# Course 1  Linux Foundations for DevOps

**Filesystem  identities and permissions  processes  pipelines  package management**

This course establishes the operating-system substrate on which the rest of the course depends. Students investigate how Linux organizes persistent and runtime state, how the kernel evaluates identity and permissions, how shells construct process groups and pipelines, and how administrators observe and change a running system.

| Chapter convention | Meaning |
|---|---|
| Specification requirement | This defines the expected course scope. |
| Teaching example | A small, reproducible illustration introduced in this chapter; it is not an additional deliverable. |
| Optional advanced activity | Enrichment for early finishers; it does not change the mandatory scope. |

# 1 Learning Objectives

By the end of this course, students should be able to:

- analyze a Linux host using the Filesystem Hierarchy Standard and distinguish persistent configuration, variable data, runtime interfaces, and application-owned state;
- explain how real and effective user and group identities participate in access decisions, and design a collaborative directory using ownership, group permissions, umask, and the set-group-ID directory bit;
- interpret symbolic and octal mode representations and diagnose a failed pathname access by checking every directory component;
- construct pipelines and redirections that transform log records while reasoning about file descriptors, exit status, quoting, locale, and error streams;
- observe processes through ps, top, and procfs; relate PIDs, parent processes, process groups, jobs, and controlling terminals;
- compare graceful termination with forced termination and justify why SIGTERM should normally precede SIGKILL;
- perform a dry-run-first file cleanup and evaluate race, quoting, traversal, and least-privilege risks;
- explain the roles of repositories, package metadata, dependency resolution, installation databases, and transactional limitations in Linux package management.

# 2 Why This Matters in DevOps

DevOps automation ultimately acts on operating-system resources: files, users, processes, sockets, packages, logs, and permissions. A deployment can fail even when application code is correct because a service account cannot traverse a parent directory, a stale PID owns a port, a pipeline silently discards an error, a cleanup rule deletes live state, or two hosts resolve different package sets. Linux literacy therefore supports reliability, reproducibility, security, and incident response across build agents, virtual machines, containers, Kubernetes nodes, and cloud instances.

In production, these concepts appear in configuration management that enforces file ownership, CI runners that interpret exit status, supervisors that send termination signals during rollout, log pipelines that aggregate evidence, and immutable image builds that install pinned packages. The command line is not the objective; the objective is a correct mental model that remains valid when the interface changes.

# 3 Connection to Previous and Future Courses

This is the first course, so it has no prerequisite artifact from the previous courses. It creates the operational vocabulary used immediately in Course 2 for network diagnosis and SSH, in Course 3 for robust Bash automation and cron, and in Course 4 for repository-based collaboration. The permissions and process models return in Docker (Courses 5 and 6), CI/CD runners (Courses 7 and 8), Ansible idempotency (Course 9), observability (Course 11), load diagnosis (Course 12), Kubernetes security contexts and process lifecycle (Course 13), and the integrated project (Course 14).

# 4 Core Concepts

## 4.1 Linux Filesystems and the Filesystem Hierarchy Standard

### Definition and intuition

A Linux filesystem presents objects through one rooted namespace beginning at /. The Filesystem Hierarchy Standard, or FHS, defines conventional purposes for major directories so software and administrators can make portable assumptions. The FHS is a contract about placement, not a claim that every distribution or container image contains every directory.

### How it works and internals

A pathname is resolved component by component. Each directory maps names to inode-like filesystem objects; mount points splice additional filesystems into the namespace. A process has a current working directory and a root directory, while mount namespaces can give different processes different views. This is why the same absolute path may refer to different backing storage inside a container.

| Path | Engineering role | Typical examples and caveats |
|---|---|---|
| /etc | Host-specific system configuration | passwd, group, hosts, ssh; generally text, but distribution details vary. |
| /var | Variable data that changes during operation | logs, caches, queues, databases; capacity and retention require management. |
| /run | Volatile runtime state since boot | PID files, sockets, service-manager state; normally tmpfs and not persistent. |
| /proc | Kernel-backed process and system interface | Numeric PID directories, mounts, memory and kernel data; pseudo-files, not ordinary disk files. |
| /srv | Data served by this system | A reasonable location for site-specific service data such as /srv/app. |
| /usr | Shareable, mostly read-only user-space programs and data | Binaries, libraries, documentation; modern systems may merge /bin into /usr/bin. |
| /tmp | Temporary files | Usually broadly writable; retention and cleanup policy are system-specific. |
| /home | User home directories | Interactive-user state; not a suitable implicit dependency for system services. |

