

Linux Directory Structure

Everything in Linux starts from the **root directory /**.

1. Root Directory /

- The top-most directory in Linux.
 - All other files and directories stem from /.
-

2. Common Directories Under /

Directory Purpose

/bin	Binary executables (essential commands like ls, cp, mv, cat) used by all users.
/boot	Files needed to boot the system (kernel, bootloader, initramfs).
/dev	Device files (e.g., /dev/sda for hard drives, /dev/null).
/etc	Configuration files for the system and installed programs.
/home	Home directories for normal users (/home/user1, /home/user2).
/lib, /lib64	Libraries needed by binaries in /bin and /sbin.
/media	Mount points for removable media like USB drives, CDs.
/mnt	Temporary mount point for manually mounted filesystems.
/opt	Optional third-party software .
/proc	Virtual filesystem for process information (e.g., /proc/cpuinfo).
/root	Home directory of the root user .
/run	Temporary runtime data (like PID files, sockets).
/sbin	System binaries (commands for admin tasks like fdisk, mkfs).
/srv	Data for services (like web server files in /srv/www).
/sys	Information about devices and the kernel .
/tmp	Temporary files (cleared at reboot).

Directory Purpose

/usr **User programs** and utilities (e.g., /usr/bin, /usr/lib).

/var Variable files: logs (/var/log), mail, spool, temporary cache.

📌 Visual Representation:

```
/  
└── bin      → Essential command binaries  
└── boot     → Boot loader files  
└── dev      → Device files  
└── etc      → Configuration files  
└── home     → User home directories  
└── lib       → Essential shared libraries  
└── media    → Removable media mount points  
└── mnt      → Temporary mount points  
└── opt       → Optional software  
└── proc     → Process and kernel information  
└── root     → Root user's home  
└── run      → Runtime data  
└── sbin     → System binaries  
└── srv      → Service data  
└── sys      → Kernel and hardware info  
└── tmp      → Temporary files  
└── usr      → User programs and utilities  
└── var      → Variable data like logs
```

If you want, I can make you a **colorful diagram of the Linux directory structure** so it's easier to remember visually. That would make it much simpler for EL10 study.

Got it — let's go into **full detail** on each Linux directory, not just *what* it is but **why it's used**, with examples so you can connect it to real-world Linux use.

Linux Directory Structure — Detailed Explanation

Everything starts from **/ (root directory)**.

Think of **/** as the **trunk of a tree**, and all other folders are branches.

1. / — Root Directory

- **Purpose:** The base of the entire Linux filesystem.
 - **Why it's used:** Without **/**, nothing else exists — it's where all paths begin (**/home**, **/etc**, **/usr**, etc.).
 - **Example:** When you type **/etc/passwd**, Linux starts at **/** and looks inside etc.
-

2. /bin — Essential User Binaries

- **Purpose:** Stores **basic commands** needed by all users (both normal users and root).
 - **Why:** These commands are necessary even if no other filesystems are mounted (like in rescue mode).
 - **Example:** Commands like **ls**, **cp**, **mv**, **cat** are here. Without **/bin/ls**, you couldn't list files.
-

3. /boot — Boot Loader Files

- **Purpose:** Contains everything needed to start Linux (kernel, bootloader configs).
 - **Why:** Without **/boot**, your system can't start.
 - **Example:** Files like **/boot/vmlinuz** (Linux kernel) and **/boot/grub/grub.cfg** (bootloader config).
-

4. /dev — Device Files

- **Purpose:** Treats hardware as files (everything in Linux is a file!).
- **Why:** Lets programs interact with hardware through file operations (read, write).

- **Example:** /dev/sda is your first hard disk, /dev/null discards data.
-

5. /etc — Configuration Files

- **Purpose:** Stores **system-wide config files**.
 - **Why:** Keeps all configuration centralized for easy management.
 - **Example:** /etc/passwd (user accounts), /etc/ssh/sshd_config (SSH server settings).
-

6. /home — User Home Directories

- **Purpose:** Stores personal files, documents, downloads, etc. for each user.
 - **Why:** Keeps user data separate from system files.
 - **Example:** /home/suneel/Documents/resume.pdf.
-

7. /lib and /lib64 — Shared Libraries

- **Purpose:** Stores **library files** needed by binaries in /bin and /sbin.
 - **Why:** Without libraries like libc.so.6, most programs won't run.
 - **Example:** /lib/x86_64-linux-gnu/libc.so.6.
-

8. /media — Removable Media

- **Purpose:** Auto-mount point for USB drives, CDs, DVDs.
 - **Why:** Standard place to find plugged-in storage devices.
 - **Example:** /media/suneel/USB_DRIVE.
-

9. /mnt — Temporary Mount Point

- **Purpose:** Used by admins to **manually mount** storage devices.
 - **Why:** For temporary testing or manual mounting before adding permanent config.
 - **Example:** mount /dev/sdb1 /mnt to access a drive.
-

10. /opt — Optional Software

- **Purpose:** Stores third-party software that's not part of default Linux.
 - **Why:** Keeps custom software separate from system packages.
 - **Example:** /opt/google/chrome/ for Chrome browser installed manually.
-

11. /proc — Process & Kernel Info

- **Purpose:** A virtual filesystem showing system and process information.
 - **Why:** Lets you check CPU, memory, processes without special tools.
 - **Example:** cat /proc/cpuinfo shows CPU details.
-

12. /root — Root User Home

- **Purpose:** Home directory for the **root** superuser.
 - **Why:** Separate from /home for security reasons.
 - **Example:** /root/.bashrc stores root's shell settings.
-

13. /run — Runtime Data

- **Purpose:** Temporary data stored while system is running (clears on reboot).
 - **Why:** Stores PID files, sockets, and system state during runtime.
 - **Example:** /run/sshd.pid stores SSH server process ID.
-

14. /sbin — System Binaries

- **Purpose:** Commands for **administrative tasks** (only root usually uses them).
 - **Why:** Needed for managing the system.
 - **Example:** fdisk, mkfs, ifconfig.
-

15. /srv — Service Data

- **Purpose:** Data served by network services (web, FTP, etc.).
- **Why:** Keeps service content in a predictable location.

