**🧭 What is a Path in Linux?**

A **path** is the **location of a file or directory** in the Linux file system.  
There are two types:

**1️ Absolute Path 🛤️**

It starts from the **root directory /** and shows the **complete location** of a file or folder, no matter where you are currently.

**📌 Format:**

/parent\_folder/sub\_folder/filename

**✅ Example:**

cd /home/koushik/projects/devops/

* This path **starts from / (root)**.
* It will **always go to the exact location**, no matter where you are.

**2️ Relative Path 🧭**

It is **relative to your current working directory**.  
You don’t start from /, you just give the path based on **where you are now**.

**📌 Format:**

folder\_name/

**✅ Examples:**

* cd logs/ → Go into logs directory inside your current directory
* cd ../ → Go **one level up**
* cd ../../configs → Go up 2 levels, then into configs

**🔍 Key Differences:**

| **Feature** | **Absolute Path** | **Relative Path** |
| --- | --- | --- |
| Starts from | Root / | Current directory (. or no symbol) |
| Always valid | ✅ Yes | ❌ Depends on where you are |
| Begins with / | ✅ Yes | ❌ No |
| Used in scripts | Preferred for accuracy | Used for short paths inside same folder |
| Example | /home/koushik/devops/file.txt | ../devops/file.txt |

**🧑‍💼 🎯 Interview-Ready Answer:**

"An **absolute path** is the full path that starts from the root /, like /home/user/file.txt. A **relative path** is based on the current location, like ../folder/. DevOps engineers use both depending on whether the script needs a fixed or dynamic path."

**👤 1. Normal User (Regular User)**

A normal user is a **non-administrative user** created to perform limited tasks like reading files, running apps, or editing their own files.

**✅ Characteristics:**

* Limited **permissions** (cannot modify system files or configurations).
* Has a **home directory** (e.g., /home/koushik).
* Cannot install packages or restart services without sudo.
* Safer to use for daily tasks (avoids accidental system changes).

**🔍 Example:**

koushik@ubuntu:~$ whoami

koushik

**👑 2. Root User (Superuser)**

The **root user** is the **administrator** of the system.  
They have **full control** over everything — they can install, delete, modify any file, user, or service.

**✅ Characteristics:**

* Full **read/write/execute** permissions.
* Can run **all commands** without restrictions.
* **Home directory** is usually /root
* Dangerous if misused — a wrong command can crash the system.

**🔍 Example:**

root@ubuntu:~# whoami

root

**🔐 Switching Between Users:**

| **Command** | **Description** |
| --- | --- |
| su | Switch user (to root if no username) |
| sudo | Run a command with root privileges |
| sudo su | Become root temporarily |
| exit | Go back to normal user from root |

**🧑‍💼 🎯 Interview-Ready Answer:**

"In Linux, a **normal user** has limited access and can only work in their own space. The **root user**, or superuser, has full control of the system and can make administrative changes. We use sudo to safely perform root tasks without logging in as root directly."

**🛡️ What is sudo in Linux?**

sudo stands for **“Superuser Do”**.  
It allows a **normal user** to run commands with **root (superuser) privileges** without logging in as the root user.

**📋 Syntax:**

sudo [command]

**🧪 Examples:**

| **Command** | **Meaning** |
| --- | --- |
| sudo apt update | Update package list (requires root) |
| sudo reboot | Restart the system |
| sudo mkdir /var/log/test | Create a folder where only root has permission |
| sudo useradd devuser | Add a new user (admin task) |
| sudo rm -rf /foldername | Force delete system folder (be careful ⚠️) |

**⚠️ Best Practices:**

* Only use sudo when needed — it prevents accidental damage.
* Always **double-check commands** before running with sudo.
* Use sudo -k to clear the sudo timestamp (forces password next time).

Great! Let’s understand sudo su clearly:

**🔄 What does sudo su mean?**

✅ It combines two powerful commands:

* **sudo** – Run a command with superuser privileges
* **su (substitute user)** – Switch to another user (default is root)

**💡 So, sudo su means:**

👉 “Run the su command as **root** using sudo permissions,”  
which effectively **switches you to the root user shell**.

**📌 When do you use it?**

* When you need to become the **root user** to perform multiple admin tasks
* Useful for **interactive sessions** where you're running several commands as root

**⚠️ Example:**

sudo su

🧑‍💻 Output: Your prompt changes to # instead of $, indicating **you are now root**.

Now you can run any command **without typing sudo again** for each one.

