LINUX

For DevOps Engineers and Cloud Engineers

Chandra Sekhar Ravella

Table of Contents:

Fundamentals of Linux

Operating System (OS):

An **Operating System (OS)** is system software that acts as an intermediary between computer hardware and the user. It manages hardware resources and provides a set of services for application software.

🧠 What Does an Operating System Do?

At a high level, an OS performs five key functions:

1. **Process Management** 🧩
    - Manages processes in the system: creating, scheduling, and terminating.
    - Ensures that multiple programs can run concurrently without interfering.
2. **Memory Management** 🧠
    - Keeps track of each byte in a computer's memory and manages allocation and deallocation.
    - Handles **virtual memory**, where programs can use more memory than physically available.
3. **File System Management** 📁
    - Organizes and manages data on storage devices like hard drives or SSDs.
    - Provides an interface for reading/writing files and managing directories.
4. **Device Management** 🔌
    - Controls hardware devices via device drivers.
    - Manages communication between software and peripherals like printers, keyboards, etc.
5. **User Interface** 🖥️
    - Offers either a Command-Line Interface (CLI) or a Graphical User Interface (GUI) to interact with the system.

💻 Examples of Operating Systems:

- **Windows** (by Microsoft)
- **macOS** (by Apple)
- **Linux** (open source, many distributions like Ubuntu, Fedora)
- **Android** (based on Linux, for mobile devices)
- **iOS** (by Apple, for iPhones/iPads)

🕰️ History of Operating Systems:

Period	Focus	Key OSs
1940s–50s	Manual operation	–
1950s–60s	Batch processing	GM-NAA I/O, early IBM OS
1960s–70s	Time-sharing, multiprogramming	Multics, CTSS
1970s	Portability, UNIX	UNIX, VMS, BSD
1980s	Personal computing, GUIs	MS-DOS, Mac OS, Windows
1990s	Networking, open source	Windows 95, Linux
2000s–Today	Mobile, cloud, virtualization	Android, iOS, Linux

🐧 Why Linux Over Windows?

1. Cost-Effectiveness

- **Free and Open Source**: Linux does not require expensive licensing fees, making it a cost-effective choice for companies.
- **Lower Maintenance Costs**: Linux is stable and requires minimal maintenance, reducing operational expenses.

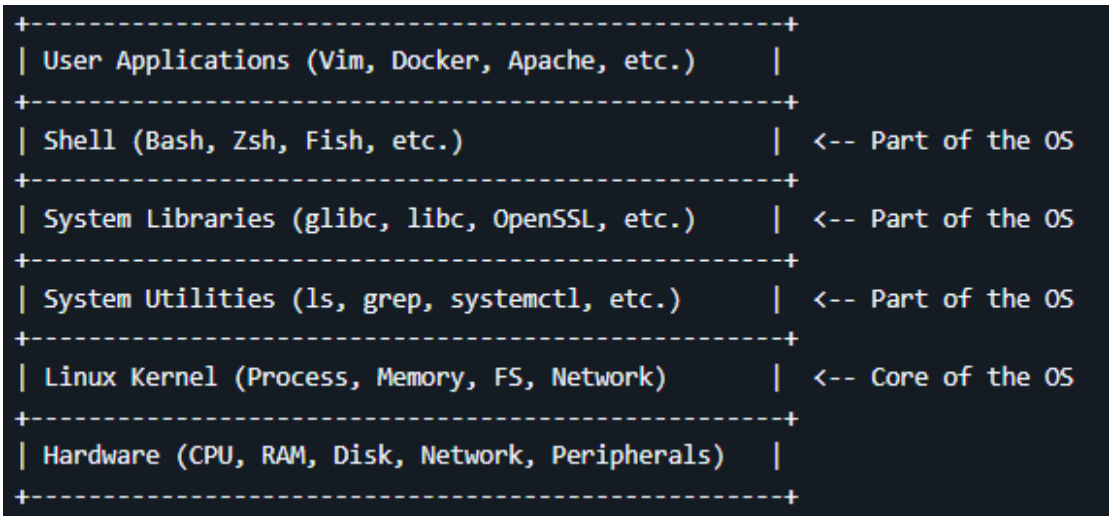
2. Performance and Efficiency

- **Better Resource Utilization**: Linux is lightweight and consumes fewer system resources compared to Windows.
- **High Scalability**: Linux efficiently scales from small embedded systems to enterprise data centers without performance degradation.

