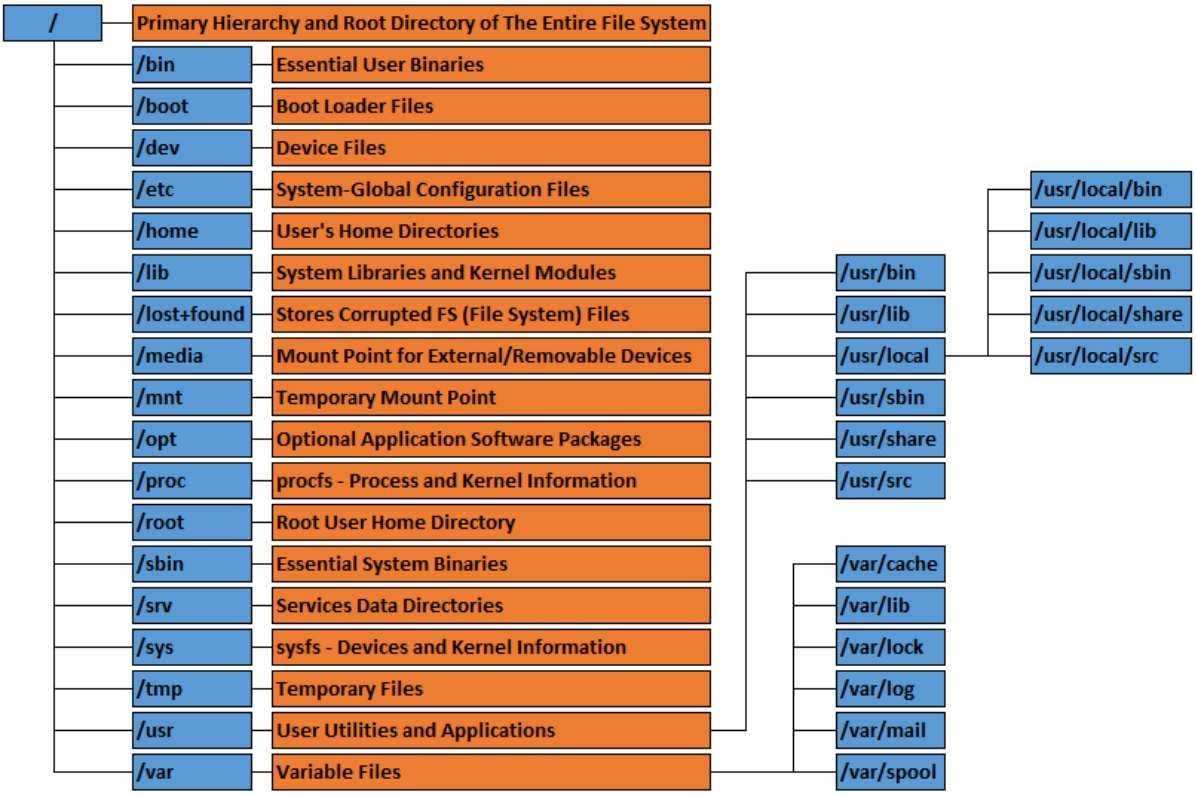
Vishal Kurane

vishalarunkurane@gmail.com | LinkedIn Profile

Contents

Linux file system hierarchy:



1. / (Root Directory)

- The topmost directory in Linux.
- Everything in the Linux file system is stored under /.
- Mounted file systems and devices are all accessible from this single directory.

2. /bin (Binaries)

- Contains essential binary executables (commands) needed by all users, including system administrators.
- Commonly used commands like ls, cp, mv, rm, cat, echo, mkdir, and rmdir are stored here.
- These binaries are required for system recovery and are available in single-user mode.

3. /boot (Boot Loader Files)

- Stores boot-related files required to start Linux.
- Contains the Linux kernel (vmlinuz), initial RAM disk (initrd or initramfs), bootloader configuration files (like grub.cfg for GRUB).
- Example files:
    - /boot/vmlinuz – The compressed Linux kernel.
    - /boot/initrd.img – Initial RAM disk image used during boot.

4. **/dev (Device Files)**

- Contains special device files representing hardware and virtual devices.
- Examples:
  - /dev/sda – First hard disk.
  - /dev/tty – Terminal devices.
  - /dev/null – Discard data.
  - /dev/random – Generate random numbers.
- These files allow interaction with devices as if they were regular files.

5. **/etc (Configuration Files)**

- Stores system-wide configuration files and shell scripts used during system boot.
- Examples:
  - /etc/passwd – User account details.
  - /etc/shadow – Encrypted passwords.
  - /etc/fstab – File system mount points.
  - /etc/network/interfaces – Network configuration.

6. **/home (User Home Directories)**

- Contains personal directories for each user.
- Example:
  - /home/vishal/ – Home directory for user "vishal".
- Each user has their own configuration files, documents, downloads, and settings in this directory.

7. **/lib (System Libraries)**

- Contains shared libraries required by binaries in /bin and /sbin.
- Includes kernel modules (/lib/modules/).
- Libraries are similar to Windows .DLL files.
- Examples:
  - /lib/libc.so.6 – Standard C library.
  - /lib/modules/ – Kernel modules.

8. **/media (Mount Points for Removable Media)**

- Automatically mounts external storage devices like USB drives and CD-ROMs.
- Example:
  - /media/usb/ – Mounted USB drive.
  - /media/cdrom/ – Mounted CD/DVD.

9. **/mnt (Mount Directory)**

- Temporary mount point for manually mounting file systems.
- Used for mounting external drives, network shares, or additional partitions.
- Example:
  - mount /dev/sdb1 /mnt/mydrive – Mounts a USB drive to /mnt/mydrive.

10. /opt (Optional Software)

- Contains third-party software and add-on applications.
- Used for installing non-standard applications.
- Example:
    - /opt/google/chrome/ – Google Chrome installed here.

11. /proc (Process Information)

- A virtual file system providing real-time system and process information.
- Contains directories named after running process IDs (PIDs).
- Examples:
    - /proc/cpuinfo – CPU details.
    - /proc/meminfo – Memory usage.
    - /proc/uptime – System uptime.

12. /root (Root User's Home Directory)

- Home directory for the root user.
- Different from /home as it is not stored under it.
- Contains configuration files and scripts for the root user.

13. /sbin (System Binaries)

- Contains essential binaries used by the system administrator.
- Requires root privileges to execute most commands.
- Examples:
    - fsck – File system check.
    - mount – Mount a file system.
    - shutdown – Shut down the system.

14. /srv (Service Data)

- Contains data for system services like web servers and FTP.
- Example:
    - /srv/www/ – Web server files.
    - /srv/ftp/ – FTP server files.

15. /sys (System Information)

- Exposes kernel and hardware information.
- Similar to /proc, but more structured.
- Examples:
    - /sys/class/net/ – Network interfaces.
    - /sys/block/ – Block devices (disks).

16. /tmp (Temporary Files)

- Stores temporary files created by applications and users.
- Data is deleted on system reboot.
- Example:
    - /tmp/session.log – Temporary session log file.

17. /usr (User System Resources)

- Contains user-installed software, libraries, and documentation.
- Divided into subdirectories:
    - /usr/bin/ – Non-essential command binaries (nano, vim, git).
    - /usr/sbin/ – Non-essential system administration binaries.
    - /usr/lib/ – Shared libraries.
    - /usr/share/ – Documentation, icons, themes.

18. /var (Variable Data)

- Stores variable data such as logs, mail, and print queues.
- Examples:
    - /var/log/ – System logs (syslog, auth.log).
    - /var/mail/ – Email storage.
    - /var/spool/ – Print jobs, cron jobs.

19. /run (Runtime Data)

- Stores volatile runtime data since the system booted.
- Replaces older directories like /var/run/ and /var/lock/.
- Examples:
    - /run/utmp – Active user sessions.
    - /run/systemd/ – Systemd runtime information.

Linux file Permissions:

Each file and directory in Linux have **permissions** associated with three categories:

1. **Owner** (User) – The person who created the file.
2. **Group** – A collection of users with shared permissions.
3. **Others** – Everyone else who has access to the system.

Linux uses **three types of permissions**:

- **Read (r)** – Allows reading a file's contents or listing a directory.
- **Write (w)** – Allows modifying or deleting a file, or adding/removing files in a directory.
- **Execute (x)** – Allows executing a file (script/program) or accessing a directory.

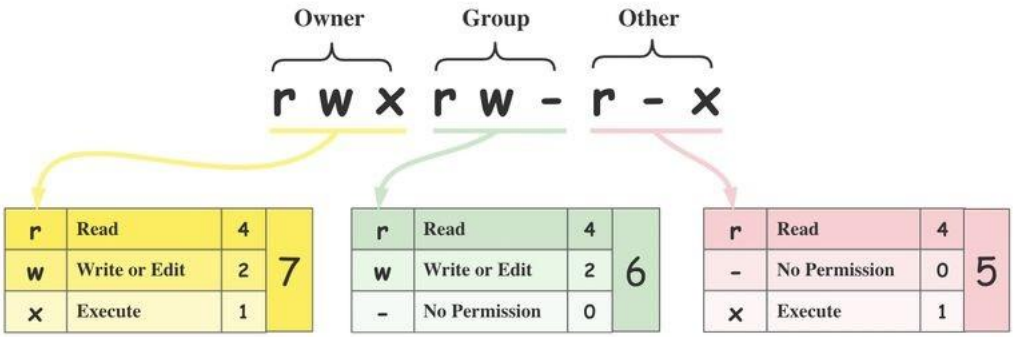
When you run the `ls -l` command, you see file permissions in the following format:

-rwxr-xr-- 1 user group 1234 Aug 20 10:00 myfile.sh

Position	Meaning
-	File type (- for file, d for directory, l for link)
rwx	Owner (User) permissions
r-x	Group permissions
r--	Others permissions

Linux File Permissions

Binary	Octal	String Representation	Permissions
000	0 (0+0+0)	---	No Permission
001	1 (0+0+1)	--x	Execute
010	2 (0+2+0)	-w-	Write
011	3 (0+2+1)	-wx	Write + Execute
100	4 (4+0+0)	r--	Read
101	5 (4+0+1)	r-x	Read + Execute
110	6 (4+2+0)	rw-	Read + Write
111	7 (4+2+1)	rwx	Read + Write + Execute



Linux Shell Command Components:

Along with a Linux shell command, we can provide several additional elements to control its behavior Below are the key components:

1. **Arguments**

- Arguments are values passed to a command to specify what it should operate on.
- **Example:**

```
cp file1.txt /home/user/
```

  - cp is the command.
  - file1.txt is the first argument (source file).
  - /home/user/ is the second argument (destination directory).

