**Linux**

* **What is an Operating System?**

The OS is software specifically designed to operate, control, and coordinate all the hardware and software resources available in the computer.

An **operating system (OS)** is like the manager of a computer. It helps all the parts of the computer work together smoothly. Here’s a simple way to think about it:

1. **Coordinator**: It coordinates between the computer’s hardware (like the keyboard, mouse, and screen) and software (like apps and programs).
2. **Resource Manager**: It manages the computer’s resources, such as memory and processing power, to ensure everything runs efficiently.
3. **User Interface**: It provides a way for you to interact with the computer, like through windows, icons, and menus.

In essence, the OS is the backbone that allows you to use your computer. Examples include Windows, macOS, Linux, and Android.

A diagram of a application

Description automatically generated

## What is Linux?

## Linux is an open-source operating system based on the Unix architecture. It was created by Linus Torvalds in 1991 and has since grown into a robust and flexible OS used worldwide.

## Key Features of Linux:

## Open Source: The source code of Linux is freely available to anyone. This means anyone can view, modify, and distribute the code, fostering a collaborative development environment.

## Security: Linux is known for its strong security features. It has a robust permission system and is less susceptible to viruses and malware compared to other operating systems.

## Stability and Performance: Linux systems are highly stable and can run for long periods without crashing. They are also efficient in managing system resources, making them ideal for servers and high-performance computing.

## Flexibility: Linux can be customized to meet specific needs. There are various distributions (distros) of Linux, each tailored for different purposes, such as Ubuntu for general use, CentOS for servers, and Kali Linux for security testing.

## Community Support: A large and active community of developers and users contributes to Linux. This community support ensures continuous improvement and a wealth of resources for troubleshooting and learning.

## Components of Linux: (See diagram below of Shukla)

## Kernel: The Linux Kernel is like the brain of the operating system because it manages how the computer interacts with its hardware and resources. It is the intermediary between hardware and software. It makes sure everything works smoothly and efficiently

## System Libraries: These are special functions or programs that applications use to interact with the kernel.

## System Utilities: Basic tools and commands that perform essential tasks, such as file management and system monitoring.

## User Interface: Linux can be used with a graphical user interface (GUI) or a command-line interface (CLI). Popular GUIs include GNOME and KDE.

## Popular Linux Distributions:

## Ubuntu: User-friendly and widely used for desktops and servers.

## Fedora: Known for its cutting-edge features and technologies.

## Debian: Stable and versatile, often used as a base for other distributions.

## CentOS: Community-supported version of Red Hat Enterprise Linux, ideal for servers.

## Arch Linux: Aimed at advanced users who prefer to customize their system from the ground up.

* **Why Linux?**
* **Open Source**: Free to use and modify, promoting collaboration and innovations in the community.
* **Security**: Known for its robust security features and resistance to malware and viruses.
* **Flexibility**: It can be used for anything from a desktop to servers and also embedded systems.
* **Performance**: Efficient resource management, suitable for both high-performance computing and low-spec devices.
* **Community Support**: A very large, active community offering a lot of support and resources in terms of documentation and tools.
* **Difference between linux and unix**

| **Feature** | **Linux** | **Unix** |
| --- | --- | --- |
| **Origin** | Created by Linus Torvalds in 1991 | Developed at AT&T’s Bell Labs in the late 1960s and early 1970s |
| **License** | Free and open-source | Mostly paid and proprietary |
| **Development** | Built by a global community of developers | Developed by specific companies (e.g., IBM, HP) |
| **Cost** | Free to use | Often requires buying a license |
| **Customization** | Highly customizable with many versions (distros) | Less customizable, more standardized |
| **Kernel Type** | Modular, can add/remove parts while running |  |
| **Usage** | Used on PCs, servers, phones, and more | Mainly used in big companies and servers |
| **Security** | Very secure, less prone to viruses | Very stable and reliable |
| **Examples** | Ubuntu, Fedora, Debian, CentOS | IBM AIX, HP-UX, Oracle Solaris |
| **Performance** | Efficient and good for high-performance tasks | Known for stability and performance in business environments |
| **User Interface** | Can use both graphical (GUI) and command-line (CLI) | Mostly command-line (CLI), some have graphical interfaces |

# **What are Linux Distributions ?**

A complete Linux system package called a distribution. Many Linux distributions are available to meet just about any computing requirement you could have. Most distributions are customized for a specific user group, such as business users. Multimedia enthusiasts, software developers, or average home users. The two most popular linux distributions used are Redhat and Ubuntu. Redhat is popular in Banks, Airlines, Telecoms, Healthcare, Government. Ubuntu is popular in SaaS, Social Networks, Cloud Based.

**The different Linux distributions are often divided into three categories:**

* Full core Linux distributions
* Specialized Linux distributions
* LiveCD test distributions
* **Core Linux Distributions:** A core Linux distribution contains a kernel, one or more graphical desktop environments, and just about every Linux application that is available, recompiled for the kernel. It provides one-stop shopping for a complete Linux installation.

**Examples:**

* **Slackware**– One of the original Linux distribution sets, popular with Linux geek.
* **Debian**– Popular with Linux experts and commercial Linux products

### **Specialized Linux Distributions:** A new subgroup of Linux distributions has started to appear. These are typically based on one of the main distributions but contain only a subset of applications that would make sense for a specific area of use. These are literally hundreds of specialized Linux distributions, and more are popping up all the time on the internet. No matter what your specialty, you’ll probably find a Linux distribution made for you.

**Examples :**

* **CentOS**– A free distribution built from the Red Hat Enterprise Linux source code
* **Mint –**A free distribution for home entertainment use

**Popular Linux Distributions:**

1. **Ubuntu:**
   * **User-Friendly:** Known for its ease of use and strong community support.
   * **Target Audience:** Beginners and general users.
   * **Package Management:** Uses APT and DEB packages.
2. **Fedora:**
   * **Cutting-Edge:** Features the latest software and technologies.
   * **Target Audience:** Developers and tech enthusiasts.
   * **Package Management:** Uses DNF and RPM packages.
3. **Debian:**
   * **Stability:** Known for its stability and reliability.
   * **Target Audience:** Servers and advanced users.
   * **Package Management:** Uses APT and DEB packages.
4. **CentOS:**
   * **Enterprise-Grade:** Community-supported version of Red Hat Enterprise Linux (RHEL).
   * **Target Audience:** Servers and enterprise environments.
   * **Package Management:** Uses YUM and RPM packages.
5. **Arch Linux:**
   * **Customizable:** Minimalist and highly customizable.
   * **Target Audience:** Advanced users who prefer to build their system from scratch.
   * **Package Management:** Uses Pacman.
6. **Kali Linux:**
   * **Security-Focused:** Designed for penetration testing and security research.
   * **Target Audience:** Security professionals and ethical hackers.
   * **Package Management:** Uses APT and DEB packages.

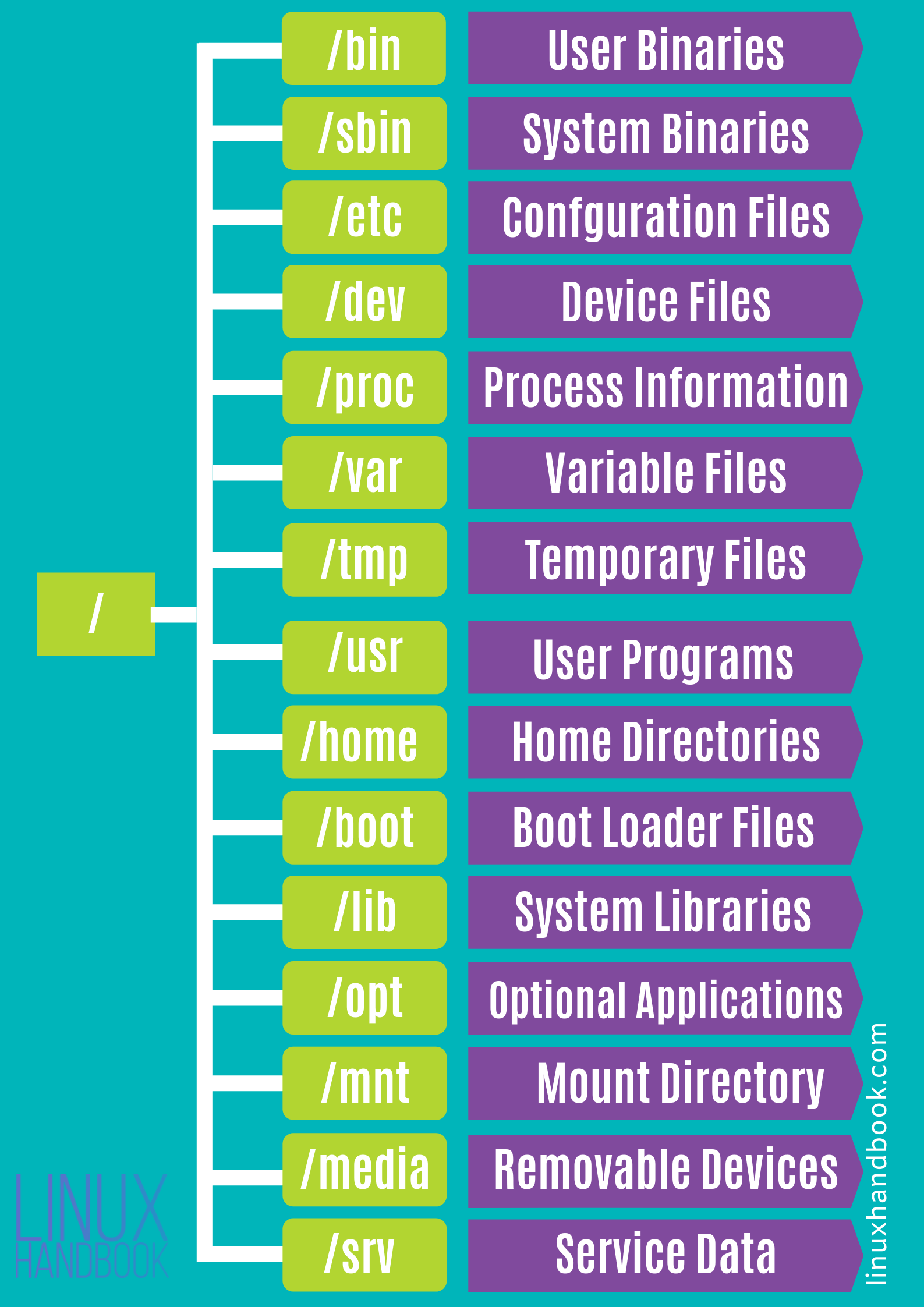
# **Linux Directory Structure**

In Linux operating system everything is a file even directories are files, files are files, and devices like mouse, keyboard, printer, etc are also files.

## ****Types**** of files in the Linux system.

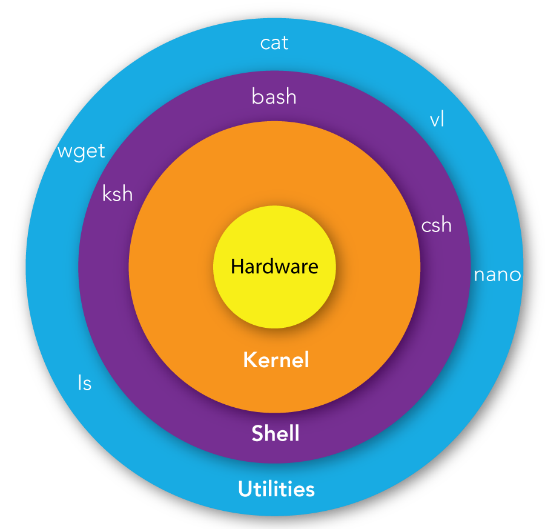
1. **General Files** – It is also called ordinary files. It may be an image, video, program, or simple text file. These types of files can be in ASCII or Binary format. It is the most commonly used file in the Linux system.
2. **Directory Files**– These types of files are a warehouse for other file types. It may be a directory file within a directory (subdirectory).
3. **Device Files –** In a Windows-like operating system, devices like CD-ROM, and hard drives are represented as drive letters like F: G: H whereas in the Linux system devices are represented as files. As for example, /dev/sda1, /dev/sda2, and so on.