- **Example:** /srv/www for a website.
-

16. /sys — Kernel and Device Info

- **Purpose:** Interface to kernel and hardware settings.
 - **Why:** Allows user-space programs to interact with hardware.
 - **Example:** /sys/class/net/ lists network interfaces.
-

17. /tmp — Temporary Files

- **Purpose:** Stores short-lived temporary files.
 - **Why:** Shared space for apps and users; gets cleared on reboot.
 - **Example:** Browser cache, temporary install files.
-

18. /usr — User Programs

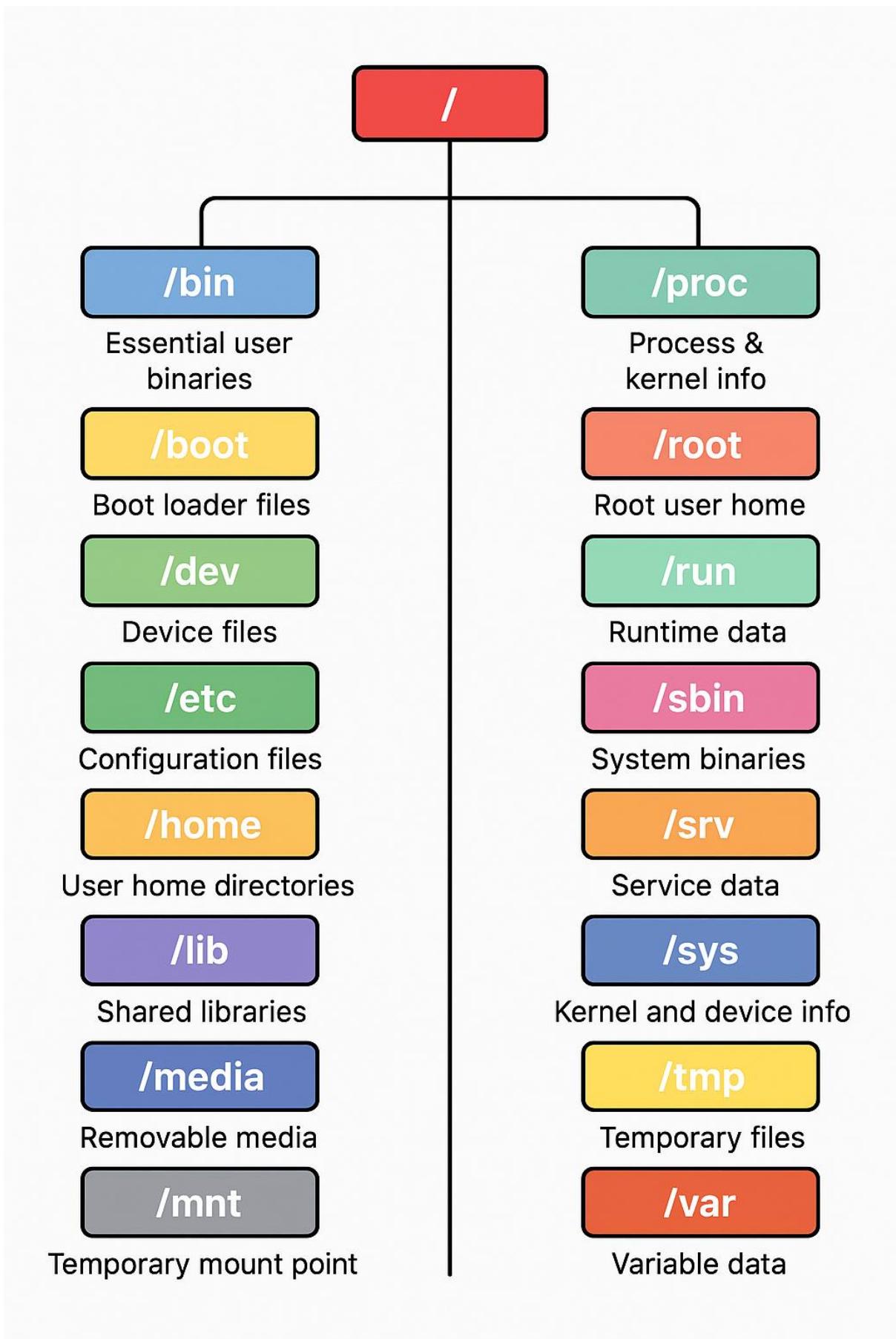
- **Purpose:** Stores most installed applications and utilities.
 - **Why:** Separates OS core from user applications.
 - **Example:** /usr/bin/firefox, /usr/lib/python3.10/.
-

19. /var — Variable Data

- **Purpose:** Stores frequently changing data.
 - **Why:** Prevents constant writes to system directories.
 - **Example:** /var/log/syslog (system logs), /var/mail.
-

Tip:

- /bin, /sbin, /lib → Essential for booting.
 - /usr, /opt → Applications & software.
 - /var, /tmp → Changing data & temp files.
 - /etc → Config brain of Linux.
-



Here's the **difference between cat and touch** in Linux for EL10 level:

1. cat Command

- **Full form:** Concatenate
- **Purpose:** Displays the content of a file, or combines multiple files and prints output.
- **Main use:** View or create a file with content.
- **Example:**
- `cat file.txt`

→ Shows the content of file.txt.

`cat > notes.txt`

Hello World

(Press **CTRL+D** to save) → Creates notes.txt with "Hello World" inside.

2. touch Command

- **Purpose:** Creates an **empty file** or updates the timestamp of an existing file.
- **Main use:** Create blank files quickly.
- **Example:**
- `touch file.txt`

→ Creates file.txt if it doesn't exist.

→ If it exists, updates its **last modified time** without changing the content.

Key Difference Table

Feature	cat	touch
Creates file	<input checked="" type="checkbox"/> Yes (with redirection > and content)	<input checked="" type="checkbox"/> Yes (empty)

Feature	cat	touch
Adds content	<input checked="" type="checkbox"/> Yes	<input type="checkbox"/> No
Displays content	<input checked="" type="checkbox"/> Yes	<input type="checkbox"/> No
Updates timestamp	<input checked="" type="checkbox"/> No	<input checked="" type="checkbox"/> Yes

 **Shortcut to remember:**

- **cat** = "See or write content"
 - **touch** = "Make an empty file or refresh its time"
-
-

1. cp Command

- **Full form:** Copy
- **Purpose:** Creates a **duplicate** of a file or directory in another location.
- **Effect:** The original file **remains**, and a **new copy** is created.
- **Example:**
- `cp file.txt /home/user/Documents/`

→ Copies file.txt to /home/user/Documents/ without removing the original.

2. mv Command

- **Full form:** Move
- **Purpose:** Moves a file or directory from one location to another.
- **Effect:** The **original file is removed** from the source location (like cut-paste).
- **Also used for:** Renaming files.
- **Example (Move):**
- `mv file.txt /home/user/Documents/`

→ Moves file.txt to /home/user/Documents/, removing it from the original location.

