Lab 1: Introduction to DevOps, OS & Linux Fundamentals

Lab Overview

This lab introduces you to the foundational concepts of DevOps, operating systems, and Linux. You will learn what DevOps is, why it matters in modern software development, and how Linux serves as the backbone of most DevOps environments. By the end of this lab, you will be comfortable navigating the Linux filesystem and executing basic terminal commands.

Topics Covered:

- Overview of DevOps and its role in SDLC
- Introduction to Operating Systems
- Linux Filesystem Structure
- Basic Terminal Commands and Navigation

Learning Objectives

By the end of this lab, you will be able to:

1. Explain what DevOps is and its importance in software development
2. Understand the role of operating systems in computing
3. Navigate the Linux filesystem using command-line interface
4. Execute basic Linux commands for file and directory operations
5. Use essential terminal utilities for system exploration

Part 1: Introduction to DevOps

1.1 What is DevOps?

DevOps is a culture, set of practices, and collection of tools that aims to bridge the gap between software development (Dev) and IT operations (Ops). Traditionally, developers would write code and "throw it over the wall" to operations teams who would then deploy and maintain it. This created delays, miscommunication, and reliability issues.

DevOps combines development and operations into a collaborative workflow where:

- Developers understand deployment and infrastructure
- Operations teams are involved early in the development process
- Automation reduces manual errors and speeds up delivery
- Continuous feedback improves quality

Key Benefits of DevOps:

- Faster time to market for new features
- Improved collaboration between teams
- Higher quality and more reliable software
- Automated and repeatable processes
- Rapid recovery from failures

1.2 DevOps and the Software Development Life Cycle (SDLC)

The traditional SDLC follows these phases: Planning, Analysis, Design, Implementation, Testing, Deployment, and Maintenance. In traditional models like Waterfall, these phases happen sequentially, often taking months or years.

DevOps transforms the SDLC by introducing:

Continuous Integration (CI): Developers frequently merge code changes into a shared repository. Automated tests run on every change to catch bugs early.

Continuous Delivery (CD): Code is always in a deployable state. Releases can happen at any time with minimal manual intervention.

Continuous Deployment: Every code change that passes automated tests is automatically deployed to production.

Infrastructure as Code (IaC): Infrastructure is managed using code and configuration files, making it version-controlled and reproducible.

Monitoring and Feedback: Continuous monitoring of applications provides feedback to developers, closing the loop.

This creates a cycle: Plan → Code → Build → Test → Release → Deploy → Operate → Monitor → Plan (and repeat).

1.3 DevOps Tools Landscape

Throughout this course, you will work with various DevOps tools:

- **Version Control:** Git, GitHub
- **Containerization:** Docker
- **CI/CD:** GitHub Actions
- **Cloud Infrastructure:** AWS EC2
- **Infrastructure as Code:** Terraform
- **Operating System:** Linux

Each tool serves a specific purpose in the DevOps pipeline, and you will gain hands-on experience with all of them.

Part 2: Introduction to Operating Systems

2.1 What is an Operating System?

An Operating System (OS) is system software that manages computer hardware and software resources and provides common services for computer programs. It acts as an intermediary between users/applications and the computer hardware.

Key Functions of an OS:

- **Process Management:** Creating, scheduling, and terminating processes
- **Memory Management:** Allocating and deallocating memory to programs
- **File System Management:** Organizing and storing files on storage devices
- **Device Management:** Managing input/output devices
- **Security and Access Control:** Protecting resources from unauthorized access
- **User Interface:** Providing a way for users to interact with the computer

2.2 Types of Operating Systems

Single-User, Single-Tasking: Can run one program at a time (e.g., MS-DOS)

Single-User, Multi-Tasking: One user can run multiple programs simultaneously (e.g., Windows, macOS)

Multi-User: Multiple users can use the system simultaneously (e.g., Linux, Unix)

Real-Time OS: Designed for time-critical applications (e.g., embedded systems)

Distributed OS: Manages a group of independent computers and makes them appear as a single system

2.3 Why Linux for DevOps?

Linux is the dominant operating system in DevOps and server environments for several reasons:

Open Source: Free to use, modify, and distribute. This reduces costs and provides transparency.

Stability and Reliability: Linux servers can run for years without rebooting. Critical for production systems.

Security: Strong permission model and active security community. Regular updates and patches.

