

# Linux System Administration

# **Who created Linux?**

- Linus Torvalds, a computer science student, conceived Linux while he was young. He worked on UNIX OS and believed it required upgrades. When his recommendations for UNIX were rejected, he thought of creating a user-modifiable OS.

# **Linux Distribution**

- Linux distribution is an operating system that is made up of a group of programs that are all based on the Linux kernel. You can also say that distribution is made up of the Linux kernel and software and libraries that help it work. And you can get a Linux-based operating system by downloading one of the Linux distributions. These distributions are available for different types of devices, such as embedded devices, personal computers, etc. There are more than 600 Linux distributions, and some of the most popular ones are:

# **Linux Distribution**

- MX Linux
- Manjaro
- Linux Mint
- Elementary
- Ubuntu
- Debian
- Solus
- Fedora
- openSUSE
- Deepin.

# Linux Distributions

There are on an average six hundred Linux distributors providing different features. Here are some of the popular Linux distros-

## Ubuntu-

- It is the most well-known Linux distribution.
- It came into existence in 2004 by Canonical and quickly became popular.
- It can be used as easy graphical Linux desktop without the use of command line.
- Ubuntu is a next version of Debian and easy to use for newbies.
- It comes with a lots of pre-installed apps and easy to use repositories libraries.
- It releases every six months and currently working to expand to run on tablets and smartphones.

# Linux Distributions

## Linux Mint-

- Mint is based on Ubuntu and uses its repository software so some packages are common in both.
- Earlier it was an alternative of Ubuntu because media codecs and proprietary software are included in mint but was absent in Ubuntu.
- But now it has its own popularity and it uses cinnamon and mate desktop instead of Ubuntu's unity desktop environment.

# Linux Distributions

## Debian-

- Debian has its existence since 1993 and releases its versions much slowly then Ubuntu and mint.
- This makes it one of the most stable Linux distributor.
- Ubuntu is based on Debian and was founded to improve the core bits of Debian more quickly and make it more user friendly.
- Every release name of Debian is based on the name of the movie Toy Story.

# Linux Distributions

## RedHat Enterprise / CentOS-

- Red hat is a commercial Linux distributor.
- There products are red hat enterprise Linux (RHEL) and Fedora which are freely available.
- RHEL is well tested before release and supported till seven years after the release, whereas, fedora provides faster update and without any support.
- CentOS is a community project that uses RedHat enterprise Linux code but removes all its trademark and make it freely available. In other words, it is a free version of RHEL and provide a stable platform for a long time.

# Linux Distributions

## Fedora-

- It is a project that mainly focuses on free software and provides latest version of software.
- It doesn't make its own desktop environment but used 'upstream' software.
- By default, it has GNOME3 desktop environment.
- It is less stable but provides the latest stuff.

## **Does Linux differ from Ubuntu in any way?**

- The right answer is yes. Linux is a family of open-source operating systems based on the Linux kernel. Ubuntu, on the other hand, is a free, open-source operating system and Linux distribution based on Debian. Or, to say it another way, Linux is the main system, and Ubuntu is how Linux is used. Linux is made by Linus Torvalds, and it came out in 1991. Ubuntu was made by Canonical Ltd. and came out in 2004.

# **How things work**

Here are a few of the most important things about the Linux Operating System.

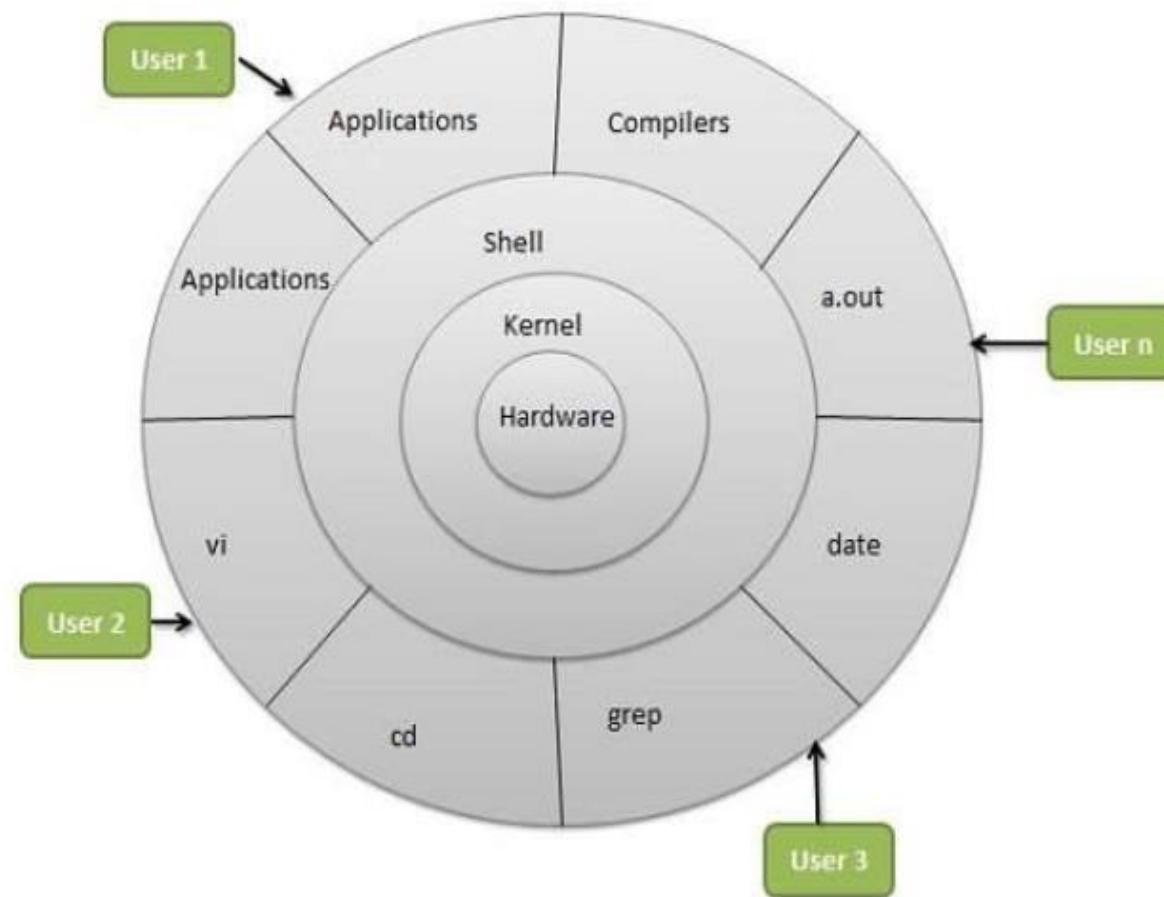
- **Portable:** When software is portable, it can run the same way on different kinds of hardware. The Linux kernel and application programs can be installed on any kind of hardware.
- **Open Source:** Linux's source code is free to use, and it is built by a group of people working together. Several teams work together to make the Linux operating system better, and it is always getting better.

- **Multi-User:** Linux is a multiuser system, which means that more than one person can use system resources like memory, RAM, and application programs at the same time.
- Linux is a multiprogramming system, which means that more than one app can run at the same time.
- **Hierarchical File System:** Linux has a standard way to organize files, both system files, and user files.

- **Shell:** Linux has a special program called an interpreter that can be used to run operating system commands. It can be used to call application programs, do different kinds of operations, etc.
- **Security:** Linux keeps its users safe with authentication features like password protection, limited access to certain files, and data encryption.

# Architecture

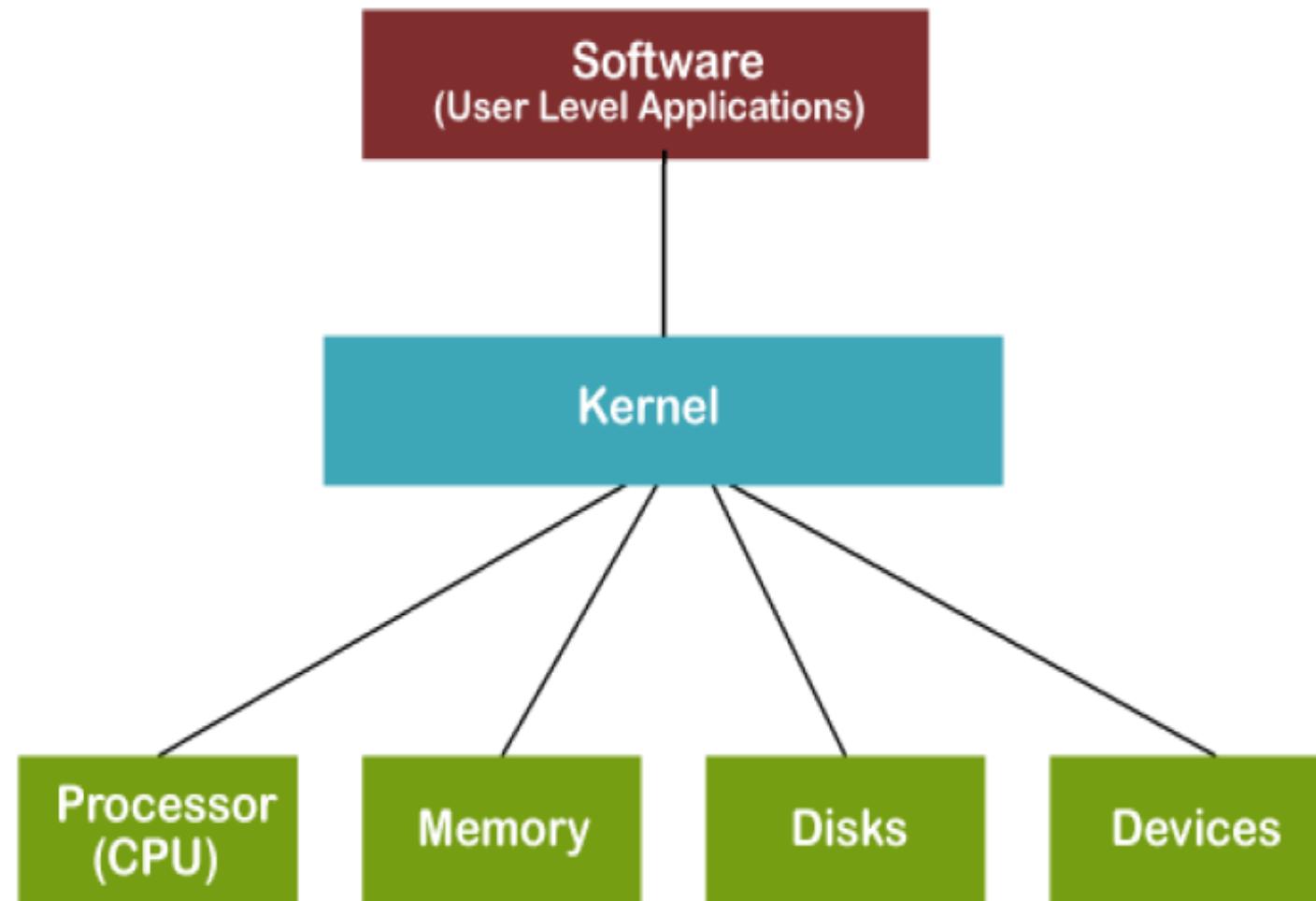
- The following illustration shows the architecture of a Linux system –



## A Linux system's design is made up of the following layers:

- **Hardware layer:** Hardware is made up of all the extra devices, like RAM, HDD, CPU, and so on.
- **The kernel** is the most important part of an operating system. It communicates directly with the hardware and gives low-level services to higher-level parts.
- **Shell** is an interface to the kernel that hides the complexity of how the kernel works from the user. The user gives commands to the shell, which then runs the functions of the kernel.

# What is Kernel in Operating System?



# Functions of a Kernel

- Device Management
- Memory Management
- Resource Management
- Accessing Computer Resources

# Shell

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

# Types of Shell

- There are many different types of shell available in Linux. Some of them are-
  - ✓ Bash- Bourne Again Shell (default)
  - ✓ Dash- Debian Alquist Shell
  - ✓ Csh- C Shell
  - ✓ Ksh- Korn Shell

# Types of Shell

- To see the available shells use the following command-

```
amitabh@LAPTOP-8F40GMIU:~$ cat /etc/shells
# /etc/shells: valid login shells
/bin/sh
/usr/bin/sh
/bin/bash
/usr/bin/bash
/bin/rbash
/usr/bin/rbash
/usr/bin/dash
/usr/bin/tmux
```

- To see the active shell use the following command-  
**\$ echo \$SHELL**

# Ubuntu OS

- **Ubuntu** is a Linux distribution derived from **Debian** and composed mostly of free and open-source software.
- Ubuntu is officially released in multiple editions- Desktop, Server, and Core for Internet of things devices and robots.
- The operating system is developed by the British company **Canonical** and a community of other developers.

# Ubuntu Desktop

- Ubuntu Desktop has always been free to download, use and share. Each release of Ubuntu Desktop delivers the latest applications, libraries and toolchains.
- It is a primary platform for all major IDEs, game development tools and AI/ML software.
- It offers essential applications for web browsing, messaging, gaming and content creation, including Firefox, Chrome, OBS Studio, supporting all daily computing needs.
- It prioritizes user privacy and system integrity, making it the secure choice for those who value data protection.

# Ubuntu Server

- Ubuntu Server is a Free and Open-source operating system.
- Developed and maintained by **Canonical Ltd.** with regular LTS (Long Term Support) releases.
- Designed for server environments, focusing on stability, security, and performance.
- No GUI by default, which reduces system overhead and makes it ideal for performance.
- Pre-packaged software in the Ubuntu repository (e.g., Apache, Nginx, MySQL, PostgreSQL, Docker).
- Excellent support for cloud platforms like **AWS**, **Azure**, and **Google Cloud**.
- **Automatic security updates** for critical packages and vulnerabilities.

# Kubuntu

- It's an open-source, user-friendly operating system that provides a powerful, customizable, and visually appealing environment.
- Kubuntu is an official **Ubuntu** flavour that uses the **KDE Plasma** desktop environment instead of the default GNOME desktop found in Ubuntu.
- Plasma offers a sleek, visually stunning desktop with modern animations and effects.
- Users can modify themes, widgets, taskbars, and window decorations easily with the **KDE System Settings**.
- Plasma includes many productivity tools like **Krita**, **Kdenlive**, and **Okular** for drawing, video editing, and document viewing.
- **KRunner** is a powerful launcher that helps you quickly access apps, settings, and files.

# Key Apps of Kubuntu

- **Dolphin File Manager-**

A powerful file manager with features like tabbed browsing, advanced file search, and integration with cloud services.

- **Konsole Terminal-**

A robust terminal emulator, great for developers and advanced users who need command-line access.

- **Kate Text Editor-**

A highly capable text editor for coding, with syntax highlighting, and features for advanced users.

- **LibreOffice Suite-**

Kubuntu comes with LibreOffice pre-installed for word processing, spreadsheets, and presentations.

# Xubuntu

- Xubuntu is an official **Ubuntu** flavor that uses the **XFCE** desktop environment, focusing on providing a lightweight, efficient, and user-friendly experience.
- **XFCE** is a lightweight desktop environment that uses fewer system resources, making it ideal for low-resource or older hardware.
- Xubuntu provides a fast, responsive experience even on systems with limited RAM or CPU power.
- XFCE offers a simple, straightforward interface with a traditional desktop layout (menu, taskbar, system tray).
- XFCE is customizable, allowing users to tweak the appearance and functionality of the desktop environment to suit their preferences.
- Includes the ability to change themes, panel layouts, and add widgets, but it remains simple and uncluttered.

# Key Apps of xubuntu

- **Thunar File Manager-**

It is a lightweight, easy to use, and highly customizable, allowing users to manage files quickly and efficiently. Includes features like customizable actions, a simple search function, and file previews.

- **XFCE Terminal-**

A fast and efficient terminal emulator, perfect for users who prefer command-line operations.

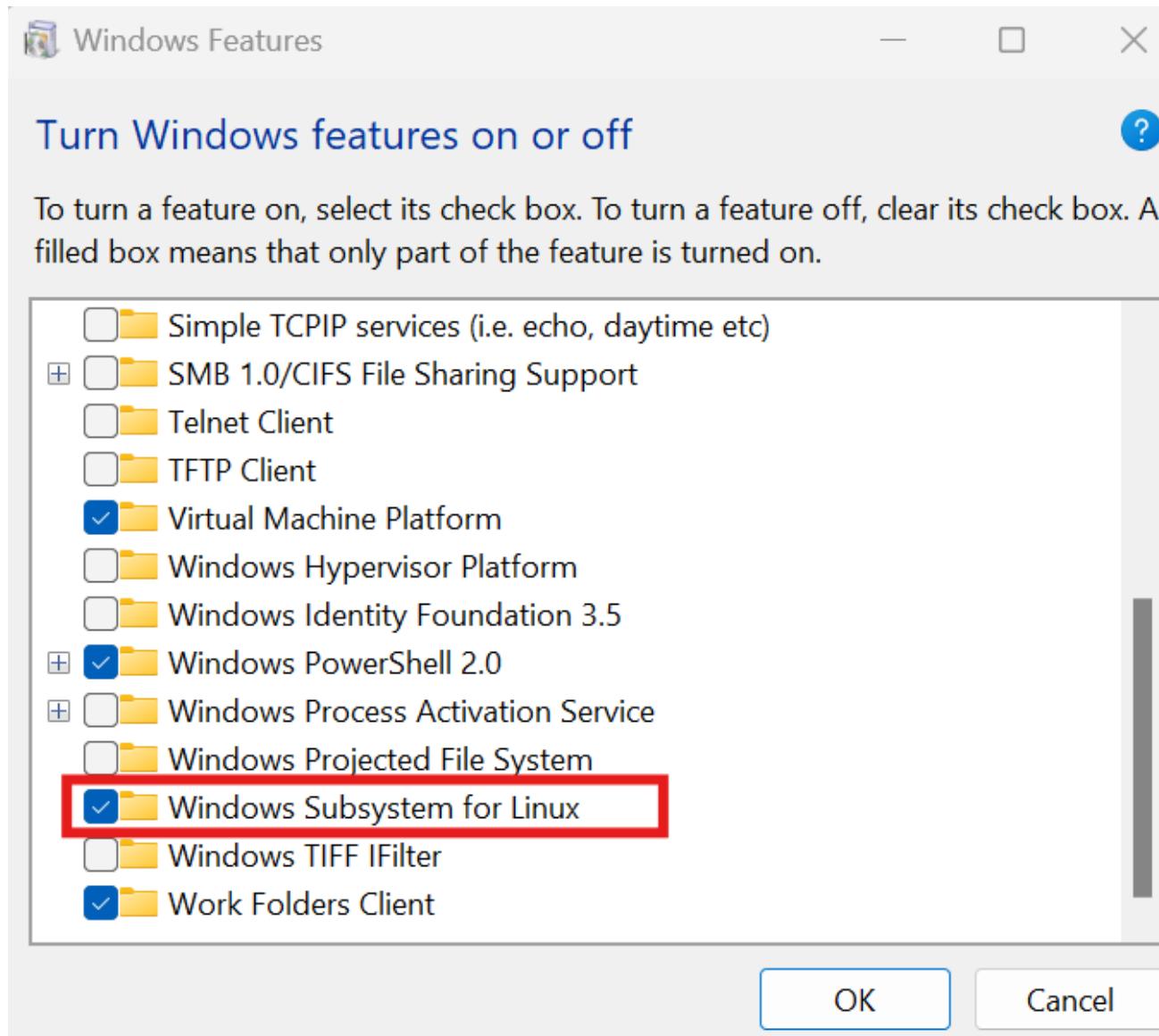
- **Mousepad Text Editor-**

A simple and lightweight text editor designed for fast note-taking and basic text manipulation.

- **Ristretto Image Viewer-**

A lightweight image viewer for quick and easy viewing of common image formats.

# Activating / Installing Linux (ubuntu)



# Activating / Installing Linux

```
D:\>wsl --install
Windows Subsystem for Linux is already installed.
The following is a list of valid distributions that can be installed.
Install using 'wsl --install -d <Distro>'.

NAME                      FRIENDLY NAME
Ubuntu                     Ubuntu
Debian                     Debian GNU/Linux
kali-linux                 Kali Linux Rolling
Ubuntu-18.04                Ubuntu 18.04 LTS
Ubuntu-20.04                Ubuntu 20.04 LTS
Ubuntu-22.04                Ubuntu 22.04 LTS
Ubuntu-24.04                Ubuntu 24.04 LTS
OracleLinux_7_9              Oracle Linux 7.9
OracleLinux_8_7              Oracle Linux 8.7
OracleLinux_9_1              Oracle Linux 9.1
openSUSE-Leap-15.6            openSUSE Leap 15.6
SUSE-Linux-Enterprise-15-SP5 SUSE Linux Enterprise 15 SP5
SUSE-Linux-Enterprise-15-SP6 SUSE Linux Enterprise 15 SP6
openSUSE-Tumbleweed           openSUSE Tumbleweed

D:\>Ubuntu
'Ubuntu' is not recognized as an internal or external command,
operable program or batch file.

D:\>wsl --install -d Ubuntu
Installing: Ubuntu
[=====63.0%=====] ]
```

# Partitioning with Gparted

- Disk Partitioning in Linux is done using command line programs like fdisk.
- GParted(GNOME Partition Editor) is a Graphical user program based on GTK which allows Disk Partitioning with just a click of the buttons.
- GParted is used for creating, deleting, resizing, moving, checking, and copying disk partitions and manipulating file systems such as exfat, fat32/64, ext2/3/4, Linux-swap and more.

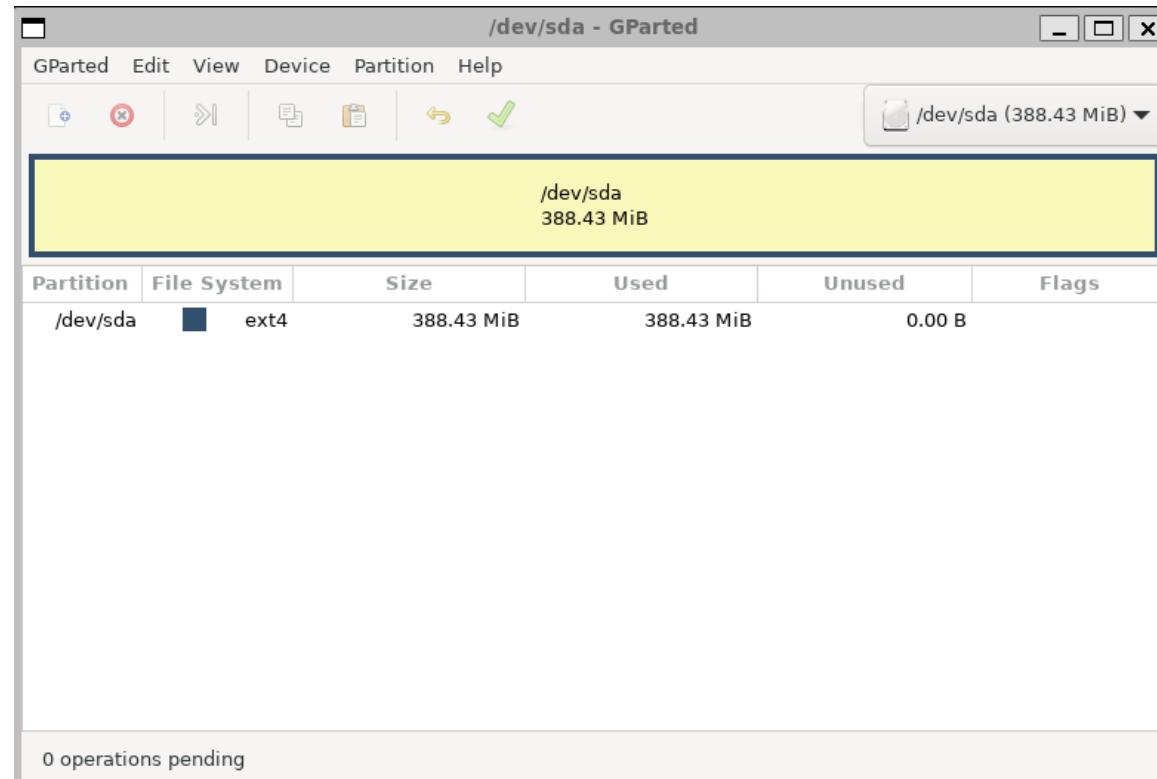
# Installing Gparted

- We can install GParted in Ubuntu with a single command using APT-  
**sudo apt install gparted**
- You can verify whether GParted is successfully installed or not using the below command which gives an output similar to below image-

```
amitabh@LAPTOP-8F40GMIU:~$ sudo apt policy gparted
[sudo] password for amitabh:
gparted:
  Installed: 1.5.0-1build4
  Candidate: 1.5.0-1build4
  Version table:
 *** 1.5.0-1build4 500
      500 http://archive.ubuntu.com/ubuntu noble/main amd64 Packages
        100 /var/lib/dpkg/status
amitabh@LAPTOP-8F40GMIU:~$
```

# Gparted Interface

- To open GParted , give the following command-
- **sudo gparted**
- It will open the Gparted application same as below image-



# Gparted Menu

- **new** – create a new partition
- **delete** – delete an existing partition
- **resize/move** – to resize a partition, either shrink(or)increase the size
- **copy/paste** – used to copy/paste the text(or) information
- **undo** – undo the previous action
- **apply** – to perform the operation chosen from above, you have to click the apply icon to commit any selected operation in gparted

# Bootloader in Linux

- The bootloader in Linux serves as a critical component in the startup process of our Operating systems. It plays an important role in initiating the system and facilitating the loading of the Linux kernel.
- Once we turn on our Linux machines the first thing that runs is the **BIOS** (Basic Input Output System). The BIOS is responsible for basic hardware initialization. It checks for connected devices, performs a power-on self-test (POST), and searches for the **MBR** (Master Boot Record) in the BIOS boot menu.
- The **MBR** is a small piece of information (512 bytes) that contains the bootloader program and disk partitioning information. After the MBR is run the bootloader begins to start. The bootloader is a crucial component of the Linux booting process and is responsible for loading the Linux kernel into memory. Then followed by the kernel initialization phase of the Linux booting process.