### Why it matters  use cases  trade-offs

Predictable placement supports automation and incident response: configuration under /etc can be reviewed separately from variable data under /var, and a full /var filesystem can be diagnosed without confusing it with application binaries. Containers and minimal distributions relax some expectations; code should discover capabilities rather than assume a full traditional host. For applications, /opt, /srv, /var/lib, and distribution-native package paths are alternatives whose suitability depends on ownership, mutability, and lifecycle.

### Commands and misconceptions

```
pwd
ls -ld /etc /var/log /proc /run /srv
findmnt -T /proc
stat /etc/passwd
file /proc/uptime
```

A common misconception is that everything visible as a file is stored on disk. procfs exposes kernel data through file-like operations. Another is that FHS compliance determines security; placement improves

predictability, but permissions, mount options, MAC policy, namespaces, and service configuration enforce access.

## 4.2 Users Groups Credentials and Permission Checks

### Definition and intuition

Linux authorization starts with numeric identities. User names and group names are human-readable mappings, while processes carry real, effective, saved, filesystem, and supplementary identifiers. For ordinary file access, think of the process as the subject, the inode and path as the object, and the kernel as the policy decision point.

### How access is evaluated

For traditional discretionary access control, the kernel selects exactly one class: owner bits if the effective filesystem UID matches the file owner; otherwise group bits if a relevant group matches; otherwise other bits. It does not combine the most permissive bits across classes. Before the final object is reached, each directory component must grant search permission. Access control lists, capabilities, read-only mounts, SELinux or AppArmor, and filesystem-specific behavior can further constrain the result.

| Bit on regular file | Bit on directory |
|---|---|
| r reads file data | r lists entry names, subject to other permissions |
| w modifies file data | w creates, removes, or renames entries, normally requiring x as well |
| x permits execution when format and mount policy allow | x searches or traverses the directory |

### Special bits and collaborative directories

The set-group-ID bit on a directory causes new entries to inherit the directory group on Linux, supporting shared project trees. It does not automatically grant group write permission: the creating process umask still influences new modes, and existing content may need correction. The sticky bit limits deletion in shared writable directories such as /tmp. Set-user-ID and set-group-ID on executables alter effective credentials and should be treated as security-sensitive mechanisms, not convenient permission shortcuts.

```
id
namei -l /srv/app/example.txt
stat -c '%A %a %U %G %n' /srv/app
getfacl -p /srv/app  # if the acl package is installed
umask
```

### Trade-offs and misconceptions

Unix groups are simple and widely supported, but changes to supplementary groups normally require a new login session. POSIX ACLs express exceptions more precisely, at the cost of less obvious state. Broad mode 0777 is not a collaboration design: it removes meaningful isolation and can enable deletion or replacement attacks. chmod changes mode bits; chown changes ownership. Neither proves that an application can access a path end to end.

## 4.3 Processes Procfs Jobs and Signals

### Definition and mental model

A process is a running program instance with an address space, open file descriptors, credentials, signal dispositions, resource limits, and scheduling state. A PID identifies it within a PID namespace. Processes

form parent-child relationships, while process groups and sessions support shell job control. A shell job may contain several processes when it is a pipeline.

