🖥️ What is an Operating System (OS)?

➔ An **Operating System** is a software that acts like a manager between:

- **You** (the user) 👨‍💻
- **Computer hardware** 🧠💻 (CPU, RAM, hard drive, etc.)

➔ You can't use a computer without an OS — it's what runs everything.

🎯 What Does an OS Do?

Function	Simple Explanation
🧍 Interface	Provides a screen/interface to interact (Windows, icons, menus)
🧠 Memory Manager	Controls how much RAM each app can use
⚙️ Process Manager	Manages which app runs, and when
🗂️ File Manager	Lets you save, open, delete files and folders
🌐 Device Manager	Controls hardware like keyboard, mouse, printer, USB
🛡️ Security	Protects your data with passwords, user permissions

👨‍💻 Simple Example:

When you:

- Open a browser → OS **loads it into RAM**
- Save a file → OS **stores it on hard drive**
- Press print → OS **sends it to the printer**

🧱 Types of Operating Systems

Type	Description	Example
1. Batch OS	Jobs run in groups without user interaction	Old mainframes
2. Time-sharing OS	Multiple users at the same time	UNIX
3. Distributed OS	Manages many computers as one	Google server systems
4. Embedded OS	Used in machines/devices	Washing machine, ATM
5. Real-Time OS (RTOS)	Instant response systems	Airbags, traffic lights
6. Mobile OS	OS for mobile phones	Android, iOS
7. Desktop OS	OS for PCs/laptops	Windows, macOS, Linux

🪟 Popular Operating Systems

OS	Used In	Made By
Windows	PC, Laptops	Microsoft
macOS	Apple Laptops	Apple
Linux	Servers, Dev use	Open source
Android	Phones, Tablets	Google
iOS	iPhones, iPads	Apple

==========

📦 Components of an Operating System

1. **Kernel** – the brain; controls everything (low-level tasks)
2. **Shell** – your way to give commands (like terminal)
3. **File System** – organizes data on the hard drive
4. **User Interface (UI)** – what you see (buttons, windows, etc.)

==========

🔹 What is a Kernel?

➔ The **kernel** is the **core part of any operating system**.
It connects **software (apps)** with **hardware (CPU, RAM, etc.)** and controls everything in the background.

➢ **Why is it Important?**

The kernel:

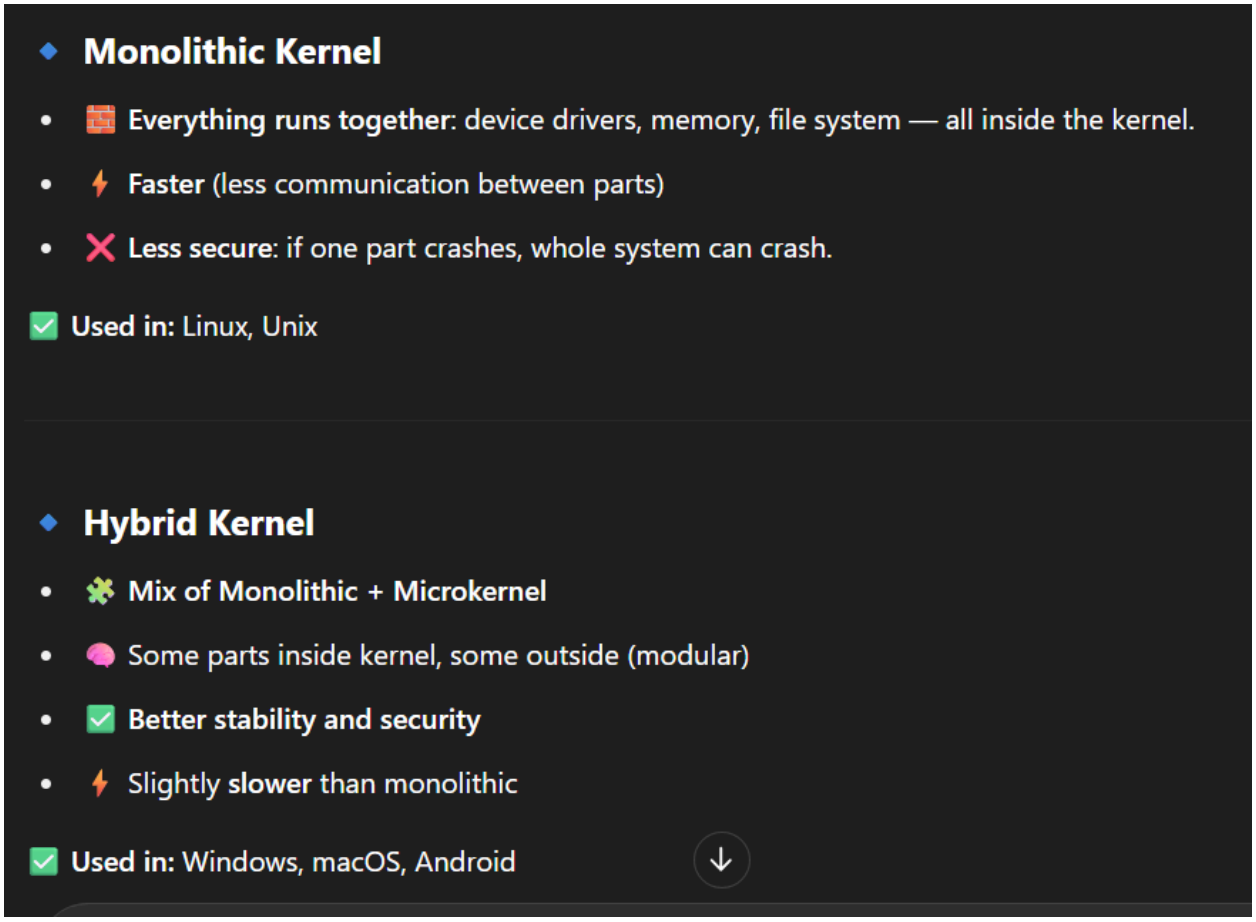
- Gives apps access to **CPU and memory**
- **Controls hardware** (keyboard, mouse, printer)
- Manages **file systems**
- Keeps the system **secure and stable**

🔹 Is Kernel in All OS?

➔ ✅ Yes — every OS like **Windows, Linux, macOS, Android** has a **kernel**.

🔹 **Types of Kernels:**

Type	Used In
Monolithic	Linux
Microkernel	Embedded OS
Hybrid	Windows, macOS, Android



📊 Quick Comparison Table:

Feature	Monolithic Kernel	Hybrid Kernel
Structure	Everything inside	Mixed (some inside, some outside)
Speed	Fast	Moderate
Stability	Less (one bug can crash system)	More stable
Used In	Linux, Unix	Windows, macOS, Android

==========

🐚 What is a Shell?

➔ A Shell is a program that lets you interact with the Operating System.

➔ You give commands, and the shell passes them to the OS, which then executes them.

➔

🎯 Simple Example:

When you open a terminal and type:

```
arduino

mkdir myfolder
```

→ The **Shell** sends this command to the OS → OS creates the folder.

🧱 Types of Shells

There are **two main types**:

Type	Description	Example
Command Line Shell (CLI)	Text-based interface where you type commands	Bash, Zsh, PowerShell
Graphical Shell (GUI)	Uses icons, windows, mouse clicks	Windows Explorer, GNOME, KDE

➔

💻 Common **Command Line Shells:**

Shell	Used In	Description
Bash	Linux, macOS	Most popular, easy to use
Zsh	Linux, macOS	Advanced features, used by developers
Sh (Bourne Shell)	Unix	Older, basic shell
Csh (C Shell)	Unix	C-like syntax
PowerShell	Windows	Powerful shell for system automation
Fish	Linux	Friendly, user-focused shell

➔

📊 Bash vs PowerShell (Example):

Task	Bash (Linux/macOS)	PowerShell (Windows)
List files	`ls`	`Get-ChildItem`
Remove file	`rm file.txt`	`Remove-Item file.txt`
Print text	`echo Hello`	`Write-Output Hello`

➔

🧁 Simple Analogy:

Think of the **shell as a translator**:

- You (user) speak in **commands**
- Shell translates to the **OS**
- OS performs the task

➔

🧪 Quick Example: Bash vs Zsh

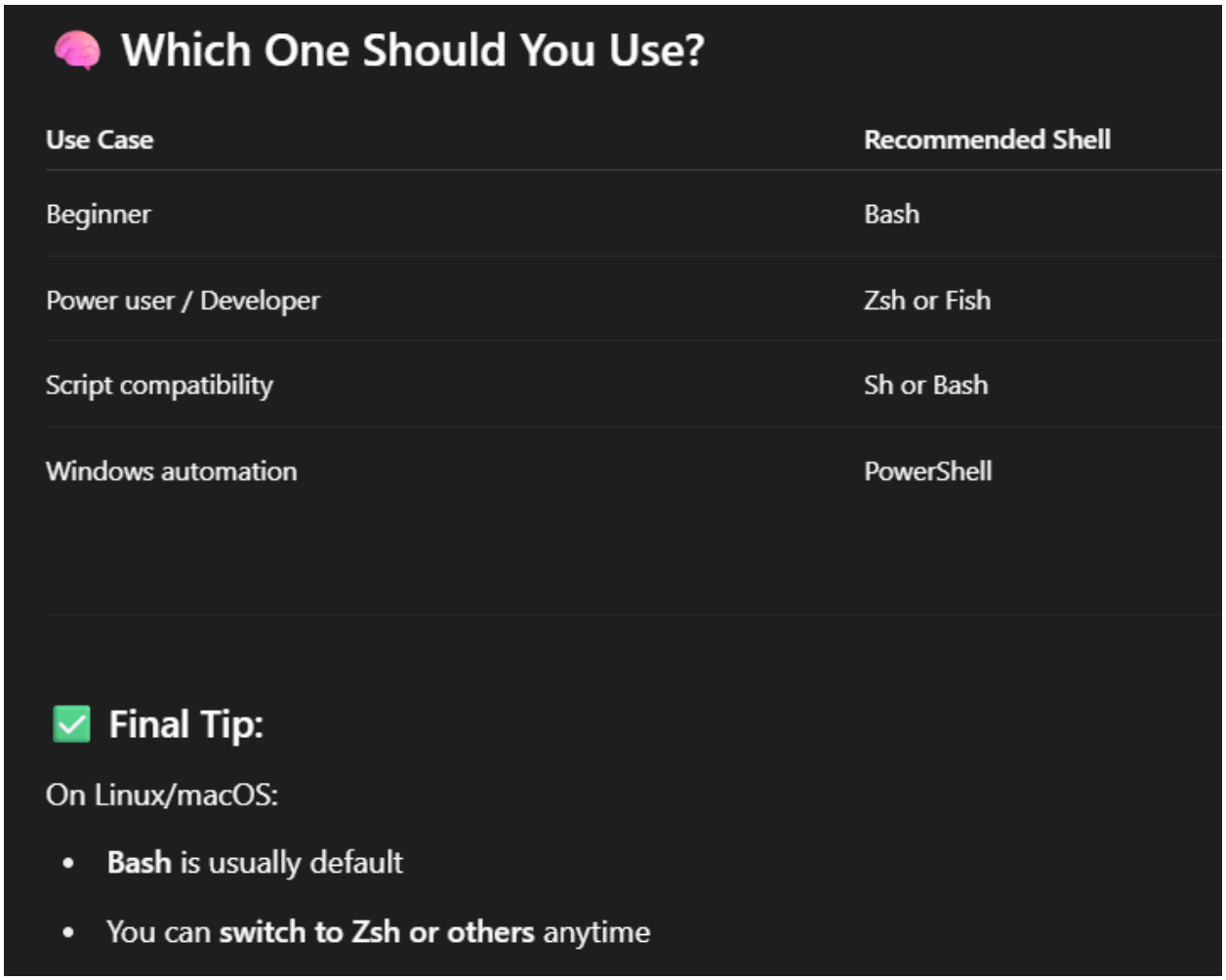
Both use similar commands like:

bash

```
cd /home
ls
mkdir test
```

But **Zsh** gives:

- **Auto-suggestions**
- **Better tab-completion**
- **Plugins and themes (e.g., oh-my-zsh)**



==========

What is a Terminal?

➔ A **Terminal** is the **window or interface** where you **interact with the shell**.

🧠 **Simple Definition:**

The **Terminal is the screen**, and the **Shell is the brain** that understands and runs your commands.

🧾 **How They Work Together:**

- **Terminal** = Tool to type commands
- **Shell** = Program that runs those commands

When you open **Command Prompt**, **Terminal**, or **PowerShell**, you are opening a terminal that runs a shell.

🧁 Analogy:

Thing	Role
Terminal	Phone screen (where you type)
Shell	The operating system (which makes the call)
OS	The network (that completes the task)

🧠 In Short:

Tool	What It Is	Used On	Main Use
Command Prompt (CMD)	Old Windows command-line tool	Windows	Basic commands (e.g., `dir`, `copy`)
PowerShell	Advanced scripting tool by Microsoft	Windows, macOS, Linux	Automation & scripting (`Get-Process`, `Remove-Item`)
Terminal	A **window/interface** to run **any shell**	Windows (via Windows Terminal), Linux, macOS	Opens Bash, PowerShell, CMD, etc.

📊 Quick Comparison Table:

Feature	Command Prompt	PowerShell	Terminal
Type	Shell	Shell	Interface
Platform	Windows only	Cross-platform	Cross-platform
Scripting	Limited	Powerful	Runs other shells
Purpose	Basic tasks	System automation	Tool to run shells

==========

Understand Unix, Linux, macOS, and Windows Os

🖥️ 1. UNIX

- 📌 **Oldest OS**, created in the 1970s.
- Used mostly in **servers**, **mainframes**, and **scientific systems**.
- It's **stable**, **secure**, and mostly **command-line based**.

✅ **Used by:** Telecom, universities, large servers.

🐧 2. Linux

- 📌 **Open-source clone of UNIX**.
- Free to use and **highly customizable**.
- Used in **servers, desktops, Android phones, routers, smart TVs, IoT**.

✅ **Used by:** Developers, system admins, websites, cloud servers.

Popular Linux versions (called distros):

- Ubuntu
- Fedora
- Debian
- Kali Linux

🍎 3. macOS

- 📌 Apple's OS for **Mac computers**.
- Based on UNIX underneath (called Darwin).
- **Smooth, secure, user-friendly GUI**.
- Limited to Apple hardware.

✅ **Used by:** Designers, developers, video editors, Apple users.

🪟 4. Windows

- 📌 Developed by Microsoft.
- Most popular OS for **home and office use**.
- Easy to use, runs **most commercial software and games**.
- Closed-source, GUI-based.

✅ **Used by:** Office workers, gamers, general users, developers (with Visual Studio)

📊 Comparison Table:

Feature	UNIX	Linux	macOS	Windows
Free	❌	✅	❌	❌
Open Source	❌	✅	Partially	❌
User Interface	CLI	CLI + GUI	GUI	GUI
Customizable	❌	✅	❌	❌
Used In	Servers	Everything	Apple devices	PC, Office, Gaming

🧠 When to Use What?

Goal	Best OS
Learning OS, coding, hacking	**Linux**
Stable server or cloud system	**Linux or UNIX**
Design, video editing	**macOS**
Office, general use, gaming	**Windows**

==========

==========

Deep Dive in Linux

==========

🐧 What is a Linux Distribution (Distro)?

A **Linux distro** is a **complete version** of Linux that includes:

- Linux kernel
- System tools
- Software (like browser, file manager)
- Package manager

Different distros are made for **different users** (beginners, servers, hackers, etc.)

Popular Linux Distros (2025)

Linux Distro	Best For	Why It's Good
Ubuntu	Beginners & Developers	Easy to use, huge community, lots of tutorials
Linux Mint	Windows Users	Familiar Windows-like look, stable
Debian	Servers & Advanced Users	Very stable, secure, used as base for Ubuntu
Fedora	Developers	Latest features, backed by Red Hat
Arch Linux	Experts	Lightweight, customizable, rolling updates
Pop!_OS	Gamers & Developers	Great performance, clean UI, based on Ubuntu
Kali Linux	Ethical Hacking	Preinstalled hacking and security tools
Zorin OS	New Linux Users	Clean and Windows-like UI
elementary OS	Mac Users	Sleek macOS-like interface
CentOS Stream	Servers	Red Hat-based, good for learning RHEL environment

🏆 Which One is Best?