2. **Options (Flags)**

- Options modify the behaviour of a command.
- **Example:**

```
ls -l --human-readable
```

  - -l enables long format listing.
  - --human-readable makes file sizes easier to read.

3. **Pipelines (|)**

- Pipes allow the output of one command to be used as the input for another.
- **Example:**

```
ls -l | grep "test"
```

4. **Redirections (>, >>, <)**

- Redirects standard input/output to or from files.
- **Examples:**

```
ls > output.txt   # Overwrites
ls >> output.txt # Appends
sort < file.txt   # Reads from a file
```

5. **Environment Variables**

- Variables that store data used by the shell and scripts.
- **Example:**

```
echo $HOME
```

6. **Command Substitution ($(), ``)**

- Uses the output of a command as an argument for another.
- **Example:**

```
echo "Today is $(date)"
```

7. **Background Execution (&)**

- Runs commands in the background.
- **Example:**

```
sleep 10 &
```

8. **Job Control (fg, bg, jobs)**

- Manages running jobs.
- **Examples:**

```
sleep 100 &
jobs
fg %1
bg %1
```

9. **Brace Expansion ({})**

- Generates multiple strings or sequences.
- **Examples:**

```
mkdir {dir1,dir2,dir3}
echo {1..5}
```

10. **Wildcard Characters (*, ?, [])**

- Pattern matching in filenames.
- **Examples:**

```
ls *.txt
ls file?.txt
ls file[1-3].txt
```

11. **Quoting (", ', \)**

- Prevents interpretation of special characters.
- **Examples:**

```
echo 'Hello $USER'   # Prints literally
echo "Hello $USER"  # Expands variable
echo "Hello \"World\""
```

12. **Shebang (#!)**

- Specifies the interpreter for scripts.
- **Example (myscript.sh):**

```
#!/bin/bash
echo "This is a Bash script"
```

13. **Control Operators (&&, ||, ;)**

- Used for command sequencing.
- && executes the second command only if the first succeeds.
- || executes the second command only if the first fails.
- ; executes commands sequentially, regardless of success or failure.
- **Examples:**

```
mkdir test && cd test  # Creates and navigates to test if successful
mkdir test || echo "Failed"  # Prints message if mkdir fails
echo "Hello"; ls      # Executes both commands
```

14. **Process Substitution (<(), >())**

- Treats the output of a command as a file.
- Useful for comparing outputs without creating temporary files.
- **Example:** Compares the file listings of two directories.

```
diff <(ls /dir1) <(ls /dir2)
```

15. **Tilde Expansion (~)**

- Represents the home directory.
- **Example:**

```
cd ~  # Navigates to the home directory
```

16. **Arithmetic Expansion ($(( )))**

- Performs arithmetic calculations inside shell scripts.
- **Example:**

```
echo $((5 + 3)) # Outputs 8
```

17. **Aliases**

- Shortcuts for longer commands.
- **Example:**

```
alias ll="ls -la"
ll  # Runs 'ls -la'
```

18. Functions

- Define reusable command sequences.
- Functions allow writing custom commands inside scripts or interactive shells.
- **Example:**

```
myfunc() {
  echo "Hello, $1"
}
myfunc Vishal # Prints 'Hello, Vishal'
```

Basics Linux Commands:

1. **ls (List Files and Directories)**

   - The ls command is used to list files and directories in a given directory.
   - **Syntax:**

     ```
     ls [OPTIONS] [DIRECTORY]
     ```

   - **Common Options:**

     - -l : Long listing format (shows details like permissions, ownership, size, and modification time).
     - -a : Displays hidden files (files starting with .).
     - -h : Human-readable file sizes.
     - -R : Recursively lists files in subdirectories.

   - **Example:**

     ```
     ls -lah /home/user
     ```

2. **cd (Change Directory)**

   - The cd command is used to navigate between directories.
   - **Syntax:**

     ```
     cd [DIRECTORY]
     ```

   - **Examples:**

     ```
     cd /home/user/documents  # Go to the documents folder
     cd ..                    # Move one directory up
     cd ~                     # Move to home directory
     ```

3. **pwd (Print Working Directory)**

   - The pwd command prints the current working directory.
   - **Syntax:**

     ```
     pwd
     ```

   - **Example:**

     ```
     pwd # Outputs: /home/user/documents
     ```

4. **mkdir (Make Directory)**

- The mkdir command creates new directories.
- **Syntax:**

```
mkdir [OPTIONS] DIRECTORY_NAME
```

- **Common Options:**
  - **-p : Creates parent directories if they do not exist.**

- **Example:**

```
mkdir -p /home/user/new_folder/subfolder
```

5. **rm (Remove Files and Directories)**

- The rm command deletes files and directories.
- **Syntax:**

```
rm [OPTIONS] FILE/DIRECTORY
```

- **Common Options:**

  - -r : Recursively deletes a directory and its contents.
  - -f : Force deletion without confirmation.

- **Example:**

```
rm file.txt # Deletes a single file
```

6. **rm -r devops/**

- This command deletes the devops directory and all its contents recursively.
- **Example:**

```
rm -r devops/
```

7. **rmdir (Remove Empty Directory)**

- **The rmdir command removes an empty directory.**
- **Syntax:**

```
rmdir DIRECTORY_NAME
```

- **Example:**

```
rmdir empty_folder
```

8. **cat (Concatenate and Display File Contents)**

- The cat command displays the contents of a file.
- **Syntax:**

```
cat [OPTIONS] FILE
```

- **Example:**

```
cat file.txt
```

9. **zcat (View Compressed Files Without Extracting)**

- The zcat command displays the contents of a compressed .gz file.
- **Syntax:**

```
zcat FILE.gz
```

- **Example:**

```
zcat archive.gz
```

10. **touch (Create an Empty File or Update Timestamp)**

- The touch command creates a new empty file or updates an existing file's timestamp.
- **Syntax:**

```
touch FILE_NAME
```

- **Example:**

```
touch newfile.txt
```

11. **head (View First 10 Lines of a File)**

- The head command displays the first 10 lines of a file.
- **Syntax:**

```
head [OPTIONS] FILE
```

- **Example:**

```
head -n 5 file.txt # Displays the first 5 lines
```

12. **tail (View Last 10 Lines of a File)**

- The tail command displays the last 10 lines of a file.
- **Syntax:**

```
tail [OPTIONS] FILE
```

- **Example:**

```
tail -n 5 file.txt # Displays the last 5 lines
```

13. **tail -f (Follow File Updates in Real-Time)**

- The tail -f command is used to monitor file changes in real-time.
- **Example:**

```
tail -f /var/log/syslog
```

14. **less (View Large Files Page by Page)**

- The less command allows you to view file contents one page at a time.
- **Syntax:**

```
less FILE
```

- **Example:**

```
less largefile.txt
```

15. **more (Similar to less, but Without Backward Navigation)**

- The more command allows you to view file contents but only scroll forward.
- **Example:**

```
more largefile.txt
```

16. **cp (Copy Files and Directories)**

- The cp command copies files and directories.
- **Syntax:**

```
cp [OPTIONS] SOURCE DESTINATION
```

- **Example:**

```
cp file1.txt /home/user/backup/
```

17. **mv (Move or Rename Files and Directories)**

- The mv command moves or renames files and directories.
- **Example:**

```
mv oldname.txt newname.txt
```

18. **wc (Word Count, Line Count, and Character Count)**

- The wc command counts words, lines, and characters in a file.
- **Example:**

```
wc -l file.txt # Counts lines in a file
```

19. **ln -s (Create a Symbolic Link)**

- The ln -s command creates a symbolic (soft) link to a file.
- **Example:**

```
ln -s /home/user/file.txt shortcut.txt
```

20. **ln (Create a Hard Link)**

- The ln command creates a hard link to a file.
- **Example:**

```
ln file.txt hardlink.txt
```

21. **cut (Extract Specific Columns from a File)**

- The cut command extracts specific fields or characters from a file.
- **Example:**

```
cut -d ',' -f 2 file.csv # Extracts the second column
```

22. **tee (Read from Standard Input and Write to File and Output)**

- The tee command redirects output to both the screen and a file.
- **Example:**

```
ls | tee output.txt
```

23. **sort (Sort Lines in a File Alphabetically or Numerically)**

- The sort command sorts lines in a file.
- **Example:**

```
sort file.txt
```

24. clear (Clear the Terminal Screen)

- The clear command clears the terminal display.
- **Example:**

```
clear
```

25. diff (Compare Two Files Line by Line)

- The diff command compares two files and highlights the differences.
- **Example:**

```
diff file1.txt file2.txt
```

Login Commands:

1. ssh (Secure Shell)

- The ssh command is used to securely log into a remote machine over an encrypted connection. It is commonly used for remote administration of servers.
- **Syntax:** (logs into example.com as 'user')

```
ssh user@example.com
```

- **Common Options:**
  - -p <port>: Specifies a port number (default is 22)
  - -i <identity_file>: Uses a specific private key for authentication
  - -X or -Y: Enables X11 forwarding for graphical applications
  - -C: Enables compression

CPU Management Commands:

1. lscpu

Displays detailed information about the CPU architecture, including the number of cores, threads, model, and cache sizes.

Syntax:

```
lscpu [OPTIONS]
```

Common Options:

- -e → Displays a detailed list of CPU cores with additional information.
- -p → Outputs CPU details in a parseable format.

Example:

```
lscpu
```

Displays CPU architecture details.

2. nproc

Shows the number of processing units (cores) available to the system.

Syntax:

```
nproc [OPTIONS]
```

Common Options:

- --all → Displays the total number of cores (including those that are offline).

Example:

```
nproc
```

Outputs the number of available CPU cores.

```
nproc --all
```

Shows the total number of CPU cores, including offline ones.

3. top

Displays real-time information about CPU usage, memory, and running processes.

Syntax:

```
top [OPTIONS]
```

Common Options:

- -d SECONDS → Refresh interval (default is 3 seconds).
- -p PID → Show only a specific process ID.
- -u USER → Show processes of a specific user.

Example:

```
top
```

Starts the live CPU and process monitoring.

```
top -d 1
```

Updates the statistics every **1 second**.

