# Linux: System Administration

TEKsystems Global Services

# WORKFORCE DEVELOPMENT

# LOGISTICS

## Class Hours:

- Instructor will set class start and end times.
- There will be regular breaks in class.

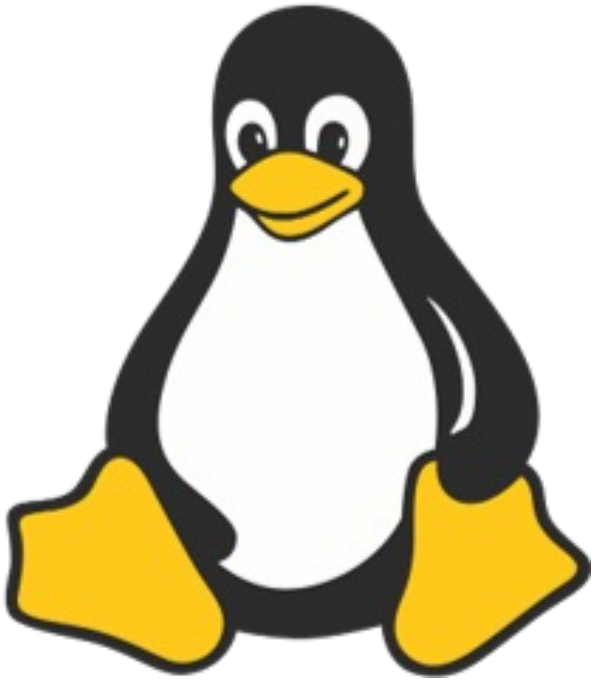
## Telecommunication:

- Turn off or set electronic devices to silent (not vibrate)

## Learning:

- Run the commands with the instructor as the slides are presented to you
- Ask questions and participate

# Objectives



1. Install and manage installations with package manager and manually
2. Understand Network configurations and monitoring
3. Maintain Services running on your system
4. Manage Disk and file systems

# Linux: Package Management

Package management allows you to **install, update, and remove software** safely and efficiently.

It keeps systems **consistent, secure, and up to date**, ensuring dependencies are handled automatically.

Most distributions use tools like **dnf**, **yum**, or **apt** — mastering them helps you manage everything from servers to desktops with confidence.

Knowing how to query, verify, and roll back packages lets you **control change** and **maintain stability** across environments.

💡 A good admin doesn't just use software — they understand how it's installed, updated, and maintained.

# Linux: Package Management

## 📦 Understanding Package Managers in Linux

Linux uses **package managers** to install, update, and remove software while handling dependencies automatically.

They come in **two layers** — high-level tools and low-level tools.

# Linux: Package Manager

## 🧩 High-Level Package Managers

Front-end tools that handle dependencies, repositories, and updates automatically:

| Distribution | Tool | Description |
|---|---|---|
| RHEL / CentOS / Fedora | `dnf`, `yum` | High-level front-ends for RPM packages |
| Debian / Ubuntu | `apt`, `apt-get` | High-level front-ends for DEB packages |
| Arch Linux | pacman | Unified tool that manages `.pkg.tar.zst` packages and dependencies |

💡 **Best practice:** use the *high-level manager* (dnf, apt, or pacman) — it ensures dependency resolution and system consistency automatically.

# Linux: Package Manager

## ⚙️ Low-Level Package Managers

Work directly with package files (no dependency resolution):

| Package Type | Tool | Example Command |
|---|---|---|
| .rpm | `rpm` | sudo rpm -ivh package.rpm |
| .deb | `dpkg` | sudo dpkg -i package.deb |

💡 **Best practice:** use the *high-level manager* (dnf, apt, or pacman) —
it ensures dependency resolution and system consistency automatically.

# Linux: Package Manager

## 📦 Working with DNF – Basic Commands

dnf is the **package manager** used on RHEL, CentOS, and Fedora systems.It handles installation, updates, and dependency resolution automatically.

```
# Search for a package
dnf search httpd

# View detailed info about a package
dnf info httpd

# Install a package
sudo dnf install httpd –y # -y installs without prompting
```

💡 dnf automatically pulls in required dependencies from configured repositories.

# Linux: Package Manager

## 📋 Listing Installed Packages with DNF

You can view all packages installed on your system using dnf list installed.
Combine it with commands like grep or head to filter and manage large outputs.

```
# List all installed packages
dnf list installed

# Show only the first 10 entries
dnf list installed | head

# Search for a specific package
dnf list installed nginx
```

# Linux: Package Manager

## 🧩 Managing Package Versions with DNF

Sometimes you need a **specific version** of a package — for compatibility, stability, or rollback purposes.
dnf makes it easy to view and install older or alternate versions from your repositories.

```
# Search available versions of a package
dnf list httpd --showduplicates
```

You'll see output like:

```
httpd.x86_64  2.4.57-1.el9_2   appstream
httpd.x86_64  2.4.58-1.el9_3   appstream
```

To install a specific version:

```
sudo dnf install httpd-2.4.57-1.el9_2
```

💡 **Notes:**

- Always match the full version–release string (e.g., 2.4.57-1.el9_2).
- Downgrading or pinning versions can prevent breakage when other packages depend on older libraries.
- Use dnf versionlock (from dnf-plugins-core) if you need to **lock** a version.

# Linux: Package Manager

## 🔄 Downgrading or Removing Packages

Sometimes a newer version introduces bugs or breaks compatibility.You can **downgrade** or **uninstall** packages with DNF safely and cleanly.

```
# Downgrade to a previous version
sudo dnf downgrade httpd

# Or specify an exact version
sudo dnf install httpd-2.4.57-1.el9_2

# Remove a package
sudo dnf remove httpd
```

💡Unlike low-level tools, DNF automatically handles **dependencies and conflicts**

# Linux: Package Manager

## 🧠 Understanding Updates vs Upgrades

Before managing software, it's important to know what each action means:

| Term | Scope | Description |
|---|---|---|
| Update | *Package-Level* | Installs the **latest version** of existing packages from current repositories (bug fixes, security patches, small feature updates). |
| Upgrade | *System-level* | Moves the system to a **new release version** (e.g., Fedora 38 → 39 or RHEL 8 → 9). Includes new repositories and dependencies. |
| Patch / Security Update | *Targeted* | Fixes specific vulnerabilities or stability issues — often applied selectively or automatically. |

# Linux: Package Manager

## 🔍 Checking for Package Updates & Upgrades

Regularly checking for package updates keeps your system **secure and current**, though updates can occasionally introduce **breaking changes**.

Always **stage and test** updates before deploying them to production.

```
# List available package updates and OS upgrades
dnf check-update

# Show information about security updates only
dnf updateinfo list security
```

💡 **Tip:** Reviewing updates first helps you plan maintenance windows and avoid unplanned restarts.
If the OS release metadata changes, **DNF will also alert you** that a new system release (upgrade) is available.

# Linux: Package Manager

## ⬆️ Updating Packages with DNF

You can update all packages or just specific ones when newer versions are available.

```
# Update a specific package
sudo dnf update -y httpd

# Verify the version after update
dnf info httpd
```

💡 Tip:

- -y auto-confirms the update prompt (use with caution).
- Not every update is safe — test in a staging environment to catch regressions early.

# Linux: Package Manager

## ⚠️ dnf update vs dnf upgrade

Both commands bring your system **up to date**, but there are important differences — and neither should ever be run *blindly* on a production system.

```
# Update all installed packages to the latest versions
sudo dnf update

# Upgrade all packages and handle obsolete ones
sudo dnf upgrade
```

**Key Points:**

- **update** → updates packages already installed.
- **upgrade** → updates packages *and* removes or replaces obsolete ones.
- Both can change critical libraries, services, or dependencies.