You Are...	Best Distro
🆕 New to Linux	Ubuntu or Linux Mint
💻 Developer	Fedora or Pop!_OS
🥷 Hacker	Kali Linux
🧠 Learner	Ubuntu or Debian
🖥️ Server Admin	Debian or CentOS Stream
🎮 Gamer	Pop!_OS
💻 Windows Switcher	Linux Mint or Zorin OS
🍎 Mac Switcher	elementary OS

==========

🐧 Why Linux is Used for Servers

✅ 1. Free and Open Source

- No license cost — companies save money.
- Full control over source code.

✅ 2. Stable and Reliable

- Can run for **months or years** without crashing.

- Perfect for servers that need 24/7 uptime.

✅ 3. Secure

- Strong user permissions and firewall.
- Regular security updates and less targeted by viruses.

✅ 4. Lightweight

- Can run even on low-resource systems.
- No need for heavy GUI (can be command-line only).

✅ 5. Highly Customizable

- You install only what you need — nothing extra.
- Better performance and faster boot time.

✅ 6. Community and Support

- Massive global community.
- Tons of tools and documentation available.

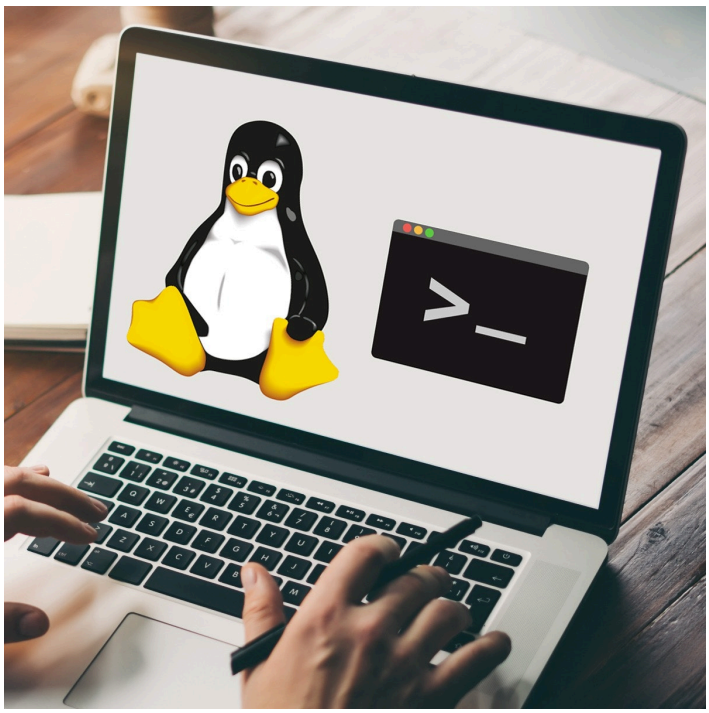
🔧 Examples of Linux Server Use:

- Web servers (Apache, Nginx)
- Cloud servers (AWS, Azure)
- Database servers (MySQL, PostgreSQL)
- Hosting platforms (cPanel, Plesk)
- Networking devices (firewalls, routers)

==========

🐧🐧 Linux Commands 🐧🐧

==========



A **command** is an instruction you type in the **terminal** to make the **Linux operating system** do something.

It could be to:

- Show files,
- Create folders,
- Copy/move/delete files,
- Install software,
- Show system info etc.

==========

Command Not found 👋:

Spelling mistake

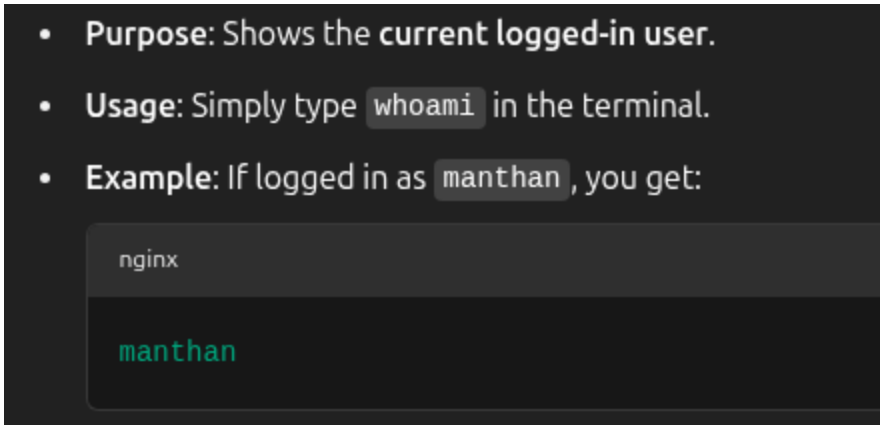
🔍 Example: Typing sl instead of ls

❗ Solution: Check for typos.

```
admin123@admin123:~$ ddfefd
ddfefd: command not found
admin123@admin123:~$ sssssssssss
sssssssssss: command not found
admin123@admin123:~$ SSSSSS
SSSSSS: command not found
```

==========

Whoami Command :



==========

man Command — The Manual Helper

➔ The man command shows the **manual (help documentation)** for any Linux command.

🧪 **Syntax:**

man [command-name]

🗒️ **Example Usage:**

Command	What It Shows

man ls	Manual for ls command
man pwd	Manual for pwd
man mkdir	Manual for mkdir

==========

clear Command

➔ Purpose: Clears the terminal screen.

Usage:

clear

➔ It doesn't delete anything—just clears the visual clutter from your terminal screen so you can work with a clean interface.

✅ Tip: You can also press Ctrl + L for the same effect.

==========

📁 **What is a Directory?**

➔ A **directory** is like a **folder** on your computer. It is used to organize and store files and other directories.

➔ For example:

/home/manthan/Documents

In this example:

- `/` is the **root** directory (the top-level folder).
- `home` is a directory inside `/`.
- `manthan` is your personal directory inside `home`.
- `Documents` is a folder inside `manthan`.

So, directories help structure and organize your data in a tree-like format.

==========

pwd Command — Print Working Directory

Use: Shows your **current directory path** (i.e., where you are in the folder structure).

Command:

pwd

Example Output:

```
arduino

/home/manthan/Documents
```

This means you're currently in the `Documents` directory inside your user folder.

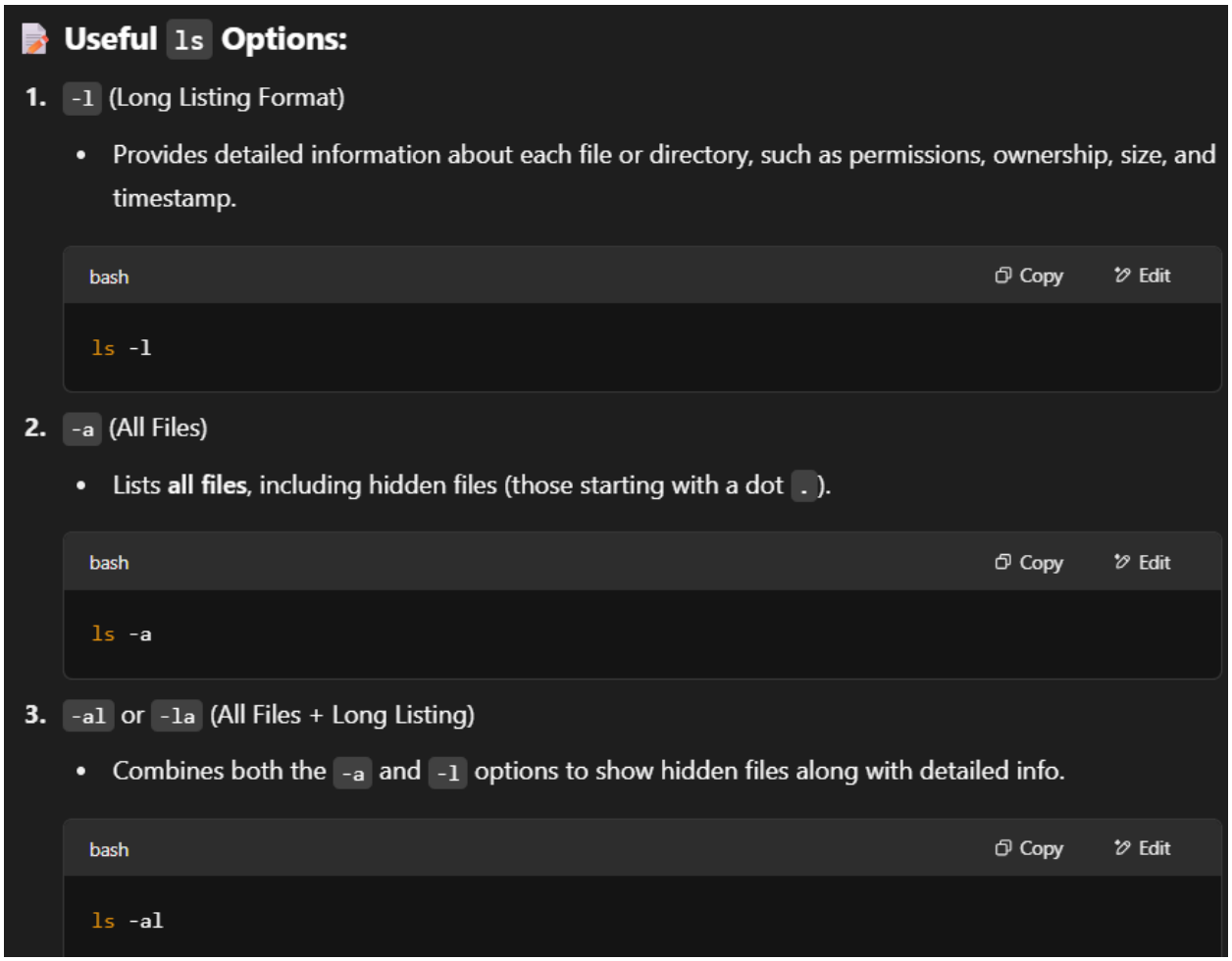
==========

`ls` Command — List Directory Contents

Basic Command:

ls

This will list the files and directories in the **current directory**.



4. `-h` (Human-Readable)

   - Shows file sizes in a **human-readable** format (e.g., KB, MB, GB).

```bash
ls -lh
```

5. `-R` (Recursive)

   - Lists files **recursively** in all subdirectories.

```bash
ls -R
```

6. `-S` (Sort by Size)

   - Sorts files by their **size**, with the largest files listed first.

```bash
ls -S
```

7. `-t` (Sort by Modification Time)

- Sorts files by the **time they were last modified**, with the most recent first.

```bash
ls -t
```

8. `-r` (Reverse Order)

- Reverses the order in which files are listed (useful with `-t` or `-S`).

```bash
ls -lr
```

9. `--color` (Colorized Output)

- Highlights files and directories with different colors to differentiate between types of files (directories, executables, etc.).

```bash
ls --color
```

==========

Cd Command — Change Directory

➔ The cd command is used to **navigate** from one directory (folder) to another in the Linux terminal.

➔ ✅ Basic Usage:

◆ **cd directory_name**

📌 Common `cd` Usages:

1. **Go to a specific directory**
    - **cd Documents**

---

2. **Go up one directory**
    - **cd ..**

➔ If you're in /home/manthan/Documents, this will take you to /home/manthan.

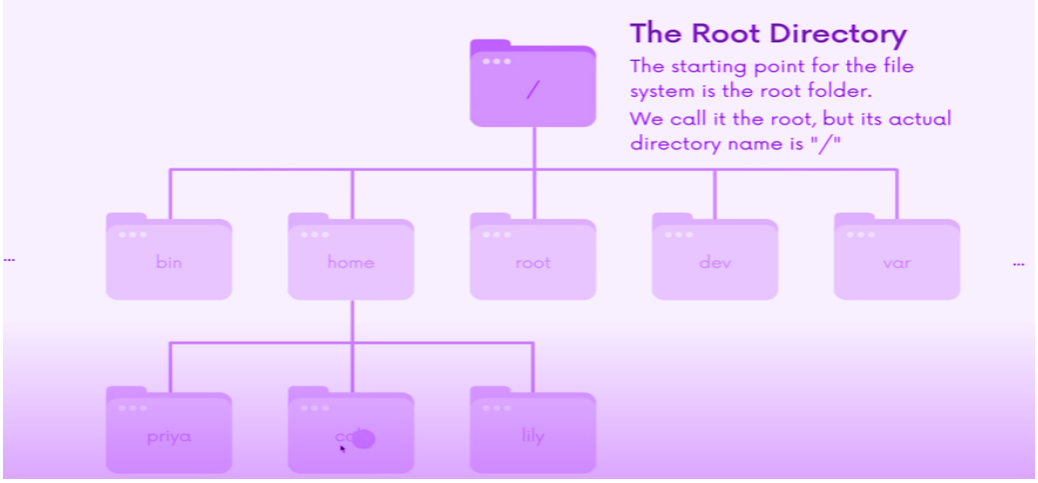
---

3. **Go back to home directory**
    - **cd**

      Or
    - **cd ~**

---

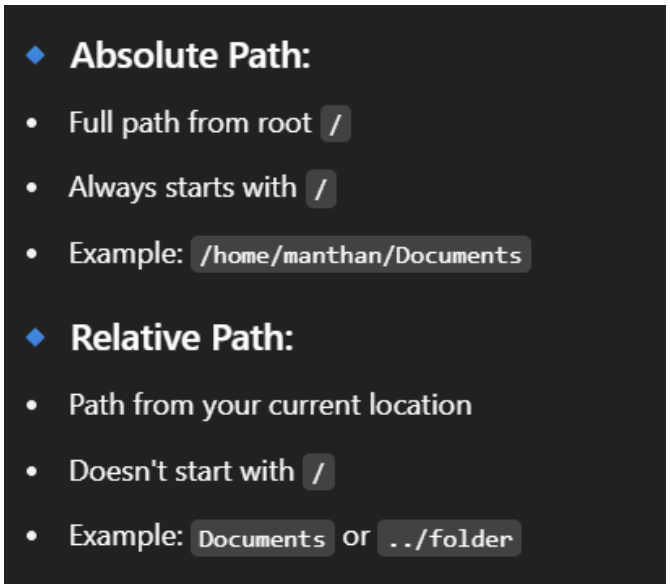
4. **Go to root directory**
    - **cd /**



---

5. **Go to a full path**

   cd /home/manthan/Desktop



---

6. **Return to previous directory**

   cd -

➔ Switches back to the directory you were in before the last cd.