4. htop

An interactive and improved version of top with a better UI and sorting capabilities.

Syntax:

```
htop [OPTIONS]
```

Common Options:

- -u USER → Show processes for a specific user.
- -p PID → Show only specific process IDs.

Example:

```
htop
```

Launches the htop monitoring interface.

```
htop -u root
```

Shows processes owned by the **root user**.

5. mpstat

Reports CPU usage for each processor core. Part of the sysstat package.

Syntax:

```
mpstat [OPTIONS] [INTERVAL] [COUNT]
```

Common Options:

- -P ALL → Displays CPU usage for all cores.
- -u → Shows CPU usage in percentage.

Example:

```
mpstat -P ALL 2 5
```

Displays CPU usage for all cores, refreshing every **2 seconds** for **5 cycles**.

6. iostat

Displays CPU and disk I/O statistics. Part of the sysstat package.

Syntax:

```
iostat [OPTIONS] [INTERVAL] [COUNT]
```

Common Options:

- -c → Show only CPU usage.
- -x → Show extended statistics.

Example:

```
iostat -c 2 3
```

Displays CPU usage every **2 seconds**, for **3 cycles**.

7. vmstat

Reports system performance metrics, including CPU usage, memory, and disk I/O.

Syntax:

```
vmstat [OPTIONS] [INTERVAL] [COUNT]
```

Common Options:

- -s → Show a summary of CPU, memory, and swap usage.
- -a → Show active and inactive memory.

Example:

```
vmstat 2 5
```

Displays CPU statistics every **2 seconds** for **5 cycles**.

8. sar

Collects, reports, and saves CPU and system performance metrics. Part of the sysstat package.

Syntax:

```
sar [OPTIONS] [INTERVAL] [COUNT]
```

Common Options:

- -u → Show CPU usage.
- -P ALL → Show CPU usage per core.

Example:

```
sar -u 1 5
```

Shows CPU usage every **1 second** for **5 cycles**.

9. taskset

Assigns a specific CPU core(s) to a process.

Syntax:

```
taskset [OPTIONS] CPU_MASK COMMAND
```

Common Options:

- -p PID → Show or set CPU affinity for an existing process.
- CPU_MASK → Specifies which cores the process should run on.

Example:

```
taskset -c 0,1 my_script.sh
```

Runs my_script.sh on **CPU cores 0 and 1**.

```
taskset -p 0x1 1234
```

Assigns **CPU core 0** to process **PID 1234**.

10. chrt

Changes the scheduling priority and policy of a process.

Syntax:

```
chrt [OPTIONS] PRIORITY COMMAND
```

Common Options:

- -r → Use real-time scheduling.
- -p → Modify an existing process by PID.

Example:

```
chrt -r 50 ./my_program
```

Runs my_program with a **real-time priority of 50**.

```
chrt -p 30 1234
```

Changes the priority of **PID 1234** to **30**.

11. nice

Starts a process with a specific priority level (-20 is highest, 19 is lowest).

Syntax:

```
nice [OPTIONS] COMMAND
```

Common Options:

- -n VALUE → Set the priority (default is 10).

Example:

```
nice -n -5 my_script.sh
```

Runs my_script.sh with a **higher priority**.

12. renice

Changes the priority of an already running process.

Syntax:

```
renice [OPTIONS] PRIORITY -p PID
```

Common Options:

- -p → Change priority of a process by PID.
- -u USER → Change priority of all processes owned by a user.

Example:

```
renice -10 -p 1234
```

Sets process **1234** to **higher priority (-10)**.

Disk Management Commands:

1. **df -h (Disk Free Space)**

- The df command displays information about disk space usage on file systems.
- The -h flag makes the output human-readable by showing sizes in KB, MB, or GB.
- **Common Options:**
  - -T: Displays file system type
  - -i: Shows inode usage instead of block usage
  - -x <type>: Excludes file systems of a certain type
- **Syntax:** displays disk space usage in a readable format

  ```
  df -h
  ```

2. **du (Disk Usage)**

- The du command estimates and displays the disk usage of files and directories.
- **Syntax:**

  ```
  du [options] [file/directory]
  ```

- **Common Options:**
  - -h: Human-readable format
  - -s: Shows only the total size
  - -a: Displays the size of all files
  - --max-depth=N: Limits the output to N directory levels
- **Example:**

  ```
  du -sh /home/user (shows the total size of /home/user)
  ```

Memory Management Commands:

1. **free (Memory Usage Information)**

- The free command provides information about system memory usage, including RAM and swap.
- **Syntax:**

  ```
  free [options]
  ```

- **Common Options:**
  - -h: Human-readable format
  - -g: Displays output in GB
  - -s <seconds>: Refreshes output at a specified interval
- **Example:** (shows memory usage in MB)

  ```
  free -m
  ```

2. **vmstat (Virtual Memory Statistics)**

   - The vmstat command provides information about system performance, including memory, CPU, and disk activity.
   - **Syntax:**

     ```
     vmstat [interval] [count]
     ```

   - **Common Options:**
     - -s: Displays memory statistics
     - -d: Shows disk statistics
     - -t: Includes timestamps
   - **Example: (updates stats every 5 seconds, 10 times)**

     ```
     vmstat 5 10
     ```

3. **meminfo (Memory Details from /proc)**

   - The cat /proc/meminfo command displays detailed memory usage statistics from the /proc file system.
   - **Syntax:** (outputs detailed memory usage stats)

     ```
     cat /proc/meminfo
     ```

4. **slabtop (Kernel Memory Slabs Information)**

   - The slabtop command provides a real-time view of memory usage by kernel slabs.
   - **Syntax:**

     ```
     Slabtop (displays detailed memory slab information)
     ```

5. **smem (Detailed Memory Usage by Process)**

   - The smem command provides an advanced memory usage breakdown by process.
   - **Syntax:**

     ```
     smem [options]
     ```

   - Common Options:
     - -t: Shows totals
     - -k: Displays values in KB
   - **Example:** (shows memory usage with process names and total memory usage)

     ```
     smem -tk
     ```

Process related Commands:

1. **ps (Process Status)**

- The ps command provides information about currently running processes.
- **Syntax:**

```
ps [options]
```

- **Common Options:**
  - -e: Displays all processes
  - -f: Full format listing
  - -u <user>: Shows processes for a specific user
  - -o <format>: Customizes the output format
- Example: (shows detailed information about all processes)

```
ps aux
```

2. **top (Real-time Process Monitoring)**

- The top command provides a real-time view of system processes, CPU, and memory usage.
- **Syntax:**

```
top [options]
```

- **Common Options:**
  - -o <field>: Sorts by a specified field
  - -d <seconds>: Refresh interval
  - -n <count>: Displays output a fixed number of times
- **Example:** (opens an interactive process monitor)

```
top
```

3. **fuser (Identify Processes Using a File or Socket)**

- The fuser command identifies processes that are using a specified file or socket.
- **Syntax:**

```
fuser [options] <file>
```

- **Common Options:**
  - -k: Kills the processes using the file
  - -m: Shows processes using a mounted file system
- **Example:**

```
fuser /var/log/syslog # shows processes using syslog
```

4. **kill (Terminate Processes)**

- The kill command is used to terminate processes using their process ID (PID).
- **Syntax:**

```
kill [options] <PID>
```

- **Common Options:**
  - -9: Force kill the process
  - -15: Gracefully terminate the process (default)
  - -l: List all signal names
- **Example:** (forcefully terminates process with PID 1234)

```
kill -9 1234
```

5. **nohup (Run a Command in Background & Ignore Hangups)**

- The nohup command allows a process to continue running even after logging out.
- **Syntax:**

```
nohup command &
```

- **Example:** (runs script.sh in the background, ignoring hangups)

```
nohup ./script.sh &
```

System Level commands:

1. **uname**

- The uname command is used to display system information such as the kernel name, version, and hardware details. It is useful for checking system architecture and OS details.
- **Syntax:**

```
uname [OPTION]
```

- **Common Options:**
  - -a : Displays all available system information.
  - -s : Shows only the kernel name.
  - -r : Displays the kernel release version.
  - -v : Prints the kernel version.
  - -m : Shows the machine hardware name.
  - -n : Displays the network node hostname.
- **Example:**

```
uname -a
```

2. **uptime**

- The uptime command is used to check how long the system has been running. It also provides the number of active users and system load averages.
- **Syntax:**

```
uptime
```

- **Common Options:**
  - -p : Displays uptime in a human-readable format.
  - -s : Shows the system's startup time.
- **Example:**

```
uptime -p
```

3. **date**

- The date command is used to display or set the system date and time. It allows formatting and manipulating date values.
- **Syntax:**

```
date [OPTION] [+FORMAT]
```

- **Common Options:**

  - +%Y-%m-%d : Displays the date in "YYYY-MM-DD" format.
  - +%H:%M:%S : Shows the current time in "HH:MM:SS" format.
  - -u : Prints the date/time in Coordinated Universal Time (UTC).

- **Example:**

```
date +"%Y-%m-%d %H:%M:%S"
```

4. **who**

- The who command shows a list of users currently logged into the system.
- **Syntax:**

```
who [OPTION]
```

- **Common Options:**

  - -b : Displays the last system boot time.
  - -H : Prints column headers.
  - -q : Shows only the number of logged-in users.