- **Example (Rename):**

- `mv oldname.txt newname.txt`
- Renames oldname.txt to newname.txt.
-

Key Difference Table

Feature	<code>cp</code>	<code>mv</code>
Original file kept	<input checked="" type="checkbox"/> Yes	<input type="checkbox"/> No
Creates a duplicate	<input checked="" type="checkbox"/> Yes	<input type="checkbox"/> No
Moves file	<input type="checkbox"/> No	<input checked="" type="checkbox"/> Yes
Can rename	<input type="checkbox"/> No	<input checked="" type="checkbox"/> Yes

💡 Shortcut to remember:

- `cp` → Copy & keep original
 - `mv` → Move & remove original (or rename)
-

let's break down **Soft Link vs Hard Link** for EL10 in a simple but detailed way.

1. What They Are

In Linux, **links** are pointers to files — like shortcuts.

There are **two types**:

- **Hard Link** → Direct reference to the **data** of a file.
 - **Soft Link (Symbolic Link)** → A pointer to the **file's path** (like a shortcut in Windows).
-

2. Difference Between Soft Link and Hard Link

Feature	Hard Link	Soft Link (Symbolic)
Points to	File's actual data (inode)	File's path (name)

Feature	Hard Link	Soft Link (Symbolic)
File must exist	Can exist even if original is deleted (still points to data)	Breaks if original file is deleted
Cross filesystem	✗ Cannot link across different partitions	<input checked="" type="checkbox"/> Can link across partitions
Size	Same size as original (because it's another name for same data)	Very small (just stores path)
Identification	Same inode number as original	Different inode number
Command example	<code>ln file1 file2</code>	<code>ln -s file1 linkname</code>
Use case	Backup references, multiple names for same file	Shortcuts, linking configs, scripts

3. Example

```
# Create a file
```

```
echo "Hello World" > original.txt
```

```
# Hard Link
```

```
ln original.txt hardlink.txt
```

```
# Soft Link
```

```
ln -s original.txt softlink.txt
```

- **Hard Link:** hardlink.txt is another *name* for original.txt.
If you delete original.txt, hardlink.txt still works.
- **Soft Link:** softlink.txt points to original.txt.
If you delete original.txt, softlink.txt becomes broken.

4. Where You Can Use Them

- **Hard Links:**

- Avoid file duplication while keeping multiple names.
- Keep backup names for important files (same data, no extra storage).

- **Soft Links:**

- Create shortcuts to frequently used files/directories.
 - Point scripts or programs to the latest version of a file.
 - Redirect configuration files (e.g., /etc/nginx/sites-enabled/ links to /etc/nginx/sites-available/).
-

 **Quick Memory Tip:**

- **Hard Link** = *Another doorway to the same room.*
 - **Soft Link** = *A signboard pointing to a room — if the room is gone, the signboard is useless.*
-

— for **EL10** let's cover **File Permissions in Linux**, how they work, and **the ways to give them**.

1. Types of Permissions

In Linux, permissions control **who can do what** with a file or directory.

There are **three types** of permissions:

Symbol	Meaning For a File	For a Directory
r	Read	View file content
w	Write	Modify file content
x	Execute	Create/delete files inside
	Run as a program	Enter/access the directory

2. Permission Groups

Each file has permissions for **three groups**:

1. **Owner (u)** → The user who owns the file.
 2. **Group (g)** → Users in the file's group.
 3. **Others (o)** → Everyone else.
-

3. Viewing Permissions

Use:

`ls -l`

Example:

`-rwxr-xr--`

Breakdown:

- - → Regular file
 - rwx → Owner: read, write, execute
 - r-x → Group: read, execute
 - r-- → Others: read only
-

4. Ways to Give Permissions

There are **two main ways**:

A. Symbolic Method

Use letters (u, g, o, a) and symbols (+, -, =):

Symbol Action

- | | |
|---|----------------------|
| + | Add permission |
| - | Remove permission |
| = | Set exact permission |

Examples:

`chmod u+x file.txt # Add execute for owner`

```
chmod g-w file.txt # Remove write from group  
chmod o=r file.txt # Set read-only for others  
chmod a+rwx file.txt # Give read/write to everyone
```

B. Numeric (Octal) Method

Each permission has a number:

- Read (r) = **4**
- Write (w) = **2**
- Execute (x) = **1**

Add them for each group:

- 7 = read+write+execute (**4+2+1**)
- 6 = read+write (**4+2**)
- 5 = read+execute (**4+1**)
- 4 = read-only

Example:

```
chmod 755 script.sh
```

Breakdown:

- **7** → Owner: rwx
 - **5** → Group: r-x
 - **5** → Others: r-x
-

Summary Table

Method	Example	Meaning
--------	---------	---------

Symbolic chmod u+x file Add execute for owner

Numeric chmod 644 file Owner read/write, others read-only

In Linux, you change **ownership** of a file or directory using the **chown** command.

1. Ownership in Linux

Each file has:

- **Owner (User)** → The person who owns the file.
- **Group** → A set of users who share certain permissions.

Example:

```
ls -l
```

```
-rw-r--r-- 1 suneel devgroup 1024 Aug 14 file.txt
```

Here:

- Owner = **suneel**
 - Group = **devgroup**
-

2. Change Ownership Command

Syntax

```
chown [new_owner] [file]
```

Example

```
chown raj file.txt
```

→ Changes owner of file.txt to **raj**.

3. Change Owner and Group Together

Syntax:

```
chown owner:group file
```

Example:

```
chown raj:developers file.txt
```

→ Owner becomes **raj**, group becomes **developers**.

4. Change Ownership for Multiple Files

```
chown raj file1.txt file2.txt
```

5. Change Ownership Recursively (for folders)

chown -R raj:developers /project

→ Changes ownership of /project and **everything inside it.**

6. Important Notes

- Only **root** or a user with sudo privileges can change ownership.
 - You can **only change the group** if you are in that group.
-

Shortcut to remember:

- **chown** = *C*hange *O*WNership
 - **Format:** chown user:group file
-

Here's the **difference between head and tail commands** for EL10, explained clearly.

1. head Command

- **Purpose:** Shows the **first part** of a file.
- **Default behavior:** Displays the **first 10 lines**.
- **Syntax:**
- head filename
- **Example:**
- head notes.txt

→ Shows the first 10 lines of notes.txt.

- **Custom lines:**
- head -n 5 notes.txt

→ Shows the first 5 lines.