3. Security and Reliability

- **Less Vulnerable to Malware**: Linux has strong user privilege separation, making it more secure against viruses and malware.
- **Frequent and Transparent Updates**: Regular security patches ensure system stability without requiring frequent reboots.
- **High Stability**: Linux systems can run for years without crashes, ensuring better uptime and reliability.

Core Components of a Linux Machine:



(a) Hardware Layer

- The physical components of the computer (CPU, RAM, disk, network interfaces, etc.).
- The OS interacts with hardware using device drivers.

(b) Kernel (Core of Linux OS)

- The Linux Kernel is responsible for directly managing system resources, including:

  Process Management – Schedules processes and handles multitasking.

  Memory Management – Allocates and deallocates RAM efficiently.

  Device Drivers – Acts as an interface between software and hardware.

  File System Management – Manages how data is stored and retrieved.

  Network Management – Handles communication between systems.

(c) System Call Interface

- A bridge between user space and kernel space.
- Allow user programs to request services from the kernel (like reading a file or starting a process).

🧩 Examples: open (), read (), fork (), exec ()

(d) Shell (Command Line Interface - CLI)

- A command interpreter that allows users to interact with the kernel.
- Examples: Bash, Zsh, Fish, Dash, Ksh.
- Converts user commands into system calls for the kernel.

(e) User Applications

- End-user programs like web browsers, text editors, DevOps tools, etc.
- Applications interact with the OS using system calls via the shell or GUI.

🧠 **Fun Fact:**

Everything in Linux is treated as a **file** — including hardware devices!

🐧 Linux Distributions

A Linux distribution (often called a distro) is a complete operating system built using the Linux kernel along with:

- System software
- User interface (GUI or CLI)
- Package Manager
- Pre-installed tools and applications

Each distro is tailored for different users — beginners, developers, security experts, or servers.

Here are some popular Linux distributions:

Ubuntu – One of the most beginner-friendly distros, widely used for personal and server use. It has great community support.

CentOS (discontinued, replaced by AlmaLinux/Rocky Linux) – Previously a popular choice for servers, based on Red Hat Enterprise Linux (RHEL).

Debian – A very stable and reliable distro, often used as a base for other distros like Ubuntu.

Fedora – A cutting-edge distro that introduces new features before they reach RHEL.

Arch Linux – A lightweight, rolling-release distro for advanced users who like customization.

Kali Linux – Designed for cybersecurity and penetration testing.

Alpine Linux – A lightweight, security-focused distro often used in containers.

💻 Setting Up a Linux Environment on Windows and macOS:

There are multiple ways to setup a Linux environment on a Windows or Mac machines such as cloud vm, wsl2, virtualbox, Hyperkit e.t.c., However, what I would recommend is using a container as a Linux environment.

Just install Docker desktop, run the below command and create Linux container of any distribution without worrying about the cost and connectivity issues.

Docker Command to Run Ubuntu Linux Container (Persistent & Long-Term)

```
docker run -dit \
  --name ubuntu-container \
  --hostname ubuntu-dev \
  --restart unless-stopped \
  --cpus="2" \
  --memory="4g" \
  --mount type=bind,source=C:/ubuntu-data,target=/data \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -p 2222:22 \
  -p 8080:80 \
  --env TZ=Asia/Kolkata \
  --env LANG=en_US.UTF-8 \
  ubuntu:latest /bin/bash
```

Explanation of Each Parameter:

Parameter	Description
`-dit`	Runs the container in **detached (-d)**, **interactive (-i)**, and **terminal (-t)** mode.
`--name ubuntu-container`	Assigns a name to the container for easy management.
`--hostname ubuntu-dev`	Sets the container's hostname.
`--restart unless-stopped`	Ensures the container restarts automatically unless manually stopped.
`--cpus="2"`	Limits the container to **2 CPU cores**.
`--memory="4g"`	Allocates **4GB RAM** to the container.
`--mount type=bind,source=C:/ubuntu-data,target=/data`	**Mounts a folder** from Windows into the container to persist data.
`-v /var/run/docker.sock:/var/run/docker.sock`	Allows running Docker commands inside the container (optional).
`-p 2222:22`	Maps port **2222** on the host to **22** (SSH) inside the container.
`-p 8080:80`	Maps port **8080** on the host to **80** (for web services).
`--env TZ=Asia/Kolkata`	Sets the **timezone** (modify based on your location).
`--env LANG=en_US.UTF-8`	Sets the **language** settings inside the container.
`ubuntu:latest /bin/bash`	Uses the latest **Ubuntu** image and runs Bash shell.