**🚪 To Exit Root:**

exit

**🔐 Tip for DevOps/Best Practice:**

Using sudo su gives **full root access**, which is **not always recommended**.  
It's safer to use sudo for **individual commands** unless absolutely needed.

**🔍 What does sudo su - do?**

sudo su - means:

**"Use sudo to run su as root, and start a login shell for the root user."**

So, it:

* Switches to the **root user** (su)
* Loads the **root user's environment variables**, path, and settings (-)

**✅ Difference between sudo su and sudo su -:**

| **Command** | **What it does** |
| --- | --- |
| sudo su | Switches to root **without** loading root's environment |
| sudo su - | Switches to root **and loads** root’s full login environment |

**🧪 Example:**

sudo su -

You'll now be:

* In root’s home directory (/root)
* Using root’s environment and PATH
* Seeing # in your shell prompt

**🔐 Why use sudo su -?**

✅ Best when:

* You’re doing **administrative tasks** that rely on **root's environment**
* You want to **simulate logging in directly as root**

**🚪 To exit:**

exit

**📁 What is** ls **Command?**

ls stands for **“list”**.  
It is used to **list files and directories** in the current directory or a specified path.

**📌 Basic Syntax:**

ls [options] [directory or file path]

**🔍 Basic Examples:**

| **Command** | **Description** |
| --- | --- |
| ls | Lists files/folders in current directory |
| ls /home/koushik | Lists contents of that path |
| ls Documents | Lists files inside Documents folder |

**✅ Most Commonly Used ls Options:**

| **Option** | **Meaning** |
| --- | --- |
| -l | Long listing format (permissions, owner, date, size) |
| -a | Shows all files including hidden (. files) |
| -h | Human readable size (used with -l) like KB, MB |
| -R | Recursively lists subdirectories too |
| -t | Sort by modification time (latest first) |
| -r | Reverse order of sorting |
| -S | Sort by file size |
| -i | Show inode number of files |
| -d | List only the directory name, not contents |

**🧪 Examples of Combined Options:**

ls -l # Detailed info with size, date, owner, etc.

ls -la # Includes hidden files

ls -lh # Human-readable sizes

ls -ltr # Sorted by time (oldest first)

ls -lhS # Sort by size (largest to smallest)

**📦 Real-Time DevOps Usage:**

| **Use Case** | **Command** |
| --- | --- |
| Check logs folder size/info | ls -lh /var/log |
| View hidden config files | ls -a ~ |
| Explore recursive app folders | ls -R /opt/tomcat |
| See file modified recently | ls -lt /etc/nginx/ |

**📌 What is touch Command?**

touch is used to **create empty files** or **update the timestamp** (access and modification time) of existing files.

**🧠 Basic Syntax:**

touch [options] filename

**✅ Common Use Cases:**

| **Command** | **Description** |
| --- | --- |
| touch file.txt | Creates a new empty file named file.txt |
| touch a.txt b.txt | Creates multiple files at once |
| touch log.txt | If file exists, updates its timestamp |

**🧪 Real-Time Examples:**

touch hello.txt # Creates 'hello.txt'

touch devops1.txt devops2.txt # Creates two files

touch /tmp/mylog.txt # Creates file in another folder

**🔁 Timestamp Update:**

If the file **already exists**, touch updates:

* **Access time (atime)** – last read time
* **Modification time (mtime)** – last content change

**⚙️ Useful Options:**

| **Option** | **Description** |
| --- | --- |
| -c | Do not create a file if it doesn’t exist |
| -t | Set a specific timestamp (YYYYMMDDhhmm) |
| -r file1 | Use timestamp of another file |
| -a | Update only access time |
| -m | Update only modification time |

**🔧 Examples:**

touch -c test.txt # Only update if exists, else do nothing

touch -t 202504290915 my.txt # Set specific date/time

touch -r old.txt new.txt # Set new.txt's time same as old.txt

**🔐 Real-time DevOps Use Cases:**

| **Scenario** | **Command** |
| --- | --- |
| Creating test or log files | touch /var/log/test.log |
| Timestamp syncing for deployment | touch -r lastbuild.txt app.js |
| Script creating temp files | touch tempfile.txt |

**🐱 What is cat Command?**

cat stands for **“concatenate”**.  
It is used to **display the contents of a file**, **create a file**, or **combine multiple files** into one.

**📌 Basic Syntax:**

cat [options] [filename]

**✅ Common Use Cases:**

| **Command** | **Description** |
| --- | --- |
| cat file.txt | Shows contents of file.txt |
| cat > newfile.txt | Creates a new file, takes input from user |
| cat file1 file2 | Displays contents of both files |
| cat file1 > file2 | Copies content from file1 to file2 |
| cat file1 >> file2 | Appends content of file1 to file2 |

**✍️ Example: Create a file**

cat > demo.txt

Then type your content and press Ctrl + D to save.

