Linux File System & Permissions Deep Dive

Understanding Storage Architecture, Ownership Models, and Advanced Access Control

Preface

Every Linux system, from the smallest embedded device to the most powerful cloud server, rests on a foundation that many administrators take for granted: the filesystem. It is the silent architecture beneath every command you type, every service you deploy, and every security boundary you enforce. Yet for something so fundamental to Linux, it is remarkably misunderstood—even by experienced practitioners.

This book exists to change that.

Linux File System & Permissions Deep Dive is a focused, thorough exploration of how Linux organizes, stores, protects, and controls access to data. Whether you are a junior system administrator trying to understand why a process can't write to a directory, or a senior engineer designing secure multi-tenant file structures on Linux, this book was written with you in mind.

Why This Book?

In my years of working with Linux systems, I've observed a recurring pattern: professionals learn just enough about filesystems and permissions to get things running, then move on. The result is a pervasive surface-level understanding that leads to misconfigured permissions, security vulnerabilities, and hours of frustrating troubleshooting. The Linux filesystem deserves deeper study—not because it is unnecessarily complex, but because mastering it unlocks a level of control and confidence that transforms how you work.

What You'll Find Inside

This book is organized into a deliberate progression that mirrors how Linux itself layers its storage and security abstractions.

Chapters 1–4 build your foundation in Linux filesystem architecture. You'll explore the Filesystem Hierarchy Standard, understand how inodes and metadata work beneath the surface, compare ext4 with modern Linux filesystems like XFS and Btrfs, and master mounting and persistent configuration through `/etc/fstab`.

Chapters 5–8 take you deep into the Linux ownership and permission model. Far beyond a surface review of `rwx`, these chapters dissect how users, groups, and ownership interact, how octal and symbolic modes truly work, and how special permission bits like SUID, SGID, and the sticky bit operate in practice.

Chapters 9–10 address Access Control Lists (ACLs)—a powerful but often misused feature of Linux. You'll learn both their limitations and their legitimate production use cases.

Chapters 11–15 pivot to security and resilience. Here we examine secure permission strategies, SELinux's interaction with Linux filesystems, systematic approaches to diagnosing permission errors, filesystem corruption recovery, and the principles behind designing secure file structures.

Chapter 16 ties everything together, challenging you to think not just as a sysadmin who *uses* Linux filesystems, but as a **filesystem architect** who *designs* them with intention.

Finally, **Appendices A–E** provide quick-reference materials—cheat sheets, reference tables, troubleshooting checklists, and a curated Linux security learning path—so this book remains useful long after your first reading.

How to Use This Book

If you're newer to Linux administration, I recommend reading sequentially. Each chapter builds on the last. If you're more experienced, feel free to jump directly to the topics that challenge you most—each chapter is designed to stand on its own when needed.

Throughout the text, you'll encounter *real commands, real scenarios, and real mistakes*. This is not an abstract treatment of theory. It is a practical guide grounded in the realities of managing Linux systems in production.

Acknowledgments

This book would not exist without the extraordinary open-source community that has built and documented Linux over decades. I am indebted to the kernel developers, distribution maintainers, and countless contributors who have made Linux the remarkable platform it is today. I also owe deep gratitude to the technical reviewers whose sharp eyes and honest feedback strengthened every chapter, and to the readers of early drafts whose questions revealed what needed to be explained more clearly.

The Linux filesystem is not just where your data lives. It is where your security begins, your architecture takes shape, and your expertise is tested. Let's dive deep.

Miles Everhart

Table of Contents

Chapter 1: Linux Filesystem Hierarchy Explained

The moment you open a terminal on a Linux system and type `ls /`, you are peering into the very skeleton of the operating system. Unlike other operating systems that scatter files across lettered drives and loosely defined folders, Linux organizes every single file, directory, device, and process reference into a single, unified tree structure. This tree begins at the root, represented by a single forward slash `/`, and branches outward in a carefully designed hierarchy that has been refined over decades of Unix and Linux development. Understanding this hierarchy is not optional knowledge for anyone who aspires to work with Linux professionally. It is foundational. Every configuration change, every software installation, every troubleshooting session, and every security audit depends on your ability to navigate and comprehend where files live and why they live there.