==========

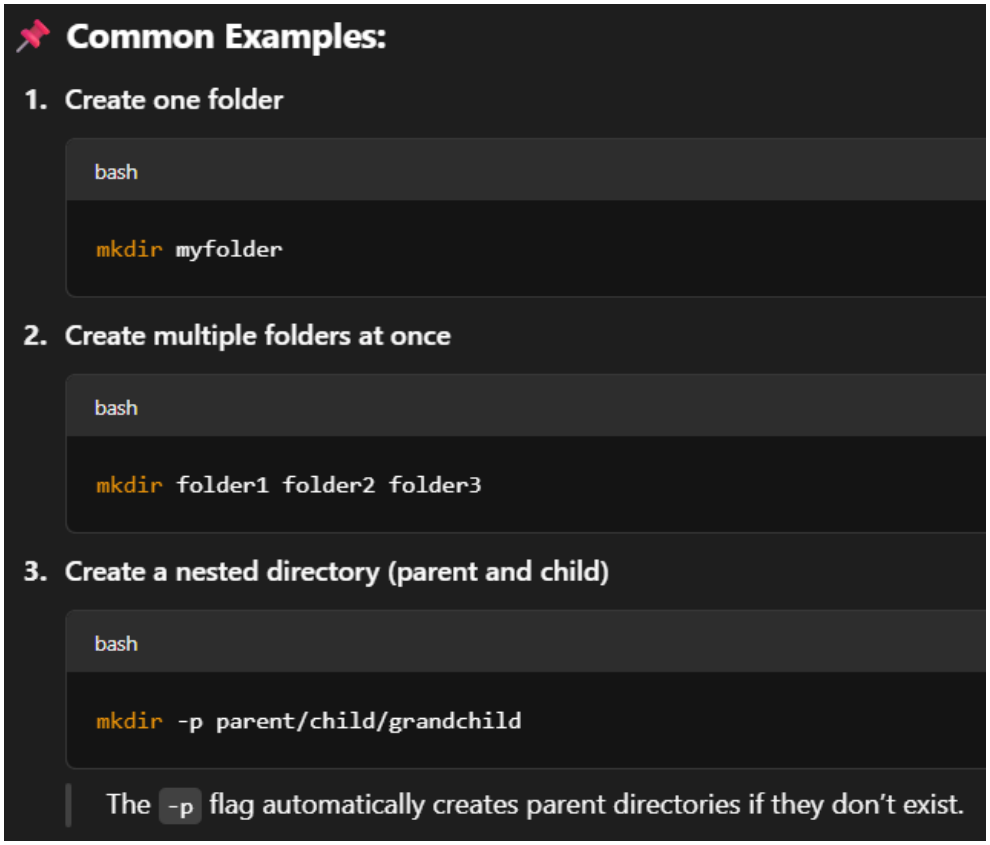
mkdir Command — Make Directory

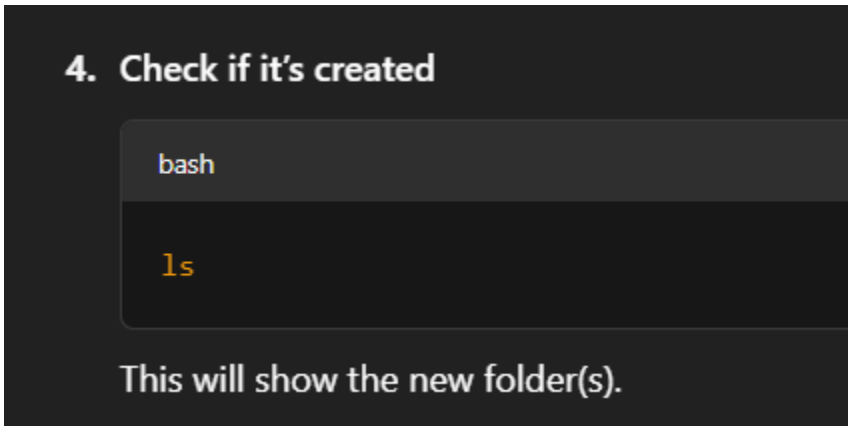
Creates new directories (folders) in your current or specified location.

✅ Basic Usage:

mkdir folder_name

Creates a folder named folder_name in the current directory.





==========

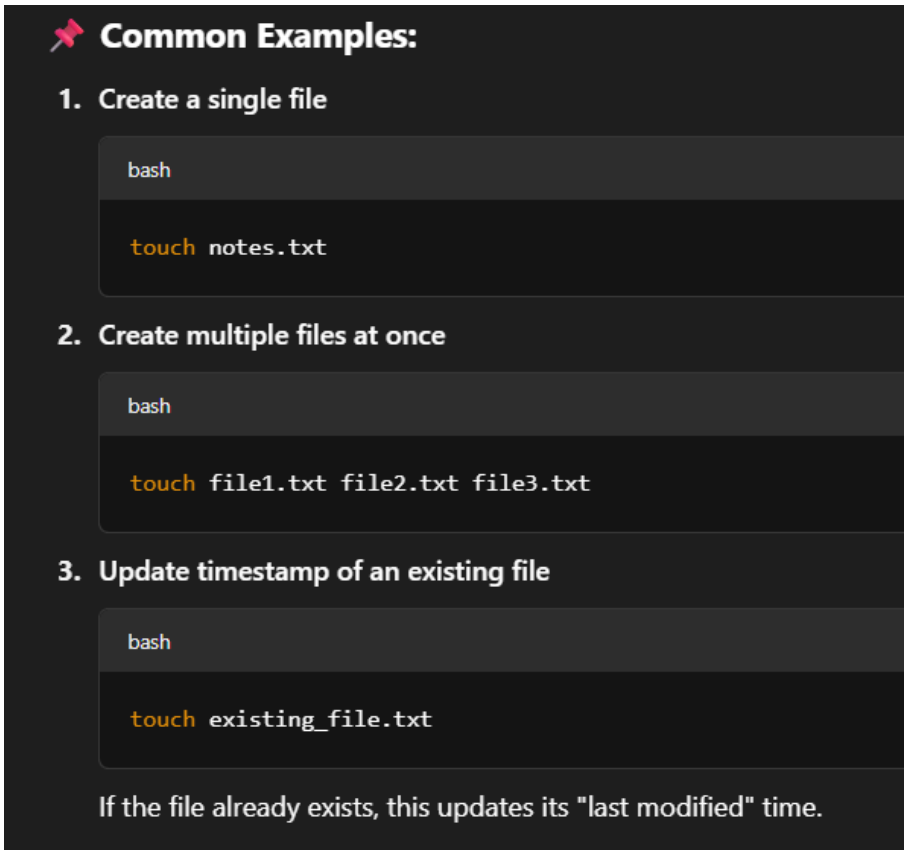
touch Command📄 — Create File(s)

➔ Creates new empty files or updates the timestamp of existing ones.

✅ Basic Usage:

touch filename.txt

➔ Creates a file named filename.txt in the current directory.



==========

rm Command 🗑️ — Remove Files or Directories

✅ Basic Usage:

rm filename.txt

➔ Deletes the file filename.txt.

📌 Common Examples:

1. **Remove a file**

```bash
rm file.txt
```

2. **Remove multiple files**

```bash
rm file1.txt file2.txt
```

3. **Remove an empty directory**

```bash
rmdir foldername
```

> Only works if the folder is empty.

4. **Remove a directory and its contents**

```bash
rm -r foldername
```

- `-r` means **recursive** — deletes all files and subfolders inside.

5. **Force delete without asking**

```bash
rm -rf foldername
```

- `-f` means **force** — doesn't ask for confirmation.
- ⚠️ **Dangerous** — use carefully, especially with `sudo`.

❗ *Be Careful:*

The rm command **permanently deletes** files — they don't go to a trash bin.

==========

mv Command📦 — Move or Rename Files/Folders

✅ **Basic Usage (Move a file):**

mv source.txt destination_folder/

➔ Moves source.txt into the destination_folder.

📌 Common Examples:

1. **Move a file to another directory**

```bash
mv file.txt /home/manthan/Documents/
```

2. **Rename a file**

```bash
mv notes.txt todo.txt
```

3. **Move and rename at the same time**

```bash
mv file.txt /home/manthan/Documents/newfile.txt
```

4. **Move a folder**

```bash
mv myfolder/ /home/manthan/Desktop/
```

==========

cp Command 📄 — Copy Files and Folders

✅ Basic Usage (Copy a file):

cp source.txt destination.txt

- This copies source.txt to a new file called destination.txt.

📌 Common Examples:

1. **Copy a file to another directory**

```bash
cp file.txt /home/manthan/Documents/
```

2. **Copy and rename a file**

```bash
cp file.txt newfile.txt
```

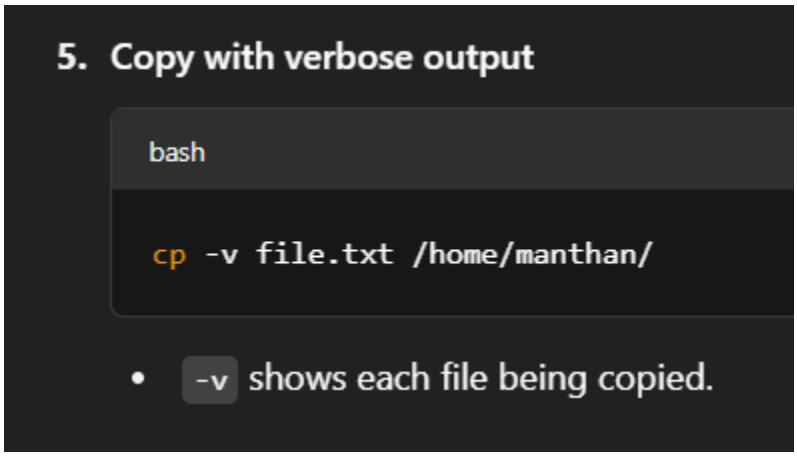
3. **Copy multiple files to a directory**

```bash
cp file1.txt file2.txt /home/manthan/Documents/
```

4. **Copy a folder (including all contents)**

```bash
cp -r folder1/ folder2/
```

- `-r` means **recursive**, required for copying folders.

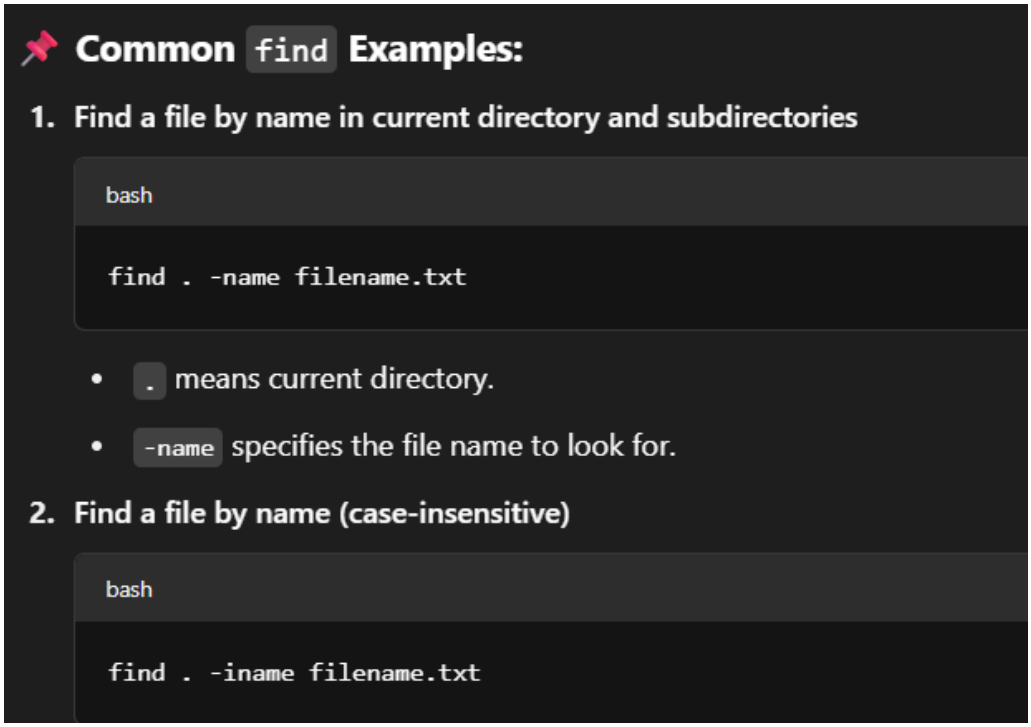


==========

find Command 🔍— Search for Files and Directories

✅ Basic Syntax:

find [path] [options] [expression]



3. Find all `.txt` files

bash

```
find . -name "*.txt"
```

4. Find a directory

bash

```
find . -type d -name "foldername"
```

5. Find files by size (e.g., files larger than 10MB)

bash

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

6. Find files modified in the last 7 days

bash

```
find . -mtime -7
```

7. Find and delete files

⚠️ Use with care!

bash

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

- `find /home -name "*.txt"` - Find all .txt files in /home
- `find . -type d -name "dir*"` - Find directories starting with "dir"

==========

`cat` *Command 📖 — View or Concatenate Files*

✅ Basic Usage:

cat filename.txt

➔ Displays the content of filename.txt on the terminal.

📌 Common `cat` Examples:

1. **View the contents of a file**

```bash
cat file.txt
```

2. **View multiple files**

```bash
cat file1.txt file2.txt
```

3. **Create a new file**

```bash
cat > newfile.txt
```

- Type content and press `Ctrl + D` to save.

4. **Append content to an existing file**

```bash
cat >> existingfile.txt
```

- Type content and press `Ctrl + D` to append.

5. **Join two files into one**

```bash
cat file1.txt file2.txt > combined.txt
```

6. **Show line numbers**

```bash
cat -n file.txt
```

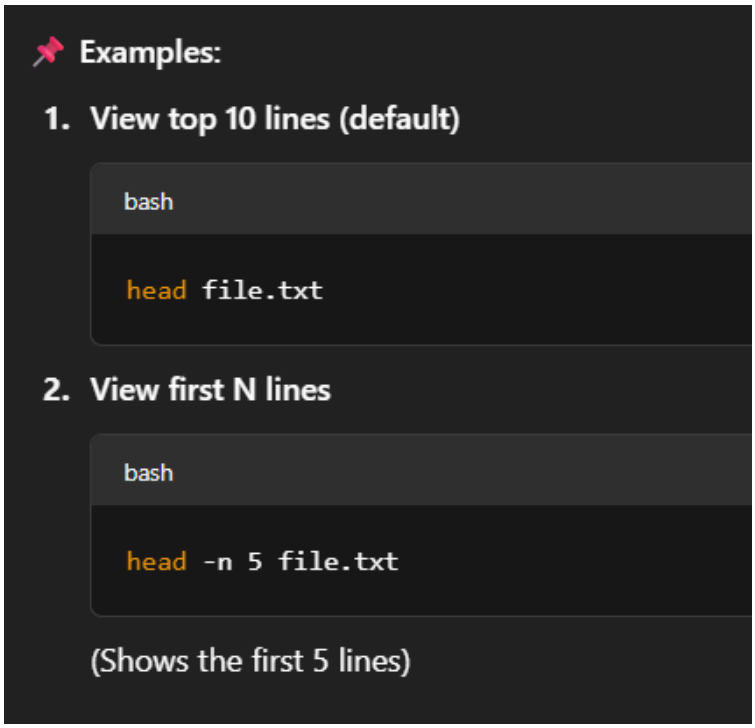
==========

head Command 🔼 — View the Top of a File

✅ Basic Usage:

head filename.txt

- Shows the **first 10 lines** of the file by default.

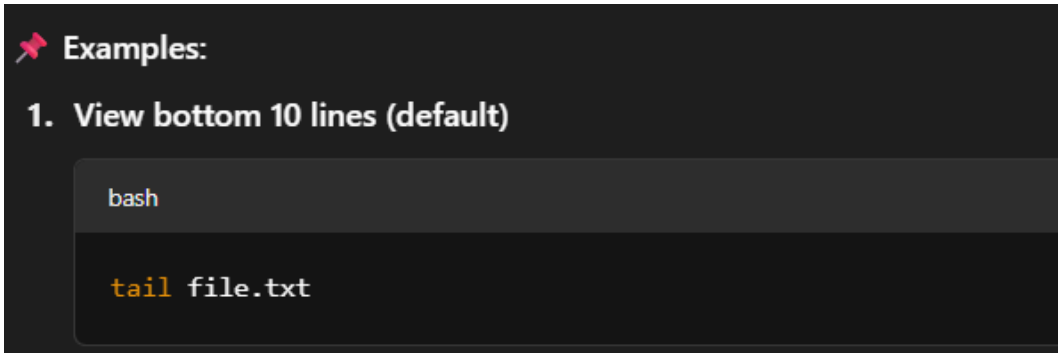


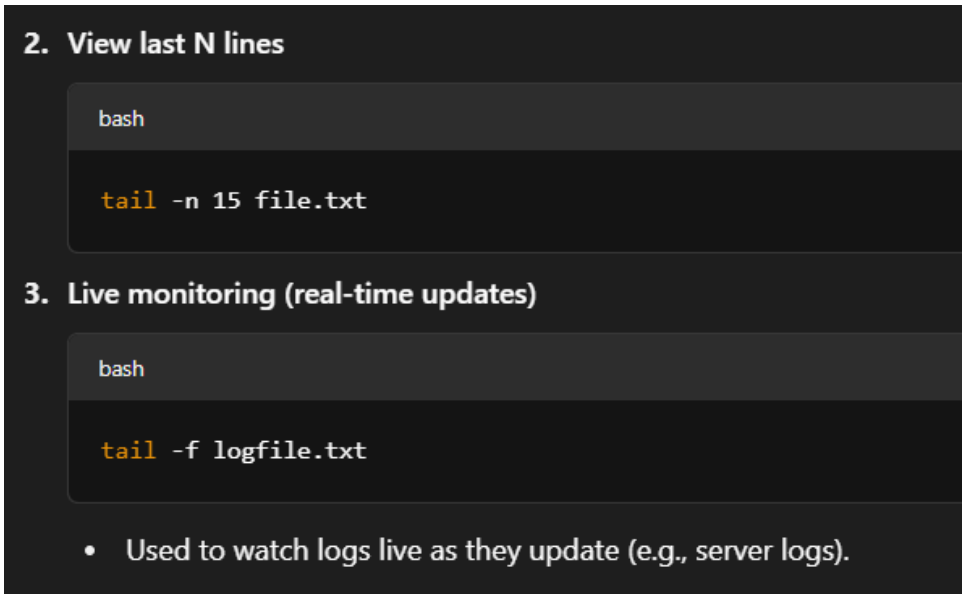
==========

`tail` Command 🔽— View the Bottom of a File

✅ Basic Usage:

tail filename.txt

- Shows the **last 10 lines** of the file by default.
- 



==========

less Command 📚 — View Big Files Page by Page

✅ Basic Usage:

less filename.txt

- Opens the file in a **scrollable view** (no need to display all lines at once).