💡 **Reminder:**
**These are not hot-run commands** — always test in a **staging environment** first, review changelogs, and plan maintenance windows before applying full system updates.

# Linux: Package Manager

## OS Images and Snapshots

Running dnf update or dnf upgrade in staging doesn't guarantee the same results in production —
repositories change, and package versions can shift daily.

Instead of updating live systems, admins use **frozen OS images** or **snapshots**:

- **OS Images (AMIs, VM Templates):** Contain a *known good* system state with tested package versions.
- **Snapshots:** Capture a system's disk state before changes, allowing easy rollback.
- **Immutable Infrastructure:** Servers are rebuilt from images instead of being updated in place.

💡 *Use images to deploy identical, stable systems across environments.*

# Linux: Package Manager

## 🕒 Viewing DNF History

When managing systems, it's important to track what packages were installed, upgraded, or removed.
The **dnf history** command helps you review and audit these changes.

```
# View a list of past DNF transactions
dnf history
```

💡 *Each transaction has an ID — useful for viewing or undoing changes.*

To see exactly what happened in a specific transaction (like what was installed or updated):

```
# View detailed info for transaction ID 1
dnf history info 1
```

💡 *This helps verify who made the change and what packages were affected.*
*Never rerun blindly* — understand what changed before repeating or undoing a transaction.

# Linux: Package Manager

## 🧹 Cleaning DNF Cache

Over time, cached packages and metadata can take up space.
Use **dnf clean** to remove old or unnecessary data.

```
# Remove cached package files and metadata
sudo dnf clean all
```

💡 *This frees disk space and ensures fresh metadata on the next install or update.*
*Avoid using it before critical updates* — caching speeds up future operations.

# Linux: Package Manager

## 📦 Managing DNF Repositories

Repositories are online or local sources of packages.
Different repositories provide **different versions or types of software**, such as open-source, enterprise, or testing builds.

Displays all repositories currently enabled on the system.
dnf repolist

Registers a new repository for DNF to use when installing packages.
sudo dnf config-manager --add-repo <repo_url>

Deletes the configuration for a specific repository from /etc/yum.repos.d/.
sudo dnf config-manager --remove-repo <repo_name>

💡 We viewed, added, and removed repositories — learning that each repo provides its own set of available packages.

# Linux: Package Manager

## 🔄 Checking and Performing System Reboots

After updates, some services or kernel changes may require a reboot to take effect.
Always confirm it's safe to reboot — especially on production systems.

Check if any services need restarting
```
sudo dnf needs-restarting
```

Check if a full reboot is required
```
sudo dnf needs-restarting -r
```

Reboot the system immediately (only if safe)
```
sudo reboot
```

Schedule a reboot in 5 minutes
```
sudo shutdown -r +5 "System will reboot in 5 minutes for updates"
```

Cancel a scheduled reboot
```
sudo shutdown -c
```

💡 We verified whether a reboot is needed, then safely rebooted or scheduled one as required.

# Lab 4.1 Package Manager

Estimated Time: 30

# POP QUIZ:

What does the dnf update command do?

A. Installs a new package
B. Removes all outdated packages
C. Updates installed packages to latest version
D. Checks for broken depencies

# POP QUIZ:

What does the dnf update command do?

"Is this always safe?"

A. Installs a new package
B. Removes all outdated packages
C. Updates installed packages to latest version
D. Checks for broken depencies

# POP QUIZ:

What command can show me what is installed on my system?

A. dnf list installed
B. dnf installed list
C. dnf search installed
D. dnf install-list

# POP QUIZ:

What command can show me what is installed on my system?

A. dnf list installed
B. dnf installed list
C. dnf search installed
D. dnf install-list

"How can I check for a specific package?"

# Linux: Networking

Networking is the backbone of all modern computing. Every system, service, and application depends on reliable communication — whether it's reaching another server, a database, or the internet itself. Understanding how networks function enables administrators to configure, secure, and troubleshoot connectivity across systems.

For a Linux system administrator, networking knowledge is essential. It allows you to verify connections, diagnose issues, manage interfaces, and ensure services are accessible and resilient. From simple pings to advanced routing and firewall management, networking skills keep your infrastructure running smoothly and securely.

# Linux: Networking

## 🧭 Networking Fundamentals Overview

In this lesson, you'll explore the essential networking tools built into Linux and learn how to use them for real-world troubleshooting and configuration. Each command you practice will help you understand what's happening behind the scenes when systems communicate.
You'll learn how to:

- View and manage network interfaces and routing tables
- Test connectivity and analyze latency
- Resolve domains and check DNS configuration
- Inspect listening ports and active connections
- Trace network routes and analyze packet flow
- Capture and review network traffic for diagnostics
- Manage firewall rules and network security

# Linux: Networking

## 🧰 The ip Command (Overview)

ip is the modern, unified tool to **inspect and manage** Linux networking (replaces older ifconfig/route).

It's organized by **object → action** (e.g., ip addr show, ip route show, ip link show).

Today we're using it in **read-only mode** to understand the host's network state.

General syntax (for reference)

ip <object> <action> [options]

💡 We'll read addresses, links, and routes to identify the host and how it reaches other networks.

```
[ec2-user@ip-172-31-1-87 ~]$ ip
Usage: ip [ OPTIONS ] OBJECT { COMMAND | help }
       ip [ -force ] -batch filename
where  OBJECT := { address | addrlabel | fou | help | ila | ioam | l2tp | link |
                   macsec | maddress | monitor | mptcp | mroute | mrule |
                   neighbor | neighbour | netconf | netns | nexthop | ntable |
                   ntbl | route | rule | sr | stats | tap | tcpmetrics |
                   token | tunnel | tuntap | vrf | xfrm }
       OPTIONS := { -V[ersion] | -s[tatistics] | -d[etails] | -r[esolve] |
                    -h[uman-readable] | -iec | -j[son] | -p[retty] |
                    -f[amily] { inet | inet6 | mpls | bridge | link } |
                    -4 | -6 | -M | -B | -0 |
                    -l[oops] { maximum-addr-flush-attempts } | -echo | -br[ief] |
                    -o[neline] | -t[imestamp] | -ts[hort] | -b[atch] [filename] |
                    -rc[vbuf] [size] | -n[etns] name | -N[umeric] | -a[ll] |
                    -c[olor]}
```

# Linux: Networking

## 🧩 Viewing IP Addresses – ip addr show (Part 1)

Displays all network interfaces along with their **IPv4/IPv6 addresses**, flags, and link information.

```
ip addr show
# or the shorter form
ip a
```

**Key fields in the output:**

- **Interface name** – e.g. 2: eth0: identifies the device
- **State** – UP, DOWN, LOWER_UP indicate admin and physical link status
- **Layer 2** – link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff
- Shows the **MAC address** and **broadcast address**
- **inet / inet6** – show IPv4 and IPv6 addresses with prefix length (CIDR)
- inet 10.0.0.12/24 → IPv4
- inet6 fe80::d1:22ff:fe7a:1e43/64 → IPv6

💡 Use ip addr show to confirm which interfaces are active and what addresses are currently assigned.

