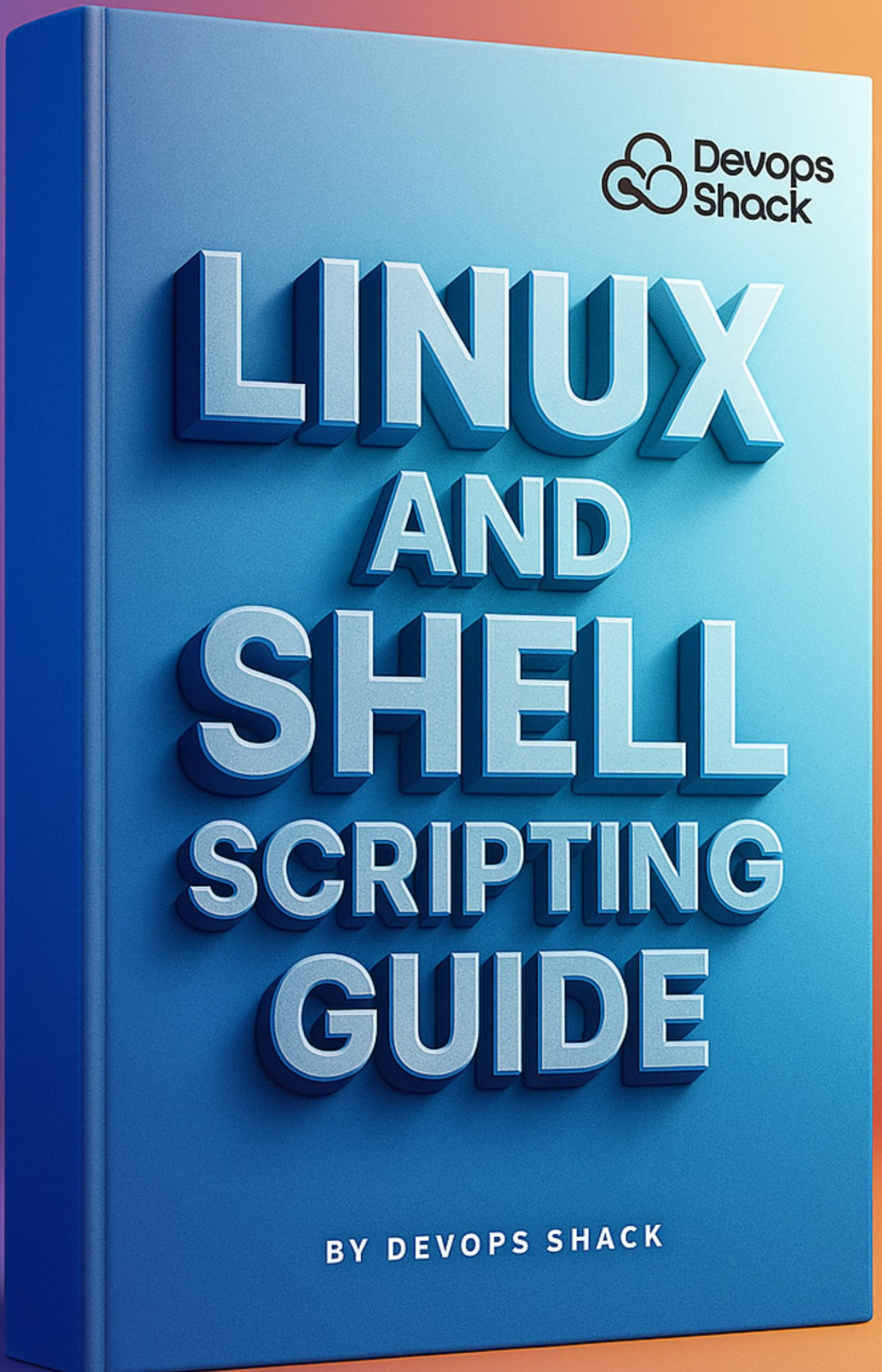

Devops Shack
LINUX AND SHELL SCRIPTING GUIDE
BY DEVOPS SHACK

Comprehensive Guide To Linux

MODULE 1: Introduction to Linux

- What is Linux? History, Evolution, and Impact
- GNU/Linux vs Unix
- Open Source Licensing (GPL, etc.)
- Use Cases: Servers, Desktops, Embedded Systems, Cloud, Mobile

MODULE 2: Linux Architecture

- Kernel Space vs User Space
- System Calls & the Kernel Interface
- Init Systems: SysVinit vs systemd
- How Processes are Created (fork, exec)

MODULE 3: Filesystem Hierarchy & Linux Directories

- FHS Standard and Purpose of Each Directory
- Deep Dive into /etc, /var, /home, /proc, /sys, /dev
- Mounting, fstab, and Virtual Filesystems

MODULE 4: Essential Linux Commands

- File Management (cp, mv, rm, mkdir, etc.)
- File Viewing (cat, less, head, tail, etc.)
- Text Processing (grep, awk, sed, cut)
- System Monitoring (top, ps, vmstat, iostat)
- Advanced Search (find, locate, xargs)

MODULE 5: User & Permission Management

- User/Group Creation & Configuration (useradd, groupadd)
- Permissions (chmod, chown, umask, ACLs)
- Special Bits: SUID, SGID, Sticky Bit

MODULE 6: Process Management

- Foreground/Background Jobs

- Signals, kill, nice, renice
- Process Trees & pstree, htop

MODULE 7: Package Management

- Debian-based: apt, dpkg
- RHEL-based: yum, dnf, rpm
- Source Installation, Snap, Flatpak

MODULE 8: Networking in Linux

- Interfaces: ip, ifconfig, nmcli
- DNS, Routing, Netmasking, Ports
- Tools: ping, netstat, ss, nmap, tcpdump, iptables

MODULE 9: Shell Scripting & Automation

- Variables, Conditionals, Loops
- Cron Jobs, Shell Functions, Exit Codes
- Real-World Automation Use Cases

MODULE 10: Systemd and Services

- Managing Services: systemctl, Units, Targets
- Journaling: journalctl, Log Rotation
- Boot Process and Recovery

MODULE 11: Disk, Partitions & LVM

- lsblk, fdisk, parted, mkfs, mount, df, du
- Logical Volume Management (LVM) Setup & Use
- RAID Concepts (mdadm)

MODULE 12: Linux Security

- File Permissions & Ownership
- Firewalls with iptables/firewalld
- SELinux/AppArmor Basics
- sudo vs su, PAM

MODULE 13: Logging & Troubleshooting

- /var/log Deep Dive
- Kernel Logs, dmesg, auditd
- Logrotate, Analyzing Logs for Root Cause

MODULE 14: Linux Performance Tuning

- CPU, Memory, I/O, and Network Performance Tools
- Kernel Parameters (sysctl)
- Swap Management & Hugepages

MODULE 15: Linux in the Cloud & Containers

- Running Linux on AWS, GCP, Azure
- Linux Containers: Docker, Podman
- Linux Inside Kubernetes Nodes

Module 1: Introduction to Linux

1.1 What is Linux?

Linux is a powerful, open-source **Unix-like** operating system kernel first developed by **Linus Torvalds** in **1991**. Unlike Windows or macOS, which are proprietary OSes, Linux is part of a broader ecosystem where **freedom**, **modularity**, and **transparency** are core principles.

◆ **Technical Definition:**

Linux = A **monolithic kernel** at the core of a Unix-like OS.
GNU/Linux = Linux kernel + GNU tools/utilities (bash, coreutils, etc.)

◆ **Characteristics:**

- **Multitasking**
- **Multi-user** support
- **Portability** across architectures (x86, ARM, RISC-V, etc.)
- **Security** via permission models, namespaces, cgroups
- **Stability** and **Uptime** (often runs for years without reboot)

1.2 History of Linux – A Revolution Born from Minix

Year	Milestone
1969	UNIX created at Bell Labs
1983	Richard Stallman launches GNU Project
1987	MINIX (educational Unix-like OS) released by Andrew Tanenbaum
1991	Linus Torvalds writes Linux kernel for x86 on MINIX
1992	Linux licensed under GPL (v2) — becoming truly open-source
1994	Linux 1.0 released
2000s	Enterprise adoption grows (Red Hat, SUSE)
2010s+	Cloud, IoT, Android, WSL, Containers use Linux extensively

---

1.3 GNU vs Linux vs UNIX vs POSIX

- **GNU** (GNU's Not Unix): Open-source tools intended to replace UNIX tools.
- **Linux**: Just the kernel originally. GNU + Linux = GNU/Linux (complete OS).
- **UNIX**: Proprietary operating systems like Solaris, AIX, HP-UX.
- **POSIX**: A standard for UNIX compatibility.

Think of Linux as a LEGO set — you can assemble your own operating system the way you want.