- **Example:**

```
who -q
```

5. **whoami**

- The whoami command prints the username of the currently logged-in user.
- **Syntax:**

```
whoami
```

6. **which**

- The which command locates the executable path of a given command. It is useful for finding out where a command is installed.
- **Syntax:**

```
which [COMMAND]
```

- **Example:**

```
which python3
```

7. **id**

- The id command prints user and group IDs for a given user.
- **Syntax:**

```
id [OPTION] [USERNAME]
```

- **Common Options:**
  - -u : Shows the user ID (UID).
  - -g : Displays the group ID (GID).
  - -G : Prints all group IDs.
- **Example:**

```
id
```

8. **sudo**

- The sudo command allows a permitted user to execute a command as another user, typically as root.
- **Syntax:**

```
sudo [COMMAND]
```

- **Example:**

```
sudo apt update
```

9. **shutdown**

- The shutdown command is used to power off or restart the system safely.
- **Syntax:**

```
shutdown [OPTION] [TIME]
```

- **Common Options:**
  - -h : Halts the system (shutdown).
  - -r : Reboots the system.
  - now : Executes shutdown immediately.
- **Example:**

```
sudo shutdown -h now
```

10. **reboot**

- The reboot command restarts the system.
- **Syntax:**

```
sudo reboot
```

11. **apt**

- The apt command is a package management tool for Debian-based distributions, used for installing, updating, and removing software packages.
- **Syntax:**

```
sudo apt [COMMAND] [PACKAGE]
```

- **Common Commands:**
  - update : Updates the package lists.
  - upgrade : Installs available package updates.
  - install [PACKAGE] : Installs a specific package.
  - remove [PACKAGE] : Removes a package.
- **Example:**

```
sudo apt install vim
```

12. **yum**

- The yum (Yellowdog Updater, Modified) command is used for package management in RHEL-based Linux distributions. It allows users to install, update, and remove packages.
- **Syntax:**

```
yum [COMMAND] [PACKAGE]
```

- **Common Commands:**
  - install [PACKAGE] : Installs a specified package.
  - update : Updates all installed packages.
  - remove [PACKAGE] : Removes a package.
  - list : Lists all available and installed packages.
- **Example:** Installs the Apache web server package

```
sudo yum install httpd
```

13. **rpm**

- The rpm (Red Hat Package Manager) command is used for managing .rpm packages on RHEL-based systems.
- **Syntax:**

```
rpm [OPTION] [PACKAGE]
```

- **Common Options:**
  - -i : Installs an RPM package.
  - -e : Removes an RPM package.
  - -q : Queries installed packages.

- -U : Upgrades an existing package.

- **Example:** Installs a package with verbose output and hash progress

```
rpm -ivh package.rpm
```

14. **dnf**

- The dnf (Dandified YUM) command is the next-generation package manager for Fedora and RHEL-based distributions.
- **Syntax:**

```
dnf [COMMAND] [PACKAGE]
```

- **Common Commands:**

  - install [PACKAGE] : Installs a package.
  - update : Updates installed packages.
  - remove [PACKAGE] : Uninstalls a package.
  - list : Lists available and installed packages.

- **Example:** Installs the Nginx web server

```
sudo dnf install nginx
```

15. **pacman**

- The pacman command is the package manager for Arch Linux and its derivatives.
- **Syntax:**

```
pacman [OPTION] [PACKAGE]
```

- **Common Options:**

  - -S [PACKAGE] : Installs a package.
  - -R [PACKAGE] : Removes a package.
  - -Syu : Updates the system (syncs package database and upgrades packages).
  - -Q : Queries installed packages.

- **Example:** Installs the Firefox browser.

```
sudo pacman -S firefox
```

16. portage

- The portage system is the package manager for Gentoo Linux, managed through the emerge command.
- **Syntax:**

```
emerge [OPTIONS] [PACKAGE]
```

- **Common Options:**
  - --ask : Asks before performing actions.
  - --update : Updates installed packages.
  - --depclean : Removes unused dependencies.
  - --search [PACKAGE] : Searches for a package.

- **Example:** Asks for confirmation before installing Vim

```
sudo emerge --ask vim
```

User and Group Management Commands:

1. **sudo**

- The sudo (Super User Do) command allows a permitted user to execute a command as the superuser (root) or another user, as specified in the /etc/sudoers file. It is commonly used for executing administrative tasks that require elevated privileges.
- **Syntax:**

```
sudo [OPTIONS] COMMAND
```

- **Common Options:**
  - -s → Starts a shell with root privileges.
  - -u [username] → Runs the command as a specified user.
  - -l → Lists the allowed and forbidden commands for the current user.
- **Example:** Runs the package update command with root privileges

```
sudo apt update
```

2. **useradd**

The useradd command is used to create a new user account in a Linux system. It modifies the system account files to add a new user. The user's home directory and other attributes can be customized using various options.

- **Syntax:**

```
useradd [OPTIONS] USERNAME
```

- **Common Options:**
  - -m → Creates a home directory for the new user.
  - -d [directory] → Specifies a custom home directory.
  - -s [shell] → Assigns a specific shell to the user.
  - -G [group] → Adds the user to a specific group.
- **Example:** Creates a user 'john' with a home directory, bash shell, and adds to 'developers' group

```
useradd -m -s /bin/bash -G developers john
```

3. **whoami**

- The whoami command displays the username of the currently logged-in user. It is useful for verifying which user account is being used in a terminal session.
- **Example:** Outputs the current username

```
whoami
```

4. **su**

- The su (Substitute User) command allows switching to another user account without logging out. If no username is specified, it defaults to switching to the root user.
- **Syntax:**

```
su [OPTIONS] [USERNAME]
```

- **Common Options:**
  - - → Provides an environment similar to a fresh login.
  - -c [command] → Executes a single command as the specified user.
- **Example:** Switches to user ‘vishal’ with a fresh environment

```
su - vishal
```

5. **passwd**

- The passwd command is used to change the password of a user account. It can also be used by the root user to change passwords for other users.
- **Syntax:**

```
passwd [OPTIONS] [USERNAME]
```

- **Common Options:**
  - -e → Forces the user to change the password at the next login.
  - -l → Locks the user account.
  - -u → Unlocks the user account.
- **Example:** Changes the password for user 'vishal’

```
passwd vishal
```

6. **userdel**

- The userdel command is used to delete a user account from the system. It removes the user's account but does not delete their home directory by default.
- **Syntax:**

```
userdel [OPTIONS] USERNAME
```

- **Common Options:**
  - -r → Removes the user's home directory and mail spool.
- **Example:** Deletes user ‘vishal’ and removes their home directory

```
userdel -r vishal
```

7. **usermod**

- The usermod command is used to modify an existing user account in a Linux system. It allows changing the user's details such as group membership, login shell, and home directory.

- **Syntax:**

```
usermod [OPTIONS] USERNAME
```

- **Common Options:**
  - -c [comment] → Adds or modifies the user's description.
  - -d [directory] → Changes the user's home directory.
  - -m → Moves the user's files to the new home directory.
  - -G [group] → Adds the user to a supplementary group.
  - -aG [group] → Appends the user to a new group without removing them from existing groups.
  - -s [shell] → Changes the user's login shell.
  - -L → Locks the user account.
  - -U → Unlocks the user account.

- **Example:** Add user john to the sudo group:

```
usermod -aG sudo john
```

8. **groupadd**

- The groupadd command is used to create a new group in the system. Groups are used to manage permissions for multiple users efficiently.
- **Syntax:**

```
groupadd [OPTIONS] GROUPNAME
```

- **Common Options:**
  - -g [GID] → Assigns a specific Group ID (GID) to the new group.

- **Example:** Creates a new group named 'developers'

```
groupadd developers
```

9. **gpasswd**

- The gpasswd -a command is used to add a user to a specific group. It modifies the /etc/group file.
- The gpasswd -d command is used to remove a user from a specific group.
- **Syntax:**

```
gpasswd -a USERNAME GROUPNAME
```

- **Example:** Adds user 'vishal' to the 'developers' group

```
gpasswd -a vishal developers
```

10. **gpasswd -m**

- The gpasswd -m command is used to assign multiple users to a group at once.
- **Syntax:**

```
gpasswd -m "USER1,USER2,..." GROUPNAME
```

- **Example:** Adds ‘vishal and ‘sam’ to the 'developers' group

```
gpasswd -m "vishal, sam" developers
```

11. **groupdel**

- The groupdel command is used to remove a group from the system. It deletes the group but does not affect users who were part of it.
- **Syntax:**

```
groupdel GROUPNAME
```

- **Example:** Deletes the 'developers' group

```
groupdel developers
```

File Permission Commands:

1. **umask**

- The umask (User File Creation Mode Mask) command sets default permissions for newly created files and directories. It determines which permission bits should be masked (disabled) by default. The umask value is subtracted from the system's default permissions (usually 666 for files and 777 for directories) to determine the final permission set.
- **Syntax:**

```
umask [OPTION] [MASK]
```

- **Common Options:**
  - No argument: Displays the current umask value.
  - [MASK] → Sets a new umask value using octal notation.
- **Example:**

```
umask 022  # Sets default permissions to 755 for directories and 644 for files
umask      # Displays the current umask value
```

2. **ls -l**

- The ls -l command lists files and directories in long format, displaying detailed information including permissions, ownership, size, and modification date.
- **Syntax:**

```
ls -l [OPTIONS] [FILE/DIRECTORY]
```

- **Common Options:**
  - -a → Shows hidden files.
  - -h → Displays file sizes in human-readable format.
  - -t → Sorts by modification time.
  - -r → Reverses the order of the listing.
- **Example:**

```
ls -l /home/user  # Lists all files and directories in the home folder with details
ls -lah  # Lists all files including hidden ones in human-readable format
```

3. **chmod**

- The chmod (Change Mode) command is used to change the permissions of files and directories. It can modify read (r), write (w), and execute (x) permissions for the owner, group, and others.
- **Syntax:**

```
chmod [OPTIONS] MODE FILE/DIRECTORY
```

- **Common Options:**
  - -R → Recursively changes permissions for all files in a directory.
  - -v → Shows verbose output of permission changes.
  - MODE → Can be numeric (e.g., 755) or symbolic (e.g., u+rwx, g-w).
- **Example:**

```
chmod 755 script.sh # Assigns rwxr-xr-x permissions to script.sh
chmod -R 700 /secure_folder # Assigns full access to owner and no access to others
```

4. **chown**

- The chown (Change Owner) command changes the ownership of files and directories, allowing you to assign a new owner or group.
- **Syntax:**

```
chown [OPTIONS] USER:[GROUP] FILE/DIRECTORY
```

- **Common Options:**
  - -R → Recursively changes ownership of all files in a directory.
  - --reference=[file] → Uses ownership from a reference file.
- **Example:**

```
chown vishal script.sh # Changes owner of script.sh to 'vishal'
chown -R vishal:developers /project # Changes owner to 'vishal' and group to 'developers' for all files in /project
```

5. **chgrp**

- The chgrp (Change Group) command changes the group ownership of a file or directory.
- **Syntax:**

```
chgrp [OPTIONS] GROUP FILE/DIRECTORY
```

- **Common Options:**

- -R → Recursively changes group ownership for all files in a directory.
- --reference=[file] → Uses group ownership from a reference file.

