Linux for DevOps & Cloud Engineers

Hands-On Practice Labs

TS ACADEMY

All labs tested and 100% safe for beginners

Recommended VM: Ubuntu 22.04 / 24.04 LTS

You have `sudo` access

Oluwatobiloba Durodola 2025 Batch

December 3, 2025

"The best way to learn Linux is to break things... safely."

Table of Contents

Contents

1 Introduction

Welcome to the **Linux for DevOps & Cloud Engineers** hands-on labs! These exercises are designed to take you from Linux beginner to confident sysadmin who can:

- Navigate production servers like a pro
- Debug issues at 2 AM without panic
- Manage services and containers confidently
- Troubleshoot network and performance problems

1.1 Prerequisites

- Ubuntu 22.04/24.04 LTS VM (or any Debian-based distro)
- Terminal access with `sudo` privileges
- Internet connection for package installation
- 4-6 hours for all labs (do them in any order!)

1.2 How to Use This Lab Book

1. Read the Goal for each lab
2. Follow commands in order (copy-paste into terminal)
3. **Type everything manually** when you see TRY IT
4. Check Expected Results to verify
5. Experiment! Break things safely and learn from errors

- Never run `rm -rf /` or `rm -rf /home`
- Always test scripts in `/tmp` first
- Use `man command` when stuck
- These labs won't break your VM

2 LAB 1: Filesystem Navigation & Manipulation

Master the essential commands for creating, moving, copying, and deleting files and directories. These are the building blocks of every DevOps task.

2.1 Task Overview

Create a realistic project structure, manipulate files, and practice safe deletion.

2.2 Step-by-Step Instructions

Listing 1: Create Project Structure

```
# 1. Create nested directories (project      app      logs)
mkdir -p ~/project/app/logs ~/project/config ~/project/scripts
```

Listing 2: Create Sample Files

```
# 2. Navigate to app directory and create 5 files
cd ~/project/app
touch index.html style.css app.py README.md config.json

# Verify creation
ls -la
```

Listing 3: Move & Copy Files

```
# 3. Move config.json to logs directory
mv config.json logs/

# 4. Copy README.md to config directory
cp README.md ../config/

# 5. Verify the moves
ls -la logs/
ls -la ../config/
```

Listing 4: Safe Backup & Cleanup

```
# 6. Create project backup
cp -r ~/project ~/project-backup-$(date +%Y%m%d)

# 7. Verify backup exists
ls -ld ~/project-backup*

# 8. Install tree utility (optional but helpful)
sudo apt update && sudo apt install tree -y

# 9. Visualize your structure
tree ~/project
```

2.3 Expected Results

- `/home/student/project/app/` contains 4 files
- `logs/` contains `config.json`
- `config/` contains `README.md`
- `tree` shows complete structure

Pro Tip

Always use `mkdir -p` for nested directories. The `-p` flag creates parent directories if they don't exist and never fails if they do.

3 LAB 2: File Permissions & Ownership

Understand Linux permissions and ownership - the #1 reason deployments fail in production.

3.1 Real-World Scenario

Create a deployment script that needs root privileges, then debug permission issues.

Listing 5: Create Deployment Script

```
# 1. Create a realistic deployment script
cat > ~/deploy.sh << 'EOF'
#!/bin/bash
echo "=== Deployment Started at $(date) ==="
echo "This script would:"
echo "1. Pull latest code from Git"
echo "2. Run database migrations"
echo "3. Restart services"
echo "4. Notify Slack channel"
echo "=== Deployment Complete ==="
EOF
```

Listing 6: Test Permissions

```
# 2. Check current permissions
ls -la ~/deploy.sh

# 3. Make it executable
chmod +x ~/deploy.sh

# 4. Run the script
./deploy.sh

# 5. Remove execute permission
chmod -x ~/deploy.sh

# 6. Try running again (should fail)
./deploy.sh
```

Listing 7: Root Ownership Simulation

```
# 7. Make script root-owned (production scenario)
sudo chown root:root ~/deploy.sh
sudo chmod 755 ~/deploy.sh

# 8. Check ownership
ls -la ~/deploy.sh

# 9. Try running as normal user
./deploy.sh
```

Listing 8: sudo Solution

```
# 10. Run with sudo (production fix)
sudo ./deploy.sh
```

```

# 11. Clean up
sudo chown $USER:$USER ~/deploy.sh
rm ~/deploy.sh
```

3.2 Expected Results

- Script runs successfully with `chmod +x`
- Fails with `permission denied` after `chmod -x`
- Fails again when owned by root, even with 755 permissions
- Succeeds with `sudo ./deploy.sh`

Never deploy with root-owned scripts! Use service accounts and `sudoers` configuration instead.