This chapter will walk you through the entire Linux Filesystem Hierarchy Standard, commonly abbreviated as FHS, explaining every major directory, its purpose, and the kinds of files you will encounter inside it. Along the way, you will run commands, examine real output, and build the kind of intuitive understanding that separates a casual user from a confident administrator.

The Root of Everything

In Linux, there is no concept of drive letters like C: or D: that you might encounter in Windows. Instead, every storage device, every partition, every network share,

and every virtual filesystem is mounted somewhere within a single directory tree. The very top of that tree is called the root directory, and it is written simply as `/`. Every path on a Linux system begins here. When you write `/home/alex/documents/report.txt`, you are describing a path that starts at the root, enters the `home` directory, then the `alex` directory, then `documents`, and finally arrives at the file `report.txt`.

To see the top-level contents of this tree on your own system, open a terminal and run:

```
ls /
```

On a typical Linux distribution, you will see output similar to this:

```
bin   dev  home  lib64  mnt  proc  run   srv  tmp  var
boot  etc  lib   media  opt  root  sbin  sys  usr
```

Each of these directories serves a specific, well-defined purpose. The Linux community, through the Filesystem Hierarchy Standard maintained by the Linux Foundation, has established conventions that virtually all major distributions follow. Let us now examine each of these directories in detail.

A Complete Map of the Filesystem

The following table provides a comprehensive reference for every major directory found at the root level of a Linux system. Study this table carefully, as it will serve as your reference throughout the rest of this book.

Directory	Full Name or Meaning	Purpose and Contents
`/`	Root	The top-level directory of the entire filesystem hierarchy. Everything exists beneath this single point. It is analogous to the trunk of a tree from which all branches grow.
`/bin`	Essential User Binaries	Contains fundamental command-line utilities needed for single-user mode and basic system operation. Commands like `ls`, `cp`, `mv`, `cat`, `echo`, and `bash` reside here. On modern systems using systemd, this is often a symbolic link to `/usr/bin`.
`/boot`	Boot Loader Files	Holds everything required to start the boot process, including the Linux kernel image (typically named `vmlinuz`), the initial RAM disk (`initrd` or `initramfs`), and boot loader configuration files for GRUB or other boot managers.
`/dev`	Device Files	Contains special files that represent hardware devices and virtual devices. Every hard drive, partition, terminal, USB device, and even random number generators appear here as files. For example, `/dev/sda` represents the first SCSI or SATA disk.
`/etc`	System Configuration	Stores system-wide configuration files in plain text. This is where you find network settings, user account information, service configurations, and startup scripts. Files like `/etc/fstab`, `/etc/passwd`, and `/etc/hostname` live here.

`/home`	User Home Directories	Each regular user on the system receives a personal directory under `/home`. For a user named `alex`, their home directory would be `/home/alex`. Personal files, desktop settings, shell configurations, and documents are stored here.
`/lib`	Essential Shared Libraries	Contains shared library files (similar in concept to DLL files on Windows) required by the binaries in `/bin` and `/sbin`. Kernel modules are also stored in a subdirectory here. On modern systems, this often links to `/usr/lib`.
`/lib64`	64-bit Libraries	On 64-bit systems, this directory holds 64-bit versions of essential shared libraries. It exists separately to support systems that may need both 32-bit and 64-bit libraries simultaneously.
`/media`	Removable Media Mount Points	When you insert a USB flash drive, CD-ROM, or external hard drive, the system typically mounts it automatically under `/media/username/device_label`. This is the standard location for automatically mounted removable media.
`/mnt`	Temporary Mount Points	Traditionally used by system administrators for manually and temporarily mounting filesystems. If you need to mount a network share or an additional partition for a quick task, `/mnt` is the conventional location.
`/opt`	Optional Software Packages	Designed for third-party software that does not follow the standard Linux packaging conventions. Commercial applications, proprietary tools, and self-contained software bundles are often installed here.