# Functions of Bootloader

□ Some common functions of a Bootloader are-

- Performs boot device selection based on configuration.
- Takes control from the BIOS firmware after the initial bootloader is loaded.
- Provides a menu or configuration options for selecting an operating system (in multi-boot setups, if applicable).
- Loads the operating system's kernel and necessary files into memory from the boot device.
- Sets up the necessary environment for the operating system to start execution.
- Transfers control to the operating system, handing off the boot process to the kernel.

# Best Linux Bootloader

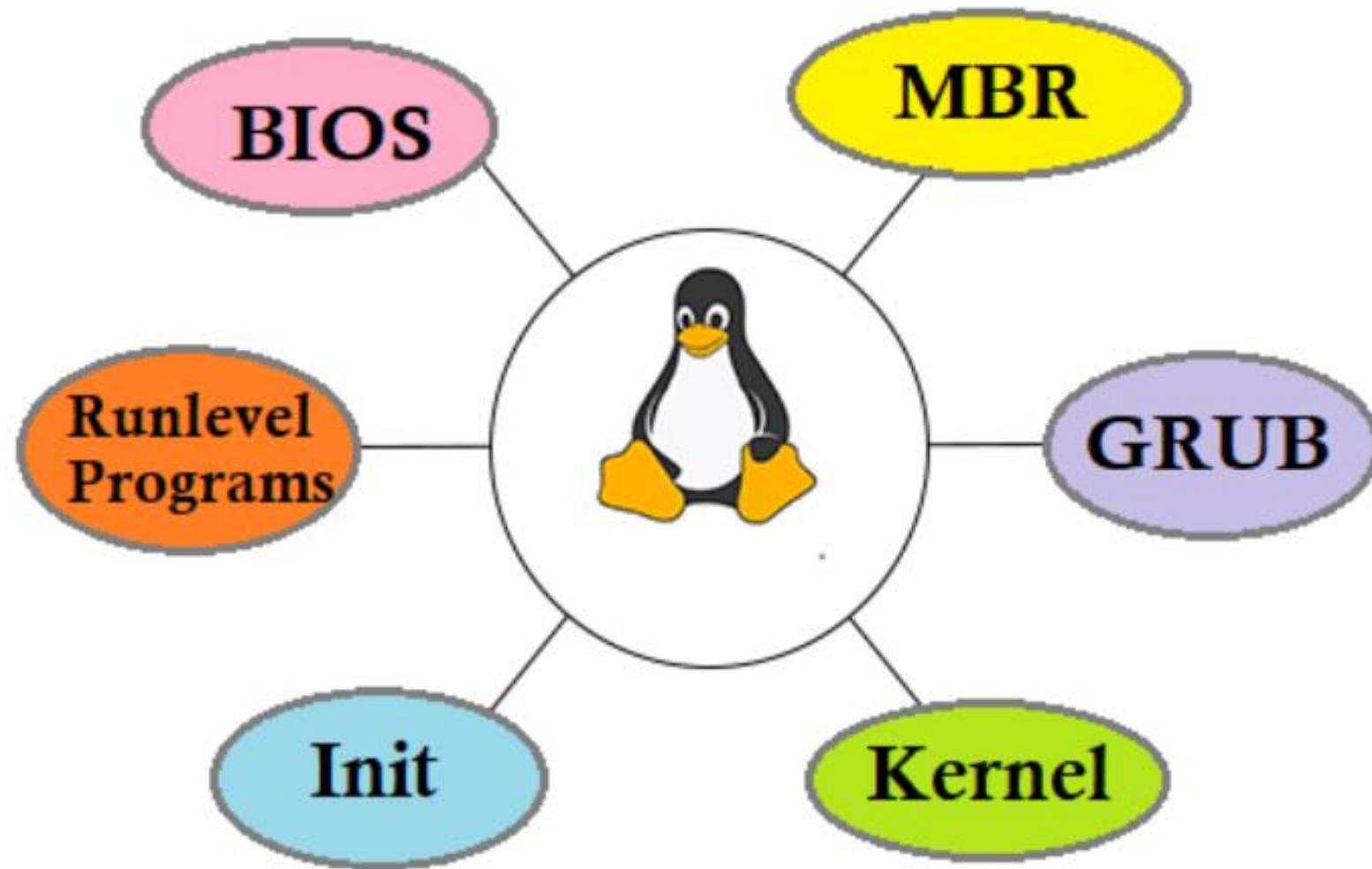
## GRUB-

GRUB or Grand Unified Bootloader is one of the most popular and widely used bootloaders in Linux. It supports a wide range of operating systems, provides a user-friendly interface, and offers advanced features. GRUB supports features like dual-booting, customizable boot menus, and compatibility with various filesystems. It is highly flexible and is the default bootloader in many Linux distributions.

## LILO-

LILO or Linux Ioader is another well-known bootloader that has been used in Linux for a long time. It is known for its simplicity and reliability. LILO can handle various disk layouts and supports multiple boot options. While it may not have as many advanced features as GRUB, LILO is still a solid choice, especially for users seeking a straightforward and stable bootloader.

# Linux Boot Process



# Linux Boot Process

## STEP-1- BIOS

BIOS is an acronym for Basic Input/Output System. In other words, the BIOS can load and run the MBR (Master Boot Record) boot loader. When we first turn on our system, the BIOS first implements a few integrity checks of the SSD or HDD. After that, the BIOS finds, loads, and runs the boot loader function, which can be detected in the MBR. Then, the boot loader function is loaded into memory.

## Step-2- MBR

MBR is an acronym for Master Boot Record and is liable to load and run the GRUB boot loader. MBR is placed in the first bootable disk sector, which is generally `/dev/sda`, relying on our hardware.

# Linux Boot Process

## STEP-3- GRUB

GRUB is known as **G**rand **U**nified **B**ootloader. It is the classic bootloader for almost all the latest Linux systems. The splash screen of GRUB is often the initial thing we see when we boot our system.

It contains a general menu where we can choose some portions. We can use our keyboard to choose the one we wish our system to initiate with if we have multiple installed kernel images. The latest kernel image is chosen by default.

The splash screen will delay for some seconds for us to choose options. It will load the kernel image (default) if we don't. In several systems, we can see the GRUB configuration file at **/etc/grub.conf** or **/boot/grub/grub.conf**.

# Linux Boot Process

## STEP-4- KERNEL

Often, the kernel is called the code of an operating system. In this boot process stage, the kernel mounts the base file system that was chosen that is set up in the file, i.e., ***grub.conf***.

Then, it runs the ***/sbin/init*** function, which is always the initial function to be run. We can confirm it with its PID (process id), which should be always 1.

Then, the kernel creates a temporary base file system with the help of ***initrd*** (Initial RAM Disk) until the actual file system is mounted.

# Linux Boot Process

## STEP-5- INIT

**init** is the parent of all processes and it is executed by the kernel and is responsible for starting all other processes.

At this stage, our system runs runlevel programs. It would find an **init** file, generally detected at **/etc/inittab**, to determine the run level of Linux. Modern Linux systems use **systemd** to select a run level rather. There are six runlevels in the Linux-

|             |                        |
|-------------|------------------------|
| Runlevel 0- | halt                   |
| Runlevel 1- | single-user mode       |
| Runlevel 2- | multiuser, without NFS |
| Runlevel 3- | Full multiuser mode    |
| Runlevel 4- | unused                 |
| Runlevel 5- | X11                    |
| Runlevel 6- | reboot                 |

**To see the current Runlevel-**

```
$ runlevel  
N 5
```

```
$ who -r  
run-level 5 2024-12-10 13-30
```

# Linux File System Structure

- Linux file system has a hierachal file structure as it contains a root directory and its subdirectories.
- All other directories can be accessed from the root directory.
- The directory structure of Linux is well-documented and defined in the **Linux FHS (Filesystem Hierarchy Standard)**.
- To access the sequentially deeper directory names of the directory linked by '/' forward slash like **/var/spool/mail** and **/var/log**. These are known as **paths**.

# Top-level directories

| Directories | Description  |
|-------------|--|
| /bin        | binary or executable programs.                       |
| /etc        | system configuration files.                          |
| /home       | home directory. It is the default current directory. |
| /opt        | optional or third-party software.                    |
| /tmp        | temporary space, typically cleared on reboot.        |
| /usr        | User related programs.                               |
| /var        | log files.   |

# touch command

**touch** command is used to create empty files and also to modify the timestamp of a file.

## Options-

**touch file1.txt**

It will create an empty file called ‘file1.txt’.

**touch file1.txt file2.txt file3.txt**

It will create three empty files in current directory called ‘file1.txt’, ‘file2.txt’ and ‘file3.txt’.

**touch -- -file.txt**

It will create an empty file as ‘**-file.txt**’ whose first character is hyphen (-). in current directory.

**touch -a file1.txt**

It will change the access time of file called ‘file1.txt’ and set it as current time.

*(use ‘stat <filename>’ command to check the access time of file)*

**touch -m file1.txt**

It will change only the modification date & time of file ‘file1.txt’ to current date and time.

**touch -t 201210280605 file1.txt (format used- YYYYMMDDHHMM)**

It will change the access & modification date & time of file ‘file1.txt’ to 2012-10-28 06-05.

**touch -r oldfile newfile**

It will change the modification date & time of newfile as per oldfile.

# mkdir command

**mkdir** command is used to create directories/folders.

It can create multiple directories at once as well.

## Options-

**mkdir mydir**

It will create the directory “**mydir**” in the current directory.

**mkdir -p one/two/mydir**

The **-p** option allows the creation of parent directories if required. So, the above command will create directories **one** and **two** if they do not exists and then finally create **mydir** inside the directory **two**.

```
amitabh@LAPTOP-8F40GMIU:~$ tree one
one
└── two
    └── mydir

3 directories, 0 files
```

# Changing ‘root’ user password

- To change the ‘root’ user password in wsl Ubuntu, use the following command-

**wsl –u root**

```
C:\Users\hp>wsl -u root
root@LAPTOP-8F40GMIU:/mnt/c/Users/hp# passwd
New password:
Retype new password:
passwd: password updated successfully
root@LAPTOP-8F40GMIU:/mnt/c/Users/hp#
```

# Switching to ‘root’ user

```
amitabh@LAPTOP-8F40GMIU:~$ pwd  
/home/amitabh  
amitabh@LAPTOP-8F40GMIU:~$ su - root  
Password:  
root@LAPTOP-8F40GMIU:~# pwd  
/root  
root@LAPTOP-8F40GMIU:~# exit  
logout  
amitabh@LAPTOP-8F40GMIU:~$
```

# rm command

**rm** command is used to remove files, directories and links from a specific directory.

By default, it does not remove directories. It works silently so we should be careful while deleting any file/directory using ‘rm’ command.

## Example-

**rm file1.txt**-> this command will delete file called ‘file1.txt’.

## Options-

**rm file1 file2 file3**

This will delete all three mentioned files.

**rm -i <filename(s)>**

It will interactively confirm the deletion of file from the user.

**rm -f <filename>**

If the file is write-protected, it will not ask for the confirmation and forcefully deletes the file.

**rm -r <dirname> or rm -R <dirname>**

It performs a tree-walk and delete all the files and sub-directories recursively from the current directory.

**rm-- <-filename>**

It will delete the file whose first character is hyphen (-).

```
amitabh@LAPTOP-8F40GMIU:~$ tree one
one
└── file1.txt
two
└── mydir
    ├── testfile1
    └── testfile2
3 directories, 3 files
amitabh@LAPTOP-8F40GMIU:~$ rm -r one
```



# rmdir command

**rmdir** command is used to remove empty directories.

By default, it does not remove directories. It works silently so we should be careful while deleting any file/directory using ‘rm’ command.

## Options-

**rmdir dir1**

This will delete single directory ‘dir1’ only if it is empty.

**rmdir dir1 dir2 dir3**

This will delete directories ‘dir1’, ‘dir2’ and ‘dir3’ only if they are empty.

**rmdir –v dir1 dir2 dir3 (verbose mode)**

This will delete directories ‘dir1’, ‘dir2’ and ‘dir3’ only if they are empty and also displays the message of removal.

**rmdir –p dir1/d1/d2/d3**

This will delete the entire given directory structured only if it is empty(no files/subdirectories should be inside).

# Creating / Merging Files

- **cat** command is used to create file or merge two or more files in Linux.

## Examples-

**cat > myfile.txt**   (*press ctrl+c to save the file*)

**cat file1 file2**

It will display the contents of file1 and file2 but not save them in any other file.

- Standard Output redirection is achieved through > symbol.

**cat file1 file2 > file3**

It will send/redirect the output of cat command to file ‘file3’.

- >> symbol will append the output in an existing file.

**cat file1 file2 >> file3**

It will send/redirect the output of cat command to file ‘file3’ and appended the output to file3.

- Standard Input redirection is achieved through < symbol.

# Merging Commands

- We can also execute two or multiple commands in a single line.
- To do so separate the commands with semicolon (;) symbol.

# Some Useful Commands

- **mv file1 file2**

It will change the name (rename) of ‘file1’ to ‘file2’.

- **mv file1 dir1/file2**

It will move the file ‘file1’ into the directory ‘dir1’ and set the filename as ‘file2’.

- **pwd**

It will display the absolute path of current directory. (Present Working Directory)

- **whoami**

It will print the logged in username.

- **man <command>**

It will display the manual/help of the given command. ‘spacebar’ to move screen forward, ‘b’ to move screen backword and ‘q’ to quit from manual.

- **file <filename>**

It will display the file type information along with filename.

- **file -b <filename>**

Will display the file type information in brief mode. Removes the filename from output.

# cp Command

- cp command is used to create copy of file(s).

- Syntax-

*cp source\_file destination*

- This command creates a copy of the **source\_file** at the specified **destination**.

- Examples-

- **cp a.txt b.txt**

This command contains two file names, it copies the contents of the first file (**a.txt**) to the second file (**b.txt**). If the second file (**b.txt**) doesn't exist, it is created, and the content is copied into it. However, if the second file (**b.txt**) already exists, it is overwritten without warning.

# cp Command- Examples

- Examples-
- **cp a.txt b.txt c.txt dir1/**

This command contains three file names and a directory (**dir1**). it copies the contents of all the three files (**a.txt, b.txt and c.txt**) to the specified directory (**dir1**).
- **cp -R dir1 dir2 or cp -r dir1 dir2**

This command contains two directory names (**dir1 and dir2**). With **-R** or **-r** option, it copies all the files from the source directory (**dir1**) to the destination directory (**dir2**).
- To Interactive copying with a warning before overwriting the destination file, we can use **-i** option with cp command.

# Practice Question

- Create the files and directories as per the below tree diagram-

```
amitabh@LAPTOP-8F40GMIU:~$ tree dir1
dir1
├── d1
└── dir2
    ├── d2
    │   ├── data1.txt
    │   └── data2.txt
    ├── dir3
    │   └── myfile
    ├── file1
    └── file2
└── file1.txt

5 directories, 6 files
```

# ls command

- The **ls** command will show the full list or content of your directory.
- By default, it will display the files in ascending order of their names.

## Options-

### **ls-I**

It will show the list of files / directories in a long list format in total 7 respective columns.

**1<sup>st</sup> column-** permissions on the file.

**2<sup>nd</sup> column-** number of links.

**3<sup>rd</sup> column-** owner of the file.

**4<sup>th</sup> column-** group that owns the file.

**5<sup>th</sup> column-** size of the file in bytes.

**6<sup>th</sup> column-** last modification date and time.

**7<sup>th</sup> column-** name of the file / directory.

# What is total in ls command ?

- The size of one memory block in Linux is generally 1024 bytes (1KB).
- When any file or directory is created it occupies at least 4KB of memory space which means 4 blocks.
- The Total represents the memory blocks occupied by the listed directories and files.

# Is command...cont.

## Options-

### **Is -a**

In Linux, hidden files start with . (dot) symbol and they are not visible in the regular directory. This option will enlist the whole list of the current directory including the hidden files.

### **Is -i or Is--inode**

Shows the index number of each file and directory.

### **Is -lh**

This command will show you the file sizes in human readable format.

### **Is -ls**

It will display the files in descending order (highest at the top) according to their size.

### **Is -lr**

It will display the files in reverse order.

# find command

**find** command is used to find files and directories in a Linux file system.

## Options-

**-name <filename>**

Search files with specific filename.

**-inum <inode\_number>**

Search file having specified i-node / index number.

**-empty**

Search only empty files and directories.

**-perm <permission\_in\_number>**

Search files and directories having specified permissions.

**-user <username>**

Search files owned by specified username.

**-type f/d**

Display regular files with ‘f’ and directories with ‘d’.

# locate command

**locate** is an utility command used to search files and directories quickly. It is more convenient and effective than ‘find’ command.

locate command doesn’t search the entire file system, but it looks through a regularly updated file database called ‘**mlocate**’ or ‘**plocate**’ in the system. To update the locate database, ‘**updatedb**’ command is used.

If any file created after updating the locate database, it will not include in the search result.

## Example-

**locate file1**-> this command will search and display all files containing the ‘file1’ pattern in the filename.

## Options-

**locate file1 | less**

This will be convenient way to see the output if there are large number of files in the output. ‘**Spacebar**’ is used to move forward, ‘**b**’ key to move backward and ‘**q**’ key to quit.

**locate -c file1 or locate--count file1**

Display only the **count** of files containing ‘file1’ pattern.

**locate file1 -n 10 or locate file1 -l 10 or locate file1--limit 10**

Limit the number of search results.

# locate command...cont.

## Options-

**locate -i <filename> or locate --ignore-case <filename>**

Ignores the case of filename.

**locate-0 <filename> or locate--null <filename>**

Displays the result containing the specified pattern but omits newline character.

**locate -h or locate--help**

Displays the usage of locate command with its available options.

# more command

- **more** command is used to display the contents of text files on command window (terminal window).
- It facilitates to display the contents one screen/page at a time in case of large files **along with the display ratio (in percentage)**.

## Options-

**more <filename>**

It will display the contents one screen/page at a time.

**more -d <filename>**

It will display the help to the user regarding navigation controls at the end of every screen.

“[Press space to continue, ‘q’ to quit.]”

**more-5 <filename>**

It will display 5 number of lines per screen of the given filename.

**more +10 <filename>**

It will start displaying the contents after leaving 10 lines of the given filename.

**more -p <filename>**

It will first clear the screen and then start displaying the contents of given filename.

# less command

- **less** is used to display the contents of a text file one screen/page at a time.
- It is fast because for large files, it does not access the entire file at a time but access the contents page-by-page.
- G- Moves to the end of the file
- g- Moves to the beginning of the file.
- **/pattern** is used to search the pattern in a file.
- PageUp, PageDown, Up Arrow and Down Arrow keys are working as usual.
- To quit from 'less' utility, we must need to press the '**q**' key.

## Options-

**less <filename>**

It will display the contents one screen/page at a time.

**less -E <filename>**

It will automatically exit when reaching at the end of the file.

**less-F <filename>**

Automatically exits if the complete file can be displayed on the first screen.

**less-N <filename>**

It will display the content of the given filename along with the line numbers.

# history command

- history command is used to view previously executed commands.
- Supported by BASH and Korn shells and not available under Bourne Shell.
- Stored under **.bash\_history** file in user's home directory.

## **history**

It will display the list of previously used commands.

## **history 5**

It will display the last/recent 5 commands used. (including this command)

**!!**

It will execute the last command used.

## **!<command\_number>**

It will display and executes the the command from history at given command\_number.

Example-      **!1201**

## **history -d <command\_number>**

It will delete the command from history at given command number position.

# head command

- **head** command is used to display the top N number of lines from the given input.
- By default, it will print first 10 lines from the given file/input.

## Options-

### **head myfile**

It will display the first 10 lines of file called ‘myfile’.

### **head -n 7 <filename> or head –7 <filename>**

It will display the first 7 lines from the given filename.

### **head -c <num\_of\_chars> <filename>**

It will display mentioned number of characters/bytes from the beginning of the given filename.

Example- head -c 8 myfile.txt

### **head -q <file 1> <file 2>.....<file N>**

This option is used with multiple files. If used, it would not show the filename. For long files, user may confuse that where first file contents are finished and contents of next file starts.

### **head -v <filename>**

It will always displays the filename before the contents of the file.

# head Examples

## ■ Examples-

**head -n-7 file1.txt**

It will ignore last 7 lines from the file ‘file1.txt’.

**head <file1> <file2> <file3>**

It will display the first 10 lines from both the files along with the filenames.

**head -q <file1> <file2> <file3>**

It will display the first 10 lines from both the files but ignores the filenames.

**head -n 5 <file1> <file2>**

It will display the first 5 lines from both the files along with the filenames.

# tail command

- **tail** command is used to display the last N number of lines from the given input.
- By default, it will print last 10 lines from the given file/input.

## Options-

**tail myfile**

It will display the last 10 lines of file called ‘myfile’.

**tail -n 7 <filename> or tail –7 <filename>**

It will display the last 7 lines from the given filename.

**tail -c <num\_of\_chars> <filename>**

It will display mentioned number of characters/bytes from the end of the given filename.

Example- tail -c 8 myfile.txt

**tail -q <file 1> <file 2>....<file N>**

This option is used with multiple files. If used, it would not show the filename.

**tail -v <filename>**

It will always displays the filename before the contents of the file.

## Extra feature not in ‘head’ command-

**tail +5 <filename> OR tail –n +5 <filename>**

It will display the contents starting from line number 5 up to end of the file.

## Watch a file for changes-

**tail-f <filename>**

It will display all the newly added lines to standard output (monitor).

# head & tail Applications

## Q.1

- Print lines between 3 and 7 from file ‘file1’. Means the actual lines need to be displayed are 4, 5 and 6.
- To achieve this we can use the pipe symbol to redirect/send the output of one command to another command.
- Solution-

tail +4 file1 | head-3

Or

head-6 file1 | tail-3

## Q.2

- Print the most recently 3 modified files from the current directory.
- Solution-

ls -t | head-3

# id command

- Prints the User and Group IDs of currently logged user if username is omitted.
- If username is given, will display the User and Group IDs of given username.

## Options-

**\$ id -g or id --group**

Prints only the effective group id.

**\$ id -G or id --groups**

Prints all the group ids user belongs to.

**\$ id -n or id--name (for ugG)**

Prints the group names instead of group ids.

**\$ id-u**

Prints only the user id of currently logged user.

**\$ id <username>**

Prints all the User and Group IDs of specified username.

# uname command

- Prints the system information.

## Options-

**\$ uname-s**

Prints the kernel name.

**\$ uname -n**

Prints the host name.

**\$ uname -r**

Prints the kernel release.

**\$ uname-v**

Prints the kernel version with release date.

**\$ uname-m**

Prints the machine's hardware type.

**\$ uname-o**

Prints the operating system name.

**\$ uname-a**

Prints all the system information.

**\$ uname-p**

Prints the processor type.

# date & timedatectl command

**date** command displays the current date and time, including the abbreviated day name, abbreviated month name, day of the month, the time separated by colons, the time zone name, and the year.

**timedatectl** command lets you set your time, date, and time-zone for your system clock.

## Options-

**timedatectl list-timezones**

It will display the list of all time zones.

**sudo timedatectl set-timezone Asia/Kolkata**

It will set the time-zone as “Asia/Kolkata”.

```
amitabh@LAPTOP-8F40GMIU:~$ sudo timedatectl set-timezone Asia/Kolkata
[sudo] password for amitabh:
amitabh@LAPTOP-8F40GMIU:~$ date
Wed Dec  4 12:42:12 IST 2024
amitabh@LAPTOP-8F40GMIU:~$
```

# date command

- **date** command is used to display the current system date and time information.

## Options and Examples-

### **date**

It will display the current system date and time. The format is-

**Tue Nov 28 10:35:59 IST 2023**

### **date -u**

It will display the system date and time in GMT(Greenwich Mean Time)/UTC(Coordinated Universal Time).

**Tue Nov 28 05:14:16 UTC 2023**

### **date -d “02/03/2010” or date--date “02/03/2010”**

It will display the given date in the date format.

