**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. It possesses a user interface for the management of tasks that can be done by end users—such as running applications, file management, and input/output management. The operating system, for example, is Windows, macOS, and Linux.

A diagram of a application

Description automatically generated

## What is Linux?

[Linux](https://www.geeksforgeeks.org/introduction-to-linux-operating-system/) is a Unix-based, open-source operating system. Created by Linus Torvalds in 1991, Linux has grown to be a major force in the tech world: powering anything from smartphones and servers to supercomputers. is built upon the Linux 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. But the Linux Kernel alone is not enough to make a complete operating system. To create a full and functional system, the Linux Kernel is combined with a collection of software packages and utilities, which are together called Linux distributions. These distributions make the Linux Operating System ready for users to run their applications and perform tasks on their computers securely and effectively.

* **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.

# **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

### **The Linux LiveCD**

A relatively new phenomenon in the Linux world is the bootable Linux CD distribution. This lets you see what a Linux system is like without actually installing it. Most modern PCs can boot from a CD instead of the standard hard drive. To take advantage of this some Linux distributions create a bootable CD that contains a sample Linux system (called a Linux Live CD). Because of the limitations of the single CD size, the sample can’t contain a complete Linux system, but you’d be surprised at all the software they can cram in there. The result is anything on your hard drive.

# **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.

A diagram with text and words

Description automatically generated with medium confidence

The Linux/Unix file system hierarchy base begins at the root and everything starts with the root directory.

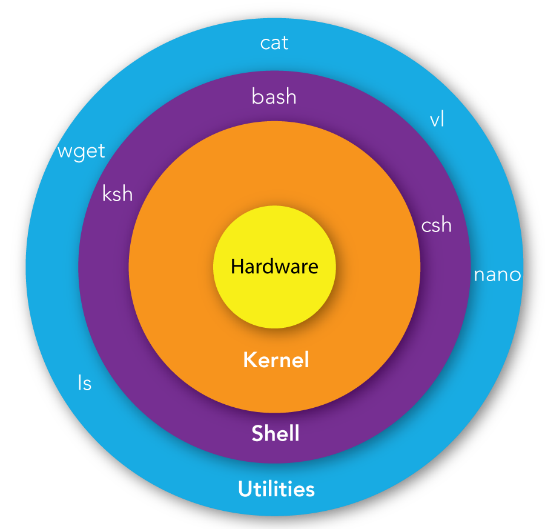
### **These are the common top-level directories associated with the root directory:**

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

### Some other directories in the Linlscleux system:

| **Directorie s** | **Description** |
| --- | --- |
| **/boot** | It contains all the boot-related information files and folders such as conf, grub, etc. |
| **/dev** | It is the location of the device files such as dev/sda1, dev/sda2, etc. |
| **/lib** | It contains kernel modules and a shared library. |

* **Linux Shell**
* A Linux shell is a command-line interface that provides users with an environment to interact with the operating system. It interprets and executes commands entered by the user or from scripts. The shell acts as an intermediary between the user and the kernel, which is the core of the operating system.



