🗂️ What is a File System?

A **file system** is the way an operating system organizes and manages data on storage (disk, SSD, USB, etc.).

- It decides **how files are stored, named, accessed, and secured**.
- Without it, data would just be raw 0s and 1s with no meaning.

👉 In Linux, **everything is a file** (text, binary, device, process).

🔹 Types of File System Concepts in Linux

1. Physical File System

- Lowest level — how data is actually stored on **disks/partitions**.
- Deals with **blocks, sectors, inodes**.
- Examples:
  - ext4, XFS, Btrfs, NTFS, FAT32.
- Tools: `mkfs.ext4`, `fsck` (check & repair).

👉 Think of it as the **roads and buildings** on a city map.

👍 Here's a **super short summary** of common file systems you'll encounter in Linux/DevOps:

- **ext4** → Default Linux FS, reliable, supports large files (up to 16TB).
- **XFS** → High-performance, good for big data & servers.
- **Btrfs** → Modern Linux FS, supports snapshots, compression, self-healing.
- **NTFS** → Windows default FS, Linux can read/write with drivers.
- **FAT32** → Old, works everywhere (USB, cameras), but max 4GB file size.

👉 Trick to remember:
"Every Xtra Big Network File" → **ext4, XFS, Btrfs, NTFS, FAT32**

📂 Common File Systems & Their Full Forms / Origins

- **ext4** → **Fourth Extended File System**
  - Successor of ext3, ext2. Standard Linux FS.
- **XFS** → Originally from **X Operating System (XOS)** project at SGI (Silicon Graphics).

    - Not really a “full form,” but often just referred to as **X File System**.
- **Btrfs → B-tree File System**
    - Uses B-trees for organization, supports snapshots and advanced features.
- **NTFS → New Technology File System**
    - Microsoft Windows default FS since Windows NT.
- **FAT32 → File Allocation Table (32-bit version)**
    - Very old, used in USBs, cameras, etc. Limited features.

👉 Easy recall trick:

- ext → “extended” (Linux native)
- XFS → “X” (high-performance, enterprise)
- Btrfs → “B-tree” (modern, advanced)
- NTFS → “New Tech” (Windows)
- FAT → “File Allocation Table” (old but universal)

---

2. Logical File System

- Provides **structure and rules** for organizing files.
- Handles **metadata**: file names, permissions, directories, ownership.
- It maps user-level file operations (`open`, `read`, `write`) to the physical layer.

👉 Think of it as **city planning** — how houses (files) are arranged, who owns them, and what rules apply.

---

3. Virtual File System (VFS)

- A **kernel layer** that provides a **uniform interface** for all file systems.
- You can use the same commands (`ls`, `cp`, `rm`) whether the backend is ext4, NTFS, NFS, etc.
- Allows Linux to support **multiple file systems at once**.
- Example: You mount an **NTFS USB drive** and an **ext4 partition**, and both work seamlessly because of VFS.

👉 Think of it as a **translator** between you (the user) and different storage systems.

🔹 How They Work Together (Layered View)

```
+-----------------------+   <- User commands (ls, cp, grep)
| Applications & Shell  |
+-----------------------+   <- Virtual File System (VFS layer in kernel)
| VFS API (open, read)  |
+-----------------------+   <- Logical File System (directories, permissions)
| Metadata, Inodes,     |
| Ownership, Security   |
+-----------------------+   <- Physical File System (ext4, xfs, ntfs)
| Disk blocks, sectors  |
+-----------------------+   <- Storage hardware (SSD, HDD, USB)
```

🔹 Example Walkthrough

When you type:

```
cat /home/nyapu/notes.txt
```

User command (`cat`) asks kernel to open the file.

1. **VFS** receives request → decides which FS driver (ext4, NTFS, etc.) to use.
2. **Logical FS** → checks metadata (path `/home/nyapu/`, permissions, inode).
3. **Physical FS** → fetches data blocks from disk sectors.
4. Data is returned and shown on your screen.

🔹 Special File Systems in Linux

Not all filesystems are on disk! Some are **virtual/pseudo filesystems**:

- `/proc` → process & kernel info (not real files, but generated by kernel).
- `/sys` → system and hardware info.
- `tmpfs` → in-memory temporary filesystem (used for `/tmp`, `/run`).

👉 These act like files, but live in **RAM or kernel memory**, not on disk.

🎯 DevOps Takeaway

- **Physical FS** → How data lives on storage.
- **Logical FS** → How Linux organizes files (permissions, dirs, inodes).

- **Virtual FS (VFS)** → Provides a single interface for different FS types.