Package Managers in Linux

What is a Package Manager?

A package manager is a tool that automates the process of installing, updating, configuring, and removing software in a Linux system. It ensures that software and its dependencies are managed efficiently.

🔍 How Does a Package Manager Work?

1. **Repositories (Repos):**
   - A package manager fetches software from official repositories (online storage of packages).
   - Example: Ubuntu gets packages from archive.ubuntu.com.
2. **Installing Software:**
   - When you install the software, the package manager:
     - ✅ Downloads the package from the repository.
     - ✅ Resolves dependencies (installs additional required software).
     - ✅ Installs and configures the software automatically.
3. **Updating Software:**
   - A single command updates all installed packages to the latest version.
4. **Removing Software:**
   - The package manager also removes software cleanly without leaving unnecessary files.

📦 Popular Package Managers in Linux

Linux Distro	Package Manager	Command Example
Ubuntu, Debian	`apt` (Advanced Package Tool)	`sudo apt install nginx`
Fedora, RHEL, CentOS	`dnf` (or `yum` for older versions)	`sudo dnf install nginx`
Arch Linux	`pacman`	`sudo pacman -S nginx`
OpenSUSE	`zypper`	`sudo zypper install nginx`

🌍 How Package Managers Fetch Software from Repositories

A repository is a server that stores software packages. When a package manager installs software:

1. It checks the repository list (e.g., /etc/apt/sources.list in Ubuntu).
2. It downloads the package and its dependences.
3. It installs and configures the software automatically.

📁 Example of an Ubuntu Repository Entry

Types: deb

URIs: http://ports.ubuntu.com/ubuntu-ports/

Suites: noble noble-updates noble-backports noble-security

Components: main universe restricted multiverse

Signed-By: /usr/share/keyrings/ubuntu-archive-keyring.gpg

🔄 Why Should You Run apt update After Installing Ubuntu?

When you install Ubuntu, the packages included in the ISO image might be outdated. Running:

```
sudo apt update
```

✅ Updates the package list from repositories.

Then, to install the latest versions of packages, run:

```
sudo apt upgrade -y
```

🛠 Essential Package Manager Commands

APT (Debian, Ubuntu)

```
sudo apt update          # Update package lists
sudo apt upgrade -y      # Upgrade installed packages
sudo apt install nginx   # Install a package
sudo apt remove nginx    # Remove a package
sudo apt autoremove      # Remove unused dependencies
sudo apt search nginx    # Search for a package
```

DNF (Fedora, RHEL, CentOS)

```
sudo dnf check-update   # Check for updates
sudo dnf update         # Update all packages
sudo dnf install nginx  # Install a package
sudo dnf remove nginx   # Remove a package
```

Pacman (Arch Linux)

```
sudo pacman -Syu        # Sync and update all packages
sudo pacman -S nginx    # Install a package
sudo pacman -R nginx    # Remove a package
```

Zypper (OpenSUSE)

```
sudo zypper refresh     # Refresh package list
sudo zypper update      # Update all packages
sudo zypper install nginx  # Install a package
sudo zypper remove nginx   # Remove a package
```

🚀 Best Practices for Using Package Managers:

✅ Always update your package list before installing software:

```
sudo apt update && sudo apt upgrade -y
```

✅ Use **autoremove** to clean up unused dependencies:

```
sudo apt autoremove
```

✅ Enable automatic security updates (Ubuntu):

```
sudo apt install unattended-upgrades
sudo dpkg-reconfigure unattended-upgrades
```

🗀 Linux Directory Structure

In Linux, everything is organized in a tree-like hierarchy, starting from the root directory **/**.

Each folder has a specific purpose, and understanding this structure helps you navigate, troubleshoot, and manage the system effectively.