2. tail Command

- **Purpose:** Shows the **last part** of a file.
- **Default behavior:** Displays the **last 10 lines**.
- **Syntax:**
- `tail filename`
- **Example:**
- `tail notes.txt`

→ Shows the last 10 lines of notes.txt.

- **Custom lines:**
- `tail -n 8 notes.txt`

→ Shows the last 8 lines.

- **Live monitoring:**
- `tail -f logfile.log`

→ Shows new lines in a file as they are added (used for log monitoring).

3. Key Difference Table

Feature	head	tail
Shows	First lines of a file	Last lines of a file
Default lines	10	10
Live follow	 No	 Yes (-f option)

Memory Trick:

- **head** → *Top of the file*
 - **tail** → *Bottom of the file*
-

```
tail -f CAN EXPALIN
```

In Linux,

tail -f is used to **continuously monitor a file** in real-time, showing new lines as they are added.

How it works

- Normally, tail filename shows only the **last 10 lines** of a file.
 - With the -f (follow) option, it **keeps the terminal open** and updates the output when the file changes.
-

Syntax

```
tail -f filename
```

Example

Imagine a log file that keeps getting updated:

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

This will:

- Show the **last 10 lines** of syslog.
 - Keep printing new log entries as they're written.
-

When to use it

- **Monitoring logs** in real-time (e.g., Apache, Nginx, system logs)
 - Watching output of an application or service
 - Debugging to see live events
-

Tip:

If you want to see a specific number of last lines and follow updates:

```
tail -n 20 -f /var/log/syslog
```

This shows the last **20 lines** and continues monitoring.

:

Here's the **explanation** for the top command in Linux:

1. What is the top command?

- The top command shows a **real-time view of system processes**.
- It is like the **Task Manager** in Windows but for Linux in the terminal.
- Displays:
 - Running processes
 - CPU usage
 - Memory usage
 - Process IDs
 - Uptime
 - Load average

2. Why do we use top?

- To **monitor system performance** in real-time.
- To **find high CPU or memory consuming processes**.
- To **troubleshoot system slowdowns**.
- To **kill unresponsive processes**.

3. Syntax

```
top
```

4. Key Actions inside top

Key Action

- q Quit top
 - k Kill a process (you enter PID)
 - P Sort by CPU usage
 - M Sort by memory usage
 - h Help menu
-

5. Example Output (Simplified)

```
top - 14:25:30 up 2 days, 3:14, 1 user, load average: 0.23, 0.15, 0.10
Tasks: 193 total, 2 running, 191 sleeping, 0 stopped, 0 zombie
%Cpu(s): 12.5 us, 3.0 sy, 0.0 ni, 80.0 id, 4.5 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 7976.0 total, 3050.0 free, 2500.0 used, 2426.0 buff/cache
MiB Swap: 2048.0 total, 2000.0 free, 48.0 used.
```

6. Where can we use top?

- **DevOps & System Administration:** To monitor server performance.
 - **Developers:** To check if their program is consuming too much CPU or memory.
 - **Troubleshooting:** Find and stop runaway processes.
 - **Performance tuning:** Check load average before deploying services.
-

 **Tip:**

You can use htop (if installed) for a more colorful and user-friendly version of top.

Tar command

Here's **explanation** for the tar command in Linux:

1. What is the tar command?

- tar stands for **Tape Archive**.
 - It's used to **combine multiple files/folders into a single archive file**.
 - Often used for **backups or file transfers**.
 - The archive usually has a .tar extension.
-

2. Why we use tar

- To store multiple files in one package.
 - To **compress files** (when used with gzip/bzip2/xz).
 - To **extract** files from an archive.
 - To **backup directories** while preserving permissions.
-

3. Basic Syntax

tar [options] filename.tar files...

4. Common Options

Option Meaning

- c Create a new archive
- x Extract an archive
- v Verbose (show progress)
- f File name of the archive
- z Compress with gzip (.tar.gz)

Option Meaning

j Compress with bzip2 (.tar.bz2)

5. Examples

Create an archive

`tar -cvf backup.tar /home/user/`

- c → create
 - v → show progress
 - f → filename
-

Extract an archive

`tar -xvf backup.tar`

Create a compressed archive

`tar -czvf backup.tar.gz /home/user/`

Extract a compressed archive

`tar -xzvf backup.tar.gz`

6. Where can we use tar

- **System backups**
 - **Log storage**
 - **Transferring multiple files together**
 - **Packaging software releases**
-

yum upadte and Yum update packagename differnces .

ChatGPT said:

Here's the difference between

yum update vs yum update <packagename> in Linux (RHEL/CentOS-based distros):

1. yum update

- **Updates all packages** on the system to the latest version available in enabled repositories.
- Also updates **dependencies** and **kernel** if available.
- Example:

`sudo yum update`

 **Use when** you want your **whole system** up-to-date.

2. yum update <packagename>

- Updates **only the specified package** to the latest version.
- Does **not** touch other packages unless they're dependencies of the one you're updating.
- Example:

`sudo yum update httpd`

This updates only **Apache HTTP Server**.

 **Use when:**

- You want to **patch or upgrade a specific package**.
- Avoid changing unrelated packages to prevent breaking software.

3. Quick Comparison

Feature	yum update	yum update packagename
Scope	Whole system	Single package
Risk	Higher (may break configs)	Lower (affects only that package)
Speed	Slower	Faster
Use Case	Regular system maintenance	
	Targeted package updates	

The **find** command in Linux is used to **search for files and directories** in a directory hierarchy based on different criteria like name, type, size, permissions, or modification time.

Syntax

`find [path] [options] [expression]`

- **path** → Where to search (e.g., `/home/user`)
 - **options/expression** → What to search for (e.g., name, size, type)
-

Common Examples

1. Find files by name

`find /home -name "file.txt"`

Searches for `file.txt` in `/home`.

2. Case-insensitive search

`find /home -iname "file.txt"`

Finds `file.txt`, `File.txt`, `FILE.TXT`, etc.

3. Find by file type

```
find /var/log -type f
```

-type f → files only

-type d → directories

4. Find files bigger than a size

```
find / -size +100M
```

Finds files larger than **100 MB**.

5. Find files modified in last 2 days

```
find /home -mtime -2
```

-mtime -2 → modified within last 2 days

6. Find and delete

```
find /tmp -type f -name "*.log" -delete
```

Deletes .log files in /tmp.

When to use

- Locating misplaced files
 - Cleaning up old/large files
 - Searching logs or configs quickly
 - Combining with other commands for automation
-

```
: df -h
```

The **df -h** command in Linux is used to **display the disk space usage** of all mounted file systems in a **human-readable format**.