🖱️ Navigation Controls:

- `↑ / ↓` or `k / j` → Scroll up/down line by line
- `Space` → Scroll down one page
- `b` → Scroll up one page
- `/word` → Search for a word
- `n` / `N` → Next/previous search result
- `q` → Quit `less`

📌 Examples:

1. **Open a big file**

```bash
less largefile.txt
```

2. **View logs in scrollable mode**

```bash
less /var/log/syslog
```

3. **Search inside file**

```bash
/error
```

VS `cat` vs `less`:

- `cat` shows everything at once (good for small files).
- `less` allows scroll/search in big files (better for long files).

more similar to less ?

Ans. ***Yes. more is similar to less.***

VS `less` vs `more`

Feature	`less`	`more`
Scroll up	✅ Yes	❌ No
Scroll down	✅ Yes	✅ Yes
Search (forward & backward)	✅ Yes	❌ Forward only
Opens faster	✅ Yes	✅ Yes
Handles large files	✅ Better	✅ Okay

🔔 In Short:

- `less` **is more powerful** and commonly used today.
- `more` **is simpler** and came earlier in Unix systems.

==========

date Command 🕑 — Show Current Date and Time

✅ Basic Usage:

date

Displays the current **date, time, time zone**, and **day**.

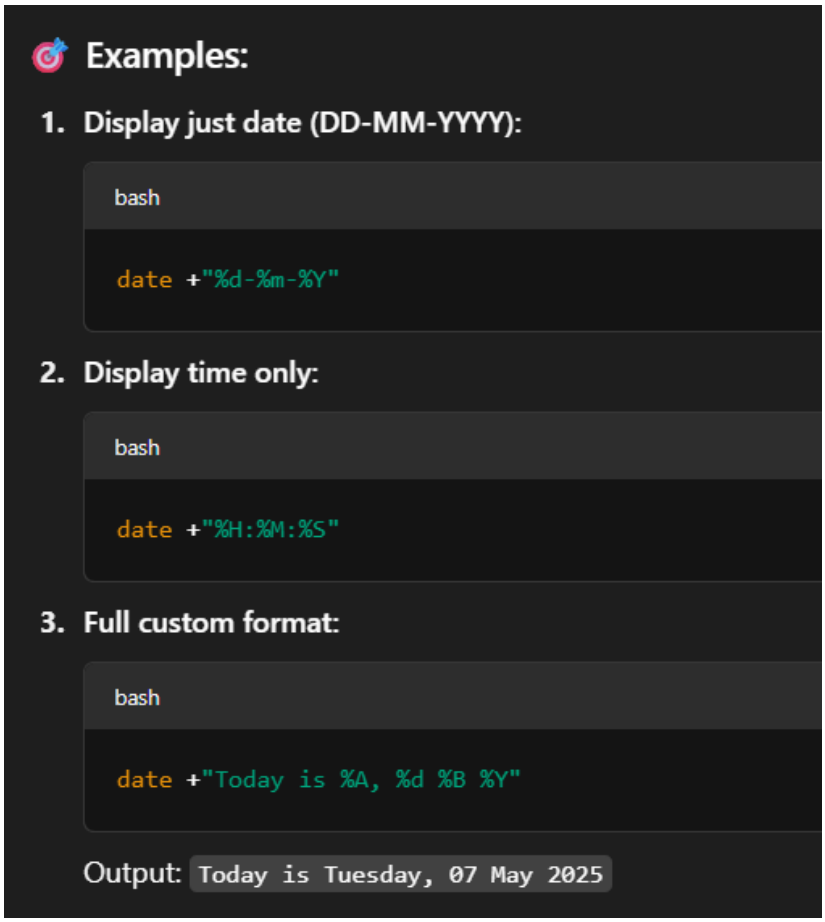
📌 Example output:

Tue May 7 14:32:01 IST 2025

📌 Format the Output:

You can customize how the date/time looks using `+` and format options.

Format	Meaning	Example Output
`%d`	Day of month (01–31)	07
`%m`	Month (01–12)	05
`%Y`	Year (4 digits)	2025
`%H`	Hour (00–23)	14
`%M`	Minute (00–59)	32
`%S`	Seconds (00–59)	01



==========

Redirecting Standard Output 📤 (stdout)

✅ 1. Use > to Overwrite a File

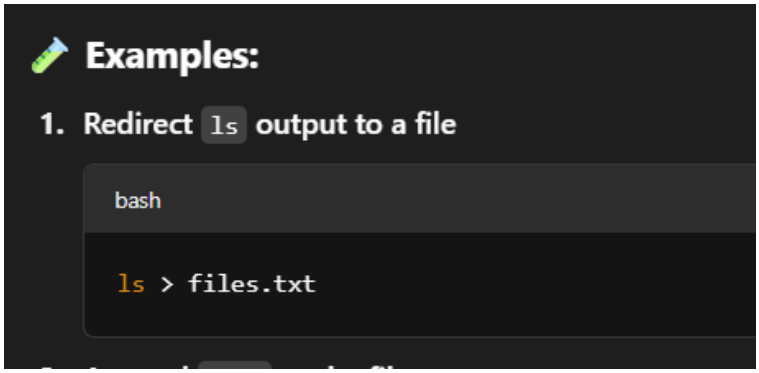
echo "Hello Manthan" > output.txt

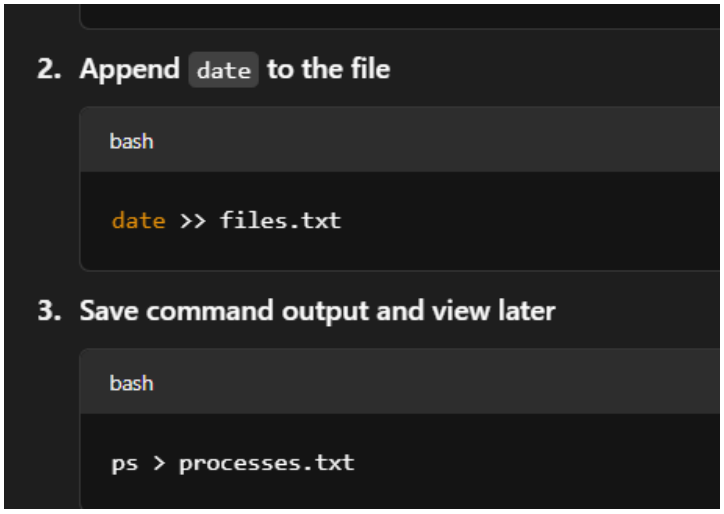
- Saves the output to output.txt.
- If the file exists, it will **overwrite** the contents.

✅ 2. Use >> to Append to a File

echo "Another line" >> output.txt

- Appends the output to the end of the file.
- Does **not delete** existing content.





==========

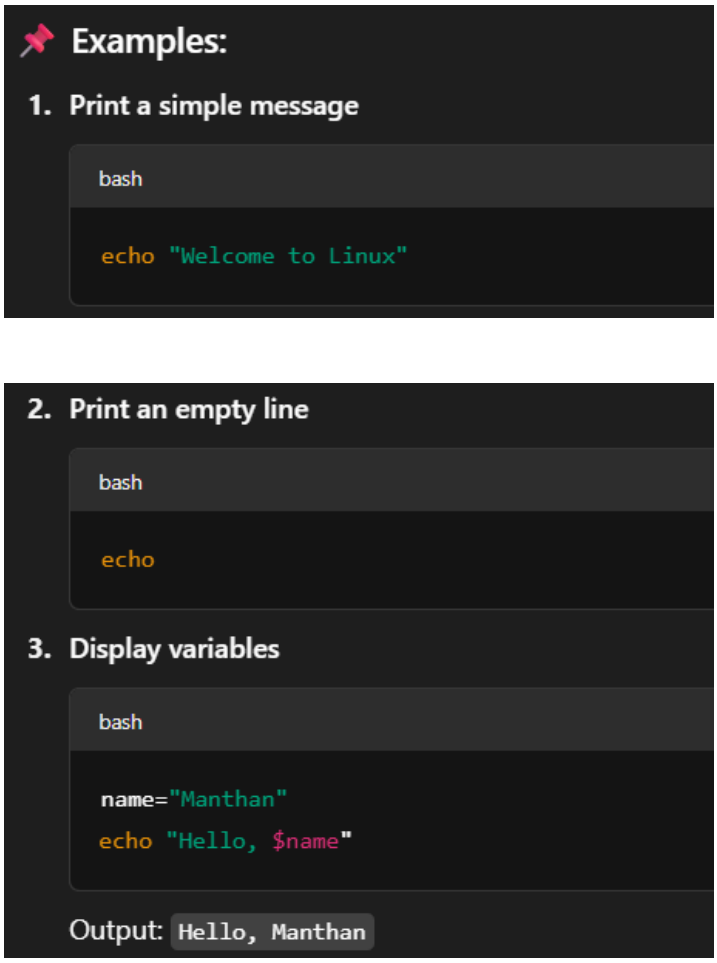
echo Command 📢 — Print Output to Terminal

✅ Basic Usage:

```
echo Hello World
```

Output:

```
Hello World
```



4. **Use escape characters (like newlines, tabs)**

   Use `-e` option:

```bash
echo -e "Line1\nLine2\tTabbed"
```

5. **Redirect echo output to a file**

```bash
echo "This is a test" > test.txt
```

🔧 Common Options:

Option	Description
`-e`	Enable escape sequences
`-n`	Do not output trailing newline

==========

wc Command🔢 — Word Count and More

✅ Basic Usage:

wc filename.txt

Output example:

5 12 78 filename.txt

This means:

- 5 lines
- 12 words
- 78 bytes

🧪 Examples:

1. **Count lines only**

```bash
wc -l file.txt
```

2. **Count words only**

```bash
wc -w file.txt
```

3. **Count characters only**

```bash
wc -m file.txt
```

4. **Count multiple files**

```bash
wc file1.txt file2.txt
```

5. **Use with a command (pipe)**

```bash
ls | wc -l
```

→ Counts how many items are in the current directory.

==========

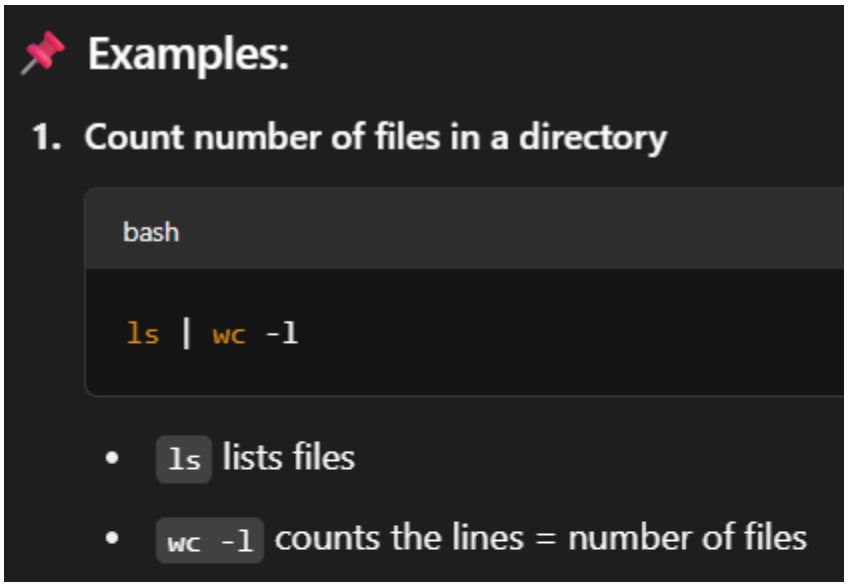
Piping (|) 🚰 — Connect Commands

✅ What is a Pipe?

➔ The **pipe symbol (|)** takes the **output of one command** and sends it as **input to another command**.

🔧 Syntax:

command1 | command2



2. Search inside a file

bash

```
cat file.txt | grep "error"
```

- `cat` shows the file
- `grep` searches for "error"

3. Sort and remove duplicates

bash

```
cat names.txt | sort | uniq
```

4. Show processes and search for `chrome`

bash

```
ps aux | grep chrome
```

==========

< — *Input Redirection* 📥

✅ Basic Syntax:

command < filename

- Runs the command using input from the file instead of from typing manually.

> **Note:**
>
> - This is the **opposite** of `>` (output redirection).
> - `>` writes to a file, while `<` reads from a file.

==========

`sort` Command 🔠— Sort Text Alphabetically or Numerically

✅ Basic Usage:

sort filename.txt

- Sorts lines in **alphabetical (ASCII)** order.

📌 Examples:

1. **Sort a file alphabetically**

```bash
sort names.txt
```

2. **Sort in reverse order**

```bash
sort -r names.txt
```

3. **Sort numerically**

```bash
sort -n numbers.txt
```

(Useful when the file has numbers like 1, 12, 3, etc.)

4. **Sort and remove duplicates**

```bash
sort filename.txt | uniq
```

5. Sort by column (e.g., second word)

```bash
sort -k 2 file.txt
```

🧪 Combine with input redirection:

```bash
sort < file.txt
```

Common Options:

Option	Description
`-r`	Reverse sort
`-n`	Numerical sort
`-k`	Sort by column number
`-u`	Unique (skip duplicates)
`-o`	Output result to a file

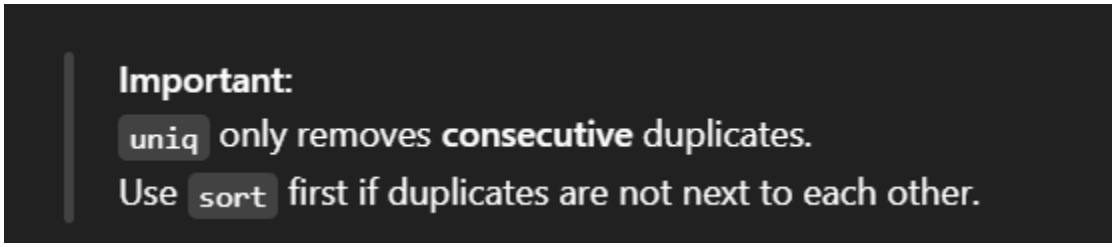
==========

`uniq` *Command* 🔂 *— Filter Out Repeated Lines*

✅ **Basic Usage:**

uniq filename.txt

- Removes **adjacent duplicate lines** from a file.