`/proc`	Process and Kernel Information	A virtual filesystem that does not exist on disk. It is generated dynamically by the kernel and provides a window into running processes, system hardware information, and kernel parameters. Each running process has a numbered directory here.
`/root`	Root User Home Directory	The home directory for the root (superuser) account. Unlike regular users whose homes are under `/home`, the root user's home is placed at `/root` to ensure it is available even if the `/home` partition fails to mount.
`/run`	Runtime Variable Data	A tmpfs filesystem that stores volatile runtime data since the last boot. Process IDs, socket files, and lock files that should not persist across reboots are placed here. This directory is cleared and recreated at every boot.
`/sbin`	System Binaries	Contains essential system administration binaries that typically require root privileges to run. Commands like `fdisk`, `mkfs`, `iptables`, `reboot`, and `shutdown` are found here. On modern systems, this often links to `/usr/sbin`.
`/srv`	Service Data	Intended to hold data served by the system. For example, if the machine runs a web server, website files might be placed under `/srv/www`. If it runs an FTP server, FTP files might go under `/srv/ftp`.
`/sys`	Kernel and Device Information	Another virtual filesystem, similar to `/proc`, but specifically focused on device and driver information. It provides a structured view of the kernel's device model and allows certain kernel parameters to be modified at runtime.

`/tmp`	Temporary Files	A world-writable directory where any user or application can store temporary files. Files here are typically deleted on reboot or after a set period. The sticky bit is set on this directory to prevent users from deleting each other's files.
`/usr`	User System Resources	A major secondary hierarchy containing the bulk of user-space programs, libraries, documentation, and shared data. Subdirectories include `/usr/bin`, `/usr/lib`, `/usr/share`, and `/usr/local`. This is often the largest directory on the system.
`/var`	Variable Data	Stores files that are expected to grow and change in size over time. Log files (`/var/log`), mail spools (`/var/mail`), print queues (`/var/spool`), and cached package data (`/var/cache`) all reside here.

Exploring the Hierarchy in Practice

Reading about directories is one thing. Exploring them with your own hands is another. Let us walk through several practical exercises that will deepen your understanding.

Examining the boot directory:

The `/boot` directory is critical because it contains the files your system needs before the root filesystem is even fully available. Run the following command to see its contents:

```
ls -lh /boot
```

You will likely see files like these:

```
-rw-r--r-- 1 root root  89M  Mar 15 10:22 initramfs-5.15.0-91-
generic.img
-rw-r--r-- 1 root root 256K  Mar 15 10:22 config-5.15.0-91-
generic
-rw-r--r-- 1 root root  12M  Mar 15 10:22 vmlinuz-5.15.0-91-
generic
-rw-r--r-- 1 root root 5.8M  Mar 15 10:22 System.map-5.15.0-91-
generic
```

The `vmlinuz` file is the compressed Linux kernel itself. The `initramfs` (or `initrd`) file is a temporary root filesystem loaded into memory during boot to help the kernel find and mount the real root filesystem. The `config` file records the options used when the kernel was compiled. The `System.map` file maps kernel symbol names to memory addresses, which is useful for debugging.

Exploring the virtual proc filesystem:

The `/proc` directory is fascinating because none of its files exist on any physical disk. The kernel generates them on the fly. To see information about your CPU, run:

```
cat /proc/cpuinfo
```

To see how long the system has been running:

```
cat /proc/uptime
```

To see current memory usage in a human-readable format:

```
cat /proc/meminfo
```

Every running process on your system has a directory inside `/proc` named after its process ID. For example, if a process has PID 1 (which is always the init system, typically systemd on modern distributions), you can examine it:

```
ls /proc/1
```

You will see files like `cmdline`, `status`, `environ`, `fd`, and many others, each revealing different aspects of that process.