**👓 Example: View a file**

cat demo.txt

**➕ Example: Append to an existing file**

cat >> demo.txt

**🔍 Useful Options:**

| **Option** | **Description** |
| --- | --- |
| -n | Show line numbers |
| -b | Show line numbers (skip blank lines) |
| -E | Show $ at end of each line |
| -s | Remove extra blank lines |
| -T | Show tab characters as ^I |

**💡 Examples with Options:**

cat -n file.txt # Show contents with line numbers

cat -b file.txt # Skip blank lines while numbering

cat -E file.txt # See line ends with $

**💼 Real-Time DevOps Usage:**

| **Task** | **Command** |
| --- | --- |
| View logs quickly | cat /var/log/syslog |
| Combine two config files | cat a.conf b.conf > final.conf |
| Check file created by script | cat output.txt |

**📁 What is mkdir?**

mkdir stands for **"make directory"**.  
It is used to **create new directories (folders)** in Linux.

**📌 Basic Syntax:**

mkdir [options] directory\_name

**✅ Common Use Cases:**

| **Command** | **Description** |
| --- | --- |
| mkdir devops | Creates a directory named devops |
| mkdir test1 test2 | Creates multiple folders at once |
| mkdir /tmp/logs | Creates logs folder in /tmp |

**⚙️ Useful Options:**

| **Option** | **Description** |
| --- | --- |
| -p | Create parent directories if they don't exist |
| -v | Show message for each directory created (verbose) |
| -m | Set permissions while creating directory (mode) |

**🧪 Real-Time Examples:**

mkdir projects # Creates 'projects' folder

mkdir dev logs backups # Creates 3 folders

mkdir -p app/logs/error # Creates full path, even if intermediate folders missing

mkdir -v newfolder # Shows message: mkdir: created directory 'newfolder'

mkdir -m 755 newdir # Creates directory with permission 755

**💼 Real-Time DevOps Usage:**

| **Task** | **Command** |
| --- | --- |
| Create log folders for app | mkdir -p /var/logs/myapp/errors |
| Organize backups or deployments | mkdir /backup/2025-April |
| Script-based directory creation | mkdir -p $HOME/scripts/output |

**🔐 Notes:**

* If the folder already exists without -p, it shows an error.
* mkdir needs **write permissions** in the parent directory.

**🗑️ What is rmdir?**

rmdir stands for **"remove directory"**.  
It is used to **delete empty directories only** in Linux.

**📌 Basic Syntax:**

rmdir [options] directory\_name

**⚠️ Important Note:**

* rmdir **only removes empty folders**.
* If the directory has files or subfolders, it **won’t delete it**.

**✅ Common Use Cases:**

| **Command** | **Description** |
| --- | --- |
| rmdir demo | Deletes the folder demo if it's empty |
| rmdir test1 test2 | Deletes multiple empty folders |
| rmdir -p a/b/c | Removes directory c, then b, then a if all are empty |

**🧪 Examples:**

mkdir testdir

rmdir testdir # Works as it's empty

mkdir -p project/code

rmdir -p project/code # Removes 'code', then 'project' if both are empty

**🔥 Real-Time DevOps Usage:**

| **Use Case** | **Command** |
| --- | --- |
| Remove temp empty folders | rmdir /tmp/oldbuild/emptyfolder |
| Clean up unused empty dirs | rmdir -p /opt/tools/old/utils |

**🗑️ What is rm?**

rm stands for **“remove”**.  
It is used to **delete files and directories** in Linux.

**📌 Basic Syntax:**

rm [options] file\_or\_directory

**✅ Common Use Cases:**

| **Command** | **Description** |
| --- | --- |
| rm file.txt | Deletes the file named file.txt |
| rm -r folder | Deletes a folder **and its contents** |
| rm -f file.txt | Force delete without confirmation |
| rm -rf folder/ | Forcefully deletes folder + all inside |

**⚠️ Be Careful!**

* **rm deletes permanently** — no Recycle Bin!
* Especially with -rf, there is **no warning**.

