**PROCESS MANAGEMENT**

**🔧 What is a Process in Linux?**

A **process** is a program in execution. Each process has a unique **PID (Process ID)** and runs in its own environment.

**📂 Types of Processes:**

1. **Foreground Process** – Runs directly with user interaction (e.g., vim, cat file.txt).
2. **Background Process** – Runs in the background using & (e.g., ./script.sh &).
3. **Daemon Process** – Background service processes (e.g., sshd, httpd).
4. **Zombie Process** – Completed execution but still has an entry in process table.
5. **Orphan Process** – Parent terminated, child continues under init (PID 1).

**🔍 Important Linux Process Management Commands:**

| **Command** | **Description** | **Example** |
| --- | --- | --- |
| ps | View running processes | ps aux, ps -ef |
| top | Dynamic real-time view of system processes | top |
| htop | Enhanced version of top (install separately) | htop |
| jobs | Lists background jobs | jobs |
| bg | Resume a stopped job in the background | bg %1 |
| fg | Bring a background job to foreground | fg %1 |
| kill | Send signal to process (default is TERM) | kill 1234 |
| killall | Kill all processes by name | killall firefox |
| nice | Run a process with priority | nice -n 10 myscript.sh |
| renice | Change priority of a running process | renice 5 -p 1234 |
| nohup | Run process immune to logout | nohup ./script.sh & |
| watch | Run a command repeatedly | watch -n 2 df -h |

**📌 What is ps Command?**

ps stands for **Process Status**.  
It is used to display information about active processes in the system.

**🛠️ Syntax**

ps [options]

**✅ Commonly Used ps Options**

| **Option** | **Description** |
| --- | --- |
| -e or -A | Show all processes |
| -f | Full format listing (with UID, PPID, C, STIME, TTY, TIME, CMD) |
| -u <user> | Show processes for a specific user |
| -x | Show processes without controlling terminal |
| -ef | All processes with full formatting (very common) |
| aux | BSD-style output for all users |
| -u <user> | Show processes owned by a specific user |
| -o <format> | Display custom output format |
| --sort=<key> | Sort output (e.g., --sort=-%mem for top memory users) |

**🧠 Popular ps Command Examples**

**1. Show all processes (System-wide)**

ps -e

**2. Full info about all processes (Very commonly used!)**

ps -ef

**3. Show processes of a specific user**

ps -u koushik

**4. Show processes with no terminal (like daemons)**

ps -x

**5. Combine all (like top)**

ps aux

**📋 Useful Columns Explained**

| **Column** | **Meaning** |
| --- | --- |
| PID | Process ID |
| PPID | Parent Process ID |
| UID | User ID of the process owner |
| CMD | The command that launched the process |
| %CPU | CPU usage |
| %MEM | Memory usage |
| TTY | Terminal type |
| TIME | Total CPU time used by the process |
| STIME | Start time of the process |

**🧠 What is the kill Command?**

The kill command in Linux is used to **send signals to processes**, primarily to **terminate** them, though it can be used to **pause, continue, or customize behavior** based on the signal sent.

Despite the name, kill **does not always terminate** a process — it just sends a **signal**, and the process decides what to do with it.

**🔧 Basic Syntax**

kill [options] <PID>

* PID: Process ID (you can get it from ps, top, htop, or pgrep)

**📌 Commonly Used Signals with kill**

| **Signal** | **Name** | **Meaning** |
| --- | --- | --- |
| 1 | SIGHUP | Reloads the process (hangup) |
| 2 | SIGINT | Interrupts process (Ctrl+C equivalent) |
| 9 | SIGKILL | Force kill (cannot be ignored) |
| 15 | SIGTERM | Terminate gracefully (default) |
| 18 | SIGCONT | Resume a stopped process |
| 19 | SIGSTOP | Pause the process (cannot be ignored) |

**📥 Examples**

**✅ Kill a process gracefully**

kill 1234

(This sends signal 15 — SIGTERM — by default)

**✅ Force kill a process (if it's stuck)**

kill -9 1234

(SIGKILL is harsh and immediately stops the process — use with caution!)

**✅ Pause (stop) a process**

kill -STOP 1234

**✅ Resume a paused process**

kill -CONT 1234

**🛠️ How to Find the PID of a Process**

**Using ps**

ps aux | grep firefox

**Using pgrep**

pgrep nginx

**🧠 Summary**

* kill sends signals to control processes.
* Default signal: SIGTERM (15), graceful shutdown.
* Use SIGKILL (9) only if graceful fails.
* Use kill with ps, pgrep, or top for best control.