**Wed Feb 3 00:00:00 IST 2010**

# date command...contd.

## Options and Examples-

### **date--date “2 years ago”**

It will display the date and time 2 years ago from the current system date and time.

```
Sun Nov 28 10:54:23 IST 2021
```

### **date--date “5 minutes ago”**

It will display the date and time 5 minutes ago from the current system date and time.

### **date--date “10 seconds ago”**

It will display the date and time 10 seconds ago from the current system date and time.

### **date--date “2 months ago”**

It will display the date and time 2 months ago from the current system date and time.

### **date--date “next tuesday” or date--date “tomorrow”**

It will display the date on next Tuesday or of tomorrow.

### **date--date “5 days”**

It will display the date 5 days after the current system date and time.

# date command...contd.

## Options and Examples-

```
$ cat datefile.txt
```

```
Sep 23 2010
```

```
Nov 27 2022
```

```
Oct 28 2009
```

```
$ date--file="datefile.txt"
```

```
Thu Sep 23 00:00:00 IST 2010
```

```
Sun Nov 27 00:00:00 IST 2022
```

```
Wed Oct 28 00:00:00 IST 2009
```

# Date- Format Specifiers

Following are the format specifiers used to display the date and time information in a specific format-

**%D:** Display date as mm/dd/yy.

**%d:** Display the day of the month (01 to 31).

**%a:** Displays the abbreviated name for weekdays (Sun to Sat).

**%A:** Displays full weekdays (Sunday to Saturday).

**%h:** Displays abbreviated month name (Jan to Dec).

**%b:** Displays abbreviated month name (Jan to Dec).

**%B:** Displays full month name(January to December).

**%m:** Displays the month of year (01 to 12).

**%y:** Displays last two digits of the year(00 to 99).

**%Y:** Display four-digit year.

**%T:** Display the time in 24 hour format as HH:MM:SS.

**%H:** Display the hour.

**%M:** Display the minute.

**%S:** Display the seconds.

**To use these specifiers, follow the syntax-**

**\$ date +format-specifier**

**Examples-**

**\$ date +%D**

**11/28/23**

**\$ date +"Today is- %B %d %Y"**

**Today is- November 28 2023**

# cal command

It is used to display the calendar on Linux terminal.

## Options and examples-

### **cal**

It will display the calendar of current month.

### **cal <mm> <yyyy>**

It will display the calendar of given/selected month and year.

### **cal <yyyy>**

It will display all the months of given year.

### **cal -j <yyyy>**

It will display the calendar of given year in Julian calendar format. In Julian calendar date is not reset to 1<sup>st</sup> after every month.

# bc command

- Stands for **Basic Calculator** used for command line calculations.
- It can perform simple arithmetic operations to complex calculations including mathematical functions.

## Options and examples-

```
$ echo '20+5' | bc
```

```
$ echo 'scale=2; 3/2' | bc
```

```
$ echo 'a=5; b=12; a+b' | bc
```

```
$ echo 'a[1]=10; a[2]=20; a[1]+a[2]' | bc
```

```
$ echo 'a=0; while(a<=5) { print a; a+=1; print "\n";}' | bc
```

```
$ echo 'x=45; sqrt(x)' | bc
```

```
$ echo 'x=5674; length(x)' | bc
```

Returns the no. of digits in x.

```
$ bc-i
```

Starts interactive mode. To quit from interactive mode type 'quit'.

# Adding User in ‘sudo’ Group

- Adding a user in **sudoers list / sudo group** allows the user to run commands with admin privileges.
- To add any user in sudo group, just give the following command-
  - **Syntax-** adduser <username> <groupname>
  - **Example-** adduser user100g2 sudo
- To confirm if the user is successfully added in sudo group, just open the /etc/group file and see the added username entry against the sudo group.
  - **\$ cat /etc/group**  
**sudo:x:27:amitabh,user100g2**

# chmod command

- **chmod** (Change Mode) command is used to change the access mode of a file.
- It actually allows to set the permissions on a file/directory like who can read the file, who can edit the file contents and who can execute the file (if file is executable).
- Symbols “r”, “w” and “x” are used to represent the read, write and execute permissions, respectively.
- These letters are collectively form a tri-group.
- There are 3 tri-groups or triads as-
  - “User/Owner” (**u**)    “Group” (**g**)              “Other” (**o**)
- All the three permissions (r, w and x) can be set on any triad (**u,g,o**).
- Octal numbers used to represent these permissions are-  
Read (4) [100]   Write (2) [010]      Execute (1) [001]

*‘a’ is used for all groups (ugo).*



## Example-

To give permissions r, w and x to the owner and r and x to the group

And r to others on file **file1.txt**-

**\$chmod 754 file1.txt OR \$chmod u+rwx,g+rx,o+r file1.txt**

**\$chmod u-x,g-x,o-wx file1.txt (to remove permissions, use- sign)**

# chmod Operators

| Operators | Definition                                  |
|-----------|---|
| '+'       | Add permissions                             |
| '-'       | Remove permissions                          |
| '='       | Set the permissions to the specified values |

## Examples-

**\$ chmod u=rwx file1**

It will apply read, write and execute permissions to the given file for the owner only.

**\$ chmod g=rw,o-x file1**

It will apply read and write permissions to the given file for the group and removes execute permission from others.

# chmod Options

## Options and examples-

**\$ chmod -v 744 <directory/filename>**

It will apply the given permissions to the directory/file and also display a message indicating permission changes.

**\$ chmod -R 744 dir1**

It will apply the given permissions to directory **dir1** recursively to all the files and directories.

**\$ chmod u+rwx <directory/filename>**

It will apply read, write and execute permissions to the given file/directory for the owner only.

**\$ chmod go-w <directory/filename>**

It will remove write permission from the given file/directory for the group and others only.

**\$ chmod u+rw, go+r <directory/filename>**

It will apply read, write and execute permissions to the given file/directory for the owner only.  
And, also apply read permission to the group and others on the specified directory/file.

# Default Permissions- umask

- Full permission set for a **file** is **666** ( read, write permission for all )
- Full permission set for a **directory** is **777** ( read, write and execute permissions )
- **umask** value is responsible to set default permissions for files and directories created by the user.
- Default umask value is set to **022**.
- umask value is subtracted from the full file permission and full directory permission.
- Therefore, when user creates any-
  - **File**- it has the default permissions **644**
  - **Directory**- it has the default permission **755**
- Default umask value can be defined under user's home directory **.bashrc** file.



```
amitabh@LAPTOP-8F40GMIU:~$ umask  
0022
```

A screenshot of a terminal window. The window title is 'amitabh@LAPTOP-8F40GMIU:~\$'. Inside the window, the command 'umask' is entered and the output '0022' is displayed. The background of the terminal window is black, and the text is white.

# chown command

- **chown** command is used to change the owner or group of a file/directory.
- Only **root** or **sudo** users can run this command.

## Options-

**chown <username> <filename/directory>**

It will change the current user/owner of the file/directory to the given username.

**chown <username>:<groupname> <filename/directory>**

It will change the current owner and group of the file/directory to the given username and groupname.

**chown :<groupname> <filename/directory>**

It will change only the current group of the file/directory to the given groupname.

**chown -c <username>:<groupname> <filename/directory>**

This option (**-c**) will display a confirmation message after changing the ownership. **(same as-v)**

**chown -v <username>:<groupname> <filename/directory>**

The verbose option (**-v**) will display a message on the screen after changing the ownership.

Prepared by- Amitabh Srivastava

# chgrp command

- chgrp command is used to change the group ownership of a file/directory.

## Options-

**chgrp <groupname> <filename>**

It will change the current group of the file to the given groupname. The groupname must be exist and can be checked within **/etc/group** file.

**chgrp <groupname> <file...1> <file...2> <file...3> <file...N>**

It will change the current group of all the files to the given groupname.

**chgrp <groupname> <directory>**

It will change the current group of the directory to the given groupname but does not change the group of files and directories inside it.

**chgrp -R <groupname> <directory>**

It will change the current group of the directory to the given groupname and also recursively change the group ownership of all folders and files inside it.

**chgrp -c <groupname> <file/directory>**

It will display a confirmation message after changing the group ownership.

# Advanced Packaging Tool (apt)

- The **apt** command is a powerful command-line tool, known as Ubuntu's Advanced Packaging Tool (APT).
- It is used to install new software packages, upgrade existing software packages, update the package list index, and even upgrade the entire Ubuntu system.
- Actions like installation and removal of packages are logged under the **/var/log/dpkg.log** file.
- Examples-

✓ Installing packages-

```
sudo apt install <package_name>
```

✓ Removing packages-

```
sudo apt remove <package_name>
```

adding the **--purge** option with remove will remove the package along with its configuration files as well, so use it with caution.

# Advanced Packaging Tool (apt)

- Examples-

- ✓ Updating package index-

To update the local package index with the latest changes made in the repositories, type the following command-

*sudo apt update*

- ✓ Upgrading packages-

To upgrade your system, first update your package index and then perform the upgrade as follows-

*sudo apt update*  
*sudo apt upgrade*

# Advanced Packaging Tool (apt)

- Options-

- ✓ \$ apt list

List packages based on package names

- ✓ \$ apt show <package\_name>

Show package details

Examples-

*apt show tree*

*apt show ncal*

- ✓ \$ apt-file list <package\_name>

Show package details

Examples-

*apt-file list tree*

**To install apt-file:**

```
# apt install apt-file
```

```
# apt update
```

# dpkg Command

- **dpkg** is a package manager for Debian-based systems.
- It can install, remove, and build packages, but unlike other package management systems, it cannot automatically download and install the dependencies required for that package.
- **APT** is newer than the **dpkg**.

List packages-

`dpkg -l` (more readable than ‘`apt list`’ command)

List package installed files-

`dpkg -L <package_name>`

Installing a package-

`dpkg -i <package_name>` (**.deb** file is required)

Removing a package-

`dpkg -r <package_name>`

# df command

- Disk Free utility provides valuable information on disk space utilization.
- Displays information about file system's disk space usage on the mounted file system.

## Options-

### **df**

It displays information about all the mounted file systems like total size, used space, usage percentage, and the mount point.

### **df <filename>**

It will display the mount information of the given filename.

### **df -h**

It will display the disk space usage in human readable form (in KB/MB/GB).

### **df --total**

It will display the grand total of disk space usage at the end.

### **df --help**

It will display the help on 'df' command.

# du command

- **Disk Usage** utility provides information of a directory or file space usage.
- Displays information about the storage consumption of files and directories.

## Options-

**du**

It displays the disk usage information of current directory. (*by default, numbers in KB*)

**du <filename>**

It will display the storage information of the given filename.

**du -h**

It will display the disk space usage in **human readable** form (in KB/MB/GB).

**du --time <filename>**

It will display the disk space usage along with the **last modification time** of the given filename.

**du -c <dirname> OR du --total <dirname>**

It will display the disk space usage along with the **total disk usage** of the given directory.

**du --help**

It will display the **help** on 'du' command.

# Linux System Administration

[ ST.2 ]

# User Management

- A user is an entity in Linux OS that can manipulate files and perform several other operations.
  - Each user has a unique id.
  - ID of 'root' user (Superuser) is always 0.
  - System user IDs starts from 1 to 999.
  - Local user IDs starts from 1000 onwards.
  - User Management Commands ('root' user)-
    - passwd
    - useradd
    - usermod
    - groupadd
    - groupdel
    - userdel
    - id
- PAM(Pluggable Authentication Module)** is an Unix integrated login framework. PAM is used to authenticate users at the time of logging process.

# Primary Group of User

- When a user is created, a primary group for that user is also automatically created which is the same name used in creating the user.
- Creating a user is automatically creating a primary group for the user. So, a user is automatically a member of its primary group.
- For example, if I create a user Kavita, Kavita group is automatically a primary group for the user Kavita.
- If you create a new user and open the “**/etc/passwd**” file where the attributes of users are found, you will notice in one of the fields, a primary group ID is automatically assigned to the user.
- Similarly, open the “**/etc/group**” file where the group information is stored, you will by default see the name of the group as the username with the corresponding group ID in the file.

# Secondary Group of User

- The secondary group is the second, third, n.. group that a user is added to.
- In other words, the secondary group is the other group a user is added to.
- A secondary group is usually created by the administrator.
- No user can have more than one primary group but a user can be added to more than one secondary group.

# passwd command

- It is used to change/modify the password of an existing user account through command line.
- The ‘root’ user can change the password of any user account but a normal user can change the password only for his/her account.

## Options-

### **passwd**

If given without any options or username, it will ask to change the account password of currently logged user.

### **passwd <username>**

It will change the password of the specified username.

### **passwd -l <username>**

It will lock the password of specified user so that user cannot login to his/her account.

### **passwd -u <username>**

It will unlock the password of specified user so that user can login to his/her account.

### **passwd -d <username>**

It will delete the password of the specified user. Means, user can login without password.

### **passwd -e <username>**

It will enforce the user to change the account password on next login attempt.

# passwd command...(cntd.)

- All the password information is stored in a file **/etc/shadow**.
- User password expiry information can be viewed using command '**chage -l <username>**'

## Options-

### **passwd -n <no. of days> <username>**

It will change the minimum number of days between password change. Means, if the no. of days is set to 2, and user changes its own password one time then he/she cannot change its own password until 2 days have passed. By default, this value is set to 'zero' (0) means, user can change its own password at any time.

### **passwd -S <username>**

It will show the password status of the user in 7 fields.

First field- User's login name

Second field- Indicates if user has a locked password (**L**), has no Password (**NP**), or has a usable password (**P**).

Third field- Date of the last password change.

Next four fields are the minimum age, maximum age (enforce to change the password), warning period, and inactivity period for the password. These ages are expressed in days.

# chpasswd command

- Routine bulk password resets are a common need in large organizations, server management, and web hosting companies.
- Manually updating passwords for numerous users is inefficient and prone to error.
- The **chpasswd** command in Linux offers a streamlined solution for updating user passwords, either in bulk or individually.
- It allows system administrators to change passwords for multiple users simultaneously.
- It simplifies the process of changing passwords by reading **username-password** pairs from an input or a file and applying the changes automatically.

## Example:1- To change a single user password

**Syntax-**      **echo 'username:password' | chpasswd**  
                  **\$ echo 'user001:12345' | chpasswd**

# chpasswd command: Example-2

- Create a text file and write the username and passwords in the following format-

username1:password1

username2:password2

.....

.....

usernameN:passwordN

- Now, save the file and give the chpasswd command with following syntax-

**chpasswd < filename**

## Example- To change multiple user passwords

```
$ cat > users-passwords.txt
```

```
user001:user001@123
```

```
user002:user002@xyz
```

```
user003:321@user003
```

```
$ chpasswd < users-passwords.txt
```

OR

```
$ cat users-passwords.txt | chpasswd
```

# adduser command

- It is used to create a new user account through command line interactively.

```
root@LAPTOP-8F40GMIU:~# adduser testuser1
info: Adding user 'testuser1' ...
info: Selecting UID/GID from range 1000 to 59999 ...
info: Adding new group 'testuser1' (1001) ...
info: Adding new user 'testuser1' (1001) with group 'testuser1 (1001)' ...
info: Creating home directory '/home/testuser1' ...
info: Copying files from '/etc/skel' ...
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for testuser1
Enter the new value, or press ENTER for the default
      Full Name []:
      Room Number []:
      Work Phone []:
      Home Phone []:
      Other []:
Is the information correct? [Y/n] y
info: Adding new user 'testuser1' to supplemental / extra groups 'users'
info: Adding user 'testuser1' to group 'users' ...
root@LAPTOP-8F40GMIU:~#
```

## Benefits of ‘adduser’ command

- Home directory is automatically created under **/home** directory.
- Ownership of user for user & group assigns to the home directory.
- Prompt is properly configured for the user.
- Default files can be copied from **/etc/skel** directory.
- Primary group is also created for the user with same name as username.

# useradd command

- It is used to add/create new user accounts through command line.

## Options-

**useradd -c <comment> <username>**

It will add a new user and add the specified comment in /etc/passwd file.

**useradd -d <homedirectorypath> <username>**

It will add a new user and set the home directory as specified.

**useradd -m <username>**

It will add a new user and creates the home directory as specified.

**useradd -u <userid> <username>**

It will create new user with specified user id.

**useradd -e <yyyy-mm-dd> <username>**

It will create new user and set the account expiry date as mentioned.

**useradd --help**

It will display the help section of useradd command.

# usermod command

- It is used to change/modify the properties of an existing user through command line.
- After creating a user, if you want to change its home directory or group information, etc. then you can use ‘usermod’ command.

## Options-

**usermod -c “Local user” <username>**

It will add a comment “Local user” for the given username. Comments are stored in 5<sup>th</sup> field of /etc/passwd file.

**usermod -d /home/ram ram001**

It will change the default home directory of user ram001 to /home/ram. This new home directory must be created separately. Home directories are stored in 6<sup>th</sup> field of /etc/passwd file.

**usermod -e 2023-12-25 <username>**

It will set the expiry date for the given username. Expiry dates are stored inside /etc/shadow file.

Check expiry information using command- ‘**chage -l <username>**’

**usermod -g <newgroup> <username>**

It will change the primary group for the given username to <newgroup>. Group-id is stored in 4<sup>th</sup> field of /etc/passwd file. All group names are stored in /etc/group file.

**usermod -aG sudo <username>**

It will add the given username to the ‘sudo’ group. ‘**id**’ command is used to check the group information.

# usermod: Practice Exercise

- Create a new user account by name: **testuser001**
- Now, do the following-
  - Change testuser001 username to **test\_user1**
  - Set **test\_user1** home directory to **/home/test\_user1**
  - Add **test\_user1** to sudo group to grant him administrative privileges.

## Solution-

Change testuser001 username to **test\_user1**.

```
$ sudo usermod -l test_user1 testuser001
```

Set **test\_user1** home directory to **/home/test\_user1**

```
$ sudo usermod -d /home/test_user1 -m test_user1
```

Add **test\_user1** to sudo group to grant him administrative privileges.

```
$ sudo usermod -aG sudo test_user1
```

# chage command

- It is used to manage user's password expiry information and account aging information.
- With **chage** command, administrators can view and modify password expiry details, set mandatory password change intervals and specify account expiry date.

## Options-

**\$ chage -l <username>**

It will display the account aging information of specified user.

**\$ chage -d 2024-12-31 newuser001**

Will set the last password change date to the specified date in the command for user 'newuser001'.

**\$ chage -E 2025-12-31 newuser001**

use **-1** with **-E** option to remove the expiration date.

Will set the account expiration date to 31<sup>st</sup> December 2025 for user 'newuser001'.

**\$ chage -W 3 newuser001**

Will set the warning period (in days) before the password expires.

**\$ chage -M 5 newuser001**

user **-m** for minimum no. of days between password change.

Will set the maximum number of days between password change.

# chage: Practice Question

You are administering a Linux system and need to manage password expiration policies for a user named **test\_user1**. Perform the following actions using the **chage** command-

1. Set the maximum password age for test\_user1 to 6 days.
2. Set the minimum password age for test\_user1 to 3 days.
3. Set the password expiration warning to 14 days before the password expires.
4. Set the account to expire on 28<sup>th</sup> October 2025.

## Solution-

1. `$ sudo chage -M 6 test_user1`
2. `$ sudo chage -m 3 test_user1`
3. `$ sudo chage -W 14 test_user1`
4. `$ sudo chage -E 2025-10-28 test_user1`

*Check your result with following command-*

**\$ chage -l test\_user1**

# groupadd command

- It is used to add a new group.
- Using groups, we can group together a number of users, and set privileges and permissions for the entire group.

## Options-

**groupadd <grpname>**

It will create the specified group.

**groupadd -g <grpid> <grpname>**

It will create the specified group with mentioned group id.

**groupadd -r <grpname>**

It will create the specified system group. The IDs of system groups are chosen from a range 1 to 999 defined for system groups in the configuration file. The IDs of normal users start from 1000.

# groupdel command

- It is used to delete an existing group.

## Options-

**groupdel <grpname>**

It will delete the specified group.

**groupdel -f <grpname>**

It will delete the specified group even if it is the primary group of a user.

**groupdel -h or groupdel --help**

It will display the help of groupdel command with all its options.

# userdel command

- It is used to delete an existing user account.

## Options-

**userdel <username>**

It will delete the specified user account.

**userdel -f <username>**

It will delete the specified user even if the user still logged in.

**userdel -r <username>**

It will also delete the user's home directory.

**userdel -h or userdel --help**

It will display the help of userdel command with all options.

# Check for User's Home Directory Path

```
amitabh@LAPTOP-8F40GMIU:~$ echo ~  
/home/amitabh  
amitabh@LAPTOP-8F40GMIU:~$ echo ~testuser1  
/home/testuser1  
amitabh@LAPTOP-8F40GMIU:~$ echo ~root  
/root  
amitabh@LAPTOP-8F40GMIU:~$
```

# Blocking User Account

- To block a user's account, follow the below steps-

## Step-1-

Login as 'root' user.

## Step-2-

Run the following command-

```
# chsh -s /usr/sbin/nologin <username>
```

## Step-3-

Exit from 'root' user and try to login with blocked username.

# Blocking Account with Customize Message

- To block a user's account with customize message, follow the below steps-

## Step-1-

Login as 'root' user.

## Step-2-

Create a file with name 'security' under /home and write following contents in it-

```
#!/usr/bin/tail +2
```

This account has been temporarily blocked due to SECURITY BREACH!

Please contact your system administrator or call at 0512-06321.

## Step-3-

Run the following command-

```
# chmod 755 security
```

## Step-4-

Run the following command-

```
# chsh -s /home/security <username>
```

To unblock give below command as  
'root'-

```
# chsh -s /bin/bash <username>
```

## Practice Question

- Add 3 users as Ram, Shiva and Shree.
- Secondary group of Shiva and Shree should be ‘tester’.
- Block Ram’s account with the following message-

Your account has been locked!

Please contact your system administrator.

# Vi or Vim Editor

- Vi editor is elaborated as **visual** editor.
- Available to every Unix system.
- Available to all Linux distros.
- Improved version of Vi is known as **Vim**.
- Used to create simple text files in Linux using command line interface.
- Two vi modes- *Command* mode and *Insert* mode.
- To switch from command mode to insert mode press “i” key.
- To switch from insert mode to command mode press ‘Esc’ key.
- By default, Vi editor starts in command mode.

# vi Editor Quick Reference Chart

Execute in Command Mode

| Save Files and Exit  |                           | Cursor Movement |                             | Add/Append Text      |                             |
|----------------------|---------------------------|-----------------|-----------------------------|----------------------|-----------------------------|
| :w                   | write buffer to disk      | h               | left                        | a                    | append after cursor         |
| :w file              | write buffer to file      | j               | down                        | A                    | append to end of line       |
| :wl file             | write Absolutely          | k               | up                          | i                    | insert before cursor        |
| :wq                  | write buffer and quit     | l               | right                       | 5i                   | insert 5 times              |
| :q                   | quit                      |                 |                             | I                    | insert at beginning of line |
| :q!                  | quit and discard buffer   | 0               | go to beginning of line     |                      |                             |
| :c!                  | redit and discard changes | \$              | go to end of line           |                      |                             |
| Move and Insert Text |                           | %               | go to match                 | Add New Lines        |                             |
| :3,8d                | delete lines 3-8          | G               | go to last line             | o                    | new line below cursor       |
| :4,9m 12             | move lines 4-9 to 12      | 3G              | go to line 3                | O                    | new line above cursor       |
| :2,9m 13             | copy lines 2-5 to 13      | W               | go forward 1 word           | Change Text          |                             |
| :59w file            | write lines 5-9 to file   | 3W              | go forward 3 words          | cw                   | change word                 |
| Cancel Edit Function |                           | B               | go back 1 word              | 3cw                  | change 3 words              |
| u                    | undo last change          | 3B              | go back 3 words             | C                    | change line                 |
| Ctrl+r               | do last change again      | Delete Text     |                             | r                    | replace 1 character         |
| Search Functions     |                           | x               | delete 1 character          | R                    | replace line                |
| /exp                 | go forward to expression  | dw              | delete 1 word               | Copy and Insert Text |                             |
| ?exp                 | go backward to expression | dd              | delete 1 line               | yy                   | copy a line                 |
| n                    | repeat previous search    | D               | delete to end of line       | 3yy                  | copy 3 lines                |
| N                    | reverse previous search   | d0              | delete to beginning of line | p                    | put below cursor            |
|                      |                           | dG              | delete to end of file       | P                    | put above cursor            |
|                      |                           | 4dd             | delete 4 lines              |                      |                             |

# Pico/Nano Editor

- **Nano** is a user-friendly, simple and WYSIWYG(**What You See Is What You Get**) text editor.
- Unlike vim editor or any other command-line editor, it doesn't have any mode.
- Available as PICO editor in modern Linux distros.
- To create a file using nano editor, give the following command-
  - **nano <filename>**
- To save a file- **Ctrl + o**
- To cut the selection- **Ctrl + k**
- To paste- **Ctrl + u**
- To search and replace- **Ctrl + \**
- To read/insert another file text- **Ctrl + r**
- To copy and paste- **Ctrl + 6, Alt + 6 and Ctrl + u**
- To quit from nano editor- **Ctrl + x**

# grep command

- Grep (Global Regular Expression Print) is a filter command that searches a given pattern of characters in a file.
- It displays all the lines containing that given pattern.