Types of Shells in Linux

* **Bash (Bourne Again Shell)**: The most common and default shell in many Linux distributions.
* **sh (Bourne Shell)**: The original Unix shell developed by Stephen Bourne.
* **csh (C Shell)**: A shell with syntax similar to the C programming language, developed by Bill Joy.
* **ksh (Korn Shell)**: Developed by David Korn, it includes features of both the Bourne and C Shells.
* **zsh (Z Shell)**: An extended Bourne Shell with many improvements, including better customization and interactive features.
* **Linux Commands**
* SSH: The ssh (Secure Shell) command in Linux is used to securely connect to remote systems over a network
* 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
* Pwd: Tells us the present (current) working directory
* Cd: We use this command to change the directory. Cd / will take you to the root of the file system. Cd .. will take you one path back
* 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
* 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.
* Cat: To see what is inside the file use the cat command. Cat filename
* Mkdir: this command is used to make a new directory
* Cp: we can copy a file using this command. Cp filename destination path
* 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
* Rmdir: we can remove a directory using this command.
* Ln: we can create a link to a file using this command. Ln -s filename and then link
* Clear: clear the screen, also you can press Ctrl+l
* 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
* 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.
* Finger: Finger command is used to inspect the user. Finger username
* Man: the man command is short for manual. It displays details about a certain command. Man command\_name
* 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
* Zip : this command is used to zip a file. Zip zipfile data\_to\_zip
* Unzip: unzip the zipped file
* 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 commands. cmp file1 file2
* Diff: this command is used to compare two commands. 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
* Find: This command is used to find the file or directory. Sudo file / -name “filename”
* 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
* Ip address: same as above
* Grep: to see a particular interface’s ip address, use this command. Ip address | grep eth0
* Ping: to know if your website is up. Ping -c 5 websitename
* Netstat: this command is used to get to know which ports are available on the linux machine. Netstat
* Ss: same as above
* Uname: to get to know about the system. Uname -a
* Sort: The sort command is used to sort lines of text files. It can sort alphabetically, numerically, and in various other ways. sort file.txt
* 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
* History: to know about all the commands you worked on use this command.
* Reboot: sudo reboot
* Shutdown: sudo shutdown
* **Additional Shortcut Tips**
* **Tilde(**~**):** shortcut to your home base
* **Dot(.):** the folder you’re in right now
* **double dot(..):** the folder one level above
* **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
* Basic movement:

h l k j character left, right; line up, down (also arrow keys)

b w word/token left, right

ge e end of word/token left, right

0 $ jump to first/last character on the line

* x delete
* u undo
* **Symbolic links**

Symbolic links (also known as symlinks or soft links) in Linux are a type of file that points to another file or directory. They are similar to shortcuts in Windows. Symlinks can be used to create convenient ways to access files and directories without duplicating data.

**Creating Symbolic Links**

To create a symbolic link, you use the ln command with the -s option. The syntax is:

ln -s [target] [link\_name]

* **target**: The file or directory you want to link to.
* **link\_name**: The name of the symbolic link you are creating.

For example, to create a symbolic link named mylink pointing to a file named myfile.txt, you would use:

ln -s myfile.txt mylink

* **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 traverse a directory.

**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.
3. **Others (o)**: All other users.

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:

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.
* **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 value is a three-digit octal number that defines the permissions to be masked (i.e., turned off) for new files and directories. Each digit corresponds to the owner, group, and others, respectively.

**Default Permissions**

When a new file or directory is created, it initially has a set of default permissions:

* Files: 666 (read and write for everyone, no execute)
* Directories: 777 (read, write, and execute for everyone)

**How umask Works**

* The umask value is subtracted (bitwise AND operation) from the default permissions to determine the final permissions for the new file or directory.
* **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**: qqc2

* **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"

* **which Command**

The which command is used to find the location of executable files in the directories listed in the user's PATH environment variable. Syntax: which [command]

* **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: tail filename
5. Tail: The tail command displays the last few lines of a file (default is 10 lines). Syntax: tail filename

* **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. Syntax: nano filename. Basic Commands in nano: Moving Around, Editing, Searching, Saving and Editing

* **VI Editor**

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.
  + 0: 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 character under the cursor.
  + 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.
* **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.

**Saving and Exiting**

In Command-Line Mode:

* :w: Write (save) the file.
* :q: Quit vi.
* :wq or :x: Write and quit.
* **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.
* **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**:

* Multi-document interface
* Syntax highlighting
* Code folding
* Search and replace with regular expressions
* Built-in terminal
* Plugins support

1. **Sublime Text:** Sublime Text is a popular 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
* 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**

**Tar:** tar (short for tape archive) is a command-line utility 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

* **Wildcards**

Wildcards in Linux are special characters that can be used in place of other characters or strings to simplify the process of working with multiple files and directories. 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.
2. **Question Mark (?)**:
   * Matches exactly one character.
   * Example: ls file?.txt matches file1.txt, file2.txt, etc., but not file10.txt.
3. **Square Brackets ([])**:
   * Matches any one of the enclosed characters.
   * Example: ls file[123].txt matches file1.txt, file2.txt, and file3.txt.
4. **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.
* **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.