🔗 Symbolic Links (Less Significant):

Directory	Linked To	Description
/sbin	/usr/sbin	System binaries for administrative commands (e.g., reboot, shutdown, ifconfig)
/bin	/usr/bin	Essential user binaries (e.g., ls, cp, mv, bash)
/lib	/usr/lib	Shared libraries and kernel modules

🛠️ Important System Directories:

Directory	Description
/boot	Stores files needed to boot the system (e.g., kernel, GRUB configs)
/usr	Contains most user-installed applications and libraries
/var	Stores logs, caches, and frequently changing data
/etc	Contains system configuration files

👤 User & Application-Specific Directories:

Directory	Description
/home	Default location for user home directories (e.g., /home/chandu)
/opt	Used for installing optional or third-party software
/srv	Holds data for system services like web servers (rarely used in containers)
/root	Home directory for the root (superuser)

🔄 Temporary & Volatile Directories:

Directory	Description
/tmp	Temporary files; (cleared on reboot)
/run	Stores runtime process information (e.g., PID files, sockets)
/proc	Virtual filesystem for process/system info (e.g., /proc/cpuinfo)
/sys	Virtual filesystem with hardware/kernel info (e.g., /sys/class/)
/dev	Contains device files (e.g., /dev/null, /dev/sda).

💾 Mount Points:

Directory	Description
/mnt	Temporary mount point for external filesystems
/media	Mount point for removable devices like USBs and CDs
/data	Custom mount point (commonly used in WSL for Windows volumes like C:)

👥 User Management in Linux

Linux is a multi-user operating system, meaning multiple users can use the system at the same time or at different times. User management ensures each user has the right permissions, resources, and isolation.

👤 Types of Users in Linux:

User Type	Description
Root User	The superuser with full system access. Username: root. Can execute all commands.
Normal Users	Created by the administrator or during OS installation. Limited permissions.
System Users	Created by the system for services (e.g., www-data, mysql). Typically, don't log in.

📁 Important Files for User Management:

File	Purpose
/etc/passwd	Stores user account info (username, UID, home directory, shell)
/etc/shadow	Stores encrypted passwords and password aging info
/etc/group	Contains group information
/etc/sudoers	Configures who can use sudo and how

Note: Use visudo to edit /etc/sudoers safely.

➕ Creating Users in Linux:

To create a new user in Linux, use:

- useradd Command (For most Linux distributions)

```
useradd username
```

Note: This creates a user without a home directory.

- To create a user with a home directory:

```
useradd -m username
```

- To specify a shell:

```
useradd -s /bin/bash username
```

- **adduser** Command (For Debian-based systems):

```
adduser username
```

This is an interactive command that asks for a password and additional details.

🔐 Set or Change User Password:

To set or change a user's password:

```
passwd username
```

Enforcing Password Policies:

- Password expiration: Set password expiry days

```
chage -M 90 username
```

- Lock a user account

```
passwd -l username
```

- Unlocking a user account

```
passwd -u username
```

🔄 Modify a User:

Modify an existing user with usermod:

- Change the username:

```
usermod -l new_username old_username
```

- Change the home directory:

```
usermod -d /new/home/directory -m username
```

- Change the default shell:

```
usermod -s /bin/zsh username
```

➖ Delete a User:

- To remove a user but keep their home directory:

```
userdel username
```

- To remove a user and their home directory:

```
userdel -r username
```

👥 Group Management:

Groups help assign shared permissions.

- Create a Group:

```
groupadd groupname
```

- Add User to Group:

```
usermod -aG groupname username
```

- View Groups of a User:

```
groups username
```

- Changing Primary Group:

```
usermod -g new_primary_group username
```

Sudo Access and Privilege Escalation:

Adding a User to Sudo Group:

- On Debian-based systems:

```
usermod -aG sudo username
```

- On RHEL-based systems:

```
usermod -aG wheel username
```

🔐 Granting Specific Commands with Sudo:

Edit the sudoers file:

```
visudo
```

Then add:

```
username ALL=(ALL) NOPASSWD: /path/to/command
```

🗀 File Management in Linux

Linux treats everything as a file, including text files, directories, devices, and even hardware!

Learning file management is essential for navigating, creating, editing, and organizing files and directories in a Linux environment.

📌 Basic File Terminology:

Term	Description
File	A collection of data (e.g., text, images, scripts).
Directory	A folder that contains files or other directories.
Path	Location of a file or directory. Can be absolute or relative.