- **Example:**

```
chgrp developers project.txt # Changes the group of project.txt to 'developers'
chgrp -R admins /secure_folder # Assigns group 'admins' to all files in /secure_folder
```

File Compression Commands:

1. **zip**
   - The zip command is used to compress files or directories into a .zip archive. It reduces the file size to save disk space or makes it easier to transfer multiple files together.
   - **Syntax**

     ```
     zip [options] zipfile file1 file2 ...
     ```

   - **Common Options**

     - -r: Recursively include files in directories.
     - -e: Encrypt the archive with a password.
     - -q: Quiet mode, suppresses the output.
     - -9: Use the maximum compression.

   - **Example:** To zip a directory named docs into a file named docs.zip

     ```
     zip -r docs.zip docs
     ```

2. **gunzip**
   - The gunzip command is used to decompress files that have been compressed using the gzip command. It works by removing the .gz extension and restoring the original file.
   - **Syntax**

     ```
     gunzip [options] file.gz
     ```

   - **Common Options**

     - -c: Write the decompressed data to standard output instead of replacing the original file.
     - -d: Force decompression (useful for compressed .zip files).
     - -f: Force decompression even if the file is already decompressed.

   - **Example**: To decompress a file file.gz

     ```
     gunzip file.gz
     ```

3. **gzip**
   - The gzip command is used to compress files in .gz format. It is typically used to reduce the size of files for storage or transmission.
   - **Syntax**

     ```
     gzip [options] file1 file2 ...
     ```

   - **Common Options**

     - -d: Decompress the file.
     - -v: Display compression information.
     - -r: Compress files recursively.

- -1 to -9: Set the compression level, where -1 is the fastest and -9 is the best compression.

- **Example:** To compress a file named file.txt

```
gzip file.txt
```

4. **tar**

- The tar command is used to create, modify, and extract .tar archives. It is commonly used for packaging files together, often used in backup processes.
- **Syntax**

```
tar [options] archive.tar file1 file2 ...
```

- **Common Options**

  - -c: Create a new archive.
  - -x: Extract an archive.
  - -f: Specify the archive file name.
  - -v: Verbosely list files as they are archived or extracted.
  - -z: Compress the archive using gzip after creation.
  - -j: Compress the archive using bzip2.

- **Example:** To create a .tar archive of a directory docs

```
tar -cvf docs.tar docs
```

5. **untar**

- The untar command is essentially the process of extracting files from a .tar archive, often used as a synonym for the tar -x command.
- **Syntax**

```
tar -x [options] -f archive.tar
```

- **Common Options**

  - -x: Extract the files from the archive.
  - -v: Verbosely list files during extraction.
  - -f: Specify the archive file name.
  - -z: Decompress the archive using gzip during extraction.
  - -j: Decompress the archive using bzip2.

- **Example:** To extract files from a docs.tar archive

```
tar -xvf docs.tar
```

File Transfer Commands:

1. **scp**

- The scp (secure copy) command is used to securely transfer files between two systems over a network. It uses SSH for data transfer, ensuring that the data is encrypted during transit.
- **Syntax**

  ```
  scp [options] source_file destination
  ```

- **Common Options**

  - -r: Recursively copy entire directories.
  - -P: Specify the port to connect to on the remote host (note that it is uppercase).
  - -i: Specify a private key for authentication.
  - -v: Enable verbose mode for debugging.
  - -C: Enable compression during transfer.

- **Example**
  To copy a file file.txt from your local system to a remote system
  ```
  scp file.txt user@remote_host:/path/to/destination/
  ```

  To copy an entire directory recursively
  ```
  scp -r docs user@remote_host:/path/to/destination/
  ```

  ```
  scp -i "c:\Users\Vishal\Downloads\linux-for-devops.pem" c:\Users\Vishal\Downloads\linux-for-devops.pem ubuntu@ec2-15-27-22-252.ap-south-1.compute.amazonaws.com:/home/ubuntu/keys
  ```

  ```
  scp -i "c:\Users\Vishal\Downloads\linux-for-devops.pem" c:\Users\Vishal\Downloads\2025\* ubuntu@ec2-15-27-22-252.ap-south-1.compute.amazonaws.com:/home/ubuntu/keys
  ```

2. rsync

- The rsync command is used for fast and efficient file copying and synchronization between two locations. It can be used locally or remotely and can handle incremental file transfers, ensuring that only changes are copied, which speeds up subsequent syncs.
- **Syntax**

  ```
  rsync [options] source destination
  ```

- **Common Options**

  - -r: Recursively copy directories.
  - -a: Archive mode, preserves permissions, symbolic links, etc.
  - -z: Compress file data during the transfer.
  - -v: Verbose mode, shows detailed output.
  - -u: Skip files that are newer on the destination.
  - --delete: Delete files in the destination that no longer exist in the source.

- **Example**
  To synchronize the contents of the docs directory to a remote system
  ```
  rsync -av docs/ user@remote_host:/path/to/destination/
  ```

  To copy a file and compress during transfer
  ```
  rsync -z file.txt user@remote_host:/path/to/destination/
  ```

Network Management Commands:

1. ping

The ping command is used to test the connectivity between a local machine and a remote host by sending ICMP (Internet Control Message Protocol) echo request packets and measuring the response time. It helps diagnose network issues such as packet loss and latency.

Syntax:

```
ping [options] destination
```

Common Options:

- -c <count>: Specifies the number of echo requests to send.
- -i <interval>: Sets the time interval between sending packets.
- -t <ttl>: Sets the Time To Live value for packets.
- -s <size>: Specifies the packet size in bytes.
- -q: Outputs only summary statistics.

Example:

```
ping -c 4 google.com
```

This sends 4 ICMP echo requests to google.com and displays the response time.

2. netstat

The netstat command displays network connections, routing tables, interface statistics, masquerade connections, and multicast memberships. It is useful for troubleshooting network issues and monitoring active connections.

Syntax:

```
netstat [options]
```

Common Options:

- -a: Shows all active connections and listening ports.
- -t: Displays TCP connections.
- -u: Displays UDP connections.
- -n: Shows numerical addresses instead of resolving hostnames.
- -p: Displays the process ID and name associated with the connection.

Example:

```
netstat -tulnp
```

This displays all active listening TCP and UDP ports along with the process IDs.

3. ifconfig

The ifconfig command is used to configure, manage, and display network interface parameters in Linux. It is commonly used for checking IP addresses, setting up network interfaces, and enabling/disabling them.

Syntax:

```
ifconfig [interface] [options]
```

Common Options:

- -a: Displays all interfaces, including those that are down.
- up: Activates a network interface.
- down: Disables a network interface.
- netmask <mask>: Sets a network mask for the interface.
- mtu <size>: Sets the maximum transmission unit (MTU) for an interface.

Example:

```
ifconfig eth0
```

This displays information about the eth0 network interface.

4. traceroute

The traceroute command tracks the route packets take to reach a network destination. It helps diagnose network latency and routing issues by displaying each hop along the path.

Syntax:

```
traceroute [options] destination
```

Common Options:

- -m <max_ttl>: Sets the maximum number of hops.
- -n: Displays only numerical IP addresses without resolving hostnames.
- -q <queries>: Specifies the number of probes per hop.
- -w <wait_time>: Sets the wait time for each response.

Example:

```
traceroute google.com
```

This traces the route packets take to reach google.com.

5. tracepath

The tracepath command traces the path to a remote host, similar to traceroute, but does not require root privileges. It provides MTU discovery along with hop information.

Syntax:

```
tracepath [options] destination
```

Common Options:

- -n: Displays numerical IP addresses.
- -b: Displays both IP addresses and hostnames.
- -l <size>: Sets the initial packet length.

Example:

```
tracepath google.com
```

This traces the network path to google.com.

6. mtr (My Trace Route)

The mtr command is a combination of ping and traceroute, providing real-time network diagnostic information.

Syntax:

```
mtr [options] destination
```

Common Options:

- -r: Generates a report instead of interactive mode.
- -n: Disables hostname resolution.
- -c <count>: Specifies the number of pings per hop.

Example:

```
mtr -r google.com
```

This generates a report of the network path to google.com.

7. nslookup

The nslookup command queries DNS servers to obtain domain name or IP address mapping information.

Syntax:

```
nslookup [options] [hostname]
```

Common Options:

- -type=<record>: Queries a specific DNS record type (e.g., A, MX, TXT).
- -debug: Provides detailed debugging information.
- -query=<type>: Specifies the type of DNS record to retrieve.

Example:

```
nslookup google.com
```

This retrieves the IP address associated with google.com.

8. telnet

Telnet is a command-line network protocol used for interactive text-based communication with a remote host over a TCP/IP network. It provides a way to connect to remote systems and issue commands as if you were physically present on that machine. Telnet operates on port 23 by default and does not encrypt the transmitted data, making it vulnerable to security threats. Due to its lack of encryption, Telnet has largely been replaced by SSH for secure remote access.

Syntax:

```
telnet [hostname or IP address] [port]
```

Common Options:

- -8 : Allows an 8-bit data path.
- -E : Disables escape character functionality.
- -l user : Logs in as the specified user.
- -a : Attempts automatic login.

Example:

```
telnet 192.168.1.1 23
```

This command initiates a Telnet session to the IP 192.168.1.1 on port 23.

9. hostname

The hostname command is used to display or set the system's hostname, which is the label assigned to a machine on a network. Hostnames help identify devices in a networked environment and can be used to configure DNS settings. This command is useful for checking the hostname of a machine and modifying it as needed.

Syntax:

```
hostname [option]
```

Common Options:

- -i : Displays the IP address associated with the hostname.
- -d : Shows the domain name of the system.
- -f : Displays the fully qualified domain name (FQDN).
- -s : Shows only the short hostname.

Example:

```
hostname -I
```

This command retrieves the system's IP addresses.

10. ip

The ip command is a powerful tool used to manage network interfaces, configure IP addresses, modify routing tables, and view detailed network information. It is a modern replacement for the older ifconfig command.

Syntax:

```
ip [options] OBJECT {COMMAND}
```

Common Options:

- addr : Shows or configures IP addresses.
- link : Displays or configures network interfaces.
- route : Displays or modifies the system's routing table.
- -s : Shows network statistics.

Example:

```
ip addr show
```

This command displays the IP address assigned to all interfaces.

11. iwconfig

The iwconfig command is used to configure wireless network interfaces, allowing users to set parameters such as SSID, mode, frequency, and encryption type for Wi-Fi connections. Unlike ifconfig, which manages wired connections, iwconfig is specific to wireless networks.