 In a Windows-like operating system, files are stored in different folders on different data drives like C: D: E: whereas in the Linux/Unix operating system files are stored in a tree-like structure starting with the root directory as shown in the below diagram.



The Linux directory structure is organized in a hierarchical way, similar to a tree. Here’s a simple explanation of the main directories you’ll find in a typical Linux system:

* **Root Directory ( / )**: The main folder that contains all other folders and files.
* **/bin**: Holds important programs needed for the system to start and run, like basic commands (e.g., ls, cp, mv).
* **/boot**: Contains files needed to start the computer, including the Linux kernel.
* **/dev**: Contains files that represent hardware devices (e.g., hard drives, printers).
* **/etc**: Stores system settings and configuration files (e.g., network settings, user accounts).
* **/home**: Each user has a personal folder here (e.g., /home/john for user John).
* **/root**: The home folder for the root (superuser) account.
* **/lib**: Contains essential libraries needed by programs in /bin and /sbin.
* **/media**: Used for mounting (connecting) removable media like USB drives and CDs.
* **/mnt**: A place to temporarily mount (connect) filesystems (e.g., additional hard drives).
* **/opt**: Used for installing optional software packages.
* **/proc**: A virtual folder that provides information about running processes and system details.
* **/run**: Stores temporary data for processes started since the last boot.
* **/sbin**: Contains important system programs used by the root user (e.g., fdisk, reboot).
* **/srv**: Holds data for services provided by the system (e.g., web servers).
* **/sys**: Another virtual folder that provides information about the system and hardware.
* **/tmp**: A temporary folder where programs can store temporary files.
* **/usr**: Contains user-related programs and data, with subfolders like:
  + **/usr/bin**: Non-essential programs for users.
  + **/usr/lib**: Libraries for programs in /usr/bin.
  + **/usr/local**: Locally installed software and custom scripts.
  + **/usr/share**: Shared data like documentation and icons.
* **/var**: Contains files that change frequently, like logs and temporary files created by applications.
* **Linux Shell**
* Shell is a command interpreter. It is the layer between the operating system kernel and the user. Shells provide a "command line" interface which allows the user to enter commands which are translated by the shell into something the kernel can comprehend and then is sent off to the kernel for it to act upon. To check which is your shell enter the command: echo $0

 Say on top, like on top of hardware is kernel

Types of Shells in Linux

* **Bash (Bourne Again Shell)**: The most common and default shell in many Linux distributions. Path: /bin/bash
* **sh (Bourne Shell)**: The original Unix shell developed by Stephen Bourne. Path: /bin/sh
* **csh (C Shell)**: A shell with syntax similar to the C programming language, developed by Bill Joy. Path: /bin/csh
* **ksh (Korn Shell)**: Developed by David Korn, it includes features of both the Bourne and C Shells. Path: /bin/ksh
* **zsh (Z Shell)**: An extended Bourne Shell with many improvements, including better customization and interactive features. Path: /bin/zsh
* **Linux Commands**
* ssh: The ssh (Secure Shell) command in Linux is used to securely connect to remote systems over a network. Syntax: ssh username@ipaddress
* Ls: This command will list all the files in your current working directory
* Ls -l: gives us a nice list
* Ls -a: to see the hidden files
* Ls -t: It shows the file time created
* Ls -f: It is used to reveal the file type. It can be directory (/), Link(@) or Executable(\*)
* Ls -r: Reverse order, printing data
* Pwd: Tells us the present (current) working directory
* Cd: We use this command to change the directory. cd dir will take you to the particular directory. Cd / will take you to the root of the file system. Cd .. will take you one path back
* Cat: To see what is inside the file use the cat command. Cat filename
* Clear: clear the screen, also you can press Ctrl+l
* Man: the man command is short for manual. It displays details about a certain command. Man command\_name. Keyboard commands of man command
* Enter: One line down
* Space: display the next page
* g: Top of the page
* G: Bottom of the page
* Q: quit
* Exit: To exit from current working session. We can also use exit in conditional statements. If exit returns 0 means the code was successful, other than 0 code is not successful.
* Help: This command is used to get information about the other command, like how can we work with it. Syntax: -h or --help. Example: ls -h or ls -help
* Mkdir: this command is used to make a new directory. Syntax: mkdir [-P] directory\_name. The -P is for parent and it is optional.
* Rmdir: we can remove a directory using this command. It only removes the empty directory and if it has some contents in it. Syntax: rmdir [-P] directory\_name. The -P is for parent and it is optional.
* Rm -rf: Remove recursively. This command removes the directory alongwith its contents. There is no undo or redo in linux so if you delete the directory, its permanently gone.
* Touch: it is the easiest way to create a file. Touch filename. We can also create a file in the future using touch -d command
* Echo: this command is used to print something. Echo data > filename, allows you to add data to a new or an existing file
* Find: This command is used to find the file or directory. Use find for more detailed and flexible searches, especially when you need to search based on criteria other than just the file name. Syntax: Sudo file / -name “filename”
* Locate: This command lists the files that match the pattern. It is faster than find command. Use locate for quick, simple searches when you need speed and the database is up-to-date. Syntax: locate pattern. Locate command may not be enable on all systems
* Nano command: This command is used to edit the details of the file. Nano filename. After editing the file, press ctrl X, and Y and press enter.
* Vim: This command is also used to edit the details of the file. Vim filename. Enter I to start inserting text, after editing press ESC, colon(:) and write wq to quit.
* Cp: we can copy a file using this command. Cp filename destination path. More description is given below
* Mv: we can move the file using this command. Mv filename destination path.
* Rm: We can remove a file using this command. Rm filename. We can also delete a directory using this command. Rm -r directory name.
* Ln: we can create a link to a file using this command. Ln -s filename and then link
* Whoami: to know about who you are, type this command
* Useradd command: To add an user to the terminal. sudo useradd username
* Su: We can switch users using su command. Su username
* Finger: Finger command is used to inspect the user. Finger username
* Whatis: Shorter version of man command, gives less information
* Which: To know where the command is stored use this command. Which command\_name
* Wget: This command helps to get stuff from the internet. Wget link
* Curl: this command also downloads the stuff and stores in into a file. Curl link > filename
* Less: shows only single page of details of a particular file, similar to man command. Less filename
* Head: to see beginning details of the file use the head command. head filename
* Tail: to see the end details of the file use the tail command. tail filename
* Cmp: this command is used to compare two files. cmp file1 file2
* Diff: this command is used to compare two files. diff file1 file2. It compares and shows the main difference.
* Sort: This command is used to sort the contents of the file. Cat filename | sort
* Chmod: The chmod command is used to change the mode to rwx (Read, Write and Execute). Chmod +x
* Chown: change the ownership of the file using this command. Chown username filename
* Ifconfig: to know the ip address. Ifconfig
* Ipaddress: same as above
* Grep: study from below
* Ping: to know if your website is up. Ping -c 5 websitename
* Netstat: netstat is a command in Linux that shows you information about your network connections and ports. It’s like a tool to see what’s happening with your network. Syntax: netstat
* Ss: same as above
* Uname: to get to know about the system. Uname -a
* Uniq: The uniq command is used to report or filter out repeated lines in a file. It only removes adjacent duplicate lines, so it is often used in combination with sort.
* Cal: to display the calendar. cal
* Free: to know about the memory, like how much memory is free and how much is used. free
* Ps: To know about the processes on the system use this command. Ps -aux
* Dmesg: The dmesg command in Linux is used to display and control the kernel ring buffer messages and kernel log messages.
* History: to know about all the commands you worked on use this command.
* Reboot: sudo reboot
* Shutdown: sudo shutdown
* **Directory Shortcuts**
* **Tilde(**~**):** shortcut to your home base
* **Dot(.):** It shows the current directory that you are working in
* **double dot(..):** It shows the parent directory
* **cd -:** Change to previous working directory
* **Hard and Soft link (Also see from the Shukla notes)**

**Hard Link:** A hard link is an additional name for an existing file. Both the original file and the hard link share the same inode number, meaning they point to the same data on the disk. Changes made to one will reflect in the other.

**Characteristics:**

* **Same Inode**: Hard links share the same inode number as the original file.
* **Data Integrity**: Even if the original file is deleted, the data remains accessible through the hard link.
* **Limitations**: Cannot be created for directories and cannot span across different filesystems.

**Example:**

# Create a file

echo "Hello, World!" > original.txt

# Create a hard link

ln original.txt hardlink.txt

# Check inode numbers

ls -i original.txt hardlink.txt

Output:

123456 original.txt

123456 hardlink.txt

Both files have the same inode number, indicating they point to the same data.

**Soft Links (Symbolic Links):** A soft link is a special type of file that contains a reference to another file or directory in the form of a path. Unlike hard links, soft links have their own inode and can span across different filesystems.

**Characteristics:**

* **Different Inode**: Soft links have a different inode number from the original file.
* **Path Reference**: They store the path to the target file.
* **Flexibility**: Can link to directories and can span across different filesystems.
* **Dependency**: If the original file is deleted, the soft link becomes a “dangling” link and no longer works.

**Example:**

# Create a file

echo "Hello, World!" > original.txt

# Create a soft link

ln -s original.txt softlink.txt

# Check inode numbers

ls -i original.txt softlink.txt

Output:

123456 original.txt

654321 softlink.txt

The inode numbers are different, indicating that the soft link is a separate file that points to the original file.

* **File and directory permissions in Linux**

File and directory permissions in Linux determine who can read, write, and execute files and directories. These permissions are crucial for system security and proper user access control. Understanding how to view and modify these permissions is essential for any Linux user or administrator.

* **Permission Types:** There are three types of permissions in Linux:

1. **Read (r)**: Permission to read the contents of a file or list the contents of a directory.
2. **Write (w)**: Permission to modify the contents of a file or directory.
3. **Execute (x)**: Permission to execute a file (if it is a script or binary) or Allows access to contents and metadata for entries

* **Permission Categories:** Permissions are assigned to three categories of users:

1. **Owner (u)**: The user who owns the file or directory.
2. **Group (g)**: The group to which the file or directory belongs. Every user belongs to atleats one group. Groups are used to organize users
3. **Others (o)**: All other users.
4. **All (a)**: It represents everybody

A diagram of a computer program

Description automatically generated

* **Viewing Permissions:** To view the permissions of files and directories, use the ls -l command. This command outputs something like: -rwxr-xr—

1. **First character**: File type (- for regular file, d for directory, l for symlink, etc.).
2. **Next nine characters**: Permissions, divided into three sets of three:
   * First set (3 characters): Owner permissions.
   * Second set (3 characters): Group permissions.
   * Third set (3 characters): Others permissions.

For example, -rwxr-xr-- translates to:

* rwx (read, write, execute) for the owner.
* r-x (read, execute) for the group.
* r-- (read only) for others.
* **Changing Permissions**

**Using chmod:** You can change file and directory permissions using the chmod command. There are two ways to specify permissions: symbolic mode and numeric mode.

