19

Linux and Command Line

Your Production Survival Kit

THE MODERN QA ENGINEER SKILLSET

Linux and Command Line: Your Production Survival Kit

Shell Skills Every QA Engineer Needs for Modern Production Systems

Yuri Syuganov

Modern QA Course

https://modernQAcourse.com

Linux and Command Line: Your Production Survival Kit

Shell Skills Every QA Engineer Needs for Modern Production Systems

Book 19 of 26 in the series *The Modern QA Engineer Skillset*

First Edition · Version 1.0.0 · 2026-07-11

Set in Charter, IBM Plex Sans, and IBM Plex Mono. Typeset with Pandoc and XeLaTeX.

Modern QA Course · https://modernQAcourse.com · support@modernQAcourse.com

Contents

Introduction

About This Book

This book is a practical, hands-on guide to Linux and the command line for software professionals who work with modern production systems. It is not a theoretical operating-systems textbook. Every command, every script, and every exercise is drawn from real-world scenarios that QA engineers, developers, and DevOps practitioners encounter daily: debugging failed CI pipelines, analyzing application logs at 2 AM, spinning up isolated test environments with Docker, and automating the repetitive tasks that steal hours from your week.

Test environments run on Linux. CI/CD pipelines run on Linux. Docker containers run on Linux. Application servers, databases, and log aggregators run on Linux. A professional who cannot navigate a terminal is limited to whatever the GUI tools expose. Command-line skills let you investigate issues directly, automate repetitive tasks, and work effectively in environments where there is no graphical interface.

The book moves from foundational concepts (where files live, how permissions work) through essential daily-driver commands (grep, curl, jq, awk, sed) to advanced automation (Bash scripting, Docker orchestration, log analysis pipelines). Along the way, you will find 100 progressively challenging exercises, real log snippets to practice with, and a capstone project that ties everything together into a practical toolkit.

Who This Book Is For

- **QA Engineers** who need to debug test failures in CI/CD pipelines, analyze server logs, and automate testing workflows
- **Junior Developers** transitioning from GUI-only workflows to command-line proficiency
- **Manual Testers** moving into automation and needing to understand the infrastructure their tests run on
- **DevOps Beginners** building foundational Linux and scripting skills
- **Bootcamp Graduates** who learned to write code but not how to operate it in production environments
- **Anyone** who has seen a terminal and wants to stop being afraid of it

Prerequisites

- **A computer with a terminal.** macOS and Linux have one built in. Windows users should install WSL2 (Windows Subsystem for Linux) or use Git Bash.
- **Basic computer literacy.** You should be comfortable creating files, installing software, and using a web browser.
- **No prior Linux or command-line experience is required.** This book starts from scratch.
- **Curiosity and willingness to type commands.** The best way to learn the command line is to use it. Open a terminal and type every command in this book.

How to Use This Book

Start with Part I (Linux Basics) to build a foundation. Move into Part II (Essential Commands) for the tools you will use daily. Finish with Part III (Scripting and Containers) for automation and environment management. The exercises increase in difficulty within each chapter, labeled as Beginner, Intermediate, or Advanced.

Do not just read. Type every command. Break things. Fix them. That is how command-line proficiency develops.

Part I: Linux Fundamentals

CHAPTER 1

The Linux Directory Structure

1.1 The Filesystem Hierarchy

Linux organizes everything as files in a single directory tree starting from the root `/`. There is no `C:\` drive, no `D:\` drive. Everything – your documents, your configurations, your hardware devices, even running processes – is represented as a file somewhere under `/`. Understanding this structure helps you navigate test environments, find configuration files, and locate logs quickly.

Here is the standard Linux filesystem layout:

```
/                   Root of the filesystem
├── home/user/      Your home directory (~)
├── etc/            Configuration files
├── var/
│   ├── log/        Application and system logs
│   └── lib/        Variable data (databases, packages)
├── tmp/            Temporary files (cleared on reboot)
├── opt/            Optional/third-party software
├── usr/
│   ├── bin/        User-installed binaries
│   └── local/      Locally compiled software
├── proc/           Virtual filesystem (process and kernel info)
├── dev/            Device files
└── srv/            Service data (web servers, FTP)
```

Each directory has a specific purpose. Knowing what lives where means you never have to guess where to look when something goes wrong.