### Creation and observation

Conceptually, a shell creates a child process, arranges descriptors and credentials, and executes a program. Linux exposes process metadata in /proc/<PID>. ps produces a snapshot; top repeatedly samples. The command name alone is weak evidence because it can be ambiguous. Good diagnosis correlates PID, PPID, user, state, elapsed time, arguments, open files, and logs.

```
ps -eo pid,ppid,pgid,sid,user,stat,etime,cmd --forest
cat /proc/$$/status | sed -n '1,12p'
readlink /proc/$$/exe
ls -l /proc/$$/fd
```

### Job control

An interactive shell assigns a process group to each job and gives one job control of the terminal. Appending & starts a background job. Ctrl-Z normally sends SIGTSTP to the foreground process group; bg resumes it in the background and fg returns it to the foreground. jobs is a shell builtin and therefore reports jobs known to that shell, not every process on the host.

```
sleep 300 &
jobs -l
fg %1
# Press Ctrl-Z, then:
bg %1
kill %1
```

### Signals and termination semantics

A signal is an asynchronous notification. SIGTERM requests termination and can be caught or handled, allowing cleanup. SIGKILL cannot be caught, blocked, or ignored; the kernel terminates the target without application-level cleanup. SIGSTOP is similarly uncatchable and suspends execution. A successful kill command means the signal was accepted for delivery, not necessarily that the process has completed shutdown.

| Action | Meaning | Engineering implication |
|---|---|---|
| kill -TERM <PID> | Request graceful termination | Preferred first step; observe shutdown and bounded timeout. |
| kill -KILL <PID> | Force kernel termination | Last resort; cleanup handlers do not run and partial state may remain. |
| kill -0 <PID> | Send no signal; test existence and permission | Useful probe, but PID reuse and zombies require careful interpretation. |
| wait <PID> | Wait for a child and collect status | Prevents orchestration races in shell scripts. |

In modern production systems, a service manager or orchestrator owns lifecycle decisions. Prefer systemctl stop for a systemd service and the orchestrator API for a container workload, because direct PID signaling bypasses policy, dependencies, restart logic, and audit context.

## 4.4 File Descriptors Redirection and Pipelines

### Definition and intuition

A file descriptor is a small per-process integer that refers to an open file description or another I/O endpoint. By convention, descriptor 0 is standard input, 1 standard output, and 2 standard error. Redirection changes

where a descriptor points before a command executes. A pipeline connects one command standard output to the next command standard input through a kernel pipe.

```
command >out.log 2>error.log
command >>out.log 2>&1
producer | filter | summarizer
```

### How pipelines work

Pipeline components normally run concurrently. Pipes are bounded buffers: a writer may block if downstream consumers do not read, and readers observe end-of-file only after all write descriptors are closed. In Bash, the pipeline status is normally the status of the last command; enabling pipefail makes it reflect the rightmost failing component. This matters because a successful sort can otherwise hide a failed log reader.

```
set -o pipefail
grep 'Failed password' sample-auth.log | awk '{print $(NF-3)}' | sort | uniq -c | sort -nr | head -n 5
printf 'pipeline status=%s\n' "$?"
```

### Correctness trade-offs

Text pipelines are excellent for exploration and small automation, but they encode assumptions about record structure, locale, escaping, and newline delimiters. Parsing an IP by a fixed field works only for a known log format. Structured logs and tools such as jq reduce ambiguity. Quote variable expansions, use -- before untrusted path operands where supported, set a deterministic locale when ordering matters, and explicitly decide whether stderr belongs in the data stream.

A frequent misconception is that cat is inherently wrong in a pipeline. The engineering question is whether it clarifies the source or adds an unnecessary process. More serious errors are parsing unspecified human output, ignoring upstream failure, and treating a demonstration pipeline as a production log-processing architecture.

## 4.5 Safe Discovery and Cleanup with find

### Definition and mechanism

find traverses a directory tree, evaluates an expression for each entry, and performs selected actions. Expression order matters. Tests such as -type, -mtime, -path, and -prune control matching and traversal. Destructive actions should follow a dry run that prints the exact candidate set.

```
# Dry run in a test tree: regular files whose data was last modified more than 7*24 hours ago
find ./tmp-lab -type f -mtime +7 -print

# Only after reviewing the output; this deletes the selected files
find ./tmp-lab -type f -mtime +7 -delete
```

### Trade-offs and failure modes

-mtime +7 is based on rounded 24-hour periods and means strictly greater than seven such periods; it is not the same as older than a calendar date. A concurrent process can modify or replace files between observation and action. -delete also implies depth-first behavior in GNU find and can interact badly with -prune, so exclusions should be designed and tested deliberately. Production cleanup is often better expressed with systemd-tmpfiles, logrotate, application retention policy, or lifecycle rules because these systems make ownership and retention explicit.