4 LAB 3: Process Management

Learn to identify, monitor, and terminate processes - essential for debugging stuck deployments.

Listing 9: Start Background Processes

```bash
# 1. Start ping in background (simulates long-running process)
ping 8.8.8.8 > /dev/null &

# 2. Start another process
sleep 1000 &

# 3. Check background jobs
jobs
```

Listing 10: Find Process IDs

```bash
# 4. Find ping process three ways
ps aux | grep ping
pgrep ping
ps -ef | grep ping
```

Listing 11: Graceful Termination

```bash
# 5. Kill processes gracefully
kill $(pgrep ping)
kill %1  # kill job 1

# 6. Verify they're gone
pgrep ping
jobs
```

Listing 12: Force Kill Practice

```bash
# 7. Start a stubborn process
ping 8.8.8.8 > /dev/null &

# 8. Find its PID
PID=$(pgrep ping)
echo "Ping PID: $PID"

# 9. Try graceful kill first
kill $PID
sleep 2

# 10. Check if still running
pgrep ping

# 11. Force kill if needed
kill -9 $PID

# 12. Verify termination
pgrep ping || echo "Process terminated successfully"
```

4.1 Expected Results

- `jobs` shows background processes
- `ps aux | grep ping` finds the process
- `kill PID` terminates gracefully
- `kill -9 PID` force-kills stubborn processes

Pro Tip

Always try `kill PID` first. Only use `kill -9` as last resort - it can leave resources uncleaned.

5 LAB 4: Service Management with Systemd

Install and manage Nginx - the most common web server in DevOps.

Listing 13: Install & Start Nginx

```
# 1. Update package lists
sudo apt update

# 2. Install Nginx
sudo apt install nginx -y

# 3. Start the service
sudo systemctl start nginx

# 4. Enable at boot
sudo systemctl enable nginx

# 5. Check status
sudo systemctl status nginx
```

Listing 14: Test the Installation

```
# 6. Test that Nginx is serving
curl localhost
curl -I http://127.0.0.1

# 7. Check Nginx process
ps aux | grep nginx

# 8. View Nginx logs
sudo tail -f /var/log/nginx/access.log
```

Listing 15: Service Management Practice

```
# 9. Reload configuration (no downtime)
sudo systemctl reload nginx

# 10. Restart service (with downtime)
sudo systemctl restart nginx

# 11. Stop service
sudo systemctl stop nginx

# 12. Check it's stopped
sudo systemctl status nginx

# 13. Start it again
sudo systemctl start nginx
```

Listing 16: Cleanup

```
# 14. Remove Nginx completely
sudo systemctl stop nginx
sudo systemctl disable nginx
sudo apt purge nginx -y
sudo apt autoremove -y
```

5.1 Expected Results

- `curl localhost` returns Nginx welcome page
- `systemctl status nginx` shows `active (running)`
- Service stops/starts/restarts without errors

In production, always use `systemctl reload` for config changes to avoid downtime.

6 LAB 5: Networking Diagnostics

Master the essential networking commands every DevOps engineer needs.

Listing 17: Check Network Interfaces

```
# 1. View all network interfaces
ip a

# 2. Check routing table
ip route

# 3. Show network connections
ss -tuln
```

Listing 18: Connectivity Testing

```
# 4. Test DNS resolution
nslookup google.com

# 5. Ping multiple destinations
ping -c 4 google.com
ping -c 4 github.com
ping -c 4 8.8.8.8

# 6. Test with packet loss stats
ping -c 10 google.com
```

Listing 19: HTTP Testing

```
# 7. Download a webpage
curl -s https://httpbin.org/html | head -20

# 8. Test API endpoint
curl -s https://api.github.com | jq '.current_user_url' 2>/dev/null ||
    echo "jq not installed"

# 9. Download a file
curl -O https://httpbin.org/robots.txt
ls -la robots.txt
rm robots.txt
```

Listing 20: Advanced Network Tools

```
# 10. Install and use nmap
sudo apt install nmap -y
nmap localhost

# 11. Check listening services
sudo netstat -tlnp 2>/dev/null || sudo ss -tlnp

# 12. Test port connectivity
nc -zv localhost 80 2>/dev/null || echo "Port 80 not accessible"
```

6.1 Expected Results

- `ip a` shows your network interface with IP
- `ping` succeeds with <1
- `curl` returns HTML content
- `nmap localhost` shows open ports

7 LAB 6: Users & Groups Management

Create users and groups for team-based deployments.

