Lab 2: Linux Files, Permissions & Process Management

Lab Overview

This lab builds upon your foundational Linux knowledge by introducing file permissions, user and group management, and process monitoring. Understanding these concepts is critical for managing servers, controlling access to resources, and troubleshooting application issues in DevOps environments.

Topics Covered:

- File operations and manipulation
- File permissions and ownership
- User and group management
- Process monitoring and management
- System resource management

Learning Objectives

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

1. Perform advanced file operations including searching and finding files
2. Understand and modify file permissions using chmod
3. Manage file ownership with chown and chgrp
4. Create and manage users and groups
5. Monitor running processes and system resources
6. Manage processes using kill, killall, and background jobs

Part 1: Advanced File Operations

1.1 Finding Files and Directories

find Command

The find command searches for files and directories in a directory hierarchy based on various criteria.

Basic syntax:

```
find [path] [options] [expression]
```

Examples:

```
find . -name "file.txt"          # Find file by exact name in current directory
```

```
find /home -name "*.txt"            # Find all .txt files in /home
find . -type f -name "*.log"        # Find only files (not directories)
find . -type d -name "temp"         # Find only directories
find . -size +10M                   # Find files larger than 10MB
find . -mtime -7                    # Find files modified in last 7 days
find . -user student                # Find files owned by user "student"
```

locate Command

Faster than find but uses a database that needs to be updated.

```
locate filename                     # Quick search for filename
sudo updatedb                       # Update the locate database
```

which Command

Shows the full path of shell commands.

```
which python                        # Shows where python is installed
which ls                            # Shows path to ls command
```

1.2 Searching Inside Files

grep (Global Regular Expression Print)

Searches for patterns within files.

Basic syntax:

```
grep [options] pattern [files]
```

Examples:

```
grep "error" logfile.txt            # Search for "error" in file
grep -i "error" logfile.txt         # Case-insensitive search
grep -r "function" .                # Recursive search in all files
grep -n "TODO" script.sh            # Show line numbers
grep -v "error" logfile.txt         # Show lines that DON'T contain "error"
grep -c "warning" logfile.txt       # Count matching lines
```

Combining grep with other commands:

```
ls -l | grep "txt"                  # List only .txt files
cat file.txt | grep "important"     # Search in output
```

1.3 File Comparison

diff Command

Compares two files line by line.

```
diff file1.txt file2.txt          # Show differences
diff -u file1.txt file2.txt       # Unified format (more readable)
diff -r dir1/ dir2/               # Compare directories recursively
```

1.4 File Compression and Archiving

tar (Tape Archive)

Creates and extracts archive files.

Creating archives:

```
tar -cvf archive.tar files/         # Create tar archive
tar -czvf archive.tar.gz files/     # Create compressed (gzip) archive
tar -cjvf archive.tar.bz2 files/    # Create compressed (bzip2) archive
```

Extracting archives:

```
tar -xvf archive.tar              # Extract tar archive
tar -xzvf archive.tar.gz          # Extract gzip archive
tar -xjvf archive.tar.bz2         # Extract bzip2 archive
```

Common options:

- c: create
- x: extract
- v: verbose (show progress)
- f: file (specify filename)
- z: gzip compression
- j: bzip2 compression

gzip and gunzip

Compress individual files:

```
gzip file.txt                 # Compresses to file.txt.gz
gunzip file.txt.gz            # Decompresses to file.txt
```

zip and unzip

```
zip archive.zip file1.txt file2.txt  # Create zip archive
zip -r archive.zip folder/           # Zip directory recursively
unzip archive.zip                    # Extract zip archive
unzip -l archive.zip                 # List contents without extracting
```

Part 2: Understanding File Permissions

2.1 Linux Permission Model

Every file and directory in Linux has three types of permissions for three categories of users.

Permission Types:

- **r (read)**: Permission to read the file content or list directory contents
- **w (write)**: Permission to modify the file or create/delete files in a directory
- **x (execute)**: Permission to execute a file or enter a directory

User Categories:

- **u (user/owner)**: The user who owns the file
- **g (group)**: Users who are members of the file's group
- **o (others)**: All other users
- **a (all)**: All three categories combined

2.2 Viewing Permissions

Use `ls -l` to view detailed file information including permissions:

```
ls -l file.txt
```

Output example:

```
-rw-r--r-- 1 student student 1234 Jan 20 10:30 file.txt
```

Let's break this down:

```
-rw-r--r--
│ │ │ │ │ │ │ │ │ └─ Others: read
│ │ │ │ │ │ │ │ └── Others: no write
│ │ │ │ │ │ │ └──── Others: no execute
│ │ │ │ │ │ └────── Group: read
│ │ │ │ │ └──────── Group: no write
│ │ │ │ └────────── Group: no execute
│ │ │ └──────────── Owner: read
```

```
│ │ └──────────── Owner: write
│ └────────────── Owner: no execute
└──────────────── File type (- for file, d for directory, l for link)
```

The number following permissions (1) is the number of hard links. Then comes the owner (student), group (student), file size (1234 bytes), date, and filename.