Never practice destructive cleanup against /tmp, /var, or a production path. Use a lab-owned tree, print candidates, validate boundary assumptions, and only then delete. Avoid unquoted command substitution over file names; null-delimited interfaces such as -print0 and xargs -0 are safer when a separate command is required.

## 4.6 Package Management as State Reconciliation

### Definition and internals

A package manager reconciles desired software with configured repositories and a local installation database. On Debian-derived systems, APT resolves dependencies and downloads repository metadata and archives, while dpkg performs low-level package installation and records state. Other families use DNF/RPM, Zypper/RPM, pacman, or image-oriented systems. Commands are not interchangeable, but the model is similar.

```
sudo apt update
apt-cache policy curl
sudo apt install curl
dpkg-query -W -f='${Package} ${Version}\n' curl
```

apt update refreshes metadata; it does not upgrade installed packages. Reproducibility requires controlling repository identity, package versions, and time. Installing “latest” on two dates can produce different systems. For production automation, prefer distribution-supported repositories, verify repository signing, minimize packages, pin when justified, and test upgrades. Container images improve immutability but still depend on package metadata at build time.

### Trade-offs and misconceptions

Distribution packages integrate with security updates and service conventions but may lag upstream versions. Language package managers provide application libraries but should not silently replace system package policy. Curl-pipe-shell installers reduce review and provenance. A successful installation is not proof that a service is correctly configured, enabled, healthy, or securely exposed.

# 5 Architecture and Mental Models

## 5.1 Access Decision for a Path

The following model separates pathname traversal from the final object check:

```
Process credentials
    |
    v
Resolve /srv/app/config.yml component by component
    |-- search permission on /
    |-- search permission on /srv
    |-- search permission on /srv/app
    v
Select owner OR group OR other class on final inode
    |
    +--> additional ACL / capability / mount / MAC checks
    v
Allow or return an error such as EACCES
```

Step 1: the process supplies credentials and a pathname. Step 2: the kernel traverses each directory, requiring search permission. Step 3: traditional mode-bit selection uses one class, not a union. Step 4: additional mechanisms may further restrict or authorize access. Step 5: the operation is allowed or an error is returned. This model explains why a readable file may still be unreachable.

## 5.2 Shell Pipeline and Exit Status

```
log file --> grep --> awk --> sort --> uniq -c --> sort -nr --> head
        kernel pipes connect stdout to stdin
            stderr remains separate unless redirected
```

The shell creates processes and pipes, duplicates descriptors into the required positions, closes unused descriptors, and waits according to foreground/background rules. Each stage can fail independently. Without pipefail, the final head command can return zero even if an earlier stage failed. Production automation should capture and interpret failure at the correct boundary.

# 6 Practical Examples

## Example 1 Inspect persistent and runtime state

Goal. Compare an ordinary configuration file with procfs data.

```
stat /etc/passwd
stat /proc/uptime
findmnt -T /proc
head -n 3 /proc/meminfo
```

Expected result. /etc/passwd appears on a persistent filesystem, while findmnt reports procfs for /proc. Values in /proc/meminfo change with system state.

What to observe. Both are accessed with file APIs, but their lifecycle and backing mechanisms differ. Do not infer persistence from a pathname alone.

## Example 2 Diagnose directory traversal

Goal. Show why final-file mode is insufficient to explain access.

```
mkdir -p ./permission-demo/private
printf 'demo\n' > ./permission-demo/private/data.txt
chmod 700 ./permission-demo/private
chmod 644 ./permission-demo/private/data.txt
namei -l ./permission-demo/private/data.txt
```

Expected result. data.txt is world-readable, but another user cannot reach it because private lacks search permission for that user. namei -l exposes permissions on every component.

Cleanup. rm -rf ./permission-demo removes only the lab-owned demonstration tree after you verify the path.

