📂 Top-level Linux Directories and Their Purpose

🔹 `/`

- The **root directory** — everything starts here.
- All other files and directories exist under `/`.

🔹 `/bin`

- **Essential user binaries (commands)** required for all users.
- Examples: `ls`, `cp`, `mv`, `cat`, `bash`.
- Needed even if no other filesystems are mounted.

🔹 `/sbin`

- **System binaries** (mostly for system administration).
- Examples: `ifconfig`, `mount`, `shutdown`.
- Only root/admin usually runs these.

🔹 `/etc`

- **System configuration files** (text-based).
- Example: `/etc/passwd` (user accounts), `/etc/ssh/sshd_config` (SSH config), `/etc/fstab` (mount info).
- Think of `/etc` as the **settings folder** of the OS.

🔹 `/var`

- **Variable data** that changes frequently.
- Examples:
    - `/var/log` → system & app logs
    - `/var/spool` → print/mail queues
    - `/var/tmp` → temporary files that survive reboot

🔹 `/lib` and `/lib64`

- **Shared libraries** needed by programs in `/bin` and `/sbin`.
- Similar to DLLs on Windows.
- Example: `/lib/x86_64-linux-gnu/libc.so.6`.

🔹 `/usr`

- **User applications and utilities** (not essential for boot, but for normal usage).
- Contains programs, libraries, docs.
- Inside `/usr`:
    - `/usr/bin` → extra commands for users (`vim`, `python`, `git`)
    - `/usr/sbin` → extra system admin commands (`apache2`, `nginx`)
    - `/usr/lib` → libraries for above binaries
    - `/usr/share` → shared files (icons, docs, man pages)

🔹 **`/opt`**

- **Optional software** (usually third-party apps).
- Example: if you install Google Chrome manually, it may go in `/opt/google/chrome`.

🔹 **`/home`**

- **User home directories**.
- Example: `/home/vikram` for your files, downloads, configs.
- Like "C:\Users" in Windows.

🔹 **`/root`**

- The **home directory of the root user**.
- Do not confuse with `/`.

🔹 **`/tmp`**

- **Temporary files**, deleted after reboot.
- All users and apps use it for scratch space.

🔹 **`/boot`**

- Files needed for booting Linux.
- Example: Kernel (`vmlinuz`), initramfs, bootloader (GRUB config).

🔹 **`/dev`**

- **Device files** (special files that represent hardware).
- Example: `/dev/sda` (disk), `/dev/tty` (terminal), `/dev/null`.

🔹 `/mnt`

- **Temporary mount point** for mounting filesystems (manual usage).
- Example: If you mount a USB drive manually, you might use `/mnt/usb`.

🔹 `/media`

- **Automatic mount point** for removable devices.
- Example: Plugging in a USB drive → `/media/vikram/USB`.

🔹 `/proc`

- **Virtual filesystem** that provides info about running processes and kernel.
- Example: `/proc/cpuinfo`, `/proc/meminfo`, `/proc/1234/` (process with PID 1234).
- It's not real files, just runtime info.

🔹 `/sys`

- Another **virtual filesystem** to interact with the kernel and devices.
- Example: `/sys/class/net/` shows network interfaces.

🔹 `/srv`

- **Service data** served by the system.
- Example: web server files may be stored in `/srv/www`.

🔹 `/run`

- Stores **runtime process data** since last boot.
- Example: PID files (`/run/nginx.pid`).
- Cleared on reboot.