Listing 21: Create Development Team

```
# 1. Create developers group
sudo groupadd developers

# 2. Create devops engineer user
sudo useradd -m -G developers -s /bin/bash devops1
sudo passwd devops1  # Set password: devops123

# 3. Create another team member
sudo useradd -m -G developers -s /bin/bash developer2
sudo passwd developer2  # Set password: dev123
```

Listing 22: Verify User Setup

```
# 4. Check user information
id devops1
groups devops1

# 5. Check user home directories
ls -ld /home/dev*

# 6. View user accounts
cat /etc/passwd | grep -E "(devops1|developer2)"
```

Listing 23: User Switching

```
# 7. Switch to devops1 user
su - devops1
whoami
pwd
exit

# 8. Switch to developer2
su - developer2
id
groups
exit
```

Listing 24: Team Permissions

```
# 9. Create shared project
sudo mkdir -p /opt/team-project
sudo chown :developers /opt/team-project
sudo chmod 775 /opt/team-project

# 10. Test group access
ls -ld /opt/team-project
su - devops1 -c "touch /opt/team-project/test.txt"
```

Listing 25: Cleanup

```
# 11. Remove users and group
sudo userdel -r devops1
sudo userdel -r developer2
sudo groupdel developers
sudo rm -rf /opt/team-project
```

7.1 Expected Results

- `id devops1` shows membership in developers group
- `su - devops1` switches user successfully
- `devops1` can write to `/opt/team-project/`

8 LAB 7: Log Analysis

Master log navigation and analysis - 80

Listing 26: Explore System Logs

```
# 1. View authentication log
sudo ls -la /var/log/auth.log*

# 2. Show recent log entries
sudo tail -20 /var/log/auth.log

# 3. Real-time log monitoring
sudo tail -f /var/log/syslog &
```

Listing 27: Search Failed Logins

```
# 4. Find failed authentication attempts
sudo grep -i "fail" /var/log/auth.log | tail -10

# 5. Count failed logins today
sudo grep "$(date +%b\ %d)" /var/log/auth.log | grep -i fail | wc -l

# 6. Show failed login details
sudo grep "Failed password" /var/log/auth.log | tail -5
```

Listing 28: Advanced Log Filtering

```
# 7. Generate some log activity (in another terminal)
# Try wrong password 3 times, then login successfully

# 8. Filter syslog for errors only
sudo grep -i error /var/log/syslog | tail -10

# 9. Monitor service-specific logs
sudo journalctl -u cron -f  # Press Ctrl+C to stop
```

Listing 29: Log Analysis Practice

```
# 10. Create sample application log
cat > /tmp/app.log << 'EOF'
2025-01-15 10:30:15 ERROR Database connection failed
2025-01-15 10:30:16 INFO User login successful
2025-01-15 10:30:17 WARN Slow query detected
2025-01-15 10:30:18 ERROR Payment gateway timeout
2025-01-15 10:30:19 INFO Cache hit ratio: 85%
EOF

# 11. Analyze the log
grep ERROR /tmp/app.log
grep -v INFO /tmp/app.log | wc -l

# 12. Cleanup
rm /tmp/app.log
```

8.1 Expected Results

- `tail -f` shows real-time log entries
- `grep ERROR` finds error messages
- Log analysis commands return expected counts

Debugging Pro Tip

Always start with `tail -f` to see what's happening NOW, then use `grep` to find patterns in history.

9 LAB 8: Vim Crash Course

Edit configuration files like a Linux professional (because nano isn't production-ready).

Listing 30: Create Configuration File

```
# 1. Create application configuration
cat > ~/app.conf << 'EOF'
# Production Configuration
APP_ENV=production
APP_PORT=8000
DB_HOST=localhost
DB_PORT=5432
DB_NAME=app_production
DB_USER=app_user
DB_PASS=secret123
DEBUG=false
LOG_LEVEL=info
```

Listing 31: Basic Vim Editing

```
# 2. Open file in vim
vim ~/app.conf

# 3. Inside vim, try these commands:
# i                Enter INSERT mode
# Esc              Return to NORMAL mode
# :w               Save (write)
# :q               Quit
# :wq              Save and quit
# :q!              Force quit without saving
```

Listing 32: Practical Editing Exercise

```
# 4. Edit the configuration:
# - Change APP_PORT from 8000 to 9000
# - Change DB_PASS to SuperSecret2025!
# - Set DEBUG=true
# - Change LOG_LEVEL to debug

# Commands you'll use:
# /APP_PORT          Search for APP_PORT
# n                  Next search result
# x                  Delete character under cursor
# i                  Insert before cursor
# a                  Append after cursor
# dd                 Delete current line
# yy                 Copy (yank) line
# p                  Paste line
```