📂 File & Directory Navigation:

Command	Description
pwd	Show current working directory
ls	List of all files and directories
cd	Change directory
cd ..	Go one level up
cd /	Go to the root directory
cd ~	Go to home directory

🗂️ File and Directory Operations:

📁 Create:

```bash
touch filename        # Create an empty file
mkdir dirname         # Create a new directory
```

🖍️ Rename / Move:

```bash
mv oldname newname    # Rename file or directory
mv file /path/        # Move file to another directory
```

❌ Delete:

```bash
rm filename           # Delete a file
rm -r directory       # Delete a directory and its contents
```

📄 Copy:

```bash
cp file1 file2        # Copy file1 to file2
cp -r dir1 dir2       # Copy directory recursively
```

🔍 View File Contents and Editing:

Command	Description
cat file.txt	Displays file content.
tac file.txt	Displays file content in reverse order.
head -n 10 file.txt	Displays the first 10 lines of a file.
tail -n 10 file.txt	Displays the last 10 lines of a file.
less file.txt	Opens a file for viewing with scrolling support.
more file.txt	Like less but only moves forward.
nano file.txt	Opens a simple text editor.
vi file.txt	Opens a powerful text editor.
echo 'Hello' > file.txt	Writes text to a file, overwriting existing content.
echo 'Hello' >> file.txt	Appends text to a file without overwriting.

✍️ VI Editor Shortcuts in Linux

The vi editor (or its improved version vim) is a powerful text editor in Linux, commonly used to edit configuration files and scripts directly from the terminal.

It works in three modes:

Mode	Description
Normal mode	Default mode when vi starts — used for navigation and commands
Insert mode	Used for typing/editing text
Command mode	Used for saving, quitting, searching, etc.

🚀 Launching VI:

```bash
vi filename
```

- Open the file in normal mode.
- If the file doesn't exist, it will be created upon saving.

🧭 Mode Switching:

Key	Action
i	Enter insert mode before the cursor
I	Enter insert mode at the beginning of the line
a	Enter insert mode after the cursor
A	Enter insert mode at the end of the line
o	Open a new line below
O	Open a new line above
Esc	Exit insert mode and return to normal mode

💾 Basic Command Mode Shortcuts (: commands)

Command	Action
: w	Save (write) file
: q	Quit
: wq or ZZ	Save and quit
: q!	Quit without saving
:x	Save and exit (same as: wq)
: e filename	Open another file in vi
: set nu	Show line numbers
: set nonu	Hide line numbers

📝 Text Editing in Normal Mode:

Command	Action
yy or Y	Copy (yank) current line
p	Paste after the cursor
P	Paste before the cursor
dd	Delete (cut) current line
x	Delete character under cursor
u	Undo last change
Ctrl + r	Redo
r	Replace a single character
R	Overwrite mode (until Esc)

🧭 Navigation Shortcuts:

Key	Action
h	Move left
l	Move right
j	Move down
k	Move up
0	Move to the beginning of line
^	First non-blank of line
$	End of line
w	Jump forward one word
b	Jump back one word
G	Go to the end of file
gg	Go to the beginning of file
: n	Go to line number n

🔍 Search and Replace:

Command	Action
/text	Search forward for "text"
?text	Search backward for "text"
n	Repeat last search
:%s/old/new/g	Replace all "old" with "new" in entire file
:s/old/new/g	Replace in current line

🔐 SSH (Secure Shell)

✅ What is SSH?

SSH stands for Secure Shell. It's a protocol used to securely connect to a remote computer over a network — typically used to access servers, Linux systems, and cloud instances.

Think of SSH as a secure way to control another computer using the terminal just like you're sitting in front of it!

🛡️ Why is SSH Important?

- **Encrypted Communication** – All data sent between you and the server is encrypted (safe from eavesdropping).
- **Remote Access –** Control remote systems from anywhere**.**
- **File Transfer** – Send and receive files securely via SCP/SFTP.
- **Automation** – Useful for running scripts, updates, and deployment tasks on servers.

🔐 How SSH Works

SSH uses **key-based** or **password-based** authentication.

1. You run an SSH command.
2. SSH connects to the remote machine's SSH server (usually on **port 22**).
3. You log in using a **username + password** or **SSH key**.
4. Once connected, you can run commands, manage files, or even tunnel other connections securely.

👷 Basic SSH Command:

```bash
ssh username@hostname_or_ip
```

- Example:

```bash
ssh ubuntu@192.168.1.10
```

- **ubuntu:** username on the remote machine.
- **192.168.1.10:** IP address or domain name of the remote machine.

📁 Common SSH Use Cases:

Use Case	Command Example
SSH into server	`ssh user@ip`
Copy file to server	`scp file user@ip:/path/`
SFTP file transfer	`sftp user@ip`
Port forwarding (tunneling)	`ssh -L 8080:localhost:80 user@ip`
Run remote command via SSH	`ssh user@ip "ls -l"`

📦 SCP (Secure Copy Protocol) in Linux

✅ What is SCP?