**🔧 Useful Options:**

| **Option** | **Description** |
| --- | --- |
| -f | Force delete (no prompt) |
| -i | Ask before every delete (interactive mode) |
| -r | Recursively delete directories and contents |
| -v | Verbose mode (shows what is being deleted) |

**🧪 Examples:**

rm report.txt # Deletes a file

rm -i config.json # Asks before deleting

rm -r logs/ # Deletes directory and all inside

rm -rf /tmp/project # Deletes without any prompt (DANGEROUS!)

rm -rv myfolder/ # Verbose recursive delete

**💼 Real-Time DevOps Usage:**

| **Scenario** | **Command** |
| --- | --- |
| Remove old logs | rm -rf /var/log/myapp/old/ |
| Delete temporary build folders | rm -rf /tmp/build/ |
| Clean unused config files | rm -i \*.conf |

**🧠 Bonus Tip: Safe Alternative**

Use trash-cli tool in Linux if you want a Recycle Bin-like feature (trash-put, trash-list).

**🧾 What is CRUD?**

**CRUD** stands for:  
**C**reate, **R**ead, **U**pdate, and **D**elete.

It refers to the **four basic operations** you can do on **data** in a database or any storage system (like files, APIs, or cloud services).

**🔠 CRUD Operations Explained:**

| **Operation** | **Action** | **Example in Real Life** |
| --- | --- | --- |
| **Create** | Add new data | Add a new user to a website |
| **Read** | View existing data | View a user's profile |
| **Update** | Change existing data | Change a user's email address |
| **Delete** | Remove existing data | Delete a user from the system |

**📂 What is cp?**

cp stands for **“copy”**.  
It is used to **copy files or directories** from one location to another in Linux.

**📌 Basic Syntax:**

cp [options] source destination

**✅ Common Use Cases:**

| **Command** | **Description** |
| --- | --- |
| cp file1.txt /tmp/ | Copy file1.txt to /tmp/ directory |
| cp file1.txt file2.txt | Copy file1.txt and rename it as file2.txt |
| cp -r folder1 /home/user/ | Copy an entire directory (folder1) |

**⚙️ Useful Options:**

| **Option** | **Description** |
| --- | --- |
| -r | Copy directories recursively |
| -i | Prompt before overwriting a file |
| -u | Copy only if the source file is newer than the destination file |
| -v | Verbose mode: show the files being copied |
| -f | Force copy (overwrites files without confirmation) |
| -a | Archive mode: preserves all attributes (recursive, symbolic links, timestamps) |

**🧪 Examples:**

cp file1.txt /tmp/ # Copy file1.txt to /tmp/ directory

cp -r folder1 /home/user/ # Copy folder1 recursively

cp -i file1.txt backup.txt # Prompt before overwriting

cp -v file1.txt /tmp/ # Copy file with verbose output

cp -u file1.txt /tmp/ # Copy file only if it's newer

cp -a folder1/ /home/user/backup/ # Archive copy (preserves attributes)

**💼 Real-Time DevOps Usage:**

| **Scenario** | **Command** |
| --- | --- |
| Copy logs to backup directory | cp -r /var/logs/ /backup/logs/ |
| Backup configuration files before changes | cp -i /etc/nginx/nginx.conf /backup/ |
| Deploy updated code files to the server | cp -r /local/code/ /var/www/html/ |

**🚨 Caution: Overwriting Files**

* Be careful when using cp with files. Without -i, it will **overwrite** files in the destination without warning.

**🌐 What is wget?**

**wget** is a command-line utility used to **download files** from the web using HTTP, HTTPS, or FTP protocols.

It is widely used for downloading files, webpages, and even entire websites.

**📌 Basic Syntax:**

wget [options] [URL]

**✅ Common Use Cases:**

| **Command** | **Description** |
| --- | --- |
| wget https://example.com/file.zip | Download file.zip from the given URL |
| wget -O newfile.txt https://example.com/file.txt | Download and save as newfile.txt |
| wget -r https://example.com/ | Recursively download an entire website |

**⚙️ Useful Options:**

| **Option** | **Description** |
| --- | --- |
| -O | Save the downloaded file with a specific name |
| -r | Download files recursively (e.g., an entire website) |
| -c | Continue downloading a file from where it was last stopped (resume) |
| -q | Quiet mode — suppress output (useful in scripts) |
| -np | No parent — avoid downloading files from the parent directory |
| -P | Specify a download directory |
| --limit-rate=200k | Limit download speed to a specific rate (e.g., 200 KB/s) |

**🧪 Examples:**

wget https://example.com/file.zip # Download a single file

wget -O downloaded\_file.zip https://example.com/file.zip # Download and rename file

wget -r https://example.com/ # Download the whole website

wget -c https://example.com/largefile.zip # Resume downloading a file

wget -q https://example.com/file.txt # Silent mode, no output

wget -P /tmp/ https://example.com/file.txt # Download to /tmp/ directory

**💼 Real-Time DevOps Usage:**

| **Scenario** | **Command** |
| --- | --- |
| Downloading application source code | wget https://example.com/app.tar.gz |
| Downloading logs from remote servers | wget -r https://example.com/logs/ |
| Backup website files | wget -r -np https://example.com/backup/ |
| Fetching updates for servers | wget -q https://updates.example.com/patches/ |

**🧑‍💼 Interview-Ready Answer:**

"wget is a Linux command-line utility used to download files from the web via HTTP, HTTPS, or FTP. It supports various options like -r for recursive downloading, -O to specify a filename, and -c to resume downloads. It's commonly used in DevOps for downloading software packages, website files, or backup data."