* **Symbolic Mode:** In symbolic mode, you use letters to specify permissions:Example: chmod u+rwx,g+rx,o+r file.txt

You can also remove permissions:

chmod g-w file.txt

* **Numeric Mode:** In numeric mode, you use a three-digit octal number to represent the permissions:

chmod 755 file.txt

The digits represent the sum of **read (4), write (2), and execute (1)** permissions:

* 7 (4+2+1) means read, write, and execute.
* 5 (4+1) means read and execute.
* 5 (4+1) means read and execute.
* **Commonly used permissions**

700: -rwx------

755: -rwxr-xr-x

660: -rw-rw----

664: -rw-rw-r--

644: -rw-r--r--

* **File creation Mask**

The file creation mask, commonly known as umask in Linux, is a default setting that controls the permissions set for newly created files and directories. The umask command in Linux sets the default permissions for new files and directories. Here’s a simple explanation:

**What is**umask**?**

umask stands for “user file creation mask.” It determines the default permissions for new files and directories. When you create a file or directory, the system uses the umask value to set the permissions.

**How Does**umask**Work?**

Permissions in Linux are represented by three sets of three bits: read ®, write (w), and execute (x). Each set corresponds to the owner, group, and others.

The umask value subtracts permissions from the default settings:

* Default permissions for files: 666 (read and write for everyone)
* Default permissions for directories: 777 (read, write, and execute for everyone)

**Example**

Let’s say your umask value is 022. Here’s how it works:

1. **Check Current**umask**Value**:

umask

Output might be:

0022

1. **Create a New File**:

touch newfile.txt

1. **Check File Permissions**:

ls -l newfile.txt

Output might be:

-rw-r--r-- 1 user group 0 Aug 26 09:19 newfile.txt

**Explanation**

* **Default File Permissions**: 666 (read and write for everyone)
* **umask Value**: 022 (subtracts write permission for group and others)
* **Resulting Permissions**: 644 (read and write for owner, read-only for group and others)
* **Finding files and Directories**

Finding files and directories in Linux is a common task. The most commonly used commands for searching files and directories are **find, locate,** and **which**.

* **find Command:** The find command is a powerful tool for searching files and directories based on various criteria such as name, type, size, permissions, and modification time.

**Basic Syntax:**

find [path] [expression]

**Examples**

1. **Find files by name**: find /path/to/search -name "filename"
2. To perform a case-insensitive search: find /path/to/search -iname "filename"
3. **Find directories by name**: find /path/to/search -type d -name "directoryname"
4. **Find files by extension**: find /path/to/search -type f -name "\*.txt"
5. **Find files modified in the last 7 days**: find /path/to/search -type f -mtime -7
6. **Find files larger than 100MB**: find /path/to/search -type f -size +100M

* **locate Command:** The locate command searches for files and directories by name using a pre-built database. It is faster than find but depends on the database being up-to-date. It is faster than find command. Syntax: locate [pattern]

**Examples**

1. **Find files or directories by name**: locate filename
2. **Find files with a specific extension**: locate "\*.txt"

* **Viewing Files**

A screenshot of a computer

Description automatically generated

In Linux, there are several commands to view the contents of files, each with its own use cases and advantages

1. Cat: The cat command is used to concatenate and display the content of files. Syntax: cat filename
2. Less: The less command is used to view the content of files one screen at a time. Syntax: less filename
3. More: The more command is similar to less, but with fewer features. Syntax: more filename
4. Head: The head command displays the first few lines of a file (default is 10 lines). Syntax: head filename
5. Tail: The tail command displays the last few lines of a file (default is 10 lines). Syntax: tail filename
6. Keyboard buttons: When using more or less, press spacebar to advance to next page, use enter key to advance one line, and type q to quit the file.

* **Nano Editor**

Nano is a simple, user-friendly text editor that is included with many Linux distributions. It is a good choice for beginners due to its simplicity and ease of use. It is not as advanced as vi or emacs. Syntax: nano filename. Basic Commands in nano: Moving Around, Editing, Searching, Saving and Editing. To exit nano editor, press ctrl + x.

* **VI Editor (Important answer)**

The vi editor, short for "visual editor," is a powerful and versatile text editor available on almost all Unix-like systems, including Linux. It has a steep learning curve but offers a wide range of features for efficient text editing. vi has been succeeded by vim (Vi IMproved), which includes enhancements and additional features.

**Modes in vi**

vi operates in different modes, primarily:

1. **Normal Mode**: For navigation and command execution.
2. **Insert Mode**: For inserting text.
3. **Visual Mode**: For selecting text.
4. **Command-Line Mode**: For running commands

**Basic Commands in Normal Mode**

* **Navigation**:
  + h: Move left.
  + j: Move down.
  + k: Move up.
  + l: Move right.
  + ^: Move to the beginning of the line.
  + $: Move to the end of the line.
  + w: Move to the beginning of the next word.
  + b: Move to the beginning of the previous word.
* **Editing**:
  + x: Delete the current character
  + dd: Delete the current line.
  + yy: Yank (copy) the current line.
  + p: Paste after the cursor.
  + P: Paste before the cursor.
  + u: Undo the last action.
  + Ctrl + r: Redo the undone action.
* **Searching**:
  + /pattern: Search forward for pattern.
  + ?pattern: Search backward for pattern.

**Commands in Insert Mode**

* **Insert Text**: Start typing to insert text at the cursor position.
* **i**: insert at cursor position
* **I**: Insert at the beginning of the line
* **Backspace**: Delete the character before the cursor.
* **Ctrl + h**: Delete the character before the cursor.
* **Ctrl + w**: Delete the word before the cursor.
* **Ctrl + u**: Delete everything before the cursor on the current line.
* **dw**: Delete Word
* **dd**: Delete Line

**Saving and Exiting**

In Command-Line Mode:

* :w: Write (save) the file.
* :q: Quit vi.
* :wq or :x: Write and quit.
* **Basic Vim commands**
* w: Write the current file
* wq: Write the current file and exit.
* :q!: Quit without writing
* i or a: To change into insert mode
* esc button: To exit the mode
* N: search forward /, repeat the search backwards
* Set nu: turn on line numbering
* Set nonu: turn of line numbering
* Basic movement:
* x: delete
* u: undo
* **Emacs Editor**

Emacs is a highly customizable and extensible text editor widely used in Unix-like operating systems, including Linux. It is known for its powerful features, extensive package system, and the ability to be customized with Emacs Lisp. Emacs can be used for a variety of tasks, from simple text editing to complex software development.

When you start Emacs, you'll enter the editor's main interface. The screen is divided into several parts:

* **Text Area**: The main area where you edit text.
* **Mode Line**: Displays information about the current buffer (e.g., file name, editing mode).
* **Minibuffer**: Located at the bottom, used for commands and prompts.

**Basic Commands**

**Navigation**

* C-f: Move forward one character.
* C-b: Move backward one character.
* C-n: Move to the next line.
* C-p: Move to the previous line.
* M-f: Move forward one word.
* M-b: Move backward one word.
* C-a: Move to the beginning of the line.
* C-e: Move to the end of the line.
* M-<: Move to the beginning of the buffer.
* M->: Move to the end of the buffer.

**Editing**

* C-d: Delete the character under the cursor.
* M-d: Delete the word forward.
* C-k: Kill (cut) text from the cursor to the end of the line.
* M-k: Kill (cut) text from the cursor to the end of the sentence.
* C-y: Yank (paste) the most recently killed text.
* C-w: Kill (cut) the selected region.
* M-w: Copy the selected region.

**Undo and Redo**

* C-/ or C-x u: Undo the last action.
* C-g: Cancel the current command.

**File Operations**

* C-x C-f: Open a file.
* C-x C-s: Save the current buffer.
* C-x C-w: Write the buffer to a specific file (Save As).
* C-x C-c: Exit Emacs.

**Searching**

* C-s: Incremental search forward.
* C-r: Incremental search backward.
* M-%: Search and replace.
* **Difference between vi editor and emacs editor**

1. **Design Philosophy**

* vi (Vim): Vim, short for “Vi Improved,” is designed to be a highly efficient text editor. It focuses on speed and minimalism, with a small footprint and quick startup time. [Vim operates in different modes (normal, insert, visual, and command) to streamline text editing tasks1](https://www.linuxfordevices.com/tutorials/linux/emacs-vs-vim).
* Emacs: Emacs is more than just a text editor; it’s an extensible computing environment. It provides a wide range of functionalities beyond text editing, such as email, web browsing, and even games. [Emacs uses a Turing complete language, Lisp, for customization1](https://www.linuxfordevices.com/tutorials/linux/emacs-vs-vim).

**2. User Interface**

* vi (Vim): Primarily operates in the terminal with a command-line interface. [It uses different modes for different tasks, which can be challenging for beginners but very efficient once mastered2](https://linuxsimply.com/linux-basics/text-editors/comparison/emacs-vs-vim/).
* Emacs: Offers both a graphical user interface (GUI) and a command-line interface. [It supports menus, toolbars, and dialog boxes, making it more approachable for users familiar with GUI-based editors](https://www.linuxfordevices.com/tutorials/linux/emacs-vs-vim)[2](https://linuxsimply.com/linux-basics/text-editors/comparison/emacs-vs-vim/).

**3. Customizability**

* vi (Vim): Highly configurable through its scripting language, Vimscript. [Users can create custom keybindings and automate tasks, but it generally focuses on text editing enhancements](https://www.linuxfordevices.com/tutorials/linux/emacs-vs-vim)[3](https://monovm.com/blog/emacs-vs-vim/).
* Emacs: Extremely customizable with Emacs Lisp. [Users can extend Emacs to perform virtually any task, from text editing to managing files, browsing the web, and more3](https://monovm.com/blog/emacs-vs-vim/)
* **Graphical Editor in Linux**

Graphical editors in Linux provide a user-friendly interface for text editing and are often preferred by users who are more comfortable with graphical interfaces than command-line tools. Here are some of the most popular graphical text editors available in Linux:

1. **Gedit**: Gedit is the default text editor for the GNOME desktop environment. It is simple and user-friendly, making it ideal for basic text editing tasks.

**Features**:

* Syntax highlighting for various programming languages
* Undo and redo
* Search and replace
* Auto-indentation
* Plugins support

1. **Kate:** Kate is a powerful text editor for the KDE desktop environment, but it can be used on any desktop environment.

**Features**:

* Split-view mode for editing multiple documents simultaneously.
* Syntax highlighting and code folding.
* Integrated terminal.
* Plugin support for additional features.
* [Session management to save and restore opened files](https://itsfoss.com/command-line-text-editors-linux/)

1. **Sublime Text:** Sublime Text is a cross-platform text editor known for its speed and features

**Features:**

* Goto anything (quickly navigate to files, lines, or symbols)
* Multiple selections
* Command palette
* Distraction-free mode
* Split editing

1. **Atom:** Atom is an open-source text editor developed by GitHub. It’s known for its hackability and customization.

* **Features**:
  + Built-in package manager for installing plugins.
  + Smart autocompletion.
  + File system browser.
  + Multiple panes for split editing.
  + [Themes and customization option](https://linuxsimply.com/linux-basics/text-editors/comparison/)
* **Deleting, Copying, Moving, and Renaming Files**

In Linux, you can perform file operations such as deleting, copying, moving, and renaming using command-line tools.