SCP stands for **Secure Copy Protocol**. It is a command-line tool used to copy files or directories between two systems over SSH.

- It encrypts files during transfer (unlike cp or ftp).
- Works **locally → remote**, **remote → local**, or even **remote → remote.**

📌 Basic Syntax:

```
scp [options] source destination
```

- source: file or folder you want to copy.
- destination: where you want to copy it (can be local or remote).
- Both can be in the format: user@host:/path.

📁 Common Use Case:

Task	Command Example
Upload file to server	`scp file.txt user@ip:/path/`
Download file from server	`scp user@ip:/path/file.txt ./`
Upload folder	`scp -r folder/ user@ip:/path/`
Use custom port	`scp -P 2222 file.txt user@ip:/path/`

⚙️ Useful Options:

Option	Description
-r	Recursively copy entire directories
-P	Specify port number if not using default (22)
-i	Use a specific identity file (SSH key)
-C	Enable compression during transfer
-v	Verbose output (helpful for debugging)

🔐 File Permissions Management in Linux

Introduction to File Permissions:

Linux file permissions determine who can read, write, or execute files and directories. Each file and directory have three levels of permission:

👥 Types of Users:

- **Owner (User):** The creator of the file.
- **Group**: Users belonging to the assigned group.
- **Others**: All other users on the system.

Permissions are represented as:

- **Read (r or 4)** – View file contents.
- **Write (w or 2)** – Modify file contents.

- **Execute (x or 1)** – Run scripts or programs.

📄 Viewing File Permissions:

```bash
ls -l filename
```

Example Output:

```bash
-rwxr-xr--  1 user group  4096 Apr 15  myscript.sh
```

Breakdown:

- - = regular file (d = directory)
- rwx = Owner can read, write, execute
- r-x = Group can read, execute
- r-- = Others can only read

🖍️ Changing Permissions with chmod:

Using Symbolic Mode

Modify permissions using symbols:

- Add (+), remove (-), or set (=) permissions.

Examples:

```
chmod u+x filename  # Add execute for user
chmod g-w filename  # Remove write for group
chmod o=r filename  # Set read-only for others
chmod u=rwx,g=rx,o= filename  # Set full access for user, read/execute for group, and no access for others
```

Using Numeric (Octal) Mode

Each permission has a value:

- Read (4), Write (2), Execute (1).

Examples:

```
chmod 755 filename  # User (rwx), Group (r-x), Others (r-x)
chmod 644 filename  # User (rw-), Group (r--), Others (r--)
chmod 700 filename  # User (rwx), No access for others
```

👑 Changing Ownership with chown:

Modify file owner and group:

```
chown newuser filename  # Change owner
chown newuser:newgroup filename  # Change owner and group
chown :newgroup filename  # Change only group
```

Recursively change ownership:

```
chown -R newuser:newgroup directory/
```

🔁 Changing Group Ownership with chgrp:

```
chgrp newgroup filename  # Change group
chgrp -R newgroup directory/  # Change group recursively
```

📁 Special Permission Bits (Advanced):

SetUID (s on user execute bit):

Allows users to run a file with the file owner's permission.

```
chmod u+s filename
```

Example: /usr/bin/passwd allows users to change their passwords.

SetGID (s on group execute bit):

Files: Users run the file with the group's permissions.

Directories: Files created inside inherit the group.

```
chmod g+s filename  # Set on file
chmod g+s directory/  # Set on directory
```

Sticky Bit (t on others execute bit): Used directories to allow only the owner to delete their files.

```
chmod +t directory/
```

Example: /tmp directory.

Default Permissions: umask

umask defines default permissions for new files and directories. Check current umask:

```
umask
```

Set a new umask:

```
umask 022  # Default: 755 for directories, 644 for files
```

Process Management in Linux

What is a Process?

A process is a running instance of a program. Every time you execute a command; Linux creates a process to handle it.

Each process has:

- PID (Process ID) – Unique identifier
- PPID (Parent Process ID) – ID of the parent process
- UID (User ID) – Who owns the process
- Resource info – CPU, memory usage, etc.