Syntax:

```
iwconfig [interface] [options]
```

Common Options:

- essid <name> : Sets the ESSID of the wireless network.
- mode <mode> : Configures the operating mode (e.g., Managed, Ad-hoc, Monitor).
- freq <frequency> : Sets the frequency or channel number.
- txpower <value> : Adjusts the transmission power of the wireless interface.

Example:

```
iwconfig wlan0 essid "MyWiFi"
```

This command sets the ESSID of interface wlan0 to MyWiFi.

12. ss

The ss command is a modern replacement for netstat and is used to display socket connection details. It provides information about active TCP, UDP, and raw sockets, making it useful for network troubleshooting and performance monitoring.

Syntax:

```
ss [options]
```

Common Options:

- -t : Displays TCP connections.
- -u : Shows UDP connections.
- -l : Lists listening ports.
- -p : Displays the process using the sockets.
- -n : Shows numeric addresses instead of resolving hostnames.

Example:

```
ss -tulnp
```

This command shows all active listening ports along with the process details.

13. arp

The arp (Address Resolution Protocol) command is used to view and manipulate the ARP table, which maps IP addresses to MAC addresses. ARP is essential for communication within a local network as it resolves IP addresses to physical hardware addresses.

Syntax:

```
arp [options]
```

Common Options:

- -a : Displays all ARP table entries.
- -d <host> : Deletes an ARP entry for a specific host.
- -s <IP> <MAC> : Manually sets an ARP entry.

Example:

```
arp -a
```

This command lists all ARP table entries.

14. dig

The dig (Domain Information Groper) command is used to query DNS name servers and retrieve information about domain names. It is widely used for troubleshooting DNS-related issues.

Syntax:

```
dig [options] [domain]
```

Common Options:

- +short : Displays concise output.
- -x <IP> : Performs a reverse DNS lookup.
- @<server> : Uses a specific DNS server for the query.

Example:

```
dig google.com
```

This command retrieves DNS records for google.com.

15. nc (netcat)

The nc (Netcat) command is a versatile networking tool used for reading and writing data over network connections using TCP or UDP. It can be used for port scanning, file transfer, or even setting up simple network-based chat applications.

Syntax:

```
nc [options] [destination] [port]
```

Common Options:

- -l : Listens for incoming connections.
- -u : Uses UDP instead of TCP.
- -v : Enables verbose output.
- -z : Scans for open ports without sending data.

Example:

```
nc -v -z 192.168.1.1 80
```

This command scans if port 80 is open on 192.168.1.1.

16. whois

The whois command is used to query information about domain names, IP addresses, and autonomous systems. It retrieves data from the WHOIS database, which contains information like the domain owner, the registrar, and the registration date.

Syntax:

```
whois [OPTION] DOMAIN_NAME
```

Common Options:

- -h HOST: Query the WHOIS server at the specified host.
- -p PORT: Connect to the WHOIS server on the specified port.
- -i: Print information about a specific IP address.
- -r: Show raw WHOIS output.

Example:

```
whois example.com
```

This command will fetch WHOIS information for the domain example.com.

17. ifplugstatus

The ifplugstatus command is used to check whether a network interface is plugged in or not. It is commonly used on systems with network interfaces to determine if the cable is physically connected to the network.

Syntax:

```
ifplugstatus [interface]
```

Common Options:

- -h: Displays help.
- interface: Specify the network interface to check, such as eth0 or wlan0.

Example:

```
ifplugstatus eth0
```

This command will check if the eth0 interface is connected to the network.

18. iptables

iptables is a powerful command-line tool used for configuring the Linux kernel firewall. It enables users to set up rules for traffic filtering, network address translation (NAT), and more. It is primarily used to secure Linux-based systems.

Syntax:

```
iptables [OPTION] [CHAIN] [MATCH] [TARGET]
```

Common Options:

- -A CHAIN: Append a rule to the end of a chain.
- -I CHAIN: Insert a rule at the beginning of a chain.
- -D CHAIN: Delete a rule from a chain.
- -L: List the rules in the current chain.
- -F: Flush the rules in a chain.
- -t TABLE: Specify the table to use (e.g., filter, nat).

Example:

```
iptables -A INPUT -p tcp --dport 80 -j ACCEPT
```

This command appends a rule to allow incoming TCP traffic on port 80 (HTTP) to the INPUT chain.

19. nmap

The nmap command is a network exploration and security auditing tool. It is used to scan networks, discover hosts and services, and identify open ports. It can also be used for security assessment by detecting vulnerabilities in a network.

Syntax:

```
nmap [OPTIONS] [TARGET]
```

Common Options:

- -sP: Ping scan to discover hosts.
- -sT: Perform a TCP connect scan.
- -sU: Perform a UDP scan.
- -p PORTS: Specify which ports to scan (e.g., -p 80,443).
- -O: Enable OS detection.
- -v: Increase verbosity.

Example:

```
nmap -sP 192.168.1.0/24
```

This command performs a ping scan on the entire subnet 192.168.1.0/24 to discover live hosts.

20. route

The route command is used to display or manipulate the IP routing table in Linux. It helps configure the paths that network traffic takes to reach destinations, set up static routes, and troubleshoot network issues.

Syntax:

```
route [OPTION]
```

Common Options:

- -n: Show numerical addresses instead of resolving hostnames.
- add: Add a new route to the routing table.
- del: Remove a route from the routing table.
- -A FAMILY: Specify the address family (e.g., inet for IPv4).

Example:

```
route -n
```

This command displays the routing table in numeric format.

Web Requests Commands:

1. curl

The curl command is a tool for transferring data to or from a server using various protocols like HTTP, FTP, and more. It is commonly used for interacting with APIs, downloading files, and sending data over the network.

Syntax:

```
curl [OPTIONS] [URL]
```

Common Options:

- -O: Save the output to a file with the same name as the remote file.
- -L: Follow redirects.
- -X: Specify the HTTP method to use (e.g., -X GET, -X POST).
- -d DATA: Send data in the body of the request (used with POST).
- -H "Header: Value": Add headers to the request.

Example:

```
curl -O https://example.com/file.txt
```

This command will download the file from the URL https://example.com/file.txt and save it with the same name.

2. jq

jq is a lightweight and flexible command-line JSON processor. It is used for parsing, filtering, and modifying JSON data. It can handle large JSON datasets and is useful for working with APIs that return JSON responses.

Syntax:

```
jq [OPTIONS] 'FILTER' [FILE]
```

Common Options:

- -r: Output raw strings (without quotes).
- -f FILE: Read filter expressions from the specified file.
- .: This represents the entire JSON data.
- []: Filter elements of an array.
- {}: Filter fields of an object.

Example:

```
curl -s https://api.example.com/data | jq '.name'
```

This command fetches JSON data from the API and filters the value of the name field.

3. wget

The wget command is used to download files from the web. It supports multiple protocols like HTTP, HTTPS, and FTP. Unlike curl, wget is designed to download entire websites or files with more advanced options like recursive downloading and resuming interrupted downloads.

Syntax:

```
wget [OPTIONS] [URL]
```

Common Options:

- -O FILE: Save the output to a specified file.
- -r: Download files recursively.
- -c: Resume a download if it was interrupted.
- -P DIR: Save the file in a specified directory.
- -np: Prevent downloading files from parent directories.

Example:

```
wget -O file.txt https://example.com/file.txt
```

This command downloads the file from the URL https://example.com/file.txt and saves it as file.txt.

4. watch

The watch command is used to execute a program periodically, displaying the output in the terminal. It is often used to monitor the real-time output of a command, such as the status of a process, system resource usage, or the contents of a file.

Syntax:

```
watch [OPTIONS] COMMAND
```

Common Options:

- -n SECONDS: Run the command every specified number of seconds.
- -d: Highlight differences between updates.
- -t: Turn off the header (e.g., time and command).
- -g: Exit the command when the output changes.

Example:

```
watch -n 5 df -h
```

This command runs the df -h command every 5 seconds to show disk space usage in a human-readable format.

Text Processing & Manipulation Commands:

1. awk

awk is a powerful text processing tool used for pattern scanning, processing, and reporting. It is commonly used to extract, manipulate, and analyze text from structured data such as logs, CSV files, and reports. It operates on a line-by-line basis and divides each line into fields.

Syntax:

```
awk [OPTIONS] 'pattern { action }' FILE
```

Common Options:

- -F delimiter – Specifies a field separator (default is whitespace).
- -v var=value – Assigns a value to a variable.
- NR – Represents the current record (line) number.
- NF – Represents the total number of fields in a line.

Example:

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

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

2. sed

sed (Stream Editor) is used for text manipulation and transformation. It reads input line by line, applies operations such as search, replace, insert, and delete, and outputs the modified text. It is commonly used for batch editing in scripts.

Syntax:

```
sed [OPTIONS] 'COMMAND' FILE
```

Common Options:

- -i – Edit the file in place (modifies original file).
- -e – Allows multiple expressions to be applied.
- s/pattern/replacement/ – Substitutes a pattern with a replacement.
- -n – Suppresses automatic printing of lines.

Example:

```
sed 's/Linux/Unix/g' file.txt
```

This command replaces all occurrences of "Linux" with "Unix" in file.txt.

3. grep

grep (Global Regular Expression Print) is used to search for specific patterns in files or output streams. It scans line-by-line and prints matching lines based on the given pattern.

Syntax:

```
grep [OPTIONS] "pattern" FILE
```

Common Options:

- -i – Case-insensitive search.
- -v – Invert match (show lines that do not match).
- -r – Recursively search in directories.
- -n – Show line numbers of matches.
- -E – Use extended regular expressions.

Example:

```
grep -i "error" logfile.txt
```

This command searches for the word "error" (case-insensitive) in logfile.txt.

Log Analysis & Monitoring Commands:

1. Journalctl:

journalctl is used to query system logs managed by systemd.

Syntax:

```
journalctl [OPTIONS]
```

Common Options:

- -f – Follow real-time logs.
- -u SERVICE – Show logs for a specific service.
- --since "YYYY-MM-DD HH:MM:SS" – Show logs since a specific time.

Example: Shows logs for the nginx service from the last hour.

```
journalctl -u nginx --since "2024-08-17 10:00:00"
```

2. dmesg

dmesg displays kernel-related logs, useful for diagnosing hardware issues.

Syntax:

```
dmesg [OPTIONS]
```

Common Options:

- -H – Human-readable format.
- -T – Show timestamps.
- -w – Follow logs in real time.