## Options and Examples-

**grep linux about.txt**

It searches the given pattern “linux” in about.txt file.

**grep -i linux about.txt**

It will displays all the lines containing the given pattern by ignoring the case of pattern.

**grep -c Linux about.txt**

Will display only the count of matches.

**grep -l “linux” \***

It will display the filenames that matches that pattern.

**grep -w “user” about.txt**

It will display the lines having match pattern as a whole word and not as a substring (default behaviour).

```
amitabh123@Amitabh-laptop:~$ cat about.txt
Linux is great Operating System. It was developed by Linus Torvalds.
It has many many different distros as it has open source license.
Linux is easy to learn. It is a multiuser OS.
It is very user friendly OS.
```

# grep command...contd.

## Options and Examples-

**grep -o “Linux” about.txt**

It will display all the matched patterns only and not the entire line containing matched pattern.

**grep -n “Linux” about.txt**

It will displays all the lines containing the given pattern along with the line numbers.

**grep -v “Linux” about.txt**

It will displays all the lines NOT containing the given pattern.

**grep “^Linux” about.txt**

It will displays all the lines containing the given pattern at the beginning of line.

**grep “OS.\$” about.txt**

It will displays all the lines containing the given pattern at the end of line.

**grep -e “Linux” -e “Linus” about.txt OR grep -E “Linux|Linus” about.txt**

Used to search multiple patterns in a file.

# grep command...contd.

## Options and Examples-

**grep -A1 "Linus" about.txt**

It will displays all the lines containing the given pattern along with one line below the matched one.

**grep -B1 "Linus" about.txt**

It will displays all the lines containing the given pattern along with one line above the matched one.

**grep -C1 "source" about.txt**

It will displays all the lines containing the given pattern along with one line above and one line below the matched one.

# grep -f option

**grep -f** will take the pattern from the given file (one pattern per line) and searches inside the given file.

It will display all the lines containing that pattern.

## Example-

```
amitabh@LAPTOP-8F40GMIU:~$ cat patterns.txt
Linux
Linus
OS
amitabh@LAPTOP-8F40GMIU:~$
```

```
$ grep -f patterns.txt about.txt
```

# grep -r OR -R option

**grep -r / -R** will look for the given pattern in the given directory recursively in all files and displays all the lines containing matched pattern.

## Example-

```
$ mkdir dir1; cd dir1
```

```
$ cat file1.txt  
Linux is a user friendly OS.  
It is multiuser OS.  
It has many distros.
```

```
$ cd ..
```

```
$ grep -r user *
```

```
$ cat file2.txt  
Windows is also a user friendly OS.  
It is developed by Microsoft Corp.
```

# grep command with pipe

## Examples-

**ls -l | grep "file"**

It will display all the files having name containing the given pattern.

**ls -l /etc | grep "Nov"**

It will display all the files from /etc directory having November in the month field.

To display all the files having extension .txt.

**ls -l | grep ".txt\$"**

To display total no. of directories.

**ls -l | grep -c ^d**

To display all the directories only.

**ls -l | grep "^d"**

To display the user information of the users those names start with the word user.

**cat /etc/passwd | grep ^user**

To display the group information named “sudo”.

**cat /etc/group | grep "sudo"**

# Grep- Practice Question

## Filename: Poem.txt

In the world of code, there's the Linux way,  
Then freedom to build, create, and play.  
Those who know its power, feel the delight,  
That open-source world, a beacon of light.  
The kernel hums quietly, efficient and true,  
Then comes the magic, as commands break through.  
Those who choose Linux, know what it's worth!

## Question-

**Write a grep command to display all those lines with their line numbers containing words either 'then' or 'those'.**

## Solution-

```
cat poem.txt | grep -niE "then|those"
```

# egrep command

- Egrep (Extended Global Regular Expression Print) is a filter command that searches a file for a given pattern of characters.
- It is faster than grep.
- Difference between grep and egrep is that egrep uses extended regular expressions.
- All the options are same as grep command.

Example- \$ cat file1.txt | egrep “[A-Za-z]”

Hello how are you?  
I am fine.  
How about you  
I am fine too.  
The End

Output-

How about you  
The End

# Regular Expressions in Linux

| <u>Symbol</u> | <u>Description</u>  |
|---------------|---|
| .             | It is called wild card character, matches any one character other than newline. |
| ^             | Matches the beginning of a line/string.   |
| \$            | Matches the end of line/string.   |
| *             | Matches zero or more occurrences of the preceding character.                    |
| ?             | Makes the preceding character optional, matching zero or one.                   |
| { }           | Used to define a specific quantity of characters to match.                      |
| \             | Used to escape following character.   |
| [ ]           | Used to match any of the characters.  |
| +             | Matches one or more occurrences of the preceding character.                     |

# Regular Expressions with egrep

- **\$ egrep “^Linux” about.txt**  
Matches all lines start with ‘Linux’.
- **\$ egrep “OS.\$” about.txt**  
Matches all lines end with ‘OS’.
- **\$ egrep “[aeiou]” about.txt**  
Matches all the vowels.
- **\$ egrep “[a-h]” about.txt**  
Matches lowercase letters from a to h.
- **\$ egrep “[0-9]{4}” about.txt**  
Matches digits from range 0 to 9 with minimum length of 4.
- **\$ egrep “[Pp]a\*s\*words” about.txt**  
Matches both spellings “Passwords” and “Paaswords”.

```
amitabh@LAPTOP-8F40GMIU:~$ cat about.txt
Linux is great Operating System. It was developed by Linus Torvalds in 1991.
It has many different ^"distros"^. Passwords can be set as blank for any user.
Linux is easy to learn. It is a multiuser OS.
It is very user friendly OS. Default umask value is 0022, but read as 022.
Paaswords are stored in encrypted form.
```

- **\$ egrep “\^” about.txt**  
Matches all ^ symbols.

- **\$ egrep “Linu[a-z]?” about.txt**  
Matches all “Linux” and “Linus” because ? means “Zero or More”.

# egrep: Assignment-1

- Suppose you have a file named “contacts.txt” in which some email addresses are stored-

```
alok.gupta@example.com
ram_pratap@company.org
gourav@domain.co
james@my-domain.net
alok123@school.edu
smart123.email@website.com
hellofriend@sub.domain.com
invalid-email@com
noreply@domain123.com
geeta17@domain.io
amitabh.sri@rediffmail.net.org
```

*Think how to display only  
invalid email address?*

- Write the egrep command with correct regular expression that matches only valid email addresses.
- Solution-

```
$ egrep '^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]+$' contacts.txt
```

# egrep: More Examples

- File named “mypasswords.txt” contain some passwords-
- **Select passwords with at least one digit in it-**  
\$ egrep “[0-9]” mypasswords.txt
- **Select passwords with at least one uppercase letter in it-**  
\$ egrep “[A-Z]” mypasswords.txt
- **Write egrep command to select passwords with exactly 8 characters long-**  
\$ egrep “^.{8}\$\$” mypasswords.txt
- **Select passwords with at least 8 characters long-**  
\$ egrep “^.{8,}” mypasswords.txt
- **Select passwords with at least one special symbol-**  
\$ egrep “[^a-zA-Z0-9]” mypasswords.txt



password123  
1234pass!  
abc@1234  
mypass7  
12@Pw!21  
hello123!  
1234@567  
h33L0!

# egrep: Assignment-2

- File named “mypasswords.txt” contains following passwords-

```
password123  
1234pass!  
abc@1234  
mypass7  
12@Pw!21  
hello123!  
1234@567  
h33L0!
```

- Write the egrep command with correct regular expression pattern that matches the passwords having **exactly** one uppercase and one lowercase letter in it.
- Solution-

```
$ egrep '[a-z].*[A-Z] | [A-Z].*[a-z]' mypasswords.txt
```

# sort command

- It is used to sort a file and arrange the records/lines in a particular order.
- By default, files are sorted by assuming the contents are ASCII (characters).

## Features-

- Sorts the contents of a text file line-by-line.
- Supports sorting alphabetically, by number, by months and in reverse order.
- It can also remove duplicate records.
- Supports sorting on columns/fields. (default field separator is blank space)

## Options & Examples-

|  |                          |   |
|--|--------------------------|---|
| <pre>amitabh123@Amitabh-laptop:~\$ cat names</pre> | <pre>\$ sort names</pre> | <pre>\$ sort -f names (ignore-case)</pre> |
| Deepak   | Alok                     | Alok                                      |
| amit   | Chandra                  | amit                                      |
| bhaskar  | Deepak                   | bhaskar                                   |
| Harish   | Harish                   | Chandra                                   |
| Alok   | Lalit                    | Deepak                                    |
| Chandra  | Om                       | Harish                                    |
| Om   | amit                     | Lalit                                     |
| Lalit  | bhaskar                  | mahesh                                    |
| mahesh   | mahesh                   | Om  |

# sort command...contd.

## Options & Examples-

**\$ sort -r names**

Sort in reverse order.

```
mahesh  
bhaskar  
amit  
Om  
Lalit  
Harish  
Deepak  
Chandra  
Alok
```

**!! TRICK !!**  
**Try 'tac' command to display  
lines in reverse order  
(last line first).**

File having numbers  
(numeric sort)-

**\$ cat > numbers**

```
12  
23  
5  
100  
67  
32  
16  
4  
28  
45  
^C
```

Sorting is now correct-

**\$ sort -n numbers**

```
4  
5  
12  
16  
23  
28  
32  
45  
67  
100
```

Sorting is not correct-

**\$ sort numbers**

```
100  
12  
16  
23  
28  
32  
4  
45  
5  
67
```

**Think how to sort  
numbers in  
descending order?**

# sort command...contd.

## ➤ Sorting on Columns-

**\$ cat items**

|     |            |
|-----|------------|
| 12  | Laptop     |
| 3   | Mouse      |
| 100 | Cables     |
| 68  | Usb drives |
| 45  | Monitors   |
| 55  | Watches    |

**\$ sort -k2 items**

|     |            |
|-----|------------|
| 100 | Cables     |
| 12  | Laptop     |
| 45  | Monitors   |
| 3   | Mouse      |
| 68  | Usb drives |
| 55  | Watches    |

**\$ sort -nk1 items**

|     |            |
|-----|------------|
| 3   | Mouse      |
| 12  | Laptop     |
| 45  | Monitors   |
| 55  | Watches    |
| 68  | Usb drives |
| 100 | Cables     |

## ➤ Storing sort result in a file-

**\$ sort -k2 -o <output filename> <input filename>**

OR

**\$ sort <filename> > <output filename>**

# sort command...contd.

## ➤ Check whether a file is sorted-

```
$ cat fruits  
Apple  
Banana  
Cherry  
Orange
```

```
$ sort -c fruits  
no output means  
file is sorted!
```

```
$ cat fruits  
Apple  
Cherry  
Banana  
Orange
```

```
$ sort -c fruits  
sort- fruits-3- disorder- Banana
```

## ➤ Remove duplicates-

```
$ cat fruits  
Apple  
Banana  
Cherry  
Orange  
Banana  
Papaya  
Cherry  
Mango
```

```
$ sort -u fruits  
Apple  
Banana  
Cherry  
Mango  
Orange  
Papaya
```

# sort command...contd.

## ➤ Sort by Month names-

\$ cat months

February  
March  
April  
September  
May  
January

\$ sort -M months

January  
February  
March  
April  
May  
September

\$ sort --month-sort months

January  
February  
March  
April  
May  
September

## ➤ How to list the files/directories on ascending order of their month?

### Solution-

\$ ls -l | sort -Mk6

# Sort on Multiple Columns

Example File- items.txt

|     |             |       |
|-----|-------------|-------|
| 12  | Laptop      | 37500 |
| 3   | Mouse       | 250   |
| 100 | Cable       | 180   |
| 55  | USB_Drive   | 1050  |
| 45  | Monitor     | 7500  |
| 55  | Smart_Watch | 3100  |
| 3   | Mouse       | 310   |
| 55  | Mobile      | 12999 |
| 37  | Earphone    | 2100  |
| 12  | Speaker     | 750   |

To sort the rows by column 1-

\$ sort -k1,1n items.txt

|     |             |       |
|-----|-------------|-------|
| 3   | Mouse       | 250   |
| 3   | Mouse       | 310   |
| 12  | Laptop      | 37500 |
| 12  | Speaker     | 750   |
| 37  | Earphone    | 2100  |
| 45  | Monitor     | 7500  |
| 55  | Mobile      | 12999 |
| 55  | Smart_Watch | 3100  |
| 55  | USB_Drive   | 1050  |
| 100 | Cable       | 180   |

To sort the rows by column 1  
and then by column 2  
in reverse order-

\$ sort -k1,1n -k2,2r items.txt

|     |             |       |
|-----|-------------|-------|
| 3   | Mouse       | 250   |
| 3   | Mouse       | 310   |
| 12  | Speaker     | 750   |
| 12  | Laptop      | 37500 |
| 37  | Earphone    | 2100  |
| 45  | Monitor     | 7500  |
| 55  | USB_Drive   | 1050  |
| 55  | Smart_Watch | 3100  |
| 55  | Mobile      | 12999 |
| 100 | Cable       | 180   |

# Sort on Multiple Columns...cont.

Example File- items.txt

|     |             |       |
|-----|-------------|-------|
| 12  | Laptop      | 37500 |
| 3   | Mouse       | 250   |
| 100 | Cable       | 180   |
| 55  | USB_Drive   | 1050  |
| 45  | Monitor     | 7500  |
| 55  | Smart_Watch | 3100  |
| 3   | Mouse       | 310   |
| 55  | Mobile      | 12999 |
| 37  | Earphone    | 2100  |
| 12  | Speaker     | 750   |

To sort the rows by column 1 and then by column 2 in reverse and then by column 3 in descending order-

\$ sort -k1,1n -k2,2r -k3,3nr items.txt

|     |             |       |
|-----|-------------|-------|
| 3   | Mouse       | 310   |
| 3   | Mouse       | 250   |
| 12  | Speaker     | 750   |
| 12  | Laptop      | 37500 |
| 37  | Earphone    | 2100  |
| 45  | Monitor     | 7500  |
| 55  | USB_Drive   | 1050  |
| 55  | Smart_Watch | 3100  |
| 55  | Mobile      | 12999 |
| 100 | Cable       | 180   |

# Sort: Practice Exercise

- Create a file **data.txt** with following data-

**Q1.** Sort the file by Profession in descending order and then by age in descending order?

**\$ sort -t "," -k3,3r -k2,2nr data.txt**

**Q2.** Sort the file by country in ascending order and then by name in descending order?

**\$ sort -t "," -k4,4 -k1,1r data.txt**

**Q3.** Sort the file by country in ascending order and then by profession in descending order and then by age in descending order?

**\$ sort -t "," -k4,4 -k3,3r -k2,2nr data.txt**

Sandeep, 32, Engineer, Canada  
Ram, 27, Doctor, Australia  
Chirag, 35, Artist, Canada  
Deepak, 28, Teacher, USA  
Kavita, 43, Doctor, Australia  
Alok, 33, Artist, USA  
Gourav, 25, Engineer, Canada  
James, 24, Teacher, USA

# WC command

- Stands for **Word Count** and used mainly for counting of characters in a file.
- By default, it produces 4 columnar output- no. of lines, word count, char(byte) count and filename, respectively.

## Options-

**wc <filename>**

It will produce 4 columnar output- no. of lines, word count, char(byte) count and filename, respectively.

**wc <filename1> <filename2> <filename..N>**

It will produce 4 columnar output for all the filename mentioned including totals of all files.

**wc -l <filename1>**

It will display no. of lines for the mentioned filename.

**wc -w <filename1>**

It will display no. of words for the mentioned filename.

**wc -c <filename1> OR wc -m <filename>**

It will display no. of characters (bytes) for the mentioned filename.

# wc command...contd.

## Options-

### **wc -L <filename>**

It will display the length of longest line (characters count) of mentioned filename.

### **wc--version**

It will display the version, author and copyright information of wc command.

## Examples-

### **ls | wc -l**

To count the no. of files and directories in the current directory.

### **cat /etc/passwd | wc -l**

To count the total no. of users.

# cut command

- Used to cut out the specific sections from each line of files and display the result to standard output.
- It slices the line by character(byte) or column/field position and extracts the text.

## Options and Examples-

**\$ cat Employees**

```
SN,Ecode,Name,Dept,Salary  
1,emp001,Alok,Sales,25000  
2,emp002,Deepak,Accounts,45000  
3,emp003,Firoz,Sales,28000  
4,emp004,Ram,IT,78000  
5,emp005,Gopal,IT,86000
```

**\$ cut -c 1 Employees**

```
S  
1  
2  
3  
4  
5
```

**\$ cut -c 1,4 Employees**

```
SE  
1m  
2m  
3m  
4m  
5m
```

**\$ cut -c 3-8 Employees**

```
,Ecode  
emp001  
emp002  
emp003  
emp004  
emp005
```

# cut command...contd.

## Options and Examples-

```
$ cut -c 3- Employees
```

```
,Ecode,Name,Dept,Salary  
emp001,Alok,Sales,25000  
emp002,Deepak,Accounts,45000  
emp003,Firoz,Sales,28000  
emp004,Ram,IT,78000  
emp005,Gopal,IT,86000
```

```
$ cut -c -3 Employees
```

```
SN,  
1,e  
2,e  
3,e  
4,e  
5,e
```

```
$ cut -d "," -f 3 Employees
```

```
Name  
Alok  
Deepak  
Firoz  
Ram  
Gopal
```

```
$ cut -d "," -f3,5 Employees
```

```
Name,Salary  
Alok,25000  
Deepak,45000  
Firoz,28000  
Ram,78000  
Gopal,86000
```

# cut command...contd.

## Options and Examples-

**\$ cut -d "," -f 3- Employees**

Name ,Dept ,Salary  
Alok ,Sales ,25000  
Deepak ,Accounts ,45000  
Firoz ,Sales ,28000  
Ram ,IT ,78000  
Gopal ,IT ,86000

**\$ cut --complement -d "," -f 3- Employees**

SN ,Ecode  
1 ,emp001  
2 ,emp002  
3 ,emp003  
4 ,emp004  
5 ,emp005

**\$ cut -d "," -f 3- --output-delimiter="|" Employees**

Name |Dept |Salary  
Alok |Sales |25000  
Deepak |Accounts |45000  
Firoz |Sales |28000  
Ram |IT |78000  
Gopal |IT |86000

**\$ cut --version**

Displays the version, author and copyright information.

**\$ cat /etc/passwd | cut -d ":" -f1**

Display the names of all the user accounts.

# uniq command

- The **uniq** command displays duplicate lines only once from the sorted records/file.

## Options and Examples-

```
$ cat fruits.txt
```

```
Apple  
Banana  
Orange  
Cherry  
Orange  
Banana  
Papaya  
Apple
```

```
$ sort fruits.txt | uniq
```

```
Apple  
Banana  
Cherry  
Orange  
Papaya
```

```
$ sort fruits.txt | uniq -c
```

Displays the count of repeated lines.

```
2 Apple  
2 Banana  
1 Cherry  
2 Orange  
1 Papaya
```

```
$ sort fruits.txt | uniq -d (--repeated)
```

Print only lines that are repeated and discards non-duplicate lines.

```
Apple  
Banana  
Orange
```

# uniq command...cont.

## Options and Examples-

```
$ cat fruits.txt
```

```
Apple  
Banana  
Orange  
Cherry  
Orange  
Banana  
Papaya  
Apple
```

```
$ sort fruits.txt | uniq -D (--all-repeated)
```

```
Apple  
Apple  
Banana  
Banana  
Orange  
Orange
```

```
$ sort fruits.txt | uniq -u (--unique)  
Print only non-duplicate lines.
```

```
Cherry  
Papaya
```

```
$ sort fruits.txt | uniq -i (--ignore-case)
```

Ignores the case sensitivity of characters.

```
$ uniq --version
```

Prints the version details of uniq command.

# Practice Question-1

Consider the file ‘Employees’-

```
SN,Ecode,Name,Dept,Salary
1,emp001,Alok,Sales,25000
2,emp002,Deepak,Accounts,45000
3,emp003,Firoz,Sales,28000
4,emp004,Ram,IT,78000
5,emp005,Gopal,IT,86000
```

## Question-

Display the fourth field (Dept.) in descending order only with the uniq values without the heading (Dept).

```
Sales
IT
Accounts
```

## Solution-

```
$ cut -d "," -f4 Employees | tail +2 | sort -r | uniq
```

# Practice Question-2

Consider the file ‘Emp’-

```
Firstname,Lastname,empcode,dept
Deepak,Gupta,em001,Sales
Geeta,Bajaj,em012,Accounts
Alok,Saxena,em102,Security
Tina,Malik,em004,R&D
Ronit,Misra,em405,Administration
Gyan,Sharma,em045,Agriculture
Amar,Mehta,em410,Statistics
Dheeraj,Saxena,em010,Sports
```

## Question-

Display the employees firstname and Department whose firstname start with letter ‘D’ and department name starts with letter ‘S’ AND also the employees whose firstname starts with either ‘G’ or ‘R’ and department name starts with letter ‘A’. Header row not required.

## Solution-

```
$ cut -d "," -f 1,4 Emp | tail +2 | grep -E '^D.*,[S]|^([GR]).*,[A]'
```

# uptime command

- This command is used to find out the time for which the system has been in active(running) state.
- It returns a set of values that include the **current time**, and the amount of **time the system is in running state**, number of **users currently logged** into, and the **load time** for the past 1, 5 and 15 minutes respectively.

**\$ uptime**

```
13:36:51 up 58 min, 1 user, load average: 0.00, 0.00, 0.00
```

**Options-**

**\$ uptime -p**

Returns the total no. of hours and minutes of system active.

**\$ uptime -s**

Returns the starting time by when the system is running.

```
2023-11-29 12:38:03
```

**\$ uptime -V or uptime --version**

Returns the version information.

```
uptime from procps-ng 3.3.17
```

# Process Management

- Involves controlling and monitoring the processes running on a Linux system.
- Includes managing process resources, scheduling processes to run on the CPU, and terminating processes when necessary.
- Understanding the different types of processes, their states, and the available commands for process management are important for managing processes effectively.
- **Process-**

A process is an instance of a program currently running on a computer system. In Linux, processes are managed by the operating system's kernel, which allocates system resources and schedules processes to run on the CPU.

In Linux, there are two types of processes-

**Foreground processes**- Processes that require input from the user and are characterized by their interactivity. For instance, a foreground process would be a running Office application on the Linux system.

**Background processes**- Non-interactive operations carried out in the background that do not require any participation from the user. Antivirus software is an example of a Background Process.

Process can be a system process or user process. **System processes** are initiated by the kernel, while users initiate **User processes**.

# Process States in Linux

- The state of a running process at any point in time is called the context of a process.
- In Linux, a process can be in one of the five states-
  - **Running-**
    - ✓ The process is currently executing on the CPU.
  - **Sleeping-**
    - ✓ The process is waiting for a resource to become available.
  - **Stopped-**
    - ✓ The process has been terminated by a user.
  - **Zombie-**
    - ✓ The process has completed execution but has not yet been cleaned by the system.
  - **Orphan-**
    - ✓ The parent process of the current process has been terminated.

# Process Identity

- A process is identified by the process id (PID).
- When a process is created, it is assigned a unique identifier called the process id.
- This process id is useful when the process needs to be referred to by a user or by a program.
- The process id is used to specify processes to the operating system.
- When a process needs to be controlled, the process id is given as an argument to the system calls/commands to identify the process.

# Process Management Commands

- Linux provides several commands for managing processes.
- Few important commands are-

| <u>Commands</u> | <u>Description</u>  |
|-----------------|---|
| <u>ps</u>       | Displays information about the currently running processes.                     |
| <u>top</u>      | Provides real-time information about system processes and their resource usage. |
| <u>kill</u>     | Terminates a process by sending a signal to it.                                 |
| <u>nice</u>     | Adjusts the priority of a process.  |
| <u>renice</u>   | Changes the priority of a running process.                                      |
| <u>ps PID</u>   | Shows the state of an exact process.  |
| <u>bg</u>       | For sending a running process to the background.                                |
| <u>fg</u>       | For running a stopped process in the foreground.                                |

# Foreground Processes

- By default, every process that you start runs in the foreground. It gets its input from the keyboard and sends its output to the screen.
- For example, if you give the **vi file1.txt** command, it creates a new process to run the vi- editor application which runs in the foreground and wait for the user input.
- No other commands can be run in the foreground until the process related to vi-editor finishes.

# Background Processes

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

# ps command

- The **ps** (Process Status) command is used to find the process ID (PID) of the process that needs to be managed.
- For example, if we want to kill/terminate a process, we first need to know the PID of that exact process.
- It reads the process information from the virtual files in /proc file-system.

## Options and Examples-

**\$ ps**

| PID | TTY   | TIME     | CMD          |
|-----|-------|----------|--------------|
| 363 | pts/0 | 00:00:00 | basht shell. |
| 443 | pts/0 | 00:00:00 | ps           |

Output contains four columns-

**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.

**\$ ps -u** Shows username with memory & CPU usage.

**\$ ps -A or \$ ps -e**

Shows all running processes.