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.

Example:

#!/bin/bash

ls -l > output.txt

This script redirects the output of the ls -l command to output.txt.

* **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.

Example:

#!/bin/bash

ls /nonexistent 2> error.txt

This script redirects the error message from the ls command to error.txt.

**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

* **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: The grep command is used to search for patterns within files. It stands for "global regular expression print".

**Search for a pattern in a file:** grep "pattern" filename

**Search recursively in all files within a directory:** grep -r "pattern" directory

Ignore case (-i): grep -i "pattern" filename

Invert match (show lines that do not match the pattern). (-v): grep -v "pattern" filename

Show line numbers (-n): grep -n "pattern" filename

Match whole words (-w): grep -w "pattern" filename

Count the number of matches. (-c): grep -c "pattern" filename

* **Pipes**

Pipes (|) allow you to use the output of one command as the input to another, enabling powerful command chaining. 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. The shell prompt is defined by the PS1 environment variable in most shells, such as bash and zsh

**Understanding PS1**

The PS1 variable controls the primary prompt string, which is the main prompt you see when you start a shell session. It can include various placeholders for different information, such as the username, hostname, current directory, time, etc.

**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'

* **Environment variable**

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

**Common Environment Variables**

* **HOME**: The home directory of the current user.
* **USER**: The name of the current 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.
* **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 a snapshot of current processes.

Basic Process Management Commands:

**kill**: Send a signal to a process, usually to terminate it. SYNTAX: kill PID

**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**

* **&:** Run a command in the background.
* **jobs**: List active jobs.
* **fg**: Bring a background job to the foreground.
* **bg**: Resume a stopped job in the background.
* **Ctrl+Z**: Suspend the currently foregrounded job.
* **Scheduling Repeated Jobs with Cron**

Scheduling repeated jobs with cron is a common practice in Linux for automating tasks. The cron daemon runs scheduled tasks at specified times and intervals. Here's a guide on how to use cron for scheduling repeated jobs. cron uses a configuration file called crontab to manage scheduled tasks. Each line in the crontab file represents a job and its schedule.

Crontab Syntax

A crontab entry consists of six fields:

\* \* \* \* \* command\_to\_be\_executed

- - - - -

| | | | |

| | | | +----- day of the week (0 - 7) (Sunday=0 or 7)

| | | +------- month (1 - 12)

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

| +----------- hour (0 - 23)

+------------- minute (0 - 59)

Examples of Crontab Entries

* Run a Job Every Minute: \* \* \* \* \* /path/to/command
* Run a Job Every Monday at 7:00 AM:- 0 7 \* \* 1 /path/to/command
* **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.

**Run a command as root: sudo command\_to\_run**

* **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:** The shell history keeps a record of the commands you have entered in the terminal. You can view, search, and reuse these commands.

**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]

* **Package and Package Manager**

Package management in Linux is a fundamental aspect of system administration, allowing users to install, update, and remove software packages efficiently. 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 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.

**BIOS (Basic Input/Output System) Initialization**

**Power-On Self Test (POST):** The BIOS performs a POST to check the hardware components (CPU, RAM, disk controllers, etc.) and ensure they are functioning correctly.

**Boot Device Selection:** The BIOS reads the boot sequence from the BIOS settings and identifies the bootable device (hard disk, CD/DVD, USB drive).

**Bootloader Stage**

**MBR (Master Boot Record) and GRUB**

**MBR**: The first sector of the bootable disk contains the MBR, which includes the bootloader information.

**GRUB (GRand Unified Bootloader):** The most common bootloader used in Linux systems. GRUB loads the kernel into memory and hands over control to it.