Syntax

```
df -h
```

- **df** → Disk Free (shows available and used space)
 - **-h** → Human-readable (sizes in KB, MB, GB instead of blocks)
-

Output Example

Filesystem	Size	Used	Avail	Use%	Mounted on
/dev/sda1	50G	20G	27G	43%	/
tmpfs	1.9G	2.0M	1.9G	1%	/dev/shm
/dev/sdb1	100G	60G	36G	63%	/mnt/data

Columns Explained

1. **Filesystem** → Device or partition name
 2. **Size** → Total size of the filesystem
 3. **Used** → How much space is currently used
 4. **Avail** → How much space is available for use
 5. **Use%** → Percentage of space used
 6. **Mounted on** → The directory where the filesystem is mounted
-

Why and Where to Use

- Check **disk space before installing software**
 - Identify **full partitions** to clean up
 - Monitor **server storage** in real time
 - Useful for **troubleshooting** low-space issues
-

du

The **du** command in Linux is used to **estimate and display disk usage** of files and directories.

Syntax

du [options] [path]

If no path is given, it shows usage for the current directory.

Common Options

Option	Meaning
-h	Human-readable (KB, MB, GB)
-s	Summary only (total size)
-a	Show sizes for files and directories
--max-depth=N	Limit depth of recursion
-c	Show a grand total at the end

Examples

1 Show size of current directory (human-readable)

du -h

2 Show only total size of a folder

du -sh /var/log

Example output:

120M /var/log

3 Show size of all subdirectories (1 level deep)

du -h --max-depth=1 /home

4 Show size for files too

du -ah /etc

du vs df

du

df

Shows disk **usage** by files/directories Shows **overall filesystem** free/used space

Works at file level

Works at partition level

Slower for big directories

Faster because it checks filesystem metadata

Where It's Used

- Finding large folders/files consuming space
 - Troubleshooting **low disk space**
 - Pre-cleanup analysis before deleting files
 - Monitoring storage usage in projects or logs
-

If you want, I can make an **EL10-style diagram** showing how du works internally and how it differs from df visually. That would make it easier to remember.

You said:

yes

ChatGPT said:

You've hit the free plan limit for image generations requests. You can create more images when the limit resets in 4 hours and 0 minutes.

:ps -a expalin

ps -a Command

The **ps** command is used to **display information about running processes** in Linux. When you add the **-a** option, it **shows processes for all users** that are attached to a terminal (except session leaders).

Syntax

```
ps -a
```

What ps -a Shows

- **PID** → Process ID (unique number for each running process)
- **TTY** → Terminal from which the process is running
- **TIME** → Amount of CPU time the process has used
- **CMD** → The command that started the process

Example output:

```
PID TTY      TIME CMD
1234 pts/0    00:00:01 bash
1256 pts/0    00:00:05 top
1289 pts/1    00:00:00 vim
```

Common Variations

Command Purpose

ps	Shows processes for the current shell
ps -a	Shows all processes attached to terminals (not daemons)
ps -u	Shows processes for a specific user
ps -x	Includes processes not attached to a terminal
ps -aux	Full system process list (popular combo)

Where It's Used

- Checking which processes are running for all users

- Troubleshooting why a terminal is busy
 - Identifying interactive processes (vs. background services)
 - Debugging apps you started in a terminal
-

If you want, I can also give you **ps -a vs ps -aux** comparison so you'll know when to use each. That's a common interview question in Linux basics.

: env command can explain

Here's the **EL10 explanation** for the **env** command in Linux:

env Command

The **env** command is used to **display, set, or run programs in a modified environment**.

In simpler terms: it shows the environment variables of your current shell or runs a command with a temporary set of environment variables.

1. Display Environment Variables

When run without options:

env

You'll see output like:

SHELL=/bin/bash

USER=suneel

HOME=/home/suneel

PATH=/usr/local/bin:/usr/bin:/bin

LANG=en_US.UTF-8

Each line is **KEY=VALUE** format.

2. Set Temporary Environment Variables

You can set a variable for a command without permanently changing your shell environment:

`env VAR1=value command`

Example:

`env DEBUG=true python app.py`

- Only python app.py will see DEBUG=true,
 - It won't affect the shell after the command finishes.
-

3. Clear All Environment Variables

`env -i`

This runs with an **empty environment**.

Common Use Cases

Use Case	Example
View current environment variables	<code>env</code>
Run command with a specific variable	<code>env PORT=8080 node server.js</code>
Clear environment & run command	<code>env -i bash</code>
Debug environment issues	<code>` env</code>

Why It's Useful

- Debugging application behavior that depends on environment variables
 - Testing programs in a clean environment
 - Temporarily overriding system variables without permanent changes
 - Inspecting variables passed to scripts in CI/CD pipelines
-

If you want, I can also give you a **list of the most important Linux environment variables** (like PATH, HOME, USER, SHELL, etc.) that are often asked in interviews.
Do you want me to prepare that list?

You said:

yes

Here's the **EL10 list of the most important Linux environment variables** you should know—useful for interviews and daily work.

◆ Common & Important Linux Environment Variables

Variable	Example Value	Purpose
PATH	/usr/local/bin:/usr/bin:/bin	List of directories searched when running commands.
HOME	/home/suneel	User's home directory.
USER	suneel	Logged-in username.
SHELL	/bin/bash	Default shell for the user.
PWD	/home/suneel/projects	Current working directory.
LANG	en_US.UTF-8	System language and encoding.
LOGNAME	suneel	User's login name.
MAIL	/var/spool/mail/suneel	Path to the user's mailbox.
EDITOR	vim	Default text editor for the system.
HISTSIZE	1000	Number of commands stored in shell history.
TERM	xterm-256color	Terminal type (affects color & formatting).
PS1	[\u@\h \W]\\$	Primary shell prompt format.
TZ	Asia/Kolkata	Timezone setting.
OLDPWD	/home/suneel	Previous working directory (used with cd -).
SHLVL	1	Shell nesting level (increases when you open sub-shells).
-	/usr/bin/env	Last executed command.

◆ How to View Them

env

OR

```
printenv  
# OR view a single variable  
echo $PATH
```

◆ How to Set Them Temporarily

```
export MY_VAR="Hello World"  
(This lasts only for the current shell session.)
```

◆ How to Set Them Permanently

Edit `~/.bashrc` or `~/.bash_profile`:

```
export MY_VAR="Hello World"
```

Then reload:

```
source ~/.bashrc
```

If you want, I can also make you a **full “Environment Variables Cheat Sheet” with examples for each variable** so you can keep it as a reference for interviews. That would make remembering them much easier.