**🧠 What is the top Command?**

The top command is a real-time system monitoring tool in Linux. It displays a **dynamic, real-time view** of the system’s processes, CPU usage, memory usage, uptime, and more.

**🎯 Why Use top in Real-Time?**

| **Real-Time Scenario** | **How top Helps** |
| --- | --- |
| Server suddenly slow | Identify processes eating up CPU/Memory. |
| Monitor system health | Live overview of CPU, RAM, load average, etc. |
| Detect runaway processes | See any stuck or looping process consuming 100% CPU. |
| Debug performance | Track app memory usage, I/O wait time. |
| Kill heavy process | Find and kill it directly from top. |

**🛠️ How to Run It**

top

You’ll see a live updating dashboard. Press **q** to quit.

**📊 Default top Display Breakdown**

**Top 5 lines (system summary):**

* top - 12:34:56 up 2 days, 1 user, load average: 0.04, 0.03, 0.01
* Tasks: Total, running, sleeping, etc.
* %Cpu(s): CPU usage (user, system, idle, etc.)
* MiB Mem: Total, used, free, buffers.
* MiB Swap: Total, used, free.

**Below: Per-process information**

| **Column** | **Meaning** |
| --- | --- |
| PID | Process ID |
| USER | User running the process |
| PR | Priority |
| NI | Nice value |
| VIRT | Virtual memory used |
| RES | Resident memory used |
| SHR | Shared memory used |
| S | State (R, S, Z, etc.) |
| %CPU | CPU usage |
| %MEM | Memory usage |
| TIME+ | Total CPU time |
| COMMAND | Command running the process |

**🔄 Interactive Keys (While in top)**

| **Key** | **Action** |
| --- | --- |
| q | Quit top |
| h | Help |
| k | Kill a process (you’ll be asked for PID) |
| r | Renice a process |
| P | Sort by CPU usage (default) |
| M | Sort by memory usage |
| T | Sort by time |
| N | Sort by PID |
| 1 | Show all CPU cores |
| u | Show processes of specific user |
| Shift + > / < | Change sort column |

**🎯 Real-Time Scenarios and Examples**

**✅ Scenario 1: See top CPU-consuming processes**

top

# Press 'P' to sort by %CPU

**✅ Scenario 2: See memory-hogging processes**

top

# Press 'M'

**✅ Scenario 3: Kill a high CPU process**

top

# Press 'k', enter PID, then press Enter

**✅ Scenario 4: Check usage per CPU core**

top

# Press '1'

**✅ Scenario 5: View processes of a specific user**

top

# Press 'u' → enter username → Enter

**🧑‍💻 How top Helps in DevOps / Production**

| **Situation** | **Use of top** |
| --- | --- |
| Web server slow | Check if Apache/Nginx is using high CPU |
| DB query stuck | Find if MySQL is hogging memory |
| Debug a cronjob | Monitor it live during execution |
| Auto-alert scripts | Run top -b -n 1 in cron + parse |
| Monitor docker container host | Check resource usage in the host system |

**🧠 Summary**

* Use top for **real-time monitoring**.
* Interactive controls allow **live process management**.
* Use **ps for snapshot**, **top for live**, and **htop for better visuals**.

**🧠 What is htop?**

htop is an **interactive and improved alternative to top**, providing a **visual, colorized, real-time view** of system processes, CPU, memory, and swap usage. It allows **easy navigation**, **scrolling**, and **direct process management** using keyboard shortcuts.

**✅ Why Use htop Instead of top?**

| **Feature** | **top** | **htop** |
| --- | --- | --- |
| Color display | ❌ | ✅ |
| Scroll through processes | ❌ | ✅ |
| Mouse support | ❌ | ✅ |
| Kill multiple processes | ❌ | ✅ |
| Tree view of processes | ❌ | ✅ |
| Custom column management | ❌ | ✅ |
| Easy to use | Moderate | Very Easy |

**On RHEL/CentOS:**

sudo yum install htop

**🎛️ Keyboard Shortcuts (Very Important)**

| **Key** | **Action** |
| --- | --- |
| F1 or h | Help |
| F2 | Setup (configure columns, meters) |
| F3 | Search process |
| F4 | Filter processes |
| F5 | Tree view (show process hierarchy) |
| F6 | Sort by column (choose CPU, MEM, PID, etc.) |
| F7 / F8 | Change nice value (renice) |
| F9 | Kill selected process |
| F10 or q | Quit |

✅ **Arrow keys**: Move up/down or left/right  
✅ **Mouse support**: Click on columns or scroll easily

**📌 htop vs ps vs top**

| **Feature** | **ps** | **top** | **htop** |
| --- | --- | --- | --- |
| Snapshot | ✅ | ❌ | ❌ |
| Real-time | ❌ | ✅ | ✅ |
| Interactivity | ❌ | Moderate | High |
| Visual layout | ❌ | Minimal | Colorful |
| Mouse support | ❌ | ❌ | ✅ |
| Process tree | ❌ | ❌ | ✅ |

**🧠 Summary**

* htop is an **essential real-time monitoring tool**.
* Much easier and more user-friendly than top.
* Perfect for performance tuning, debugging, and process management.