1.2 Directories You Will Use Most

/home/user/ – Your Home Directory (~)

This is where you work. Your scripts, test configurations, SSH keys, and shell customizations live here. The tilde (~) is a shortcut for your home directory path.

```
# Navigate to home directory
cd ~
# Or simply:
cd

# Common QA files in home directory
~/.ssh/config          # SSH connection shortcuts
~/.bashrc              # Shell customization (aliases, PATH)
~/.npmrc               # npm configuration
~/.env                 # Personal environment variables (not committed)
~/projects/            # Your cloned repositories
```

Your `.bashrc` (or `.zshrc` on macOS) is one of the most important files here. It runs every time you open a new terminal. This is where you put aliases, custom PATH entries, and shell functions that make your daily work faster.

/var/log/ – Logs

This is where you investigate failures. Application logs, system logs, and web server logs all live here. When a test fails at 2 AM and you need to figure out what happened, this directory is your first stop.

```
# Common log locations
/var/log/syslog                    # System messages
/var/log/auth.log                  # Authentication events
/var/log/nginx/access.log          # Nginx access logs
/var/log/nginx/error.log           # Nginx error logs
/var/log/app/application.log       # Application-specific logs
/var/log/postgresql/postgresql.log  # Database logs
```

> **Pro Tip:** Log files can grow enormous. Never open a multi-gigabyte log file in a text editor. Use `tail`, `grep`, `less`, and `awk` to extract only the lines you need.

/etc/ – Configuration

Configuration files for system services and applications. When tests fail due to misconfiguration, this is where you look.

```
# Common configuration files
/etc/hosts                # DNS overrides (useful for test environments)
/etc/nginx/nginx.conf     # Nginx web server config
/etc/environment          # System-wide environment variables
/etc/postgresql/pg_hba.conf  # PostgreSQL access control
/etc/ssl/certs/           # SSL certificates
```

> **Common Mistake:** Editing files in `/etc/` usually requires `sudo` (administrator) privileges. If you get "Permission denied" when trying to edit a config file here, prepend `sudo` to your command. But be careful – changes in `/etc/` affect the entire system.

/tmp/ – Temporary Files

Temporary data that is cleared on reboot. Useful for staging test data or storing intermediate results during test runs.

```
# Use /tmp for temporary test data
cp test-data.json /tmp/
# Run tests that use /tmp/test-data.json
# On reboot, the file is automatically cleaned up
```

/proc/ – Virtual Filesystem

Not real files – this is a window into the kernel and running processes. Useful for debugging performance issues and inspecting running processes.

```
# Check system memory
cat /proc/meminfo | head -5

# Check CPU information
cat /proc/cpuinfo | grep "model name" | head -1

# Check a specific process (replace 1234 with actual PID)
cat /proc/1234/status | grep -E "(Name|State|VmRSS)"
```

1.3 Navigating the Filesystem

Basic Navigation Commands

These are the commands you will type hundreds of times per day. They need to become muscle memory.

```
# Print current directory (where am I right now?)
pwd

# List files (basic)
ls

# List files with details (permissions, size, date)
ls -la

# List files sorted by modification time (newest first)
ls -lt

# List files with human-readable sizes
ls -lh

# Change directory
cd /var/log       # Go to an absolute path
cd ..             # Go up one level
cd -              # Go to previous directory (toggle between two directories)
cd ~/projects     # Go to projects in home directory
```

> **Pro Tip:** `cd -` is incredibly useful when you are switching between two directories. It toggles back and forth like the "last channel" button on a TV remote.

Finding Files

When you know what you are looking for but not where it lives:

```
# Find a file by name (recursive search from root)
find / -name "playwright.config.ts" 2>/dev/null

# Find files modified in the last 24 hours
find /var/log -mtime -1 -type f

# Find files larger than 100MB (hunt for disk space issues)
find / -size +100M -type f 2>/dev/null

# Find all .spec.ts files in the project
find ~/projects/webapp -name "*.spec.ts" -type f

# Faster alternative: locate (uses a pre-built index)
locate playwright.config.ts
```

The `2>/dev/null` at the end of those commands suppresses "Permission denied" errors that occur when `find` tries to read directories you do not have access to. You will learn exactly what this syntax means in the Redirection chapter.

Disk Usage

Disk full is a surprisingly common cause of test failures. These commands help you diagnose it:

```
# Disk usage summary (all mounted filesystems)
df -h

# Directory size
du -sh /var/log/

# Top 10 largest directories
du -h /var/log/ | sort -rh | head -10

# Check if disk is nearly full (common cause of test failures)
df -h | grep -E "9[0-9]%|100%"
```

> **Common Mistake:** Running `du` on the root directory (`/`) without limiting depth will take a very long time and produce enormous output. Always target a specific directory or use `du -h --max-depth=1 /` to limit the depth.