Do you want me to prepare that cheat sheet next?

You said:

EL10: Sleep command what why exapin

ChatGPT said:

Here's the **EL10 explanation of the sleep command** in Linux — simple but very useful in scripting and automation.

Sleep Command in Linux

The **sleep** command pauses (delays) the execution of the next command for a specified amount of time.

Syntax

`sleep NUMBER[SUFFIX]`

- **NUMBER** → the amount of time to pause.
 - **SUFFIX** → unit of time:
 - `s` → seconds (default, can be omitted)
 - `m` → minutes
 - `h` → hours
 - `d` → days
-

Examples

1. Sleep for 5 seconds

`sleep 5`

(Waits for 5 seconds before proceeding.)

2. Sleep for 2 minutes

`sleep 2m`

(Waits for 2 minutes.)

3. Sleep for 1.5 seconds

`sleep 1.5`

(Supports decimal values for seconds.)

4. Using in scripts

```
echo "Starting..."
```

```
sleep 3
```

```
echo "Done!"
```

Output:

Starting...

(wait 3 seconds)

Done!

5. Multiple sleeps in a loop

```
for i in {1..5}  
do  
    echo "Iteration $i"  
    sleep 2  
done
```

(Pauses for 2 seconds between each iteration.)

📌 Why & Where We Use sleep

Use Case	Example
Delay in scripts	Waiting before retrying a command.
Rate-limiting tasks	Prevent sending too many requests quickly.
Simulating process time	Demo/testing long-running processes.
Coordinating with other systems	Wait for a service to start before executing next step.

:Wget and curl command diffence

ChatGPT said:

Here's the **EL10-level explanation of the difference between wget and curl in Linux.**

📌 wget vs curl — What's the Difference?

Both **wget** and **curl** are command-line tools used to transfer data over the internet, but they have different **focus, features, and use cases**.

1. Basic Purpose

Command Main Purpose

wget Primarily used to **download files** from the web.

curl General-purpose tool to **transfer data** (upload & download) between a client and a server.

2. Syntax

wget

wget [URL]

curl

curl [options] [URL]

3. Key Differences

Feature	wget	curl
Download files	<input checked="" type="checkbox"/> Yes (main function)	<input checked="" type="checkbox"/> Yes (with -O or -o)
Upload files	<input type="checkbox"/> No	<input checked="" type="checkbox"/> Yes (-T, -F)
Supported Protocols	HTTP, HTTPS, FTP	20+ protocols (HTTP, HTTPS, FTP, SCP, SFTP, SMTP, etc.)
Recursive Download	<input checked="" type="checkbox"/> Yes (-r)	<input type="checkbox"/> No
Resume Downloads	<input checked="" type="checkbox"/> Yes (-c)	<input checked="" type="checkbox"/> Yes (-C -)
Scripting/Automation	Limited	Very powerful, handles APIs easily
Follow Redirects	<input checked="" type="checkbox"/> Yes (default)	<input checked="" type="checkbox"/> Yes (with -L)
Lightweight Syntax	Easier for basic downloading	More options, more complex

4. Examples

Download a file with wget

wget https://example.com/file.zip

Download a file with curl

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

Resume a download with wget

```
wget -c https://example.com/file.zip
```

Resume a download with curl

```
curl -C - -O https://example.com/file.zip
```

Upload a file using curl

```
curl -T localfile.txt ftp://ftp.example.com/
```

(You can't do this with wget.)

Use curl with an API (GET request)

```
curl -X GET https://api.example.com/data
```

5. Quick Summary

- Use **wget** → when you need to **quickly download** files or websites, especially with recursion.
 - Use **curl** → when you need **flexibility**, like working with APIs, different protocols, or uploading files.
-

: Ssh can explain

-level explanation of ssh in Linux:

What is SSH?

SSH stands for **Secure Shell**.

It's a protocol and command used to securely connect to another computer (usually a remote server) over a network.

It encrypts the communication, so **nobody can read or modify the data** in transit.

Why We Use SSH

- **Secure Login** to remote systems

- **Run Commands** on a remote server
 - **Transfer Files** securely (with scp or sftp)
 - **Manage servers** from anywhere
-

Syntax

ssh [username]@[hostname or IP]

Example

Connecting to a remote server:

ssh ubuntu@192.168.1.100

- ssh → Command
 - ubuntu → Remote username
 - 192.168.1.100 → Server IP address
-

How It Works

1. You type the **ssh** command.
 2. SSH asks for the **password** (or uses a stored key for authentication).
 3. Once authenticated, you get a **remote shell** as if you are sitting at that server.
-

Common Options

Option Meaning

- p Specify port (default is 22)
 - i Use a specific private key file
 - v Verbose output (debugging)
 - X Enable X11 forwarding (GUI apps)
-

◆ Examples

Connect using a different port

```
ssh -p 2222 user@server.com
```

Connect with a private key

```
ssh -i ~/.ssh/id_rsa user@server.com
```

Run a single command on remote server

```
ssh user@server.com "ls -l /var/www"
```

Interview Tip

If they ask:

How is SSH more secure than Telnet?

You say:

- SSH encrypts data, including passwords.
 - Telnet sends everything in **plain text**, making it vulnerable.
-

scp command where can use wht is this command

ChatGPT said:

Here's your **EL10-level explanation of scp** in Linux:

What is scp?

scp stands for **Secure Copy**.

It's a command to **copy files and directories between local and remote systems** over SSH.

It uses **SSH encryption** to keep the data transfer secure.

❖ Where We Can Use scp

- Upload files **from local to remote server**
 - Download files **from remote server to local**
 - Copy files **between two remote servers** (from your local machine)
-

Syntax

scp [options] source destination

Examples

1 Copy a local file to a remote server

scp file.txt user@192.168.1.100:/home/user/

2 Copy from remote to local

scp user@192.168.1.100:/home/user/file.txt /local/path/

3 Copy an entire folder

scp -r myfolder user@192.168.1.100:/home/user/

(-r = recursive, for directories)

4 Copy between two remote servers

scp user1@server1:/path/file.txt user2@server2:/path/

Common Options

Option Purpose

- P Specify SSH port
 - i Use identity (private key) file
 - r Copy directory recursively
 - v Verbose mode (debugging)
-

◆ Key Points

- scp uses **SSH** for authentication and encryption.
 - Good for **one-time transfers**.
 - For frequent syncs, use rsync.
-

Interview Tip

If they ask:

How is scp different from cp?