Index of Commands Covered:

Viewing Processes:

- `ps aux` – View all running processes along with memory usage
- `ps -ef` - View all running process in full format listing
- `ps -u username` – View processes for a specific user
- `ps -C processname` – Show a process by name
- `pgrep processname` – Find a process by name and return its PID
- `pidof processname` – Find the PID of a running program

Managing Processes:

- `kill PID` – Terminate a process by PID
- `pkill processname` – Terminate a process by name
- `kill -9 PID` – Force kill a process
- `pkill -9 processname` – Kill all instances of a process
- `kill -STOP PID` – Stop a running process
- `kill -CONT PID` – Resume a stopped process
- `renice -n 10 -p PID` – Lower priority of a process
- `renice -n -5 -p PID` – Increase priority of a process (requires root)

Background & Foreground Processes:

- `command &` – Run a command in the background
- `jobs` – List background jobs
- `fg %jobnumber` – Bring a job to the foreground
- `Ctrl + Z` – Suspend a running process
- `bg %jobnumber` – Resume a suspended process in the background

Monitoring System Processes:

- `top` – Interactive process viewer
- `htop` – User-friendly process viewer (requires installation)
- `nice -n 10 command` – Run a command with a specific priority
- `renice -n -5 -p PID` – Change priority of an existing process

Priority Value	Meaning
-20	Highest priority
0	Default priority
+19	Lowest priority

Note: Only the root user can set negative nice values.

Daemon Process Management:

- `systemctl list-units --type=service` – List all system daemons
- `systemctl start service-name` – Start a daemon/service
- `systemctl stop service-name` – Stop a daemon/service
- `systemctl enable service-name` – Enable a service at startup

📊 Linux System Monitoring

System Monitoring in Linux is about keeping an eye on the system's health — such as CPU usage, memory, disk, network, and running processes — to troubleshoot issues or optimize performance.

🔍 Why Monitor a Linux System?

- Detect high CPU or memory usage
- Identify misbehaving processes
- Monitor system uptime and load
- Ensure system is stable, secure, and performant
- Perform resource planning for scaling

Index of Commands Covered:

🧠 CPU and Memory Monitoring:

🔹 **top – Real-Time System Monitoring**

```bash
top
```

- Shows a live view of processes and system resource usage (CPU, memory, etc.)
- Press q to quit, k to kill a process, P to sort by CPU usage.

🔹 **htop – Interactive Process Viewer (Requires Installation)**

```bash
htop
```

- A colorful, user-friendly alternative to top.

- Allows scrolling, sorting, and killing processes with keyboard shortcuts.
- To install:

```bash
sudo apt install htop
```

🔹 **vmstat – System Performance Statistics:**

```bash
vmstat 1 5
```

- Reports on memory, processes, CPU, I/O, and more.
- 1 5 = run every 1 second, 5 times.

🔹 **free -m – Memory Usage in MB**:

```bash
free -m
```

- Shows total, used, and free memory (RAM and swap) in megabytes.

🔹 **free -h – Human-Readable Memory Usage:**

```bash
free -h
```

- Same as free -m but output is easier to read (e.g., 2.3G, 512M).

💾 Disk Monitoring:

🔹 **df -h – Disk Space Usage:**

```bash
df -h
```

- Displays total and available disk space for mounted file systems.
- -h = human-readable sizes (e.g., GB/MB).

- **du -sh * – Directory Size Usage:**

```bash
du -sh *
```

  - Shows size of each item (file/folder) in the current directory.
  - -s: summary; -h: human-readable

- **iostat – CPU and Disk I/O Statistics:**

```bash
iostat
```

  - Displays CPU load and disk input/output stats.
  - May require:

```bash
sudo apt install sysstat
```

🌐 Network Monitoring:

- **ifconfig – Show Network Interfaces (Deprecated):**

```bash
ifconfig
```

  - Shows IP addresses and interface status.
  - Now deprecated in favor of ip a.

- **ip a – Show Network Interface Details:**

```bash
ip a
```

  - Modern replacement for ifconfig.
  - Shows interface names, IP addresses, MAC addresses, and more.

- **netstat -tulnp – Active Connections & Ports:**

```bash
netstat -tulnp
```

- -t: TCP
- -u: UDP
- -l: Listening ports
- -n: Show numeric addresses
- -p: Show process IDs and names

Note: netstat may need to be installed with net-tools.

- **ss -tulnp – Faster Alternative to netstat:**

```bash
ss -tulnp
```

- Performs the same job as Netstat but faster and with more options.
- Pre-installed on most modern distros.

- **ping hostname – Test Network Connectivity**

```bash
ping google.com
```

- Send ICMP packets to test if the host is reachable.
- Use Ctrl+C to stop.