**\$ ps -x**

Shows processes owned by currently logged user.

**\$ ps -C <command\_name>**

Shows process by command name.

# top command

- The **top** command is used to display real-time information about processes running on a system, including CPU and memory usage.
- It allows users to monitor the processes in real-time.
- It will open an interactive command mode where the top-half screen will contain the statistics of processes and resource usage.
- The lower-half screen contains a list of the currently running processes.
- To kill a process, highlight it with up/down arrow keys and press ‘k’ key.
- To exit from the top view press ‘q’ key.

```
top - 19:39:20 up 35 min, 1 user, load average: 0.00, 0.00, 0.00
Tasks: 35 total, 1 running, 34 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.0 us, 0.0 sy, 0.0 ni, 100.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 1833.8 total, 800.3 free, 466.8 used, 566.7 buff/cache
MiB Swap: 1024.0 total, 1024.0 free, 0.0 used. 1219.6 avail Mem
```

| PID | USER | PR | NI | VIRT   | RES   | SHR   | S | %CPU | %MEM | TIME+   | COMMAND          |
|-----|------|----|----|--------|-------|-------|---|------|------|---------|------------------|
| 1   | root | 20 | 0  | 167100 | 12612 | 8320  | S | 0.0  | 0.7  | 0:01.99 | systemd          |
| 2   | root | 20 | 0  | 2448   | 1332  | 1220  | S | 0.0  | 0.1  | 0:00.07 | init-systemd(Ub) |
| 5   | root | 20 | 0  | 2484   | 212   | 196   | S | 0.0  | 0.0  | 0:00.00 | init             |
| 46  | root | 19 | -1 | 47724  | 15684 | 14668 | S | 0.0  | 0.8  | 0:00.49 | systemd-journal  |
| 73  | root | 20 | 0  | 21956  | 5768  | 4296  | S | 0.0  | 0.3  | 0:00.64 | systemd-udevd    |

# top command- Summary Area

## ➤ Uptime and Load Averages-

**First line** consists of program name, current time, length of time since last boot, total no. of users logged in and system load average over the last 1, 5 and 15 minutes.

## ➤ Tasks and CPU State-

**Second line** shows the total tasks classified as running, sleeping, stopped and zombie.

**Third line** shows CPU state percentages based on the interval since last refresh. The percentages are divided into following categories-

**us**- time running un-niced user processes

**ni**- time running niced user processes

**wa**- time waiting for I/O completion

**si**- time spent servicing software interrupts

**st**- time stolen from this virtual machine by the hypervisor

**sy**- time running kernel processes

**id**- time spent in the kernel idle handler

**hi**- time spent servicing hardware interrupts

## ➤ Memory Usage-

**Fourth line** reflects Physical memory classified as total, free, used and buffered/cached.

**Fifth line** reflects Virtual memory classified as total, free, used and available.

# top command- Process Details

Top command displays a list of processes in following different columns-

**PID:** Unique Process ID given to each process.

**User:** Username of the process owner.

**PR:** Priority given to a process while scheduling.

**NI:** ‘nice’ value of a process.

**VIRT:** Amount of virtual memory used by a process.

**RES:** Amount of physical memory used by a process.

**SHR:** Amount of memory shared with other processes.

**S:** State of a process.

    D- Uninterruptible process

    R- Running

    S- Sleeping

    T- Stopped

    Z- Zombie

**%CPU:** Percentage of CPU used by the process.

**%MEM:** Percentage of RAM used by the process.

**TIME+:** Total CPU time consumed by the process.

**Command:** Command used to activate the process.

# top command options

## ➤ \$ top -b

It starts top in Batch mode, which could be useful for sending output from top to other programs or to a file. In batch mode, top will not accept input from user and runs until the iterations limit you've set with the '-n' option or until killed by pressing ctrl + c.

### Example-

**\$ top -bn 3**      this will run top in batch mode for only three iterations.

## ➤ \$ top -d

Specifies the delay in seconds between screen updates.

### Example-

**\$ top -d 2**      this will refresh top screen every 2 seconds.

**\$ top--delay 2**      same as -d option.

## ➤ \$ top -E

Instructs top to force summary area memory to be scaled as kibibytes/mebibytes/gibibytes/tebibytes.

### Example-

**\$ top -E k**      displays the memory usage summary in kilobytes.

# nice/renice command

- **nice** and **renice** commands are used to manage the priority of a running process.
- The nice command is used to modify the priority of a process. It assigns a lower priority to a process to reduce its resource usage.
- The renice command is used to modify the priority of an already running process. It can increase or decrease the priority of a process which depends on the given value.

- To set the priority of a process-

```
$ nice -10 sleep 30 &
```

- To check the nice value of a process-

```
$ ps -el | grep sleep
```

```
 0 S 1000      803      363  0  90  10 -  802 hrtme pts/0    00:00:00 sleep
```

The eighth column value is the nice value of the process.

- To set the nice value of already running process-

```
$ renice -n 2 -p 881
```

# fg/bg/jobs command

- To start any job/command/process in the background just add the ampersand (&) symbol at the end of that command.

**\$ sleep 50 &**

It will start sleep/delay timer of 50 seconds in the background.

- To check the background running jobs/processes, use jobs command-

**\$ jobs -l**

It will show all the running and stopped jobs with their job number and PIDs.

- To resume the background job in foreground, use fg command-

**\$ fg <job\_number>**

- To stop a running foreground job, press CTRL+Z key. (complete/terminate CTRL+C)

- To resume the stopped job in background, use bg command-

**\$ bg <job\_number>**

# Terminating a Process (KILL)

- To stop a process in Linux, use the 'kill' command. kill command sends a signal to the process.
- There are different types of signals that you can send. However, the most common one is 'kill -9' which is 'SIGKILL'.

- The default signal is 15 (SIGTERM).
- You can list all the signals using:

**\$ kill -L**

- Syntax for killing a process is-

**kill pid**

```
amitabh@LAPTOP-8F40GMIU:~$ kill -L
 1) SIGHUP      2) SIGINT      3) SIGQUIT     4) SIGILL      5) SIGTRAP
 6) SIGABRT    7) SIGBUS      8) SIGFPE      9) SIGKILL    10) SIGUSR1
11) SIGSEGV    12) SIGUSR2     13) SIGPIPE     14) SIGALRM    15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP    20) SIGTSTP
21) SIGTTIN    22) SIGTTOU     23) SIGURG      24) SIGXCPU    25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO      30) SIGPWR
31) SIGSYS     34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3 62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

- If process ignore the default kill signal and doesn't stop, you can send the SIGKILL signal to terminate the process.

**\$ kill -9 pid**

# ‘trap’ Command

The **trap** command runs a certain piece of code in response to a particular signal. It monitors signals and activates the specific code when the selected signal is received.

## Example 1-

```
$ trap "echo You pressed CTRL+C" SIGINT
```

## Example 2-

```
$cat > myscript.sh  
echo "Hello Tech Guy!"  
echo "How are you?"
```

```
$ chmod 744 myscript.sh
```

```
$ trap "./myscript.sh" SIGINT
```

OR

```
$ trap "./myscript.sh" SIGTSTP
```

**trap -p** option will display the piece of code related to each signal.

**trap -p <signalname>** option will display the piece of code related to specified signal only.

# sed command

- **SED** is a Stream Editor to perform inserting, deleting, searching and replacing text in a text file without using any editor. In simple words, it is a powerful text manipulation utility.

## Options and Examples-

```
Linux is great Operating System. Linux was developed by Linus Torvalds in 1991.  
It has many different distros. Passwords can be set as blank for any user.  
Linux is easy to learn. Linux is a multiuser OS.  
This is section 1 of 4th line. fourth line is finished here.  
This is 5th and last line of this file.
```

**\$ sed 's/\bis\b/was/' about.txt**

It will replace the 1<sup>st</sup> occurrence of “is” as a whole word with “was” in each line. \b is used for word boundary.

**\$ sed 's/Linux/LINUX/' about.txt**

It will **replace first occurrence** of word “Linux” with “LINUX” in each line in the file called about.txt. Here, ‘s’ specifies the substitution operation. The “/” is the delimiter symbol.

**Note-** By default, the sed command replaces the first occurrence of the pattern/word in each line and it wouldn’t replace the second or other occurrences in the same line.

**\$ sed 's/is/was/g' about.txt**

It will **replace all the occurrence** of pattern “is” with “was” in each line in the file called about.txt. Here, “g” specifies the “global” replacement.

# sed command...contd.

## Options and Examples-

**\$ sed 's/is/was/2' about.txt**

It will replace second occurrence of word “is” with “was” in each line in the file called about.txt. Here, ‘2’ specifies the 2<sup>nd</sup> occurrence of the pattern.

**\$ sed 's/is/was/2g' about.txt**

It will replace the pattern starting from second occurrence to all the occurrences in each line.

**\$ sed '3 s/is/was/' about.txt**

It will replace the first occurrence of the matching pattern in 3rd line only.

**\$ sed '3 s/is/was/2' about.txt**

It will replace the 2<sup>nd</sup> occurrence of the matching pattern in 3rd line only.

**\$ sed '3 s/is/was/g' about.txt**

It will replace all the occurrences of the matching pattern in 3rd line only.

**\$ sed 's/is/was/p' about.txt**

It will replace the 1<sup>st</sup> occurrence of the matching pattern in each line and prints the replaced line twice on the terminal. If a line doesn’t have any matching pattern then it prints only once.

## **Changes inside File –**

Use **-i** option with **sed** command to make your changes permanent inside a file!

# sed command...contd.

## Options and Examples-

**\$ sed '1,3 s/is/was/' about.txt**

It will replace 1<sup>st</sup> occurrence of word “is” with “was” in lines 1 to 3.

**\$ sed '3,\$ s/is/was/' about.txt**

It will replace 1<sup>st</sup> occurrence of word “is” with “was” from 3rd line to last line (\$) in the file about.txt.

**\$ sed '5d' about.txt**

It will delete 5<sup>th</sup> line from the file about.txt.

**\$ sed '2d;4d' about.txt**

It will delete only 2<sup>nd</sup> and 4<sup>th</sup> line from the file about.txt

**\$ sed '2, 4d' about.txt**

It will delete lines from 2<sup>nd</sup> to 4<sup>th</sup> from the file about.txt.

**\$ sed '2, \$d' about.txt**

It will delete 2<sup>nd</sup> to last line from the file about.txt.

**\$ sed '/Linus/d' about.txt**

It will delete all the lines in which the matching pattern ‘Linus’ is found.

## -- Practice Question --

How to delete last line from file about.txt?

### Solution-

**\$ sed '\$d' about.txt**

# sed command...contd.

## Options and Examples-

**\$ sed 'G' about.txt**

It will insert a blank line after each line.

**\$ sed '2G' about.txt**

It will insert a blank line after 2nd line.

**\$ sed 'G;G' about.txt**

It will insert 2 blank lines after each line.

**\$ sed '/Linux/G' about.txt**

It will insert a blank line after each line in which matching pattern 'Linux' found.

**\$ sed '/^\$/d' about.txt**

Deletes all blank/empty lines.

**\$ sed 's/^/ /' about.txt**

It will insert 5 spaces at the beginning of every line.

**\$ sed = about.txt | sed 'N;s/\n/\t/'**

Inserts line number at the beginning of each line. \t is used for a tab between number and sentence

Prepared by- Amitabh Srivastava

# sed: Practice Question-1

Consider the below file **fruitcolor.txt**-

|            |        |
|------------|--------|
| Apple      | Red    |
| Cherry     | Red    |
| Coconut    | Brown  |
| Banana     | Yellow |
| Orange     | Orange |
| Guava      | Yellow |
| Darkcherry | Red    |
| Kiwi       | Brown  |

=====OUTPUT====→

|            |        |
|------------|--------|
| Apple      | Red    |
| Cherry     | Maroon |
| Coconut    | Brown  |
| Banana     | Yellow |
| Orange     | Orange |
| Guava      | Yellow |
| Darkcherry | Maroon |
| Kiwi       | Brown  |

Q. Write the sed command to replace all occurrences of “Red” with “Maroon” only on lines containing the pattern “Cherry” or “cherry”?

Solution-

\$ sed '/[cC]herry/s/Red/Maroon/' fruitcolor.txt

# Sed: Practice Question-2

Consider the below file **fruitcolor.txt**-

|            |        |
|------------|--------|
| Apple      | Red    |
| Cherry     | Red    |
| Coconut    | Brown  |
| Banana     | Yellow |
| Orange     | Orange |
| Guava      | Yellow |
| Darkcherry | Red    |
| Kiwi       | Brown  |

=====OUTPUT====→

|            |           |
|------------|-----------|
| Apple      | Redish    |
| Cherry     | Redish    |
| Coconut    | Brownish  |
| Banana     | Yellowish |
| Orange     | Orange    |
| Guava      | Yellowish |
| Darkcherry | Redish    |
| Kiwi       | Brownish  |

Q. Write the sed command to add “ish” at the end of each line except for the color “Orange”?

Solution-

\$ sed '/Orange/!s/\$/ish/' fruitcolor.txt

# Sed: Practice Question-3

Consider the below file **fruitcolor.txt**-

|            |        |
|------------|--------|
| Apple      | Red    |
| Cherry     | Red    |
| Coconut    | Brown  |
| Banana     | Yellow |
| Orange     | Orange |
| Guava      | Yellow |
| Darkcherry | Red    |
| Kiwi       | Brown  |

===== OUTPUT =====→

| Fruit      | Color  |
|------------|--------|
| Apple      | Red    |
| Cherry     | Red    |
| Coconut    | Brown  |
| Banana     | Yellow |
| Orange     | Orange |
| Guava      | Yellow |
| Darkcherry | Red    |
| Kiwi       | Brown  |

Q. Write the sed command to add heading in the file  
as “Fruit              Color”?

Solution-

**\$ sed -i '1i Fruit              Color' fruitcolor.txt**

The **-i** option edits the file in place and **1i** tells the sed command to insert before the first line.

# Sed: Creating Backup Copy

Consider the below file **fruitcolor.txt**-

|            |        |
|------------|--------|
| Apple      | Red    |
| Cherry     | Red    |
| Coconut    | Brown  |
| Banana     | Yellow |
| Orange     | Orange |
| Guava      | Yellow |
| Darkcherry | Red    |
| Kiwi       | Brown  |

===== OUTPUT =====→

| Fruit      | Color  |
|------------|--------|
| Apple      | Red    |
| Cherry     | Red    |
| Coconut    | Brown  |
| Banana     | Yellow |
| Orange     | Orange |
| Guava      | Yellow |
| Darkcherry | Red    |
| Kiwi       | Brown  |

Q. Write the sed command to add heading in the file as “Fruit                      Color” and also creates a backup copy as **fruitcolor.txt.bak**?

Solution-

**\$ sed -i.bak '1i Fruit              Color' fruitcolor.txt**

The **-i** option make the changes inside file and **.bak** creates a backup copy of original file **fruitcolor.txt** as **fruitcolor.txt.bak**.

## Sed: Practice Question-4

**Q.** Create a pipeline using grep and sed command to display all the lines containing the pattern “**error**” (as a whole word and ignore case sensitivity) from file **/var/log/syslog** and also insert a blank line after every matched line.

**Solution-**

```
$ grep -iw "error" /var/log/syslog | sed 'G'
```

# Sed: Practice Question-5

Q. Consider below a file “**contacts.txt**”. Now write a pipeline command to display the lines containing the mobile numbers only and also replace all mobile number digits with # symbol.

```
Alok,Delhi,992313123  
Deepak,Ludhiana,986535412  
Geeta,Kanpur,geeta.sri@rediffmail.com  
Jatin,Noida,932983491  
Krishna,Delhi,krishgupta@gmail.com
```

===OUTPUT===>

```
Alok,Delhi,#####  
Deepak,Ludhiana,#####  
Jatin,Noida,#####
```

## Solution-

```
$ grep '[0-9]' contacts.txt | sed 's/[0-9]/#/g'
```

# Sed: Practice Question-6

Q. Consider below a file “**contacts.txt**”. Now write a pipeline command to display the lines containing the mobile numbers only and also replace all mobile number digits with # symbol.

```
Alok,Delhi,992313123  
Deepak,Ludhiana,986535412  
Geeta,Kanpur,geeta.16@rediffmail.com  
Jatin,Noida,932983491  
Krishna,Delhi,krishgupta@gmail.com
```

===OUTPUT===>

```
Alok,Delhi,#####  
Deepak,Ludhiana,#####  
Jatin,Noida,#####
```

## Solution-

```
$ egrep "[0-9]{9}" contacts.txt | sed 's/[0-9]/#/g'
```

# Practice Question

Q. Consider below a file “**contacts.txt**”. Now write a pipeline command to display only the domain names.

```
Alok,Delhi,992313123  
Deepak,Ludhiana,986535412  
Geeta,Kanpur,geeta.16@rediffmail.com  
Jatin,Noida,932983491  
Krishna,Delhi,krishgupta@gmail.com
```

====OUTPUT====→

```
rediffmail.com  
gmail.com
```

## Solution-

```
$ grep '@' contacts.txt | cut -d "@" -f 2
```

# awk command

- **Awk** is a scripting language used for manipulating data and generating reports. The awk command programming language requires no compiling and allows the user to use variables, numeric functions, string functions, and logical operators.
- Awk operations include-
  - Scans a file line by line
  - Splits each input line into fields
  - Compares input line/fields to pattern
  - Performs action(s) on matched lines

## Options and Examples-

\$ cat Employees

```
1, emp001, Alok, Sales, 25000
2, emp002, Deepak, Accounts, 45000
3, emp003, Firoz, Sales, 28000
4, emp004, Ram, IT, 78000
5, emp005, Gopal, IT, 86000
```

\$ awk '{print}' Employees

```
1, emp001, Alok, Sales, 25000
2, emp002, Deepak, Accounts, 45000
3, emp003, Firoz, Sales, 28000
4, emp004, Ram, IT, 78000
5, emp005, Gopal, IT, 86000
```

# awk command...contd.

## Options and Examples-

\$ awk '/Sales/ {print}' Employees **(you can omit {print})**

```
1 emp001 Alok Sales 25000
3 emp003 Firoz Sales 28000
```

\$ awk '{print NR, \$2, \$5}' Employees

```
1 emp001 25000
2 emp002 45000
3 emp003 28000
4 emp004 78000
5 emp005 86000 (NR- Record No.)
```

\$ awk 'END { print NR }' Employees

5 (displays the line count)

\$ awk 'length(\$0) > 28' Employees

Displays the line whose length is greater than 28 chars.

\$ awk '{ if(\$5 == 45000) print}' Employees

Displays only those records where value of 5<sup>th</sup> field(Salary) is equal to 45000.

\$ awk -F "," '{print \$3, \$5}' Employees

```
Alok 25000
Deepak 45000
Firoz 28000
Ram 78000
Gopal 86000
```

\$ awk -F "," '{print \$2, \$NF}' Employees

```
emp001 25000 (NF- Last Field)
emp002 45000
emp003 28000
emp004 78000
emp005 86000
```

# Awk: Practice Question-1

Consider the below file ‘Employees’-

```
1, emp001, Alok, Sales, 25000
2, emp002, Deepak, Accounts, 45000
3, emp003, Firoz, Sales, 28000
4, emp004, Ram, IT, 78000
5, emp005, Gopal, IT, 86000
```

**Q. Write an awk statement/command to print all the employee names only whose salary is greater than 45k?**

**Solution-**

```
$ awk -F ", " '{ if($5 > 45000) print $3}' Employees
```

# Awk: Practice Question-2

Consider the below file 'Employees'-

```
1, emp001, Alok, Sales, 25000
2, emp002, Deepak, Accounts, 45000
3, emp003, Firoz, Sales, 28000
4, emp004, Ram, IT, 78000
5, emp005, Gopal, IT, 86000
```

- Explanation**
- `-F ", "` sets the field separator as a comma and space.
  - `$4 == "Sales"` checks if the department is 'Sales'.
  - `{sum += $5}` adds the salary (\$5) to the sum variable.
  - `END {print sum}` prints the final total salary after processing the lines.

**Q. calculate the total salary of all employees in the 'Sales' department?**

**Solution-**

```
$ awk -F', ' '$4 == "Sales" {sum += $5} END {print sum}' Employees
53000
```

```
$ awk -F', ' '$4 == "Sales" {sum += $5} END {print "Total salary of Sales: " sum}' Employees
Total salary of Sales: 53000
```

# Awk: Practice Question-3

Consider the below file ‘Employees’-

```
1, emp001, Alok, Sales, 25000
2, emp002, Deepak, Accounts, 45000
3, emp003, Firoz, Sales, 28000
4, emp004, Ram, IT, 78000
5, emp005, Gopal, IT, 86000
```

**Q. calculate the average salary of all employees in the ‘IT’ department?**

Solution-

```
awk -F', ' '$4 == "IT" {sum += $5; count++} END {print sum/count}' Employees
82000
```

```
$ awk -F', ' '$4 == "IT" {sum += $5; count++} END {print "Average Salary(IT): " sum/count}' Employees
Average Salary(IT): 82000
```

# Awk: Practice Question-4

Consider the below file ‘Employees’-

```
1, emp001, Alok, Sales, 25000
2, emp002, Deepak, Accounts, 45000
3, emp003, Firoz, Sales, 28000
4, emp004, Ram, IT, 78000
5, emp005, Gopal, IT, 86000
```

Q. Print the name and salary of the employee with highest salary?

Solution-

```
$ awk -F', ' 'max < $5 {max = $5; name = $3} END {print name, max}' Employees
Gopal 86000
```

```
$ awk -F', ' 'BEGIN {max = 0} max < $5 {max = $5; name = $3} END {print name, max}' Employees
Gopal 86000
```

# Awk: Practice Question-5

Consider the below file ‘Employees’-

```
1, emp001, Alok, Sales, 25000
2, emp002, Deepak, Accounts, 45000
3, emp003, Firoz, Sales, 28000
4, emp004, Ram, IT, 78000
5, emp005, Gopal, IT, 86000
```

Q. Print the name and salary of the employee with minimum salary?

Solution-

```
$ awk -F', ' 'BEGIN {min = 999999} min > $5 {min = $5; name = $3} END {print name, min}' Employees
```

Alok 25000

# Awk: Practice Question-6

Consider the below file ‘Employees’-

```
1, emp001, Alok, Sales, 25000
2, emp002, Deepak, Accounts, 45000
3, emp003, Firoz, Sales, 28000
4, emp004, Ram, IT, 78000
5, emp005, Gopal, IT, 86000
```

**Q. Print the name and salary of the employees belongs to Sales and IT department?**

**Solution-**

```
$ awk -F', ' '$4 == "Sales" || $4 == "IT" {print $3, $5}' Employees
```

Alok 25000

Firoz 28000

Ram 78000

Gopal 86000

# Awk: Practice Question-7

Consider the below file ‘Employees’-

```
1, emp001, Alok, Sales, 25000
2, emp002, Deepak, Accounts, 45000
3, emp003, Firoz, Sales, 28000
4, emp004, Ram, IT, 78000
5, emp005, Gopal, IT, 86000
```

**Q. Print the name and salary of the employees belongs to Sales and IT department?  
Show your output in the descending order of their salary?**

**Solution-**

```
$ awk -F', ' '$4 == "Sales" || $4 == "IT" {print $3, $5}' Employees | sort -t " " -k2,2nr
```

**Output-**

Gopal 86000

Ram 78000

Firoz 28000

Alok 25000

# Shell Scripting

## First Shell Script-

```
$ cat script1.sh
```

```
#!/bin/bash
```

```
echo "Hello World!"
```

The first line **#!** Is known as **Shebang**, a special comment which indicates what interpreter to use when running this script. This comment is optional, if not used, the current shell is used as an interpreter.

```
$ sh script1.sh
```

```
Hello World!
```

```
$ ./script1.sh
```

```
-bash- ./script1.sh- Permission denied
```

```
$ chmod 744 script1.sh
```

This command sets the executable permission to the owner of the file.