- cp → Copies files locally, no network involved.
 - scp → Copies files securely over a network using SSH.
-

You've hit the free plan limit for image generation requests. You can create more images when the limit resets in 3 hours and 40 minutes.

If you want, I can still **make a text-based diagram** showing how scp works so you can visualize it without an image. Would you like me to do that?

crond tab exapin what why where can use

In Linux, **crontab** is used to schedule tasks (called *cron jobs*) that run automatically at specific times or intervals.

It's like telling your system: "*Hey, run this command/script every day at 5 PM*" — and it never forgets.

1. What is crontab?

- **cron** → A background service (daemon) that runs scheduled jobs.
 - **crontab** → The configuration file that tells cron *what* to run and *when*.
 - Each user (including root) can have their own crontab.
-

2. Why use it?

You use crontab when you want tasks to run **automatically** without manual execution, such as:

- Automated backups
 - Sending reports daily
 - Cleaning temporary files
 - Monitoring scripts
 - Syncing files between servers
-

3. Where can we use it?

- **System Maintenance** → Example: Clear logs every week
 - **DevOps** → Trigger CI/CD steps at fixed times
 - **Data Processing** → Run ETL jobs every night
 - **Monitoring** → Send email alerts regularly
 - **Custom scripts** → Any script you want to repeat automatically
-

4. Syntax of crontab

A crontab entry looks like this:

```
* * * * * command_to_execute  
| | | | |  
| | | | | Day of week (0-6) [0=Sunday]  
| | | | | Month (1-12)  
| | | | | Day of month (1-31)  
| | | | | Hour (0-23)  
| | | | | Minute (0-59)
```

5. Examples

1. Run a script every day at 5 PM:

```
0 17 * * * /home/user/myscript.sh
```

2. Run every Monday at 9 AM:

```
0 9 * * 1 /home/user/weekly_report.sh
```

3. Run every 15 minutes:

```
*/15 * * * * /home/user/check_status.sh
```

6. Useful Commands

Command	Purpose
crontab -e	Edit your crontab

Command	Purpose
crontab -l	List your crontab
crontab -r	Remove your crontab
service cron start	Start cron service (varies by distro)

If you want, I can also give you a **simple diagram** showing how cron and crontab interact inside Linux so it's easier to visualize.

Do you want me to make that for you in text form?

You said:

EL10: sed command also exapin

ChatGPT said:

Alright — let's break down the **sed** command in Linux in a simple, interview-friendly way.

1. What is sed?

sed stands for **Stream Editor**.

It's used to **search, find, replace, insert, or delete** text in a file or data stream — without opening it in an editor.

Think of it like a "text processor robot" that edits on the fly.

2. Why use it?

- Automating repetitive text changes
 - Editing large files quickly
 - Modifying logs or config files without opening them manually
 - Useful in shell scripts for data cleanup
-

3. Where can we use it?

- **DevOps** → Update configuration files during deployments
- **Data Processing** → Clean CSV or log data
- **Scripting** → Search & replace without human interaction

- **System Admin** → Change file permissions or paths in bulk scripts
-

4. Basic Syntax

```
sed [options] 'command' filename
```

5. Common Examples

(A) Replace text

```
sed 's/old_text/new_text/' file.txt
```

- **s** → substitute
- Replaces **first occurrence** of old_text in each line.

Replace **all occurrences**:

```
sed 's/old_text/new_text/g' file.txt
```

(B) Delete lines

```
sed '3d' file.txt # Deletes line 3
```

```
sed '2,5d' file.txt # Deletes lines 2 to 5
```

(C) Print specific lines

```
sed -n '2,4p' file.txt
```

- **-n** → suppress normal output
 - **p** → print only the specified lines
-

(D) Edit file in place

```
sed -i 's/localhost/127.0.0.1/g' config.txt
```

- **-i** → directly edits the file (no need to redirect output)
-

6. Quick Reference Table

Task	Command Example
Replace word	sed 's/foo/bar/' file
Replace all	sed 's/foo/bar/g' file
Delete line 5	sed '5d' file
Show only lines 3-7	sed -n '3,7p' file
In-place edit	sed -i 's/a/b/g' file

netstat command can explain

ChatGPT said:

Alright — let's go step-by-step with the **netstat** command.

1. What is netstat?

netstat (**Network Statistics**) is a command-line tool used to display **network connections, routing tables, interface statistics, and listening ports**. It's like the **X-ray for your network activity**.

2. Why use it?

- To troubleshoot **network connectivity**
 - To find **open ports** on a system
 - To check which process is using a port
 - To monitor **incoming and outgoing connections**
-

3. Where can we use it?

- **System Administration** → Check which ports are open and secure them
 - **DevOps** → Verify if an application is listening on the correct port
 - **Security** → Detect suspicious network activity
 - **Troubleshooting** → Diagnose server connection issues
-

4. Basic Syntax

netstat [options]

5. Common Options & Examples

(A) Show all connections

netstat -a

- Displays **all active TCP and UDP connections** and listening ports.
-

(B) Show listening ports only

netstat -l

(C) Show process using the port

netstat -tulpn

- t → TCP
 - u → UDP
 - l → listening ports
 - p → show process ID/program name
 - n → show numeric addresses
-

(D) Show routing table

netstat -r

(E) Show interface statistics

netstat -i

6. Quick Reference Table

Option Purpose

- a All connections & listening ports
 - l Only listening ports
 - t TCP connections
 - u UDP connections
 - p Process & PID using the port
 - n Numeric output (no DNS resolution)
 - r Routing table
 - i Interface statistics
-

 **Note:** On newer Linux systems, netstat is deprecated — replaced by:

`ss -tulpn`

It's faster and provides similar information.

NSlookup command can explain

Alright — let's break down **nslookup** so it's crystal clear.

1. What is nslookup?

nslookup (Name Server Lookup) is a network command-line tool used to **query DNS servers** and get details about a **domain name or IP address**.