📌 Summary (Easy Mapping)

- `/bin` → Basic commands
- `/sbin` → System commands
- `/etc` → Configs
- `/var` → Logs, caches, variable files
- `/lib` → Libraries
- `/usr` → Extra software
- `/opt` → Third-party apps
- `/home` → User files
- `/root` → Root user’s home
- `/tmp` → Temporary files

- `/boot` → Boot loader & kernel
- `/dev` → Devices
- `/mnt`, `/media` → Mount points
- `/proc`, `/sys` → Kernel & process info
- `/srv` → Service data
- `/run` → Runtime info

👉 Now, in companies (MNCs), when they deploy apps, they usually:

- Put **config** files in `/etc/appname/`
- Put **binaries** in `/usr/bin` or `/opt/appname/`
- Put **logs** in `/var/log/appname/`
- Keep **data** in `/var/lib/appname/`

🏗️ Best Way to Install 3rd Party Software with Docker Compose

1. Use `/srv` for all deployments

- Why: `/srv` is designed for **service data & deployments**.
- Each app gets its **own folder** inside `/srv`.
- Inside that folder, you keep:
    - `docker-compose.yml`
    - configs
    - persistent volumes

📂 Example layout:

```None
/srv/
├── caddy/
│    ├── docker-compose.yml
│    ├── configs/          # app configs (e.g., Caddyfile)
│    ├── site/             # static site files
│    └── data/             # persistent container state
│
├── airflow/
│    ├── docker-compose.yml
│    ├── configs/          # airflow.cfg, env vars
│    └── logs/             # persistent logs
│
├── mlflow/
│    ├── docker-compose.yml
│    ├── configs/          # mlflow.conf, creds
│    └── data/             # experiments, models
```

---

2. Use named volumes for container internals

- Inside Compose, map container paths to **named volumes** → durability.
- Use **bind mounts only for configs & site files** (things you want to edit easily).

🔹 Example (Caddy):

```
None
version: '3.9'
services:
  caddy:
    image: caddy:latest
    container_name: caddy
    restart: unless-stopped
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./configs/Caddyfile:/etc/caddy/Caddyfile   # host config
→ container config
      - ./site:/usr/share/caddy                   # host site
files → container site
      - caddy_data:/data                          # named volume
→ container state
      - caddy_config:/config                      # named volume
→ container config state

volumes:
  caddy_data:
  caddy_config:
```

---

3. Document the volume mapping clearly

When you write docs, explain each line:

```None
- ./configs/Caddyfile:/etc/caddy/Caddyfile    # Host config file
→ Container config
- ./site:/usr/share/caddy                     # Host HTML files →
Container site
- caddy_data:/data                            # Container state
(auto-created by Docker)
- caddy_config:/config                        # Auto-created
named volume for config state
```

---

5. General Rules for Docker-Compose-only Installations

✅ Put **all services under `/srv/<app>`**
✅ Keep configs under `/srv/<app>/configs`
✅ Keep site/static files under `/srv/<app>/site`
✅ Use **named volumes for container data**
✅ Always write a **README.md** with steps & mappings

🔹 `/opt` vs `/srv`

`/opt`

- Stands for **"optional software"**.
- Purpose: for **add-on software packages** that are *self-contained bundles*.
- Example: if you download and install Google Chrome, MATLAB, or a vendor-provided binary → it often goes under `/opt/<package>`.
- Think of it like: *"I'm installing an external app on this machine itself."*
- **Not intended** for per-service data, logs, configs.

👉 `/opt` is good if you're installing a *single binary/package* (like `opt/zoom`, `opt/vscode`), not managing services with configs + data.

`/srv`

- Stands for **"service data"**.
- Purpose: to hold **data and configs for network services** you run on this server.
- Example: `/srv/http` for a web server, `/srv/git` for Git repos, `/srv/mlflow` for MLflow experiments.
- Meant for **things your server *serves*** (apps, APIs, websites, pipelines).
- Keeps deployments clean: each service lives in `/srv/<app>` with its configs, site files, volumes, logs.

👉 `/srv` is the **right place** for Docker Compose setups because you are running services (Caddy, Airflow, MLflow, Kafka, etc.), not just installing software binaries.

🔷 TL;DR

- **Use `/srv` for Docker Compose deployments** (Caddy, Airflow, MLflow, Kafka, RabbitMQ, etc.).
- **Use `/opt` for vendor apps/binaries** that aren't really "services" you are hosting, but optional software you installed.