**🌐 What is curl?**

**curl** stands for **Client URL**.  
It is a command-line tool used for **transferring data to or from a server** using various protocols like HTTP, HTTPS, FTP, SCP, SFTP, and more.

curl is more versatile than wget and can be used for interacting with APIs, sending HTTP requests, downloading files, and more.

**📌 Basic Syntax:**

curl [options] [URL]

**✅ Common Use Cases:**

| **Command** | **Description** |
| --- | --- |
| curl https://example.com | Fetch the content of a URL using HTTP(S) |
| curl -O https://example.com/file.zip | Download a file and save it with the original filename |
| curl -I https://example.com | Get the headers of the HTTP response |
| curl -X POST https://example.com/api | Send a POST request to an API |
| curl -d "key=value" https://example.com | Send data to a server using the POST method |

**⚙️ Useful Options:**

| **Option** | **Description** |
| --- | --- |
| -O | Save the downloaded file with the same name as the remote file |
| -o | Save the downloaded content with a custom filename |
| -X | Specify the HTTP request method (GET, POST, PUT, DELETE, etc.) |
| -d | Send data in a POST request (usually used for API interactions) |
| -I | Fetch only the HTTP headers (no content) |
| -L | Follow redirects (e.g., if the URL is redirected to another URL) |
| -v | Verbose mode, show detailed information about the connection and request |
| -u | Use basic authentication (username:password) |
| -H | Send a custom header in the request |

**🧪 Examples:**

curl https://example.com # Get content from a website

curl -O https://example.com/file.zip # Download file and save it with the original name

curl -o newfile.zip https://example.com/file.zip # Download and save as 'newfile.zip'

curl -I https://example.com # Fetch only headers from the website

curl -X POST https://example.com/api -d "name=Koushik&email=koushik@example.com" # Send POST request with data

curl -u username:password https://example.com # Send request with authentication

curl -L https://example.com/redirected-url # Follow redirects until the final URL

**💼 Real-Time DevOps Usage:**

| **Scenario** | **Command** |
| --- | --- |
| Testing an API endpoint | curl -X GET https://api.example.com/users/123 |
| Downloading a configuration file | curl -O https://example.com/config.json |
| Sending form data to a server | curl -X POST https://example.com/form -d "key=val" |
| Debugging HTTP response headers | curl -I https://example.com |
| Uploading a file via FTP | curl -T file.zip ftp://example.com/upload/ |

**🧑‍💼 Interview-Ready Answer:**

"curl is a command-line tool used to transfer data between a client and a server using a variety of protocols (HTTP, HTTPS, FTP, etc.). It's widely used for interacting with web APIs, downloading files, sending data via POST requests, and even debugging HTTP responses. It offers numerous options for customizing requests, including sending headers, following redirects, and authenticating."

**🌐 Bonus: curl vs wget**

| **Feature** | **curl** | **wget** |
| --- | --- | --- |
| Protocols | HTTP, HTTPS, FTP, SCP, SFTP, and more | HTTP, HTTPS, FTP |
| Use Case | Primarily for APIs and data transfer | Primarily for downloading files |
| Redirects | Follows redirects with -L | Follows redirects by default |
| Verbosity | Detailed with -v (verbose) | Simple, uses -v for verbosity |
| POST/PUT | Supports sending data with -d and -X | No native support for POST/PUT |

**📌 What is Piping?**

**Piping** (|) allows you to **combine commands** by connecting the output of one command to the input of another. This is done using the pipe operator (|).

**📌 Basic Syntax:**

command1 | command2

* **command1** generates output (stdout).
* **command2** takes the output of command1 as its input (stdin).