**🌐 What is Network Management in Linux?**

**Network management** in Linux involves configuring, monitoring, troubleshooting, and controlling the network interfaces and services of the system. It includes:

* IP address configuration (static/dynamic)
* Interface management (up/down)
* DNS settings
* Routing and gateway setup
* Network status checking
* Testing connectivity
* Managing services like SSH, FTP, etc.

**🧰 Basic Network Commands in Linux**

| **Command** | **Use** |
| --- | --- |
| ip a or ip addr | Show IP addresses of interfaces |
| ip link | Show all network interfaces |
| ip route | Show routing table |
| ifconfig *(older)* | Display/set interface config |
| ping | Check connectivity to a host |
| traceroute | Track route packets take |
| netstat *(deprecated)* | Show open ports/connections |
| ss | Show sockets and connections |
| nmcli | NetworkManager CLI tool |
| ethtool | Get/set Ethernet device settings |
| dig, nslookup | DNS lookups |

**🧠 What is netstat?**

netstat (network statistics) is a powerful command-line tool used to **monitor network connections**, **routing tables**, **interface statistics**, **masquerade connections**, and **multicast memberships**.

⚠️ Note: netstat is **deprecated** on modern Linux distros in favor of the ss command, but it's still widely used and available.

**🔧 Basic Syntax**

netstat [options]

**🗂️ Most Commonly Used Options with Examples**

| **Option** | **Use** | **Example** |
| --- | --- | --- |
| -a | Show all sockets (listening + non-listening) | netstat -a |
| -t | Display TCP connections | netstat -at |
| -u | Display UDP connections | netstat -au |
| -l | Show only listening ports | netstat -l |
| -n | Show numeric IPs/ports (no DNS resolution) | netstat -n |
| -p | Show process ID and program name | netstat -p |
| -r | Display routing table | netstat -r |
| -i | Show network interfaces | netstat -i |
| -s | Show protocol statistics | netstat -s |
| -c | Continuous output (refresh every second) | netstat -c |

**✅ Most Useful netstat Command Combinations**

**1. Show all listening ports**

netstat -tuln

* t → TCP
* u → UDP
* l → Listening
* n → No DNS (faster)

**2. See which process is using which port**

sudo netstat -tulnp

→ Shows port → PID → application name  
Useful when checking if Apache, NGINX, MySQL, etc., are running.

**🧪 Real-Time Scenarios**

| **Scenario** | **Use netstat Command** |
| --- | --- |
| Check if Apache or NGINX is running | `netstat -tulnp |
| Find process using port 3306 (MySQL) | `netstat -tulnp |
| Check if server is listening on SSH | `netstat -tuln |
| View real-time incoming/outgoing TCP | netstat -ant |
| Identify open ports on a machine | netstat -tuln |
| Troubleshoot DNS or routing | netstat -rn or netstat -s |

**🧵 netstat vs ss (replacement)**

| **Feature** | **netstat** | **ss** |
| --- | --- | --- |
| Speed | Slower | Faster |
| Modern support | Deprecated | Actively supported |
| Show connections | ✅ | ✅ |
| Show processes | ✅ | ✅ (with -p) |
| Installed by default | Often yes | Yes (in newer distros) |

**🧠 Summary**

* netstat helps in **port, connection, and routing diagnostics**.
* Use -tulnp to monitor services.
* Combine with tools like ps, ping, and ss for complete network troubleshooting

**🏛️ What is 3-Tier Architecture?**

**3-tier architecture** is a software design pattern that separates an application into **three logical and physical layers**, each responsible for a specific function:

1. **Presentation Tier (Client/UI)**
2. **Application Tier (Business Logic)**
3. **Data Tier (Database/Storage)**

**📊 Diagram: 3-Tier Architecture Overview**

[Presentation Layer] <-- Web/App/Mobile UI

↓

[Application Layer] <-- Backend code (APIs, services)

↓

[Data Layer] <-- Database (MySQL, MongoDB, etc.)