1.4 File Operations

These commands form the foundation of all file manipulation:

```
# Create directories (including parents)
mkdir -p tests/integration/api

# Copy files
cp source.json destination.json
cp -r tests/ tests-backup/       # Recursive copy (for directories)

# Move/rename files
mv old-name.ts new-name.ts
mv test-results/ archive/test-results-sprint23/

# Remove files
rm unwanted-file.txt
rm -r old-directory/              # Recursive remove
rm -rf node_modules/              # Force recursive remove (use carefully!)

# Create an empty file
touch new-test.spec.ts

# View file content
cat small-file.txt                # Entire file (for small files only)
head -20 large-file.log           # First 20 lines
tail -50 large-file.log           # Last 50 lines
tail -f /var/log/app.log          # Follow file in real-time (live log monitoring)
less large-file.log               # Paginated viewer (press q to quit)
```

> **Common Mistake:** `rm -rf` is the most dangerous command in Linux. It recursively force-deletes everything in the specified path with no confirmation and no recycle bin. Double-check your path before pressing Enter. `rm -rf /` will attempt to delete your entire filesystem. Some horror stories involve a misplaced space: `rm -rf / tmp/old-files` (deletes root!) versus `rm -rf /tmp/old-files` (deletes only the temp directory).

1.5 Understanding Paths

Absolute vs. Relative Paths

There are two ways to refer to any file or directory:

```
# Absolute path: starts from root (/)
# Unambiguous -- works no matter what your current directory is
/home/qa/projects/webapp/tests/login.spec.ts

# Relative path: relative to current directory
# Shorter, but depends on where you are
tests/login.spec.ts               # If you are in /home/qa/projects/webapp/
../webapp/tests/login.spec.ts     # If you are in /home/qa/projects/api/
```

Special Path Symbols

Symbol	Meaning	Example
`/`	Root directory	`cd /`
`~`	Home directory	`cd ~/projects`
`.`	Current directory	`cp ./config.json /tmp/`
`..`	Parent directory	`cd ..`
`-`	Previous directory	`cd -`

> **Pro Tip:** In scripts, always use absolute paths. Relative paths break when the script is executed from a different working directory.

1.6 Exercises

Beginner

1. Open a terminal and run `pwd` to see your current directory. Then navigate to `/var/log/` and list the files. Identify which logs belong to which service.
2. Navigate to your home directory and list all files, including hidden ones (`ls -la`). Identify the dotfiles (files starting with `.`) and guess what each one is for.
3. Create the following directory structure using a single `mkdir -p` command: `~/qa-practice/tests/unit/`, `~/qa-practice/tests/integration/`, `~/qa-practice/tests/e2e/`.

Intermediate

4. Find all `.log` files modified in the last 24 hours anywhere on the system using `find`.
5. Check disk usage on your machine with `df -h`. Which filesystem is the fullest? Use `du -h --max-depth=1` on a large directory to find what is taking up space.
6. Create a file in `/tmp/`, verify it exists, reboot your machine (or wait until the next reboot), and check if the file still exists. Explain why.

Advanced

7. Use `tail -f` to monitor a log file in real-time while performing an action that generates log entries in another terminal window.
8. Write a command that finds the 10 largest files anywhere on the system and displays their sizes in human-readable format.
9. Use `find` to locate all files larger than 50MB that have not been accessed in the last 30 days. Consider how this could be used for disk cleanup on CI runners.

Career Translation

Resume phrasing

- Navigated and managed Linux filesystem structures across CI/CD runners and production servers, reducing average incident investigation time by 40%.
- Diagnosed and resolved disk-space-related test failures on shared CI infrastructure by implementing targeted `find` and `du` cleanup routines.
- Maintained standardized directory layouts for test artifacts, configurations, and logs across multiple deployment environments.

Cover letter framing

Understanding the Linux filesystem is the foundation of every infrastructure investigation I perform. When a CI pipeline fails or a production alert fires, I know exactly where configuration files, logs, and temporary artifacts live – and I can navigate to the relevant data in seconds rather than

minutes. This translates directly into faster incident resolution and less downtime for the team.

Interview framing

"I approach filesystem navigation as the first layer of any debugging workflow. Before I grep a log or inspect a process, I need to know where things live – `/var/log` for logs, `/etc` for config, `/tmp` for ephemeral data, `/proc` for runtime kernel info. I optimize for speed by building muscle memory around `cd -`, `find`, and `du`, and I always use absolute paths in scripts to avoid working-directory bugs. The trade-off between `find` and `locate` is freshness versus speed – I pick the right tool depending on whether I need real-time results or a fast index lookup."