1.4 Why Use Linux?

Use Case	Example
Servers	95% of public cloud workloads (e.g., Ubuntu, RHEL, Amazon Linux)
Desktops	Ubuntu, Fedora, Pop!_OS
Mobile	Android is Linux-based
Embedded	Routers, TVs, Raspberry Pi
Containers	Alpine Linux, BusyBox
Supercomputing	100% of top 500 supercomputers run Linux

1.5 Key Benefits of Linux

Feature	Explanation
Open Source	Modify, inspect, and redistribute freely
Customizability	Build your own Linux distro (e.g., Arch, Gentoo)
Security	Powerful access control and namespaces
Community	Millions of contributors worldwide
Package Ecosystem	apt, yum, snap, flatpak, pacman
CLI Proficiency	Rich shell environments (bash, zsh, fish)

1.6 Linux vs Windows – Deep Comparison

Feature	Linux	Windows
Source Code	Open	Closed
User Interface	GUI + CLI	GUI-focused
Security Model	Permissions + Root + SELinux	ACL + UAC
Updates	Modular, less rebooting	Frequent reboots
Virus Risk	Lower	Higher

Feature	Linux	Windows
Performance	Lightweight, scalable	Heavy on resources
Target Users	Developers, sysadmins, hackers	General consumers

---

1.7 The Philosophy Behind Linux

- **Everything is a File**: Devices, sockets, pipes, configs — all treated as files
- **Small is Beautiful**: Programs do one thing and do it well
- **Text is the Universal Interface**: Use plain-text config files, scripts, and logs
- **Transparency**: Logs, processes, services — everything is inspectable

---

1.8 Linux Distributions (Distros)

A **distribution** = Linux Kernel + Package Manager + Default Utilities + Desktop/CLI + System Configs

Popular Distros:

Category	Examples
General Use	Ubuntu, Debian, Fedora
Enterprise	Red Hat (RHEL), SUSE, Oracle Linux
Lightweight	Alpine, Arch, Puppy Linux
Cloud/Server	Ubuntu Server, Amazon Linux, CentOS Stream
Hacker/Pentest	Kali Linux, ParrotOS
Container-Optimized	Alpine, Distroless

1.9 Who Maintains Linux?

- **Linus Torvalds**: Still leads kernel development

- **Linux Foundation**: Coordinates funding, security, and enterprise support
- **Thousands of contributors** from companies like Google, Red Hat, Intel, Meta, Microsoft

Interesting fact: Microsoft is now one of the top 10 contributors to the Linux kernel.

1.10 Real-World Applications of Linux

- **Cloud Platforms**: AWS, Azure, GCP all use Linux underneath
- **Web Servers**: Apache, NGINX, HAProxy, Node.js — all commonly run on Linux
- **DevOps Tools**: Docker, Kubernetes, Jenkins, GitLab — all built for Linux first
- **AI/ML/Dev Environments**: TensorFlow, PyTorch — commonly deployed on Ubuntu/Debian
- **Gaming**: Steam Deck uses Arch Linux; Proton enables Windows game compatibility

---

1.11 Linux Today: Stats and Impact

Metric	Value
Cloud Workloads	~96% run on Linux
Supercomputers	100% run Linux
Mobile Market	75% (Android) is Linux-based
Web Servers	~70% use Linux

1.12 Challenges in Using Linux (For Beginners)

- Initial learning curve (especially CLI-based interaction)
- Hardware driver support for niche devices
- Gaming ecosystem (improving steadily via Proton)
- Fragmentation of distributions (many choices can confuse newcomers)

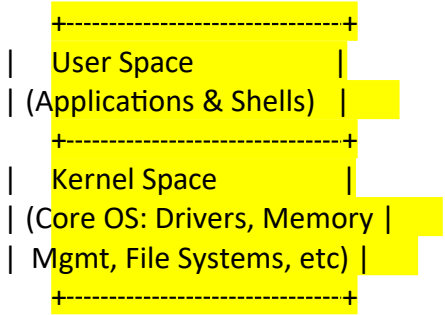
1.13 Summary & What's Next

Linux is **not just an OS**; it's an **ecosystem**, a **philosophy**, and the **foundation of modern computing** — from smartphones and supercomputers to containers and clouds.

Module 2: Linux Architecture

2.1 Overview of Linux Architecture

At a high level, Linux architecture is divided into two main domains:



- **User Space**: Programs, GUIs, CLI tools, shells, services
- **Kernel Space**: The core OS that interacts directly with hardware

The **kernel** is the heart — managing CPU, memory, I/O, filesystems, and inter-process communication (IPC).

2.2 The Linux Kernel: What It Actually Does

The kernel performs 5 primary functions:

Subsystem	Function
Process Scheduler	Manages CPU time for processes
Memory Manager	Allocates RAM, handles swapping
File System Interface	Mounts and manages filesystems (ext4, xfs, btrfs)
Device Drivers	Interfaces with hardware (disks, NICs, GPUs)
System Call Interface	Provides APIs for user applications to interact with kernel

System calls like open(), read(), fork() are how userspace apps request services from the kernel.

2.3 Linux Kernel Types

Kernel Type	Description
Monolithic	Everything (drivers, memory, scheduling) in one large binary – Linux follows this
Microkernel	Minimal core + userspace services (e.g., MINIX, QNX)
Hybrid	Mix of monolithic + micro (e.g., Windows NT, macOS XNU)

2.4 Boot Process in Detail (BIOS to Shell)

The Linux boot process involves the following sequence:

1. **BIOS/UEFI**: Power-On Self Test (POST), locates bootloader
2. **Bootloader** (e.g., GRUB, LILO): Loads the kernel image (vmlinuz)
3. **Kernel Initialization**: Mounts root filesystem, starts PID 1
4. **Init System** (systemd, SysVinit, upstart):
    - Mounts additional partitions
    - Initializes services (systemctl start nginx)
    - Brings system to a target (multi-user, graphical)

Key Files Involved:

- /boot/vmlinuz-* → Kernel binary
- /boot/initrd.img → Init RAM disk
- /etc/fstab → Filesystem mount instructions
- /etc/systemd/system/*.service → Services configuration

2.5 Init Systems: systemd vs Others

Feature	systemd	SysVinit	Upstart
Parallel Startup	Yes	No	Yes
Socket Activation	Yes	No	Yes
Service Dependencie	Yes	No	Yes
Default in	RHEL 7+, Ubuntu 15+	Old Debian/RHEL	Ubuntu 14

Run systemctl list-units or systemctl status nginx to inspect service states.

2.6 Understanding System Calls

System calls are how user apps interact with the kernel:

Category	Examples
File	open(), read(), write(), close()
Process	fork(), exec(), wait()
Memory	mmap(), brk()
Signals	kill(), signal()
Networking	socket(), bind(), connect()

Use strace ls to trace system calls made by ls command.

2.7 How Linux Creates a Process (fork-exec model)

1. **fork()** → Clones current process (parent + child)

2. **exec()** → Child replaces memory with new binary
3. **wait()** → Parent waits for child completion

Example in C:

```
pid_t pid = fork();
if (pid == 0) {
    execlp("/bin/ls", "ls", NULL); // child
} else {
    wait(NULL); // parent
    }
```

Run pstree -p to visualize process trees

2.8 Threads vs Processes in Linux

Feature	Process	Thread
PID	Unique	Shared within same PID (uses TID)
Memory	Isolated	Shared
Scheduling	Independent	Shared timeslice
Tools	ps, top, kill	htop, ps -eLf, pthread

Linux implements threads using **clone()** syscall — making them lightweight.

2.9 Interrupts, Context Switching, Scheduling

- **Interrupt**: Hardware signals the CPU (e.g., network packet arrival)
- **Context Switch**: CPU saves state of current process and switches to another
- **Schedulers**:
    - **CFS** (Completely Fair Scheduler): Default
    - **Real-Time**: SCHED_FIFO, SCHED_RR

Use vmstat, pidstat, and top to observe context switching.

2.10 Virtual Memory Management

- **Paging**: Physical memory divided into pages
- **Swapping**: Move inactive pages to disk
- **mmap()**: Map files or devices to memory

Tools:

- free, top, /proc/meminfo, vmstat

2.11 Kernel Modules

Kernel can dynamically load/unload modules (drivers) at runtime:

Command	Purpose
lsmod	List loaded modules
modprobe e1000	Load Ethernet driver
rmmod	Remove a module
/lib/modules/<kernel>/	Location of modules

2.12 The /proc and /sys Virtual Filesystems

- /proc: Process and kernel information as files
    - /proc/cpuinfo, /proc/meminfo, /proc/<PID>/
- /sys: Exposes kernel objects (devices, buses, modules)

These are essential for monitoring and debugging system

state.

2.13 Filesystems and VFS Layer

- **VFS (Virtual File System)** provides abstraction over various file systems.
- Supported filesystems: ext4, xfs, btrfs, vfat, ntfs, proc,

tmpfs Mount Example:

```
sudo mount -t ext4 /dev/sda1 /mnt
```

2.14 Devices and Udev

- Linux treats devices as files: /dev/sda, /dev/null, /dev/random
- **udev** dynamically manages device nodes
- Use udevadm monitor to watch real-time device events

2.15 Summary

Linux's architecture is highly modular, efficient, and open. Whether it's process management, memory allocation, or filesystem abstraction, the Linux kernel orchestrates everything through cleanly defined interfaces and syscalls — giving you unmatched power and flexibility.

Module 3: Filesystem Hierarchy & Linux Directories

3.1 Filesystem Hierarchy Standard (FHS) – The Foundation

The **FHS** defines the directory structure of Linux. It ensures consistent placement of files and directories across distributions.

Top-level layout:

```
/
├── bin/
├── boot/
├── dev/
├── etc/
├── home/
├── lib/
├── media/
├── mnt/
├── opt/
├── proc/
├── root/
├── run/
├── sbin/
├── srv/
├── sys/
├── tmp/
├── usr/
└── var/
```

Each of these directories has a **specific purpose**, which we'll now explore in extreme detail.

3.2 / – The Root Directory

- Top of the directory tree.
- Everything starts from here.
- Mounted first during boot.

Analogy: Like C:\ in Windows — but in Linux, all disks mount under this tree.

3.3 /bin – Essential Binaries

- Stores basic user commands needed in **single-user mode**.
- Available **before /usr is mounted**.