# Linux: Networking

## Viewing IP Addresses – ip addr show

More fields in the output:

- **Flags** – shown as <BROADCAST,MULTICAST,UP,LOWER_UP>
  - **BROADCAST** – supports broadcast traffic
  - **MULTICAST** – used for service discovery or routing protocols
  - **UP** – interface enabled by admin
  - **LOWER_UP** – physical link detected

- **MTU (Maximum Transmission Unit)** – largest packet size (in bytes) that can be sent without fragmentation
  - Typical: 1500 for Ethernet
  - Too low → inefficient; too high → dropped packets

- **Scope** – defines how far the address can communicate
  - **global** – reachable from outside networks
  - **link** – only within the same network segment
  - **proto kernel** – assigned automatically by the kernel

- **valid_lft / preferred_lft** – for dynamic IPs (DHCP or IPv6 SLAAC)
  - valid_lft → how long the IP remains valid
  - preferred_lft → how long it's preferred for new connections
  - forever → static or manually configured

# Linux: Networking

```
[ec2-user@ip-172-31-1-87 ~]$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
 qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: ens5: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001 qdisc mq state UP group defau
lt qlen 1000
    link/ether 02:d1:22:7a:1e:43 brd ff:ff:ff:ff:ff:ff
    altname enp0s5
    altname eni-0467fc1c9861043c7
    altname device-number-0.0
    inet 172.31.1.87/24 metric 512 brd 172.31.1.255 scope global dynamic ens5
       valid_lft 2197sec preferred_lft 2197sec
    inet6 fe80::d1:22ff:fe7a:1e43/64 scope link proto kernel_ll
       valid_lft forever preferred_lft forever
```

# Linux: Networking

## 🔗 Viewing Network Links – ip link show

Displays **Layer 2 (Data Link Layer)** information — how data moves **between devices on the same network**, without passing through a router.

It focuses on **interfaces themselves**, not IP addresses.
ip link show

**Key fields in the output:**

- **Interface name & index** – e.g. 2: eth0:
- **Flags** – <BROADCAST,MULTICAST,UP,LOWER_UP> show operational status
- **MTU** – maximum frame size at Layer 2 (default 1500 bytes)
- **qdisc** – queueing discipline, controls how packets are buffered or scheduled
- **link/ether** – the **MAC address** used for communication on the local network
- **brd ff:ff:ff:ff:ff:ff** – broadcast address, used to reach all devices on the same link

💡 ip link show reveals the physical and data-link health of interfaces — it's the **foundation** of communication before IP routing even begins.

# Linux: Networking

```
[ec2-user@ip-172-31-1-87 ~]$ ip link show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT
group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: ens5: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001 qdisc mq state UP mode DEFAUL
T group default qlen 1000
    link/ether 02:d1:22:7a:1e:43 brd ff:ff:ff:ff:ff:ff
    altname enp0s5
    altname eni-0467fc1c9861043c7
    altname device-number-0.0
```

```
[ec2-user@ip-172-31-1-87 ~]$ ip r
default via 172.31.1.1 dev ens5 proto dhcp src 172.31.1.87 metric 512
172.31.0.2 via 172.31.1.1 dev ens5 proto dhcp src 172.31.1.87 metric 512
172.31.1.0/24 dev ens5 proto kernel scope link src 172.31.1.87 metric 512
172.31.1.1 dev ens5 proto dhcp scope link src 172.31.1.87 metric 512
```

# Linux: Networking

## 🧭 Viewing Routes – ip route show

Displays the **kernel routing table**, which tells Linux **where to send packets** once they leave the local network.
This is **Layer 3 (Network Layer)** — it decides the path packets take, unlike ip link which stays at Layer 2.

```
ip route show
# or the shorter form
ip r
```

**Key fields in the output:**

- **default via 10.0.0.1 dev eth0** → the **default gateway**; all non-local traffic goes here (usually router)
- **10.0.0.0/24 dev eth0 proto kernel scope link src 10.0.0.12**
- **Network & mask** – 10.0.0.0/24 is the subnet reachable directly
- **dev eth0** – which interface handles that subnet
- **scope link** – means reachable without a router (directly attached)
- **src 10.0.0.12** – preferred source IP for outbound packets on that network
- **metric 100 proto dhcp** – lower metric = higher priority; proto shows who created the route (e.g., DHCP, kernel)

💡 ip route show lets you trace how packets leave the system — check the **default route** first when external connectivity fails.

# Linux: Networking

## 🧭 Viewing the ARP Table – ip neigh show

The **Address Resolution Protocol (ARP)** maps **IP addresses (Layer 3)** to **MAC addresses (Layer 2)**.
It allows devices on the same network to find each other without using a router.

```
ip neigh show
# or the older command
arp -n
```

**Example Output:**
172.31.1.1 dev ens5 lladdr 02:42:ac:11:00:01 REACHABLE

**What it means:**

- **172.31.1.1** → the IP address you recently communicated with
- **dev ens5** → the interface used for communication
- **lladdr 02:42:ac:11:00:01** → the **MAC address** (Layer 2 address) of that device
- **REACHABLE** → connection state; other states include STALE, DELAY, or FAILED

💡 ARP bridges IP and MAC layers — ip neigh show confirms which devices your system knows on the local network and how they're physically reached.

# Linux: Networking

| Area | Command | Purpose |
|---|---|---|
| **Layer 3 – Addresses** | `ip addr show` | View IPs, MTU, flags, lifetime |
| **Layer 2 – Links** | `ip link show` | View interfaces, MACs, MTU, operational state |
| **Routing** | `ip route show` | View routing table and gateway paths |
| **Neighbor Discovery** | `ip neigh show` | View ARP cache (IP ↔ MAC mappings) |

# Linux: Networking

| Additional Commands | When to Use It | Notes |
|---|---|---|
| `ip link set dev eth0 down/up` | Bring interfaces down/up | Simple interface management |
| `ip addr add 192.168.1.10/24 dev eth0` | Temporarily assign IPs | Useful for testing network changes |
| `ip route add 10.0.0.0/24 via 192.168.1.1` | Add a static route | Teaches manual routing |
| `ip -s link` | View interface statistics | Shows dropped packets, errors, collisions |
| `ip monitor` | Watch for live interface/route changes | Advanced observation command |
| `ip addr flush dev eth0` | Clear IPs from an interface | Helpful during network reconfiguration |

💡 *These extra ip commands let you go **beyond observation into control** — bringing interfaces up or down, adding or removing routes, and monitoring changes in real time. They're powerful tools for testing network behavior, troubleshooting issues, and safely reconfiguring systems without a reboot.*

# Linux: Networking

## 📡 Testing Connectivity – ping

The **ping** command tests basic **network reachability** using the **ICMP (Internet Control Message Protocol).**
Ping is **not TCP or UDP**. It sends **echo requests** and waits for **echo replies** to confirm that a host is alive and reachable.

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

**What to look for:**

- **Replies received** → host is reachable
- **Time (ms)** → round-trip latency
- **Packet loss** → network congestion or firewall blocking ICMP

**Important:**

- ICMP works at **Layer 3 (Network Layer)** — it checks connectivity, not specific services.
- To test a **TCP connection Layer 4 (Transport)** (like a web or SSH port), use **netcat** instead:

```
# sudo dnf install -y nmap-ncat
nc -zv <host> <port>
```

💡 ping confirms that the network path is open and the host is responding — the first step in any connectivity test.

# Linux: Networking

## 🚨 Disclaimer: Legal Risks of Scanning Tools

**Instructor:** read this verbatim before any demo of nc, nmap, or port-scanning tools.

Class: **do not** run these tools against systems you **do not own** or **lack explicit written permission** to test.

- Unauthorized scanning can be a **criminal offense** or civil violation in many jurisdictions — it may lead to **arrest, fines, civil suits, or criminal charges**.
- Cloud providers and employers can suspend accounts, terminate access, or pursue legal action for misuse. Even well-intentioned scans can trigger incident response, generate forensic evidence, and create serious legal exposure.

💡 Keep written approval on file (who approved, scope, time window) before any hands-on scanning.