Listing 33: Vim Navigation

```
# 5. Practice navigation:
# h j k l           Left down up right
# w                 Next word
# b                  Previous word
# 0                  Start of line
# $                  End of line
# gg                 Start of file
# G                  End of file
# :10                Go to line 10
# Ctrl+G             Show current line number
```

Listing 34: Save Your Work

```
# 6. Save and exit
:wq

# 7. Verify changes
cat ~/app.conf

# 8. Cleanup
rm ~/app.conf
```

9.1 Vim Cheat Sheet

Vim Quick Reference

Mode	Keys
Normal	Default mode - navigation & commands
Insert	`i` (insert) `a` (append)
Visual	`v` (character) `V` (line)
Search	`/pattern` `n` (next) `N` (previous)
Edit	`x` (delete char) `dd` (delete line)
Copy	`yy` (copy line) `p` (paste)
Save/Quit	`:w` (save) `:q` (quit) `:wq` (save & quit)

10 LAB 9: System Monitoring

Monitor CPU, memory, disk, and I/O - the vital signs of your servers.

Listing 35: Disk Usage

```
# 1. Check disk usage
df -h

# 2. Check specific mount points
df -h / /home

# 3. Check inode usage
df -i /
```

Listing 36: Memory Monitoring

```
# 4. Check memory usage
free -h

# 5. Detailed memory info
cat /proc/meminfo | grep -E "(MemTotal|MemFree|MemAvailable)"

# 6. Check swap usage
swapon --show
```

Listing 37: Process Monitoring

```
# 7. Interactive process viewer
top
# Press 'q' to quit

# 8. Install and use htop (better alternative)
sudo apt install htop -y
htop
# Press 'q' or F10 to quit
```

Listing 38: Directory Size Analysis

```
# 9. Check home directory usage
du -sh ~/*

# 10. Find largest directories
du -h /var | sort -rh | head -10

# 11. Find large files
sudo find /var/log -type f -size +100M -exec ls -lh {} \;
```

Listing 39: System Load

```
# 12. Check system uptime and load
uptime

# 13. System stats summary
vmstat 1 5
```

```

# 14. CPU and I/O statistics
iostat -x 1 3
```

10.1 Expected Results

- `df -h` shows disk usage percentages
- `free -h` shows available memory
- `top/htop` shows running processes sorted by CPU/memory
- `uptime` shows system load averages

Monitoring Pro Tip

Set up alerts for:

- Disk > 85
- Memory < 10
- Load average $>$ CPU cores $\times$ 1.5

11 Bonus Challenge: Deploy Static Website

Deploy a complete static website with Nginx, custom configuration, and monitoring. (45-60 minutes)

11.1 Phase 1: Infrastructure Setup

Listing 40: Reinstall & Configure Nginx

```
# 1. Fresh Nginx installation
sudo apt update
sudo apt install nginx -y

# 2. Create website directory structure
sudo mkdir -p /var/www/mysite/{html,logs,backup}

# 3. Create realistic website content
cat > /tmp/index.html << 'EOF'
<!DOCTYPE html>
<html>
<head>
    <title>DevOps Demo Site</title>
    <style>
        body { font-family: Arial; margin: 40px; background: #1e1e1e; color: #fff; }
        .container { max-width: 800px; margin: 0 auto; }
        .status { background: #2d2d2d; padding: 20px; border-radius: 5px; }
    </style>
</head>
<body>
    <div class="container">
        <h1>      DevOps Demo Site</h1>
        <div class="status">
            <p><strong>Server:</strong> $(hostname)</p>
            <p><strong>Deployed by:</strong> $(whoami)</p>
            <p><strong>Deployed at:</strong> $(date)</p>
            <p><strong>Status:</strong> <span style="color: #4CAF50">      Live</span></p>
        </div>
        <h2>Production Ready Features:</h2>
        <ul>
            <li>Automated deployment pipeline</li>
            <li>Log monitoring and alerting</li>
            <li>SSL certificate (coming soon)</li>
            <li>CDN integration ready</li>
        </ul>
    </div>
</body>
</html>
EOF

sudo cp /tmp/index.html /var/www/mysite/html/
sudo chown -R www-data:www-data /var/www/mysite/
```