```
$ ./script1.sh
```

```
Hello World!
```

# User Defined Variables

Assignment with equal-to sign and returns value with \$ sign as prefix in variable name.

**\$ x=10**

**\$ y=20**

**\$ echo \$x and \$y**

10 and 20

Spaces are not allowed while assignment.

**\$ x = 10**

*x- command not found*

Variable names are case sensitive.

**\$ rad=5**

**\$ RAD=45**

**\$ echo rad- \$rad and RAD- \$RAD**

rad- 5 and RAD- 45

To unset/delete the variable

**\$ unset rad**

To make a variable read only

**\$ readonly rad**

(read only variable cannot be unset)

Variables names can start with underscore sign.

**\$ \_var1=100**

**\$ echo \$\_var1**

100

# Built-in Shell Variables

|                   |   |   |
|-------------------|---|---|
| <b>\$PWD</b>      | = | Stores working directory name                   |
| <b>\$HOME</b>     | = | Stores home directory name with absolute path   |
| <b>\$SHELL</b>    | = | Stores the path of the shell that is being used |
| <b>\$LOGNAME</b>  | = | Current user's login name                       |
| <b>\$USER</b>     | = | Current user's login name                       |
| <b>\$HOSTNAME</b> | = | Stores host name                                |
| <b>\$PATH</b>     | = | Stores all the paths defined.                   |

# Special Shell Variables

| Variable   | Description   |
|------------|---|
| <b>\$0</b> | The filename of the current script.   |
| <b>\$n</b> | 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). |
| <b>\$#</b> | The number of arguments supplied to a script.   |
| <b>\$*</b> | All the arguments are double quoted. If a script receives two arguments, \$* is equivalent to \$1 \$2.  |
| <b>\$@</b> | All the arguments are individually double quoted. If a script receives two arguments, \$@ is equivalent to \$1 \$2.   |
| <b>\$?</b> | The exit status of the last command executed.   |

# Script to change directory

- In shell scripting, if you change the directory using the **cd** command within a script, the change will only affect the script's current process, and once the script finishes, the directory change will not persist in the user's shell session.
- To make sure the directory change persists even after the script finishes, you need to ensure that the **cd** command is executed in the **same shell session** as the user running the script.
- The trick is to **source** the script rather than running it. Sourcing the script will cause it to run in the current shell session, so any changes to the environment (such as changing directories) will persist even after the script finishes.
- To source the script, use following command-  
**\$ . scriptfilename.sh**

# The expr utility

- In shell scripting, is a command-line utility used to evaluate expressions, particularly for arithmetic operations, string operations, and logical comparisons. It allows you to perform calculations or check conditions directly within the script.

```
x=12  
y=5  
  
echo "Value of x: $x and y: $y"  
  
echo "Sum of x and y is: `expr $x + $y`"  
  
prod=`expr $x \* $y`  
echo "Product of x and y is: $prod"  
  
div=`expr $x / $y`  
echo "Division of x and y is: $div"
```

# Converting Case Letters

The ‘tr’ (translate) command is used to convert text from lower to upper case and vice versa.

## Example-1:

```
$ echo "hello how are you?" | tr '[lower:]' '[upper:]'  
HELLO HOW ARE YOU?
```

## Example-2:

```
$ echo "LINUX IS AWESOME!" | tr '[upper:]' '[lower:]'  
Linux is awesome!
```

## 'rev' command

It will reverse the characters of each line.

### Example-

```
$ echo "Hello" | rev  
Will print 'olleH'.
```

### Example-

```
$ rev <filename>
```

Will print each line of the file in the reverse order of their characters.

# Taking Input from User

The ‘read’ command is used to take the input from the user.

## Example-1: (myscript.sh)

```
echo "Enter your name: "
read name
echo "Your name is: $name"
```

## Example-2: (myscript.sh)

```
read -p "Enter your name: " name
echo "Your name is: $name"
```

## Example-3: (myscript.sh)

```
read -p "Enter name: " name
echo $name
```

### Output-

```
Enter name: Alok Kumar
Alok Kumar
```

## Example-4: (myscript.sh)

```
read -p "Enter name: " name
echo "$name"
```

### Output-

```
Enter name: Alok Kumar
Alok Kumar
```

# Comparison Operators

| Operation                | bash               |
|--------------------------|--------------------|
| equal to                 | if [ 1 -eq 2 ]     |
| not equal to             | if [ \$a -ne \$b ] |
| greater than             | if [ \$a -gt \$b ] |
| greater than or equal to | if [ 1 -ge \$b ]   |
| less than                | if [ \$a -lt 2 ]   |
| less than or equal to    | if [ \$a -le \$b ] |

← Number Comparison

String Comparison →

| operation           | bash              |
|---------------------|-------------------|
| equal to            | if [ \$a == \$b ] |
| not equal to        | if [ \$a != \$b ] |
| zero length or null | if [ -z \$a ]     |
| non zero length     | if [ -n \$a ]     |

# Conditional Statements (if...else)

## Example-

```
$ cat myscript.sh
a=10
b=20
if [ $a -eq $b ]
then
    echo "a is equal to b"
fi
if [ $a -ne $b ]
then
    echo "a is not equal to b"
fi
```

```
$ sh myscript.sh
or
$ ./myscript.sh
```

## Example-

```
$ cat myscript.sh
read -p "Enter first number: " a
read -p "Enter second number: " b
if [ $a -eq $b ]
then
    echo "a is equal to b"
fi
if [ $a -ne $b ]
then
    echo "a is not equal to b"
fi
```

### **!!! NOTE !!!**

To compare numbers, we can either use **-eq** OR use **==** operator.  
To compare strings, we have to use **==** operator only.  
To compare numbers, we can use either **-ne** OR use **!=** operator.  
To compare strings, we have to use **!=** operator only.

# Conditional Statements (if...elif)

## Example-

```
read -p "Enter 1st no. " a  
read -p "Enter 2nd no. " b  
  
if [ $a -eq $b ]  
then  
    echo "a is equal to b!"  
elif [ $a -gt $b ]  
then  
    echo "a is greater than b!"  
elif [ $a -lt $b ]  
then  
    echo "a is less than b!"  
fi
```

**!! NOTE !!**

If you want to compare strings, use == instead of -eq.

# Conditional Statements...contd.

## Example-

```
$ cat myscript.sh
```

## Solution-

```
read -p "Enter any number: " a
if [ `expr $a % 2` -eq 0 ]
then
    echo "a is EVEN!"
else
    echo "a is ODD!"
fi
```

```
$ sh myscript.sh
```

or

```
$ ./myscript.sh
```

## Example-

```
$ cat myscript.sh
```

```
read -p "Enter 1st string: " str1
read -p "Enter 2nd string: " str2
if [ $str1 == $str2 ]
then
    echo "Both Strings are equal"
else
    echo "Given strings are not equal"
fi
```

# Practice Question - 1

Q. Write a script that asks from user to enter an age. If age is less than 14 it should print “Swimming not allowed” AND if age is greater than or equals to 14, then print “Eligible for swimming!”

## Solution-

```
read -p "Enter the age: " age  
  
if [ $age -ge 14 ]  
then  
    echo "Eligible for Swimming!"  
else  
    echo "Swimming not allowed!"  
fi
```

## Practice Question – 2 (and ‘-a’)

Q. Write a script that asks from user to enter an age. If age is greater than or equals to 18 AND less than or equals to 45 it should print “Eligible to Swim!”. Else it should print “Swimming not allowed!”

### Solution-

```
read -p "Enter your age: " age
if [ "$age" -ge 18 -a "$age" -le 45 ]
then
echo "Eligible to swim!"
else
echo "Swimming not allowed!"
fi
```

# Practice Question – 3 (and ‘&&’)

Q. Write a script that asks from user to enter an age. If age is greater than or equals to 18 AND less than or equals to 45 it should print “Eligible to Swim!”. Else it should print “Swimming not allowed!”

Solution-

```
read -p "Enter the age: " age

if [[ "$age" -ge 18 && "$age" -le 45 ]]
then
    echo "Eligible to Swim!"
else
    echo "Swimming not allowed!"
fi
```

# Practice Question – 4 (or ‘-o’)

Q. Write a script that asks from user to enter any number. If number is divisible by 2 OR 5 print “You guess correctly!” else print “Better luck next time!”?

## Solution-

```
read -p "Enter any number: " num

if [ `expr $num % 2` -eq 0 -o `expr $num % 5` -eq 0 ]
then
    echo "You guess correctly."
else
    echo "Better Luck Next Time."
fi
```

# Practice Question – 5 (or ‘||’)

Q. Write a script that asks from user to enter any number. If number is divisible by 2 OR 5 print “You guess correctly!” else print “Better luck next time!”?

## Solution-

```
read -p "Try your Guess: " num

if [[ $num%2 -eq 0 || $num%5 -eq 0 ]]
then
    echo "You guess correctly."
else
    echo "Better Luck Next Time."
fi
```

# Practice Question – 6 ('&&' AND '||')

Q. Write a script that asks from user to enter any number. If number is between 10 and 50 OR if number is exactly 100 print “Condition met!” else print “Condition not met!”

Solution-

```
read -p "Enter a number: " num

if [[ $num -ge 10 && $num -le 50 ]] || [[ $num -eq 100 ]]
then
    echo "Condition met!"
else
    echo "Condition not met!"
fi
```

# Practice Question – 7

**Q. Write a script that asks from user to enter any username. If username exists it should print “<username> already exists” else it should print “<username> does not exist” ?**

**Solution-**

```
read -p "Enter the username: " username

# Check if the user already exists
user=`grep -w "$username" /etc/passwd`

if [ "$user" != '' ] OR if [ ! -z "$user" ]
then
    echo "$username already exists!"
else
    echo "$username does not exist!"
fi
```

# Practice Question – 8

**Q. Write a script that asks from user to enter any word and prints whether it is a Palindrome or not?**

**Solution-**

```
read -p "Enter any word: " word

rev_word=`echo $word | rev`
echo "Reverse is: $rev_word"
if [ "$word" == "$rev_word" ]
then
    echo "$word is a Palindrome!"
else
    echo "$word is NOT a Palindrome!"
fi
```

# Check if File Exists

## Example-

- To check whether entered filename exists or not!

```
read -p "Enter filename: " fname

if [ -s $fname ]
then
    echo "File $fname exists!"
else
    echo "File $fname does not exist!"
fi
```

# Check if Directory Exists

## Example-

- To check whether entered directory name exists or not!

```
read -p "Enter directory name: " dname  
  
if [ -d $dname ]  
then  
    echo "Directory $dname exists!"  
else  
    echo "Directory $dname does not  
exists!"  
fi
```

# Case Statement

- A **case statement** in shell script is used when a decision has to be made against multiple choices.
- It is useful when an expression has the possibility to have multiple values.

## Example-

```
read -p "Enter your favorite color: " color
echo -n "According to your color $color, you like: "
case $color in
    red | Red | RED)
        echo "Apples!"
        ;;

    blue | Blue | BLUE)
        echo "Bikes!"
        ;;

    green | Green | GREEN)
        echo "Travel!"
        ;;

*)
    echo "Nothing!!"
    ;;

esac
```

# Case Statement.....cont.

- Regular Expression Patterns can also be used to make the options of case.

## Example-

```
read -p "Enter your favorite color: " color
echo -n "According to your color, you like: "

case $color in
    [Rr][Ee][Dd])
        echo "Apples!"
        ;;

    [Bb][Ll][Uu][Ee])
        echo "Bikes!"
        ;;

    [Gg][Rr][Ee][Ee][Nn])
        echo "Travel!"
        ;;

*)
    echo "Nothing...!"
    ;;

esac
```

# Practice Question - 1

**Q.**

Write a shell script that takes a filename as an input and determines its type using a case statement. The script should handle the following cases-

1. If the file is a **directory**, print: “*This is a directory.*”
2. If the file is a **regular file**, further check:
  - a) If it is a **shell script** (ends in .sh), print: “*This is a shell script file.*”
  - b) If it is a **text file** (ends in .txt), print: “*This is a text file.*”
  - c) **Otherwise**, print: “*This is some other type of regular file.*”
3. If the file/directory does not exist, print: “File/Dir does not exist!”

# Practice Question – 1

## Solution

```
read -p "Enter file/directory name: " file
if [ -d "$file" ]
then
    file="dir"
elif [ ! -s "$file" ]
then
    echo "File/Dir does not exist!"
    exit 1
fi
```

**exit 1** is used to terminate the script execution with the exit status of 1, which typically indicates an error or unsuccessful execution.  
**exit 0** is used to terminate the script with exit status of 0 means successful execution.

```
case $file in
    dir)
        echo "This is a directory!"
        ;;
    *.sh)
        echo "This is a shell script file!"
        ;;
    *.txt)
        echo "This is a text file!"
        ;;
    *)
        echo "This is some other type of
file!"
        ;;
esac
```

# Command Line Arguments

- In your shell script, apart from user input, we can also use command line arguments-

## Syntax-

`./scriptname arg1 arg2 arg3 ...`

- $\$0, \$1, \$2, \$3, \dots, \$n$  are known as positional parameters corresponding to the `./scriptname arg1 arg2 arg3, \dots, argN` respectively.
- $\$#$  is used for total number of command line arguments except the scriptname.
- $\$^*$  means all of the arguments seen as a single word.
- $\$@$  means all of the arguments as  $\$^*$  but each parameter is a quoted string.

# Command Line Arguments- Example1

## Example-

```
$ cat myscript.sh
```

```
USAGE=" USAGE: ./scriptname <arg1> <arg2> <arg3> <arg4>"
```

```
if [ "$#" != 4 ]
```

```
then
```

```
    echo $USAGE
```

```
    exit 1
```

```
fi
```

```
echo "Total Number of Arguments: " $#
```

```
echo "List of Arguments: " $@
```

```
echo "Name of script that you are running: " $0
```

```
$ ./myscript.sh arg1 arg2 arg3 arg4
```

# Command Line Arguments- Example2

## Example-

- **\$ ./myscript.sh Apple Banana**

```
len1=${#1}           → Syntax used to calculate length of string
len2=${#2}
```

```
if [ "$len1" -eq "$len2" ]
then
    echo "Strings passed are of same length!"
else
    echo "Strings passed are not same length!"
fi
```

# Practice Code – 1

**Q.** Write a shell script that accepts two arguments:

1. A filename
2. A number (N)

The script should check if the proper no. of arguments are provided or not.

- ✓ If the arguments are more than two, it should display the proper message to give only two arguments.
- ✓ If no argument is supplied, it should assign following default values-
  - Filename- /var/log/syslog
  - Default number- 7
- ✓ If only one argument is supplied, the 2<sup>nd</sup> argument should take the default number value.
- ✓ If provided filename exists, display top N lines from that file, if it does not exist display proper message that “file does not exist!”.

# Practice Code – 1: Solution

```
if [ $# -lt 2 -a $# -gt 0 ]
then
    if [ -s $1 ]
    then
        head -n 7 $1
    else
        echo "File $1 does not exist!"
    fi
elif [ $# -eq 0 ]
then
    head -n 7 /var/log/syslog
```

```
elif [ $# -gt 2 ]
then
    echo "There must be exactly two arguments!"
    exit 1
elif [ -s $1 ]
then
    echo "Displaying first $2 lines of $filename: "
    head -n $2 $1
else
    echo "File $1 does not exist!"
fi
```

# Practice Code – 2

Q. Write a shell script that accepts exactly three arguments:

1. A number (num1)
  2. Any of the arithmetic operators: plus(+), minus(-), multiplication(\\*) or division(/).
  3. Another number (num2)
- 
- ✓ If the arguments are more than three or less than three, it should display the proper message to the user “***Usage ./scriptname <num1> <operator> <num2>***”.
  - ✓ The script should perform the calculation (***num1 operator num2***) and display the result.
  - ✓ The script should also handle the division by zero.
  - ✓ Write your script using **case statements**.

# Practice Code – 2: Solution

```
if [ $# -ne 3 ]; then
    echo "Usage: ./scriptname <num1>
<operator> <num2>"
    exit 1
fi

num1=$1
op=$2
num2=$3

case $op in
    +) echo "$num1 + $num2 =
$((num1 + num2))" ;;
    -) echo "$num1 - $num2 = $((num1
- num2))" ;;
```

```
\*) echo "$num1 * $num2 = $((num1 *
num2))" ;;
    /
    if [ "$num2" -eq 0 ]; then
        echo "Error: Division by zero is not
allowed."
    else
        echo "$num1 / $num2 = $((num1 /
num2))"
    fi
    ;;
\*) echo "Invalid operator! Use +, -, *, or /" ;;
esac
```

# declare Command

The declare command is used to define/declare variables in shell scriping.

## Example-

```
$ cat myscript.sh
```

```
declare -i x
```

```
x=10/5
```

```
echo $x
```

```
x=10*2
```

```
echo $x
```

**!!! NOTE !!!**  
After declaring the variable, there is no  
need to use 'expr' command to evaluate the  
math expressions.

# While Loop

- A **while loop** is a statement that iterates over a block of code till the condition specified is evaluated to **FALSE**.
- We can use ‘while loop’ in our program when we do not know how many times the condition is going to evaluate to **TRUE** before evaluating to false.

## Example- reading contents of a file.

```
fname=about.txt
```

```
while read line;  
do  
    echo $line  
done < "$fname"
```

## Example- returning numbers from 1 to 10.

```
i=1  
while [ $i -le 10 ]  
do  
    echo -n " $i "  
    i=`expr $i + 1`  
done
```

# While loop: Practice Code - 1

- Write a shell script to print the table of an entered number using while loop?

## Solution-

```
read -p "Enter any number: " num
i=1
while [ $i -le 10 ]
do
    product=`expr $num \* $i`
    echo "$num x $i = $product"
    i=`expr $i + 1`
done
```

!! Alternative to 'expr' !!

x=`expr 2+3` is equivalent to  
x=\$((2 + 3))

prod=`expr \$x \\* \$i` is equivalent to  
prod=\$((x \* i))

# While loop: Practice Code - 2

- Write a shell script to print the Fibonacci series up to the given number/limit N?

Solution-

```
read -p "Enter the limit for Fibonacci series: " limit
a=0
b=1

echo "Fibonacci Series up to $limit:"
echo -n "$a $b "

while [ $($((a + b))) -le $($limit)) ]
do
    sum=$((a + b))
    echo -n "$sum "
    a=$b
    b=$sum
done
echo
```

# For Loop

- The **for loop** operate on lists of items.
- It repeats a set of commands for every item in a list.

## Example-

```
for var in 1 2 3 4 5  
do  
    echo $var  
done
```

```
a=$(seq 5 2 15)  
for i in $a  
do  
    echo $i  
done
```

## Example-

```
a=$(seq 5 10)  
for i in $a  
do  
    echo $i  
done
```

*(in seq, first number is START point and second number is END point)*

*(if 3 numbers are there in seq, then second number is considered as step)*

```
for (( i=1; i <= 10; i++ ))  
do  
    echo $i  
done
```

# 'break' & 'continue'

- **break** and **continue** are used within loops to alter the flow of the loop and terminate the loop(break) or skip the current iteration(continue).

## Example (break)

```
i=1

while [ $i -le 10 ]
do
    echo "value of i: $i"
    if [ $i -eq 5 ]
    then
        break
    fi
    i=`expr $i + 1`
done
```

## Example (continue)

```
nums="1 2 3 4 5 6 7"

for num in $nums
do
    if [ $num -eq 3 ]
    then
        continue
    else
        echo "value of num: $num"
    fi
done
```

# Example Code-1 (for loop)

Q. Write a shell script that should take an integer number from user and prints it's table. If the input is not a valid integer and less than or equals to zero, please show proper messages to the user?

Solution-

```
read -p "Enter any number: " num
if [[ "$num" =~ ^[0-9]+$ ]]
then
    echo "Can not enter ZERO!"
elif [[ "$num" =~ ^[0-9]+$ ]]
then
    for (( i=1; i<=10; i++ ))
    do
        prod=`expr $num \* $i`
        echo "$num x $i = $prod"
    done
else
    echo "Not an Integer!"
fi
```

# Practice Code-1 (for loop)

Q. Write a shell script that should ask number of rows from user and print the right-angled triangle pattern as follows-

Enter rows: 4

```
*
```

```
* *
```

```
* * *
```

```
* * * *
```

## Solution-

```
read -p "Enter rows: " rows

for ((i=1; i<=rows; i++))
do
    for ((j=1; j<=i; j++))
    do
        echo -n "* "
    done
    echo
done
```

# Practice Code-2 (for loop)

Q. Write a shell script that should ask number of rows from user and print the inverted right-angled triangle pattern as follows-

Enter rows: 4

```
* * * *
* * *
* *
*
```

## Solution-

```
read -p "Enter rows: " rows

for ((i=rows; i>=1; i--))
do
    for ((j=1; j<=i; j++))
    do
        echo -n "* "
    done
    echo
done
```

# Practice Code-3 (for loop)

**Q.** write a shell script to print the right-angled triangle pattern using odd numbers? The script must ask two things from user- **number of rows** and **starting odd number**. If the entered number is not an ODD number, print proper message to the user.

```
Enter rows: 4
Starting number: 5
5
7 9
11 13 15
17 19 21 23
```

```
read -p "Enter rows: " rows
read -p "Starting number: " start

num=$start

if [ `expr $num % 2` -ne 0 ]
then
    for ((i=1; i<=rows; i++))
    do
        for ((j=1; j<=i; j++))
        do
            echo -n "$num "
            num=$((num + 2))
        done
        echo
    done
else
    echo "Please enter an ODD
number!"
    exit 1
fi
```

# Practice Code-4 (while loop)

Q. write a shell script to print the number of digits in a number entered by the user.

## Output

```
Enter a number: 3214
The number of digits in 3214 is : 4
```

```
read -p "Enter a number: " num
count=0
temp=$num

while [ $temp -ne 0 ]
do
    temp=$((temp / 10))
    ((count++))
done

if [ $count -eq 0 ]; then
    count=1
fi
echo "The number of digits in $num is: $count"
```

# Practice Code-5 (while loop)

Q. write a shell script to print the number of digits in a number entered by the user.

## Output

```
Enter a number: 3214  
The reverse of 3214 is : 4123
```

```
read -p "Enter a number: " num  
  
reverse=0  
temp=$num  
  
while [ $temp -ne 0 ]  
do  
    digit=$((temp % 10))  
    reverse=$((reverse * 10 + digit))  
    temp=$((temp / 10))  
done  
  
echo "The reverse of $num is: $reverse"
```

# Practice Code-6 (loops)

Q. write a shell script to print all the three-digits Armstrong numbers.

## Output

Armstrong numbers-

153

370

371

407

```
echo "Armstrong numbers-"
for (( i=100; i<=999; i++ ))
do
    num=$i
    sum=0
    temp=$num
    while [ $temp -gt 0 ]
    do
        digit=$((temp % 10))
        sum=$((sum + digit * digit * digit))
        temp=$((temp / 10))
    done
    if [ $sum -eq $num ]; then
        echo "$num"
    fi
done
```

# cmp/diff command

## cmp command

- **cmp** command is used to compare the two files byte by byte and helps to find out whether the two files are identical or not.
- It reports the location of the first mismatch to the screen if difference is found.
- If the files compared are identical, it displays no message and simply returns the prompt.

### Example-

```
$ cmp file1 file2
```

file1 file2 differ- byte 38, line 1

## diff command

- **diff** stands for difference.
- It is used to display the differences in the files by comparing the files line by line.
- It also tells which lines in one file is to be changed to make the two files identical.

### Example-

```
$ cat names
```

Deepak

amit

bhaskar

```
$ cat names2
```

Deepak

amit

mahesh

```
$ diff names names2
```

3c3

< bhaskar

---

> mahesh

# Hard Links

- Every file on the Linux filesystem, by default, having one single hard link.
- The link is between the filename and the actual data stored on the filesystem.
- Using “**ls -l**” command you will observe the number of links in the 2<sup>nd</sup> column of

```
-rw-r--r-- 1 amitabh amitabh 305 Feb 25 19:56 about.txt
```

- To create a new hard link of an existing file, use “**In**” command-

**\$ In about.txt newabout.txt**

- When changes are made to any one of them, the other reflects those changes.
- The permissions, link count, ownership, timestamps and file content are exactly same in all of the links and the original file.
- Files having hard-links together share same i-node number (*use ls -li to see the inode*)
- If original file is deleted, the data still exists under the hard link. Data will only remove when all the links will be deleted including original file.

# Soft Links

- Commonly referred to as “**Symbolic Links**”.
- Soft link is a special file points to an existing file.
- To create a soft link, use the following command-

```
$ ln -s file1.txt file1_link.txt
```

```
lrwxrwxrwx 1 amitabh amitabh 9 Mar 25 10:32 file1_link.txt -> file1.txt
```

- Observe that the letter “I” will be added to the permissions column.
- To check the soft link, use **ls -l** command-
- If the original file is deleted, the soft link is broken. This situation is known as a ***dangling soft link***.

# exec Command

- The **exec** command replaces the current shell process with the specified command. Unlike other commands that spawn a new process, exec does not create a new process — it replaces the current one.

- Syntax-

exec [COMMAND] [ARGUMENTS...]

- Example-

\$ exec top

The above command replaces the current shell with top command. You'll lose the shell session— pressing ‘q’ key won’t return you to the shell because it was replaced. You’ll have to open a new terminal.

# exec: in shell script

## Example1.sh-

```
#!/bin/bash
echo "Before banner"
exec banner "Hello"
echo "After banner" # this line will never be executed
```

## Output-