**Kernel Initialization**

**Loading the Kernel:** The selected kernel image is loaded into memory.

**Decompressing the Kernel:** The kernel is typically compressed and is decompressed during this stage.

**Hardware and Driver Initialization:** The kernel initializes all the necessary hardware drivers and performs a system hardware check.

**Init/Systemd Initialization:** systemd: The modern init system used in most contemporary Linux distributions.

**Root File System Mounting:** The actual root file system is mounted, replacing the initial RAM disk file 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**

Disk management in Linux involves a variety of tasks such as partitioning, formatting, mounting, and monitoring disk usage. These tasks are essential for efficiently managing storage and ensuring system stability

* **Disk Partitioning:** Partitioning a disk involves dividing it into sections, each of which can be managed separately. Tools commonly used for partitioning include fdisk, parted, and gdisk.

**Using fdisk:** fdisk is a text-based utility for creating and managing partitions on a hard drive.

**Interactive session commands:**

p: Print the partition table.

n: Add a new partition.

d: Delete a partition.

w: Write changes and exit.

q: Quit without saving changes.

**Using parted:** parted is a more advanced tool that supports both MBR and GPT partition tables

* **Partition Tables:** The partition table keeps track of the partitions on a disk. There are two main types of partition tables:

**MBR (Master Boot Record)**

Structure: The MBR is located in the first 512 bytes of the disk. It contains the bootloader and the partition table.

Partition Limit: Supports up to four primary partitions, or three primary partitions and one extended partition containing multiple logical partitions.

Disk Size Limit: Supports disks up to 2 TB in size.

**GPT (GUID Partition Table)**

Structure: GPT is part of the UEFI standard and is stored in multiple locations on the disk for redundancy.

Partition Limit: Supports up to 128 primary partitions (default configuration).

Disk Size Limit: Supports disks larger than 2 TB, theoretically up to 9.4 ZB.

* **Formatting Partitions:** After partitioning a disk, you need to format the partitions with a filesystem. Common filesystems include ext4, xfs, and btrfs.
* **Mounting and Unmounting Filesystems:** To use a filesystem, it must be mounted to a directory in the file system hierarchy.

Create a mount point: sudo mkdir /mnt/mydisk

Mount a filesystem: sudo mount /dev/sdX1 /mnt/mydisk

* **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.

**Physical Volume (PV):** The basic building block of LVM. It can be a hard disk, a partition, or even a RAID array.Created from physical storage devices like /dev/sda1, /dev/sdb, etc.

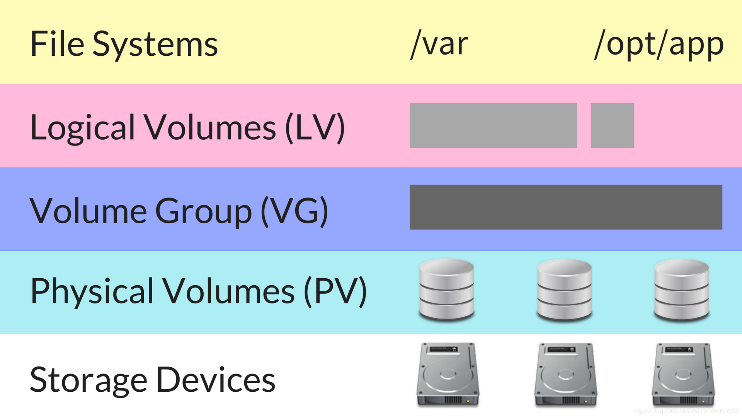
**Volume Group (VG):** A collection of Physical Volumes.Acts as a pool of storage from which logical volumes can be created.

**Logical Volume (LV):** A portion of a Volume Group.Can be thought of as a virtual partition that you can create, resize, and delete as needed.

**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**



* **Creating Physical Volumes (PVs), Volume Groups (VGs), and Logical Volumes**

Creating and managing Physical Volumes (PVs), Volume Groups (VGs), and Logical Volumes (LVs) with the Logical Volume Manager (LVM) in Linux involves a series of steps.