## Example 3 Build a format-aware log summary

Goal. Practice a pipeline on controlled input rather than assuming the host has /var/log/auth.log or permission to read it.

```
cat > sample-auth.log <<'EOF'
Jan 10 host sshd[10]: Failed password for invalid user root from 192.0.2.10 port 40100 ssh2
Jan 10 host sshd[11]: Failed password for alice from 198.51.100.7 port 40101 ssh2
Jan 10 host sshd[12]: Failed password for root from 192.0.2.10 port 40102 ssh2
EOF

LC_ALL=C awk '/Failed password/ {for (i=1;i<=NF;i++) if ($i=="from") print $(i+1)}' sample-auth.log \
  | sort | uniq -c | sort -nr | head -n 5
```

Expected result. 192.0.2.10 has count 2 and 198.51.100.7 has count 1. The documentation addresses reserved example ranges, not real attackers.

What to observe. Searching for the token from is more robust for these two message variants than a fixed field number, but it is still format-dependent. A changed locale can affect ordering; LC_ALL=C makes it deterministic.

### Example 4  Observe graceful and forced termination safely

Goal. Compare signal behavior using disposable processes.

```
bash -c 'trap "echo TERM received; exit 0" TERM; while :; do sleep 1; done' &
pid=$!
ps -o pid,ppid,pgid,stat,cmd -p "$pid"
kill -TERM "$pid"
wait "$pid"; printf 'exit=%s\n' "$?"

sleep 300 & pid=$!
kill -KILL "$pid"
wait "$pid"; printf 'exit=%s\n' "$?"
```

Expected result. The first process runs its TERM trap. The second is terminated immediately and commonly yields shell status 128 plus signal number, often 137 for SIGKILL. Shell diagnostics vary.

What to observe. Always target the PID captured in the same shell and verify it before signaling. Do not copy this pattern onto an unknown service process.

### Example 5  Design a dry-run cleanup

Goal. Separate selection from deletion in a controlled tree.

```
mkdir -p ./tmp-lab/important ./tmp-lab/scratch
touch -d '10 days ago' ./tmp-lab/old.log ./tmp-lab/important/keep.log
touch ./tmp-lab/new.log

find ./tmp-lab -path './tmp-lab/important' -prune -o \
  -type f -mtime +7 -print
```

Expected result. Only ./tmp-lab/old.log is printed. The important subtree is pruned before file selection.

What to observe. Parentheses and operator precedence matter. Before adding deletion, test with unusual names and re-read the candidate list. The main assignment asks students to design and document their own exception logic.

## 7  Engineering Discussion

### Interactive knowledge versus automation

Exploratory commands are optimized for learning and diagnosis. Production automation additionally needs idempotency, predictable locale, explicit failure handling, bounded execution, auditability, and tests. A one-liner can be appropriate for a reversible investigation but inappropriate as an undocumented scheduled cleanup. As the course progresses, students will convert these mental models into scripts, configuration management, images, and orchestrator declarations.

### Small team versus enterprise

A two-person team can use a Unix group and a shared directory effectively. At enterprise scale, identity usually comes from centralized directories, access is role-based and reviewed, privileged actions are audited, and shared mutable hosts are minimized. The simple model remains useful, but governance, federation, ephemeral credentials, mandatory access control, and fleet-wide policy become essential.

### What breaks at scale

Text logs become too large and inconsistent for ad hoc pipelines; PID-based actions suffer from reuse and namespace ambiguity; manual package installation creates drift; shared write directories accumulate

unclear ownership; and unbounded cleanup competes with live workloads. The corresponding production answers include structured telemetry, service managers, immutable artifacts, declarative configuration, dedicated service identities, retention controllers, and observability around automation.

### Assumptions and technical debt

The examples assume GNU userland, Bash, a Linux kernel, and sufficient permissions in a disposable environment. WSL, containers, macOS, BusyBox, and minimal distributions differ. Encoding undocumented assumptions about command variants or log layout is technical debt. Record the environment, check tool help or man pages, and make invariants explicit.