- Examples: ls, cat, cp, mv, rm, echo, bash

Try:

ls /bin

3.4 /sbin – System Binaries

- Contains critical **system administration** binaries.
- Typically used by root or scripts during boot/init.
- Examples: init, ifconfig, reboot, fsck, iptables

3.5 /etc – System Configuration

- Contains **host-specific system-wide config files**.
- Should NOT contain binaries or user files.
- Examples:
  - /etc/fstab – Mount points
  - /etc/hostname – System name
  - /etc/passwd – User accounts
  - /etc/ssh/sshd_config – SSH settings

Explore:

grep bash /etc/passwd

3.6 /dev – Device Files

- Treats devices as **files**: hard disks, terminals, USBs
- Managed by **udev**
- Examples:
  - /dev/sda1 – First disk partition
  - /dev/null – Bit bucket
  - /dev/random – Random number generator

Cp* This abstraction allows Linux to interact with hardware uniformly.

3.7 /proc – Process and Kernel Info (Virtual Filesystem)

- Dynamically generated system info.
- Mounts at runtime; not stored on disk.
- Examples:
    - /proc/cpuinfo
    - /proc/meminfo
    - /proc/<PID>/status

Try:

cat /proc/meminfo

3.8 /sys – Kernel Device Tree (sysfs)

- Interface between kernel and userspace
- Used to configure devices and kernel parameters
- Example: /sys/class/net/ lists all

interfaces Try:

cat /sys/class/net/eth0/address

3.9 /lib, /lib64 – Shared Libraries

- Required for binaries in /bin and /sbin
- Holds .so shared object files (dynamic libraries)
- Contains kernel modules: /lib/modules/$(uname -

r)/ Do NOT confuse with /usr/lib.

3.10 /home – User Directories

- Each user gets a personal space: /home/alice, /home/bob
- Default shell config files: .bashrc, .profile

- Permissions ensure user

isolation Try:

ls -la /home

3.11 /root – Superuser's Home

- Root user's personal home (not /home/root)
- Only accessible by root
- Useful during system recovery

Don't confuse / (root directory) with /root (root's home).

3.12 /boot – Boot Loader Files

- Contains kernel, initrd, and GRUB bootloader
- Key files:
  - vmlinuz-* – Kernel binary
  - initrd.img – Initial RAM disk
  - grub.cfg – Boot menu configuration

Try:

ls /boot

3.13 /run – Volatile Runtime Data

- Stores process ID files, sockets, and other temp info
- Cleared at every reboot
- Replaces older /var/run

3.14 /tmp – Temporary Files

- Used by programs to store temporary data
- Auto-cleared on reboot
- World-writable

Try:

touch /tmp/testfile

3.15 /var – Variable Data

- Logs, spool files, PID files, mail queues, databases
- Grows over time
- Key subdirs:
  - /var/log – Logs
  - /var/spool – Print/mail queues
  - /var/tmp – Persistent temporary files

Try:

tail -f /var/log/syslog

3.16 /usr – User System Resources

- Secondary hierarchy (mounted post boot)
- Contains non-essential binaries, libraries, docs
- Key subdirs:
  - /usr/bin – Most user commands
  - /usr/sbin – Sysadmin commands
  - /usr/lib – Shared libraries
  - /usr/local – Locally installed software

/usr ≠ “user” — it means **Unix System Resources**

3.17 /opt – Optional Packages

- For installing 3rd-party software (like Oracle, Chrome)
- Self-contained apps that don't follow FHS
- Example: /opt/google/chrome/

3.18 /media and /mnt – Mount Points

Directory	Purpose
/media	Auto-mounted external devices (USB, CD-ROM)
/mnt	Temporary mounts (e.g., ISO, disks for troubleshooting)

3.19 /srv – Service Data

- Stores site or service-specific files
- Used in enterprise setups
- Example: /srv/www/, /srv/ftp/

3.20 Permissions, Mounts, and Ownership

Use these commands to manage file structures:

Command	Usage
mount	Mount filesystems
umount	Unmount safely
df -h	Disk usage
du -sh *	Space used by directories
ls -l	Permissions and ownership

3.21 How to Explore the Hierarchy Safely

Try:

tree -L 2 /

Or use ncdu for visual disk usage exploration.

3.22 Summary

The Linux filesystem hierarchy is **not random**. Every directory has a **well-defined purpose**. Knowing where logs, binaries, configs, and runtime files live is crucial for **troubleshooting**, **automation**, and **system hardening**.

Module 4: Essential Linux Commands

4.1 Why Master Linux Commands?

Mastering Linux commands is the backbone of system administration, DevOps, shell scripting, troubleshooting, and automation. It helps you:

- Reduce dependence on GUI tools
- Work across any Linux server or cloud VM
- Build scripts, pipelines, and monitoring tools
- Debug systems quickly and efficiently

🔹 **Section 1: File and Directory Management**

🔸 **ls – List Directory Contents**

```
ls -lah /etc
```

- -l: Long listing
- -a: Show hidden files
- -h: Human-readable sizes

Pro Tip: Use ls --color=auto for syntax highlighting.

🔸 **cd – Change Directory**

```
cd /var/log
cd ~       # Go to home
cd -       # Go to previous directory
```

🔸 **pwd – Show Current Directory**

```
pwd
```

🔸 **mkdir – Make Directories**

```
mkdir -p /tmp/project/{logs,src,bin}
```

Creates nested dirs using brace expansion.

🔸 **rm – Remove Files/Directories**

```
rm -rf /tmp/project/
```

- -r: Recursive
- -f: Force (no prompt)

❗ *Be extremely careful with rm -rf.*

🔸 **cp – Copy Files**

```
cp file.txt /tmp/
```

```
cp -r /etc /tmp/backup/
```

🔸 **mv – Move/Rename Files**

```
mv old.txt new.txt
```

```
mv /tmp/file.txt ~/Downloads/
```

🔹 **Section 2: Viewing and Editing Files**

🔸 **cat, tac, more, less**

```
cat file.txt
```

```
tac file.txt          # Reverse lines
```

```
less /var/log/syslog   # Page view with scroll
```

🔸 **head and tail**

```
head -n 20 /etc/passwd
```

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

- -f: Follow logs in real time

🔸 nano / vim – Text Editors

```
nano file.txt
```

```
vim file.txt
```

🔹 Section 3: Searching Files & Content

🔸 find – Locate Files Based on Conditions

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

```
find . -type f -size +10M
```

🔸 grep – Search Inside Files

```
grep "error" /var/log/syslog
```

```
grep -r "port 80" /etc
```

- -r: Recursive
- -i: Ignore case
- -A3: Show 3 lines after match

🔸 locate – Fast File Lookup

```
locate nginx.conf
```

- Needs updatedb to be run for indexing.

🔹 Section 4: Process & Resource Monitoring

🔸 ps – View Running Processes

```
ps aux | grep nginx
```

- **top and htop – Interactive Process View**

top

htop # Requires installation

- **kill, pkill, killall**

kill -9 <pid>

pkill nginx

- **free, vmstat, iostat**

free -h

vmstat 1 5

iostat -xz 1 5

Track memory, CPU, and disk I/O.

- **Section 5: Disk & Filesystem Usage**

- **df – Disk Free Space**

df -hT

- **du – Directory Size**

du -sh /var/*

- **mount, umount, lsblk, blkid**

lsblk

mount /dev/sdb1 /mnt

- **Section 6: Permission & Ownership**

🔸 **chmod – Change Permissions**

```
chmod 755 script.sh
```

```
chmod u+x file.sh
```

🔸 **chown – Change Ownership**

```
chown user:group file.txt
```

🔸 **umask – Default Permission Mask**

```
umask      # Show current
```

```
umask 022  # Set new default
```

🔹 **Section 7: Package Management**

Debian/Ubuntu:

```
sudo apt update
```

```
sudo apt install nginx
```

```
sudo apt remove apache2
```

RHEL/CentOS:

```
sudo yum install httpd
```

```
sudo dnf remove nginx
```

🔹 **Section 8: Archiving & Compression**

🔸 **tar, gzip, xz**

```
tar -czvf archive.tar.gz /home/user/
tar -xvf archive.tar.gz
```

- **zip and unzip**

```
zip -r project.zip folder/
unzip project.zip
```

- **Section 9: User & Group Commands**

- **User Management**

```
useradd devops
passwd devops
usermod -aG docker devops
```

- **Group Management**

```
groupadd admins
gpasswd -a user admins
```

- **Section 10: Network Commands**

- **ip and ifconfig**

```
ip a
ip r
```

- **ping, netstat, ss, nmap**

```
ping 8.8.8.8
ss -tulnp
nmap -sT localhost
```

🔹 Section 11: Misc Tools

🔸 date, uptime, hostname

```
date
uptime
hostname -I
```

🔸 history, alias, whoami

```
history | grep ssh
alias ll='ls -lAh'
whoami
```

🔸 watch, xargs, tee

```
watch -n 1 df -h
ls *.log | xargs grep "fatal"
df -h | tee disk_report.txt
```

Pro Tips for Command-Line Ninjas

- Use !! to repeat last command
- Use Ctrl + R for reverse history search
- Use tab for auto-completion
- Combine commands using &&, ||, and |
- Create .bash_aliases for common commands

Summary

Mastering these commands lets you:

- Administer Linux servers confidently
- Automate daily tasks with shell scripts
- Debug and recover broken systems
- Succeed in Linux interviews and real-world DevOps work

Module 5: File/Folder Permissions in Linux

5.1 Why Permissions Matter

In a multi-user system, Linux enforces strict access controls to:

- Prevent unauthorized data access
- Protect system files from accidental changes
- Enforce isolation and

privacy Linux uses:

- **User (u)** – File owner
- **Group (g)** – Group with shared access
- **Others (o)** – Everyone else
- **Permission types** – r (read), w (write), x (execute)

5.2 Viewing Permissions with ls -l

```
ls -l myfile.txt
```

Output:

```
-rwxr-xr-- 1 aditya devops 4096 Apr 1 12:00 myfile.txt
```

Explanation:

- - = regular file (d for directory, l for symlink)
- rwx = user (owner) permissions
- r-x = group permissions
- r-- = others' permissions

5.3 Changing Permissions with chmod

Numeric (Octal) Method:

```
chmod 755 script.sh
```

Entity	Binary	Octal	Meaning
rwx	111	7	Read, Write, Execute
rw-	110	6	Read, Write
r--	100	4	Read Only

Examples:

chmod 700 secret.txt # Only owner can access

chmod 644 file.txt # Owner can write, others read-only

Symbolic Method:

chmod u+x file.sh # Add execute to user

chmod go-r file.txt # Remove read from group and others

5.4 Changing Ownership with chown

chown user file

chown user:group

file Examples:

chown devops:admins script.sh

5.5 Changing Group Ownership with chgrp

chgrp devops file.txt

5.6 Understanding Default umask

umask defines default permissions for new files/directories.