1. **Creating Physical Volumes (PVs):** A Physical Volume is the basic building block of LVM. It can be a whole disk, a partition, or a RAID array.

Example: Creating a Physical Volume

* Identify the device you want to use Use lsblk or fdisk -l to list all available disks and partitions.
* Create the Physical Volume using the pvcreate command.

2.**Creating Volume Groups (VGs):** A Volume Group is a pool of storage that consists of one or more Physical Volumes. Logical Volumes are allocated from this pool.

Example: Creating a Volume Group

* Create the Volume Group using the vgcreate command.
* Replace myvg with the name of your volume group and /dev/sdX with your physical volume identifier. You can include multiple physical volumes if needed

3. **Creating Logical Volumes (LVs)**: A Logical Volume is created from the storage space within a Volume Group. It can be formatted with a filesystem and mounted for use.

Example: Creating a Logical Volume

* Create the Logical Volume using the lvcreate command.
* Replace mylv with the name of your logical volume, 10G with the desired size, and myvg with the name of your volume group.

4. **Formatting and Mounting Logical Volumes:** After creating a Logical Volume, you need to format it with a filesystem and mount it to use it.

Example: Formatting and Mounting a Logical Volume

* Format the Logical Volume with a filesystem (e.g., ext4).
* Create a mount point
* Mount the Logical Volume
* Verify the mount
* **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**

Managing users and groups in Linux is a fundamental aspect of system administration. It involves creating, modifying, and deleting user accounts and groups, as well as assigning permissions to files and directories.

1. **Managing Users**

* Creating a User: To create a new user, use the useradd command followed by the username. You can also specify additional options such as home directory, shell, and more. Syntax:- sudo useradd -m -s /bin/bash username
* Modifying a User: To modify an existing user, use the usermod command.
* Change the user’s shell: sudo usermod -s /bin/zsh username
* Change the user’s home directory: sudo usermod -d /new/home/directory username
* Add the user to a group: sudo usermod -aG groupname username
* Deleting a User: To delete a user and their home directory, use the userdel command with the -r option. Syntax:- sudo userdel -r username

2. **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

**3. Viewing User and Group Information**

* List all users: cat /etc/passwd
* List all groups:- cat /etc/group

4. **Managing User and Group Membership**: To manage which users belong to which groups, you can directly edit the /etc/group file or use the gpasswd command.

* **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.

2. **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:** 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**: 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:** 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:** Range: 224.0.0.0 to 239.255.255.255**,** Used for multicast.

**Class E:** Range: 240.0.0.0 to 255.255.255.255**.** Reserved for future use, or research and development purposes.

3. **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

5**. 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)

6. **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.

**Components of DNS**

* **Domain Name**: A human-readable address (e.g., www.example.com).
* **IP Address**: The numerical address (e.g., 192.168.1.1) that computers use to locate each other on a network.
* **DNS Server**: A server that responds to queries asking for the IP address associated with a domain name.

**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 # Google's public DNS server

nameserver 8.8.4.4 # Google's secondary DNS server

**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-touch 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.

* **Basic Connectivity Tests**

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

ping 192.168.1.1

* **Interface Configuration and Status**

**Ifconfig:** The ifconfig command displays or configures network interfaces**.**

**Ip:** The ip command is a modern alternative to ifconfig for managing network interfaces.

* **Route and Gateway Configuration**

**Route:** The route command shows or modifies the IP routing table.

* **Analyzing Network Traffic**

**Traceroute**: The traceroute command displays the route packets take to a network host.

**Netstat:** The netstat command displays network connections, routing tables, interface statistics, masquerade connections, and multicast memberships.