Performance: Efficient resource usage, especially important for servers running multiple services.

Community and Support: Massive global community provides extensive documentation and support.

Flexibility: Highly customizable for specific needs. Can run on anything from tiny embedded devices to massive supercomputers.

Industry Standard: Most cloud servers, containers, and DevOps tools run on Linux. Learning Linux is essential for DevOps professionals.

Part 3: Linux Filesystem Structure

3.1 Understanding the Linux Filesystem Hierarchy

Unlike Windows which uses drive letters (C:, D:, etc.), Linux uses a single unified directory tree starting from the root directory denoted by a forward slash `/`. Everything in Linux is considered a file, including directories, devices, and processes.

Key Directories in Linux:

/ (Root): The top-level directory. All other directories branch from here.

/home: Contains home directories for all users. Each user has a subdirectory here (e.g., /home/username). This is where users store their personal files and configurations.

/root: The home directory for the root (administrator) user. Separate from /home.

/bin: Contains essential user command binaries (programs) that are needed in single-user mode and to bring the system up or repair it (e.g., ls, cp, cat).

/sbin: Contains system binaries, typically used by the system administrator (e.g., fdisk, ifconfig).

/usr: Contains user programs and data. Has its own bin, sbin, and lib subdirectories with additional programs and libraries.

/etc: Contains system-wide configuration files. Almost every program has configuration files here.

/var: Contains variable data like logs, databases, email, and print queues. Data that changes frequently during system operation.

/tmp: Temporary files created by programs. Usually cleared on reboot.

/dev: Contains device files representing hardware devices and virtual devices.

/lib: Contains shared library files needed by programs in /bin and /sbin.

/opt: Contains optional software packages, often third-party applications.

/mnt and /media: Mount points for temporary filesystems and removable media (USB drives, CDs).

3.2 Absolute vs Relative Paths

Absolute Path: Specifies the complete path from the root directory. Always starts with `/`.
Example: `/home/student/documents/file.txt`

Relative Path: Specifies the path relative to the current directory. Does not start with `/`.
Example: If you are in `/home/student`, the relative path to the same file would be `documents/file.txt`

Special Directory Symbols:

- `.` (single dot) represents the current directory
- `..` (double dot) represents the parent directory
- `~` (tilde) represents the current user's home directory
- `/` (forward slash) is the path separator and root directory

Part 4: Setting Up Your Linux Environment

4.1 Accessing Linux

For this lab, you can use one of the following methods:

Option 1: University Lab Computers - If Linux is already installed, simply boot into Linux or access it through dual-boot.

Option 2: Virtual Machine - Install VirtualBox or VMware and create an Ubuntu virtual machine.

Option 3: WSL (Windows Subsystem for Linux) - If you are using Windows 10/11, install WSL2 with Ubuntu.

Option 4: Live USB - Boot Ubuntu from a USB drive without installation.

Your instructor will guide you on the recommended method for this course.

4.2 Opening the Terminal

The terminal (also called shell or command line) is your primary interface for working with Linux in DevOps.

To open the terminal in Ubuntu:

- Press `Ctrl + Alt + T`
- Or search for "Terminal" in the applications menu
- Or right-click on the desktop and select "Open Terminal"

When you open the terminal, you will see a prompt that looks something like:

```
student@ubuntu:~$
```

This prompt shows:

- `student` - your username
- `ubuntu` - the hostname (computer name)
- `~` - your current directory (~ means home directory)
- `$` - indicates you are a regular user (# indicates root user)

Part 5: Basic Linux Commands

5.1 Navigation Commands

pwd (Print Working Directory)

Shows the absolute path of your current directory.

```
pwd
```

Example output: `/home/student`

ls (List)

Lists files and directories in the current directory.

```
ls
```

Common options:

- `ls -l` : Long format showing permissions, owner, size, and modification date
- `ls -a` : Show all files including hidden files (files starting with .)
- `ls -lh` : Long format with human-readable file sizes
- `ls -la` : Combine options to show all files in long format

Example:

```
ls -la
```

cd (Change Directory)

Changes your current directory.

```
cd /path/to/directory
```

Examples:

```
cd /home              # Go to /home directory
cd Documents          # Go to Documents in current directory
cd ..                 # Go up one level to parent directory
cd ~                  # Go to your home directory
cd -                  # Go to previous directory
cd /                  # Go to root directory
```