Example: Displays kernel logs related to USB devices.

```
dmesg -T | grep "USB"
```

3. logrotate

logrotate is used for managing log files by rotating, compressing, and deleting old logs automatically.

Syntax:

```
logrotate [OPTIONS] CONFIG_FILE
```

Common Options:

- -d – Debug mode (shows what will happen without executing).
- -f – Force log rotation regardless of conditions.
- -s STATEFILE – Specify a state file to track rotations.

- -v – Verbose mode (displays detailed output).
- --log [LOGFILE] – Write logrotate's output to a specified log file.

Example: Simulates a log rotation without making changes.

```
logrotate -d /etc/logrotate.conf
```

4. stat

stat displays detailed metadata of a log file, including modification time.

Syntax:

```
stat FILE
```

Example: Displays metadata of the system log file.

```
stat /var/log/syslog
```

5. find: find is used to locate log files based on size, modification time, or name.

Syntax:

```
find DIRECTORY [OPTIONS]
```

Common Options:

- -name "pattern" – Search by name.
- -mtime -N – Find files modified in the last N days.
- -size +N – Find files larger than N bytes.

Example: Finds log files modified in the last 7 days.

```
find /var/log -name "*.log" -mtime -7
```

6. cut: cut is used to extract specific columns from log entries.

Syntax:

```
cut -d' ' -fN FILE
```

Common Options:

- -d DELIM – Specifies the delimiter (default is tab).
- -f LIST – Selects specific fields (columns) to display.
- -c LIST – Cuts based on character positions.
- --complement – Shows all fields except the selected ones.

Example: Extracts the first and fifth column from system logs.

```
cut -d' ' -f1,5 /var/log/syslog
```

Linux Volume Management:

1. fdisk

fdisk is used to create, delete, and manipulate disk partitions on MBR (Master Boot Record) partitioned disks.

Syntax:

```
fdisk [OPTIONS] DEVICE
```

Common Options:

- -l – List all available partitions.
- -n – Create a new partition.
- -d – Delete a partition.
- -p – Display partition table.

Example:

```
fdisk /dev/sdb
```

Starts partition management for the /dev/sdb disk.

2. parted

parted is an alternative to fdisk, used to manage partitions, particularly for **GPT (GUID Partition Table)** disks.

Syntax:

```
parted [OPTIONS] DEVICE
```

Common Options:

- mklabel gpt – Create a GPT partition table.
- mkpart – Create a new partition.
- print – Display the partition table.

Example:

```
parted /dev/sdb mklabel gpt
```

Formats /dev/sdb with a GPT partition table.

3. mkfs

mkfs is used to create a new file system on a partition.

Syntax:

```
mkfs -t FILESYSTEM DEVICE
```

Common Options:

- -t – Specify the file system type (ext4, xfs, btrfs, etc.).

Example:

```
mkfs -t ext4 /dev/sdb1
```

Creates an ext4 file system on the /dev/sdb1 partition.

4. mount

mount is used to attach a file system to a directory.

Syntax:

```
mount DEVICE MOUNTPOINT
```

Common Options:

- -t – Specify the file system type.
- -o – Specify mount options (e.g., ro for read-only).

Example:

```
mount /dev/sdb1 /mnt
```

Mounts /dev/sdb1 to the /mnt directory.

5. umount

umount unmounts a mounted file system.

Syntax:

```
umount DEVICE or MOUNTPOINT
```

Example:

```
umount /mnt
```

Unmounts the file system from /mnt.

6. lsblk

lsblk lists information about block devices, including disks and partitions.

Syntax:

```
lsblk [OPTIONS]
```

Common Options:

- -f – Show file system details.
- -o NAME,SIZE,FSTYPE,MOUNTPOINT – Display custom columns.

Example:

```
lsblk -f
```

Shows file system details of all block devices.

7. blkid

blkid retrieves information about block devices, such as UUIDs and file system types.

Syntax:

```
blkid DEVICE
```

Example:

```
blkid /dev/sdb1
```

Displays the UUID and file system type of /dev/sdb1.

8. pvcreate

pvcreate initializes a physical volume for LVM.

Syntax:

```
pvcreate DEVICE
```

Example:

```
pvcreate /dev/sdb1
```

Initializes /dev/sdb1 as a physical volume for LVM.

9. pvs

pvs displays information about physical volumes.

Syntax:

```
pvs
```

Shows all physical volumes in the system.

10. vgcreate

vgcreate creates a new volume group in LVM.

Syntax:

```
vgcreate VG_NAME DEVICE
```

Example:

```
vgcreate my_vg /dev/sdb1
```

Creates a volume group named my_vg using /dev/sdb1.

11. vgs

vgs displays information about volume groups.

Syntax:

```
vgs
```

Shows all volume groups on the system.

12. lvcreate

lvcreate is used to create logical volumes inside a volume group.

Syntax:

```
lvcreate -L SIZE -n LV_NAME VG_NAME
```

Common Options:

- -L SIZE – Specify the size of the logical volume.
- -n NAME – Specify the logical volume name.

Example:

```
lvcreate -L 10G -n my_lv my_vg
```

Creates a 10GB logical volume named my_lv in the my_vg volume group.

13. lvs

lvs displays information about logical volumes.

Syntax:

```
lvs
```

Shows all logical volumes on the system.

14. lvextend

lvextend increases the size of a logical volume.

Syntax:

```
lvextend [OPTIONS] -L [+]SIZE LV_PATH
```

Example:

```
lvextend -L +5G /dev/my_vg/my_lv
```

Increases the logical volume my_lv by 5GB.

15. resize2fs

resize2fs resizes an ext-based file system to match the new size of a logical volume.

Syntax:

```
resize2fs [OPTIONS] DEVICE
```

Common Options:

- -p → Show progress during resizing.
- SIZE → Resize the filesystem to a specific size (e.g., resize2fs /dev/my_vg/my_lv 10G).

Example:

```
resize2fs /dev/my_vg/my_lv
```

Resizes the file system to use the full space of my_lv.

16. lvremove

lvremove deletes a logical volume.

Syntax:

```
lvremove LV_PATH
```

Example:

```
lvremove /dev/my_vg/my_lv
```

Removes the logical volume my_lv.

17. vgremove

vgremove deletes a volume group.

Syntax:

```
vgremove VG_NAME
```

Example:

```
vgremove my_vg
```

Deletes the volume group my_vg.

18. pvremove

pvremove removes a physical volume from LVM.

Syntax:

```
pvremove DEVICE
```

Example:

```
pvremove /dev/sdb1
```

Removes /dev/sdb1 as a physical volume.

19. df

df displays available and used disk space on file systems.

Syntax:

```
df [OPTIONS]
```

Common Options:

- -h – Human-readable format.
- -T – Show file system type.

Example:

```
df -h
```

Shows disk usage in human-readable format.

20. du

du shows the disk usage of files and directories.

Syntax:

```
du [OPTIONS] FILE/DIRECTORY
```

Common Options:

- -h – Human-readable format.
- -s – Display total size only.

Example:

```
du -sh /var/log
```

Shows the total disk space used by /var/log.

Service Management Commands:

1. systemctl

The systemctl command is used to manage system services in Linux. It interacts with the systemd service manager to start, stop, restart, enable, disable, and check the status of services. It is commonly used in modern Linux distributions.

Syntax:

```
systemctl [OPTIONS] COMMAND SERVICE_NAME
```

Common Options

- start – Starts a service.
- stop – Stops a service.
- restart – Restarts a service.
- reload – Reloads configuration without stopping the service.
- status – Shows the current status of a service.
- enable – Enables a service to start at boot.
- disable – Disables a service from starting at boot.
- is-active – Checks if a service is active.
- is-enabled – Checks if a service is enabled at boot.

Example:

```
systemctl start nginx
```

Linux documentation commands:

1. help

The help command in Linux provides built-in documentation for shell commands that are not standalone binaries but are instead shell built-ins. These built-ins are commands integrated into the shell itself (like cd, echo, exit, etc.), and help allows users to quickly access their syntax, available options, and usage details.

Unlike man or info, which provide documentation for external programs, help is used specifically for shell built-in commands. It is mainly used in Bash and some other shells that support built-in command documentation.

Syntax:

```
help [OPTION] [BUILTIN_COMMAND]
```

- [OPTION] – Specifies options that modify the output.
- [BUILTIN_COMMAND] – The shell built-in command for which help is needed.

Common Options:

- -d → Displays a short description of the command.
- -m → Displays help information in a man-page style format.
- -s → Displays only a short syntax summary.

Examples:

```
help cd          # Shows usage details, options, and description of the cd command.
help -s exit     # Displays only the syntax of the exit command.
help -d echo     # Provides a brief one-line description of the echo command.
help -m alias    # Displays help for alias in a structured format similar to man.
```

2. man

The man (manual) command is used to display the user manual for commands and programs in Linux. It provides detailed documentation, including command usage, options, and descriptions. The manual is divided into multiple sections, such as general commands, system calls, and library functions.

Syntax:

```
man [SECTION] command_name
```

- [SECTION] → Specifies the manual section to view (optional).
- command_name → The command or program for which help is needed.

Common Options:

- -k keyword → Searches the manual pages for a specific keyword.
- -f command_name → Displays a short description of the command.
- -a → Shows all manual pages for a command.
- -P pager → Uses a specified pager (like less or more) to display the manual.

Examples:

```
man ls              # View the manual page for the ls command:
man -k network      # Displays a list of commands related to networking.
man -f grep         # Display a short description of the grep command:
man -a printf       # View all manual pages for the printf command:
```

3. info

The info command is another documentation tool that provides more detailed and structured information compared to man. It presents information in a node-based, navigable format, often with hyperlinks to related topics.

Syntax:

```
info [command_name]
```

- command_name → The command for which information is needed.

Common Options:

- --subnodes → Displays all subtopics in a single page.
- --output=file → Saves the output to a file.
- --usage → Displays usage information for the info command.

Examples:

```
info ls                     # View the info page for ls
info --subnodes grep        # Display info in a single-page format for grep:
info --output=bash_info.txt bash#Save the documentation of bash to a file:
```

4. whatis

The whatis command provides a brief, one-line description of a command, similar to using man -f. It quickly helps users understand the purpose of a command without opening the full manual.