📌 Examples:

1. **Remove duplicates from a file**

```bash
sort names.txt | uniq
```

2. **Show only duplicated lines**

```bash
sort names.txt | uniq -d
```

3. **Show only unique lines (no duplicates at all)**

```bash
sort names.txt | uniq -u
```

4. **Count occurrences of each line**

```bash
sort names.txt | uniq -c
```

- Output:

- Output:

```
2 Alice
1 Bob
3 Charlie
```

- **Common Options:**

Option	Description
`-d`	Print only duplicates
`-u`	Print only unique lines
`-c`	Print counts with lines
`-i`	Ignore case while comparing

==========

🧠 **What Are Expansions in Linux Shell?**

➔ Expansions are **rules the shell uses to expand or replace text** in commands.

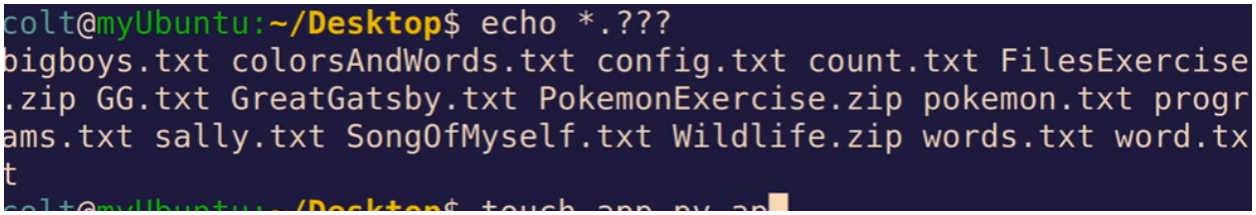
🔹 1. Pathname Expansion (Globbing)

Use wildcards to match filenames.

bash

```
ls *.txt
```

- Lists all `.txt` files
- `*` = any number of characters
- `?` = any single character
- `[a-c]*` = files starting with a, b, or c



? = exact 3 character

🔹 2. Brace Expansion

bash

```
echo file{1..3}.txt
```

Output:

```
file1.txt file2.txt file3.txt
```

Also works like:

bash

```
echo {A,B,C}
```

🔹 3. Tilde Expansion (`~`)

bash

```
cd ~
```

- Expands to your **home directory** (Example: `/home/manthan`)

🔹 4. Variable Expansion

bash

```bash
name="Manthan"
echo "Hello, $name"
```

Output:

```
Hello, Manthan
```

🔹 5. Command Substitution

bash

```bash
echo "Today is $(date)"
```

- Runs `date` and puts its output inside the echo.

🔹 6. Arithmetic Expansion

bash

```bash
echo $((5 + 3))
```

Output:

```
8
```

➔ These expansions make scripting powerful and reduce manual typing.

==========

---

diff - 🆚 Compare Two Files

➔ The **diff** command in Linux is used to **compare the contents of two files line by line**. It shows you the differences between them — very useful for checking what changed.

✅ **Basic Syntax:**

diff file1.txt file2.txt

📌 **Output Meaning:**

The output looks like this:

2c2

< Hello World

> Hello Linux

Explanation:

- Line **2** changed (**c**) in both files
- `<` = line from `file1.txt`
- `>` = line from `file2.txt`

🔧 Options:

Option	Description
`-y`	Side-by-side comparison
`-q`	Report only if files differ
`-c`	Show context around changes
`-u`	Unified format (used in patches)

🧪 Examples:

1. **Quick check: Are files different?**

```bash
diff -q file1.txt file2.txt
```

2. **Side-by-side comparison**

```bash
diff -y file1.txt file2.txt
```

3. **Unified format (for Git)**

```bash
diff -u file1.txt file2.txt
```

4. **Ignore case differences**

```bash
diff -i file1.txt file2.txt
```

==========

`find` *Command* 🔍*— Search for Files and Directories*

✅ Basic Syntax:

find [path] [options] [expression]

📌 Common `find` Examples:

1. **Find a file by name in current directory and subdirectories**

```bash
find . -name filename.txt
```

- `.` means current directory.
- `-name` specifies the file name to look for.

2. **Find a file by name (case-insensitive)**

```bash
find . -iname filename.txt
```

3. **Find all `.txt` files**

```bash
find . -name "*.txt"
```

4. **Find a directory**

```bash
find . -type d -name "foldername"
```

5. **Find files by size (e.g., files larger than 10MB)**

```bash
find . -size +10M
```

6. **Find files modified in the last 7 days**

```bash
find . -mtime -7
```

7. **Find and delete files**

⚠️ Use with care!

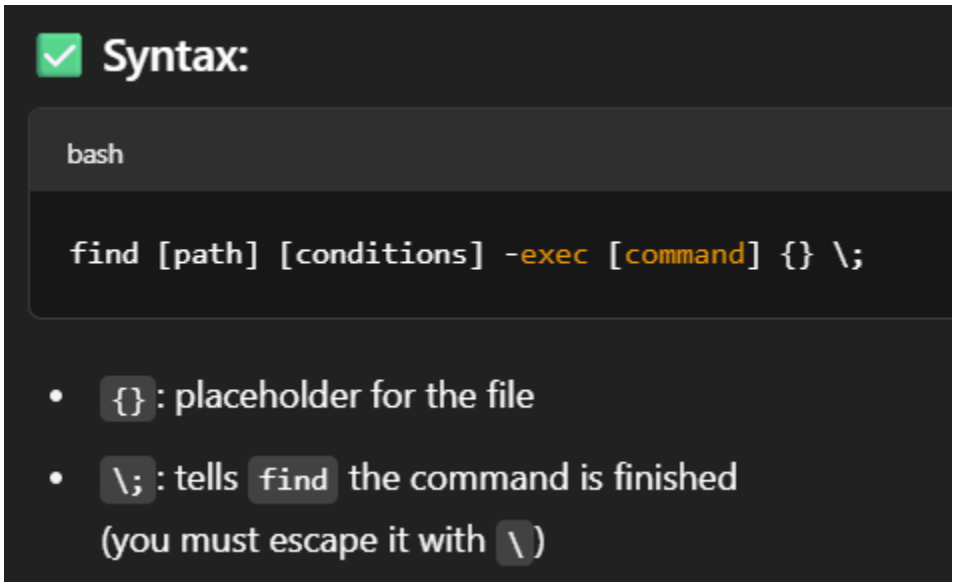
```bash
find . -name "*.log" -delete
```

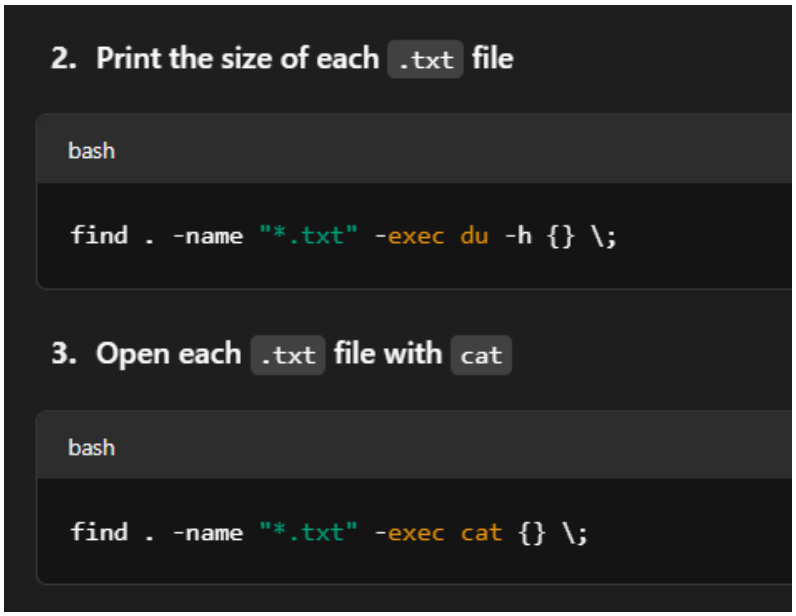
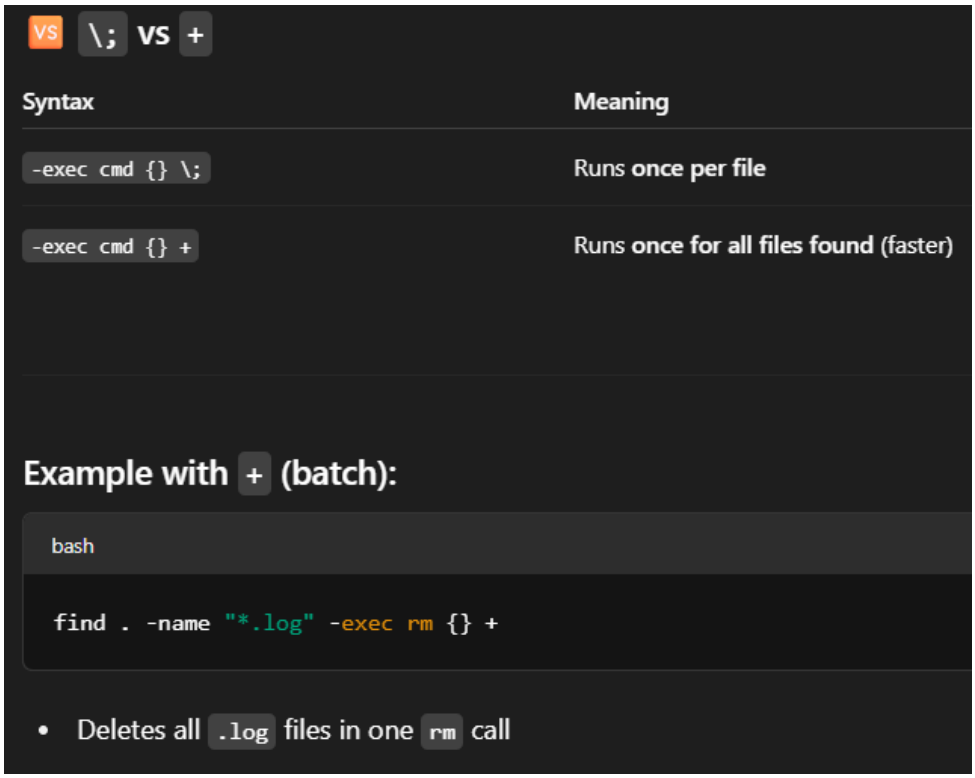
- `find /home -name "*.txt"` - Find all .txt files in /home
- `find . -type d -name "dir*"` - Find directories starting with "dir"

==========

What is `-exec` in `find`?

- -exec tells find to **execute a command** on each matching file.
- It uses {} as a placeholder for each file.
- It must end with either \; (one file at a time) or + (batch mode).



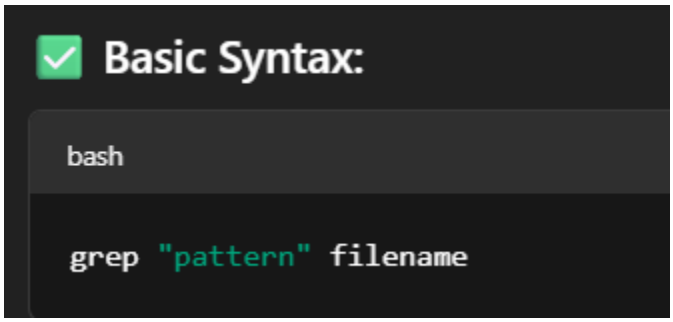
- 
2. Print the size of each .txt file
bash
find . -name "*.txt" -exec du -h {} \;
3. Open each .txt file with cat
bash
find . -name "*.txt" -exec cat {} \;
- 
VS \; vs +
Syntax
Meaning
-exec cmd {} \;
Runs once per file
-exec cmd {} +
Runs once for all files found (faster)
Example with + (batch):
bash
find . -name "*.log" -exec rm {} +
Deletes all .log files in one rm call

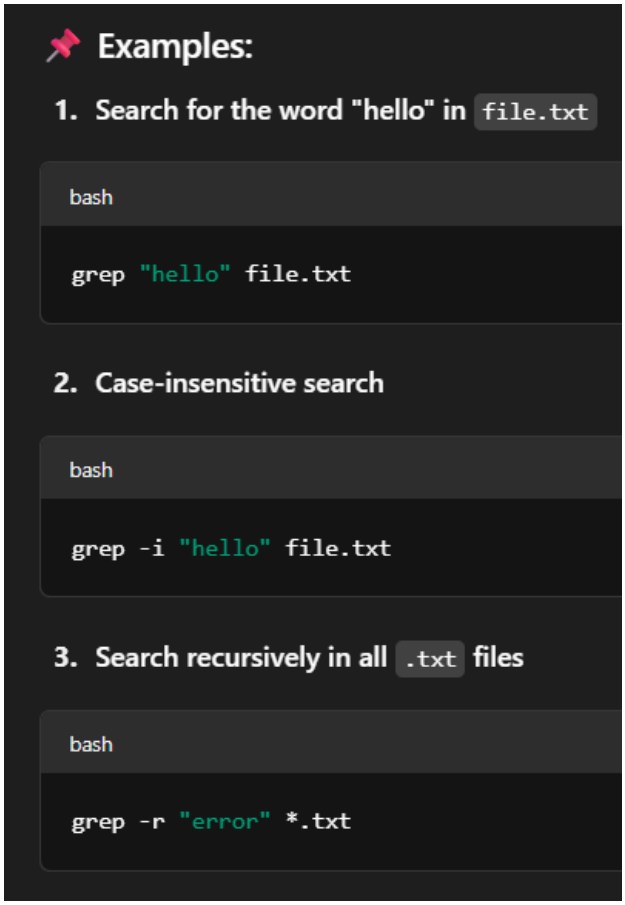
==========

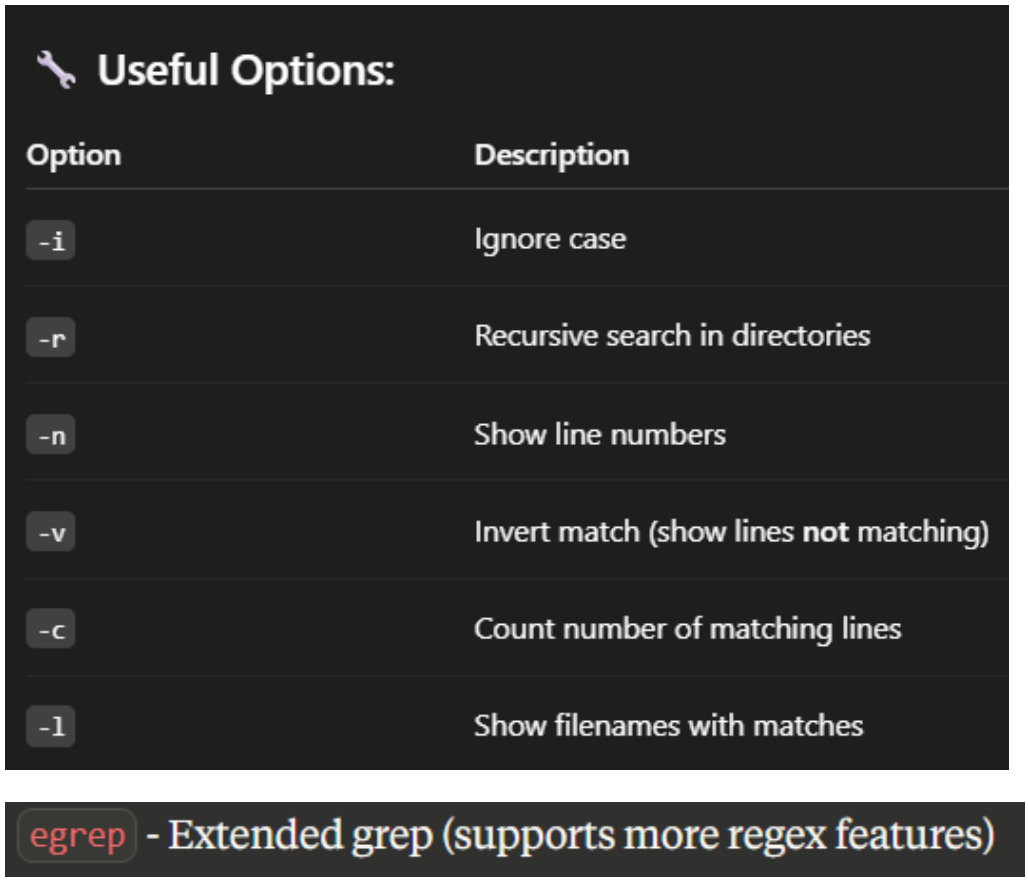
🔍 *What is `grep`?*

➔ grep stands for **Global Regular Expression Print**.