5.2 File and Directory Operations

mkdir (Make Directory)

Creates a new directory.

```
mkdir directory_name
```

Create nested directories:

```
mkdir -p parent/child/grandchild
```

The `-p` flag creates parent directories if they do not exist.

touch

Creates an empty file or updates the timestamp of an existing file.

```
touch filename.txt
```

Create multiple files:

```
touch file1.txt file2.txt file3.txt
```

cp (Copy)

Copies files or directories.

```
cp source destination
```

Examples:

```
cp file.txt backup.txt              # Copy file
cp file.txt /home/student/backup/   # Copy to another directory
cp -r folder/ backup_folder/        # Copy directory recursively
```

The `-r` flag is required when copying directories.

mv (Move)

Moves or renames files and directories.

```
mv source destination
```

Examples:

```
mv old_name.txt new_name.txt        # Rename file
mv file.txt /home/student/docs/     # Move file
mv folder/ /home/student/backup/    # Move directory
```

rm (Remove)

Deletes files or directories permanently. Be careful - there is no recycle bin!

```
rm filename
```

Examples:

```
rm file.txt                         # Delete a file
rm -r folder/                       # Delete directory recursively
rm -f file.txt                      # Force delete without confirmation
rm -rf folder/                      # Force delete directory (use with extreme caution!)
```

Important: Never run `rm -rf /` or `rm -rf /*` as this will delete your entire system!

rmdir (Remove Directory)

Removes empty directories only.

```
rmdir empty_directory
```

5.3 Viewing File Contents

cat (Concatenate)

Displays entire file contents.

```
cat filename
```

Concatenate multiple files:

```
cat file1.txt file2.txt
```

less

Views file content one page at a time. Better for large files.

```
less filename
```

Navigation in less:

- Space: Next page
- b: Previous page
- /search_term: Search forward
- q: Quit

head

Shows the first 10 lines of a file.

```
head filename
```

Show first n lines:

```
head -n 20 filename
```

tail

Shows the last 10 lines of a file.

```
tail filename
```

Show last n lines:

```
tail -n 20 filename
```

Follow a file in real-time (useful for log files):

```
tail -f logfile.txt
```

wc (Word Count)

Counts lines, words, and characters in a file.

```
wc filename
```

Count only lines:

```
wc -l filename
```

5.4 Getting Help

man (Manual)

Displays the manual page for a command.

```
man command_name
```

Example:

```
man ls
```

Navigate using arrow keys, search with `/`, and quit with `q`.

--help

Most commands support a --help flag for quick reference.

```
ls --help
```

whatis

Gives a brief description of a command.

```
whatis ls
```

Part 6: Practical Exercises

Now that you have learned the basic commands, complete the following exercises to reinforce your understanding.

Exercise 1: Exploring the Filesystem

1. Open the terminal
2. Use `pwd` to see your current location
3. Use `ls` to see what files and directories are in your home directory
4. Navigate to the root directory using `cd /`
5. List the contents of the root directory using `ls`
6. Navigate to the /etc directory using `cd /etc`
7. List the contents using `ls -l`
8. Return to your home directory using `cd ~`
9. Verify you are home using `pwd`

Exercise 2: Creating a Directory Structure

1. In your home directory, create a directory called `devops_lab`
2. Navigate into the `devops_lab` directory
3. Inside `devops_lab`, create three directories: `week1`, `week2`, `week3`
4. List the contents to verify the directories were created
5. Navigate into `week1`
6. Create a subdirectory called `notes`
7. Use `pwd` to confirm your current location
8. Go back to the `devops_lab` directory using `cd ..`

Exercise 3: Working with Files

1. Navigate to `devops_lab/week1`
2. Create a file called `intro.txt` using touch
3. Create three more files: `commands.txt`, `filesystem.txt`, `practice.txt`
4. List all files to verify creation
5. Copy `intro.txt` to `intro_backup.txt`
6. Rename `practice.txt` to `exercises.txt` using mv
7. List the files again to see the changes
8. Delete `intro_backup.txt`
9. Verify the file was deleted by listing files again