```
Before banner
#      #
#      # ##### #   #      ######
#      # #     #   #      #   #
##### #   ##### #   #      #   #
#      # #     #   #      #   #
#      # #     #   #      #   #
#      # #     #   #      #   #
#      # ##### #   ##### #   ######
```

“After banner” is never printed because banner replaces the shell running the script.

# ifconfig command (obsolete)

- **ifconfig**(interface configuration) command is used to configure the network interfaces.
- It is used at the boot time to set up the interfaces as necessary.
- This command is used to assign the IP address and netmask to the network interface or to enable or disable a network interface.

## Options and Examples-

**\$ ifconfig -a**

Display the information of all the interfaces available.

**\$ ifconfig -s**

Display the short information of all the interfaces available.

**\$ ifconfig <interface> down**

Deactivates/disables the driver for the given interface.

**\$ ifconfig <interface> up**

Activates/enables the driver for the given interface.

**\$ ifconfig <interface> <ipaddress> netmask <subnetmask>**

Set the specified ip-address and netmask to the given interface.

**\$ ifconfig --help**

Displays help of ifconfig command.

# IP Addresses & Subnet Mask

- While configuring the TCP/IP protocol on a computer, the following three TCP/IP configuration settings require:
  - An IP address
  - A subnet mask
  - A default gateway
- The subnet mask is used by the TCP/IP protocol to determine whether a host is on the local/same network or on a remote network.
- A router that is specified on a host, which links the host's subnet to other networks, is called a default gateway.
- To send packets on remote host, it is the responsibility of the router to forward the packets to the correct subnet.

# IP Addresses: Networks & Hosts

- An IP address is a **32-bit number**.
- It uniquely identifies any computer or device (such as a printer or router) on a network.
- IP addresses are normally expressed in **dotted-decimal format**, with four numbers separated by periods, such as 192.168.123.132.
- IP address (in dotted-decimal format)- 192.168.123.132
- The binary notation (32-bit number)- 11000000.10101000.01111011.10000100
- These 8-bit sections are known as **octets**.
- This number when converted from binary to decimal format becomes- 192.168.123.132.
- In every IP address, the first part of an IP address is used as a **network address**, the last part as a **host address**.
- In 192.168.123.132 the two parts are 192.168.123 known as **Network** and 132 is **Host**.
- OR 192.168.123.0 is **network address** and 0.0.0.132 is **host address**.

# ping command

- **PING (Packet Internet Groper)** command is used to check the network connectivity between two hosts in a network.
- It takes IP address or the URL as input and sends a data packet to the specified address with the message “PING” and get a response from the server/host.
- The time is recorded which is called ‘*latency*’. Low latency(less time) 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**.

## Options and Examples-

**\$ ping www.google.com**

Sends packets with echo message to the specified host URL.

**\$ ping -c 2 www.google.com**

Sends 2 packets with echo message to the specified host URL.

**\$ ping -s 40 www.google.com**

Sends packets of 40 bytes (+8 bytes) with echo message to the specified host URL.

**\$ ping -i 2 www.google.com**

Sends each packet with the interval of 2 seconds. By default, ping waits for 1 sec. to send next packet.

**\$ ping -q (press ctrl+c to see)**

Display only the summary information.

Total ICMP Packet Size is-

64 (ICMP) + 20 (Header) = 84 bytes.  
Data-payload is of 56 bytes only.

**\$ ping-V**

Display the ping version information.

# hostname command

- *hostname* command in Linux is used to obtain the DNS (Domain Name System) name and set the system's hostname.
- Its main purpose is to uniquely identify over a network.

## Options-

**\$ hostname**

Displays the system's hostname.

**\$ hostname -f or --fqdn**

Displays the system's Fully Qualified Domain Name (FQDN).

**\$ hostname -F myhost.txt                    (need reboot)**

Set the hostname as specified under the given filename. (must be 'root' or 'sudo' user).

**\$ hostname -h**

Displays the help of hostname command.

# Setting ‘alias’ / ‘fqdn’ hostname

- To set the alias hostname, open **/etc/hosts** file and edit it in following format-

| IP_ADDRESS | HOSTNAME_ALIAS | HOSTNAME        |
|------------|----------------|-----------------|
| 127.0.0.1  | MY-LAPTOP      | LAPTOP-8F40GMIU |

*to check if the alias name is set properly, test it with ping command.*

*\$ ping MY-LAPTOP*

- To set the Fully Qualified Domain Name (FQDN) hostname, open **/etc/hosts** file and edit it in following format-

| IP_ADDRESS | FQDN                  | HOSTNAME        |
|------------|-----------------------|-----------------|
| 127.0.0.1  | MY-LAPTOP.example.com | LAPTOP-8F40GMIU |

*to check if the FQDN name is set properly, test it with ping command.*

*\$ ping MY-LAPTOP.example.com*

# ip command

- **ip** command is used for network configuration and management.
- Allows user to interact with various networking components like network interfaces, routing tables, addresses, etc.
- Is a part of **iproute2** package.
- Replacement of older commands- *ifconfig* and *route*.

## Options and Examples-

**\$ ip addr show      (or, in short, you can type command- ip a)**

Displays details such as interface names (lo, eth0, wlan0, etc.) MAC addresses, IPV4 and IPV6 addresses, subnet mask, etc.

**\$ ip addr show dev eth0**

Displays details only the specified device interface eth0.

**\$ sudo ip link set lo down**

Disables the network interface “**lo**”. After this you cannot connect/ping to your loopback/localhost address.

**\$ sudo ip link set lo up**

Enables the network interface “**lo**”. After this you cannot connect/ping to your loopback/localhost address.

# ip command....cont.

## Options and Examples-

**\$ sudo ip addr add 192.168.0.123/24 dev eth0**

Set the new IP-address as specified to the interface eth0. After this command you have to delete the previously assigned IP-address from eth0 interface, only then the new IP-address will be set.

**\$ sudo ip addr del 172.30.16.188/20 dev eth0**

Deletes the assigned IP-address from the interface eth0.

**\$ ip route show**

Shows all the network routes defined under routing table.

**\$ sudo ip route add 172.30.25.0/24 via 172.30.16.1 dev eth0**

Adds another route to the routing table for interface eth0.

**\$ sudo ip route del 172.30.25.0/24 via 172.30.16.1 dev eth0**

Deletes the specified route from the routing table for interface eth0.

# netstat command

- The **netstat** command is a tool for monitoring network connections.
- It displays active connections, their status, and related statistics.
- It shows which ports are open, which services are listening on them, and which IP addresses are connected to your system.

## netstat command options-

- a** : Displays all active TCP & UDP connections and listening ports.
- t** : Displays TCP (Transmission Control Protocol) connections.
- u** : Displays UDP (User Datagram Protocol) connections.
- l** : Displays only listening ports.
- n** : Displays numerical addresses instead of resolving names.
- r** : Displays the routing table information.

# traceroute command

- **traceroute** command prints the route that a packet takes to reach the host.
- This command is useful to know about the route and about all the hops (in between nodes) that a packet takes.
- It sends three packets to each hop.

## Options and Examples-

**\$ traceroute google.com**

Displays the route information in different columns.

- First column is the hop count.
- Second column is the ip-address of that hop.
- Then the three times for three packets, respectively.

**\$ traceroute -f 10 google.com**

Instead of first hop, it starts tracing route from the 10<sup>th</sup> hop.

**\$ traceroute -n google.com**

Do not resolve the IP-addresses to their domain names.

**\$ traceroute google.com 100**

Change the default packet length from 60 bytes to 100 bytes.

# netplan command

- **netplan** command is used to configure a static IP address or a DHCP IP address.
- Configuration file is located under **/etc/netplan** directory with **.yaml** extension.

## Steps to configure IP address-

Open the configuration file situated under **/etc/netplan** with extension **.yaml** with root login or sudo privilege. The layout of the file looks like this-

```
network:  
  version: 2  
  ethernets:  
    DEVICE-NAME:  
      renderer: NetworkManager  
      match:  
        name: INTERFACE-NAME  
      dhcp4: YES/NO  
      addresses: [IP/NETMASK]  
      gateway4: GATEWAY-ADDRESS  
    nameservers:  
      addresses: [NAMESERVER,NAMESERVER]
```

- Edit the above file to fit your networking needs. Save and close the file.

# Applying netplan settings

- Configure and save the `.yaml` file.

```
network:
```

```
  version: 2
```

```
  ethernets:
```

```
    NM-2f3e1107-becd-43d2-852a-eece2133d91a:
```

```
      renderer: NetworkManager
```

```
      match:
```

```
        name: "eth0"
```

```
        dhcp4: no
```

```
        addresses: [172.30.16.188/20]
```

```
        gateway4: 172.30.16.1
```

```
      nameservers:
```

```
        addresses: [8.8.8.8,8.8.4.4]
```

If you are certain about your configuration file, you can skip the “try” option and directly go to applying the new configuration by giving the following command-

```
$ sudo netplan apply
```

- Before applying the changes, first test the configuration with the following command-

```
$ sudo netplan try
```

The above command will validate the configuration before applying it. If it succeeds, the configuration accepted. If it fails, netplan will automatically revert to the previous configuration.

# nmcli command

- **nmcli** is a command-line tool used to-
  - View network status
  - Enable/disable network interfaces
  - Connect to Wifi
  - Configure static IP address
  - Manage Ethernet and VPN connections
- **Basic Syntax-**

nmcli [OPTIONS] OBJECT { COMMAND | help }

Object: e.g. device, connection, network

Command: the action you want to perform

# nmcli command: Examples

- **Example-1:** to show all network interfaces/connections, their type and state.

```
$ nmcli device show
```

- **Example-2:** to show all available Wi-fi networks.

```
$ nmcli device wifi list
```

- **Example-3:** to connect to a Wi-fi network.

```
$ nmcli device wifi connect <"Network_Name"> password <"your_password">
```

- **Example-4:** to show all active connections.

```
$ nmcli connection show --active
```

- **Example-5:** bring an interface/connection UP or DOWN.

```
$ nmcli connection up static-eth0
```

```
$ nmcli connection down static-eth0
```

# nmcli command: Examples

- **Example-6:** to create new static IP Ethernet connection.

```
$ nmcli connection add type ethernet ifname eth0 con-name static-eth0  
ip4 192.168.1.100/24 gw4 192.168.1.1
```

- **Example-7:** to turn ON/OFF networking or Wi-fi.

```
$ nmcli networking off
```

```
$ nmcli networking on
```

```
$ nmcli radio wifi off
```

```
$ nmcli radio wifi on
```

# Installing GCC Compiler

- GCC stands for GNU Compiler Collections.
- It is used to compile programs written in C and C++ language.
- It is an Open Source collection of compilers and libraries.
- To install GCC compiler follow the below steps-

**\$ apt-get update**

It will update the packages by downloading the package's information.

**\$ sudo apt install build-essential**

It will install build-essential packages that contains the GCC compiler and all other essentials used to compile the program written in C or C++.

**\$ gcc--version**

It will show the version and copyright information of GCC compiler.

**\$ gcc prog1.c**

It will compile the C program written inside prog1.c file. After the successful compilation, it will create an executable file called **a.out**.

**\$ ./a.out**

This will run the code just compiled in above step.

# Extracting Substring

- Extracting a substring from a string is a basic and common operation of text processing in Linux.
- Using Bash's Substring Expansion we can easily extract a substring from a string.

- **Example:1-**

```
$ str="LinuxShell"
```

```
$ echo ${str:5}      Output- Shell
```

- **Example:2-**

```
$ str="LinuxShell"
```

```
$ echo ${var:5:3}    Output- She
```

# Arrays in Shell Script

- Arrays in Shell Scripting allow us to store multiple values in a **single variable**.
- Bash supports two types of arrays-
  - **Indexed arrays**
  - **Associative arrays**
- Elements of an Indexed array are accessed using **numeric indices**, starting from **0**.
- Associative arrays use **string keys** instead of numeric indices. They require explicit declaration using **declare -A** command.

# Declaring Arrays

- The **declare** keyword can be used to explicitly declare arrays.
- The options –a and –A are used to define the type of array.

## Example-

```
$ declare -A assoc_array  
$ assoc_array[key]=value
```

```
$ declare -a indexed_array  
$ indexed_array[0]=value
```

- Option uppercase **A** is used to declare an associative array while lowercase **a** is used to declare an indexed array.

# Indexed Arrays: Initializing & Adding Elements

- Initializing an empty array-

numbers=()

strings=()

names=()

- Initializing an array with values-

numbers=(12 34 7)

strings=("abc" "java" 'linux')

names=("Alok" 'Gourav' "Gauri")

- Adding/Appending elements to an array-

numbers+=(1)

strings+=("hello")

names+=("James")

# Displaying Array & Array Length

## Example-

```
declare -a names
names=("Alok" 'Gourav' "Gauri")
echo "names: ${names[@]}"
echo "value at index 1: ${names[1]}"
echo "length of names array: ${#names[@]}"
echo -----
echo "After deleting 1 element at index 1-"
unset names[1]
echo "names: ${names[@]}"
echo -----
echo "Appending an element-"
names+=("Naresh")
echo "names: ${names[*]}"
```

## Output-

```
names: Alok Gourav Gauri
value at index 1: Gourav
length of names array: 3
-----
After deleting 1 element at index 1-
names: Alok Gauri
-----
Appending an element-
names: Alok Gauri Naresh
```

# Practice Code - 1

**Q-** Write a shell script to copy all the odd numbers from an existing array to a new array called “odd\_array”?

**Solution-**

```
original_array=(1 2 3 4 5 6 7 8 9 10)

odd_array=()

for num in "${original_array[@]}"
do
    if [[ $((num % 2)) != 0 ]]
    then
        odd_array+=("$num")
    fi
done

echo "Odd numbers: ${odd_array[@]}"
```

# Practice Code - 2

**Q-** Write a shell script that takes numbers from an array and create two new arrays- one having odd numbers and another for even numbers?

**Solution-**

```
# Original array
```

```
numbers=(12 7 3 8 10 5 19 6)
```

```
even_array=()
```

```
odd_array=()
```

```
for num in "${numbers[@]}"; do
```

```
    if (( num % 2 == 0 )); then
```

```
        even_array+=("$num")
```

```
else
```

```
    odd_array+=("$num")
```

```
fi
```

```
done
```

```
# Output the results
```

```
echo "Original array:
```

```
 ${numbers[@]}
```

```
echo "Even numbers:
```

```
 ${even_array[@]}
```

```
echo "Odd numbers:
```

```
 ${odd_array[@]}
```

# Practice Code - 3

**Q-** Write a shell script that take numbers from an array having even indexes and print the multiplication table of those numbers?

**Solution-**

```
numbers=(2 3 5 7 9 11)

for ((i=0; i<${#numbers[@]}; i+=2))
do
    num=${numbers[$i]}
    echo "Table of $num:"
    for ((j=1; j<=10; j++))
    do
        echo "$num x $j = $((num * j))"
    done
    echo
done
```

# Practice Code - 4

**Q-** Write a shell script to store array items in a new array from existing array only once by removing the duplicate items?

**Solution-**

```
original_array=(12 34 12 3 45 12 34)
unique_array=()
echo "Original array:
${original_array[@]}"
for item in "${original_array[@]}"; do
    found=0
    for unique in "${unique_array[@]}";
do
    if [[ "$item" == "$unique" ]]; then
```

```
        found=1
        break
    fi
done

if [[ $found -eq 0 ]]; then
    unique_array+=("$item")
fi
done

echo "Unique items:
${unique_array[@]}"
```

# Associative Arrays: Initializing/Creating

- **Declaring an array-**

declare -A marks

- **Declaring a read-only array-**

declare -rA marks

- **Creating array with values-**

marks["Alok"]=90

marks["Geeta"]=87

marks["Ram"]=75

- **Initializing array with values-**

declare -A marks=([ "Alok" ]=90, [ "Geeta" ]=87, [ "Ram" ]=75)

**SYNTAX**

Array\_name["key"] = value

**Note**

Key must be unique in an array. If duplicate key is there, the last one will be considered only.

# Appending/Deleting/Displaying Array Values

- Example-

```
declare -A marks  
marks["Alok"]=90  
marks["Geeta"]=87  
marks["Ram"]=75  
  
echo "Values: ${marks[@]}"  
echo "Keys: ${!marks[@]}"  
echo "Marks of Alok: ${marks["Alok"]}"  
echo "Length of array marks: ${#marks[@]}"
```

- Output-

```
Values: 87 75 90  
Keys: Geeta Ram Alok  
Marks of Alok: 90  
Length of array marks: 3
```

- Appending new value-

```
Marks["Gopal"]=100
```

- Deleting a key-

```
unset marks["Alok"]
```

# Iterating Associative Array

- Example-

```
declare -A marks
marks["Alok"]=90
marks["Geeta"]=87
marks["Ram"]=75

for key in ${!marks[@]}
do
    echo "$key, ${marks[$key]}"
done
```

- Output-

```
Geeta, 87
Ram, 75
Alok, 90
```

# Check If Key Exists

- Example-

```
declare -A marks
marks["Alok"]=90
marks["Geeta"]=87
marks["Ram"]=75

if [[ -n "${marks["Geeta"]}" ]]
then
    echo "Key exists!"
else
    echo "Key not found!"
fi
```

- Output-

```
Key exists!
```

# Practice Code-1

**Q.** Write a shell script ask the user to input three names and phone numbers, then store them in an array and show them all.

```
declare -A phonebook

for i in {1..3}; do
    read -p "Enter name: " name
    read -p "Enter phone number: " phone
    phonebook["$name"]="$phone"
done

echo -e "\nPhone Book:"
for name in "${!phonebook[@]}"; do
    echo "$name - ${phonebook[$name]}"
done
```

## Output-

```
Enter name: Amitabh
Enter phone number: 12345
Enter name: Deepak
Enter phone number: 23456
Enter name: Hari
Enter phone number: 43215

Phone Book:
Deepak - 23456
Amitabh - 12345
Hari - 43215
```

## Practice Code-2

**Q.** Write a shell script ask the user to input a day abbreviation (e.g. Mon) and then display the full name of day (Monday).

```
declare -A days

# Define mapping
days["Mon"]="Monday"
days["Tue"]="Tuesday"
days["Wed"]="Wednesday"
days["Thu"]="Thursday"
days["Fri"]="Friday"
days["Sat"]="Saturday"
days["Sun"]="Sunday"
```

```
read -p "Enter a day abbreviation  
(e.g., Mon): " input

if [[ -n "${days[$input]}" ]]
then
    echo "Full name: ${days[$input]}"
else
    echo "Invalid day abbreviation."
fi
```

# Installing MySQL Server

- MySQL is an Open-Source database management system used to create/manage database.
- To install mysql server run the following commands-

**\$ sudo apt update**

It will update the packages by downloading the package's information.

**\$ sudo apt install mysql-server**

It will install mysql-server packages.

**\$ sudo service mysql start**

It will start the mysql-server service.

**\$ sudo mysql -u root -p**

It will ask the root password, which is 'root' by default. After giving the password, the MySQL shell is started. Here, **-u** represents the username and **-p** is for entering password.

# MySQL: Creating User Account

- Log into mySQL as root user. [ sudo mysql –u root –p ]  
*(‘root’ is the default password of root user)*
- To create a user account, run the following commands at mySQL prompt-

```
mysql> CREATE USER 'amitabh'@'localhost' IDENTIFIED BY 'amitabh';
mysql> create database testdb;
mysql> GRANT ALL PRIVILEGES ON testdb.* TO 'amitabh'@'localhost';
mysql> EXIT;
```

- To login with this new user account, run the following command at Ubuntu terminal-

```
$ sudo mysql -u amitabh -p
```

# MySQL: Creating a Database

- After login to **your** mySQL account, first we need to confirm if the database is successfully created, give the following command-

```
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| performance_schema |
| testdb |
+-----+
3 rows in set (0.01 sec)
```

# MySQL: Creating a Table

- After creating a database '**testdb**', next step would be to create a table '**students**' under '**testdb**' database. Give the following command to create a table-

```
mysql> use testdb;
```

```
mysql> create table students(
```

```
    -> roll varchar(5) not null,  
    -> name varchar(55) not null,  
    -> city varchar(30),  
    -> primary key(roll));
```

- To confirm if the table '**students**' is successfully created, give the following command-

```
mysql> describe students;
```

# Connecting Database & Insert Record

```
# MySQL credentials  
DB_USER="amitabh"  
DB_PASS="amitabh"  
DB_NAME="testdb"
```

*# Prompt user for input*

```
read -p "Enter student roll no.: " roll_no  
read -p "Enter student name: " std_name  
read -p "Enter student city: " std_city
```

*# Insert into MySQL database*

```
mysql -u "$DB_USER" -p"$DB_PASS" "$DB_NAME" << EOF  
INSERT INTO students(roll, name, city) VALUES('$roll_no', '$std_name', '$std_city');  
EOF  
echo "Record inserted successfully!"
```

```
amitabh@amitabh-Vostro-340:~/Desktop$ ./insertRecord  
Enter student roll no.: 45132  
Enter student name: Teena  
Enter student city: Bareilly  
mysql: [Warning] Using a password on the command line interface can be insecure.  
Record inserted successfully!
```

# Remove 'warning' from output (Secure Method)

```
# Prompt user for input
```

```
read -p "Enter Roll Number: " roll  
read -p "Enter Name: " name  
read -p "Enter City: " city
```

```
~/.testdb.cnf-  
[client]  
user=amitabh  
password=amitabh
```

```
# Run the MySQL insert command
```

```
mysql --defaults-file=~/.testdb.cnf -D testdb -e \  
"INSERT INTO students (roll, name, city) VALUES ('$roll', '$name', '$city');"
```

```
echo "Record inserted successfully."
```

# Edit Record through Script

**# Prompt user for input**

```
read -p "Enter Roll number: " roll_no  
echo "roll number: $roll_no"
```

**# Search record inside table**

```
record=$(mysql --defaults-file=~/testdb.cnf  
-D testdb -e \  
"SELECT * FROM students WHERE  
roll=$roll_no;")
```

```
if [ -z "$record" ]
```

```
then
```

```
    echo "No student found!!"
```

```
else
```

```
    read -p "Enter new name: " name  
    read -p "Enter new city: " city
```

**# Update the record**

```
    mysql --defaults-file=~/testdb.cnf -D  
testdb -e \  
"UPDATE students SET name='$name',  
city='$city' WHERE roll=$roll_no;"
```

```
    echo "Record updated successfully."
```

```
fi
```

# Delete Record through Script

**# Prompt user for input**

```
read -p "Enter Roll number: " roll_no  
echo "roll number: $roll_no"
```

**# Search record inside table**

```
record=$(mysql --defaults-file=~/testdb.cnf  
-D testdb -e \  
"SELECT * FROM students WHERE  
roll=$roll_no;")
```

```
if [ -z "$record" ]
```

```
then
```

```
    echo "No student found!!"
```

```
else
```

**# Delete the record**

```
mysql --defaults-file=~/testdb.cnf -D  
testdb -e \  
"DELETE FROM students where  
roll=$roll_no;"
```

```
echo "Record deleted successfully."
```

```
fi
```

# Shell Scripting: Functions

- Functions enables to break down the overall functionality of a script/code into smaller, logical subsections, which can then be called upon to perform their individual tasks when needed.
- Using functions to perform repetitive tasks is an excellent way to create *code reuse*.
- Shell functions are similar to subroutines, procedures, and functions in other programming languages.
- To declare a function, simply follow the below syntax-

```
function_name()  
{  
    list_of_commands  
}
```

# Function: Example

- Example-

*# Function definition*

```
Hello() {  
    echo "Hello World"  
}
```

*# Calling function*

```
Hello
```

Output-

**Hello World**

# Passing Parameters

- Function parameters would be represented by \$1, \$2 and so on.
- Example-

```
greet() {  
    echo "Welcome $1 $2"  
}
```

*# Calling function with parameters*  
greet "Alok" "Kumar"

- Output-
- Welcome Alok Kumar

# Returning a Value from a Function

- Use ‘**return**’ statement to return a value from the function.
- Use ‘**\$?**’ to print the last returned value (*use with return code 0 or 1*).

## Example-1

```
# Function definition
add() {
    echo -n "Sum of $1 and $2 is: "
    return `expr $1 + $2`
}

# Calling function with parameters
add 10 20
echo $?
```

## Example-2

```
# Function definition
add() {
    echo -n "Sum of $1 and $2 is: "
    sum=`expr $1 + $2`
    return $sum
}

# Calling function with parameters
add 20 30
echo $sum
```

# Using Return Codes 0 and 1

Example-

```
is_even_odd() {  
if (( $1 % 2 == 0 ))  
then  
    return 0  
else  
    return 1  
fi  
}
```

```
is_even_odd 43  
  
if [[ $? -eq 0 ]]  
then  
    echo "It's even."  
else  
    echo "It's odd."  
fi
```

# Practice Code-1

- Create a function that will ask the filename from user and checks whether it exists or not?

## Solution-

```
file_exists() {  
    if [[ -f "$1" ]]; then  
        echo "File '$1' exists."  
    else  
        echo "File '$1' does not exist."  
    fi  
}  
  
read -p "Enter filename: " fname  
file_exists "$fname"
```

# Practice Code-2

- Create a function that will take all the .txt files from current directory and print the line numbers of each file?

## Solution-

```
process_file() {  
    echo "Processing: $1"  
  
    # Example: Count lines  
    wc -l "$1"  
}  
  
for file in *.txt  
do  
    process_file "$file"  
done
```

# Practice Code-3

- Create functions to perform Addition, Subtraction, Multiplication and Division. Use Case-statement to call functions. Your program should be a menu driven program.