➔ It searches for **lines matching a pattern**.

➔ 

➔ 



==========

🔍 *What is `locate`?*

- `locate` searches for files **much faster** than `find`
- It uses a **database** built by updatedb
- It searches **file paths**, not file contents

✅ Basic Syntax:

locate [pattern]

📌 Examples:

1. Find all files with "config" in the name

```bash
locate config
```

2. Find all `.jpg` files

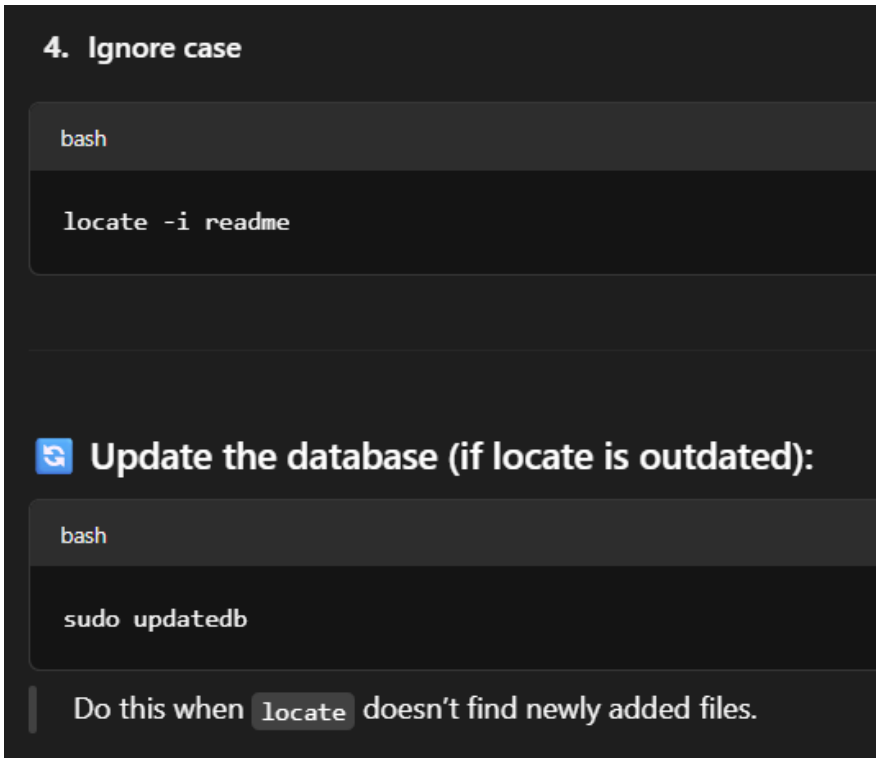
```bash
locate "*.jpg"
```

(Note: `locate` doesn't need wildcards in quotes, but it's fine to use)

3. Search for exact filename

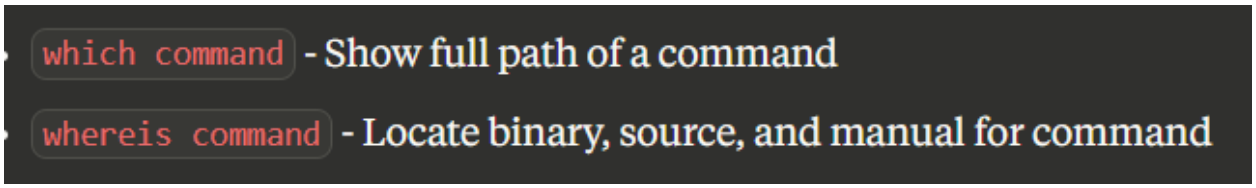
```bash
locate -b '\bashrc'
```

> Finds files ending with `bashrc`



==========

Which and whereis command



==========

System Information commands

💻 *uname -a*

Show system information

uname -a

➔ Displays all system details like kernel name, version, architecture, etc.

✅ Example output: Linux hostname 5.15.0-60-generic x86_64 ...

💻 *hostname*

Show the system's hostname

hostname

➔ Prints the computer’s network name.

👤 *whoami*

Show current username

whoami

➔ Tells which user is currently logged in.

🧾 *id*

Show user ID, group ID, and groups

id

➔ Example: uid=1000(manthan) gid=1000(manthan) groups=...

🗓️ *date*

Show current date and time

date

➔ Example: Tue May 7 16:32:10 IST 2025

⏱️ *uptime*

Show how long the system has been running

uptime

➔ Example: 16:32:10 up 3 days, 2:10, 2 users, load average: 0.10, 0.20, 0.15

👥 *w*

Show who is logged in and what they are doing

w

➔ Lists active users, login times, and their processes.

👤 *who*

Show who is logged in

who

➔ Basic info: user name, terminal, login time.

🕐 *last*

Show the login history of users

last

➔ Lists recent logins from system log.

==========

==========

Resource Monitoring

📈 `top` *command – Monitor system processes and resources*

✅ **Basic Usage:**

top

➔ This opens an interactive screen showing live updates of:

- ◆ CPU usage
- ◆ Memory usage
- ◆ Running processes
- ◆ Load average
- ◆ Uptime

🖥️ Key Columns Explained:

Column	Description
`PID`	Process ID
`USER`	Owner of the process
`%CPU`	CPU usage
`%MEM`	RAM usage
`TIME+`	Total CPU time used by the process
`COMMAND`	The command running the process

🛠️ Example: Kill a process

1. Press `k` inside `top`
2. Enter PID
3. Confirm with Enter

🔄 Update Interval:

`top` refreshes every 3 seconds by default

You can change it by pressing `d` inside `top`.

🎯 *What is h-top?*

➔ h-top is an **interactive process viewer** for Linux — it shows real-time system stats like CPU, memory, processes, and allows **easy navigation** and **process control**.

🆚 Compared to top, h-top has:

- A colorful, clean interface
- Easy-to-use arrow key navigation
- Function keys for actions (like kill, sort, search)
- Tree view of processes

✅ Basic Usage

htop

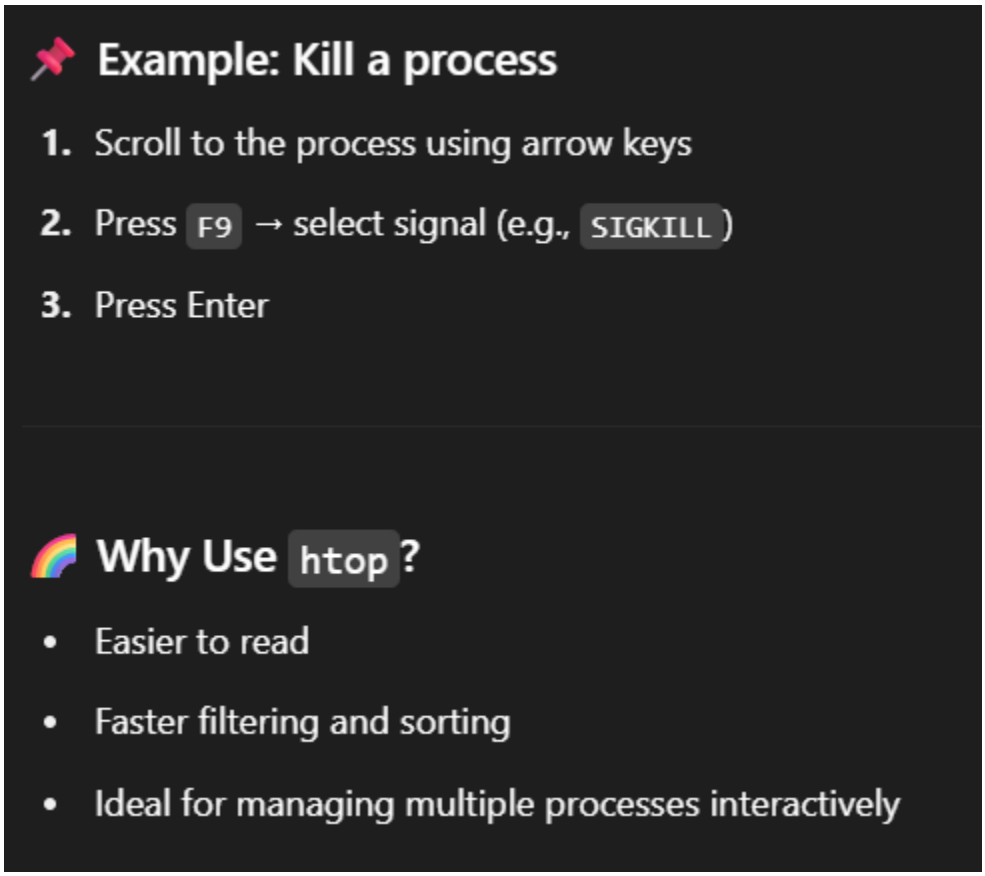
- (If not installed: sudo apt install htop)

- ### 🖥️ What You See:

Section	Description
CPU bar(s)	Per-core usage
Mem/Swap bars	RAM and swap usage
Load average	1, 5, and 15 minute averages
Tasks	Total running/sleeping processes
PID list	Like `top`, but scrollable and colored

- ## ⌨️ Keyboard Shortcuts:

Key	Action
`F1`	Help
`F2`	Setup (customize layout)
`F3`	Search for a process
`F4`	Filter processes
`F5`	Tree view (process hierarchy)
`F6`	Sort by column (CPU, MEM, etc)
`F9`	Kill selected process
`F10`	Quit `htop`

- 

💾 `free` *command – View system memory usage*

✅ Basic Syntax:

free

📌 Common Output Columns:

Column	Description
`total`	Total installed memory
`used`	Memory in use
`free`	Completely unused memory
`shared`	Memory used by tmpfs (shared memory)
`buff/cache`	Buffers + cache by the system
`available`	Memory available for new programs

📊 Example Output:

bash

```
               total        used        free      shared  buff/cache   available
Mem:        16385456     2789460     9474164      181652     4121840    12978588
Swap:        2097148           0     2097148
```

🚀 Useful Options:

1. Show in human-readable format (MB/GB):

bash

```
free -h
```

2. Show in MB:

bash

```
free -m
```

3. Show in GB:

bash

```
free -g
```

4. Watch memory usage live (update every second):

bash Copy

```
watch free -h
```

🧠 Memory Tip:

Don't panic if `used` is high — Linux uses free memory for **cache**, which is released when needed.

💽 *df command – Report disk space usage*

✅ Basic Usage:

df

➔ Shows disk usage of all mounted file systems (in 1K blocks by default).

📊 Common Output Columns:

Column	Description
`Filesystem`	Name of the disk or partition
`1K-blocks`	Total size in 1K blocks
`Used`	Space used
`Available`	Space free
`Use%`	Percentage of space used
`Mounted on`	Mount point (location in directory)

🔍 Human-readable format (recommended):

```bash
df -h
```

`-h` stands for "human-readable" (shows in KB, MB, GB)

📌 Example Output:

```bash
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda1       100G   40G   55G  42% /
tmpfs            16G  1.0M   16G   1% /dev/shm
```

🛠️ Other useful options:

- **Show only a specific filesystem:**

```bash
df -h /home
```

- **Include all file systems (even 0-size ones):**

```bash
df -ha
```

📦 *du command – Disk usage of files and directories*

✅ Basic Usage:

du

➔ Shows the size of the current directory and all subdirectories (in bytes by default).

📌 Common and Useful Options:

Command	Description
`du -h`	Human-readable format (KB, MB, GB)
`du -sh`	Summary only of current directory (human-readable)
`du -sh *`	Size of each file/folder in the current directory
`du -a`	Show sizes of **all files** and directories
`du -d 1 -h`	Limit to 1 directory depth

📄 Example:

bash

```bash
du -sh *
```

> Output:

mathematica

```mathematica
120K    documents
2.3M    music
4.0K    todo.txt
```

📁 Example: Check size of `/var/log`

bash

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

🔁 Combine with `sort` to find largest folders:

bash

```bash
du -sh * | sort -hr
```

⚙️ *ps command – Process Status*

➔ The ps command lists the currently running processes for the current user or system.

❖ ✅ **Basic Usage:**

ps

➔ Shows processes for the **current shell/session** only.

📊 Common Options:

Command	Description
`ps -e`	Show all processes
`ps -ef`	Full-format listing of all processes
`ps aux`	Detailed view (BSD-style output)
`ps -u username`	Show processes for a specific user
`ps -p <PID>`	Show details of a process by PID

📌 Example Output (`ps -ef`):

swift

```
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 08:00 ?        00:00:01 /sbin/init
manthan   1256  1221  0 08:02 pts/0    00:00:00 bash
manthan   1278  1256  0 08:03 pts/0    00:00:00 ps -ef
```

🔍 Explanation of Columns:

Column	Description
UID	User who owns the process
PID	Process ID
PPID	Parent process ID
CMD	Command that started the process

🔄 Combine with `grep` to find a specific process:

bash

```
ps aux | grep firefox
```

- `ps` - Show running processes
  - `ps aux` - Show all processes in BSD format
  - `ps -ef` - Show all processes in standard format

🔍 `pgrep` – *Find Process IDs (PIDs) by name*

Syntax:

```bash
pgrep <pattern>
```

Example:

```bash
pgrep firefox
```

> Returns the **PID(s)** of any running process whose name contains "firefox".

Options:

- `-u <user>` – Match processes for a specific user.
- `-l` – Show the process name with PID.
- `-f` – Match full command line (not just name).

📌 Example:

```bash
pgrep -fl python
```

🔫 *pkill – Kill processes by name (pattern)*

Syntax:

```bash
pkill <pattern>
```

Example:

```bash
pkill firefox
```

> Kills all processes matching "firefox".

Options:

- `-u <user>` – Kill only user's processes.
- `-f` – Match the full command line.
- `-signal` – Send a specific signal (default is SIGTERM).

📌 Example:

```bash
pkill -9 node
```

> Sends **SIGKILL** to all `node` processes.

🛑 **Warning:**

Be **very careful** with `pkill` — it doesn't ask for confirmation.