Default Mask	File Permission	Directory Permission
0022	644 (rw-r--r--)	755 (rwxr-xr-x)

View & Change:

Umask

umask 0077

5.7 Special Permissions: SUID, SGID, Sticky Bit

- **SUID (Set User ID)**
  - When executed, runs with **file owner's** privileges
  - Common for commands needing root temporarily (e.g. passwd)

```
chmod u+s /usr/bin/passwd
ls -l /usr/bin/passwd
# -rwsr-xr-x
```

- **SGID (Set Group ID)**
  - On files: runs with **file group's** permissions
  - On directories: new files inherit the **group** of the directory

```
chmod g+s /shared
ls -ld /shared
# drwxr-sr-x
```

- **Sticky Bit**
  - Applied on **directories** to restrict deletion to owner only
  - Common in /tmp

```
chmod +t /shared/tmp
ls -ld /shared/tmp
# drwxrwxrwt
```

5.8 Real-World Use Cases

Scenario	Solution
Secure temp storage	Use sticky bit
Share folder but protect group ownership	Use SGID on folder

Scenario	Solution
Limit command execution	Use chmod, chown, sudo
Run scripts as root without sudo	Use SUID carefully (only on safe binaries)

5.9 Access Control Lists (ACLs)

ACLs offer **granular permission** beyond owner/group/others.

Enable ACLs (if not default):

```
mount -o remount,acl /
```

Set ACLs:

```
setfacl -m u:devops:r-- file.txt
```

```
getfacl file.txt
```

5.10 Directory Permissions Explained

Permission	Effect
r	List contents
w	Create, delete, rename files
x	Access files inside (requires r to list them)

A user can access a file **only if they have execute permission on all parent directories**.

5.11 View All Users' Access with namei

```
namei -l /var/www/html/index.html
```

Shows permissions on **each directory in the path**.

5.12 Permission Troubleshooting Tips

Problem	Cause
Permission denied	No x on directory or no r on file
Operation not permitted	File owned by root/user has no rights
Cannot delete	Sticky bit or wrong w permission

🛠 Use ls -l, namei, stat, getfacl to debug.

5.13 Summary

Linux permissions are the foundation of a secure system. Mastering chmod, chown, umask, ACLs, and special bits like SUID/SGID/sticky gives you surgical control over **who can do what** and protects systems from misconfiguration or intrusion.

Module 5: File/Folder Permissions in Linux – Ultra Deep Dive

5.1 Why Permissions Matter

In a multi-user system, Linux enforces strict access controls to:

- Prevent unauthorized data access
- Protect system files from accidental changes
- Enforce isolation and

privacy Linux uses:

- **User (u)** – File owner
- **Group (g)** – Group with shared access
- **Others (o)** – Everyone else
- **Permission types** – r (read), w (write), x (execute)

5.2 Viewing Permissions with ls -l

```
ls -l myfile.txt
```

Output:

```
-rwxr-xr-- 1 aditya devops 4096 Apr 1 12:00 myfile.txt
```

Explanation:

- - = regular file (d for directory, l for symlink)
- rwx = user (owner) permissions
- r-x = group permissions
- r-- = others' permissions

5.3 Changing Permissions with

chmod Numeric (Octal) Method:

```
chmod 755 script.sh
```

Entity	Binary	Octal	Meaning
rwx	111	7	Read, Write, Execute
rw-	110	6	Read, Write
r--	100	4	Read Only

Examples:

```
chmod 700 secret.txt    # Only owner can access
chmod 644 file.txt      # Owner can write, others read-only
```

Symbolic Method:

```
chmod u+x file.sh     # Add execute to user
chmod go-r file.txt  # Remove read from group and others
```

5.4 Changing Ownership with chown

chown user file

chown user:group

file Examples:

```
chown devops:admins script.sh
```

5.5 Changing Group Ownership with chgrp

```
chgrp devops file.txt
```

5.6 Understanding Default umask

umask defines default permissions for new files/directories.

Default Mask	File Permission	Directory Permission
0022	644 (rw-r--r--)	755 (rwxr-xr-x)

View & Change:

```
umask
umask 0077
```

5.7 Special Permissions: SUID, SGID, Sticky Bit

- **SUID (Set User ID)**
  - When executed, runs with **file owner's** privileges
  - Common for commands needing root temporarily (e.g. passwd)

```
chmod u+s /usr/bin/passwd
ls -l /usr/bin/passwd
# -rwsr-xr-x
```

- **SGID (Set Group ID)**
  - On files: runs with **file group's** permissions
  - On directories: new files inherit the **group** of the directory

```
chmod g+s /shared
ls -ld /shared
# drwxr-sr-x
```

- **Sticky Bit**
  - Applied on **directories** to restrict deletion to owner only
  - Common in /tmp

```
chmod +t /shared/tmp
ls -ld /shared/tmp
# drwxrwxrwt
```

5.8 Real-World Use Cases

Scenario	Solution
Secure temp storage	Use sticky bit
Share folder but protect group ownership	Use SGID on folder

Scenario	Solution
Limit command execution	Use chmod, chown, sudo
Run scripts as root without sudo	Use SUID carefully (only on safe binaries)

5.9 Access Control Lists (ACLs)

ACLs offer **granular permission** beyond owner/group/others.

Enable ACLs (if not default):

```
mount -o remount,acl /
```

Set ACLs:

```
setfacl -m u:devops:r-- file.txt
```

```
getfacl file.txt
```

5.10 Directory Permissions Explained

Permission	Effect
r	List contents
w	Create, delete, rename files
x	Access files inside (requires r to list them)

A user can access a file **only if they have execute permission on all parent directories**.

5.11 View All Users' Access with namei

```
namei -l /var/www/html/index.html
```

Shows permissions on **each directory in the path**.

5.12 Permission Troubleshooting Tips

Problem	Cause
Permission denied	No x on directory or no r on file

Problem	Cause
Operation not permitted	File owned by root/user has no rights
Cannot delete	Sticky bit or wrong w permission

Use ls -l, namei, stat, getfacl to debug.

5.13 Summary

Linux permissions are the foundation of a secure system. Mastering chmod, chown, umask, ACLs, and special bits like SUID/SGID/sticky gives you surgical control over **who can do what** and protects systems from misconfiguration or intrusion.

Module 6: User and Group Management in Linux

6.1 Why User & Group Management Is Critical

In multi-user systems like servers, CI/CD machines, and production boxes, managing access based on roles, privileges, and group policies ensures:

- Accountability
- Least privilege enforcement
- Security hardening
- Separation of duties

🔹 Section 1: Understanding User Accounts

🔸 Types of Users

User Type	Description
Root	Superuser (UID=0) – full control
System Users	Non-login accounts for services (e.g., nginx, mysql)
Regular Users	Interactive human users (UID >= 1000 on most distros)

System users can be seen using awk -F: '$3 < 1000' /etc/passwd

---

🔸 User Account Fields (/etc/passwd)

Each line:

username:x:UID:GID:comment:home:shell

Example:

aditya:x:1001:1001:Aditya Jaiswal:/home/aditya:/bin/bash

- UID = User ID
- GID = Primary group
- /etc/shadow stores encrypted passwords

- /etc/group stores group memberships

🔹 Section 2: Creating & Managing Users

🔸 Create a User

```
sudo useradd -m -s /bin/bash devopsuser
```

- -m: Create home dir
- -s: Shell path

Add password:

```
sudo passwd devopsuser
```

🔸 Add User to Groups

```
usermod -aG docker,developers devopsuser
```

- -aG: Append to secondary groups

🔸 Delete a User

```
sudo userdel -r devopsuser
```

- -r: Remove home directory too

🔸 Modify Existing User

```
usermod -l newname oldname          # Change username
usermod -d /newhome -m username     # Move home dir
```

🔹 Section 3: Managing Groups

- **Create**

groupadd devops

- **Add Users to Group**

gpasswd -a aditya devops

- **Delete Group**

groupdel devops

- **List Groups for a User**

groups aditya

id aditya

- **Section 4: Files That Control User Management**

- **/etc/passwd – User metadata**
- **/etc/shadow – Passwords (hashed + aging info)**
- **/etc/group – Group memberships**
- **/etc/gshadow – Secure group passwords**

- **View a User's Entry**

grep devopsuser /etc/passwd

grep devopsuser /etc/shadow

- **Section 5: Controlling Login Access**

🔸 Lock/Unlock User Account

passwd -l username # Lock

passwd -u username # Unlock

🔸 Expire Account (set expiry date)

chage -E 2025-12-31 devopsuser

🔸 Set Password Expiry Policy

chage -M 90 -m 7 -W 14 devopsuser

- -M: Max days
- -m: Min days
- -W: Warning days before

expiry View with:

chage -l devopsuser

🔸 Disable Shell Login

usermod -s /usr/sbin/nologin apache

🔹 Section 6: Skel Directory & Default Configs

When you create a user with -m, contents from /etc/skel are copied into their home directory:

/etc/skel/.bashrc

/etc/skel/.profile

You can customize these to set default env, aliases, PS1 prompt, etc.