```
add() {  
    echo -n "Enter two numbers: "  
    read a b  
    result=$((a + b))  
    echo "Result: $a + $b = $result" }
```

```
subtract() {  
    echo -n "Enter two numbers: "  
    read a b  
    result=$((a - b))  
    echo "Result: $a - $b = $result" }
```

```
multiply() {  
    echo -n "Enter two numbers: "  
    read a b  
    result=$((a * b))  
    echo "Result: $a * $b = $result" }
```

```
divide() {  
    echo -n "Enter two numbers: "  
    read a b  
    if [ "$b" -eq 0 ]; then  
        echo "Error: Division by zero!"  
    else  
        result=$(echo "scale=2; $a / $b" | bc)  
        echo "Result: $a / $b = $result"  
    fi }  
while true  
do  
    echo ""  
    echo "===== Calculator Menu ====="
```

```
echo "1. Addition"  
echo "2. Subtraction"  
echo "3. Multiplication"  
echo "4. Division"  
echo "5. Exit"  
echo -n "Choose an option [1-5]: "  
read choice  
case $choice in  
    1) add ;;  
    2) subtract ;;  
    3) multiply ;;  
    4) divide ;;  
    5) echo "Goodbye!"; exit 0 ;;  
    *) echo "Invalid choice!" ;;  
esac  
done
```

# Function Call from Prompt

- Put the definitions for commonly used functions inside **.profile** file. These function definitions will be available whenever user log in and he can use them at the command prompt.
- OR, put the function definitions in any **.sh** script and source the script. When we source the file, no output will be displayed but we can now use the functions defined under the **.sh** file from the command prompt.
- **func2.sh-**

```
sum()  
{  
    echo "`expr $1 + $2`"  
}
```

```
$ . func2.sh  
$ sum 4 5  
9
```

**To remove a function from current shell-**

```
$ unset -f sum
```

# Nested Functions

- A function can call another function.

## Example-

```
# Calling one function from another
number_one () {
    echo "This is number_one function!"
    number_two
}

number_two () {
    echo "This is number_two function!"
}

# Calling number_one
number_one
```

# Example: Recursive Function

- When a function call itself is known as **Recursive** function.

Example-

```
count=0
number_one () {
    count=$((count + 1))
    if [ $count -le 5 ]
        then
            echo "Count: $count"
            number_one
        fi
    }
}

# Calling number_one
number_one
```

# 'local' Keyword

- The **local** command/keyword can only be used within a function.
- It makes the variable name have a **visible scope restricted to that function** and its children only.

Example-

```
name="Alok"

show_name(){
    local name=$1
    echo "Inside show_name(): name is set to $name"
}

echo "Before calling show_name name is set to $name"

show_name "Ram"

echo "After calling show_name name is set to $name"
```

Remove '**local**' from this code  
and check the output!

# Practice Code-4

- Write a script where the outer function accepts two numbers and calls a nested function to calculate the square of each number and return their sum.

Example-

```
sum_of_squares() {  
    local a=$1  
    local b=$2  
    square() {  
        echo $(( $1 * $1 ))  
    }  
  
    local square1=$(square $a)  
    local square2=$(square $b)  
    echo "Sum of squares: $((square1 + square2))"  
}  
  
sum_of_squares 2 3
```

# Practice Code-5

- Write a script that takes a string from user using outer function and uses a nested function to reverse it.

- **Solution-**

```
string_input() {  
    read -p "Enter String: " input  
  
    reverse() {  
        echo "$1" | rev  
    }  
  
    echo "Reversed string: $(reverse $input)"  
}  
  
string_input
```

# fdisk command

- Short for '**Format Disk**', a dialog driven utility used for creating and manipulating the disk partition table.
- It allows to create maximum of four primary partitions and the number of logical partitions depends on the size of the hard disk.
- The partition naming scheme follows the **/dev/xxN** form:
  - **/dev** is the directory where all device files lives.
  - **xx** indicates the device type (which contains the partition).
  - **y** indicates which device the partition is on.
  - **N** is the partition number.
- **Example-** /dev/sdb, /dev/sdc1

# fdisk options

- **fdisk -l**  
To view all partitions of specific hard-disk.
- **fdisk -l /dev/sda**  
will display all disk partitions of device **/dev/sda**.

To view all commands which are available for fdisk, use the below command by mentioning the hard disk name-

```
amitabh@LAPTOP-8F40GMIU:~$ sudo fdisk /dev/sdc
```

```
Welcome to fdisk (util-linux 2.39.3).
```

```
WARNING: DOS-compatible mode is deprecated. It's strongly recommended to  
switch off the mode (command 'c') and change display units to  
sectors (command 'u').
```

```
Command (m for help):
```

# fdisk options

- Once you press the ‘m’ key, the following command options will be displayed on your screen-

## Misc

```
m  print this menu  
u  change display/entry units  
x  extra functionality (experts only)
```

## Script

```
I  load disk layout from sfdisk script file  
O  dump disk layout to sfdisk script file
```

## Save & Exit

```
w  write table to disk and exit  
q  quit without saving changes
```

## Create a new label

```
g  create a new empty GPT partition table  
G  create a new empty SGI (IRIX) partition table  
o  create a new empty MBR (DOS) partition table  
s  create a new empty Sun partition table
```

## Help:

### DOS (MBR)

```
a  toggle a bootable flag  
b  edit nested BSD disklabel  
c  toggle the dos compatibility flag
```

### Generic

```
d  delete a partition  
F  list free unpartitioned space  
l  list known partition types  
n  add a new partition  
p  print the partition table  
t  change a partition type  
v  verify the partition table  
i  print information about a partition
```

# Logical Volume Manager (LVM)

- **Logical Volume Manager** is a system used in Linux to manage disk storage more flexibly than traditional partitioning.
- it can sort raw storage (hard disks) into logical volumes, making it easy to configure and use.

## Why Use LVM?

- Traditional partitioning is rigid and hard to modify
- LVM allows dynamic resizing of volumes
- Allows one file system to span across multiple physical disks
- Useful for servers, VMs, and large-scale systems

# LVM Architecture- 3 Core Components

- **Physical Volume (PV)-**

Think of a **PV** as the raw hard disk or a disk partition that LVM will manage. *Example- /dev/sda1, /dev/sdb*

- **Volume Group (VG)-**

A **VG** is a pool of storage created by combining one or more PVs. It is like merging multiple water tanks into one big tank.

- **Logical Volume (LV)-**

These are slices of the VG that act like partitions. You can format LVs and mount them like regular partitions. Think of LVs as **drawers in a big cabinet (VG)**.

# How to use LVM

- **Install LVM tools-**

```
$ sudo apt install lvm2
```

- **Create Physical Volumes-**

```
$ sudo pvcreate /dev/sdb /dev/sdc
```

- **Create Volume Groups-**

```
$ sudo vgcreate my_vg /dev/sdb /dev/sdc
```

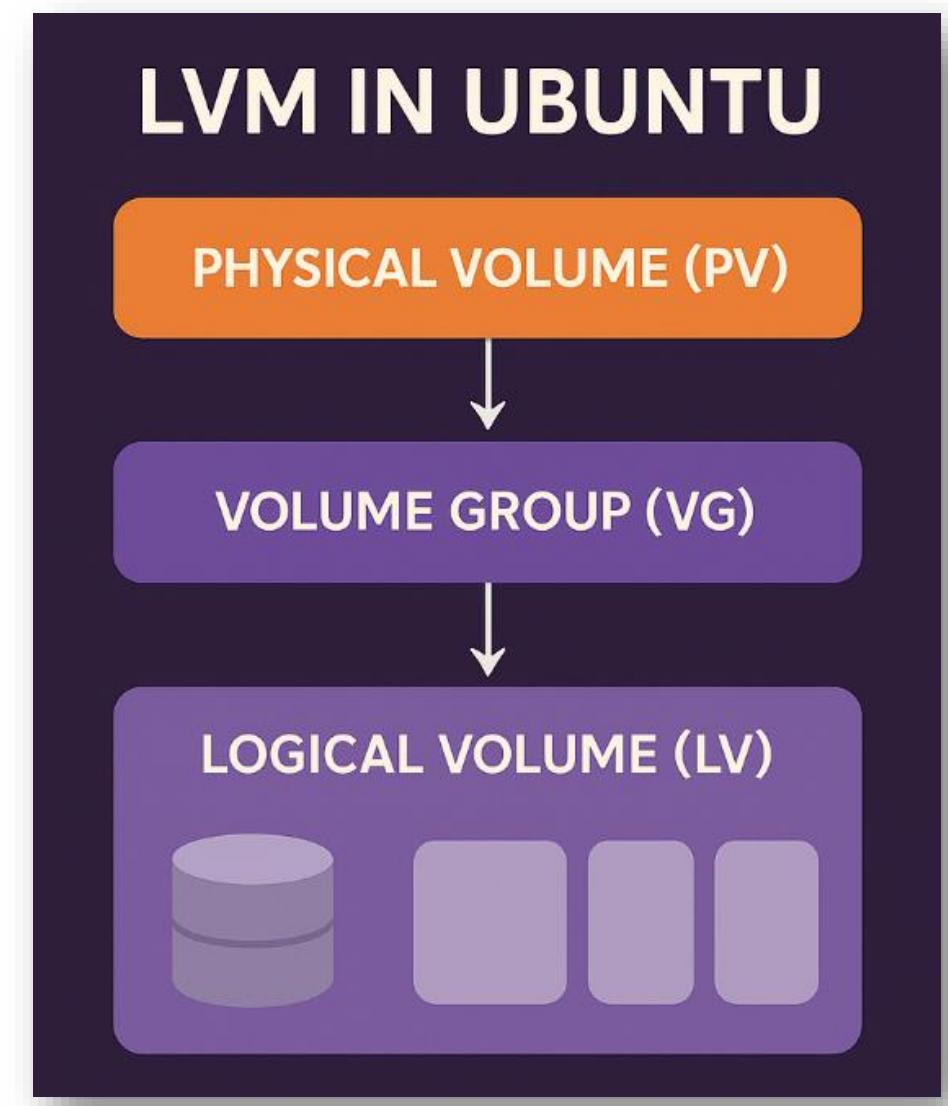
- **Create Logical Volumes-**

```
$ sudo lvcreate -L 10G -n my_lv my_vg
```

- **Format and Mount the Logical Volume-**

```
$ sudo mkfs.ext4 /dev/my_vg/my_lv
```

```
$ sudo mount /dev/my_vg/my_lv /mnt
```



# Resizing LVM

- **Increase Size-**

```
$ sudo lvextend -L +5G /dev/my_vg/my_lv  
$ sudo resize2fs /dev/my_vg/my_lv
```

- **Reduce Size- *(more caution needed)***

```
$ sudo umount /mnt  
$ sudo e2fsck -f /dev/my_vg/my_lv  
$ sudo resize2fs /dev/my_vg/my_lv 8G  
$ sudo lvreduce -L 8G /dev/my_vg/my_lv
```

# tar command

- The Linux **tar** stands for **Tape ARchive**, which is used to create Archive and extract the Archive files.
- It provides archiving functionality in Linux. We can use it to create compressed or uncompressed archive files.

## Options and Examples-

**\$ tar -cvf dir1.tar dir1**

It will create a compressed archive file called dir1.tar of the directory dir1.

Here, the option used are-

**-c** = create archive

**-v** = verbose mode, means to print all the file names on terminal those are added to archive file

**-f** = used to define the archive filename just after the **-f** option

**\$ tar -tvf dir1.tar**

It will display all the contents(files) stored inside compressed archive file called dir1.tar.

Here, the option used is-

**-t** = to display all the files stored inside the archive file

**\$ tar -xvf dir1.tar**

It will extract all the contents(files) stored inside compressed archive file called dir1.tar.

Here, the option used is-

**-x** = used to extract all files from compressed archive

# gzip/gunzip command

- **gzip** command is used to compress files.
- Each single file is compressed into a single file.
- It is a very popular methods of compressing files, in order to save space, or to reduce the amount of time needed to transmit the files across the network, or internet.

## Options and Examples-

**\$ gzip about.txt**

It will compress the original file **about.txt** as **about.txt.gz**.

**\$ gzip -k about.txt**

It will compress the file **about.txt** as **about.txt.gz** and also keeps the original file.

**\$ gzip -9 about.txt**

It will compress the file **about.txt** with highest compression level. Numbers 1 to 9 can be used from lowest to highest compression levels.

**\$ gzip -L**

It will display the gzip license information.

**\$ gunzip about.txt.gz**

It will uncompress the compressed file **about.txt.gz** as **about.txt**.

# Mounting USB Drive

- The files and directories under Linux filesystem are arranged in a root directory called `/`.
- **mount** command is used to attach the filesystem of an external device to the filesystem of an Operating System.
- It mounts (attaches) the external storage devices like hard disks, USB drives, etc. to the OS filesystem.

## Steps-

➤ Open Powershell and type the following command-

**notepad %UserProfile%\wslconfig**

➤ If the file exist edit it and if does not exist, create a new file as follows-

**[automount]  
enabled = true  
options = "metadata"  
mountFsTab = true**

➤ Restart WSL by giving following command-

**wsl –shutdown**

And reopen Ubuntu.

After performing all these steps all windows drives will mount correctly inside WSL Ubuntu OS.

To access your Windows drives under Ubuntu, give following command-

**\$ cd /mnt/  
\$ ls**

It will show the directories as '`c`', '`d`', '`e`', etc. under Ubuntu for Windows drives.

# Mount Specific Folder

- To mount a specific folder, we need to configure a file called **/etc/fstab**.
- **/etc/fstab** is a special file that tells Linux what to mount at startup.
- Open **/etc/fstab** file and add following line at the end of this file-  
**/mnt/e/my-folder /home/yourhomedirectory/new-folder none bind 0 0**
- After adding the above line, save the file and run the following command-  
**\$ sudo mount -a**  
If no error comes, your folder is mounted.
- Replace **my-folder** with your windows folder you want to mount.
- Replace **new-folder** with your directory name where you want it to appear inside Ubuntu.

To unmount-  
**\$ umount <foldername>**

# mount/umount command options

## Examples-

### **mount -t type <device> <directory>**

It will mount the filesystem of device at the specified directory. If the directory is not given, it will look for the mount-point in **/etc/fstab** file. The **/etc/fstab** file contains information about which device should be mounted where.

### **mount**

It will display all currently mounted filesystems on a system.

### **mount -V or mount--version**

It will display the version information of mount utility.

### **Umount <devicefile>**

It will umount (detach) the filesystem of device.

# Cron: Automated Jobs

- A **cron job** is a task scheduled to run at regular intervals using the **cron** daemon.
- We define cron jobs using a **crontab** (cron table) file, which lists jobs in a specific format.
- **Crontab Syntax-**

|   |   |   |   |   |                                      |
|---|---|---|---|---|--------------------------------------|
| * | * | * | * | * | command_to_run                       |
|   |   |   |   |   |                                      |
|   |   |   |   | └ | Day of week (0-7) (Sunday is 0 or 7) |
|   |   |   | └ |   | Month (1 - 12)                       |
|   |   | └ |   |   | Day of month (1 - 31)                |
|   | └ |   |   |   | Hour (0 - 23)                        |
| └ |   |   |   |   | Minute (0 - 59)                      |

# Creating/Editing Crontab File

- To open your user's crontab-

```
$ crontab -e
```

- To view current cron jobs-

```
$ crontab -l
```

- Example-

- To run a script every day at 6 AM-

```
0 6 * * * /home/user/myscript.sh > /dev/pts/0
```

- By default, cron jobs have **no terminal** and do **not show output** unless explicitly redirected to the terminal (/dev/pts/0).



# Cron Jobs Examples

- To run a command every 5 minutes-

```
*/5 * * * * echo "Hello" > /dev/pts/0
```

- To run a backup script every Sunday at 1 AM-

```
0 1 * * 0 /home/amitabh/backup.sh
```

# Firewall Basics

- **Firewall** is a virtual wall that is designed to protect our system from unwanted traffic and unauthorized access to our system.
- **iptables** is a user-space utility program that allows a system administrator to configure the **IPv4 packet filtering rules** of the **Linux kernel firewall**, which is part of the **netfilter** framework.
- **iptables** is divided into 4 tables, each used for different types of packet processing-

| Table  | Purpose   |
|--------|---|
| filter | Default table, used for <b>packet filtering</b> |
| nat    | Used for <b>Network Address Translation</b>     |
| mangle | Alters packet headers (QoS, TTL, etc.)          |
| raw    | Used for exemptions from connection tracking    |

# iptables- chains

- Each table contains **chains (pipelines)**, which are actually the lists of rules applied to packets.

| Chain   | Description                                     |
|---------|---|
| INPUT   | For packets <b>destined to the local system</b> |
| OUTPUT  | For packets <b>originating from the system</b>  |
| FORWARD | For packets <b>passing through the system</b>   |

- Each chain has a list of rules. A rule specifies-
  - ✓ Matching conditions (e.g., source IP, port, protocol)
  - ✓ Target action (e.g., ACCEPT, DROP, REJECT)
- **Example rule-**

```
iptables -A INPUT -p tcp --dport 22 -j ACCEPT
```

  - Append to INPUT chain
  - Match: TCP packets with destination port 22
  - Action: ACCEPT

# iptables- rules:target

- Following are the targets we can use while writing rules to define what happens if a packet matches a rule-

| Target | Meaning   |
|--------|---|
| ACCEPT | Allow the packet                                  |
| DROP   | Silently discard the packet                       |
| REJECT | Block the packet and send a response (e.g., ICMP) |
| LOG    | Log the packet to syslog                          |

# iptables- examples

- **To check existing rules-**

```
$ sudo iptables -L -v -n
```

L: list, v: verbose, n: no DNS lookup

- **To block all incoming traffic-**

```
$ sudo iptables -P INPUT DROP
```

now, no incoming connections are allowed.

*(Check it with **ping** command.)*

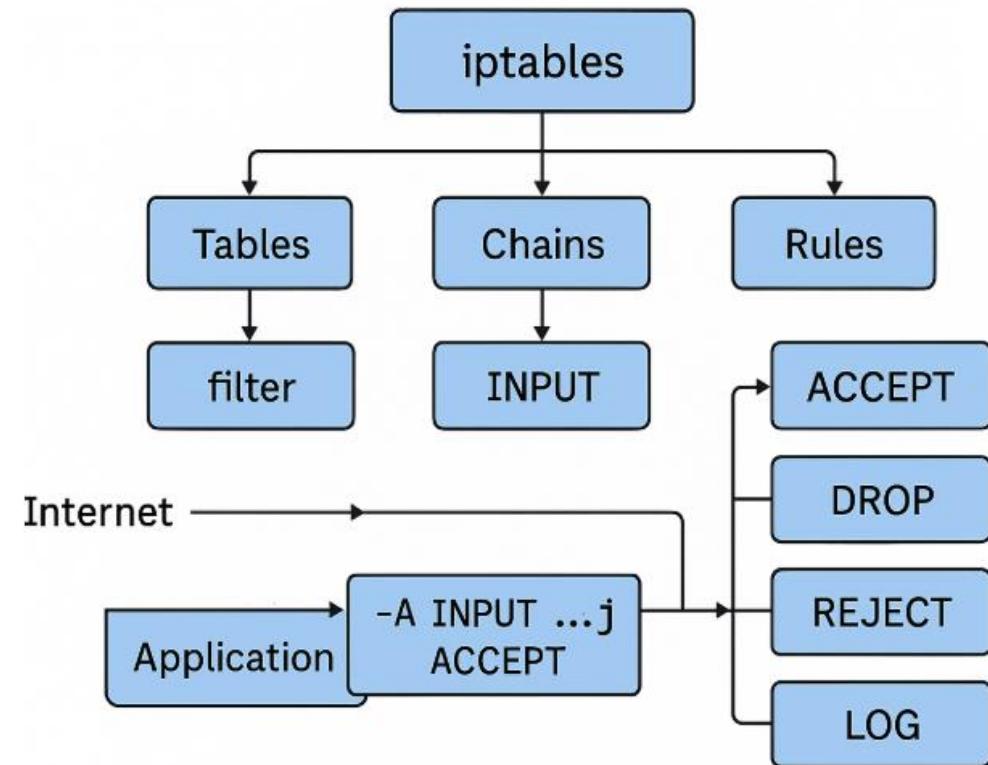
- **Allow only SSH (port 22)-**

```
$ sudo iptables -P INPUT DROP
```

```
$ sudo iptables -A INPUT -p tcp --dport 22 -j ACCEPT
```

```
$ sudo iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
```

- The above rules will block everything except SSH and existing/related connections.



# iptables- more examples

- **Block specific IP Address-**

```
$ sudo iptables -A INPUT -s 192.168.1.100 -j DROP
```

This rule will block all traffic from specified IP.

- **Allow Web Server (port 80)-**

```
$ sudo iptables -A INPUT -p tcp --dport 80 -j ACCEPT
```

This rule will allow incoming HTTP traffic.

- **Allow Loopback Interface-**

```
$ sudo iptables -A INPUT -i lo -j ACCEPT
```

This rule will allow many local apps to access loopback address.

- **Flush All Rules-**

```
$ sudo iptables -F
```

# iptables- Reset All Rules

- To **fully reset the firewall** to accept all traffic by default, do the following steps-

1. **Flush all rules-**

```
$ sudo iptables -F
```

2. **Set default policies to ACCEPT-**

```
$ sudo iptables -P INPUT ACCEPT
```

```
$ sudo iptables -P FORWARD ACCEPT
```

```
$ sudo iptables -P OUTPUT ACCEPT
```

# ufw (Uncomplicated FireWall)

- **UFW (Uncomplicated Firewall)** is a command-line tool in Linux, primarily used in Debian and Ubuntu systems, to manage and configure a firewall based on iptables.
- It simplifies firewall management by providing a more user-friendly interface than iptables.
- UFW allows to control network traffic by allowing or denying connections based on ports, services, or IP addresses.
- **Installation-** sudo apt update; sudo apt install ufw
- **Status check-** sudo ufw status
- **Enabling ufw-** sudo ufw enable                    **Disabling ufw-** sudo ufw disable

# ufw: Allowing/Denying Connections

- **Allow a specific port-**

```
$ sudo ufw allow <port>
```

- **Allow a service-**

```
$ sudo ufw allow <service_name>
```

**example-** \$ sudo ufw allow ssh

- **Allow connection from specific IP address-**

```
$ sudo ufw allow from <ip_address>
```

- **Deny a specific port-**

```
$ sudo ufw deny <port>
```

- **Deny a service-**

```
$ sudo ufw deny <service_name>
```

# Secure Shell (SSH)

- **SSH (Secure Shell)** is used to **securely access and manage remote systems** over a network. It encrypts the connection and allowing us to Log into the remote system/server, execute commands remotely, transfer files securely, manage automated tasks.
- To setup SSH, first we need to install OpenSSH Server package-  
`$ sudo apt update`  
`$ sudo apt install openssh-server`
- To check that the SSH server is running-  
`$ sudo systemctl status ssh`
- To start the SSH service-  
`$ sudo systemctl start ssh`
- To start the SSH service at boot-  
`$ sudo systemctl enable ssh`



# SSH Client- Connecting to the Server

- On the client machine (another Ubuntu system or same system using Localhost), use the following command syntax to SSH into the server-

**ssh username@server\_ip**

## Example-

\$ ssh student07@192.168.17.88

OR

\$ ssh student07@localhost

- It will ask for the user's password. After entering the correct user's password, the user logs into the server remotely through client machine.

# Docker

- **Docker** is a software program that creates containers.
- **Containers** are built to run fully developed applications that could be shared and run on other computers.
- For example, if one computer contains the source code for a Word Generator application, it could be distributed through other computers by first converting the source code into an image and running it to make a container to access the application.
- The image can be copy and load onto another system and run it.

# Installing Docker Desktop (Windows)

## Installation Steps-

1. Go to the website <https://docs.docker.com/docker-for-windows/install/> and download the docker file.
2. Then, double-click on the Docker Desktop Installer.exe to run the installer.
3. Once you start the installation process, always enable Hyper-V Windows Feature on the Configuration page.
4. Then, follow the installation process to allow the installer and wait till the process is done.
5. After completion of the installation process, click Close and restart the system.
6. Open the Ubuntu terminal and type the command- **docker**

# Installing Docker in Ubuntu

## Installation Steps-

1. \$ sudo apt update
  
2. \$ sudo apt install docker.io -y
  
3. \$ sudo systemctl enable docker --now
  
4. Open the Ubuntu terminal and check docker version-  
\$ docker --version

# Download Docker Image

**To download an existing image-**

**Syntax-**      docker pull <image:tag>  
                  \$ docker pull ubuntu:latest

**To check the downloaded images-**

\$ docker images

**To run an existing image-**

**Syntax-**

docker run -i --name=<name\_of\_container> -t <image\_name> /bin/bash  
\$ docker run -i --name=my\_ubuntu -t ubuntu /bin/bash

Type 'exit' command to leave  
the container.

# Saving and loading Docker Image

## To save an existing/created image-

Syntax-      docker save -o <filename.tar> <image\_name>  
                  \$ docker save -o my\_ubuntu.tar ubuntu

## To load an image from saved file-

Syntax-      docker load -i <filename.tar>  
                  \$ docker load -i my\_ubuntu.tar

## To check a loaded image-

\$ docker images

### To delete an image-

\$ docker rmi image:tag --force