💀 *kill – Send signal to a process (usually to terminate it)*

🔹 Basic Syntax:

bash

```
kill <PID>
```

This sends **SIGTERM (signal 15)** by default, which politely asks the proce

🔹 Example:

bash

```
kill 1234
```

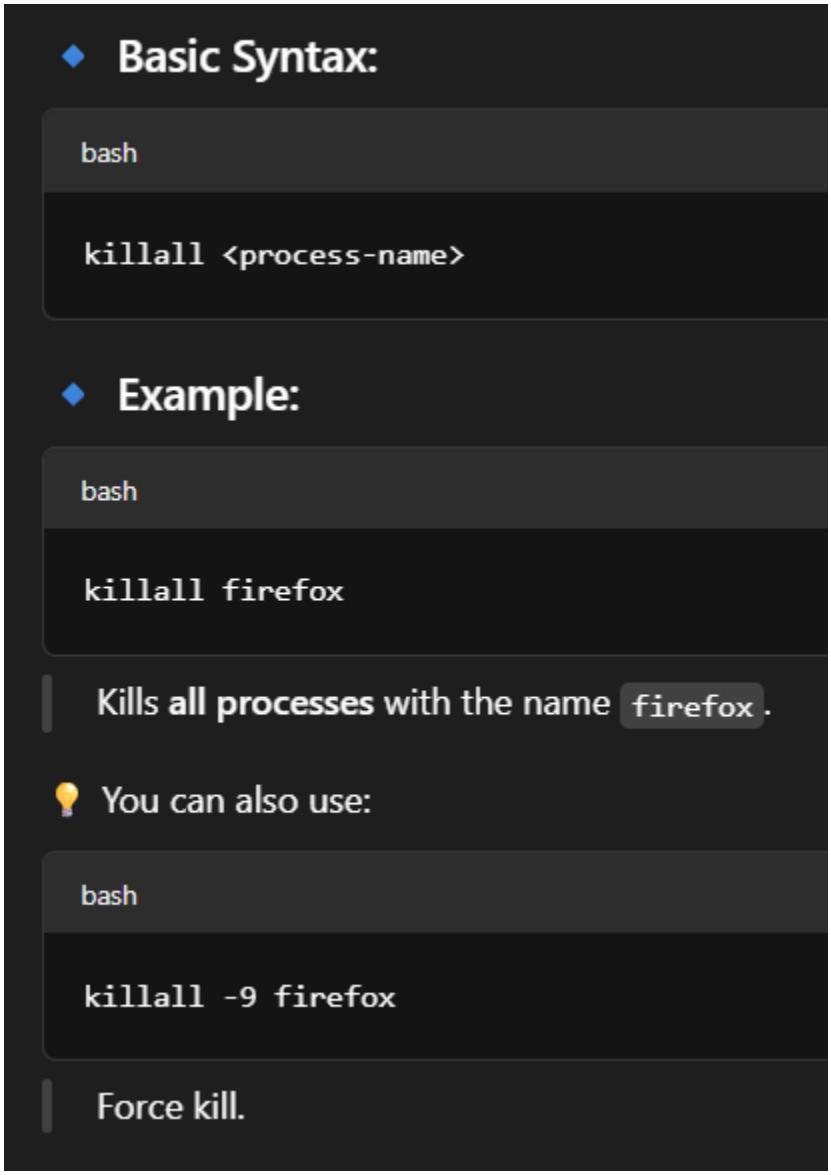
💥 Force Kill:

bash

```
kill -9 1234
```

Sends **SIGKILL (signal 9)** – **forcefully kills** the process immediately.

🔫 *killall – Kill processes by name*

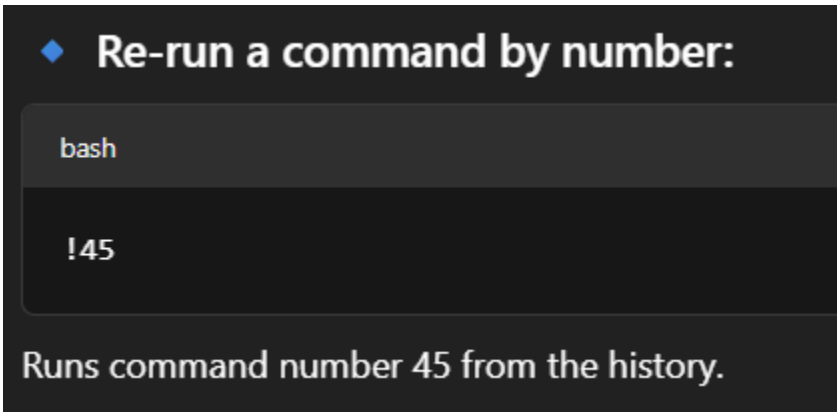


📜 `history` – *View command history*

- **Basic Syntax:**

history

➔ Shows your previous commands with line numbers.



🆘 *help – Built-in help for shell (bash) commands*

- **Syntax:**

help <command>

==========

==========

Package Management

📦 What is Package Management?

➔ Package management lets you:

- Install software
- Update or upgrade software
- Remove software
- Handle dependencies

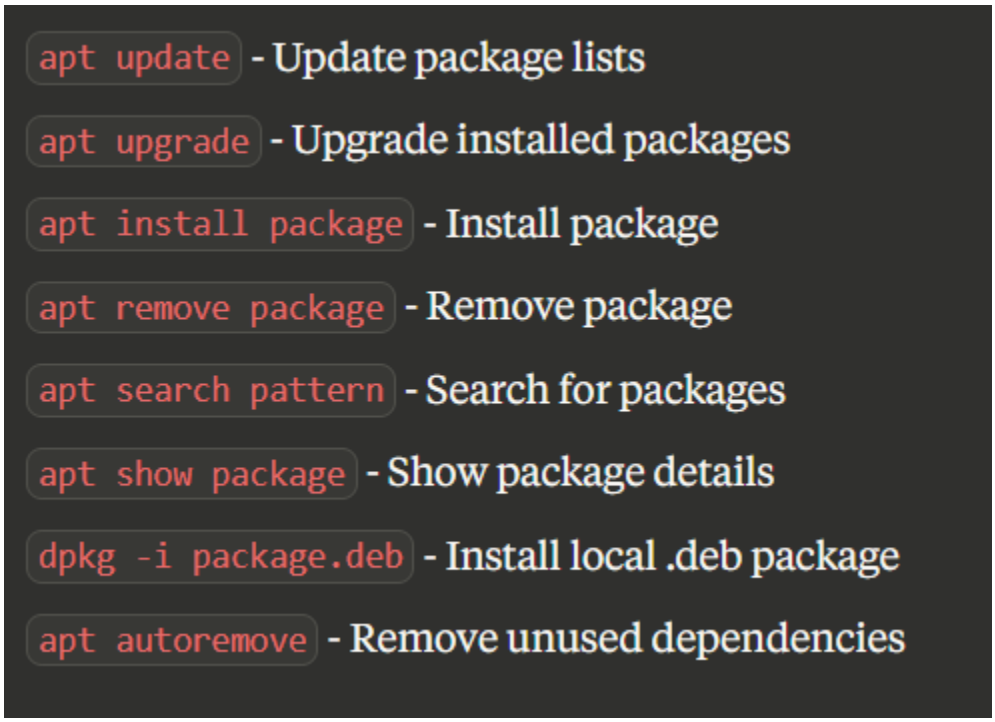
Different Linux distributions use different **package managers**.

📚 Package Managers by Distribution:

Distribution	Package Manager	Example Commands
Debian/Ubuntu	apt / dpkg	apt install, dpkg -i
Red Hat/CentOS/Fedora	yum / dnf / rpm	dnf install, rpm -i
Arch Linux	pacman	pacman -S

🛠️ Common `apt` (Debian/Ubuntu) Commands

Task	Command
Update package list	sudo apt update
Upgrade all packages	sudo apt upgrade
Install a package	sudo apt install <package>
Remove a package	sudo apt remove <package>
Completely remove (+config)	sudo apt purge <package>
Search for a package	apt search <package>
Show package info	apt show <package>
Clean cache	sudo apt clean



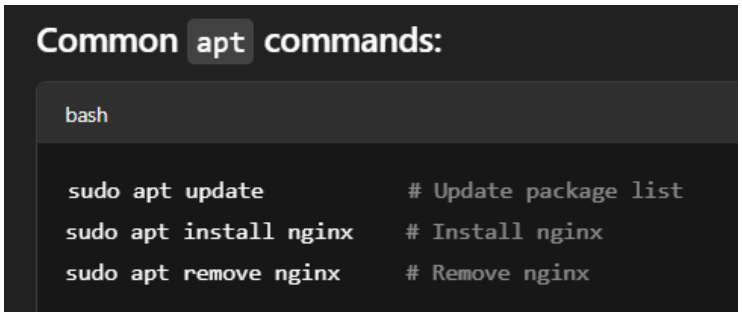
🧱 `dpkg` – Low-level tool (used by `apt`)

Task	Command
Install .deb	sudo dpkg -i file.deb
Remove a package	sudo dpkg -r <package>
List installed	dpkg -l
Find file owner	dpkg -S /path/to/file

📦 What is apt?

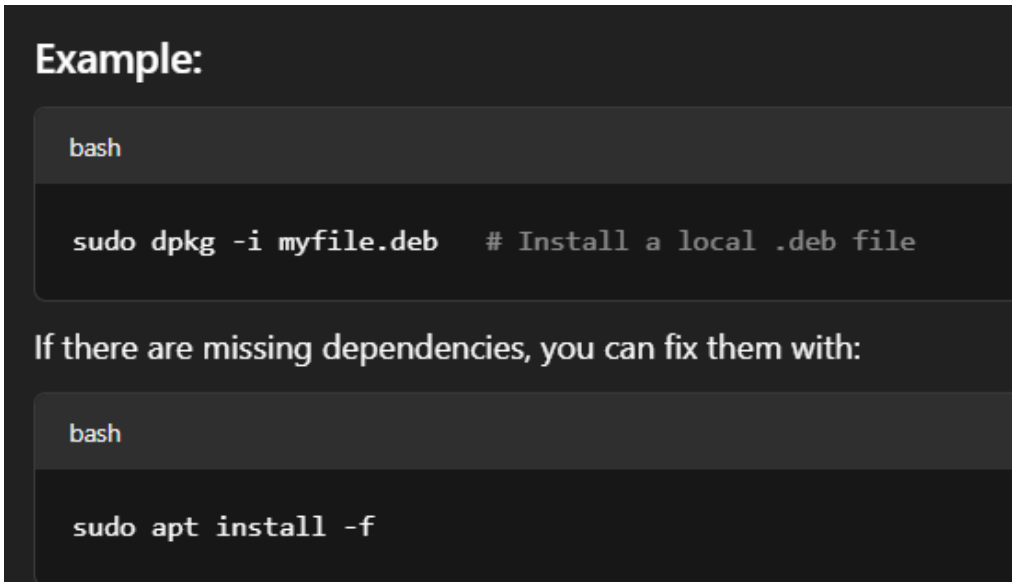
- apt = **Advanced Package Tool**
- It's a user-friendly command-line tool for managing software on **Debian-based systems** (like Ubuntu).
- It downloads, installs, upgrades, and removes software from online repositories.

Common apt commands:



What is dpkg?

- dpkg = **Debian Package**
- It's a **low-level tool** used to install, remove, and manage .deb files **manually**.
- It does **not automatically handle dependencies**.



📁 What is .deb?

- .deb is a **Debian software package file** (like .exe on Windows or .apk on Android).
- You can download .deb files from websites and install them using dpkg or apt.

==========

==========

User Management

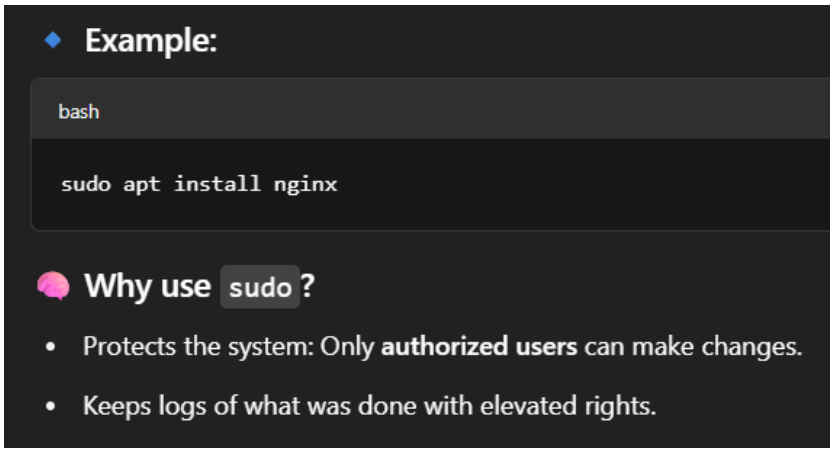
🔑 sudo – Superuser Access

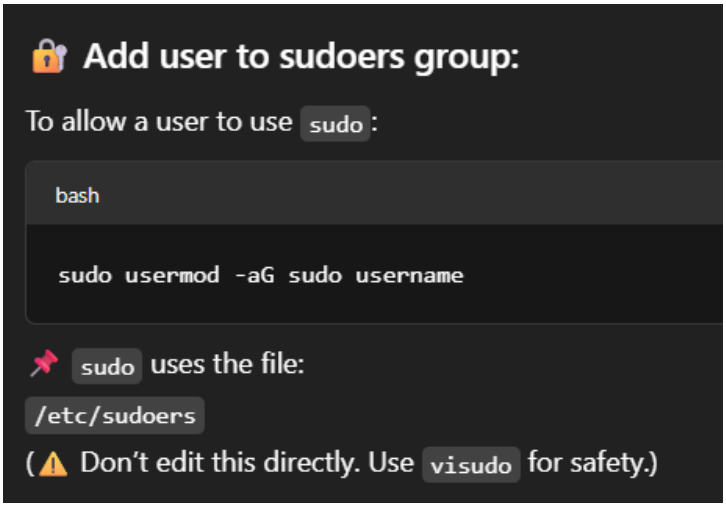
✅ **What it does:**

- Allows a **regular user** to perform **admin-level tasks** (like install software, manage users, edit system files, etc.).

🔹 **Syntax:**

sudo <command>

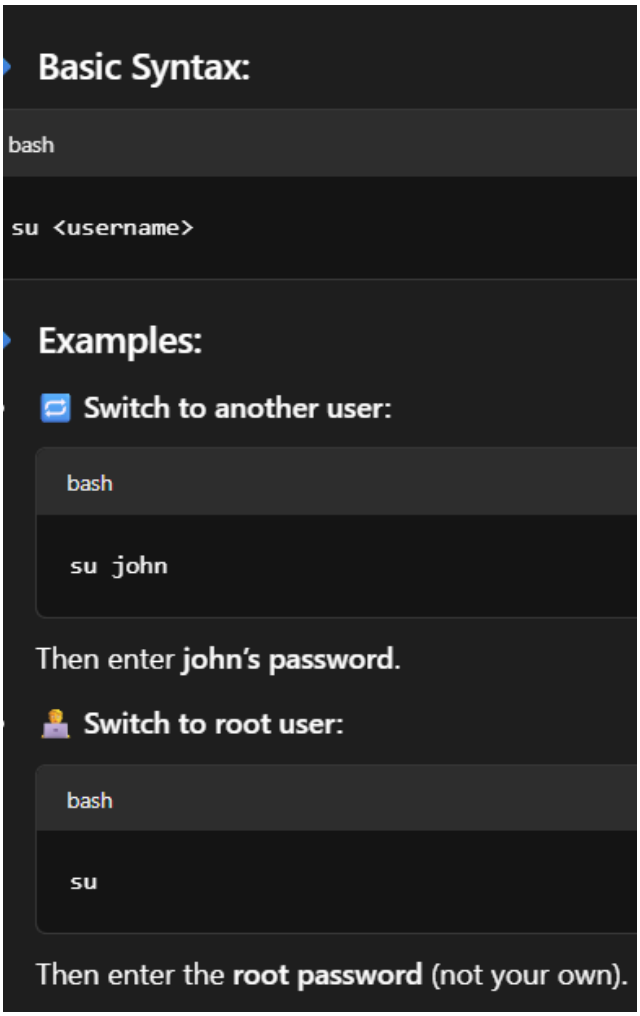


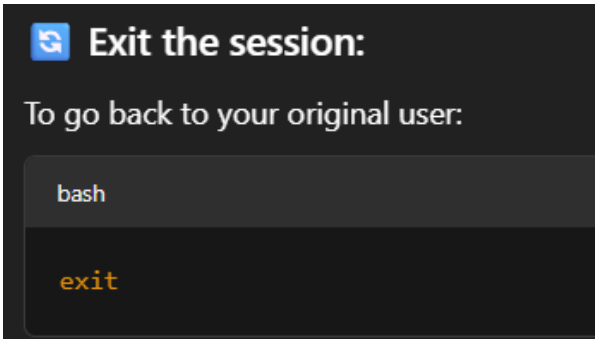


👤 **su – Switch User**

✅ What it does:

- Allows you to **switch to another user account**, including the **root** user.





🔍 `su` vs `sudo`:

Feature	`su`	`sudo`
Access	Requires **target user's password**	Uses **your password** (if authorized)
Scope	Full shell session	One command at a time
Logging	No log by default	Actions are logged

🔐 passwd – Change a User's Password

✅ What it does:

- Sets or changes the **password** of any user.