# Linux: Networking

## 🌐 DNS Lookups – nslookup

DNS (Domain Name System) converts **domain names into IP addresses**.
nslookup is a quick and simple way to verify name resolution.

nslookup google.com

**Output shows:**

- The **DNS server** used to resolve the name
- The **resolved IP address** (A or AAAA record)

Use it for quick checks when a domain isn't resolving or to confirm which DNS server your system is using.

# Linux: Networking

## 🔍 Detailed DNS Queries – dig

dig (Domain Information Groper) provides **detailed DNS results**, useful for troubleshooting or scripting.

dig google.com

**Sections in the output:**

- **ANSWER** → the actual DNS record returned
- **AUTHORITY** → which servers own the zone
- **ADDITIONAL** → related IPs (e.g., for name servers)
- **Query time / SERVER** → how long it took and which resolver answered

💡 Use dig when you need detailed DNS data or want to confirm authoritative answers. Use dig to confirm DNS updates are working and correctly configured.

# Linux: Networking

## 🧭 Targeted & Scripted DNS Checks – dig Options

Focus your DNS lookups with additional dig options.

**Query a specific DNS server:**
dig @8.8.8.8 amazon.com

**Show only the IP result (script-friendly):**
dig +short google.com

**Query specific record types:**
dig google.com MX  # Mail servers
dig google.com NS  # Name servers
dig google.com TXT  # Text records

💡 dig can test any record type, specific servers,
or even compare responses between resolvers — ideal for deeper troubleshooting.

# Linux: Networking

## 🏷️ Local Name & Host Configuration

/etc/hosts
Local file mapping hostnames to IPs.
Used for quick overrides or testing — takes priority over DNS.
cat /etc/hosts

/etc/resolv.conf
Lists DNS servers and search domains.
Usually managed by DHCP or NetworkManager.
cat /etc/resolv.conf

/etc/hostname
Stores the system's hostname — what other machines see you as on the network.
cat /etc/hostname

💡 These files define how your system identifies itself and resolves names — local mappings first, then DNS servers.

# Linux: Networking

## 🔍 Checking Network Sockets – ss

The **ss (socket statistics)** command shows what services are **listening** for connections and what traffic is currently **established**. It replaces the older netstat command.

sudo ss -tulanp

**Common options:**

- -t → TCP sockets
- -u → UDP sockets
- -l → Listening sockets
- -a → All sockets (listening + established)
- -n → Show ports as numbers (skip DNS lookup)
- -p → Show process (and PID) using the socket

**Example:**

```
State    Recv-Q Send-Q Local Address:Port   Peer Address:Port   Process
LISTEN   0      128    0.0.0.0:22          *:*                 sshd
ESTAB    0      0      172.31.1.87:22      172.31.0.5:51512    sshd
```

💡 ss shows which ports are open (LISTEN) and who's connected (ESTAB) — perfect for verifying running services or diagnosing active connections.

# Linux: Networking

## 📥 Using curl for Endpoints & File Downloads

The **curl** command tests **application-level (HTTP/HTTPS)** connectivity.
It works over **TCP (Layer 4)** to verify that a web service is responding correctly. Curl can **interact with web endpoints** and **download files**, just like wget.

**Check an API endpoint:**
curl https://api.github.com

Displays the raw JSON response from a public API — useful for testing APIs or verifying authentication.
**Download a file:**
curl -O https://example.com/file.zip

Saves the file with its original name (-O = output to file).

**Save with a custom name:**
curl -o myfile.zip https://example.com/file.zip

**Follow redirects (like shortened URLs):**
curl -L https://short.url/download

💡 curl can act like a mini web client — ideal for checking API responses, verifying web services, or downloading files directly from the terminal.

# Linux: Networking

## 🌐 Testing Web Connectivity – curl

Verify that a web service is responding correctly.
curl -I https://www.google.com

Shows only the **HTTP headers**, including status code (200 OK, 404 Not Found, etc.).
curl -o /dev/null -s -w "HTTP Status: %{http_code}\n" https://www.google.com

Quietly checks a site and prints just the status code — perfect for quick health checks.

**Common uses:**
- Confirm web services are reachable
- Measure response time or status
- Debug SSL / redirects with -v (verbose)

💡 curl goes beyond "is the host alive?" — it confirms the **application itself is responding** and returns the exact HTTP status.

# Linux: Networking

## 🛰️ Trace the Network Path – traceroute

The **traceroute** command shows the exact **path packets take** from your system to a destination.
Each "hop" represents a **router or gateway** the packet passes through.

sudo traceroute google.com

**How it works:**

- Sends packets with increasing **TTL (Time To Live)** values.
- Each router returns a message when the packet expires, revealing its address.
- Useful for finding **slow hops**, **network loops**, or **unnecessary routing**.
- Within local or cloud networks, it helps you **optimize routes** between services.

💡 traceroute maps the journey of your packets — great for spotting bottlenecks or misrouted traffic.

# Linux: Networking

## 📡 Capturing Traffic – tcpdump

The **tcpdump** command captures **real packets** as they travel through network interfaces.It's the foundation of network analysis and security monitoring.

sudo tcpdump -c 10

**Common uses:**
View live traffic in the terminal
Filter by protocol or port
sudo tcpdump -i any icmp
sudo tcpdump -i any port 80

Save packets to a file for Wireshark:
sudo tcpdump -w capture.pcap

# Linux: Networking

## 🔥 Managing the Firewall

A **firewall** controls **which network traffic is allowed or blocked** based on rules and zones.
It protects your system by filtering incoming and outgoing packets.

**firewalld** is the **modern replacement for iptables** on RHEL, CentOS, Fedora, and Amazon Linux.
It's **zone-based** (e.g., public, internal, trusted) and much easier to manage dynamically.

# Linux: Networking

## 🔥 Managing the Firewall – firewalld

**View current rules:**
sudo firewall-cmd --list-all

**Add a rule (example – allow HTTP traffic):**
sudo firewall-cmd --add-service=http --permanent
sudo firewall-cmd --reload

**Verify the update:**
sudo firewall-cmd --list-all

💡 firewalld is the modern Linux firewall. It is easier than iptables and supports dynamic changes without breaking active connections.

# Lab 4.2 networking

Estimated Time: 30

# POP QUIZ:

What command displays your system's current IP configuration?

A. nslookup
B. ping
C. ip addr show
D. ip link show

# POP QUIZ:

What command displays your system's current IP configuration?

A. nslookup
B. ping
C. ip addr show
D. ip link show

"What layer on OSI is ip addr show focused on?"

# POP QUIZ:

What command can be used to view an MX record for a domain?

A. traceroute
B. dig
C. ss
D. ip neig

# POP QUIZ:

What command can be used to view an MX record for a domain?

A. traceroute
B. dig
C. ss
D. ip neig

"What do the other commands do?"

# Linux: Service Management

Service management is at the heart of Linux administration. Every application, background process, and system component runs as a **service** that must be properly configured, monitored, and maintained. Knowing how to start, stop, enable, or inspect these services ensures that critical applications remain available and responsive across reboots and system updates.

For a system administrator, mastering service management means having full control over the system's behavior. It allows you to manage dependencies, troubleshoot startup failures, and automate recovery when something goes wrong. Whether it's a web server, database, or background daemon, understanding how services are managed through *systemd* is essential for keeping a stable and reliable environment.

# Linux: Service Management

## ⚙️ Slide: What Is a Service in Linux?

A **service** is a **background process** that continuously performs a **service for the system or users** — for example, serving files, handling network requests, or maintaining system functions.