**✅ Common Use Cases:**

| **Command** | **Description** |
| --- | --- |
| `ls | less` |
| `ps aux | grep process\_name` |
| `cat file.txt | wc -l` |
| `dmesg | tail -n 10` |
| `echo "Hello" | tee file.txt` |

**⚙️ Common Commands Used with Piping:**

| **Command** | **Description** |
| --- | --- |
| cat | Concatenate files and display their content. |
| grep | Search for a specific pattern in the input. |
| awk | A powerful text processing tool. |
| sort | Sort input based on certain criteria. |
| uniq | Remove duplicate lines from the input. |
| head | Display the first few lines of input. |
| tail | Display the last few lines of input. |
| tee | Split output to both a file and the terminal. |
| wc | Word count: Count lines, words, or characters. |

**🧪 Examples of Piping:**

**1. Viewing file content in a paginated manner:**

ls | less

* ls lists all files in the directory.
* less allows you to scroll through the output one page at a time.

**2. Counting the number of lines in a file:**

cat file.txt | wc -l

* cat file.txt displays the contents of file.txt.
* wc -l counts the number of lines.

**3. Searching for a process:**

ps aux | grep apache

* ps aux lists all running processes.
* grep apache filters and shows processes related to apache.

**4. Getting the last 10 system messages:**

dmesg | tail -n 10

* dmesg shows system messages.
* tail -n 10 shows the last 10 lines of output.

**5. Saving output to a file and displaying it:**

echo "Hello, World!" | tee output.txt

* echo "Hello, World!" prints "Hello, World!" to the screen.
* tee output.txt saves the output to output.txt and also displays it on the terminal.

**6. Sorting and removing duplicates:**

cat file.txt | sort | uniq

* cat file.txt reads the contents of file.txt.
* sort sorts the content.
* uniq removes any duplicate lines.

**🧑‍💼 Interview-Ready Answer:**

"Piping (|) in Linux allows us to pass the output of one command as the input to another. It’s used to chain multiple commands together, enhancing the power of command-line operations. For example, you can use ps aux | grep apache to find Apache processes or ls | less to view directory contents in a scrollable manner."

**🌐 Bonus: Combining Multiple Pipes**

You can chain multiple commands together with pipes. For example:

ps aux | grep apache | awk '{print $1, $3, $11}' | sort

This command:

1. Lists processes with ps aux.
2. Filters for apache using grep.
3. Prints specific columns ($1, $3, $11) with awk.
4. Sorts the output.

**🔍 What is grep?**

**grep** stands for **Global Regular Expression Print**.  
It is used to **search for specific patterns or words inside files or output of other commands**.

**📌 Basic Syntax:**

grep [options] "pattern" filename

* "pattern" → the text or word you are searching for.
* filename → the file where you want to search.

**✅ Common Use Cases:**

| **Command** | **Description** |
| --- | --- |
| grep "apple" fruits.txt | Search for the word "apple" in fruits.txt file. |
| grep -i "apple" fruits.txt | Case-insensitive search for "apple". |
| grep -r "main()" /home/user/code | Recursively search for main() in files under the directory. |
| grep -v "error" logs.txt | Show all lines that do NOT contain the word "error". |
| `ps aux | grep apache` |
| grep "^a" names.txt | Show lines starting with the letter "a". |

**🛠 Useful Options:**

| **Option** | **Description** |
| --- | --- |
| -i | Ignore case (uppercase/lowercase doesn’t matter). |
| -v | Invert match – shows lines that **do not** match the pattern. |
| -r or -R | Recursive search in all files and directories. |
| -n | Show line numbers where the pattern matches. |
| -c | Count how many lines match the pattern. |
| -l | List file names with matching lines (not the actual lines). |
| -w | Match the **whole word** only. |
| -A N | Show **N lines After** the matching line. |
| -B N | Show **N lines Before** the matching line. |
| -C N | Show **N lines Before and After** the match (Context). |

**🧪 Examples:**

**1. 🔍 Simple Search**

grep "hello" file.txt

Shows all lines in file.txt that contain the word "hello".

**2. 🔠 Case-insensitive Search**

grep -i "Hello" file.txt

Matches "hello", "Hello", "HELLO", etc.

**3. ❌ Invert Match**

grep -v "error" logs.txt

Shows all lines that **don’t** contain "error".

**4. 🔢 Line Numbers with Match**

grep -n "root" /etc/passwd

Shows line numbers where "root" appears in /etc/passwd.

**5. 🔁 Recursive Search in a Directory**

grep -r "def" /home/koushik/python\_scripts/

Search for the word def in all files inside the given directory.

**6. 📂 Show only File Names with Match**

grep -l "database" \*.txt

Shows the names of .txt files that contain "database".

