**

Open the list of valid shells:

- **sudo nano /etc/shells**

At the bottom, add:

- **/etc/shells**
- **. . .**
- **/bin/ftponly**

Update the user's shell with the following command:

- **sudo usermod sammy -s /bin/ftponly**

Now try logging in as sammy:

- **ssh sammy@203.0.113.0**

You should see something like:

Output
This account is limited to FTP access only.
Connection to 203.0.113.0 closed.

Configure Ubuntu Firewall (UFW)

Security is crucial when you run your own server. You want to make sure that only authorized users can access your server, configuration, and services.

In Ubuntu, there is a firewall that comes preloaded. It's called UFW (Ubuntu-Firewall). Although UFW is a pretty basic firewall, it is user friendly, excels at filtering traffic, and has good documentation. Some basic Linux knowledge should be enough to configure this firewall on your own.

1. Install UFW

UFW is installed by default in Ubuntu, but you can verify this: which ufw You should receive the following output:

- **/usr/sbin/ufw**

If you don't receive output, that means that UFW is not installed. You can install it yourself if this is the case:

- **sudo apt-get install ufw**

2. Allow connections

If you are running a web server, you want the world to be able to access your website(s). Therefore, you need to make sure that the default TCP ports for web are open.

- **sudo ufw allow 80/tcp**
- **sudo ufw allow 443/tcp**

In general, you can allow any port you need by using the following format:

- **sudo ufw allow <port>/<optional: protocol>**

3. Deny connections

If you need to deny access to a certain port, use the deny command:

- **sudo ufw deny <port>/<optional: protocol>**

For example, you can deny access to your default MySQL port:

- **sudo ufw deny 3306**

UFW also supports a simplified syntax for the most common service ports:

- **root@ubuntu:~$ sudo ufw deny mysql**
- **Rule updated**
- **Rule updated (v6)**

It is highly recommended that you restrict access to your SSH port, (by default, this is port 22), from anywhere except your trusted IP addresses.

4. Allow access from a trusted IP address

Typically, you would need to allow access only to publicly open ports, such as port 80. Access to all other ports should be restricted or limited. You can whitelist your home or office IP address, (preferably a static IP), to be able to access your server through SSH or FTP:

- **sudo ufw allow from 192.168.0.1 to any port 22**

You can also allow access to the MySQL port:

- **sudo ufw allow from 192.168.0.1 to any port 3306**

5. Enable UFW

Before enabling (or restarting) UFW, you need to make sure that the SSH port is allowed to receive connections from your IP address. To start/enable your UFW firewall, use the following command:

- **sudo ufw enable**

You will see the following output:

- **root@ubuntu:~$ sudo ufw enable**
- **Command may disrupt existing ssh connections. Proceed with operation (y|n)?**

Press Y, then press ENTER to enable the firewall:

- **Firewall is active and enabled on system startup**

6. Check UFW status

Print the UFW rule list:

- **sudo ufw status**

You will see output similar to the following:

- **Status: active**

```
To                         Action      From
--                         ------      ----
80/tcp                     DENY        Anywhere
443/tcp                    DENY        Anywhere
3306                       DENY        Anywhere
22                         ALLOW       192.168.0.1
3306                       ALLOW       192.168.0.1
80/tcp (v6)                DENY        Anywhere (v6)
443/tcp (v6)               DENY        Anywhere (v6)
3306 (v6)                  DENY        Anywhere (v6)
```

Use the verbose parameter to see a more detailed status report:

- **sudo ufw status verbose**

That output will resemble the following:

- **Status: active**
- **Logging: on (low)**
- **Default: deny (incoming), allow (outgoing), disabled (routed)**
- **New profiles: skip**

```
To                         Action      From
--                         ------      ----
80/tcp                     DENY IN     Anywhere
443/tcp                    DENY IN     Anywhere
3306                       DENY IN     Anywhere
22                         ALLOW IN    192.168.0.1
3306                       ALLOW IN    192.168.0.1
80/tcp (v6)                DENY IN     Anywhere (v6)
443/tcp (v6)               DENY IN     Anywhere (v6)
3306 (v6)                  DENY IN     Anywhere (v6)
```

7. Disable/reload/restart UFW

If you need to reload the firewall rules run the following:

- **sudo ufw reload**

To disable, or stop UFW:

- **sudo ufw disable**

In order to restart UFW, you will need to disable it first, and then enable it again:

- **sudo ufw disable**
- **sudo ufw enable**

Note: *Before enabling UFW, make sure that the SSH port is allowed for your IP address.*

8. Removing rules

To manage your UFW rules, you need to list them. You can do that by checking UFW status with the parameter numbered:

- **sudo ufw status numbered**

You will see output similar to the following:

- **Status: active**

```
     To                         Action      From
     --                         ------      ----
[ 1] 80/tcp                     DENY IN     Anywhere
[ 2] 443/tcp                    DENY IN     Anywhere
[ 3] 3306                       DENY IN     Anywhere
[ 4] 22                         ALLOW IN    192.168.0.1
[ 5] 3306                       ALLOW IN    192.168.0.1
[ 6] 80/tcp (v6)                DENY IN     Anywhere (v6)
[ 7] 443/tcp (v6)               DENY IN     Anywhere (v6)
[ 8] 3306 (v6)                  DENY IN     Anywhere (v6)
```

Now, to remove any of these rules, you will need to use these numbers in the square brackets:

- **sudo ufw delete [number]**

To remove the HTTP rule, (80), use the following command:

- **sudo ufw delete 1**

9. Enabling IPv6 support

If you use IPv6 on your VPS, you need to ensure that IPv6 support is enabled in UFW. To do so, open the config file in a text editor:

- **sudo vi /etc/default/ufw**

Once opened, make sure that IPV6 is set to "yes":

- **IPV6=yes**

After making this change, save the file. Then, restart UFW by disabling and re-enabling it:

- **sudo ufw disable**
- **sudo ufw enable**

10. Back to default settings

If you need to go back to default settings, simply type in the following command. This will revert any of your changes:

- **sudo ufw reset**

Congratulations, you've just set up some basic firewall rules.