🔹 Section 7: Advanced Account Controls

Set User UID or Home Directory

```
useradd -u 1050 -d /srv/ciuser ciuser
```

Set Primary Group

```
useradd -g devops ciuser
```

Assign Multiple Groups

```
usermod -G devops,docker ciuser
```

Section 8: Temporary User Access (Time-Limited)

Use at or cron to schedule removal:

```
echo "userdel -r testuser" | at 23:00
```

Section 9: Login Logs and Monitoring

Check Login Attempts

```
last
```

Failed Logins

```
lastb
```

Who is Logged In?

```
w
```

```
who
```

Section 10: Sudo Access Control

Give User Sudo Access

Add to sudo group:

```
usermod -aG sudo devopsuser
```

Or edit sudoers safely:

```
visudo
```

Example

entry:

```
devopsuser ALL=(ALL) NOPASSWD:ALL
```

◆ **Restrict Commands with sudo**

```
devopsuser ALL=(ALL) /usr/bin/systemctl restart nginx
```

Summary

User and group management in Linux is more than adding users — it's about defining **access boundaries**, **group roles**, **login security**, and **responsible privilege use**. Mastering this helps you enforce **least privilege**, secure environments, and automate account lifecycles in corporate environments.

Module 7: Package Management in Linux

7.1 Introduction to Package Management

In Linux, software is distributed in **packages** — collections of binaries, libraries, and metadata. Package managers handle:

- Installation and uninstallation
- Dependency resolution
- Upgrades and version management
- Access to remote repositories

There are two main families of package systems:

- **Debian-based**: .deb packages (apt, dpkg)
- **Red Hat-based**: .rpm packages (yum, dnf, rpm)

7.2 Debian-based Systems (Ubuntu,

Debian) apt – Advanced Package Tool

Used for package management via CLI.

Update repositories

```
sudo apt update
```

Upgrade all packages

```
sudo apt upgrade
```

Install a package

```
sudo apt install nginx
```

Remove a package (retain configs)

```
sudo apt remove nginx
```

Purge package (delete configs too)

```
sudo apt purge nginx
```

Search for a package

```
apt search docker
```

Show package details

apt show curl

dpkg – Low-level Debian package tool

Used to manage individual .deb packages directly.

Install a .deb package manually

sudo dpkg -i package.deb

Fix broken dependencies

sudo apt install -f

List installed packages

dpkg -l

Check package ownership of file

dpkg -S /usr/bin/wget

7.3 Red Hat-based Systems (CentOS, RHEL,

Fedora) yum – Yellowdog Updater Modified

Install a package

sudo yum install nginx

sudo yum remove nginx

List available updates

yum check-update

Update packages

sudo yum update

List installed packages

yum list installed

dnf – Modern replacement for yum

Used in Fedora and RHEL 8+ systems.

Install, remove, search, list

sudo dnf install tree

sudo dnf remove httpd

dnf search nodejs

View package info

dnf info nginx

Clean up cache

dnf clean all

7.4 rpm – Low-level RPM

tool Install

sudo rpm -ivh package.rpm

Upgrade

sudo rpm -Uvh package.rpm

Erase

sudo rpm -e package-name

List installed

rpm -qa

Query file ownership

rpm -qf /etc/nginx/nginx.conf

7.5 Package Repositories

Repositories are remote locations that store packages and metadata. They are defined in:

- Debian: /etc/apt/sources.list or /etc/apt/sources.list.d/
- Red Hat: /etc/yum.repos.d/

Add a custom repository (Debian)

sudo add-apt-repository ppa:graphics-drivers/ppa

sudo apt update

Add a .repo file (RHEL)

sudo vi /etc/yum.repos.d/custom.repo

7.6 Third-party Tools

Snap: Canonical's cross-distro packaging format

```
sudo snap install postman
```

Flatpak: Popular in Fedora/RedHat GUI apps

```
flatpak install flathub org.gimp.GIMP
```

AppImage: Standalone portable Linux apps

```
chmod +x myapp.AppImage
./myapp.AppImage
```

7.7 Best Practices

- Always update before installing
- Prefer package managers over manual builds
- Use apt-mark hold or dnf versionlock to freeze critical packages
- Use autoremove to clean unused dependencies
- Enable GPG signature verification for repo security

7.8 Troubleshooting Package Issues

Broken dependencies

```
sudo apt install -f
sudo dnf repoquery --unsatisfied
```

Corrupted cache

```
sudo apt clean && sudo apt update
sudo dnf clean all && sudo dnf makecache
```

Conflicting versions

```
apt-cache policy nginx
```

7.9 Summary

Linux package management is a powerful and flexible system for software delivery and updates. Whether you're using apt, yum, dnf, rpm, snap, or flatpak, mastering the ecosystem ensures you can manage software at scale, automate provisioning, and control environments precisely.

Module 8: Linux Networking Commands and Tools

8.1 Introduction to Linux Networking

Linux provides a comprehensive suite of CLI tools and utilities to:

- Configure and inspect network interfaces
- Analyze and debug connectivity issues
- Monitor bandwidth and open ports
- Test DNS, routing, and latency
- Apply firewall and socket-level control

8.2 Network Interface Configuration

View interfaces (modern command)

```
ip a
```

```
ip addr show
```

Assign IP manually

```
sudo ip addr add 192.168.1.50/24 dev eth0
```

Remove IP

```
sudo ip addr del 192.168.1.50/24 dev eth0
```

Bring interface up/down

```
sudo ip link set eth0 up
```

```
sudo ip link set eth0 down
```

8.3 Legacy Commands (ifconfig, net-tools) ifconfig (deprecated but still useful)

```
ifconfig
```

```
ifconfig eth0 up
```

View routing table

```
route -n
```

8.4 Modern Replacement – ip command

suite Show default gateway

ip route show

Add default gateway

sudo ip route add default via 192.168.1.1

Delete default route

sudo ip route del default

8.5 DNS Tools

Resolve domain name to IP

nslookup google.com

dig google.com

host google.com

View DNS settings

cat /etc/resolv.conf

8.6 Connection Testing

Tools ping – ICMP

reachability

ping 8.8.8.8

ping google.com

traceroute – Path tracing

traceroute google.com

curl – HTTP requests and APIs

curl -I https://www.example.com

curl -X POST -d "key=value" https://api.example.com

wget – Download over HTTP/S/FTP

```
wget https://www.kernel.org/pub/linux/kernel/v6.x/linux-6.0.tar.xz
```

8.7 Port & Socket Analysis

ss – Socket statistics (modern replacement for netstat)

```
ss -tuln
```

```
ss -ap
```

netstat (legacy)

```
netstat -tulnp
```

Check open ports

```
ss -ltnp
```

Find listening port for a specific process

```
lsof -i :8080
```

8.8 Network Traffic Monitoring

tcpdump – Packet capture tool

```
sudo tcpdump -i eth0 port 80
```

tshark – CLI version of Wireshark

```
sudo tshark -i eth0
```

iftop – Live bandwidth monitor

```
sudo iftop -i eth0
```

iperf3 – Measure bandwidth between two nodes

```
iperf3 -s          # Server
iperf3 -c 10.0.0.5 # Client
```

8.9 Firewalls and Packet Filtering

iptables – Packet filtering

framework Allow incoming SSH

```
sudo iptables -A INPUT -p tcp --dport 22 -j ACCEPT
```

Block an IP

```
sudo iptables -A INPUT -s 192.168.1.100 -j DROP
```

Save rules

```
sudo iptables-save > /etc/iptables/rules.v4
```

nftables – Newer firewall system (used in Debian 10+, RHEL 8+)

```
nf t list ruleset
```

```
nf t add rule ip filter input tcp dport 22 accept
```

8.10 Network Services & Hostname

Tools Check current hostname

```
hostname
```

```
hostnamectl
```

Change hostname

```
sudo hostnamectl set-hostname devops-server
```

Restart networking (if needed)

```
sudo systemctl restart networking
```

8.11 Inspecting and Managing Network

Daemons List listening services with systemd

```
systemctl list-sockets
```

Check active services

```
systemctl status nginx
```

8.12 Troubleshooting Scenarios

Scenario	Command
Can't reach a host	ping, traceroute
Port not open	ss -tuln, netstat, lsof -i
DNS failing	dig, nslookup, /etc/resolv.conf
Interface not up	ip link, systemctl restart NetworkManager

Scenario	Command
Firewall blocking	iptables -L, nft list ruleset

8.13 Summary

Networking on Linux is extremely powerful and scriptable. With tools like ip, ss, tcpdump, iptables, dig, and curl, you can diagnose, automate, and secure any network-related operations — from debugging latency to firewalling services.

Module 9: Shell Scripting and Automation

9.1 What Is Shell Scripting?

A shell script is a plain text file containing commands to be executed by the shell interpreter (e.g., bash, sh, zsh). It automates repetitive tasks like system updates, backups, file parsing, CI/CD workflows, and server health checks.

9.2 Shell Types

Shell	Path	Description
Bourne shell	/bin/sh	Original Unix shell
Bash	/bin/bash	Most widely used shell
Zsh	/bin/zsh	Advanced user shell
Fish	/usr/bin/fish	Friendly interactive shell

Most scripting is done in **bash**, which we'll use throughout this module.