**🧑‍💼 Interview-Ready Answer:**

"grep is a Linux command-line tool used to search for specific words or patterns in a file or output. It’s useful for quickly finding relevant information, debugging logs, or filtering output. Options like -i, -v, -n, and -r give it powerful capabilities."

**🔊 What is echo?**

The echo command in Linux is used to **display a line of text or string on the terminal**.  
It's one of the most commonly used commands in shell scripting and for printing messages.

**📌 Basic Syntax:**

echo [options] [string or text]

**✅ Common Use Cases:**

| **Command** | **Description** |
| --- | --- |
| echo Hello World | Prints "Hello World" to the terminal. |
| echo $USER | Displays the value of the variable $USER. |
| echo "My name is Koushik" | Displays a sentence inside quotes. |
| echo -n "Hello" | Prints "Hello" without a new line at the end. |
| echo -e "Line1\nLine2" | Enables interpretation of escape characters. |
| echo -e "Name\tKoushik" | Adds a tab between "Name" and "Koushik". |
| echo "This is a test" > file.txt | Writes "This is a test" to a file. |

**🔧 Useful Options:**

| **Option** | **Description** |
| --- | --- |
| -n | Do not print a new line after the output. |
| -e | Enable interpretation of **escape sequences** like \n, \t, \\, etc. |
| -E | (default) Disable interpretation of escape characters (used to cancel -e). |

**🔤 Escape Sequences (used with -e):**

| **Sequence** | **Meaning** |
| --- | --- |
| \n | New line |
| \t | Horizontal tab |
| \\ | Backslash |
| \" | Double quote |
| \a | Alert (beep) |

**🧪 Examples: 1. Basic Text:**

echo "Welcome to Linux"

Output:

Welcome to Linux

**2. No New Line:**

echo -n "Hello"

echo "World"

Output:

HelloWorld

**3. With Newline and Tabs:**

echo -e "Name:\tKoushik\nRole:\tDevOps Engineer"

Output:

Name: Koushik

Role: DevOps Engineer

**4. Printing Variable:**

name="Koushik"

echo "My name is $name"

Output:

My name is Koushik

**5. Writing to File:**

echo "This is Linux training" > notes.txt

This command writes the message into the file notes.txt. (It will overwrite existing content)

**🧑‍💼 Interview-Ready Answer:**

"echo is a Linux command used to print text or variables to the terminal. It's widely used in scripting to display messages or write content to files. With options like -e and -n, you can control formatting like newlines and tabs."

**✂️ What is cut?**

The cut command in Linux is used to **extract specific sections or fields** (columns or characters) from each line of a file or output.

It’s helpful when working with data files like CSV, log files, or command outputs.

**📌 Syntax:**

bash

CopyEdit

cut [OPTION]... [FILE]

You can also use cut with output from other commands using pipes (|).

**✅ Common Use Cases:**

| **Command** | **Description** |
| --- | --- |
| cut -c1-5 file.txt | Cuts the first 5 characters of each line. |
| cut -d "," -f2 file.csv | Cuts the 2nd column (field) from a comma-separated file. |
| `cat file.txt | cut -d ":" -f1` |
| cut -f1 file.txt | Cuts the first tab-separated field by default. |

**🟩 Using cut**

CopyEdit

cut -d ":" -f1 /etc/passwd

👉 Gets the first field (username) from colon-separated lines.

**🟨 Using awk (same result as above)**

CopyEdit

awk -F: '{print $1}' /etc/passwd

* -F: → Set delimiter to colon :
* $1 → Print the first field

✅ More powerful than cut because you can use conditions, format, and calculate.

**🟦 Using grep (Different Purpose)**

CopyEdit

grep "root" /etc/passwd

👉 grep is for **searching/filtering** lines.  
It **does not cut** or extract fields. It **shows full lines** that match "root".

**🎯 Summary with Examples**

**1. 🔍 Extract 2nd Column from CSV**

* Using cut:

cut -d "," -f2 students.csv

* Using awk:

awk -F, '{print $2}' students.csv

**2. 👁️ Filter lines with a name:**

* Using grep:

grep "Koushik" students.csv

**🧑‍💼 Interview Tip:**

"cut is used to extract specific fields or characters, while awk is more powerful for extracting, formatting, and even processing data. grep is used to search lines matching a pattern."

**🧠 What is awk?**

awk is a **powerful command-line tool** used for **text processing**, especially for working with **structured data** (like CSV, tab-separated files, logs, etc.).

You can use awk to **extract fields**, **filter rows**, and **perform calculations**.