🧩 Key Points

- Runs **without direct user interaction**, often starting **automatically at boot**.
- Managed by **systemd**, which controls how it starts, stops, restarts, and logs activity.
- Most services are **daemons** — special background processes that "serve" requests as they come in.
- A daemon's name often ends with **d**, indicating it "serves" something:
  - sshd → Serves secure shell connections
  - httpd → Serves web pages and files
  - crond → Serves scheduled job execution
  - systemd → Serves as the master service manager

💡 *A **service** performs a service — it stays running in the background, ready to **serve requests**, whether that means sending a file, opening a shell, or managing a resource.*

# Linux: Service Management

## ⚙️ Understanding systemd – The Linux Service Manager

**systemd** is the **modern replacement for the older init system**.
It's the **first process started after the kernel boots** and always runs with **PID 1**.

**What it does:**

- Initializes the system and manages all background **services (called units)**
- Starts required services (like sshd, network, firewalld) in the correct order
- Monitors them and restarts if they fail
- Marks the system as **"ready"** once all essential units are active

```
systemd (PID 1)
  ├─ sshd.service
  ├─ NetworkManager.service
  └─ firewalld.service
```

💡 systemd is the master process of Linux — it launches, monitors, and coordinates every service your system depends on.

# Linux: Service Management

## 🧩 What Else systemd Does

systemd doesn't just start background services — it coordinates the **entire system startup process**.It manages everything the system needs to become fully operational.

**At boot, systemd is responsible for:**

- Mounting and checking **file systems**
- Setting up **swap space** and other storage units
- Bringing up **network interfaces**
- Launching **system targets** (boot stages like multi-user.target, graphical.target)
- Spawning **login prompts** and managing **user sessions**
- Handling **timers**, **sockets**, and **device events**
- Managing **shutdown and reboot** sequences gracefully

💡 systemd is not just a service starter — it's the central **orchestrator of the entire Linux boot and runtime environment.**

# Linux: Service Management

## 🧰 Viewing and Checking Services – systemctl

systemctl is the main command used to interact with **systemd**.
It can list, start, stop, and check the status of any **unit** (service, socket, mount, etc.).

**Check version and verify systemd is active:**
`systemctl --version`

**List all running services:**
`systemctl list-units --type=service --state=running`

**View the status of a specific service:**
`systemctl status sshd`

Shows if the service is **active**, when it **started**, and its **last log messages**.

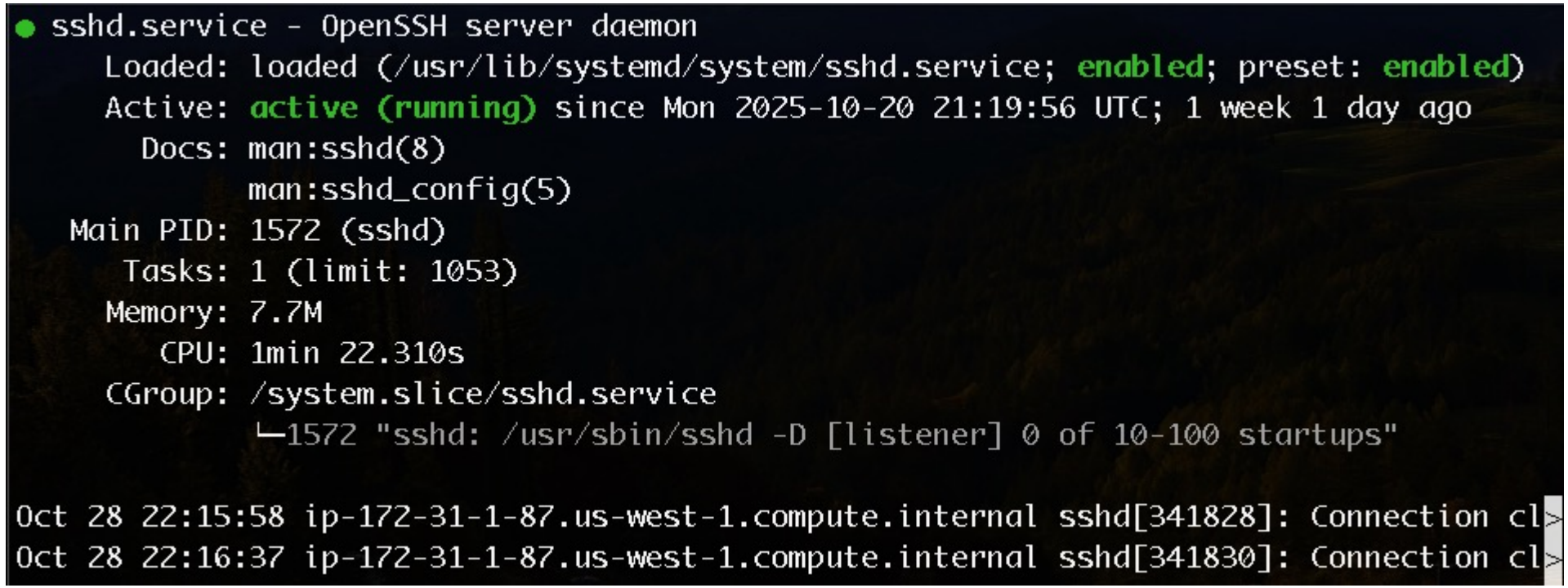
💡 systemctl gives you visibility into every process managed by systemd — from all running units to the detailed status of a single service.

# Linux: Service Management

## Viewing and Checking Services – systemctl

View the status of a specific service:

systemctl status sshd



# Linux: Service Management

```
● sshd.service - OpenSSH server daemon
     Loaded: loaded (/usr/lib/systemd/system/sshd.service; enabled; preset: enabled)
     Active: active (running) since Mon 2025-10-20 21:19:56 UTC; 1 week 1 day ago
```

Shown in the box above is the unit file. This contains the instructions for how Systemd handles the service, like when it should start, what command to run, how to restart if it fails, and what other services it depends on.

The commands below shows you how to print (cat) or edit the unit file. Take note of the ExecStart (command that starts the service).

systemctl cat sshd

# sudo systemctl edit sshd # Do not edit

```
[ec2-user@ip-172-31-1-87 ~]$ systemctl cat NetworkManager
No files found for NetworkManager.service.
[ec2-user@ip-172-31-1-87 ~]$ systemctl cat sshd
# /usr/lib/systemd/system/sshd.service
[Unit]
Description=OpenSSH server daemon
Documentation=man:sshd(8) man:sshd_config(5)
After=network.target sshd-keygen.target
Wants=sshd-keygen.target

[Service]
Type=notify
EnvironmentFile=-/etc/sysconfig/sshd
ExecStart=/usr/sbin/sshd -D $OPTIONS
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process
Restart=on-failure
RestartSec=42s

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

# Linux: Service Management

A **cgroup (control group)** is a Linux kernel feature that **organizes, isolates, and restricts processes** into groups. It allows administrators to **limit, monitor, and control** each group's access to system resources — such as CPU, memory, and I/O — ensuring both **resource fairness and process isolation**.