* **Deleting Files and Directories:** Using rm Command

Delete a file: rm filename

Delete multiple files: rm file1 file2 file3

Delete a directory and its contents: rm -r directoryname

* **Copying Files and Directories:** Using cp Command

Copy a file: cp sourcefile destinationfile

Copy multiple files to a directory: cp file1 file2 file3 /path/to/destination/

Copy a directory and its contents: cp -r sourcedirectory destinationdirectory

* **Moving and Renaming Files and Directories:** Using mv Command

Move a file: mv sourcefile destinationfile

Move multiple files to a directory: mv file1 file2 file3 /path/to/destination/

Move a directory and its contents: mv sourcedirectory destinationdirectory

Rename a file: mv oldfilename newfilename

Rename a directory: mv olddirectoryname newdirectoryname

* **Compressing Files**

**Gzip:** gzip is a command-line tool for compressing files. It reduces the size of files, making them easier to store and transfer.

Compress a file: gzip filename

Decompress a file: gunzip filename.gz

View the contents of a compressed file without decompressing: zcat filename.gz

* **Archiving Files**

Archive files are like a big box where you can store many files together. This makes it easier to move or save them.

**Tar:** The tar command in Linux helps you create and manage these archive files. Think of it as a tool to pack and unpack your files. tar (short for tape archive) is used to create, maintain, modify, and extract files from an archive file commonly referred to as a tarball.

Create a tar archive: tar -cvf archive.tar file1 file2 file3

Extract a tar archive: tar -xvf archive.tar

View the contents of a tar archive: tar -tvf archive.tar

To append file: tar -rvf file1 example.tar

* **Disk Usage : Say Df and not Du**

Du: Estimates file usage

Du -k: Display sizes in kilobytes

Du -h: display sizes in human readable format

* **Wildcards**

Wildcards in Linux are special symbols that help you select groups of files or directories without typing out every name. They are particularly useful when combined with commands like ls, cp, mv, rm, and others

**Common Wildcards**

1. **Asterisk (\*)**: Matches any number of characters (including none).

Example: ls \*.txt lists all files with a .txt extension.

1. **Question Mark (?)**: Matches exactly one character.

Example: ls file?.txt matches file1.txt, file2.txt, etc., but not file10.txt.

1. **Square Brackets ([])**: Matches any one of the characters inside the brackets.

Example: ls file[123].txt matches file1.txt, file2.txt, and file3.txt.

1. **Square Brackets with Negation ([!])**: Matches any one character except those specified.

Example: ls file[!123].txt matches files like file4.txt, filea.txt, but not file1.txt, file2.txt, or file3.txt.

1. **Square Brackets with a Hyphen ([a-z])**: Matches any one character within the specified range.

Example: ls file[a-c].txt matches filea.txt, fileb.txt, and filec.txt.

1. **Named Character Class:**

* [[:alpha:]]: matches alphabetic characters
* [[:alnum:]]: matches alphanumeric characters
* [[:digit:]]: matches decimal between 0-9
* [[:lower:]] and [[:upper:]]: matches lowercase and uppercase letters respectively

1. Escape character (\): Use if you want to match a wildcard character. Example: \*\?: matches all files that end with a question mark

* **Input-output and redirection**

In Linux, input/output (I/O) redirection allows you to control the flow of data to and from the standard input (stdin), standard output (stdout), and standard error (stderr). This is crucial for efficiently managing command-line operations.

* **Standard Input (stdin):** Standard Input is the default source of input for commands. By default, it is the keyboard, but it can be redirected from a file or another command. **File Descriptor: 0**

Example:

#!/bin/bash

while read line

do

echo "Line: $line"

done < input.txt

This script reads lines from input.txt and prints them.

* **Standard Output (stdout):** Standard Output is the default destination for output from commands. By default, it is the terminal, but it can be redirected to a file or another command. **File Descriptor: 1**

Example:

echo "Hello, World!" > output.txt

This command writes “Hello, World!” to output.txt instead of the terminal.

* **Standard Error (stderr):** Standard Error is the default destination for error messages. By default, it is the terminal, but it can be redirected separately from stdout. **File Descriptor: 2**

Example:

ls non\_existent\_directory 2> error.txt

This command writes the error message to error.txt instead of the terminal.

**Redirection Operators**

1. **>**: Redirects stdout to a file, overwriting the file if it exists.

Syntax: command > file

1. **>>**: Redirects stdout to a file, appending to the file if it exists.

Syntax: command >> file

1. 2>: Redirects stderr to a file.

Syntax: command 2> file

1. 2>>: Redirects stderr to a file, appending to the file if it exists.

Syntax: command 2>> file

1. &>: Redirects both stdout and stderr to a file.

Syntax: command &> file

1. &>>: Redirects both stdout and stderr to a file, appending to the file if it exists.

Syntax: command &>> file

1. **<**: Redirects stdin from a file.

Syntax: command < file

1. **<<**: Here document, redirects stdin from a string.

Syntax: command << EOF

text

EOF

* **Comparing files in Linux**

In Linux, comparing files can be done using various command-line tools, each with its own set of features.

* **Diff:** diff is the most commonly used command for comparing the contents of two files line by line

Compare two files: diff file1 file2

Side-by-side comparison: diff -y file1 file2

* **Sdiff:** sdiff displays the differences between two files side by side and allows you to interactively merge them.

sdiff file1 file2

* **Vimdiff**: vimdiff will use vim editor to display the difference between two files
* **Searching in files and using pipes**

In Linux, searching for text within files and using pipes to connect the output of one command to the input of another are powerful techniques for efficiently managing data.

### **Searching in Files**

Grep Command: The grep command in Linux is used to search for specific patterns within files. It’s like a powerful search tool that helps you find text in files quickly.

**Syntax**: grep [options] pattern [file...]

**Example**: grep "hello" file.txt

This command searches for the word “hello” in file.txt.

**Common Options**

1. **-i (ignore case)**: Mwhakes the search case-insensitive.

**Example**: grep -i "hello" file.txt will match “hello”, “Hello”, “HELLO”, etc.

1. **-r (recursive)**: Searches directories and subdirectories.

**Example**: grep -r "hello" /path/to/directory searches for “hello” in all files within the specified directory and its subdirectories.

1. **-v (invert match)**: Shows lines that do not match the pattern.

**Example**: grep -v "hello" file.txt displays all lines in file.txt that do not contain “hello”.

1. **-n (line number)**: Shows the line numbers of matching lines.

**Example**: grep -n "hello" file.txt displays the line numbers where “hello” appears in file.txt.

* **Pipes:** In Linux, pipes (|) are used to connect the output of one command directly into the input of another command. This allows you to combine multiple commands to perform complex tasks efficiently. Syntax: command1 | command2

**Examples**

* List all files and search for a pattern within the list: ls -l | grep "pattern"
* Combine cat and grep to search within multiple files: cat file1 file2 | grep "pattern"
* **Cut:** The cut command in Linux is used to extract sections from each line of input, typically from files or standard input. It's commonly used for extracting columns of data from text files, such as CSV files or log files. Syntax: cut [OPTION]... [FILE]...

**Options**

**1. -b (bytes):** Select only the specified bytes. Useful when dealing with fixed-width data.

Syntax: cut -b LIST [FILE]...

**2. -c (characters):** Select only the specified characters. Syntax: cut -c LIST [FILE]...

**3. -f (fields):** Select only the specified fields. Requires -d to specify the delimiter. Syntax: cut -f LIST -d DELIM [FILE]...

* **Transferring and Copying Files over the Network**

Transferring and copying files over the network in Linux can be accomplished using various tools and protocols. Use scp and sftp over ftp, as ftp is less secure.

* 1. **scp (Secure Copy):** scp (Secure Copy) is a command-line utility that allows you to securely copy files and directories between hosts over a network. Syntax: scp [options] source destination.

Example: scp localfile.txt user@remotehost:/path/to/destination/

* 1. **sftp (SSH File Transfer Protocol):** sftp is an interactive file transfer program similar to ftp, but it uses a secure SSH connection. **Syntax:** sftp [options] user@remotehost

Examples: Upload a file: sftp> put /local/path/localfile.txt /remote/path/remotefile.txt

* 1. **ftp (File Transfer Protocol):** ftp is an older protocol for transferring files, typically used with anonymous access or simple authentication. Syntax: ftp [options] [hostname] Example: Connect to an FTP server: ftp ftp.example.com
* **Customizing the Shell Prompt**

Customizing the shell prompt in Linux can make your command line experience more informative and visually appealing. Customizing the shell prompt in Linux involves modifying the **PS1** variable, which defines the appearance of the command prompt.

**PS1:** PS1 is an environment variable that controls the primary prompt string in the shell. By changing its value, you can customize how your command prompt looks. It can include various placeholders for different information, such as the username, hostname, current directory, time, etc.

**Basic Customization**

To customize the PS1 variable, you can directly assign a new value to it. For example:

PS1="\u@\h:\w\$ "

This sets the prompt to display the username (\u), hostname (\h), and current working directory (\w), followed by a dollar sign ($).

**Common PS1 Placeholders**

* \u: Username
* \h: Hostname (short)
* \H: Hostname (full)
* \w: Current working directory
* \W: Basename of the current working directory
* \d: Date
* \t: Time (24-hour HH:MM)
* \T: Time (12-hour HH:MM)
* \A: Time (24-hour HH)
* \@: Time (12-hour with AM/PM)
* \n: Newline

Example 1: Simple Customization: PS1="\u@\h:\w\$ "

# Output: user@hostname:/current/directory$

Example 2: Adding Time and Date: PS1="\d \t \u@\h:\w\$ "

# Output: Tue Jul 27 14:22:35 user@hostname:/current/directory$

* **Shell aliases**

Shell aliases in Linux are a powerful feature that allows you to create shortcuts for long or complex commands. Aliases are particularly useful for frequently used commands, allowing you to save time and reduce the potential for errors.

* **Creating Aliases:** To create an alias, use the alias command followed by the name of the alias and the command it represents.

**Basic Syntax:** alias alias\_name='command'

**Example:** Creating a shortcut for ls -la:alias ll='ls -la'

* **Removing aliases**

Unalias name: Remove the “name” alias

Unalias -a: Remove all aliases

* **Environment variable**

Environment variables in Linux are key-value pairs that are used to configure the behavior of the system and applications.

SYNTAX: export VAR= “Value”

Example: export Editor= “vi”

**Common Environment Variables**

* **HOME**: The home directory of the current user. Syntax: printenv HOME
* **USER**: The name of the current user. Syntax: printenv USER
* **PATH**: A colon-separated list of directories that the shell searches for executable files.
* **SHELL**: The path to the current user's shell.
* **LANG**: The current locale setting, which determines the language and character encoding.
* **PWD**: The current working directory.
* **OLDPWD**: shows the previous directory where we were
* **EDITOR**: The default text editor.
* **TERM**: The type of terminal to emulate when running the shell.
* **Processes and job control**

In Linux, managing processes and job control is essential for efficiently handling tasks and maintaining system performance

* **Process:** A process is an instance of a running program. Each process in Linux is assigned a unique Process ID (PID). Processes can be in various states such as running, sleeping, stopped, or zombie.

**ps**: Display the current running processes. Options:

* -e: Everything, all processes
* -f: full format listing
* -p pid: Display info of PID
* Pstree: Display processes in a tree format
* **Job Control:** Job control allows you to manage multiple tasks within a single shell session. Jobs can be paused, backgrounded, or foregrounded.