9.3 Writing Your First Shell

Script hello.sh

```
#!/bin/bash
```

```
echo "Hello, world!"
```

Make it executable:

```
chmod +x hello.sh
```

```
./hello.sh
```

9.4 Variables and Data Types

```
name="DevOps Shack"
```

```
echo "Welcome to $name"
```

Environment variables:

```
echo $HOME
```

```
export MYVAR="value"
```

Read input:

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

9.5 Conditionals

```
if [ $age -ge 18 ]; then
  echo "You are eligible"
else
  echo "Not eligible"
fi
```

Numeric operators:

- -eq, -ne, -gt, -lt, -ge, -le

String comparison:

```
if [ "$user" = "admin" ]; then
  echo "Welcome Admin"
fi
```

9.6 File Test Operators

```
if [ -f myfile.txt ]; then
  echo "File exists"
fi
```

Test	Description
-f	Regular file
-d	Directory
-e	Exists
-r, -w, -x	Readable, writable, executable

9.7 Loops

For loop

```
for i in {1..5}; do
  echo "Count $i"
done
```

While loop

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

Until loop

```
until [ "$input" = "yes" ]; do
  read -p "Type yes to continue: " input
done
```

9.8 Case Statement

```
read -p "Enter choice: " choice
case $choice in
  start) echo "Starting service...";;
  stop) echo "Stopping service...";;
  *) echo "Invalid";;
esac
```

9.9 Functions in Shell

```
greet() {
  echo "Hello, $1"
}

greet "DevOps"
Return values:
bash
CopyEdit
sum() {
  return $(($1 + $2))
}
sum 3 5
echo $?
```

9.10 Script Arguments

```
echo "Script name: $0"

echo "First arg: $1"

echo "All args: $@"
```

Loop through all:

```
for arg in "$@"; do

  echo "Arg: $arg"
```

done

9.11 Exit Codes and Error Handling

```
cp file1.txt file2.txt

if [ $? -ne 0 ]; then

  echo "Copy failed"

   exit 1
```

fi

Use set for strict error checking:

```
set -euo pipefail
```

9.12 Logging and Redirection

```
echo "Info message" >> logfile.txt
```

```
echo "Error occurred" 2>> error.log
```

Suppress output:

```
command > /dev/null 2>&1
```

9.13 Cron Jobs – Task Scheduling

Edit crontab:

```
crontab -e
```

Format:

```
* * * * * /path/to/script.sh
```

Field order:

```
min hour day month weekday command
```

Example:

```
0 2 * * * /backup/backup.sh
```

List cron jobs:

```
crontab -l
```

9.14 Real-World Examples

Website Health Check

```
#!/bin/bash
URL="https://example.com"
STATUS=$(curl -s -o /dev/null -w "%{http_code}" $URL)
if [ $STATUS -ne 200 ]; then
  echo "Website down! Code: $STATUS" | mail -s "Site Down" admin@example.com
fi
```

Disk Usage Alert

```
#!/bin/bash
USAGE=$(df / | grep / | awk '{ print $5 }' | sed 's/%//g')
if [ $USAGE -gt 80 ]; then
  echo "Disk usage above 80%" | mail -s "Disk Alert" admin@example.com
fi
```

S3 Backup

```
#!/bin/bash
tar -czf /tmp/backup.tar.gz /var/www
aws s3 cp /tmp/backup.tar.gz s3://mybucket/backups/
```

9.15 Best Practices

- Always start with #!/bin/bash
- Use set -euo pipefail for safety
- Validate inputs (if [-z "$1"])
- Use logging and timestamps
- Use trap to handle signals (trap 'cleanup' EXIT)
- Keep scripts readable, modular, and version controlled

9.16 Summary

Shell scripting is the foundation of DevOps and automation. It empowers you to interact with the OS, write CI/CD logic, monitor systems, manage backups, and much more — all using native Linux tools.

Module 10: Systemd and Service Management – Ultra Deep Dive

10.1 What is systemd?

systemd is a system and service manager for Linux, responsible for:

- Bootstrapping the user space and bringing the system to operational state
- Managing system services (nginx, docker, ssh, etc.)
- Tracking system states and dependencies
- Offering powerful tools like logging, socket activation, timers, and targets

It replaces traditional init systems with a **parallel, dependency-aware** boot-up sequence.

10.2 systemd Components Overview

Component	Purpose
unit	Basic object that systemd handles (e.g., service, mount, device)
service	Unit for background daemons
target	Group of units to reach a system state
timer	Replaces cron
socket	Listens for events before starting services
journal	Logging system
cgroups	Resource control and monitoring

10.3 Common systemctl

Commands Start, stop, enable,

disable services

sudo systemctl start nginx

sudo systemctl stop nginx

sudo systemctl restart nginx

sudo systemctl reload nginx

sudo systemctl enable nginx

sudo systemctl disable nginx

Check status

systemctl status nginx

List all active services

systemctl list-units --type=service

Show all services (even inactive)

systemctl list-unit-files --type=service

10.4 Analyzing the Boot

Process Check boot duration

systemd-analyze

Breakdown by service

systemd-analyze blame

10.5 systemd Unit Types

Type	Description
.service	System daemon or app
.socket	Socket-activated services
.target	Boot goal or group
.mount	Mount points
.timer	Scheduled jobs
.path	File path triggers
.device	Devices like /dev/sda1

10.6 Anatomy of a Service Unit File

Example: /etc/systemd/system/custom.service

```
[Unit]
Description=My Custom Service
After=network.target

[Service]
Type=simple
ExecStart=/usr/bin/python3 /opt/myscript.py
Restart=on-failure
User=ubuntu

[Install]
WantedBy=multi-user.target
```

Key Sections:

- [Unit]: Meta and dependencies
- [Service]: Execution details
- [Install]: Enable

rules Enable and start it:

```
sudo systemctl daemon-reexec
sudo systemctl enable --now custom.service
```

10.7 Reloading After Editing Unit Files

Whenever you change a .service or .target file:

```
sudo systemctl daemon-reload
```

To

reenable and start:

```
sudo systemctl restart myservice
```

10.8 systemd Targets

Targets are like runlevels.

Target	Purpose
default.target	The default boot target
graphical.target	GUI mode
multi-user.target	Command-line multi-user mode
rescue.target	Single-user rescue mode
emergency.target	Minimal emergency shell

Set a different default:

```
sudo systemctl set-default multi-user.target
```

10.9 Logging with journalctl

View service logs:

```
journalctl -u nginx
```

View system boot logs:

```
journalctl -b
```

Follow logs in real time:

```
journalctl -f
```

Show logs by date:

```
journalctl --since "2 hours ago"
```

10.10 Timers: systemd Replacement for cron

Example timer to run a script every day:

backup.service

```
[Unit]
Description=Run backup script


[Service]
```

```
Type=oneshot

ExecStart=/opt/backup.sh
```

backup.timer

```
[Unit]
Description=Daily backup

[Timer]
OnCalendar=daily
Persistent=true

[Install]
WantedBy=timers.target
```

Enable and start:

```
sudo systemctl daemon-reload

sudo systemctl enable --now backup.timer
```

List timers:

```
systemctl list-timers
```

10.11 Debugging systemd Issues

Problem	Check
Service not starting	systemctl status, journalctl -xe
Unit file not found	Check location and filename
ExecStart path wrong	Ensure full path and permissions
Permission issues	Use correct User= in service

10.12 Best Practices

- Avoid editing unit files in /lib/systemd/ — use /etc/systemd/ instead
- Use Restart=on-failure for long-running services
- Use ExecStartPre and ExecStartPost for setup/cleanup
- Set User= to avoid running as root unless required
- Monitor and manage timers instead of relying on cron

10.13 Summary

systemd is much more than a service manager — it's a full init and event-based framework for Linux. Mastering systemctl, .service files, journalctl, targets, and timers will allow you to manage, troubleshoot, and automate Linux systems reliably and professionally.

11.1 Introduction to Linux Storage Architecture

Linux treats all storage devices as files (e.g., /dev/sda, /dev/nvme0n1) and uses layers to abstract and manage storage:

Block Devices → Partitions → Filesystems → Mount Points

↳ LVM/RAID

11.2 Identify Attached Disks

lsblk – Block device listing

```
lsblk -f
```

fdisk -l – View partitions on all disks

```
sudo fdisk -l
```

blkid – Get UUIDs and labels

```
sudo blkid
```

df -h – Filesystem usage

```
df -h
```

du -sh – Directory size

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

11.3 Partitioning Disks with

fdisk Create partitions

```
sudo fdisk /dev/sdb
```

Within fdisk:

- n: new partition
- p: primary
- w: write changes
- d: delete
- t: change type

- q: quit without saving

Format new partition

```
sudo mkfs.ext4 /dev/sdb1
```

11.4 Mounting Filesystems

Manual mount

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

Unmount

```
sudo umount /mnt
```

Persistent mount via /etc/fstab

```
UUID=1234-5678  /mnt/data  ext4  defaults  0 2
```

View current mounts

```
mount | grep /mnt
```

11.5 Filesystem Types

Filesystem	Description
ext4	Most common for Linux, journaling
xfs	High performance, RHEL default
btrfs	Modern, snapshots, compression
ntfs	Windows filesystem (read-write via ntfs-3g)
vfat	USB/SD card compatibility

Create filesystem:

```
mkfs.ext4 /dev/sdc1
```

```
mkfs.xfs /dev/sdd1
```

11.6 Logical Volume Manager (LVM)

Why LVM?