**📌 Basic Syntax:**

awk 'pattern { action }' filename

Or with delimiter:

awk -F '<delimiter>' '{ action }' filename

**✅ Common Use Cases:**

| **Purpose** | **Example Command** | **What It Does** |
| --- | --- | --- |
| Print 1st column | awk '{print $1}' file.txt | Prints first word/field of each line |
| Print specific column (CSV) | awk -F, '{print $2}' data.csv | Prints 2nd column (comma-separated) |
| Match pattern and print | awk '/Koushik/ {print $0}' file.txt | Prints lines containing "Koushik" |
| Conditional print | awk '$3 > 50 {print $1, $3}' marks.txt | Print name & marks if marks > 50 |
| Calculate column total | awk '{sum += $2} END {print sum}' data.txt | Adds up 2nd column values |

**🎯 Summary with Examples**

**1. 🔍 Extract 2nd Column from CSV**

* Using cut:

cut -d "," -f2 students.csv

* Using awk:

awk -F, '{print $2}' students.csv

**2. 👁️ Filter lines with a name:**

* Using grep:

grep "Koushik" students.csv

**🧠 What is head?**

head command is used to **display the first few lines** of a file.

**✅ Syntax:**

head [options] filename

**📌 Default:**

* By default, it shows the **first 10 lines** of the file.

**📘 Examples:**

head file.txt

👉 Shows the first 10 lines.

head -n 5 file.txt

👉 Shows the first 5 lines.

head -n 15 /var/log/syslog

👉 Useful in DevOps to check the beginning of large log files.

**🧠 What is tail?**

tail command is used to **display the last few lines** of a file.

**✅ Syntax:**

tail [options] filename

**📌 Default:**

* By default, it shows the **last 10 lines**.

**📘 Examples:**

tail file.txt

👉 Shows the last 10 lines.

tail -n 5 file.txt

👉 Shows the last 5 lines.

tail -f /var/log/syslog

👉 -f means **follow the file live**, used for monitoring **live logs** (very useful in DevOps).

**🔁 Summary Table:**

| **Command** | **Description** |
| --- | --- |
| head file.txt | Show first 10 lines |
| head -n 20 | Show first 20 lines |
| tail file.txt | Show last 10 lines |
| tail -n 5 | Show last 5 lines |
| tail -f file | Live monitor the file (real-time updates) |

**🧑‍💼 Interview Answer:**

"head shows the top lines and tail shows the bottom lines of a file. These are useful for viewing the beginning or end of logs and files. The tail -f option helps monitor logs in real time, which is commonly used by DevOps engineers."

Would you like to see a **real-time log monitoring example with tail -f**?

**🧠 What is the find command?**

find is a command-line tool used to **search for files and directories** in a **directory hierarchy** based on name, size, type, date, permission, etc.

It’s very powerful and widely used in real-time DevOps and Linux administration.

**✅ Basic Syntax:**

find [path] [options] [expression]

**📘 Common Examples:**

**1. 🔍 Find a file by name:**

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

➡️ Searches for a file named file.txt inside /home/koushik

**2. 🔍 Case-insensitive name:**

find /home/koushik -iname "File.txt"

➡️ Same as above, but ignores uppercase/lowercase.

**3. 📂 Find all .txt files:**

find . -name "\*.txt"

➡️ Finds all .txt files under current directory.

**4. 📁 Find directories only:**

find . -type d

➡️ Lists all directories.

**5. 📄 Find files only:**

find . -type f

➡️ Lists all files.

**6. 🧭 Find files modified in last 1 day:**

find . -mtime -1

➡️ Shows files modified within the last 24 hours.

**7. 🔒 Find files by permissions:**

find . -perm 644

➡️ Finds files with permission 644.

**8. 🗑️ Find and delete .log files:**

⚠️ Be careful with -delete

find . -name "\*.log" -delete

➡️ Deletes all .log files in current and sub-directories.

**9. 🧹 Find large files (size > 100MB):**

find . -size +100M

➡️ Finds files bigger than 100MB.

**10. 🔧 Run a command on each file:**

find . -name "\*.sh" -exec chmod +x {} \;

➡️ Makes all .sh files executable.

**🧑‍💼 Interview Tip:**

"find is a powerful Linux command used to locate files and directories based on name, type, size, date, permissions, and more. It's commonly used in automation and server maintenance to search and perform actions on specific files."

**🧠 Summary Table:**

| **Option** | **Purpose** |
| --- | --- |
| -name | Match file name |
| -iname | Match name (case-insensitive) |
| -type f/d | File or directory |
| -size | Match file size |
| -mtime | Modified time |
| -perm | Match permission |
| -delete | Delete matched files |
| -exec | Run command on each result |