**Basic Job Control Commands**

* **&:** To start a background process place an ampersand (&) at the end of the command.
* **jobs**: List active jobs.
* **fg**: Bring a background job to the foreground.
* **bg**: Resume a stopped job in the background.
* **Ctrl -c:** kill the foreground command
* **Ctrl -z**: Suspend the foreground process
* **kill**: Send a signal to a process, usually to terminate it. SYNTAX: kill PID
* **Scheduling Repeated Jobs with Cron**

Cron and crontab are tools in Linux used for scheduling tasks to run automatically at specified times and intervals. Here’s a simple breakdown:

**Cron: Cron** is a tool in Linux that allows you to schedule tasks to run automatically at specific times or intervals. These tasks are called “cron jobs.” It’s like setting an alarm clock for your computer to do certain things at certain times.

**Crontab: Crontab** (short for “cron table”) is a file where you define the schedule and commands for your cron jobs. Each user on the system can have their own crontab file.

**Basic Syntax of a**crontab**Entry**

A typical crontab entry has five time-and-date fields followed by the command to be executed:

\* \* \* \* \* command\_to\_execute

| | | | |

| | | | +---- Day of the week (0 - 7) (Sunday is both 0 and 7)

| | | +------ Month (1 - 12)

| | +-------- Day of the month (1 - 31)

| +---------- Hour (0 - 23)

+------------ Minute (0 - 59)

**Examples**

1. **Run a script every day at 2:30 AM**

30 2 \* \* \* /path/to/script.sh

1. **Run a command every Monday at 5:00 PM**

0 17 \* \* 1 /path/to/command

1. **Run a task every hour**

0 \* \* \* \* /path/to/task

* **Switching Users and Running Commands as Others**

Switching users and running commands as other users in Linux are common tasks for system administration and security management

* **Switching Users**

Using su Command: The su (substitute user) command allows you to switch to another user account within your current session.

**Switch to another user**

**su username**: You will be prompted to enter the password for the specified user. Once authenticated, your shell will switch to the new user.

**Switch to the root user: su**

* **Using sudo Command:** The sudo (superuser do) command allows permitted users to execute a command as the superuser or another user, as specified by the security policy.

**Sudo -l:**  List available commands

Sudo command: Run command as root

Sudo -u root: Same as above

Sudo -u user: Run as user

* **Shell History and Tab Completion**

Shell history and tab completion are powerful features in Linux that enhance productivity and efficiency when working in the terminal.

* **Shell History:** Shell History keeps a record of the commands you’ve entered in the terminal. This allows you to recall and reuse previous commands without retyping them.
* **Viewing Shell History**

View entire history: history

View a specific number of past commands: history n

* **Reusing Commands from History**

Execute a specific command from history using its number: !n

Execute the last command: !!

* **Searching Command History**

Interactive search through history: Press Ctrl + r and start typing part of the command. It will search backward through your history.

* **Tab Completion:** Tab completion allows you to quickly complete commands, filenames, directory names, and other elements by pressing the Tab key.

Complete a command: Start typing a command and press Tab to complete it. Syntax:- ls[TAB]

Complete a filename or directory: Start typing a filename or directory name and press Tab to complete it. Syntax:- cd /usr/local[TAB]

* **Software on RPM Based Linux Distros: RedHat, CentOS, AlmaLinux, Rocky, Debian, ubuntu, kali linux**
* **RPM-Based Distros: RPM** stands for **Red Hat Package Manager**. These distros use .rpm files to manage software.

1. **Red Hat Enterprise Linux (RHEL)**:
   * **Target Audience**: Enterprises.
   * **Package Manager**: yum (Yellowdog Updater, Modified) and dnf (Dandified YUM).
   * **Features**: Stable, secure, and supported by Red Hat.
2. **CentOS**:
   * **Target Audience**: Community and enterprises.
   * **Package Manager**: yum and dnf.
   * **Features**: Free and open-source, derived from RHEL.
3. **AlmaLinux**:
   * **Target Audience**: Community and enterprises.
   * **Package Manager**: yum and dnf.
   * **Features**: Free and open-source, a direct replacement for CentOS.
4. **Rocky Linux**:
   * **Target Audience**: Community and enterprises.
   * **Package Manager**: yum and dnf.
   * **Features**: Free and open-source, another replacement for CentOS.

* **DEB-Based Distros: DEB** stands for **Debian Package**. These distros use .deb files to manage software.

1. **Debian**:
   * **Target Audience**: General users and developers.
   * **Package Manager**: apt (Advanced Package Tool).
   * **Features**: Very stable, large repository of software.
2. **Ubuntu**:
   * **Target Audience**: General users, developers, and enterprises.
   * **Package Manager**: apt.
   * **Features**: User-friendly, frequent updates, large community support.
3. **Kali Linux**:
   * **Target Audience**: Security professionals and ethical hackers.
   * **Package Manager**: apt.
   * **Features**: Pre-installed with numerous security tools, based on Debian.

**Key Differences**

* **Package Management**: RPM-based distros use yum/dnf, while DEB-based distros use apt/dpkg.
* **Target Audience**: Varies from general users to enterprises and specialized fields like security.
* **Stability and Support**: RHEL and its derivatives focus on enterprise stability, while Debian and Ubuntu offer a balance of stability and cutting-edge features.

* **Package and Package Manager**
* **Package**: A **package** is a bundle of files that includes everything needed to install and run a piece of software. This can include:
* The software itself (like a web browser or a text editor).
* Configuration files.
* Documentation.
* Dependencies (other software that the main software needs to work).
* **Package Manager:** A **package manager** is a tool that helps you manage these packages. It makes it easy to:
* **Install** new software.
* **Update** existing software.
* **Remove** software you no longer need.
* **Handle dependencies** automatically.

Different Linux distributions use different package managers, each with its own commands and file formats. The two main types of package managers: dpkg for Debian-based systems and rpm for Red Hat-based systems.

* **Dpkg:** dpkg is the low-level package manager for Debian-based systems like Debian, Ubuntu, and their derivatives. It handles .deb files.

**Installing a package:** sudo dpkg -i package.deb

**Removing a package**: sudo dpkg -r package\_name

**APT**: APT (Advanced Package Tool) is a higher-level tool that uses dpkg under the hood. It handles package installation, updates, and dependency resolution more efficiently.

**Updating the package list**: sudo apt update

**Upgrading installed packages:** sudo apt upgrade

* **Rpm**: rpm (Red Hat Package Manager) is the low-level package manager for Red Hat-based systems like Red Hat Enterprise Linux (RHEL), Fedora, and CentOS. It handles .rpm files.

**Installing a package:** sudo rpm -i package.rpm

**Removing a package:** sudo rpm -e package\_name

* **YUM:** YUM (Yellowdog Updater, Modified) is a higher-level tool that uses rpm under the hood, handling package installation, updates, and dependency resolution. Updating the package list: sudo yum check-update

**Upgrading installed packages:** sudo yum update

**DNF**: DNF (Dandified YUM) is the next-generation version of YUM, used in Fedora and newer versions of CentOS and RHEL.

* **The linux Boot Process (The process when you press start button of the machine)**

The Linux boot process is a sequence of events that takes place when a Linux system is powered on or rebooted. This process includes several stages, each with its own critical function to bring the system to an operational state.

**1. BIOS (Basic Input Output System)**

* **What It Is**: The BIOS is a special type of firmware that runs first when you power on your computer.
* **Purpose**: It performs basic hardware checks and then loads the boot loader or operating system.
* **POST (Power-On Self-Test)**: The BIOS checks the CPU, memory, and storage devices to ensure they are working properly.
* **Boot Devices**: The BIOS looks for a bootable device (like a hard disk, USB drive, or DVD) in a specified order. You can change this order in the BIOS settings.

**2. Boot Loader**

* **What It Is**: The boot loader is a small program that loads the operating system.
* **Common Boot Loaders**:
* **GRUB (Grand Unified Bootloader)**: The most common bootloader used in Linux systems. GRUB loads the kernel into memory and hands over control to it.
  + **LILO (Linux Loader)**: Older boot loader, less common now.
* **Purpose**: The boot loader starts the operating system and can allow you to choose between multiple operating systems if you have more than one installed.

**3. Initial RAM Disk (initrd)**

* **What It Is**: A temporary file system loaded into memory during the boot process.
* **Purpose**: It contains necessary drivers and modules to help mount the actual root file system.
* **Example**: If your root file system is on an LVM (Logical Volume Manager) volume, initrd will have the necessary modules to mount it.

**4. Linux Kernel**

* **What It Is**: The core part of the Linux operating system.
* **Location**: Stored in the **/boot** directory, typically named vmlinux or vmlinuz (if compressed).
* **Purpose**: The kernel manages the system’s resources and allows software to interact with the hardware.

**5. Kernel Ring Buffer**

* **What It Is**: A data structure that stores messages from the kernel.
* **Viewing Messages**: Use the dmesg command or check the /var/log/dmesg file to see these messages.

**6. Runlevels and Targets**

* **Runlevels**: Different states of the system, each with a specific set of services and processes running.
  + **Runlevel 0**: Power off.
  + **Runlevel 1**: Single-user mode.
  + **Runlevels 2-5**: Multi-user modes with different configurations.
  + **Runlevel 6**: Reboot.
* **Systemd Targets**: Modern systems use systemd, which replaces runlevels with targets.
  + **Example**: graphical.target is equivalent to runlevel 5.
  + **Changing Targets**: Use the systemctl command (e.g., systemctl isolate graphical.target).

**7. Shutdown and Reboot Commands**

* **Shutdown**:
  + **Command**: shutdown -h now (to power off immediately).
* **Reboot**:
  + **Command**: reboot or shutdown -r now (to reboot immediately).

**Summary**

* **BIOS**: Performs hardware checks and loads the boot loader.
* **Boot Loader**: Starts the operating system.
* **initrd**: Loads necessary modules to mount the root file system.
* **Kernel**: Manages system resources.
* **Runlevels/Targets**: Define system states and services.
* **Shutdown/Reboot**: Commands to power off or restart the system.
* **System Logging**

System logging in Linux is a critical component for monitoring and troubleshooting system activities. It provides a way to collect, store, and analyze log messages generated by the kernel, system services, and user applications.

**rsyslog**: An enhanced version of the original syslog daemon with advanced features like logging to databases and filtering capabilities.

**Log Files**

Located in /var/log/.

Common log files:

/var/log/messages: General system log messages.

/var/log/syslog: General system log messages (Debian-based systems).

/var/log/auth.log: Authentication and authorization logs.

/var/log/kern.log: Kernel logs.

/var/log/daemon.log: Daemon logs.

**Log Rotation**

Logrotate: logrotate is a utility for managing log files, including rotation, compression, and removal.

* **Disk Management: Main storage of linux**
* **Partitions:** When you partition a disk, you divide it into separate sections, each called a partition. This allows you to allocate different parts of the disk for different purposes. For example, you might have one partition for the operating system, another for applications, a third for user data, and a fourth for swap space. This separation helps in managing the system more efficiently and prevents one part from affecting another adversely.
* **MBR (Master Boot Record):** The MBR is a special type of boot sector located at the beginning of a storage device. It contains the partition table, which holds information about how the disk is divided into partitions. However, MBR has limitations:
* It can only address up to 2 TB of storage.
* It supports up to four primary partitions. If you need more, you must create an extended partition, which can contain multiple logical partitions.
* **GPT (GUID Partition Table):** GPT is a newer partitioning scheme that overcomes the limitations of MBR:
* It supports disks larger than 9.2 TB.
* It allows for up to 128 partitions by default.
* It is part of the UEFI standard, which is replacing the traditional BIOS.
* GPT is not supported by older operating systems and requires newer partitioning tools.
* **Mount Points:** A mount point is a directory where a partition is attached to the file system. For example, if you have a partition for user home directories, it might be mounted at /home. This means all files and directories under /home reside on that partition. You can mount partitions anywhere in the directory tree, and you can even mount one partition within another.
* **fdisk Utility:** fdisk is a command-line utility used to create and manage disk partitions. Here’s how you can use it:

1. **Open fdisk**: Run sudo fdisk /dev/sdX (replace /dev/sdX with your disk’s identifier).
2. **View Partitions**: Type p to print the partition table.
3. **Create a New Partition**: Type n and follow the prompts to create a new partition.
4. **Write Changes**: Type w to write the changes to the disk.