```
     CGroup: /system.slice/sshd.service
             └─1572 "sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startu
```

```
[ec2-user@ip-172-31-1-87 ~]$ ls /sys/fs/cgroup/system.slice/sshd.service/
cgroup.controllers      cpu.idle         memory.current       memory.stat
cgroup.events           cpu.max          memory.events        memory.swap.current
cgroup.freeze           cpu.max.burst    memory.events.local  memory.swap.events
cgroup.kill             cpu.pressure     memory.high          memory.swap.high
cgroup.max.depth        cpu.stat         memory.low           memory.swap.max
cgroup.max.descendants  cpu.weight       memory.max           memory.zswap.current
cgroup.pressure         cpu.weight.nice  memory.min           memory.zswap.max
cgroup.procs            io.bfq.weight    memory.numa_stat     pids.current
cgroup.stat             io.max           memory.oom.group     pids.events
cgroup.subtree_control  io.pressure      memory.peak          pids.max
cgroup.threads          io.stat          memory.pressure      pids.peak
cgroup.type             io.weight        memory.reclaim
[ec2-user@ip-172-31-1-87 ~]$ cat /sys/fs/cgroup/system.slice/sshd.service/cpu.max
max 100000
```

# Linux: Service Management

## 🌐 Managing a Web Service – httpd Example

Install and manage a simple web server to practice systemctl commands safely.

**Install Apache (httpd):**
sudo dnf install -y httpd

**Start and check the service:**
sudo systemctl start httpd
curl http://localhost # verify its working
sudo systemctl status httpd

**Enable it to start on boot:**
sudo systemctl enable httpd

**Restart or stop when needed:**
sudo systemctl restart httpd
sudo systemctl stop httpd

💡 The httpd service is perfect for learning systemctl

# Linux: Service Management


## Viewing Service Logs – journalctl

The **journal** is systemd's built-in logging system. It collects logs from all services, the kernel, and the system — including both service output (stdout/stderr) and structured log messages — into a single, searchable database.

**View logs for a specific service:**

```
sudo journalctl -u sshd --no-pager
sudo journalctl -u sshd --no-pager | grep Accepted # all ssh logins
```

**Follow logs in real time (like tail -f):**

```
sudo journalctl -u sshd –f --no-pager
```

You can see **login attempts, authentication failures, and connection events** from the SSH service directly here.

💡 journalctl provides a unified way to review and troubleshoot service activity — essential for auditing and diagnosing issues.

# Lab 4.3 Service Management

Estimated Time: 20

# POP QUIZ:

What is the **first process started** on a modern Linux system after boot?

A. systemd
B. systemctl
C. init
D. bios

# POP QUIZ:

What is the **first process started** on a modern Linux system after boot?

A. systemd
B. systemctl
C. init
D. bios

"What is systemctl?"

# POP QUIZ:

Which Linux feature is used to **limit CPU, memory, and I/O resources** for processes?

A. namespaces
B. SELinux
C. CGroups
D. systemctl

# POP QUIZ:

Which Linux feature is used to **limit CPU, memory, and I/O resources** for processes?

A. namespaces
B. SELinux
C. CGroups
D. systemctl

"What other technologies rely on CGroups?"

# POP QUIZ:

What file tells systemd how and when to start/stop/restart a service?

A. Log file
B. System file
C. CGroup file
D. Unit file

# POP QUIZ:

What file tells systemd how and when to start/stop/restart a service?

A. Log file
B. System file
C. CGroup file
D. Unit file

"How can we find/edit the unit file of a service?"

# POP QUIZ:

What does it mean if a service is enabled?

A. It is running
B. It is turned on when booting
C. The service is active and up to date
D. The service is installed

# POP QUIZ:

What does it mean if a service is enabled?

A. It is running
B. It is turned on when booting
C. The service is active and up to date
D. The service is installed

"How can we enable it and confirm? What about turn on/off/restart a service?"

# Linux: Disk Management and Filesystems

Disk management is a cornerstone of Linux system administration. Every file, directory, and service depends on properly configured storage. Understanding how to inspect disks, create partitions, format filesystems, and mount them ensures that data is organized, accessible, and resilient against failure.

For a system administrator, mastering disk and filesystem management means maintaining stability and performance. It allows you to expand storage, troubleshoot space issues, and automate mounts that persist across reboots. Whether you're managing cloud volumes or physical drives, these skills form the foundation of reliable system operations.

# Linux: Disk Management and Filesystems

## 🧩 Getting Hands-On with Disk Management

In this course, we'll explore how Linux detects, organizes, and manages storage devices. You'll learn to view disks and partitions, create filesystems, and mount them to make data accessible through the Linux directory tree.

We'll also practice using a **loop device** — a special file that behaves like a virtual disk. This lets you safely simulate real-world disk operations without touching system drives. By the end, you'll know how to mount, unmount, and verify storage devices just like a production environment.

💡 *Every filesystem begins as a block device — today, you'll learn how to turn raw storage into usable space.*

# Linux: Disk Management and Filesystems

## 🧩 Viewing Disk and Filesystem Information

We'll begin by exploring how Linux identifies and organizes disks. Using commands like **lsblk**, **df**, and **fdisk**, you'll inspect block devices, partitions, and filesystems. These tools help you confirm what storage exists, how it's mounted, and how much space is available.

You'll also use **blkid** to view filesystem UUIDs — unique identifiers that are critical for mounting drives reliably through /etc/fstab. By understanding this information, you'll know exactly what storage your system has and how it's being used.

💡 *Before you can manage disks, you must first see them clearly — inspection is always step one.*

# Linux: Disk Management and Filesystems

## 🧩 Understanding Mount Points

Every file in Linux exists within a single unified directory tree that begins at /. Disks and partitions don't appear automatically — they must be **mounted** to a directory, called a **mount point**, so that their data becomes accessible to the system.

You can view active mounts with:
mount | column -t
findmnt

These commands reveal where each filesystem is attached and what type it is.

💡 *In Linux, drives don't get letters like C: or D: — they become part of the filesystem tree once mounted.*

# Linux: Disk Management and Filesystems