**🧱 Tiers Explained**

**1️. Presentation Tier (Client/UI Layer)**

* What users **see and interact** with.
* Examples: Web browsers, mobile apps, desktop GUIs.
* Technologies: **HTML, CSS, JS, Angular, React, Flutter**.

💡 *Function:* Sends user requests to the application layer, receives results, and displays them.

**2️. Application Tier (Logic Layer)**

* Core business logic: **processes user input**, performs operations.
* It acts as a **bridge** between UI and database.
* Technologies: **Java, Python (Django/Flask), Node.js, .NET, Spring Boot**.

💡 *Function:* Validates input, applies logic, calls database, and returns results to the UI.

**3️. Data Tier (Database Layer)**

* Stores data: **relational or NoSQL** databases.
* Examples: **MySQL, PostgreSQL, MongoDB, Oracle DB**.

💡 *Function:* Responds to queries, stores persistent data, manages backups.

**🧪 Real-Time Example: Online Shopping Website**

| **Layer** | **Example Functionality** |
| --- | --- |
| **Presentation** | User browses product list, adds to cart |
| **Application** | Backend fetches products, applies discount logic |
| **Data** | DB stores products, orders, users, inventory |

**🚀 Advantages of 3-Tier Architecture**

| **Benefit** | **Explanation** |
| --- | --- |
| ✅ Separation of concerns | Each layer handles one job — easier to manage |
| ✅ Scalability | You can scale UI, logic, or DB independently |
| ✅ Security | Sensitive logic/data hidden behind app layer |
| ✅ Maintainability | Easier to update or replace layers |
| ✅ Reusability | Same backend/API can serve web and mobile apps |

**🔁 Real-Time Scenarios in DevOps/Cloud**

| **Use Case** | **How 3-Tier Helps** |
| --- | --- |
| Hosting a web app in AWS | Use **EC2/ALB** (presentation), **Lambda/ECS** (app), **RDS** (data) |
| Docker containers | UI container, app container, DB container |
| Kubernetes | Separate Pods/Deployments per layer |
| CI/CD pipelines | Independent build/deploy steps for each tier |
| Load balancing | Distribute frontend and backend layers separately |

**📌 Deployment Technologies per Layer**

| **Tier** | **Typical Technologies** |
| --- | --- |
| UI | Apache, Nginx, S3 static website |
| App | Node.js, Java Spring Boot, Python Flask (with Docker or EC2) |
| DB | MySQL, MongoDB, RDS, PostgreSQL |

**💬 Summary**

* **3-tier architecture = UI + Logic + Database**
* Enhances modularity, scalability, and security
* Used in nearly **all enterprise web applications**

**🌐 What is a Load Balancer?**

A **Load Balancer** distributes incoming network traffic across multiple servers to ensure high availability, reliability, and performance.

**🏗️ 3-Tier Architecture + Load Balancers**

**🔸 Tiers Overview**

1. **Presentation Tier** (Frontend): UI served to users (HTML, JS, etc.)
2. **Application Tier** (Backend): Business logic (APIs, processing)
3. **Data Tier** (Database): Stores data (MySQL, PostgreSQL, etc.)

**🔹 Load Balancer Types in 3-Tier Setup**

| **Tier** | **Load Balancer Type** | **Exposure** | **Purpose** |
| --- | --- | --- | --- |
| **Between Internet & Frontend** | **Public Load Balancer** | **Public (Internet-facing)** | Distributes incoming user requests to multiple frontend servers |
| **Between Frontend & Backend (App Servers)** | **Private Load Balancer** | **Private (internal)** | Routes requests from frontend servers to backend (API) servers |
| **Between Backend & Database** | Usually direct or uses DB cluster (not a load balancer) | **Private/Internal** | Databases often use internal replication/load balancing mechanisms |

**📘 Public Load Balancer (L1)**

* **Used For:** Accepting traffic from users on the internet
* **Example:** AWS Application Load Balancer (ALB) with a public IP

**🔹 Real-Time Example:**

A user opens your website in a browser:

* https://myapp.com → request hits **Public Load Balancer**
* LB routes to one of the frontend EC2s serving React app

**📘 Private Load Balancer (L2)**

* **Used For:** Internal communication (e.g., frontend → backend)
* **Example:** AWS ALB with **internal scheme**

**🔹 Real-Time Example:**

* React app in frontend sends API request to http://api.internal.local
* Request goes to **Private Load Balancer**
* Load balancer routes it to one of the backend EC2s running Flask or Node.js

**🔹 Diagram – 3-Tier with Public & Private Load Balancers**

I can provide a visual diagram showing:

* Public Load Balancer (Internet → Frontend)
* Private Load Balancer (Frontend → Backend)
* Backend accessing DB directly