11.2 Phase 2: Nginx Configuration

Listing 41: Custom Nginx Site Config

```
# 4. Create custom Nginx configuration
sudo tee /etc/nginx/sites-available/mysite << 'EOF'
server {
    listen 80;
    server_name _;
    root /var/www/mysite/html;
    index index.html;

    # Security headers
    add_header X-Frame-Options "SAMEORIGIN" always;
    add_header X-XSS-Protection "1; mode=block" always;
    add_header X-Content-Type-Options "nosniff" always;

    # Logging
    access_log /var/www/mysite/logs/access.log;
    error_log /var/www/mysite/logs/error.log;

    location / {
        try_files $uri $uri/ =404;
        # Cache static files
        expires 1y;
        add_header Cache-Control "public, immutable";
    }

    location /health {
        access_log off;
        return 200 "healthy\n";
        add_header Content-Type text/plain;
    }
}
EOF

# 5. Enable the site
sudo ln -s /etc/nginx/sites-available/mysite /etc/nginx/sites-enabled/
sudo rm /etc/nginx/sites-enabled/default

# 6. Test configuration
sudo nginx -t
```

Listing 42: Start & Test Site

```
# 7. Start Nginx with new config
sudo systemctl restart nginx
sudo systemctl enable nginx

# 8. Test the website
curl localhost
curl localhost/health

# 9. Test from browser (if you have GUI)
# http://localhost should show your custom page
```

11.3 Phase 3: Monitoring & Automation

Listing 43: Create Health Check Script

```bash
# 10. Create website health monitoring
cat > ~/monitor-site.sh << 'EOF'
#!/bin/bash

URL="http://localhost"
LOG="/var/www/mysite/logs/health.log"

check_health() {
    if curl -f -s "$URL/health" > /dev/null; then
        echo "$(date):     Site healthy - $(curl -s "$URL/health")" >>
            "$LOG"
        return 0
    else
        echo "$(date):     Site DOWN - Check Nginx logs" >> "$LOG"
        # Restart nginx if down
        sudo systemctl restart nginx
        return 1
    fi
}

# Run health check
check_health
```

Listing 44: Setup Monitoring

```bash
chmod +x ~/monitor-site.sh

# 11. Test monitoring
~/monitor-site.sh

# 12. Check health log
cat /var/www/mysite/logs/health.log

# 13. Simulate failure (stop nginx)
sudo systemctl stop nginx
sleep 2
~/monitor-site.sh  # Should auto-restart

# 14. Verify recovery
sudo systemctl status nginx
curl localhost/health
```

Listing 45: Production Cleanup

```bash
# 15. Complete cleanup
sudo systemctl stop nginx
sudo rm -rf /var/www/mysite
sudo rm /etc/nginx/sites-available/mysite
sudo rm /etc/nginx/sites-enabled/mysite
sudo apt purge nginx -y
sudo apt autoremove -y
rm ~/monitor-site.sh
```

11.4 Challenge Complete!

Congratulations!

You've successfully:

- Deployed a production-ready static website
- Configured Nginx with security headers and logging
- Implemented automated health monitoring
- Created proper file permissions and ownership
- Written a recovery script for self-healing

This is exactly what junior DevOps engineers do on day one!

12 Cheat Sheets

12.1 Command Reference

Essential Commands

2 **File Navigation**

- `pwd` - Show current directory
- `ls -la` - List all files
- `cd` - Go home
- `cd -` - Previous directory

File Operations

- `touch file.txt` - Create empty file
- `mkdir -p dir/subdir` - Create directories
- `cp -r src dest` - Copy recursively
- `mv file newname` - Move/rename
- `rm -rf dir` - Delete (careful!)

Text Processing

- `cat file` - Show file content
- `grep pattern file` - Search in file
- `head -20 file` - First 20 lines
- `tail -f file` - Follow log

System Management

- `sudo apt update` - Update packages
- `systemctl status service` - Service status
- `ps aux | grep process` - Find processes
- `top` / `htop` - Monitor system

Networking

- `ip a` - Show interfaces
- `ping -c 4 host` - Test connectivity
- `curl -I url` - HTTP HEAD request
- `ss -tuln` - Listening ports

12.2 Troubleshooting Flowchart

Troubleshooting Guide

1. **Service won't start?** → `systemctl status service`
2. **Permission denied?** → `ls -la file` then `sudo chown`
3. **Can't find file?** → `find / -name "filename" 2>/dev/null`
4. **High CPU usage?** → `top` then `kill PID`
5. **Out of disk space?** → `df -h` then `du -sh * | sort -h`
6. **Network issues?** → `ping 8.8.8.8` then `curl google.com`
7. **Command not found?** → `which command` or `apt search command`

13 Next Steps

Level Up Your Linux Skills

Recommended Reading:

- Linux Journey - Interactive tutorials
- DigitalOcean Tutorials - Practical guides
- `man` pages - Your always-available reference

You've completed Linux for DevOps Fundamentals!

Now go deploy something amazing.