What not to say

- "I just use the GUI file manager." – Signals you cannot operate in headless server environments where most production systems run.
- "I usually Google which directory to look in." – Shows a lack of foundational knowledge that every senior engineer is expected to have internalized.
- "I don't really worry about disk space." – Disk-full failures are one of the top causes of CI flakiness and production outages; ignoring them is a red flag.
- "I use `ls -R /` to find things." – Demonstrates unfamiliarity with `find`, `locate`, and efficient search strategies.

Interview Depth Check

Question 1

Prompt: A CI pipeline that was green for weeks starts failing intermittently with "No space left on device" errors. The CI runners are shared across teams. Walk me through how you would diagnose and fix this using only command-line tools.

What a strong answer should cover:

- Using `df -h` to confirm which filesystem is full
- Using `du -h --max-depth=1` to drill down into the largest directories
- Checking common culprits: Docker images (`docker system df`), old test artifacts, build caches, `/tmp` accumulation
- Distinguishing between a one-time cleanup and a systemic fix (scheduled cleanup scripts, artifact retention policies)

Example answer:

- First I would SSH into the runner and run `df -h` to confirm the disk is genuinely full and identify which mount point is affected.
- Then I would run `du -h --max-depth=1 /` to see which top-level directories are consuming the most space. Common offenders are `/var/lib/docker`, `/tmp`, and user home directories with cached build artifacts.
- I would check Docker specifically with `docker system df` since dangling images and unused volumes accumulate fast on shared runners.
- For the immediate fix, I would run targeted cleanup – `docker system prune -a`, `find /tmp -mtime +7 -delete`, and remove old test-result directories.
- For the long-term fix, I would add a cleanup step to the CI pipeline configuration and propose artifact retention policies so this does not recur.

Question 2

Prompt: You are debugging a test failure on a staging server. The application writes logs to a non-standard location and you do not know where. How would you find the log files?

What a strong answer should cover:

- Using `find` with name patterns like `*.log` or time-based filters (`-mmin -30`)
- Checking the application's configuration in `/etc/` or environment variables for log path settings

- Using `lsof` to see which files the running process has open
- Checking process information in `/proc/<PID>/fd` for open file descriptors

Example answer:

- I would start by checking the application's config files in `/etc/` or the deployment directory for a `LOG_DIR` or `LOG_PATH` setting.
- If that does not yield results, I would find the application's PID with `ps aux | grep <app-name>` and then check its open file descriptors with `ls -la /proc/<PID>/fd` or `lsof -p <PID>` – this shows every file the process has open, including log files.
- As a broader sweep, I would run `find / -name "*.log" -mmin -30 2>/dev/null` to find any log file modified in the last 30 minutes.
- I could also check environment variables of the running process with `cat /proc/<PID>/environ | tr '\0' '\n' | grep -i log` to see if a log path was configured at startup.

Question 3

Prompt: Explain the difference between `/tmp`, `/var/tmp`, and a project-level `tmp/` directory. When would you use each in a QA context, and what are the risks of each?

What a strong answer should cover:

- `/tmp` is cleared on reboot (or periodically by systemd-tmpfiles), suitable for truly ephemeral data
- `/var/tmp` persists across reboots, suitable for data that should survive a restart but is still temporary
- Project-level `tmp/` is version-control-aware and scoped to the project, but risks being committed accidentally
- Risk analysis: `/tmp` data loss on reboot, `/var/tmp` accumulation without cleanup, project `tmp/` polluting git history

Example answer:

- `/tmp` is OS-managed and cleared on reboot. I use it for throwaway test data that I need only for the duration of a single test run – staging JSON fixtures, intermediate processing files, or temporary downloads. The risk is that a reboot mid-run loses everything.
- `/var/tmp` survives reboots. I use it when a long-running process might span a reboot, like a multi-hour soak test generating artifacts. The risk is that nobody cleans it up, so it can accumulate silently.
- A project-level `tmp/` directory is useful for test artifacts that should be scoped to the project and easy to find, but I always add it to `.gitignore`. The risk is forgetting that and committing large binary files to version control.
- In CI specifically, I prefer `/tmp` because the runner is ephemeral anyway, and it avoids permission issues that sometimes occur with project-level directories when the CI user differs from the checkout user.

Question 4

Prompt: A junior team member accidentally ran `rm -rf` on a directory they did not intend to delete on a staging server. What is your response plan, and what preventive measures would you put in place?