🔹 Usage:

```bash
passwd              # Change **your** password
sudo passwd <user>  # Change another user's password
```

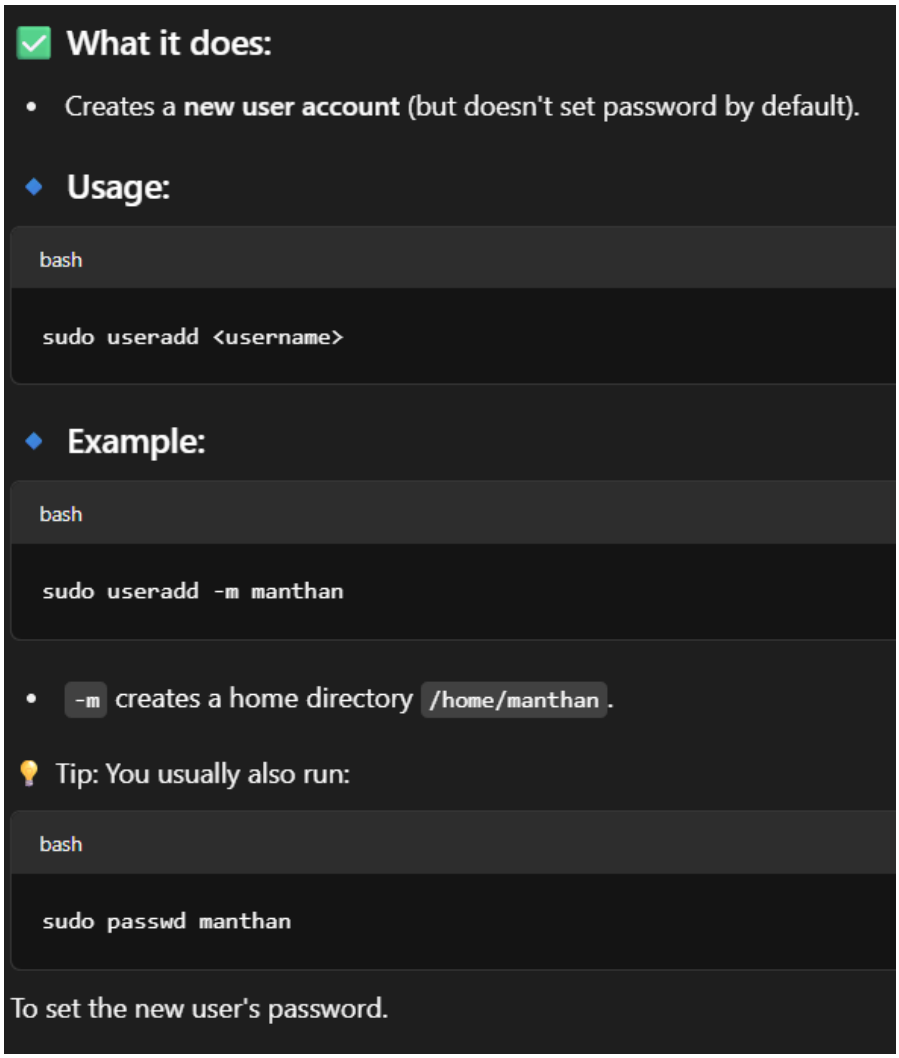
🔹 Example:

```bash
sudo passwd manthan
```

👤 useradd – Add a New User (Low-level)

✅ What it does:

- Creates a **new user account** (but doesn't set password by default).



🗑️ userdel – Delete a User

✅ What it does:

- Deletes a user account.

✏️ usermod – Modify a User Account

✅ What it does:

- Updates user settings (like shell, groups, username, etc.).

🔹 Common usage:

bash

```
sudo usermod -aG <group> <user>  # Add to group
sudo usermod -l newname oldname  # Rename user
sudo usermod -d /new/home user   # Change home directory
```

🔹 Example:

bash

```
sudo usermod -aG sudo manthan   # Add manthan to sudo group
```

👥 What is a Group in Linux?

➔ A **group** is a way to manage **permissions** for multiple users at once.

➔ Instead of setting file access for each user, you assign permissions to a **group**, and add users to that group.

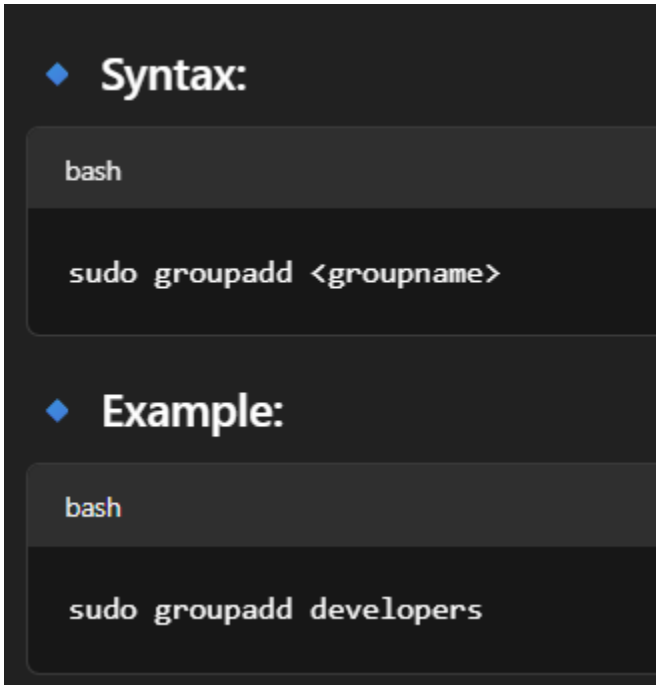
🔑 Why groups?

Easier permission management.

Example: All developers in the devs group can edit /projects.

🧩 groupadd – Create a New Group

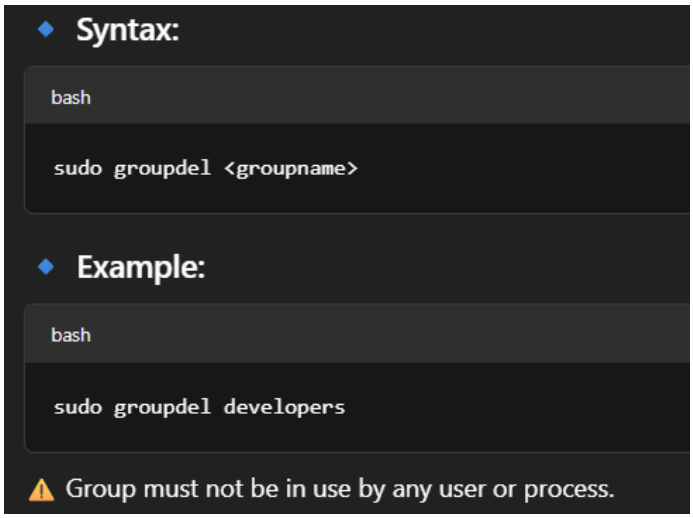
✅ What it does:

- Creates a new group on the system.
- 

🧹 groupdel – Delete a Group

✅ What it does:

- Deletes an existing group.

- 

👥 groups – Show Group Membership

✅ What it does:

- Shows which groups a user belongs to.

==========

==========

==========

File Permissions

==========

🔐 What Are Permissions in Linux?

- Permissions define **who can read, write, or execute** a file or directory.
- Every file/directory in Linux has **3 types of users** and **3 types of permissions**.

👥 Types of Users:

User Type	Meaning
Owner	The person who created the file
Group	Users who are in the same group
Others	Everyone else on the system

✅ **Types of Permissions:**

Permission	Symbol	Meaning
Read	r	View file content / list directory
Write	w	Edit or delete content
Execute	x	Run file (script/program)

==========

🔎 **How Permissions Look (in ls -l output):**

-rwxr-xr-- 1 manthan devs file.sh

Breakdown:

Field	Meaning
-	Type (- for file, d for directory)
rwx	**Owner's** permissions: read, write, execute
r-x	**Group's** permissions: read, execute
r--	**Others'** permissions: read only

❓ **Why Use Permissions?**

- 🔒 **Security**: Prevent unauthorized access or changes.
- 👫 **Collaboration**: Share files safely within groups.
- 🚫 **Accident prevention**: Stop users from deleting/modifying critical files.

==========

🔧 **chmod – Change File Permissions**

✅ **What it does:**

➔ Changes who can **read (r)**, **write (w)**, and **execute (x)** a file or folder.

🔹 ***1. Symbolic Method (r, w, x)***

🔸 **Syntax:**

chmod [who]+/-=[permission] <filename>

Who:

Symbol	User Type
u	User (owner)
g	Group
o	Others
a	All (u+g+o)

Operators:

Symbol	Meaning
+	Add permission
-	Remove permission
=	Set exact permission

- ***2. Numeric Method (Octal)***

➔ Each permission has a number:

➔

Permission	Value
Read (r)	4
Write (w)	2
Execute (x)	1

Format: `OWNER GROUP OTHERS`

Example: `chmod 754 file.sh`

User	Value	Meaning
Owner	7	rwx
Group	5	r-x
Others	4	r--

➔

bash

```
chmod 755 script.sh
```

➔

✅ **Check Result:**

bash

```
ls -l
```

- `chmod permissions filename` - Change file permissions
  - `chmod 755 filename` - Set rwx for owner, rx for group and others
  - `chmod +x filename` - Add execute permission for all
  - `chmod -R 755 directory` - Recursively change permissions

==========

🧑‍💼 chown – Change File Owner or Group

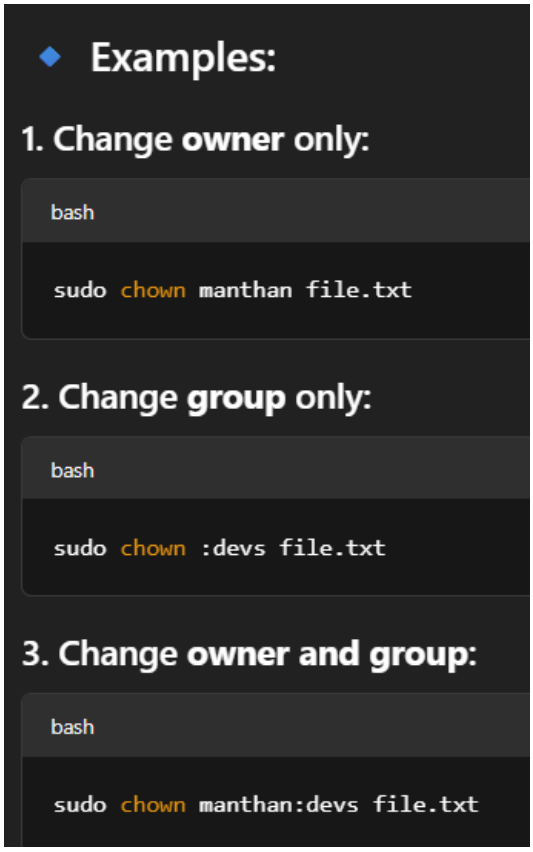
✅ What it does:

➔ Changes the **user** and/or **group** that owns a file or folder.

🔹 Syntax:

sudo chown [user][:group] <filename>

- user → new owner
- group → new group (optional)
- : → separates user and group



🔹 Recursive option:

To change ownership for all files inside a folder:

bash

```bash
sudo chown -R manthan:devs /myfolder
```

🔍 Check Ownership:

bash

```bash
ls -l
```

Shows:

css

```
-rw-r--r-- 1 manthan devs file.txt
```

==========

==========

Command Line Shortcuts

- Ctrl+C - Interrupt (kill) current process
- Ctrl+Z - Suspend current process
- Ctrl+D - Exit current shell/EOF
- Ctrl+L - Clear screen
- Ctrl+A - Move cursor to beginning of line
- Ctrl+E - Move cursor to end of line
- Ctrl+U - Cut from cursor to beginning of line
- Ctrl+K - Cut from cursor to end of line
- Ctrl+W - Cut word before cursor
- Ctrl+Y - Paste previously cut text
- Tab - Auto-complete commands or filenames
- !! - Repeat last command
- !string - Repeat last command starting with string
- !$ - Last argument of previous command
- Alt+. or Esc+. - Insert last argument of previous command

==========

==========

System Operations

- `shutdown now` - Shutdown system immediately
- `shutdown -r now` - Reboot system immediately
- `reboot` - Reboot system
- `halt` - Halt system
- `systemctl start service` - Start systemd service
- `systemctl stop service` - Stop systemd service
- `systemctl restart service` - Restart systemd service
- `systemctl status service` - Check service status
- `systemctl enable service` - Enable service at boot
- `systemctl disable service` - Disable service at boot
- `journalctl` - Query systemd journal
  - `journalctl -u service` - Show logs for specific service
  - `journalctl -f` - Follow new log entries

==========

==========

Compression and Archives

- tar -cf archive.tar files - Create tar archive
- tar -xf archive.tar - Extract tar archive
- tar -czf archive.tar.gz files - Create compressed tar archive (gzip)
- tar -xzf archive.tar.gz - Extract compressed tar archive (gzip)
- tar -cjf archive.tar.bz2 files - Create compressed tar archive (bzip2)
- tar -xjf archive.tar.bz2 - Extract compressed tar archive (bzip2)
- gzip file - Compress file with gzip
- gunzip file.gz - Uncompress gzip file
- zip archive.zip files - Create zip archive
- unzip archive.zip - Extract zip archive

==========

==========

Text Editors

📝 **nano – Simple Text Editor**

✅ **What it does:**

➔ Opens a file in a terminal-based text editor so you can **create**, **edit**, and **save** files.

🔹 **Syntax:**

nano <filename>

🔹 Example:

bash

```
nano notes.txt
```

- If the file **does not exist**, it will be created.
- If the file **exists**, it will open for editing.

📚 Basic `nano` Commands (shown at bottom of the screen):

Command	Action
`Ctrl + O`	Write (save) the file
`Ctrl + X`	Exit nano
`Ctrl + K`	Cut current line
`Ctrl + U`	Paste line (after cut)
`Ctrl + W`	Search inside file
`Ctrl + G`	Help menu (full commands)

==========

🧱 vi – Visual Text Editor

✅ What it does:

➔ Opens a file in a terminal-based editor with **modes** for navigating, editing, and saving.

- **Syntax:**

vi <filename>

- **Example:**

vi notes.txt

🚦 Modes in `vi`:

Mode	Purpose	How to Enter
Normal Mode	For moving, deleting, copying	Default when you open `vi`
Insert Mode	For typing/editing text	Press `i`, `a`, or `o`
Command Mode	For saving/exiting/searching	Press `:` from Normal mode

✍️ Common Workflow:

1. **Open a file:**

```bash
vi myfile.txt
```

2. **Press** `i` to enter Insert Mode and start typing.
3. **Press** `Esc` to return to Normal Mode.
4. Type `:w` to save, or `:q` to quit:
   - `:w` → Save
   - `:q` → Quit
   - `:wq` or `ZZ` → Save and quit
   - `:q!` → Quit without saving

🔥 Common vi Commands (in Normal mode):

Command	Description
`i`	Insert before cursor
`a`	Insert after cursor
`o`	New line below and insert
`dd`	Delete current line
`yy`	Copy current line
`p`	Paste below
`u`	Undo
`/text`	Search for "text"
`n`	Repeat search

==========

⚡ vim – Vi IMproved Editor

✅ What it does:

➔ A terminal-based text editor with syntax highlighting, multi-level undo, plugins, and many enhancements over `vi`.

- **Syntax:**

vim <filename>

- **Example:**

vim myfile.txt

🚦 Modes in `vim` (same as `vi`):

Mode	Purpose	Enter With
Normal	Navigate/edit commands	Press `Esc`
Insert	Type text	Press `i`, `a`, or `o`
Command	Save/quit/search etc.	Press `:`

🔧 Useful Vim Commands

🔸 In **Normal Mode**:

Command	Action
`i`	Insert before cursor
`a`	Append after cursor
`o`	Open new line below
`dd`	Delete (cut) current line
`yy`	Copy current line
`p`	Paste copied/cut line
`u`	Undo
`Ctrl + r`	Redo
`/text`	Search for `text`

🔸 In **Command Mode** (`:`):

Command	Action
`:w`	Save
`:q`	Quit
`:wq`	Save and quit
`:q!`	Quit without saving
`:set number`	Show line numbers