What is Wine?

Wine (originally an acronym for "Wine Is Not an Emulator") is a compatibility layer capable of running Windows applications on several POSIX-compliant operating systems, such as Linux, macOS, & BSD. Instead of simulating internal Windows logic like a virtual machine or emulator, Wine translates Windows API calls into POSIX calls on-the-fly, eliminating the performance and memory penalties of other methods and allowing you to cleanly integrate Windows applications into your desktop.

Installing Wine

Ubuntu versions of Wine (Recommended)

1. open the software center
2. Type wine and install 'wine'

Newer versions of Wine (Not Recommended)

If you are using 9.10 (Karmic) or 10.04 (Lucid), the Ubuntu Wine PPA provides the newest development versions of Wine. Add the following to Software Sources:

- `ppa:ubuntu-wine/ppa`

Note, however, that these are development packages (ie beta software), and may suffer from regressions and other problems not present in the stable version of Wine included with Ubuntu. You should avoid using them unless the current stable version of Wine does not support or is incompatible with the application you wish to use.

If you are using an older version of Wine and want support from WineHQ, you will need to upgrade to the latest development version first. If you do this, however, please file associated Wine bugs at winehq's Bugzilla Page rather than in Launch pad.

1. Initial Setup

Before using Wine, it is necessary to create the fake C: drive where your Windows applications will be installed. To do this, enter the following command into the terminal: You may find the terminal by going to Applications -> Accessories -> Terminal

- **winecfg**

You also have the option of configuring Wine via the Configure Wine option in the Applications-> Wine menu.

This will create a hidden folder (.wine) in your home directory containing the fake C: drive as well as registry files similar to those used in Windows. Once this directory is created, the **Wine Configuration Window** will appear. This window will allow you to customize a variety of settings for Wine, including which Windows Version that is emulated, drive mappings, DLL overrides, as well as application specific settings. Click the **Ok** button to close the window.

2. Installing Windows Applications With Wine

To install Windows applications using Wine, follow these instructions:

- Download the Windows application from any source (e.g. download.com). Download the .EXE (executable).
- Place it in a convenient directory (e.g. the desktop, or home folder).
- Open the terminal, and **cd** into the directory where the .EXE is located.
- Type wine *the-name-of-the-application.extension*
  (e.g. wine realplayer_installer.exe).

This will start the .EXE using Wine. If it is an installer, it should then run as it would in Windows. If the application asks for a directory to install the application to, select put it under C:\Program Files.

3. To start/run Windows programs using Wine

After installing an application using the directions above, those applications can be started and used by entering *wine programname.exe* (e.g. *wine realplayer.exe*). When done, close the application as one would normally. You must run the installed executable, which will by default be in the virtual Windows drive created by Wine, at *~/.wine/drive_c*. Generally programs will install themselves somewhere under *Program Files* inside the virtual Windows drive, following Windows conventions.

You can also use the Wine file browser, by running *winefile* in a terminal. Clicking the *C:* button in the toolbar will open a window where you can browse the virtual Windows drive created in .wine. Double clicking an executable in the Wine file browser will run it in Wine.

Instead of having to always enter the terminal or use the Wine file browser, you may also create a desktop icon, and start a Wine application using that icon.

To do this, right click on the desktop and select "Create a launcher." If you wish, select an icon from the list of available icons (or browse to an icon you would like to use), fill out other information that is requested (Name, generic name, etc.). For the command, type in **wine the-location-of-the-program.exe** (e.g. wine/home/john/ .wine/realplayer.exe). The most important part of creating a launcher is the command, the generic name is not as important. Just make sure you de-select "Run in terminal." This completes the process.

In some cases the application requires to be running from a certain location. In this case create launcher with command

- **sh -c "cd /home/USER/.wine/drive_c/Program Files/Appdir/; wine /home/USER/.wine/drive_c/Program Files/Appdir/game.exe"**

Of course you will need to replace USER and Appdir with the proper data. If you desire to have an icon on the panel, create a launcher on the panel of choice. Do this by right-clicking the panel, selecting "Add to Panel," and selecting "Custom Application Launcher." This will ask you for the same information as before.

Alternatively, to make life easier, you can set it so wine will automatically open .exe files files for you - instead of using the Wine File to locate the file each time.

To do so, right click on the .exe file, select Properties, and then select the Open With tab. Click the 'Add' button, and then click on 'Use a custom command'. In the line that appears, type in wine, then click Add, and Close. Now all .exe files will be automatically opened by Wine, so you can use Nautilus to browse and open them instead of the Wine File.

4. Uninstalling Wine Applications

Open up a terminal window and type the command below.

- **wine uninstaller**

What this will do is open up a program similar to the Windows **add/remove programs** control panel, allowing you to uninstall applications from a Wine installation. Running uninstall programs directly via Wine should also work normally. Alternatively, you could also simply delete the folder of the application. However, as when done in Windows, this method will be **unclean** and will not remove the program's configuration from the Wine registry like using an uninstaller will.