Think of it like a **phonebook lookup** for the internet:

- You give it a domain → it gives you the IP address.
 - You give it an IP → it gives you the domain (reverse lookup).
-

2. Why use it?

- To **check DNS records** for troubleshooting
- To **verify domain-to-IP mapping**

- To diagnose **DNS resolution issues**
 - To get details of **mail servers (MX records)**, **name servers (NS records)**, etc.
-

3. Where can we use it?

- **System admins** → Check if a domain resolves to the correct server
 - **DevOps** → Verify DNS after deploying services
 - **Security** → Check suspicious domains
 - **Networking** → Troubleshoot DNS problems
-

4. Basic Syntax

nslookup [hostname or IP] [DNS server]

5. Common Examples

(A) Basic lookup

nslookup google.com

Output:

- Shows the IP address(es) for google.com.
-

(B) Reverse lookup (IP → Domain)

nslookup 8.8.8.8

- Finds the domain name for an IP address.
-

(C) Use a specific DNS server

nslookup example.com 8.8.8.8

- Queries Google's public DNS server.
-

(D) Interactive mode

nslookup

Then you can type commands like:

```
> set type=MX
```

```
> example.com
```

- Shows mail exchange (MX) records for the domain.
-

6. Common DNS Record Types with nslookup

Record Type Purpose

A IPv4 address of the domain

AAAA IPv6 address of the domain

MX Mail server records

NS Name server records

CNAME Canonical name (alias)

PTR Reverse lookup record

Tip for Interviews:

If DNS isn't working, first check with:

```
nslookup domain.com
```

If it works with 8.8.8.8 but not your local DNS, the issue is with your DNS server, not the domain.

temporarymount and Permentmount command explain

ChatGPT said:

Alright — let's break down **Temporary Mount** vs **Permanent Mount** in Linux so you can remember it.

1. What is Mounting?

In Linux, **mounting** means making a **storage device** (like USB, hard drive, or network share) available to the OS so you can access files on it.

Example devices:

- `/dev/sdb1` → USB drive partition
 - `/dev/xvdf` → AWS EBS volume
-

2. Temporary Mount

A **temporary mount** is when you mount a storage device manually.

It **only lasts until reboot** — after restart, you need to mount it again.

Command:

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

Here:

- `/dev/sdb1` → the device
- `/mnt` → the folder where it will be accessible

When to use:

- Testing new storage
 - Accessing a drive temporarily
 - Quick troubleshooting
-

3. Permanent Mount

A **permanent mount** is when you configure the system to **automatically mount a device at boot**.

This is done by editing the **/etc/fstab** file.

Steps:

1. Find the device UUID:

```
blkid
```

Example output:

```
UUID="1234-ABCD" TYPE="ext4"
```

2. Edit `/etc/fstab`:

```
sudo nano /etc/fstab
```

Add a line:

```
UUID=1234-ABCD /mnt ext4 defaults 0 0
```

3. Apply changes without reboot:

```
sudo mount -a
```

When to use:

- Servers that need storage available on every boot
 - AWS EC2 with EBS volumes
 - Network storage like NFS
-

4. Key Differences Table

Feature	Temporary Mount	Permanent Mount
Setup	mount command	/etc/fstab
Lasts Until	Reboot	Persistent
Use Case	Testing / Temporary access	Production use
Ease of Use	Quick	Requires config

💡 Interview Tip:

If they ask: "You mounted a disk, but after reboot it's gone — why?" → Answer: Because it was a temporary mount. Need to add it in /etc/fstab for permanent mounting.

If you want, I can make a **simple diagram** showing how both mount types work in Linux for EL10 revision. That way it will stick in your mind.

Do you want me to create it?

You said:

EL10: telnet command

ChatGPT said:

Alright — let's go through the **telnet command** for EL10 in a clear way.

1. What is telnet?

telnet is a **network protocol** and a **command** used to connect to another machine over TCP, usually for testing or remote login.

- It works over **port 23** by default.
 - It's **unencrypted** (all data is sent as plain text, including passwords).
 - Mostly replaced by **SSH** for security reasons.
-

2. Why do we use telnet now?

These days, telnet is **not used for secure remote login**, but it is **still used for troubleshooting** network connections.

Example use cases:

- Check if a remote host and port are reachable.
 - Debug services like SMTP (email), HTTP, or databases by manually sending commands.
-

3. Syntax

telnet <hostname> <port>

Example 1 – Test a website on port 80:

telnet google.com 80

If it connects, the port is open. If not, it will show a connection error.

Example 2 – Test if SMTP server is reachable:

telnet mail.example.com 25

You can then type SMTP commands manually.

4. Where to use telnet

- **Network connectivity testing** (like ping, but for specific ports).
 - **Troubleshooting firewalls** — to see if a port is blocked.
 - **Testing custom applications** listening on specific ports.
-

5. Why is it less used for login?

Because:

- No encryption → passwords visible in network traffic.
- Vulnerable to man-in-the-middle attacks.

Instead → Use **ssh** for secure remote access.

6. Quick Example

Check if MySQL port is open

telnet 192.168.1.10 3306

Check if a website port is open

telnet example.com 443

If it says Connected, the service is reachable.

If it says Connection refused or Timeout, it's blocked or down.

: systemctl and service commands can explain

ChatGPT said:

Alright — let's break down **systemctl** and **service** commands for EL10 in a way that's easy to remember.

1. What are they?

Both systemctl and service are used to **manage services** (start, stop, restart, check status) in Linux — but they belong to **different init systems**.

Feature	systemctl	service
Works With	systemd (newer init system)	SysVinit (older init system)
Available On	Modern distros (CentOS 7+, RHEL 7+, Ubuntu 16+)	Older distros or as compatibility tool
Controls	Units (services, timers, targets, etc.)	Only services

Feature	<code>systemctl</code>	<code>service</code>
Extra Features	Can enable/disable services at boot, view logs	Basic start/stop/restart/status only

2. Why two commands exist

- **Before:** Linux used **SysVinit** → managed with service.
 - **Now:** Most distros use **systemd** → managed with systemctl.
 - service still exists for backward compatibility, but under the hood it calls systemctl on modern systems.
-

3. Common Usage

Using systemctl:

```
# Start service
```

```
sudo systemctl start nginx
```

```
# Stop service
```

```
sudo systemctl stop nginx
```

```
# Restart service
```

```
sudo systemctl restart nginx
```

```
# Check status
```

```
sudo systemctl status nginx
```

```
# Enable service at boot
```

```
sudo systemctl enable nginx
```

```
# Disable service at boot
```

```
sudo systemctl disable nginx
```

Using service:

```
# Start service
```

```
sudo service nginx start
```

```
# Stop service
```

```
sudo service nginx stop
```

```
# Restart service
```

```
sudo service nginx restart
```

```
# Check status
```

```
sudo service nginx status
```

4. Key Difference

- service works only with **services**.
 - systemctl works with **all systemd units** (services, mount points, timers, sockets, etc.).
 - systemctl can **enable/disable services** to start automatically at boot — service cannot.
-

5. Example

Q: How do you start the Apache HTTP server at boot in CentOS 8?

A:

```
sudo systemctl enable httpd
```

```
sudo systemctl start httpd
```