Syntax:

```
whatis command_name
```

Examples:

```
whatis mkdir        # Displays a one-line summary like "mkdir (1) - make directories".
whatis tar                  # Find out what tar does
```

5. apropos

The apropos command searches the manual pages for a keyword, similar to man -k, and helps users find relevant commands based on functionality.

Syntax:

```
apropos keyword
```

Examples:

```
apropos list      # Displays a list of commands containing "list" in their description.
apropos disk      # Find commands related to "disk"
```

Summary of Differences

Command	Purpose	Best Used For
help	Displays help for shell built-ins only	Quick syntax reference for built-in commands
man	Shows manual pages for commands	Detailed documentation with sections
info	Provides structured, hyperlinked documentation	Detailed command reference with navigation
whatis	Gives a one-line summary of a command	Quick command purpose lookup
apropos	Searches manual pages for related commands	Finding commands based on functionality

Shell Scripting:

1. echo

The echo command is used to display a line of text or a variable value on the terminal. It is commonly used in shell scripts to provide output messages to users.

Syntax:

```
echo [OPTIONS] [STRING]
```

Common Options:

- -n : Do not output the trailing newline.
- -e : Enable interpretation of backslash escapes.

Example:

```
echo "Hello, World!"
echo -n "Hello, No New Line"
echo -e "Line1\nLine2"
```

2. read

The read command is used to take user input from the terminal. It waits for the user to enter input and stores it in a variable.

Syntax:

```
read [OPTIONS] [VARIABLE]
```

Common Options:

- -p : Prompt the user with a message.
- -s : Silent mode (useful for password input).

Example:

```
read -p "Enter your name: " name
echo "Hello, $name"
```

3. if Statement

The if statement is used to execute a block of commands conditionally. It is useful for decision-making in scripts.

Syntax:

```
if [ condition ]; then
    commands
elif [ condition ]; then
    commands
else
    commands
fi
```

Example:

```
num=10
if [ $num -gt 5 ]; then
    echo "Number is greater than 5"
else
    echo "Number is 5 or less"
fi
```

4. for Loop

The for loop is used to iterate over a sequence of values. It simplifies repetitive tasks.

Syntax:

```
for var in list; do
    commands
done
```

Example:

```
for i in 1 2 3 4 5; do
    echo "Iteration: $i"
done
```

5. **while Loop:** The while loop repeatedly executes a block of commands as long as a condition is true.

Syntax:

```
while [ condition ]; do
    commands
done
```

Example:

```
count=1
while [ $count -le 5 ]; do
    echo "Count: $count"
    count=$((count + 1))
done
```

6. **case Statement:** The case statement is used for pattern matching and is an alternative to multiple if-elif conditions.

Syntax:

```
case variable in
    pattern1)
        commands ;;
    pattern2)
        commands ;;
    *)
        default commands ;;
esac
```

Example:

```
read -p "Enter a letter: " letter
case $letter in
    a) echo "You entered A" ;;
    b) echo "You entered B" ;;
    *) echo "Unknown letter" ;;
esac
```

7. **function in Shell Script:** Functions in shell scripting allow you to define reusable blocks of code.

Syntax:

```
function_name() {
    commands
}
```

Example:

```
hello() {
    echo "Hello, World!"
}
hello
```

Vim Editor Commands and Shortcuts:

1. Opening a File in Vim

Vim is a powerful text editor used in Linux. To open a file in Vim, use the following command.

Syntax:

```
vim filename
```

Common Options & Examples:

```
vim file.txt          # Opens 'file.txt' in Vim editor
vim +10 file.txt      # Opens 'file.txt' and places cursor at line 10
vim + file.txt        # Opens file.txt with the cursor at the last line
vim -R file.txt       # Opens file in read-only mode
```

2. Inserting Text (Insert Mode)

Vim starts in Normal mode. To insert text, you need to switch to Insert mode.

Syntax:

Command	Description
i	Insert before cursor
I	Insert at beginning of line
a	Append after cursor
A	Append at end of line
o	Open a new line below cursor
O	Open a new line above cursor

Common Options & Examples:

```
i       # Enter insert mode before cursor
A       # Enter insert mode at end of line
o       # Create a new line below the current line
```

3. Saving and Exiting

Once editing is done, you can save and exit Vim using different commands.

Syntax:

Command	Description
:w	Save file
:q	Quit Vim
:wq	Save and quit
:x	Save and quit (only if changes were made)
:q!	Quit without saving

Common Options & Examples:

```
:w          # Saves the file
:wq         # Saves and exits Vim
:q!         # Exits without saving
:x          # Saves and exits (if changes were made)
```

4. Navigation in Normal Mode

To navigate within a file, use the following commands in Normal mode.

Syntax:

Command	Description
h	Move left
l	Move right
j	Move down
k	Move up
0	Move to beginning of line
^	Move to first non-blank character of line
$	Move to end of line
gg	Move to the beginning of file
G	Move to the end of file
:n	Move to line n

Common Options & Examples:

```
10G         # Move to line 10
gg          # Move to the beginning of the file
G           # Move to the end of the file
```

5. Searching for Text

You can search for specific text in a file using the following commands.

Syntax:

Command	Description
/pattern	Search forward for 'pattern'
?pattern	Search backward for 'pattern'
n	Repeat the last search in the same direction
N	Repeat the last search in the opposite direction

Common Options & Examples:

```
/foo        # Searches forward for 'foo'
?bar        # Searches backward for 'bar'
n           # Repeat last search forward
N           # Repeat last search backward
```

6. Copy, Cut, and Paste (Yank, Delete, and Put)

To copy, cut, and paste text in Vim, use the following commands in Normal mode.

Syntax:

Command	Description
yy	Copy current line
yw	Copy current word
y$	Copy from cursor to end of line
dd	Cut (delete) current line
dw	Cut current word
d$	Cut from cursor to end of line
p	Paste after cursor
P	Paste before cursor

Common Options & Examples:

```
yy          # Copy the current line
dd          # Cut the current line
p           # Paste after cursor
P           # Paste before cursor
```

7. Undo and Redo

Vim allows undoing and redoing changes.

Syntax:

Command	Description
u	Undo last change
U	Undo all changes on current line
Ctrl + r	Redo the last undone change

Common Options & Examples:

```
u           # Undo last change
Ctrl + r    # Redo last undone change
```

8. Replacing and Substituting Text

You can replace characters or substitute text using the following commands.

Syntax:

Command	Description
r<char>	Replace current character with <char>
R	Enter Replace mode
:s/old/new/	Replace 'old' with 'new' on the current line
:%s/old/new/g	Replace 'old' with 'new' globally in the file

Common Options & Examples:

```
:s/foo/bar/          # Replace first occurrence of 'foo' with 'bar' on current line
:%s/foo/bar/g        # Replace all occurrences of 'foo' with 'bar' in file
```

9. Visual Mode (Selecting Text)

Vim provides visual mode to select text.

Syntax:

Command	Description
v	Start visual mode
V	Start visual line mode
Ctrl + v	Start visual block mode

Common Options & Examples:

```
v          # Start visual mode
V          # Start line selection mode
Ctrl + v   # Start block selection mode
```

10. Splitting Windows

You can split the Vim window into multiple panes.

Syntax:

Command	Description
:split	Split window horizontally
:vsplit	Split window vertically
Ctrl + w + w	Switch between windows
Ctrl + w + q	Close current window

Common Options & Examples:

```
:split file2.txt        # Open 'file2.txt' in a horizontal split
:vsplit file3.txt       # Open 'file3.txt' in a vertical split
```

11. Deleting Text

Vim allows various deletion operations in Normal mode.

Syntax:

Command	Description
x	Delete character under cursor
X	Delete character before cursor
dw	Delete current word
d$	Delete from cursor to end of line
dd	Delete current line
D	Delete from cursor to end of line (same as d$)
dG	Delete from cursor to end of file
dgg	Delete from cursor to beginning of file

Common Options & Examples:

```
x           # Delete character under cursor
dw          # Delete current word
dd          # Delete current line
```

12. Indentation

Vim allows adjusting indentation of lines.

Syntax:

Command	Description
>>	Indent current line right
<<	Indent current line left
5>>	Indent 5 lines right
5<<	Indent 5 lines left
=	Auto-indent current line
=%	Auto-indent matching braces block

Common Options & Examples:

```
>>          # Indent current line
<<          # Unindent current line
5>>         # Indent 5 lines to the right
```

13. Working with Buffers

Buffers in Vim allow working with multiple open files.

Syntax:

Command	Description
:e filename	Open file in a new buffer
:ls	List all open buffers
:bnext or :bn	Move to next buffer
:bprevious or :bp	Move to previous buffer
:bd	Close current buffer

Common Options & Examples:

```
:e file2.txt        # Open 'file2.txt' in a new buffer
:ls                 # List open buffers
:bn                 # Move to next buffer
```

14. Window Management

You can work with multiple split windows in Vim.

Syntax:

Command	Description
:split filename	Split window horizontally and open file
:vsplit filename	Split window vertically and open file
Ctrl + w + w	Switch between windows
Ctrl + w + q	Close current window
Ctrl + w + h/j/k/l	Move between split windows

Common Options & Examples:

```
:split file2.txt    # Split window horizontally and open 'file2.txt'
:vsplit file3.txt   # Split window vertically and open 'file3.txt'
```

15. **Macros (Recording and Replaying Commands):** Vim allows recording keystrokes as macros for automation.

Syntax:

Command	Description
q<register>	Start recording to register <register>
q	Stop recording
@<register>	Replay macro stored in <register>
@@	Replay last used macro

Common Options & Examples:

```
qa          # Start recording macro in register 'a'
q           # Stop recording
@a          # Execute macro stored in 'a'
```

16. **Working with Marks:** Marks allow bookmarking positions in a file for quick navigation.

Syntax:

Command	Description
ma	Set mark 'a' at cursor position
``a`	Move cursor to mark 'a'
'a	Move cursor to start of line where mark 'a' is
:marks	List all marks
:delmarks a	Delete mark 'a'

Common Options & Examples:

```
ma          # Set mark 'a' at cursor position
`a          # Move cursor to mark 'a'
:marks      # List all marks
```

17. **Spell Checking:** Vim supports spell checking in text files.

Syntax:

Command	Description
:set spell	Enable spell checking
:set nospell	Disable spell checking
]s	Move to next misspelled word
[s	Move to previous misspelled word
z=	Suggest correct spelling for misspelled word

Common Options & Examples:

```
:set spell  # Enable spell check
z=          # Show spelling suggestions
```