What a strong answer should cover:

- Immediate assessment: what was deleted, is it recoverable from backups or version control
- Checking if the filesystem supports undelete or if snapshots exist
- Preventive measures: aliases for `rm` with confirmation, restricted sudo access, filesystem snapshots, immutable infrastructure
- The difference between "we can recover" and "we need to rebuild"

Example answer:

- First I would assess the damage: what directory was deleted and is the data in version control or backed up. If it is application code, a fresh `git clone` restores it. If it is data or logs, I check for filesystem snapshots or backup schedules.

- For immediate recovery, I would check if the server uses LVM snapshots or ZFS snapshots that can be rolled back. On cloud infrastructure, I would check if there is a recent volume snapshot.
- For prevention, I would implement several layers: add `alias rm='rm -i'` to shared server profiles so `rm` always asks for confirmation, restrict `sudo` access so junior engineers cannot operate as root, use immutable infrastructure where possible so servers are rebuilt from images rather than modified in place, and set up automated backups with tested restore procedures.
- The deeper lesson is that staging servers should be treated as disposable. If a single `rm` command can cause a crisis, the environment setup is not automated enough.

About This Sample

You have just read **Chapter 1 of Linux and Command Line: Your Production Survival Kit** — the free sample. The complete book continues with every remaining chapter, the interview Q&A, and all of the code.

Get the full book

Sample readers get a discount at
https://leanpub.com/modernqa-19-linux-command-line/c/SAMPLE10

Read online first

Chapter one of **every** book in the series is free to read at
https://modernqacourse.com/library

About the Author

Yuri Syuganov is a software quality engineer with more than twenty years of hands-on testing experience, spanning the arc from manual test design through Selenium-era automation to today's AI-augmented, agent-driven frameworks. He is the author of The Modern QA Engineer Skillset, a 26-book series built on a blunt analysis of what companies actually expect from QA engineers, and the creator of modernQAcourse.com, where the course behind the series is free and readers can bring real testing problems to experienced practitioners. He revises the series as the industry changes — because it does.

Also in This Series

The Modern QA Engineer Skillset — a 26-book course in modern software quality assurance.

AI-Augmented QA (Books 1–10)

- **Book 1:** Agent Skills for Browser Automation: The Complete Guide
- **Book 2:** AI-Augmented Test Design: From Prompts to Test Suites
- **Book 3:** Agentic Testing Architectures: Building Autonomous QA Systems
- **Book 4:** API and Contract Testing with AI: Schema to Shield
- **Book 5:** Performance and Chaos Engineering: Breaking Things on Purpose
- **Book 6:** Observability-Driven Testing: Testing in Production
- **Book 7:** Security Testing for AI Applications: The New Attack Surface
- **Book 8:** Infrastructure as Code Testing: Terraform to Kubernetes
- **Book 9:** Mobile and Cross-Platform Testing: 24,000 Devices, One Strategy
- **Book 10:** Visual and Accessibility Testing: Pixels and People

Technical Foundations (Books 11–19)

- **Book 11:** Manual Testing Fundamentals: The Bedrock
- **Book 12:** Programming for QA: Python, JavaScript, and TypeScript from Zero to Fluent
- **Book 13:** Browser Test Automation: Playwright, Patterns, and the Death of Fragile Tests

- **Book 14:** API Testing Fundamentals: Postman to Pytest
- **Book 15:** SQL and Database Testing: Query Your Way to Quality
- **Book 16:** CI/CD Pipelines: From Commit to Confidence
- **Book 17:** Git and Version Control: Beyond Push and Pull
- **Book 18:** Test Management Tools: Making Quality Visible
- **Book 19:** Linux and Command Line: Your Production Survival Kit

Professional Skills (Books 20–26)

- **Book 20:** Agile and Scrum for QA: Quality in Every Sprint
- **Book 21:** Communication and Stakeholder Management for QA: Influence Without Authority
- **Book 22:** Test Strategy and Quality Metrics: Measuring What Matters
- **Book 23:** QA Leadership and Mentoring: Multiplying Your Impact
- **Book 24:** Technical Writing for QA: Documents That Drive Action
- **Book 25:** Interview Preparation and Career Strategy: Land the Job
- **Book 26:** Testing Like a Senior: Patterns & Anti-Patterns Across Software QA Career Levels

Visit https://modernQAcourse.com for the full course catalog, updates, and companion resources.

Stuck on a Real Problem?

The full course behind this series is free to read at https://modernQAcourse.com — and it stays free.

When a chapter meets the real world and something breaks, post it on the **Help Board** at https://modernQAcourse.com. Asking is free, and experienced practitioners answer right in the thread. Need hands-on help? Book a 1:1 screen-share session with a QA expert.