```
[ec2-user@ip-172-31-1-87 ~]$ findmnt
TARGET                          SOURCE          FSTYPE  OPTIONS
/                               /dev/nvme0n1p1  xfs     rw,noatime,seclabel,attr2,inode
├─/proc                         proc            proc    rw,nosuid,nodev,noexec,relatime
│ └─/proc/sys/fs/binfmt_misc    systemd-1       autofs  rw,relatime,fd=33,pgrp=1,timeou
│   └─/proc/sys/fs/binfmt_misc  binfmt_misc     binfmt_ rw,nosuid,nodev,noexec,relatime
├─/sys                          sysfs           sysfs   rw,nosuid,nodev,noexec,relatime
│ ├─/sys/kernel/security        securityfs      securit rw,nosuid,nodev,noexec,relatime
│ ├─/sys/fs/cgroup              cgroup2         cgroup2 rw,nosuid,nodev,noexec,relatime
│ ├─/sys/fs/pstore              pstore          pstore  rw,nosuid,nodev,noexec,relatime
│ ├─/sys/firmware/efi/efivars   efivarfs        efivarf rw,nosuid,nodev,noexec,relatime
│ ├─/sys/fs/bpf                 bpf             bpf     rw,nosuid,nodev,noexec,relatime
│ ├─/sys/fs/selinux             selinuxfs       selinux rw,nosuid,noexec,relatime
│ ├─/sys/kernel/tracing         tracefs         tracefs rw,nosuid,nodev,noexec,relatime
│ ├─/sys/kernel/debug           debugfs         debugfs rw,nosuid,nodev,noexec,relatime
│ │ └─/sys/kernel/debug/tracing tracefs         tracefs rw,nosuid,nodev,noexec,relatime
│ ├─/sys/fs/fuse/connections    fusectl         fusectl rw,nosuid,nodev,noexec,relatime
│ └─/sys/kernel/config          configfs        configf rw,nosuid,nodev,noexec,relatime
├─/dev                          devtmpfs        devtmpf rw,nosuid,seclabel,size=4096k,n
│ ├─/dev/shm                    tmpfs           tmpfs   rw,nosuid,nodev,seclabel
│ ├─/dev/pts                    devpts          devpts  rw,nosuid,noexec,relatime,secla
│ ├─/dev/hugepages              hugetlbfs       hugetlb rw,relatime,seclabel,pagesize=2
│ └─/dev/mqueue                 mqueue          mqueue  rw,nosuid,nodev,noexec,relatime
├─/run                          tmpfs           tmpfs   rw,nosuid,nodev,seclabel,size=1
│ └─/run/user/1000              tmpfs           tmpfs   rw,nosuid,nodev,relatime,seclab
├─/tmp                          tmpfs           tmpfs   rw,nosuid,nodev,seclabel,size=4
├─/boot/efi                     systemd-1       autofs  rw,relatime,fd=51,pgrp=1,timeou
│ └─/boot/efi                   /dev/nvme0n1p128
│                                               vfat    rw,noatime,fmask=0077,dmask=007
└─/var/lib/nfs/rpc_pipefs       sunrpc          rpc_pip rw,relatime
```

# Linux: Disk Management and Filesystems

## 🧩 Understanding findmnt Output

The findmnt command shows a **tree view** of all mounted filesystems and their relationships. Each entry lists where a device or virtual filesystem is mounted and how it's being used.

Key columns:

- TARGET – Directory mount point (e.g., /, /proc, /boot/efi)
- SOURCE – Device or virtual source (e.g., /dev/nvme0n1p1, proc)
- FSTYPE – Filesystem type (e.g., xfs, ext4, tmpfs, vfat)
- OPTIONS – Mount options controlling behavior and permissions

💡 *Everything in Linux — even memory and kernel data — is mounted somewhere in the filesystem tree.*

# Linux: Disk Management and Filesystems

## 🧩 Virtual Filesystems in Linux

Not all mounted filesystems are real disks. Many are **virtual filesystems** created by the kernel for system operations, monitoring, or process communication.

**Common examples:**

- **proc** – Exposes process and kernel information (/proc/cpuinfo)
- **sysfs** – Hardware and device info under /sys
- **tmpfs** – Temporary in-memory storage (used by /tmp, /run)
- **devtmpfs / devpts** – Device nodes and terminals
- **cgroup2** – Resource control for grouped processes
- **tracefs / debugfs** – Used for tracing and debugging kernel events

💡 *Virtual filesystems act as live "windows" into the kernel — showing how your system operates, not where data is stored.*

# Linux: Disk Management and Filesystems

## 🧩 Understanding Mount Options

Mount options define **how** a filesystem behaves after being attached — controlling performance, security, and access.

**Common options:**

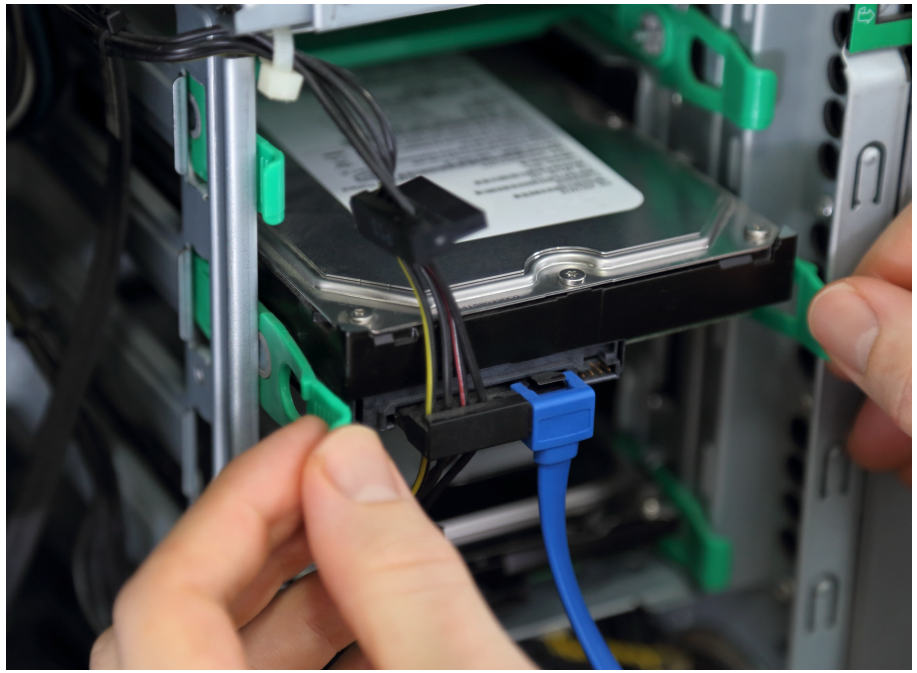
- **rw** – Read and write access (default for most)
- **noexec** – Prevent running executables (security)
- **nosuid** – Ignore set-user-ID bits (no privilege elevation)
- **nodev** – Prevent device files on this mount
- **noatime / relatime** – Controls file access time updates (performance tweak)
- **attr2** – Enables extended attributes (XFS feature)
- **seclabel** – Allows SELinux labels
- **inode** – Internal metadata structure for each file

💡 *Mount options fine-tune your filesystem's speed and security — always review them before making changes.*

# Linux: Disk Management and Filesystems

## 🧩 Simulating Disks with Loop Devices

In this section, we'll create a **virtual disk** — a simple file that behaves like a real block device. This lets us safely practice partitioning, formatting, and mounting **without touching the system's actual drives.**



💡 *Loop devices are perfect for testing — you can create, format, and mount them just like real disks, then detach when done.*

# Linux: Disk Management and Filesystems

## 🧩 Creating a Virtual Disk File

We use the dd command to create a **blank disk image** that we'll later format and mount.
dd if=/dev/zero of=~/disk_lab/virtual_disk.img bs=1M count=100

**Command breakdown:**
- **if=/dev/zero** – Input file; provides a stream of zero bytes (empty data)
- **of=virtual_disk.img** – Output file; where the data is written
- **bs=1M** – Block size of 1 MB (each write operation is 1 MB)
- **count=100** – Number of blocks to write (total 100 MB file)

Next, we attach this file as a **loop device** so Linux treats it like a real disk:
sudo losetup -fP ~/disk_lab/virtual_disk.img

**Flags explained:**
- **-f** – Finds the first free loop device (e.g., /dev/loop0)
- **-P** – Forces a partition scan so the kernel recognizes it as a block device

💡 *You've just created a "pretend" hard drive — perfect for practicing without risk to your real system.*

# Linux: Disk Management and Filesystems

## 🧩 Verifying the Loop Device

After creating the loop device, confirm that Linux recognizes it as a **block device** using inspection tools.
lsblk | grep loop
sudo losetup -l

**Command overview:**

- **lsblk** – Lists all block devices, their sizes, and mount points
- **grep loop** – Filters output to show only loop devices
- **losetup -l** – Displays detailed information about all active loop devices

Typical output shows your new loop device (e.g., /dev/loop0) with the file it's linked to and its size.

💡 *Treat loop devices like any other disk — you can partition, format, and mount them just the same.*

# Linux: Disk Management and Filesystems

## 🧩 Creating a Filesystem on the Loop Device

Now that the loop device is ready, we'll format it with a filesystem so it can store data.
sudo mkfs.ext4 -L "TestDisk" /dev/loop0

Command breakdown:
- **mkfs.ext4** – Creates a new **ext4** filesystem
- **-L "TestDisk"** – Sets a human-readable label for the filesystem
- **/dev/loop0** – Target device to format

Common filesystem types:
- **ext4** – Reliable default for most Linux systems
- **xfs** – High performance, used on RHEL/Amazon Linux
- **btrfs** – Modern, supports snapshots and compression
- **vfat** – Compatible with Windows (FAT32)

💡 *A disk isn't usable until it's formatted*
*filesystems define how data is stored and organized on disk.*

# Linux: Disk Management and Filesystems

## 🧩 Mounting the Filesystem

Once the filesystem is created, we **mount** it to a directory so it becomes accessible through the Linux filesystem tree.