**Example of Creating a Partition with fdisk**

1. Open a terminal and run sudo fdisk /dev/sdX.
2. Type n to create a new partition.
3. Choose p for a primary partition or e for an extended partition.
4. Enter the partition number, starting sector, and ending sector.
5. Type w to write the changes.

* **File Systems (Not Clear)**
* **Creating a File System:** Before a partition can be used, it needs a file system. The most common file systems in Linux are from the extended file system family (ext2, ext3, ext4). Other popular file systems include ReiserFS, JFS, XFS, ZFS, and Btrfs.

To create a file system, you use the mkfs command. The general format is:

mkfs -t TYPE DEVICE

* **TYPE**: The type of file system (e.g., ext3, xfs).
* **DEVICE**: The path to the partition (e.g., /dev/sdb2).

For example, to create an ext3 file system on the second partition of the sdb disk:

mkfs -t ext3 /dev/sdb2

* **Mounting a File System:** Mounting is like connecting a partition to a folder so you can use it. Use the mount command to do this. For example:

mount /dev/sdb3 /opt

This connects the partition /dev/sdb3 to the /opt folder.

* **Making Mounts Permanent:** If you want the partition to stay connected even after you restart your computer, you need to add it to a special file called /etc/fstab. This file tells the system which partitions to mount and where to mount them.
* **Unmounting a File System:** To disconnect a partition, use the umount command. For example:

umount /opt

or

umount /dev/sdb3

* **Preparing Swap Space:** Swap space is extra memory that your system can use when it runs out of RAM. To set up swap space, use the mkswap command:

mkswap /dev/sdb4

Then, to start using it, type:

swapon /dev/sdb4

* **Using the /etc/fstab File:** The /etc/fstab file controls where and how partitions are mounted. Each line in this file has six parts:

1. **Device**: The partition you want to mount.
2. **Mount Point**: The folder where you want to mount it.
3. **File System Type**: The type of file system (e.g., ext4).
4. **Options**: Special settings for mounting.
5. **Dump**: Whether to back up the partition (usually 0 for no).
6. **Fsck**: Whether to check the partition for errors at startup (0 for no, 1 for yes).

* **Disk UUIDs and Labels:** UUIDs and labels are unique identifiers for partitions. They help the system find the right partition even if the device name changes. To see these identifiers, use:

lsblk -f

or

blkid

To add a label to a partition, use the e2label command:

e2label /dev/sdb3 opt

* **Logical Volume Manager**

The Logical Volume Manager (LVM) in Linux is a powerful tool for managing disk storage. It provides a more flexible and advanced way to manage disk space than traditional partitioning schemes. LVM allows you to create, resize, and delete logical volumes without worrying about the physical layout of the underlying hardware.

**Layers of Abstraction in LVM:** LVM uses three main layers of abstraction:

**1. Physical Volumes (PVs):** Physical Volumes are the actual physical storage devices, such as hard drives or SSDs. These are the raw building blocks that LVM uses. You can convert a physical storage device into a PV using the pvcreate command.

Example: pvcreate /dev/sda1

This command initializes /dev/sda1 as a physical volume.

**2. Volume Groups (VGs)**

Volume Groups are created by combining multiple Physical Volumes. They act as a pool of storage from which Logical Volumes can be allocated. You can create a Volume Group using the vgcreate command.

Example: vgcreate my\_volume\_group /dev/sda1 /dev/sdb1

This command creates a Volume Group named my\_volume\_group using the physical volumes /dev/sda1 and /dev/sdb1.

**3. Logical Volumes (LVs):** Logical Volumes are created from the storage pool provided by a Volume Group. They are similar to traditional partitions but offer more flexibility. You can create a Logical Volume using the lvcreate command.

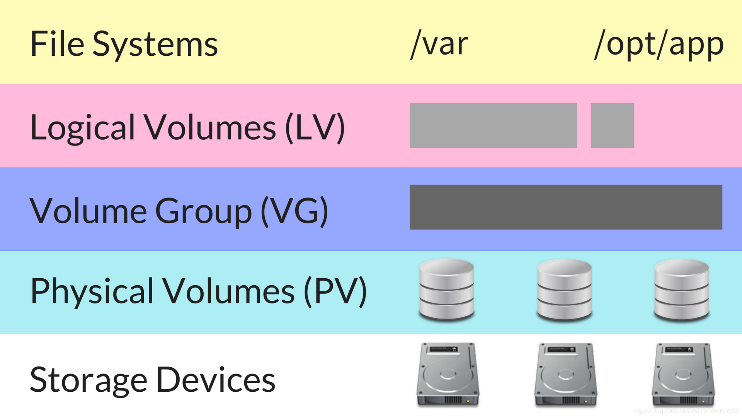
Example: lvcreate -L 10G -n my\_logical\_volume my\_volume\_group

This command creates a Logical Volume named my\_logical\_volume with a size of 10 GB from the my\_volume\_group Volume Group.

**Logical Extent (LE):** A small, fixed-size chunk of space within a Logical Volume.LVM allocates storage in units of Logical Extents.

**Physical Extent (PE):** The counterpart of Logical Extents on Physical Volumes.LVM maps Logical Extents to Physical Extents on the Physical Volumes.

**LVM: Layers of Abstraction**



**Example Workflow**

Let’s go through a complete example to illustrate how LVM works:

* Initialize Physical Volumes: pvcreate /dev/sda1 /dev/sdb1
* Create a Volume Group: vgcreate my\_volume\_group /dev/sda1 /dev/sdb1
* Create a Logical Volume: lvcreate -L 20G -n my\_logical\_volume my\_volume\_group
* Create a File System on the Logical Volume: mkfs.ext4/dev/my\_volume\_group/my\_logical\_volume
* Mount the Logical Volume: mount /dev/my\_volume\_group/my\_logical\_volume /mnt
* **Mirroring Logical Volumes**

Mirroring Logical Volumes (LVs) in Linux using the Logical Volume Manager (LVM) involves creating a Logical Volume that stores data redundantly across multiple Physical Volumes (PVs). This process enhances data availability and reliability, as it protects against data loss due to the failure of one or more Physical Volumes.

**Steps to Create a Mirrored Logical Volume**

1. Ensure LVM2 and Mirroring Support is Installed

2. Create Physical Volumes (PVs)

3. Create a Volume Group (VG)

4. Create a Mirrored Logical Volume (LV)

5. Format the Logical Volume

6. Create a Mount Point and Mount the Logical Volume

7. Verify the Mount

* **Removing Logical Volumes, Physical Volumes, and Volume Groups**

Removing Logical Volumes (LVs), Volume Groups (VGs), and Physical Volumes (PVs) in Linux using the Logical Volume Manager (LVM) involves a series of steps to ensure that the volumes are properly unmounted and removed without data loss or system errors.

Steps:-

1. Unmount the Logical Volume:- Before you can remove a Logical Volume, you need to unmount it if it is currently mounted.

2. Remove the Logical Volume:- Use the lvremove command to remove the Logical Volume.

3. Remove the Volume Group:- After removing all Logical Volumes within the Volume Group, you can remove the Volume Group.

4. Remove the Physical Volumes:- Finally, you can remove the Physical Volumes that were part of the Volume Group

* **Managing Users and Groups**
* **/etc/passwd**: This file stores essential information about user accounts. Each line represents a user and contains fields separated by colons (:). The fields include:
  + **Username**: The user’s login name.
  + **Password**: An x indicates the password is stored in /etc/shadow.
  + **UID (User ID)**: A unique number assigned to each user.
  + **GID (Group ID)**: The default group ID for the user.
  + **Comments**: Typically the user’s full name or other descriptive information.
  + **Home Directory**: The directory assigned to the user.
  + **Shell**: The program that runs when the user logs in.

**2. /etc/shadow File:** This file contains encrypted password information and is only readable by the superuser. It includes fields such as:

* + **Username**: The user’s login name.
  + **Encrypted Password**: The user’s password in encrypted form.
  + **Password Change Information**: Dates and intervals related to password changes and expirations.

**3. Creating and Managing Users**

* **useradd Command**: Used to create new user accounts. Common options include:
  + -c: Adds a comment.
  + -m: Creates a home directory.
  + -s: Specifies the user’s shell.
  + -g: Sets the default group.
  + -G: Adds the user to additional groups.

Example:

useradd -c "Grant Stewart" -m -s /bin/bash grant

1. **Creating and Managing Groups**

* **Creating a Group:** To create a new group, use the groupadd command. Syntax:- sudo groupadd groupname
* **Modifying a Group:** To modify an existing group, use the groupmod command.

Change the group’s name: sudo groupmod -n newgroupname oldgroupname

* **Deleting a Group:** To delete a group, use the groupdel command. Syntax:- sudo groupdel groupname
* **TCP/IP Networking for Linux System Administrators**

TCP/IP (Transmission Control Protocol/Internet Protocol) is the suite of communication protocols used to connect hosts on the Internet. TCP/IP dictates how data should be packetized, addressed, transmitted, routed, and received.

1. **TCP/IP Protocol Suite**:

* TCP (Transmission Control Protocol): Provides reliable, ordered, and error-checked delivery of a stream of data between applications running on hosts communicating via an IP network.
* IP (Internet Protocol): Deals with addressing and routing the data packets so they can travel across networks and arrive at the correct destination.
* **Classful Networks:**

Classful networking divides the IP address space into five classes (A, B, C, D, E). This system was used in the early days of networking but has largely been replaced by CIDR (Classless Inter-Domain Routing).

**Class A(0-127):** Range: 1.0.0.0 to 126.255.255.255**,** Default Subnet Mask: 255.0.0.0**,** Supports 16 million hosts on each of 128 networks.

**Class B(128-191)**: Range: 128.0.0.0 to 191.255.255.255, Default Subnet Mask: 255.255.0.0, Supports 65,000 hosts on each of 16,000 networks.

**Class C(192-223):** Range: 192.0.0.0 to 223.255.255.255, Default Subnet Mask: 255.255.255.0, Supports 254 hosts on each of 2 million networks.

**Class D(224-239):** Range: 224.0.0.0 to 239.255.255.255**,** Used for multicast.

**Class E(240-255):** Range: 240.0.0.0 to 255.255.255.255**.** Reserved for future use, or research and development purposes.

* **Subnet Masks**: A subnet mask is used to divide an IP address into network and host parts. It masks the IP address to indicate which bits are used for the network and which are used for the host.

**Example:**

IP Address: 192.168.1.10

Subnet Mask: 255.255.255.0

Here, the first three octets (192.168.1) represent the network, and the last octet (.10) represents the host.

4. **Broadcast Addresses:** A broadcast address allows information to be sent to all devices on a network. It is the highest address in a subnet.

Example:

Network Address: 192.168.1.0

Subnet Mask: 255.255.255.0

Broadcast Address: 192.168.1.255

* **CIDR (Classless Inter-Domain Routing)**: CIDR is a method for allocating IP addresses and IP routing. Unlike the classful network addressing, CIDR allows for more efficient use of IP address space.