This layered approach makes Linux extremely flexible for **servers, containers, and cloud environments**.

**

🗂️ Linux File System (Deep Dive)

1. Everything is a File

- In Linux, **everything is treated as a file**:
  - Normal files →Text files, images, programs text, images, executables(e.g., `document.txt`).
  - Directories → folders (special files containing file lists)
  - **Devices:** Your hard disk, keyboard, printer, and USB drive are all represented as files (e.g., `/dev/sda1`).
  - `**Processes:** Information about running programs is stored in files (e.g., /proc/123).`
  - `**Links:** Pointers to other files (like shortcuts).`

👉 This unification makes Linux powerful — same commands can work on files, devices, or sockets.

2. The Filesystem Hierarchy Standard (FHS)

Linux systems follow a standard layout called the FHS. This is the "map" of the city. Here are the most important directories you **must know**:

Directory	Purpose & Contents	DevOps Relevance
/	**The Root.** The master directory. The starting point of the entire filesystem.	Everything is under here.
/bin	**Essential User Binaries.** Core commands needed for basic system operation (e.g., `ls`, `cp`, `bash`, `cat`).	Critical for recovery and basic scripts.
/etc	**Configuration Files.** All system-wide configuration files for your OS and applications.	**EXTREMELY IMPORTANT.** You will live here. (e.g., `/etc/nginx/`, `/etc/docker/`).
/home	**User Home Directories**. Each user gets a personal folder here (e.g., `/home/nyapu`). This	Where you develop, test, and store your scripts. ~ is a shortcut for your

Directory	Purpose & Contents	DevOps Relevance
	is where your personal files and projects live.	home directory.
`/opt`	**Optional/Add-on Software**. Manually installed third-party applications often go here (e.g., `/opt/google/chrome`).	Sometimes used for custom-installed tools.
`/tmp`	**Temporary Files.** Volatile storage that is usually cleared on reboot.	Used for temporary download/cache in scripts.
`/usr`	**User Programs & Read-Only Data**. Contains the majority of user utilities and applications (e.g., `/usr/bin/`, `/usr/lib/`, `/usr/local/`).	Where most software is installed.
`/var`	**Variable Data.** Files that are expected to **grow** (vary) like logs, databases, caches, and email queues.	**EXTREMELY IMPORTANT.** Logs are here (e.g., `/var/log/nginx/`).
`/boot`	**Boot Loader Files.** Files needed to start the boot process (like the Linux kernel).	Don't touch this unless you know what you're doing!
`/dev`	**Device Files.** Where hardware components are represented as files (e.g., `/dev/sda` is your first hard disk).	Used for advanced storage management (LVM).
`/proc`	**Process Information.** A virtual filesystem that provides a window into the running **proc**esses and kernel parameters.	For monitoring and debugging running software.
`/lib`	**Essential Shared Libraries.** Library files needed by the binaries in `/bin/` and `/sbin/`.	Critical for programs to run.

Memory Technique: Remember the phrase: **"Better Eat Everything Home-cooked, Or Use Various Vegetables, But Don't Procrastinate Learning."** (B-in, E-tc, H-ome, O-pt, U-sr, V-ar, B-oot, D-ev, P-roc, L-ib).

3. File Types

Check with: `ls -l` (first character shows type)

- `-` → regular file

- `d` → directory
- `l` → symbolic link (shortcut)
- `c` → character device (`/dev/tty`)
- `b` → block device (`/dev/sda`)
- `p` → pipe
- `s` → socket

4. Inodes

- Every file has an **inode** (index node).
- Inode stores **metadata**:
  - file size, owner, permissions, timestamps, data block locations.
- View inode:

```
ls -i file.txt
stat file.txt
```

👉 Actual file contents are stored in blocks, inode is just the "address book".

7. Special Files

- `/dev/null` → "black hole", discards data.
- `/dev/zero` → infinite stream of zeros.
- `/dev/random` → random data generator.

---

8. Important Commands

- `pwd` → where am I?
- `ls -l` → list with details.
- `tree` (if installed) → show directory tree.
- `du -sh /var/log/*` → size of each log file.
- `df -h` → disk space.
- `find /etc -name "*.conf"` → find config files.

Like me, you might also wonder: if commands like `ls`, `cp`, etc. are only located inside `/bin`, then how do they work everywhere in the system?”

Why does `/bin/ls` work everywhere?

Remember the `$PATH` environment variable? It's like your brain's internal directory of where to find tools.

When you type `ls`, your system doesn't search the whole building. It quickly checks a predefined list of locations it knows contains programs. `/bin` and `/usr/bin` are at the top of that list.