```
mkdir -p ~/disk_lab/mnt_test
sudo mount /dev/loop0 ~/disk_lab/mnt_test
```

**Command breakdown:**

- **mount** – Attaches a filesystem to a directory (the mount point)
- **/dev/loop0** – The device or partition being mounted
- **~/disk_lab/mnt_test** – The directory where it will appear

Before mounting, this directory is empty on your main disk.
After mounting, it becomes the **entry point** to your new filesystem — any file you create here will be stored *inside* the mounted loop device.

# Linux: Disk Management and Filesystems

## Verifying Mounted Filesystem

**Check that the filesystem is mounted**

```
# Shows where /dev/loop0 is attached
mount | grep loop0
```

**Create a test file**

```
sudo touch ~/disk_lab/mnt_test/test.txt
sudo ls ~/disk_lab/mnt_test
```

You should see test.txt. This proves data written here is stored inside the filesystem on your image file.

**Confirm available space**

```
df -h ~/disk_lab/mnt_test
df -i ~/disk_lab/mnt_test
```

Displays disk usage — the mounted .img behaves like a real drive.

# Linux: Disk Management and Filesystems

## 🧩 Understanding /etc/fstab

The /etc/fstab file defines which filesystems should mount automatically at boot.
Each line lists a device (often by UUID), its mount point, type, and options.

**Example:**
UUID=xxxx  /data  ext4  defaults  0  2

**Fields explained briefly:**
- **Device** — identifies the drive (UUID or /dev/sdX)
- **Mount point** — where it attaches in the directory tree
- **Type** — filesystem type (ext4, xfs, etc.)
- **Options** — behaviors (defaults, noatime, ro, etc.)
- **Dump & Pass** — backup and boot check flags

💡 *fstab automates mounting, but misconfigurations can prevent the system from booting. Always test changes with mount -a before rebooting.*

# Linux: Disk Management and Filesystems

## 🧹 Cleanup and Reset

When you're done with the lab, it's good practice to remove all temporary mounts and devices.

```
sudo umount ~/disk_lab/mnt_test
sudo losetup -D
rm -rf ~/disk_lab
```

💡 *Always unmount filesystems before deleting their source files. This ensures data integrity and keeps your lab environment clean.*

# Linux: Disk Management and Filesystems

| Command | Description |
|---|---|
| lsblk | Lists all block devices (disks, partitions, loop devices) and their mount points. |
| fdisk -l | Displays detailed partition tables and disk geometry (read-only view). |
| blkid | Shows filesystem UUIDs, labels, and types — useful for identifying devices in /etc/fstab. |
| df -h | Displays mounted filesystems and their disk usage in human-readable format. |
| df -i | Shows inode usage — useful when you have many small files. |
| du -h --max-depth=1 | Summarizes directory sizes to find what's using space. |
| findmnt | Shows mounted filesystems in a tree format; can filter by type or options. |
| mount | Attaches a filesystem to a directory (mount point). |
| umount | Detaches a mounted filesystem safely. |
| losetup | Manages loop devices — maps regular files as block devices. |
| mkfs.ext4 | Creates a new filesystem (format) on a device or loop file. |
| dd if=/dev/zero of=file.img | Creates a test image file filled with zeros — simulates a blank disk. |

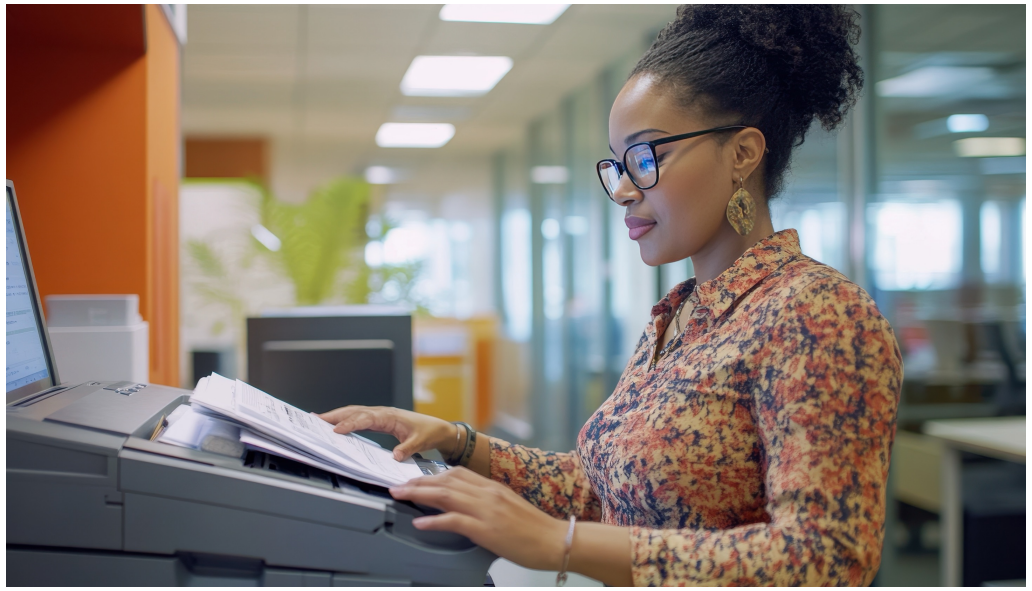
# Linux: Disk Management and Filesystems

## 🔄 File Transfer & Synchronization Tools

Linux provides several tools for securely transferring and syncing files between systems.

🧩 Common Commands:

- rsync – Synchronizes files and directories (local or remote).
  - Efficiently copies only changes.
  - Use -a to preserve permissions, ownership, and timestamps.
- scp – Securely copies files over SSH.
  - Simple one-time copy (less efficient than rsync).
  - Preserves permissions by default.
- sftp – Interactive file transfer session over SSH.
  - Works like FTP but encrypted.
  - Preserves permissions when uploading/downloading.



# Lab 4.4 Disk Management

Estimated Time: 30

# POP QUIZ:

Which command lists all block devices and their mount points?

A. blkid
B. du
C. lsblk
D. findmnt

# POP QUIZ:

Which command lists all block devices and their mount points?

A. blkid
B. du
C. lsblk
D. findmnt

"What is difference between a block device and mount?"

# POP QUIZ:

Which command attaches a filesystem to a directory (mount point)?

A. mount
B. mkfs.ext4
C. du
D. umount

# POP QUIZ:

Which command attaches a filesystem to a directory (mount point)?

A. mount
B. mkfs.ext4
C. du
D. umount

"Can files be added to a block before mounting?"

# POP QUIZ:

What does mkfs.ext4 do?

A. Mounts an ext4 filesystem
B. Checks filesystem errors
C. Formats a partition with the ext4 filesystem
D. Lists all files in a directory

# POP QUIZ:

What does mkfs.ext4 do?

A. Mounts an ext4 filesystem
B. Checks filesystem errors
C. Formats a partition with the ext4 filesystem
D. Lists all files in a directory

"Elaborate on the order of when a block is created, formatted and mounted. When do inodes get created?"

# POP QUIZ:

Which commands can I use to copy files from a server to my local machine?

A. sc
B. du
C. cp
D. rsync

# POP QUIZ:

Which commands can I use to copy files from a server to my local machine?

A. sc
B. du
C. cp
D. rsync

"Which is most efficient?"

# Linux Day 4 Complete

## 🎉 Great Work Today!

You learned how to:

- Manage software with **DNF**
- Manage **network connectivity**
- Control **services and daemons**
- Format and mount **disks & filesystems**

💡 Keep practicing — every command brings you closer to mastering Linux!