# 8 Security Considerations

- Use least privilege. Do not conduct routine exploration as root; use sudo only for the narrow command that requires elevation.
- Treat /etc/shadow, private keys, environment variables, process command lines, and logs as potentially sensitive. Do not paste real secrets into reports or AI systems.
- Avoid world-writable application directories. For collaborative paths, prefer a dedicated group, set-group-ID inheritance, an intentional umask or default ACL, and restricted deletion policy.
- Verify targets before kill, chown, chmod, find -delete, rm, and package removal. PID reuse, symlinks, mount boundaries, and path expansion can turn a correct-looking command into damage.
- Package trust is a supply-chain concern. Prefer signed distribution repositories, understand who controls a repository, and avoid unaudited remote installer scripts.
- Logs are evidence and data. Preserve timestamps and provenance, restrict access, and avoid commands that rewrite the only copy during an incident.

# 9 Common Failure Scenarios and Troubleshooting

### Permission denied despite a readable file

Symptoms. cat reports Permission denied although ls -l shows r for the expected class.

Possible causes. A parent directory lacks x; the process has stale group membership; an ACL denies expected access; the filesystem is read-only for a write; SELinux or AppArmor intervenes.

```
id
namei -l /path/to/file
getfacl -p /path/to/file
findmnt -T /path/to/file
```

Reasoning and fix. Start with the identity of the failing process, not the interactive user you assume. Walk every path component. Then inspect ACL, mount, and security logs. Correct the narrowest failed policy; do not respond with chmod -R 777.

### New files in a shared directory have the wrong group or are not writable

Symptoms. One member creates a file that the other cannot edit, or group ownership changes unexpectedly.

Possible causes. Missing set-group-ID on the directory, restrictive umask, stale login session, or an application that explicitly sets modes.

```
id
stat -c '%A %a %U %G %n' /srv/app
umask
find /srv/app -maxdepth 1 -printf '%M %u %g %p\n'
```

Reasoning and fix. Separate group inheritance from write permission. Set-group-ID addresses the first; group-write defaults require an appropriate umask or default ACL. Re-login after group changes and test from both identities.

## Pipeline prints plausible output but hides an upstream error

Symptoms. A report is empty or partial, yet the shell status is zero.

Possible causes. The last command succeeded, stderr was overlooked, input was unreadable, or the log format changed.

```
set -o pipefail
producer | filter | summarizer
printf 'status=%s PIPESTATUS=%s\n' "$?" "${PIPESTATUS[*]}"
```

Reasoning and fix. Test input acquisition independently, inspect stderr, validate record shape, then reassemble the pipeline. In scripts, use pipefail with deliberate error handling rather than assuming it solves every case.

## SIGTERM appears not to stop a process

Symptoms. kill -TERM returns success but the PID remains.

Possible causes. Graceful shutdown takes time; the process ignores or blocks TERM; the PID is a zombie; a supervisor restarted it; you targeted a wrapper rather than its child or process group.

```
ps -o pid,ppid,pgid,stat,etime,cmd -p <PID>
cat /proc/<PID>/status
journalctl --since '-5 min'  # where systemd journal access is available
```

Reasoning and fix. Observe state and ownership, then identify the supervisor and logs. Use the service manager. Escalate to SIGKILL only after a justified timeout and with awareness of data-integrity consequences.

## Cleanup selects the wrong files

Symptoms. The dry run includes protected content or excludes expected old files.

Possible causes. Misunderstood -mtime rounding, incorrect -prune ordering, shell expansion, symlinks, or running from the wrong directory.

```
pwd -P
find ./tmp-lab -printf '%TY-%Tm-%Td %TH:%TM %y %p\n' | sort
```

Reasoning and fix. Stop before deletion. Reconfirm the root, inspect timestamps and types, reduce the expression, and add one predicate at a time. Create boundary test cases at exactly seven and eight days.

## Package installation differs across student machines

Symptoms. A package name is missing, versions differ, or installation succeeds on one distribution only.

Possible causes. Different distribution/release, stale repository metadata, third-party repositories, architecture mismatch, or unavailable network access.

```
cat /etc/os-release
uname -m
apt-cache policy <PACKAGE>  # Debian/Ubuntu family
```

Reasoning and fix. Establish platform and repository state before changing it. Use the native package manager and course-supported environment. Record relevant version evidence rather than claiming commands are universal.