- Combine multiple disks
- Resize volumes dynamically
- Snapshots and cloning
- Thin provisioning

LVM Architecture

Physical Volume (PV) → Volume Group (VG) → Logical Volume (LV)

11.7 Step-by-Step LVM Setup

1. Create Physical Volume

```
sudo pvcreate /dev/sdb1 /dev/sdc1
```

2. Create Volume Group

```
sudo vgcreate datavg /dev/sdb1 /dev/sdc1
```

3. Create Logical Volume

```
sudo lvcreate -L 10G -n datalv datavg
```

4. Format and mount LV

```
sudo mkfs.ext4 /dev/datavg/datalv
sudo mkdir /data
sudo mount /dev/datavg/datalv /data
```

11.8 Resizing Volumes

Increase size of Logical

Volume

```
sudo lvextend -L +5G /dev/datavg/datalv
sudo resize2fs /dev/datavg/datalv
```

Shrink (careful!)

```
sudo umount /data
sudo e2fsck -f /dev/datavg/datalv
sudo resize2fs /dev/datavg/datalv 5G
```

sudo lvreduce -L 5G /dev/datavg/datalv

sudo mount /data

11.9 LVM Snapshots

Create snapshot

sudo lvcreate -s -L 1G -n datasnap /dev/datavg/datalv

Mount snapshot for backup

sudo mount /dev/datavg/datasnap /mnt/snapshot

Delete snapshot

sudo lvremove /dev/datavg/datasnap

11.10 LVM Thin Provisioning

Useful for allocating space on demand:

lvcreate --type thin-pool -L 20G -n thinpool datavg

lvcreate -V 10G -T datavg/thinpool -n thinvol

mkfs.ext4 /dev/datavg/thinvol

11.11 Monitoring and Managing LVM

Command	Purpose
pvs	List physical volumes
vgs	Volume group summary
lvs	Logical volume summary
lvdisplay	Detailed LV info
vgextend, vgreduce	Modify volume groups
lvremove, vgremove	Delete LVs and VGs

11.12 RAID Overview (Optional Add-on)

- RAID 0: Striping (fast, no redundancy)
- RAID 1: Mirroring
- RAID 5: Striping + Parity
- RAID 10: Mirrored +

Striped Tool: mdadm

Create RAID 1:

mdadm --create --verbose /dev/md0 --level=1 --raid-devices=2 /dev/sdb1 /dev/sdc1

11.13 Filesystem Check & Repair

sudo fsck /dev/sdb1

sudo fsck -y /dev/datavg/datalv

Schedule regular checks via /etc/fstab:

UUID=... /data ext4 defaults 0 2

11.14 Automounting with systemd

Create mount unit /etc/systemd/system/data.mount

[Unit]

Description=Mount LVM Volume

[Mount] What=/dev/datavg/datalv
Where=/data
Type=ext4

[Install]
WantedBy=multi-user.target

Enable it:

systemctl daemon-reload

systemctl enable data.mount

11.15 Summary

Linux disk and volume management is highly flexible through fdisk, mkfs, mount options, and the Logical Volume Manager (LVM). By understanding how to partition disks, create filesystems, manage LVM snapshots, and automate mounts, you're equipped to handle real-world infrastructure and production storage systems.

Module 12: Linux Security

12.1 Layers of Linux Security

Linux security can be enforced through multiple mechanisms:

- **User and group ownership**
- **File permissions and ACLs**
- **Pluggable Authentication Modules (PAM)**
- **Mandatory Access Control (SELinux, AppArmor)**
- **Network firewalls (iptables/nftables)**
- **Service-level isolation (systemd, containers)**

12.2 File and Directory Security

File Permissions Recap

```
chmod 644 file.txt    # Owner rw-, group r--, others r--
chmod 700 ~/.ssh      # Owner full, no access to others
```

Set Permissions with Symbolic Mode

```
chmod u+rwx,g-r,o-r file.txt
```

Ownership

```
chown user:group file.txt
```

View recursive permissions

```
ls -lR /secure_dir
```

12.3 Access Control Lists (ACLs)

ACLs offer per-user or per-group file permissions beyond the traditional model.

Set ACL

```
setfacl -m u:john:rw file.txt
```

View ACL

```
getfacl file.txt
```

12.4 sudo and Privilege Management

Limit who can run what as root.

Edit safely

visudo

Allow specific command

john ALL=(ALL) /usr/sbin/service nginx restart

Add user to sudo group

usermod -aG sudo john

12.5 PAM – Pluggable Authentication Modules

Used for:

- Login policies
- Password complexity
- Account lockout

Example: Lock account after failed login attempts

vim /etc/pam.d/common-auth

Add:

auth required pam_tally2.so deny=5 unlock_time=300

12.6 Password Policy & Account

Expiry Password aging

chage -M 90 -W 10 john

Expire user account on a date

chage -E 2025-12-31 john

12.7 Detecting and Managing SUID/SGID Files

SUID/SGID can be dangerous if misused.

Find SUID binaries

find / -perm -4000 -type f 2>/dev/null

Find world-writable files

find / -type f -perm -o+w 2>/dev/null

12.8 Logging and Audit

Monitor logins and authentication

last

lastb

journalctl -u ssh

Use auditd for system-wide auditing

auditctl -w /etc/passwd -p wa -k passwd_watch

ausearch -k passwd_watch

12.9 SELinux (Security Enhanced Linux)

Mandatory Access Control for RHEL-based systems.

Status

getenforce

sestatus

Modes:

- Enforcing (default): Denies unauthorized access
- Permissive: Logs violations only
- Disabled: No policy applied

Set to permissive temporarily

setenforce 0

Change context of a file

chcon -t httpd_sys_content_t index.html

Restore default SELinux context

restorecon -Rv /var/www/html

12.10 AppArmor (Ubuntu-based MAC)

Check AppArmor status

aa-status

Manage profiles

aa-complain /etc/apparmor.d/usr.sbin.mysqld

aa-enforce /etc/apparmor.d/usr.sbin.mysqld

12.11 Firewalls with iptables / firewalld /

nftables Basic iptables examples

Allow SSH

iptables -A INPUT -p tcp --dport 22 -j ACCEPT

Drop ICMP ping

iptables -A INPUT -p icmp -j DROP

List rules

iptables -L -v -n

Persist rules

iptables-save > /etc/iptables/rules.v4

Using firewalld (RHEL/Fedora)

firewall-cmd --get-zones

firewall-cmd --add-service=http --permanent

firewall-cmd --reload

---

nftables (modern replacement for iptables)

nf t list ruleset

nf t add rule inet filter input tcp dport 22 accept

12.12 Port Scanning and Intrusion Detection

Detect open ports

```
ss -tuln
```

```
nmap -sT localhost
```

Install fail2ban

```
sudo apt install fail2ban
```

Auto-ban repeated failed SSH attempts.

12.13 Kernel Hardening with

sysctl View settings

```
sysctl -a
```

Block IP spoofing

```
echo "net.ipv4.conf.all.rp_filter = 1" >> /etc/sysctl.conf
```

```
sysctl -p
```

12.14 Service Isolation and Hardening

Use systemd directives in .service files: [Service]

```
ProtectSystem=full
```

```
ProtectHome=yes
```

```
PrivateTmp=yes
```

```
NoNewPrivileges=yes
```

These restrict filesystem access, prevent privilege escalation, and isolate services.

12.15 Summary

Securing a Linux system is a layered process. You must combine:

- Proper user/group permission hygiene
- Privilege limitation via sudo and PAM
- Mandatory access controls with SELinux/AppArmor

- Network filtering via iptables/nftables
- Logging and monitoring for proactive defense

Mastering these tools will enable you to deploy hardened Linux environments that can withstand internal mistakes and external threats.

Module 13: Logging and Troubleshooting in Linux – Ultra Deep Dive

13.1 Importance of Logging and Troubleshooting

Linux logs every event: service starts, kernel warnings, user logins, SSH access, disk errors, and more. Effective troubleshooting starts with knowing:

- Where logs are stored
- How to read them efficiently
- What logs to check for which issue

13.2 Log File Locations – Key

Directories Primary log directory

/var/log/

Common log files

File	Purpose
/var/log/syslog	General system logs (Debian/Ubuntu)
/var/log/messages	General system logs (RHEL/CentOS)
/var/log/auth.log	Authentication, sudo, ssh
/var/log/secure	Similar to auth.log (RHEL)
/var/log/dmesg	Boot/kernel ring buffer
/var/log/kern.log	Kernel messages
/var/log/boot.log	Boot sequence output
/var/log/faillog	Failed login attempts
/var/log/cron	Cron job logs
/var/log/httpd/, /var/log/nginx/	Web server logs

13.3 journalctl – Query the systemd journal

Show all logs

```
journalctl
```

Show logs for a specific service

```
journalctl -u nginx
```

View boot messages

```
journalctl -b
```

View logs since a time range

```
journalctl --since "2 hours ago"
```

Follow logs live (like tail -f)

```
journalctl -f
```

13.4 dmesg – Kernel Ring Buffer

Used to view hardware-related messages, especially useful after boot or device failures.

```
dmesg | less
```

```
dmesg | grep -i error
```

13.5 Analyzing Failed

Services Check service status

```
systemctl status apache2
```

Check logs

```
journalctl -xe
```

Restart and retry

```
systemctl restart apache2
```

13.6 Disk and Filesystem

Troubleshooting Check disk space

```
df -h
```

Check directory usage

```
du -sh /var/*
```

Check inode usage

```
df -i
```

Run filesystem check

```
sudo fsck /dev/sda1
```

13.7 Network Troubleshooting

Connectivity

```
ping google.com
```

```
traceroute google.com
```

DNS resolution

```
dig google.com
```

Port availability

```
ss -tuln
```

Interface config

```
ip a
```

13.8 Process and Memory

Troubleshooting List processes

```
ps aux | grep java
```

View top resource users

```
top
```

```
htop
```

Memory usage

```
free -m
```

Monitor in real time

```
vmstat 1
```

```
iostat -xz 1
```

13.9 User Login

Troubleshooting View last

logins

last

Failed login attempts

lastb

Current sessions

who

w

13.10 Log Rotation with logrotate

Logrotate manages log file size, compression, and cleanup.

Configuration files

/etc/logrotate.conf

/etc/logrotate.d/*

Example config

```
/var/log/nginx/*.log {
    daily
    missingok
    rotate 14
    compress
    delaycompress
    notifempty
    create 0640 www-data adm
}
```

Force rotation

logrotate -f /etc/logrotate.conf

13.11 Creating Custom Log Files in Shell Scripts

Logging messages

```
#!/bin/bash

LOGFILE="/var/log/myapp.log"

echo "$(date): Starting backup" >> $LOGFILE
```

Use timestamps and exit codes for meaningful output.