CIDR Notation: Combines the IP address with a suffix that indicates the number of bits used for the network prefix.

Example: 192.168.1.0/24

IP Address: 192.168.1.0

Subnet Prefix: /24 (indicating that the first 24 bits are the network part)

* **Private Address Space**: Private IP addresses are reserved for use within private networks and are not routable on the internet. They are defined in three ranges:

Class A: 10.0.0.0 to 10.255.255.255

Class B: 172.16.0.0 to 172.31.255.255

Class C: 192.168.0.0 to 192.168.255.255

* **Networking - DNS and hostnames**

DNS (Domain Name System) and hostnames are essential components of network communication. They provide a way to map human-friendly names to machine-friendly IP addresses, enabling users and applications to connect to devices and services easily.

**1. DNS (Domain Name System):** DNS is a hierarchical and decentralized naming system for computers, services, or other resources connected to the Internet or a private network. It translates human-readable domain names to IP addresses.

* **How DNS Works**

1. **Domain Name**: When you type a domain name into your browser, it needs to be translated into an IP address.
2. **DNS Query**: Your computer sends a DNS query to a DNS server to get the IP address.
3. **DNS Server**: The DNS server looks up the IP address associated with the domain name and sends it back to your computer.
4. **Connecting**: Your computer uses the IP address to connect to the web server hosting the website.

* **DNS Components in Linux**

1. **Resolver**: The part of your system that handles DNS queries. It sends requests to DNS servers and processes the responses.
2. **DNS Server**: A server that stores DNS records and responds to DNS queries. Common DNS servers include BIND (Berkeley Internet Name Domain).

* **Configuring DNS in Linux:** DNS settings in Linux are usually configured in the /etc/resolv.conf file: sudo nano /etc/resolv.conf

Example content:

nameserver 8.8.8.8

nameserver 8.8.4.4

This configuration tells your system to use Google’s public DNS servers.

**2. Hostnames:** A hostname is a label assigned to a device on a network. It is used to identify the device in various forms of electronic communication such as within a local network or on the Internet.

**Setting the Hostname**

* **Temporary Change**: Changes the hostname for the current session (until reboot). Syntax: sudo hostname newhostname
* **Permanent Change**: Persist the hostname across reboots by editing configuration files. Syntax: sudo hostnamectl set-hostname newhostname.
* **Networking - DHCP, Dynamic and Static Addressing**
* **DHCP (Dynamic Host Configuration Protocol):** DHCP is a network management protocol used to dynamically assign IP addresses to devices on a network, allowing them to communicate with other IP networks.

**How DHCP Works**

1. **DHCP Discovery**: When a device connects to a network, it sends a DHCPDISCOVER broadcast message to find a DHCP server.
2. **DHCP Offer**: The DHCP server responds with a DHCPOFFER message, offering an IP address to the device.
3. **DHCP Request**: The device replies with a DHCPREQUEST message, requesting the offered IP address.
4. **DHCP Acknowledgment**: The DHCP server sends a DHCPACK message to confirm the lease, and the device configures itself with the provided IP address, subnet mask, gateway, and DNS server addresses.

**Configuring DHCP Client on a Linux System:** Most Linux distributions are configured as DHCP clients by default, but the configuration file is typically located at /etc/network/interfaces for Debian-based systems or /etc/sysconfig/network-scripts/ifcfg-eth0 for Red Hat-based systems.

* **Static IP Addressing:** Static IP addressing involves manually assigning an IP address to a device. This is useful for servers and devices that need a permanent IP address.

Example configuration:

DEVICE=eth0

BOOTPROTO=static

ONBOOT=yes

IPADDR=192.168.1.10

NETMASK=255.255.255.0

GATEWAY=192.168.1.1

DNS1=8.8.8.8

DNS2=8.8.4.4

* **Network Troubleshooting**

Network troubleshooting in Linux involves using various commands and tools to diagnose and resolve network connectivity issues. Network troubleshooting commands are:

* **Ping:** The ping command tests the reachability of a host on an IP network.

**Usage**: Sends packets to a specified address and waits for a response. ping google.com

ping 192.168.1.1

* **Ifconfig:** Shows details about your network interfaces (like IP address, MAC address).
* **Ip:** The ip command is a modern alternative to ifconfig.
* **Traceroute:** Trace the path packets take to a destination.

**Usage:** Shows each hop (router) the packets pass through to reach the destination

traceroute [www.google.com](http://www.google.com)

* **Netstat:** Display network connections, routing tables, and interface statistics.

**Usage**: Useful for checking open ports and active connections.

netstat -tuln

* **Ss:** Display socket statistics (more modern than netstat).

**Usage**: Provides detailed information about network connections.

ss -tuln

* **Tcpdump:** Capture and analyze network traffic.

**Usage**: Monitors packets on a network interface.

sudo tcpdump -i eth0

* **Special Permission Modes**

In addition to the standard file permissions (read, write, execute), Linux has special permission modes that can be set on files and directories to provide additional security and control. These special modes include the Setuid, Setgid, and Sticky Bit.

* 1. **Setuid (Set User ID):** When the Setuid bit is set on an executable file, it allows the file to be executed with the privileges of the file's owner, not the user who executes it. This is commonly used for programs that need elevated privileges to perform specific tasks.
* **Symbolic Representation**: s in the owner's execute position (e.g., rwsr-xr-x)
* **Octal Representation**: 4

**2. Setgid (Set Group ID):** When the Setgid bit is set on an executable file, it allows the file to be executed with the privileges of the file's group, not the user who executes it. When set on a directory, new files created within the directory inherit the group of the directory, not the primary group of the user who creates the file.

* **Symbolic Representation**: s in the group's execute position (e.g., rwxr-sr-x)
* **Octal Representation**: 2

**3. Sticky Bit:** When the Sticky Bit is set on a directory, it restricts deletion of files within the directory. Only the file's owner, the directory's owner, or the root user can delete or rename files within that directory. This is commonly used on directories like /tmp.

* **Symbolic Representation**: t in the others' execute position (e.g., rwxrwxrwt)
* **Octal Representation**: 1

**Shell Scripting**

* **Shebang: # - Sharp , ! – Bang, #! – Shebang**

A shebang (also known as a hashbang) is a character sequence at the beginning of a script file that indicates which interpreter should be used to execute the script. The shebang consists of the characters #! followed by the path to the interpreter. When you run the script, the operating system uses the specified interpreter to execute it.

Example: #!/bin/bash

Here’s a simple breakdown:

1. **#!**: This is the shebang itself. It tells the system that this file is a script and needs to be executed by an interpreter.
2. **/bin/bash**: This is the path to the interpreter. In this case, it’s pointing to the Bash shell. It could also point to other interpreters like Python (#!/usr/bin/python3) or Perl (#!/usr/bin/perl).

When you run the script, the system uses the shebang to find the right interpreter to execute the commands in the script.

* **Comments**

Comments in Linux, particularly in shell scripts, are used to include notes, explanations, or instructions that are ignored by the interpreter. They are meant to make the script easier to understand and maintain.

Example: single-line comment: # This is a comment

Multiline comment: <<comment

This is a

Multiline comment

Comment

* **Variables**
* Variables in Linux, especially within the context of shell scripting, are used to store data that can be referenced and manipulated throughout the script. They can hold strings, numbers, filenames, and other types of data
* Example:

#!/bin/bash

greeting="Hello, World!"

echo $greeting

* **Accessing Variable Values:** To access the value of a variable, prefix its name with a $.

echo $variable\_name

* **Special Variables (Important)**
* $0: The name of the script.
* $1, $2, ..., $9: The first, second, …, ninth arguments passed to the script.
* ${10}, ${11}, ...: The tenth, eleventh, … arguments (use curly braces for double-digit arguments).
* $#: The number of arguments passed to the script.
* $\*: All the arguments passed to the script as a single string.
* $@: All the arguments passed to the script as separate strings.
* $?: The exit status of the last command executed.(successfully executed or not)
* $$: The process ID (PID) of the current shell.
* $!: The PID of the last background command.
* $-: The current options set for the shell.
* $\_: The last argument of the previous command.

**Debugging in shell**

Debugging shell scripts is an essential skill for identifying and fixing errors in your code. Here are some common techniques to debug shell scripts in Linux:

1. **Using set Command**:
   * **Verbose Mode (-v)**: Displays each command before it is executed.

**set -v**

* + **Execution Trace (-x)**: Displays each command and its arguments as they are executed.

**set -x**

* + **No Execution (-n)**: Checks the script for syntax errors without executing it.

**set -n**

* **Arrays**

Arrays in shell scripting allow you to store multiple values in a single variable, making it easier to manage collections of related data. Syntax: my\_array=(value1 value2 value3)

### Accessing Array Elements: You can access individual elements of an array using the **${array\_name[index]}** syntax. Example:

### my\_array=("apple" "banana" "cherry")

### echo "First element: ${my\_array[0]}"

* **Modifying Array Elements**

my\_array=("apple" "banana" "cherry")

* my\_array[1]="blueberry"
* **Length of an Array:** To get the number of elements in an array, use ${#array\_name[@]}

my\_array=("apple" "banana" "cherry")

length=${#my\_array[@]}

echo "The array contains $length elements."

* **String**

String operations in Linux shell scripting involve manipulating and processing text data. Bash provides a variety of tools and techniques to work with strings effectively

* Assigning Strings to Variables: str="Hello, World!"
* **Accessing String Length:**

str="Hello, World!"

length=${#str}

echo "Length of the string is $length"

* **Concatenating Strings**

str1="Hello"

str2="World"

str3="$str1, $str2!"

echo $str3 # Output: Hello, World!

* **Extracting Substrings**

str="Hello, World!"

sub\_str=${str:7:5}

echo $sub\_str # Output: World

* **Replacing Substrings**

str="Hello, World!"

new\_str=${str/World/Universe}

echo $new\_str # Output: Hello, Universe!

* **User interaction**

User interaction in Linux shell scripting typically involves reading input from the user and providing feedback or prompts. This can be achieved through various commands and techniques to make scripts more interactive and user-friendly

* **Using the read Command:** The read command is used to capture input from the user.

#!/bin/bash

# Prompt the user for input

echo "Please enter your name:"

read name

# Display the input back to the user

echo "Hello, $name!"

* **Using -p Option with read:** The -p option allows you to provide a prompt directly with the read command. Example

#!/bin/bash

# Prompt the user for input with read -p

read -p "Please enter your age: " age

# Display the input back to the user

echo "You are $age years old."

* **Conditional Statements**

Conditional statements in Linux shell scripting allow you to make decisions based on conditions.

* **if Statement:** The if statement allows you to execute a block of code only if a specified condition is true. Example:

#!/bin/bash

# Check if a number is greater than 10

number=15

if [ $number -gt 10 ]; then

echo "The number is greater than 10."

fi

* **if-else Statement:** The if-else statement allows you to execute one block of code if the condition is true, and another block if it is false. Example

#!/bin/bash

# Check if a number is even or odd

number=7

if [ $((number % 2)) -eq 0 ]; then

echo "The number is even."

else

echo "The number is odd."

fi

* **if-elif-else Statement:** The if-elif-else statement allows you to check multiple conditions. Example:

#!/bin/bash

# Check the value of a variable

value="apple"

if [ "$value" == "apple" ]; then

echo "The value is apple."

elif [ "$value" == "banana" ]; then

echo "The value is banana."

else

echo "The value is not apple or banana."

Fi

* **Numeric Comparison**

-eq: Equal to

-ne: Not equal to

-lt: Less than

-le: Less than or equal to

-gt: Greater than

-ge: Greater than or equal to

* **case Statement:** The case statement is used to execute different blocks of code based on the value of a variable.