Exercise 4: File Content Operations

1. Create a new file called `my_notes.txt`
2. Open it with a text editor (nano or vim) and write: "DevOps is a set of practices that combines software development and IT operations."
3. Save and close the file
4. Display the contents using `cat my_notes.txt`
5. Add more content to the file about what you learned today
6. Use `cat my_notes.txt` again to verify
7. Use `wc -l my_notes.txt` to count how many lines you have written
8. Use `head -n 3 my_notes.txt` to display the first 3 lines

Exercise 5: Advanced Navigation

1. From anywhere, navigate directly to `/usr/bin` using an absolute path
2. List the files (you will see many command programs here)
3. Navigate to `/var/log` using an absolute path
4. List the log files here
5. Navigate back to your home directory using `cd ~`
6. Create a directory structure: `projects/web/frontend` using a single mkdir command
7. Navigate directly into the frontend directory
8. Use `pwd` to confirm your location
9. Go back to your home directory

Part 7: Understanding Command Structure

7.1 Anatomy of a Linux Command

A typical Linux command follows this structure:

```
command [options] [arguments]
```

Command: The program you want to run (e.g., ls, cp, mkdir)

Options: Modify the behavior of the command. Usually start with - (single dash) for short options or -- (double dash) for long options.

Arguments: What the command operates on (e.g., filenames, directories)

Example:

```
ls -la /home/student
```

- `ls` is the command
- `-la` are options (combined -l and -a)
- `/home/student` is the argument

7.2 Wildcards and Pattern Matching

Wildcards help you work with multiple files at once.

*** (Asterisk):** Matches any number of characters

```
ls *.txt              # List all .txt files
rm file*              # Remove all files starting with "file"
```

? (Question Mark): Matches exactly one character

```
ls file?.txt          # Matches file1.txt, fileA.txt, but not file10.txt
```

[] (Brackets): Matches any one character inside the brackets

```
ls file[123].txt      # Matches file1.txt, file2.txt, file3.txt
ls file[a-z].txt      # Matches any file followed by one lowercase letter
```

7.3 Command Chaining

You can run multiple commands in sequence:

Semicolon (;): Run commands one after another, regardless of success

```
cd Documents; ls; pwd
```

AND (&&): Run the next command only if the previous one succeeds

```
mkdir test && cd test
```

OR (||): Run the next command only if the previous one fails

```
cd nonexistent || echo "Directory not found"
```

Part 8: Common Mistakes and Troubleshooting

8.1 Case Sensitivity

Linux is case-sensitive. `File.txt`, `file.txt`, and `FILE.TXT` are three different files.

```
ls File.txt        # Different from
ls file.txt
```

8.2 Spaces in Filenames

Spaces in filenames require special handling. Use quotes or escape the space with a backslash.

```
touch "my file.txt"    # Using quotes
touch my\ file.txt     # Using backslash
```

Better practice: Avoid spaces in filenames. Use underscores or hyphens instead.

```
touch my_file.txt      # Recommended
touch my-file.txt      # Also recommended
```

8.3 Hidden Files

Files starting with a dot (.) are hidden and will not show with `ls` unless you use the `-a` option.

```
ls                # Won't show .hidden
ls -a             # Will show .hidden
```

8.4 Permission Denied Errors

If you see "Permission denied", you might be trying to access files or directories you do not have permission to read or modify. You will learn about permissions in the next lab.

8.5 Command Not Found

If you see "command not found", check for typos. Remember Linux is case-sensitive. The command might also not be installed on your system.

Lab Assessment Tasks

Complete the following tasks and submit screenshots or command outputs as directed by your instructor.

Task 1: Create the following directory structure in your home directory:

```
devops_course/
├── labs/
│   ├── lab1/
│   ├── lab2/
│   └── lab3/
└── assignments/
    ├── assignment1/
    └── assignment2/
```

Task 2: Navigate to `devops_course/labs/lab1` and create five files: `intro.txt`, `commands.txt`, `filesystem.txt`, `notes.txt`, and `summary.txt`

Task 3: Write a brief summary (3-4 lines) about what DevOps is in the `summary.txt` file using a text editor

Task 4: Copy all .txt files from lab1 to lab2

Task 5: Rename `summary.txt` in lab2 to `devops_summary.txt`

Task 6: Delete the `lab3` directory

Task 7: Create a file called `evidence.txt` in your home directory that contains the output of the following commands:

- `pwd`
- `ls -la devops_course`
- `cat devops_course/labs/lab1/summary.txt`