# 10 Laboratory Assignment

The specification describes orientations and practice ideas, not a rigid checklist. Clarifications below preserve that distinction; no new mandatory task has been added.

## Themes

- Linux filesystem and the Filesystem Hierarchy Standard; users and permissions.
- Processes and job control.
- Pipes, I/O redirection, and package management.

## Suggested Laboratory Activities

1. Navigation and structure. On a Linux machine (VM, WSL, or container), explore /etc, /var/log, and /proc. Identify five files or directories under /etc that are frequently useful in administration and briefly explain each role.
2. Multi-user permissions. Explore a scenario with a deploy group, two member users, and a shared /srv/app directory in which both members can write while others can only read. Use chmod, chown, and chmod g+s as appropriate. Verify with ls -la and writing tests from both accounts.
3. Log pipelines. Using /var/log/auth.log or a supplied sample log, construct a grep | awk | sort | uniq -c | sort -nr pipeline that reports the five IP addresses associated with the most failed authentication attempts.
4. Process management. Start a background process, observe it with ps or top, terminate it with SIGTERM, and use a new disposable process to observe SIGKILL. Record a short explanation of the difference, for example in signals.txt.
5. File cleanup. Design a useful one-liner that deletes files older than seven days from a test tmp/ directory without affecting subdirectories marked as important. Document the exceptions and test the selection safely.

## Expected Deliverables

Keep the commands used, relevant output, and brief explanations. These artifacts should demonstrate observations and reasoning rather than raw terminal transcripts. The specification also suggests signals.txt for the signal comparison. If working in a team, roles.txt helps make contributions clear.

## Team Coordination

Work individually or in groups of two. When working as a pair, use roles.txt to state who investigated or verified each part. Both students should be able to explain the whole result; division of work does not replace shared understanding.

## AI Usage Note

If AI is used, briefly record how it was used, what was verified manually, and perceived advantages and disadvantages. The purpose is to distinguish demonstrated understanding from generated material. Do not disclose credentials, private logs, or personal data to an AI service.

# 11 References and Useful Resources

## 11.1 Official Documentation

**Filesystem Hierarchy Standard 3.0**
https://refspecs.linuxfoundation.org/FHS_3.0/fhs/index.html
Normative directory-purpose definitions used for the filesystem discussion.

**GNU Coreutils Manual**
https://www.gnu.org/software/coreutils/manual/coreutils.html
Authoritative behavior for ls, chmod, chown, sort, uniq, head, and related user-space tools.

**GNU Bash Reference Manual**
https://www.gnu.org/software/bash/manual/
Shell language, pipelines, redirections, job control, and exit-status semantics.

**Linux proc(5)**
https://man7.org/linux/man-pages/man5/proc.5.html
Kernel-backed procfs organization and process/system interfaces.

**Linux credentials(7)**
https://man7.org/linux/man-pages/man7/credentials.7.html
PIDs, process groups, sessions, real/effective identities, and supplementary groups.

**Linux inode(7)**
https://man7.org/linux/man-pages/man7/inode.7.html
File types, permission bits, and special mode-bit semantics.

**Linux signal(7)**
https://man7.org/linux/man-pages/man7/signal.7.html
Signal delivery, dispositions, masks, and standard signal behavior.

**GNU Findutils Manual**
https://www.gnu.org/software/findutils/manual/html_mono/find.html
Detailed find expressions, traversal, deletion, and safe filename handling.

**APT User Guide**
https://www.debian.org/doc/manuals/apt-guide/index.en.html
The Debian APT model, dependency handling, and the relationship with dpkg.

**systemd journalctl Manual**
https://www.freedesktop.org/software/systemd/man/latest/journalctl.html
Query syntax and behavior for systems that store logs in the systemd journal.

## 11.2 Laboratory Specification References

**Filesystem Hierarchy Standard**
https://refspecs.linuxfoundation.org/FHS_3.0/fhs/index.html
Original Lab01 reference for the purpose of Linux filesystem directories.

**GNU Coreutils**
https://www.gnu.org/software/coreutils/manual/
Original Lab01 reference for chmod, chown, ls, and other foundational commands.