#!/bin/bash

# Check the value of a variable using case

fruit="banana"

case $fruit in

apple)

echo "The fruit is apple."

;;

banana)

echo "The fruit is banana."

;;

cherry)

echo "The fruit is cherry."

;;

\*)

echo "Unknown fruit."

;;

esac

* **Loops**

Loops in Linux shell scripting are used to execute a block of code repeatedly. The most common types of loops in shell scripting are for, while, and until.

* **for Loop:** The for loop iterates over a list of items and executes a block of code for each item.
* **Example: Iterating Over a List**

#!/bin/bash

# Iterate over a list of names

for name in Alice Bob Charlie; do

echo "Hello, $name!"

done

* **Example: Iterating Over a Range**

#!/bin/bash

# Iterate over a range of numbers

for i in {1..5}; do

echo "Number: $i"

done

* **While loop**: The while loop executes a block of code as long as a specified condition is true. IFS (Internal field separator) is used to separate files. Example: Simple while Loop

#!/bin/bash

# Initialize a counter

counter=1

# Execute the loop as long as the counter is less than or equal to 5

while [ $counter -le 5 ];

do

echo "Counter: $counter"

((counter++))

done

* **until Loop:** The until loop executes a block of code as long as a specified condition is false. Example: Simple until Loop

#!/bin/bash

# Initialize a counter

counter=1

# Execute the loop until the counter is greater than 5

until [ $counter -gt 5 ]; do

echo "Counter: $counter"

((counter++))

done

* **break Statement:** The break statement terminates the loop prematurely. Example:

#!/bin/bash

# Iterate over a range of numbers

for i in {1..10}; do

if [ $i -eq 5 ]; then

break

fi

echo "Number: $i"

done

* **Continue**: The continue statement skips the current iteration of the loop and proceeds to the next iteration. Example:

#!/bin/bash

# Iterate over a range of numbers

for i in {1..10}; do

if [ $i -eq 5 ]; then

continue

fi

echo "Number: $i"

done

* **Function**

Functions in Linux shell scripting allow you to encapsulate code into reusable blocks. They help in organizing code, avoiding redundancy, and improving readability. You can define a function, call it multiple times within a script, and even pass arguments to it.

* **Defining and Calling Functions:**

#!/bin/bash

# Define a function

greet() {

echo "Hello, World!"

}

# Call the function

Greet

* **Functions with Arguments:** You can pass arguments to functions and access them using $1, $2, etc., where $1 is the first argument, $2 is the second argument, and so on.

Example

#!/bin/bash

# Define a function with arguments

greet() {

local name=$1

echo "Hello, $name!"

}

# Call the function with an argument

greet "Alice"

* **Creating accounts and username**

Creating user accounts in Linux is a fundamental task for system administrators. The useradd command is a powerful tool for this purpose.

Syntax: sudo useradd [options] username

**Common Options**

* -d /home/username: Specifies the home directory for the new user.
* -m: Creates the user’s home directory if it does not exist.
* -s /bin/bash: Sets the user’s login shell.
* -u 1001: Specifies the user ID (UID) for the new user.
* -g groupname: Specifies the primary group for the new user.
* -G group1,group2: Specifies supplementary groups for the new user.
* -e YYYY-MM-DD: Sets the account expiration date.
* -c "comment": Adds a comment or description for the user.

**Example Commands**

1. **Create a new user with default settings:**

sudo useradd newuser

1. **Create a new user with a specific home directory and login shell:**

sudo useradd -d /home/newuser -m -s /bin/bash newuser

1. **Create a new user with a specific UID and primary group:**

sudo useradd -u 1001 -g users newuser

1. **Create a new user with an account expiration date:**

sudo useradd -e 2024-12-31 newuser

1. **Add a comment for the new user:**

sudo useradd -c "This is a test user" newuser

**Setting the User Password:** After creating the user, you need to set a password for them:

sudo passwd newuser

* **Random Data and Cryptographic Hash Functions.**
* **Generating Random Data:** Generating random data can be useful for creating unique identifiers, passwords, or temporary files. Here are a few methods:

**Using**$RANDOM:

echo $RANDOM

* **Cryptographic Hash Functions:** Cryptographic hash functions are used to generate a fixed-size hash value from input data, which is useful for verifying data integrity.
* **MD5**: echo -n "your\_string" | md5sum
* **SHA-256**: echo -n "your\_string" | sha256sum
* **Example Script**

#!/bin/bash

# Generate a random password

password=$(openssl rand -base64 12)

echo "Generated Password: $password"

# Hash the password using SHA-256

hashed\_password=$(echo -n "$password" | sha256sum | awk '{print $1}')

echo "SHA-256 Hashed Password: $hashed\_password"

* **Positional Parameters, Arguments, Special Parameters**
* **Positional Parameters:** Positional parameters are the arguments passed to a script or function. They are accessed using $1, $2, $3, etc., where $1 is the first argument, $2 is the second, and so on. $0 contains the name of the script itself.

Example:

#!/bin/bash

echo "Script name: $0"

echo "First argument: $1"

echo "Second argument: $2"

**Special Parameters:** Special parameters are predefined variables in the shell that provide useful information:

* $\* and $@: All positional parameters.
* $#: Number of positional parameters.
* $$: Process ID of the current shell.
* $?: Exit status of the last command.
* $!: Process ID of the last background command.

Example:

#!/bin/bash

echo "All arguments: $\*"

echo "Number of arguments: $#"

echo "Process ID: $$"

echo "Last command exit status: $?"

* **Parsing command line options with getopts**

GetOpts: getopts is a tool in Linux that helps you handle options (like -a or -b) and their arguments (like -a value) in shell scripts. It makes it easier to process and use these options in your script.

**How to Use getopts**

1. **Basic Syntax**: getopts optstring name [args]
   * **optstring**: A string that lists the options your script can accept. If an option needs an argument, you add a colon (:) after it.
   * **name**: A variable that will hold the current option being processed.
   * **args**: Optional. The arguments to be parsed. If not provided, getopts uses the script’s arguments.

* **Deleting and Disabling Linux Accounts**
* **Deleting Linux Accounts:** To delete a user account in Linux, you can use the userdel command. This command removes the user from the system. If you want to delete the user’s home directory and mail spool, you can use the -r option.

Example:

#!/bin/bash

sudo userdel -r username

[This script deletes the user username along with their home directory](https://www.cyberciti.biz/faq/linux-remove-user-command/)

* **Disabling Linux Accounts:** To disable a user account, you can use the usermod command with the -L option to lock the account. This prevents the user from logging in.

Example:

#!/bin/bash

sudo usermod -L username

* **Cut and Awk**
* **cut Command:** The cut command is used to extract specific sections from each line of a file or input. It can work with bytes, characters, or fields.

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

**Common Options**

* -b, --bytes=LIST: Select only the bytes specified in LIST.
* -c, --characters=LIST: Select only the characters specified in LIST.
* -d, --delimiter=DELIM: Use DELIM as the field delimiter character instead of the tab character.
* -f, --fields=LIST: Select only the fields specified in LIST, separated by the delimiter character.
* --complement: Complement the selection, i.e., print the fields/characters not selected.
* --output-delimiter=STRING: Use STRING as the output delimiter.

**Examples**

1. **Extract Specific Bytes**: echo "hello world" | cut -b 1-5

Output: hello

1. **Extract Specific Characters**: echo "hello world" | cut -c 1,3,5

Output: hlo

1. **Extract Fields Using a Delimiter**: echo "name:age:city" | cut -d ':' -f 2

Output: age

* **awk Command:** The awk command is a powerful programming language for pattern scanning and processing. It is used for manipulating data and generating reports.

**Syntax:** awk [options] 'pattern {action}' [file(s)]

**Common Options**

* -F fs: Use fs for the input field separator.
* -f program-file: Read the awk program source from the file.

**Basic Structure**

* **Pattern**: Specifies the condition to match.
* **Action**: Specifies what to do when the pattern matches.

**Examples**

1. **Print All Lines**: awk '{print}' file.txt

This prints every line in file.txt.

1. **Print Specific Fields**: awk '{print $1, $3}' file.txt

This prints the first and third fields of each line in file.txt.

1. **Pattern Matching**: awk '/pattern/ {print $0}' file.txt

This prints lines that match the specified pattern.

* **Ulimit Command**

The ulimit command in Linux is used to control the resources available to the shell and the processes started by it. It helps in setting limits on system resources to prevent any single process from consuming too much and potentially destabilizing the system.

**Key Concepts**

* **Soft Limits**: These are the current limits enforced by the kernel. They can be increased up to the hard limit by any user.
* **Hard Limits**: These are the maximum limits that can be set for a resource. Only the root user can increase these limits.

**Basic Syntax:** ulimit [options] [limit]

* **Options**: Specify which resource limit to set or display.
* **Limit**: The value you want to set for the specified resource.

**Common Options**

* **-a**: Display all current limits.
* **-c**: Set the core file size limit.
* **-d**: Set the data segment size limit.
* **-f**: Set the file size limit.
* **-n**: Set the maximum number of open file descriptors.
* **-u**: Set the maximum number of user processes.
* **-v**: Set the virtual memory limit.

**Examples**

* **Display All Current Limits:** To see all the current limits set for the shell, use:ulimit -a

This will output something like:

core file size (blocks, -c) 0

data seg size (kbytes, -d) unlimited

file size (blocks, -f) unlimited

max locked memory (kbytes, -l) 64

max memory size (kbytes, -m) unlimited

open files (-n) 1024

pipe size (512 bytes, -p) 8

stack size (kbytes, -s) 8192

cpu time (seconds, -t) unlimited

max user processes (-u) 4096

virtual memory (kbytes, -v) unlimited

* **Set the Maximum Number of Open Files:** To set the maximum number of open files to 2048, use:ulimit -n 2048
* **Set the Maximum File Size:** To set the maximum file size that can be created to 100 MB, use:ulimit -f 102400

(Note: The size is specified in 512-byte blocks, so 100 MB = 100 \* 1024 \* 2 = 204800 blocks.)

* **Set the Maximum Number of User Processes:** To set the maximum number of user processes to 512, use:ulimit -u 512
* **Last command**

The last command in Linux is used to display a list of the most recent login sessions of users. It reads from the /var/log/wtmp file, which logs all login and logout activities. This command is particularly useful for tracking user activity and investigating potential security issues.

**Basic Usage:** To display the most recent login sessions, simply type:last

**Output Explanation**

The output typically includes:

* **Username**: The name of the user who logged in.
* **TTY**: The terminal used for the session.
* **Hostname/IP Address**: The remote host from which the user logged in.
* **Login Time**: The date and time when the session started.
* **Logout Time**: The date and time when the session ended (or “still logged in” if the session is active).
* **Session Duration**: The length of the session.

**Common Options**

* **Display a Specific Number of Entries**: last -n 5

This command shows the last 5 login sessions.

* **Display Entries for a Specific User**: last username

Replace username with the actual username to see their login history.

* **Hide Hostname Field**: last -R

This command hides the hostname field in the output.

* **Show Full Login and Logout Times**: last -F

This command displays the full date and time for login and logout events.

* **Filter by Time Period**: last -s YYYY-MM-DD -t YYYY-MM-DD

This command shows login entries between the specified start (-s) and end (-t) dates.

**Practical Examples**

1. **List All Recent Logins**: last
2. **Show Last 10 Logins**: last -n 10
3. **Show Logins for a Specific User**: last john
4. **Show Logins from Yesterday to Today**: last -s yesterday -t today