- **traceroute hostname – Trace Network Path**

```bash
traceroute google.com
```

- Shows each hop between your system and the target server.
- It is useful to diagnose where network delays occur.

- If not installed:

```bash
sudo apt install traceroute
```

- **nslookup domain – DNS Lookup Info:**

```bash
nslookup google.com
```

  - Shows IP address of a domain and DNS server response.
  - Good for testing DNS resolution issues.

📝 Log Monitoring:

- **tail -f /var/log/syslog – Live Log Monitoring:**

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

  - Continuously shows new lines added to the system log.
  - Use CTRL + C to stop.

- **journalctl -f – Live Logs for Systemd Systems:**

```bash
journalctl -f
```

  - Monitors system logs in real-time.
  - Works only on systemd-based systems like Ubuntu, Fedora, etc.

- **dmesg | tail – View Kernel Logs:**

```bash
dmesg | tail
```

  - Displays the latest kernel messages (hardware, boot info, driver messages).
  - Useful for checking hardware-related errors.

🌐 Networking in Linux

Linux provides a powerful set of tools and commands to manage, monitor, and troubleshoot network configurations and connectivity.

Key Concepts:

Concept	Description
IP Address	Unique identifier for a device on a network
MAC Address	Hardware address of the network interface
Interface	Network device (e.g., eth0, wlan0, lo)
Hostname	Name of the Linux system on the network
DNS	Resolves domain names to IP addresses

Networking Commands:

1. ping google.com – Checks connectivity to a remote server.
2. ifconfig – Displays network interfaces (deprecated, use ip).
3. ip a – Shows IP addresses of network interfaces.
4. netstat -tulnp – Displays open network connections.
5. curl https://example.com – Fetches a webpage's content.
6. wget https://example.com/file.zip – Downloads a file from the internet.

Disk and Storage Management in Linux

Introduction to Disk and Storage Management:

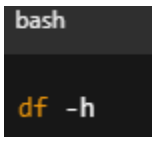
Disk management in Linux involves monitoring, formatting, partitioning, and mounting storage devices. It helps you understand how disk space is allocated and used, and how to manage additional drives or volumes.

Common Disk Management Tasks:

Task	Description
View disk usage	Check how much space is used and available
Partition disks	Divide a disk into logical sections
Format disks	Prepare partitions with a file system
Mount/unmount	Attach/detach disks to the system
Monitor I/O	Analyze disk read/write activity

Disk Usage Monitoring Commands:

- **df – Disk Free Space:**



  - Shows how much disk space is used and available on mounted filesystems.
  - -h = human-readable (e.g., GB/MB)

- **du – Disk Usage per Directory:**

  - Shows the size of a directory and its contents.
  - -s: summary, -h: human-readable

- **lsblk – List Block Devices:**

```bash
lsblk
```

- Lists all attached block devices (like disks and partitions).
- Does not show mount points by default unless -f is used.

🔹 **blkid – Show UUIDs and File System Types:**

```bash
sudo blkid
```

- Displays information about devices, UUIDs, and file system types.

🔹 **mount / umount – Mount or Unmount File Systems:**

```bash
sudo mount /dev/sdX1 /mnt/mydrive
sudo umount /mnt/mydrive
```

- mount: attaches a device to the system
- umount: detaches the device safely

🔹 **fdisk – Partition a Disk (Interactive):**

```bash
sudo fdisk /dev/sdX
```

- Used to create, delete, or modify partitions on MBR disks.

🔹 **mkfs – Create File System:**

```bash
sudo mkfs.ext4 /dev/sdX1
```

- Formats a partition with the ext4 file system (or other: xfs, ntfs, etc.)

🔹 **parted – Advanced Partitioning Tool:**

```bash
sudo parted /dev/sdX
```

- Useful for managing large drives, GPT, resizing, scripting.

🔹 **df -T – Check File System Type:**

```bash
df -T
```

- Shows the type of file system used on mounted partitions (e.g., ext4, xfs, etc.)

Logical Volume Management (LVM)

- pvcreate /dev/sdX – Create a physical volume
- vgcreate vg_name /dev/sdX – Create a volume group
- lvcreate -L 10G -n lv_name vg_name – Create a logical volume
- mkfs.ext4 /dev/vg_name/lv_name – Format an LVM partition
- mount /dev/vg_name/lv_name /mnt – Mount an LVM partition

Swap Management

- mkswap /dev/sdX – Create a swap partition
- swapon /dev/sdX – Enable swap space
- swapoff /dev/sdX – Disable swap space