13.12 Common Troubleshooting Scenarios

Issue	Logs/Commands
Service not starting	systemctl status, journalctl -xe
SSH login failure	/var/log/auth.log, lastb
Web server down	/var/log/nginx/error.log, ss -tuln
Disk full	df -h, du -sh /*
Package error	/var/log/dpkg.log, apt install -f
Boot failure	journalctl -b, dmesg

13.13 Advanced Logging Techniques

- Use rsyslog to forward logs to a central server
- Use logwatch for log summary reports
- Integrate logs with SIEM tools (Splunk, ELK, Graylog)

13.14 Best Practices

- Rotate and compress logs periodically
- Monitor disk space in /var/log
- Use centralized logging in multi-server environments
- Monitor failed logins, service restarts, and cron logs
- Automate log review scripts for anomalies

13.15 Summary

Logs are your first and most valuable resource when diagnosing system behavior. Mastery of tools like journalctl, logrotate, dmesg, and reading system logs ensures you're able to troubleshoot complex issues in real-time and in postmortem analysis.

14.1 Why Performance Tuning Matters

System administrators and DevOps engineers must monitor and optimize:

- CPU load and utilization
- Memory usage and swap activity
- Disk I/O performance
- Network throughput and congestion
- Process-level behavior

Performance issues directly affect application availability, scalability, and SLAs.

14.2 Load Average vs CPU Usage

Load Average (from uptime, top, w)

uptime

Example:

12:00:00 up 5 days, 1:23, 2 users, load average: 0.24, 0.18, 0.15

- Represents **number of active processes** in 1, 5, and 15 minutes
- Compare against **core count** for meaning (e.g., 4-core system: load 4 = 100%)

14.3 CPU Monitoring

Tools top

top

Real-time CPU/memory process stats. Press 1 to see per-core usage.

htop

htop

Advanced interactive tool. Press F6 to sort.

mpstat (from sysstat)

mpstat -P ALL 1

pidstat – per-process CPU breakdown

pidstat 1

14.4 Memory and Swap

Usage free

free -h

vmstat

vmstat 1 5

- si/so: swap in/out activity

slabtop

slabtop

Kernel slab allocator details.

smem – show actual PSS memory usage per process

smem -r -k -t

14.5 Disk I/O

Monitoring iostat

(requires sysstat)

iostat -xz 1 5

- await: time for I/O (target < 20 ms)
- util: % device busy

iotop – real-time I/O activity per process

sudo iotop

dstat – live CPU, disk, net, memory stats

dstat -cdnm 1

14.6 Filesystem Performance

Check mount options

mount | grep ext4

Use noatime to reduce unnecessary disk writes.

Filesystem tuning with tune2fs

tune2fs -l /dev/sda1

Reduce reserved blocks:

tune2fs -m 1 /dev/sda1

14.7 Network Performance

Monitoring ss – socket summary

ss -s

ss -tulnp

iftop – live bandwidth per interface

sudo iftop -i eth0

nethogs – bandwidth per process

sudo nethogs eth0

iperf3 – test network throughput

iperf3 -s # Server

iperf3 -c <host> # Client

14.8 Process-Level Performance Tuning

nice – set process priority

nice -n 10 command

renice – adjust running process

renice -n -5 -p <PID>

Lower value = higher priority

taskset – bind process to specific CPU cores

taskset -c 0,1 myscript.sh

cpulimit – limit CPU usage

cpulimit -l 50 -p <PID>

14.9 Kernel Parameter Tuning (sysctl)

View all parameters

```
sysctl -a
```

Example tunings

```
sysctl -w vm.swappiness=10
```

```
sysctl -w fs.file-max=1000000
```

Persist in /etc/sysctl.conf

```
vm.swappiness=10
```

```
fs.file-max=1000000
```

Apply changes:

```
sysctl -p
```

14.10 Track Resource Limits

(ulimit) View current limits

```
ulimit -a
```

Set soft and hard limits

```
ulimit -n 65535      # open files
```

```
ulimit -u 4096       # user processes
```

Permanent: /etc/security/limits.conf

14.11 Detecting Bottlenecks

Symptom	Suspect Area	Tools
High load average	CPU/IO Wait	top, uptime, iostat
Slow app startup	Disk I/O	iotop, iostat
OOM errors	Memory	dmesg, free, smem
Dropped packets	Network	netstat, ifconfig, dstat
App unresponsive	Process scheduling	ps, top, nice, strace

14.12 Advanced Tools

perf – low-level performance analysis

```
perf top
```

```
perf record ./app
```

strace – system call tracing

```
strace -p <PID>
```

lsof – open files per process

```
lsof -p <PID>
```

14.13 Automation & Alerting

- Use cron + scripts to log system stats periodically
- Use atop, nmon, or collectl for recording historical stats
- Integrate Telegraf + InfluxDB + Grafana for dashboards

14.14 Summary

Performance tuning is a proactive skill that combines system knowledge with tool proficiency. By mastering tools like top, iostat, vmstat, sysctl, pidstat, and iotop, you'll be able to detect, isolate, and resolve real-world bottlenecks across compute, memory, disk, and network.

Module 15: Linux in the Cloud, Containers, and Kubernetes

15.1 Why Linux Dominates in Cloud and DevOps

- Over **96% of cloud workloads** run on Linux
- All major cloud providers (AWS, GCP, Azure) offer Linux-based VMs by default
- Tools like **Docker, Kubernetes, Terraform, Jenkins** are built for Linux
- Linux is **lightweight, modular, scriptable, and scalable** — ideal for microservices, containers, and serverless

---

15.2 Linux on Cloud

Platforms AWS Linux

- **Amazon Linux 2**: CentOS-like, tuned for EC2
- AMIs available for Ubuntu, Debian, RHEL, Fedora, SUSE

Cloud VM lifecycle:

```
ssh -i key.pem ec2-user@<public-ip>
sudo yum update -y
sudo systemctl enable nginx
```

Cloud-init Scripts

```
#cloud-config
packages:
 - nginx
runcmd:
 - systemctl enable nginx
 - systemctl start nginx
```

Upload via user data when launching instances.

15.3 Linux in CI/CD and Automation

- Jenkins agents typically run on Linux
- GitHub Actions, GitLab runners, CircleCI, and others default to Linux environments

- Shell scripts, deployment hooks, and pipelines are written in Bash

Example deployment step:

```
scp app.tar.gz ubuntu@host:/var/www/
```

```
ssh ubuntu@host 'tar -xzf /var/www/app.tar.gz && systemctl restart nginx'
```

15.4 Linux + Containers (Docker)

Linux provides native support for **namespaces, cgroups, and union filesystems**, making it the backbone of container engines like Docker and Podman.

Create a container

```
docker run -it --name test-container ubuntu bash
```

Inspect container internals

```
ps aux
```

```
cat /etc/os-release
```

Check resource usage

```
docker stats
```

Linux Kernel Isolation:

- **Namespaces**: process, network, mount isolation
- **Cgroups**: resource limits
- **OverlayFS**: container layered file systems

15.5 Building Docker Images from Linux

Tools Dockerfile example

```
FROM ubuntu:20.04
```

```
RUN apt update && apt install -y nginx
```

```
COPY index.html /var/www/html/
```

```
CMD ["nginx", "-g", "daemon off;"]
```

Build and run:

```
docker build -t mynginx .
```

```
docker run -d -p 80:80 mynginx
```

15.6 Podman – Docker Alternative Without Daemon

```
sudo dnf install podman
```

```
podman run -it ubuntu bash
```

Compatible with Docker CLI and image formats.

15.7 Linux in Kubernetes

Kubernetes clusters use **Linux nodes** for:

- Running containerized pods
- Managing persistent storage (PV/PVC)
- Executing kubelet, kube-proxy, and other services

Node OS requirements:

- Must support **cgroups v1 or v2**
- Required tools: iptables, ip, ethtool, mount, nsenter

15.8 Inspecting Linux Inside a Kubernetes Pod

```
kubectl exec -it mypod -- bash
```

Once inside:

```
cat /proc/cpuinfo
```

```
ps aux
```

```
df -h
```

15.9 Using Linux Tools for Kubernetes Debugging

Tool	Usage
nsenter	Enter namespaces of containers
journalctl	View kubelet logs
crictl	Interact with container runtimes (containerd)
iptables, ip	Inspect network setup
du, df, mount	Check storage issues

15.10 Persistent Volumes in Linux

Example using hostPath:

```
volumeMounts:
  - mountPath: /data
    name: datavol

volumes:
  - name: datavol
    hostPath:
      path: /mnt/data
```

Backed by real Linux directories on the node.

15.11 Linux and Container Storage Interfaces (CSI)

- Linux tools (mkfs, mount, blkid) are used by CSI drivers
- Block storage volumes (EBS, Persistent Disk) attach as /dev/xvdf, etc.
- Used with dynamic provisioning in Kubernetes

15.12 Securing Containers with Linux Features

- Use chroot, seccomp, capsh, and AppArmor profiles
- Drop unnecessary capabilities:

```
securityContext:
  capabilities:
```

```
drop:
  - ALL
```

15.13 Monitoring Linux Nodes in the Cloud

Use tools like:

- **Prometheus Node Exporter**
- top, vmstat, dstat for node-level checks
- journalctl -u kubelet for K8s node errors

15.14 Summary

Linux is the underlying platform that powers the modern cloud and containerized world. Whether you're managing cloud VMs, writing CI/CD pipelines, building Docker images, or running Kubernetes nodes — Linux skills remain indispensable.

You've now completed an ultra-deep, module-by-module guide to Linux from basics to cloud-native integration.