**signal(7) Linux man-pages**
https://man7.org/linux/man-pages/man7/signal.7.html
Original Lab01 reference for comparing signal semantics, especially SIGTERM and SIGKILL.

**The Linux Command Line**
https://linuxcommand.org/tlcl.php
Original optional Lab01 book, legally available from the author, for extended command-line study.

## 11.3 Tutorials and Practical Examples

**Debian Reference  Package Management**
https://www.debian.org/doc/manuals/debian-reference/ch02.en.html
Intermediate guide to Debian package operations and repository concepts.

**Ubuntu Server  Package Management**
https://ubuntu.com/server/docs/package-management/
Beginner-to-intermediate operational examples for APT on Ubuntu Server.

**Bash Manual  Job Control**
https://www.gnu.org/software/bash/manual/html_node/Job-Control.html
Intermediate examples and exact shell job-control semantics.

**Bash Manual  Pipelines**
https://www.gnu.org/software/bash/manual/html_node/Pipelines.html
Intermediate reference for pipeline construction and return status.

**Linux man-pages  proc_pid_status(5)**
https://man7.org/linux/man-pages/man5/proc_pid_status.5.html
Intermediate field-by-field reference for inspecting a process through procfs.

## 11.4 Further Reading

**POSIX Shell Command Language**
https://pubs.opengroup.org/onlinepubs/9799919799/utilities/V3_chap02.html
The current Open Group specification for portable shell language semantics.

**Linux capabilities(7)**
https://man7.org/linux/man-pages/man7/capabilities.7.html
Explains how Linux decomposes traditional superuser privilege into capability units.

**Linux namespaces(7)**
https://man7.org/linux/man-pages/man7/namespaces.7.html
Architectural basis for the isolated views used by containers.

**systemd tmpfiles Manual**
https://www.freedesktop.org/software/systemd/man/latest/tmpfiles.d.html
A declarative alternative for creation, ownership, aging, and cleanup of volatile files.

## 11.5 Useful Tools

**ShellCheck**
https://www.shellcheck.net/
Static analysis for shell scripts; especially useful when one-liners evolve into Lab 3 automation.

**strace**
https://strace.io/
System-call tracing for connecting user-visible failures to kernel operations and errno values.

**procps-ng**
https://gitlab.com/procps-ng/procps
Upstream project for ps, top, free, uptime, and related process-observation tools.

**acl utilities**
https://savannah.nongnu.org/projects/acl
Project providing getfacl and setfacl for POSIX access control lists on Linux filesystems.

# 12 Sources Used in This Material

| Material section | Main sources |
|---|---|
| Filesystem hierarchy and procfs | FHS 3.0; proc(5); namespaces(7). |
| Credentials and permission checks | credentials(7); inode(7); chmod(2); GNU Coreutils manual. |
| Processes, job control, and signals | credentials(7); signal(7); GNU Bash Reference Manual. |
| Pipelines and redirections | GNU Bash Reference Manual; GNU Coreutils manual; POSIX Shell Command Language. |
| Safe traversal and cleanup | GNU Findutils Manual; systemd tmpfiles manual. |
| Package management | APT User Guide; Debian Reference; Ubuntu Server package-management documentation. |

# 13 Key Takeaways

- Linux paths express a namespace; FHS conventions communicate lifecycle and purpose, while mounts and namespaces determine the actual view.
- Access is evaluated using process credentials, pathname traversal, one traditional mode class, and any additional security mechanisms.
- A shell job can contain multiple processes; process groups and sessions explain terminal control more accurately than PID-only thinking.
- SIGTERM is a request that enables graceful shutdown; SIGKILL is an uncatchable last resort with operational consequences.
- Pipelines are concurrent dataflows over file descriptors, and reliable automation must make upstream failure visible.
- Destructive cleanup begins with a constrained test tree and an exact dry run; retention policy is an engineering interface, not merely a command.
- Package state depends on repositories, metadata, versions, architecture, and time; reproducibility requires controlling those inputs.
- Good DevOps evidence records observable outcomes, assumptions, verification, team contributions, and responsible AI use.