Investigating the etc directory:

The `/etc` directory is where system administrators spend a great deal of their time. Let us look at a few important files:

```
cat /etc/hostname
```

This displays the system's hostname. Now examine the filesystem table:

```
cat /etc/fstab
```

This file tells the system which filesystems to mount at boot, where to mount them, and with what options. A typical entry looks like this:

```
UUID=a1b2c3d4-e5f6-7890-abcd-ef1234567890  /  ext4  defaults  0
1
```

This line instructs the system to mount the partition identified by the given UUID at the root `/` mountpoint, using the ext4 filesystem type, with default mount options.

To see a list of all user accounts on the system:

```
cat /etc/passwd
```

Each line represents one user account, with fields separated by colons. The format is:

```
username:password_placeholder:UID:GID:comment:home_directory:shel
l
```

For example:

```
alex:x:1000:1000:Alex Johnson:/home/alex:/bin/bash
```

This tells us that the user `alex` has user ID 1000, group ID 1000, a home directory at `/home/alex`, and uses the Bash shell.

Understanding the var directory:

The `/var` directory is where your system stores data that changes frequently. The most commonly accessed subdirectory is `/var/log`, which contains system and application log files:

```
ls /var/log
```

You will see files like `syslog`, `auth.log`, `kern.log`, `dpkg.log`, and directories for specific services. To view the most recent system log entries:

```
sudo tail -20 /var/log/syslog
```

On systems using systemd's journal, you can also use:

```
journalctl -n 20
```

Note: The `/var/log` directory is one of the first places to look when troubleshooting any issue on a Linux system. Disk space problems are also frequently caused by log files growing unchecked in this directory.

The Relationship Between /usr and the Root Directories

One aspect of the Linux filesystem that often confuses newcomers is the apparent duplication between directories at the root level and their counterparts inside `/usr`. You will notice that both `/bin` and `/usr/bin` exist, both `/lib` and `/usr/lib` exist, and both `/sbin` and `/usr/sbin` exist.

Historically, this separation existed for a practical reason. In the early days of Unix, disk space was extremely limited. The root filesystem was kept on a small, fast disk containing only the bare essentials needed to boot and perform emergency maintenance. Everything else, including the bulk of user programs and libraries,

was placed on a separate, larger disk mounted at `/usr`. If the `/usr` partition failed to mount, the system could still boot into a minimal state using only the binaries in `/bin` and `/sbin`.

On modern Linux distributions, this separation is largely unnecessary because disk space is abundant and the boot process is more sophisticated. As a result, many distributions have adopted a "merged /usr" layout where `/bin` is simply a symbolic link to `/usr/bin`, `/sbin` links to `/usr/sbin`, and `/lib` links to `/usr/lib`. You can verify this on your system:

```
ls -la /bin
```

If you see output like this, your system uses the merged layout:

```
lrwxrwxrwx 1 root root 7 Mar 10 08:00 /bin -> usr/bin
```

The `/usr` directory itself contains several important subdirectories:

Subdirectory	Purpose
`/usr/bin`	The primary location for user command binaries
`/usr/sbin`	System administration binaries not needed for boot
`/usr/lib`	Libraries for programs in `/usr/bin` and `/usr/sbin`
`/usr/local`	A tertiary hierarchy for locally compiled software, keeping it separate from distribution-managed packages
`/usr/share`	Architecture-independent shared data such as documentation, icons, fonts, and man pages
`/usr/include`	Header files for C and C++ programming, used when compiling software from source
`/usr/src`	Source code, including kernel source if installed

The `/usr/local` directory deserves special attention. When you compile software from source code and install it with `make install`, the files typically end up in `/usr/local/bin`, `/usr/local/lib`, and `/usr/local/share`. This convention

keeps manually installed software cleanly separated from software managed by your distribution's package manager, preventing conflicts during system updates.

Virtual Filesystems and Why They Matter

Two directories in the hierarchy stand out because they do not contain traditional files stored on disk. The `/proc` and `/sys` directories are virtual filesystems, sometimes called pseudo-filesystems, that serve as interfaces to the kernel.