You can see this list by typing:

bash

```
echo $PATH
# Output will look like:
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

So, when you type `ls` in your cubicle (`/home/nyapu`), your system says "Ah, `ls`! I know I have a tool by that name in the `/bin` warehouse." It goes and gets it from there and runs it.

The command itself lives in `/bin`, but it can be **executed** from anywhere because the system knows where to find it.

This tour is your map. With this mental model, you will never be lost. You'll always know what directory you're in and what kind of files you should expect to find there.

🛠️ Mini Project1: Personal Notes Manager

👉 A simple project where you:

- Create a notes directory,
- Add/view notes,
- Backup your notes,
- Check system/storage info,
- Automate it with a bash script.

🔹 Step 1: Setup Workspace

```
mkdir ~/notes_project
cd ~/notes_project
pwd
```

" ~ " what it means?

👉 Commands: `mkdir`, `cd`, `pwd`, `ls`

🔹 Step 2: Create Notes

```
touch note1.txt  note2.txt
ls -l
```

👉 Commands: `touch`, `ls`

🔹 **Step 3: Write & View Notes**

```
echo "My first note on Linux" > note1.txt
cat note1.txt
less note1.txt
```

👉 Commands: `echo`, `cat`, `less`

🔹 **Step 4: Organize Notes**

```
mkdir old_notes
cp note1.txt old_notes/
mv note2.txt old_notes/
ls old_notes/
```

👉 Commands: `cp`, `mv`, `ls`

🔹 **Step 5: Permissions Practice**

```
chmod 600 note1.txt   # only you can read/write
ls -l
```

👉 Command: `chmod`

🔹 **Step 6: Storage & System Info**

```
df -h         # check disk space
du -sh *      # check size of each file/folder
date          # current date/time
whoami        # logged-in user
```

👉 Commands: `df, du, date, whoami`

🔹 **Step 7: Archiving Notes**

```
tar -czf notes_backup.tar.gz old_notes/
ls -lh notes_backup.tar.gz
```

👉 Command: `tar`

🔹 Step 8: Basic Bash Script (backup_notes.sh)

Create a script to automate backup:

```
#!/bin/bash
# Backup script for notes

echo "Backup started at: $(date)"

# Create backup
tar -czf notes_backup_$(date +%F).tar.gz ~/notes_project

# Check disk usage
df -h > disk_report.txt
du -sh ~/notes_project >> disk_report.txt
```

echo "Backup completed. Disk report saved."

save and exit script

Make it executable:

chmod +x backup_notes.sh
./backup_notes.sh
👉 Concepts used: **chmod, variables, command substitution, tar, df, du, date**

🎯 What You Practiced

- **Navigation** → `cd`, `pwd`, `ls`
- **File/Folder Ops** → `mkdir`, `touch`, `cp`, `mv`
- **Viewing** → `cat`, `less`
- **Storage** → `df`, `du`
- **System Info** → `date`, `whoami`
- **Permissions** → `chmod`
- **Archiving** → `tar`
- **Bash scripting** → variables, command substitution, executable scripts

🔥 **By completing this Personal Notes Manager, you've tied together nearly all fundamental Linux skills into a single practical workflow.**

Project2: The DevOps System Checkup & Log Archive Script

Project Goal: You are a DevOps engineer. Your task is to create a script that gathers crucial system information, archives it, and stores it in a well-organized manner. This is a real-world task for monitoring system health.

What you will practice:

- **Navigation** (`cd`, `pwd`, `ls`)
- **File/Folder Operations** (`mkdir`, `touch`, `cp`, `mv`)
- **File Viewing** (`cat`, `less`)
- **Storage** (`df`, `du`)
- **System Info** (`date`, `whoami`)
- **Permissions** (`chmod`)
- **Archiving** (`tar`)
- **Basic Bash Scripting**

Project Steps

NOTE: this is you task like homework do it yourself

Step 1: Setup and Navigation

1. **Navigate** to your home directory and create a new project folder.

Step 2: Gathering System Information (Manual Checks)

Step 3: Working with Logs (A Common DevOps Task)

Step 4: Archiving and Permissions

Step 5: The Grand Finale - Create the Script

- **Save** the file (`Ctrl+O`, `Enter` in nano) and exit (`Ctrl+X`).
- **Make the script executable** (Change its mode!).
- **Run your script!**

How to Use This as a Refresher:

Once a week, or whenever you feel rusty:

1.

To practice the commands manually, and start the entire project from **Step 1** again. This is your personal practice lab!