**Tcpdump:** The tcpdump command captures and analyzes network packets.

* **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**

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

* **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
* **$0** - The filename of the current script.
* **$#** - The number of arguments supplied to a script.
* **$\*** - All the arguments are double quoted. If a script receives two arguments, $\* is equivalent to $1 $2.
* **$@** - All the arguments are individually double quoted. If a script receives two arguments, $@ is equivalent to $1 $2.
* **$?** - The exit status of the last command executed.
* **$**$ - The process number of the current shell. For shell scripts, this is the process ID under which they are executing.
* **$!** - The process number of the last background command.
* **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**

The getopts command in Linux shell scripting is used to parse command-line options and arguments. It’s a built-in utility in Bash that simplifies handling options and their arguments. Here’s a detailed explanation:

**Basic Syntax**

getopts optstring variable

* optstring: A string containing the option characters to be recognized. If a character is followed by a colon (:), it indicates that the option requires an argument.
* variable: The name of the variable that will hold the current option being processed.

**Example Script**

Here’s a simple example to illustrate how getopts works:

#!/bin/bash

while getopts ":a:b:c" opt; do

case $opt in

a)

echo "Option -a with argument: $OPTARG"

;;

b)

echo "Option -b with argument: $OPTARG"

;;

c)

echo "Option -c"

;;

\?)

echo "Invalid option: -$OPTARG" >&2

;;

:)

echo "Option -$OPTARG requires an argument." >&2

;;

esac

done

In this script:

* :a:b:c specifies that -a and -b require arguments, while -c does not.
* $OPTARG contains the argument for the option if it requires one.
* The \? case handles invalid options.
* The : case handles missing arguments for options that require them.
* **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 sections from each line of input, typically from a file. It can cut by bytes, characters, or fields.

**Examples:**

1. **Cut by Field**:

# Extract the first and third fields from a comma-separated file

cut -d',' -f1,3 file.txt

Here, -d',' specifies the delimiter as a comma, and -f1,3 specifies the fields to extract.

1. **Cut by Character**:

# Extract characters 1 to 5 from each line

cut -c1-5 file.txt

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

**Basic Syntax:**

awk 'pattern { action }' file

pattern: The condition to match.

action: The action to perform on matching lines.

**Examples:**

1. **Print Specific Fields**:

# Print the first and third fields of a space-separated file

awk '{print $1, $3}' file.txt

1. **Pattern Matching**:

# Print lines containing the word "manager"

awk '/manager/ {print}' employee.txt

1. **Field Manipulation**:

# Print the first field and the sum of the second and third fields

awk '{print $1, $2 + $3}' file.txt

* **Configuring a Mini Network and Scripting for Remote Systems**

Configuring a mini network and scripting for remote systems in Linux can be quite powerful for managing and automating tasks across multiple machines. Here are some key steps and concepts:

**Configuring a Mini Network**

1. **Setting Up Virtual Machines**: Use tools like VirtualBox or VMware to create virtual machines (VMs).
2. Assigning IP Addresses: Assign static IP addresses to each VM for easier management. Edit the /etc/network/interfaces file (Debian-based) or /etc/sysconfig/network-scripts/ifcfg-eth0 (Red Hat-based) to set static IPs.
3. Configuring SSH: Install and configure SSH on each VM to allow remote access.

Scripting for Remote Systems

1. **Using SSH for Remote Commands**: You can execute commands on remote systems using SSH.

ssh user@remote\_host 'command'

1. **Automating Tasks with Scripts**: Create scripts to automate tasks across multiple systems.

#!/bin/bash

for host in host1 host2 host3; do

ssh user@$host 'sudo apt update && sudo apt upgrade -y'

done

1. **Copying Files to Remote Systems**: Use scp or rsync to copy files to remote systems.

scp file.txt user@remote\_host:/path/to/destination

1. **Running Local Scripts on Remote Systems**: You can run a local script on a remote system by copying it and then executing it.

scp script.sh user@remote\_host:/tmp/

ssh user@remote\_host 'bash /tmp/script.sh'

1. **Using**expect**for Automated Interactions**: The expect command can automate interactions with programs that require user input.

#!/usr/bin/expect

spawn ssh user@remote\_host

expect "password:"

send "your\_password\r"

interact