The `/proc` filesystem, mounted as type `proc`, has been part of Unix-like systems for decades. It exposes process information and kernel state as a collection of readable (and sometimes writable) files. System monitoring tools like `top`, `htop`, `ps`, and `free` all read their data from `/proc`. When you run `free -h` to check memory usage, the command is actually parsing `/proc/meminfo` behind the scenes.

The `/sys` filesystem, mounted as type `sysfs`, is newer and provides a more structured view of the kernel's device model. It organizes information about devices, drivers, buses, and kernel subsystems into a clean directory hierarchy. System administrators and scripts can read from and write to files in `/sys` to query hardware information or change kernel behavior at runtime. For example, to check the brightness level of a laptop screen or to enable or disable a network interface, you might interact with files in `/sys`.

You can verify these virtual filesystems are mounted by running:

```
mount | grep -E "proc|sysfs"
```

You will see entries confirming that `proc` is mounted on `/proc` and `sysfs` is mounted on `/sys`.

Important note: Because `/proc` and `/sys` are virtual, they consume no actual disk space. Running `du` on them will show zero or minimal sizes, and the "files" within them are generated on demand by the kernel each time they are read.

Practical Exercises

Exercise 1: Map the Filesystem

Run the following command to display the top two levels of the entire filesystem tree:

```
tree -L 2 /
```

If the `tree` command is not installed, install it first:

```
sudo apt install tree          # Debian/Ubuntu
sudo dnf install tree          # Fedora/RHEL
sudo pacman -S tree            # Arch Linux
```

Study the output and identify each top-level directory from the table presented earlier in this chapter. Note which directories contain the most subdirectories.

Exercise 2: Identify Symbolic Links

Run this command to check whether your system uses a merged `/usr` layout:

```
file /bin /sbin /lib
```

If these are symbolic links, the output will indicate the target. Record your findings and explain what this means for where binaries are actually stored on your system.

Exercise 3: Explore a Running Process

Find the PID of your current shell session:

```
echo $$
```

Then explore the corresponding directory in `/proc`:

```
ls /proc/$$/
cat /proc/$$/cmdline
cat /proc/$$/status
```

Examine the `status` file and identify the process name, state, parent PID, and memory usage.

Exercise 4: Analyze Disk Usage by Directory

Run the following command to see how much disk space each top-level directory consumes:

```
sudo du -sh /* 2>/dev/null | sort -rh | head -20
```

This command calculates the size of each directory under `/`, sorts the results from largest to smallest, and displays the top 20. Note which directories consume the most space and consider why. The `2>/dev/null` portion suppresses error messages from virtual filesystems that cannot be measured.

Exercise 5: Read the Filesystem Table

Open the filesystem table and study each entry:

```
cat /etc/fstab
```

For each line that is not a comment, identify the device or UUID, the mount point, the filesystem type, and the mount options. Then compare this with the currently mounted filesystems:

```
df -hT
```

The `df` command shows disk usage along with filesystem types, letting you verify that the entries in `/etc/fstab` match what is actually mounted.

Bringing It All Together

The Linux filesystem hierarchy is not a random collection of folders. It is a carefully designed architecture where every directory has a defined role, and every file has a logical home. The `/bin` and `/sbin` directories hold essential commands. The `/etc` directory holds configuration. The `/home` directory holds user data. The `/var` directory holds data that changes over time. The `/proc` and `/sys` directories provide windows into the kernel. The `/tmp` directory provides scratch space. The `/usr` directory holds the bulk of installed software.

When you understand this hierarchy, you stop guessing where files might be and start knowing where they must be. When a service fails, you know to check its configuration in `/etc` and its logs in `/var/log`. When you need to find a binary, you know to look in `/usr/bin` or `/usr/sbin`. When you need to understand what hardware the kernel sees, you know to explore `/sys` and `/proc`.

This knowledge forms the foundation upon which everything else in this book is built. In the chapters that follow, we will explore how Linux controls who can access these files and directories through its ownership and permission models. But first, you must know where the files are. Now you do.