2.3 Numeric (Octal) Representation

Permissions can also be represented numerically:

- r (read) = 4
- w (write) = 2
- x (execute) = 1

Add the values to get the permission for each category:

- 0 = no permissions (---)
- 1 = execute only (--x)
- 2 = write only (-w-)
- 3 = write and execute (-wx)
- 4 = read only (r--)
- 5 = read and execute (r-x)
- 6 = read and write (rw-)
- 7 = read, write, and execute (rwx)

Example: 755 means:

- Owner: 7 (rwx) - read, write, execute
- Group: 5 (r-x) - read, execute
- Others: 5 (r-x) - read, execute

Common permission sets:

- 644: rw-r--r-- (standard file)
- 755: rwxr-xr-x (executable file)
- 700: rwx------ (private file)
- 777: rwxrwxrwx (everyone can do everything - generally not recommended)

2.4 Changing Permissions with chmod

Symbolic Mode:

```
chmod u+x file.sh          # Add execute for owner
chmod g-w file.txt         # Remove write for group
chmod o+r file.txt         # Add read for others
```

```
chmod a+x script.sh              # Add execute for all
chmod u+rw,g+r,o-rwx file.txt    # Multiple changes at once
```

Numeric Mode:

```
chmod 644 file.txt               # rw-r--r--
chmod 755 script.sh              # rwxr-xr-x
chmod 700 private.txt            # rwx------
chmod 777 shared.txt             # rwxrwxrwx (be careful!)
```

Recursive Changes:

```
chmod -R 755 directory/          # Apply to directory and all contents
```

2.5 Changing Ownership

chown (Change Owner)

Changes the owner of a file or directory. Requires root privileges.

```
sudo chown newowner file.txt         # Change owner
sudo chown newowner:newgroup file.txt # Change owner and group
sudo chown -R newowner directory/    # Recursive change
```

chgrp (Change Group)

Changes the group ownership.

```
sudo chgrp newgroup file.txt         # Change group
sudo chgrp -R newgroup directory/    # Recursive change
```

2.6 Special Permissions

Setuid (Set User ID):

When set on an executable, it runs with the permissions of the file owner, not the user running it.

```
chmod u+s executable             # Symbolic
chmod 4755 executable            # Numeric (4 prefix)
```

Setgid (Set Group ID):

When set on a directory, files created within inherit the directory's group.

```
chmod g+s directory              # Symbolic
```

```
chmod 2755 directory          # Numeric (2 prefix)
```

Sticky Bit:

When set on a directory, only the file owner can delete their files (useful for /tmp).

```
chmod +t directory            # Symbolic
chmod 1755 directory          # Numeric (1 prefix)
```

Part 3: User and Group Management

3.1 Understanding Users and Groups

In Linux, every user has:

- A unique username
- A unique User ID (UID)
- A primary group
- Optionally, membership in multiple secondary groups
- A home directory
- A login shell

Users are defined in `/etc/passwd` Groups are defined in `/etc/group` Encrypted passwords are stored in `/etc/shadow` (root only)

3.2 User Information Commands

whoami

Shows your current username.

```
whoami
```

id

Shows user ID, group ID, and group memberships.

```
id                    # Your information
id username           # Another user's information
```

who

Shows who is currently logged in.

```
who
```

w

Shows who is logged in and what they are doing.

```
w
```

users

Lists currently logged-in users.

```
users
```

3.3 Creating and Managing Users

adduser (High-level command)

Interactive way to create users.

```
sudo adduser newuser                # Creates user with home directory
```

This command prompts for password and user information.

useradd (Low-level command)

More control but requires manual setup.

```
sudo useradd -m -s /bin/bash newuser # -m creates home, -s sets shell
sudo passwd newuser                 # Set password
```

Modifying Users:

```
sudo usermod -c "Full Name" username # Change full name
sudo usermod -d /new/home username   # Change home directory
sudo usermod -s /bin/zsh username    # Change shell
sudo usermod -l newname oldname      # Rename user
```

Deleting Users:

```
sudo userdel username               # Delete user (keeps home directory)
sudo userdel -r username            # Delete user and home directory
```

3.4 Creating and Managing Groups

groupadd

Creates a new group.

```
sudo groupadd developers            # Create group
sudo groupadd -g 1500 testers       # Create with specific GID
```

Adding Users to Groups:

```
sudo usermod -aG groupname username  # Add user to group (-a appends)
sudo gpasswd -a username groupname   # Alternative method
```

Removing Users from Groups:

```
sudo gpasswd -d username groupname   # Remove user from group
```

Viewing Group Members:

```
getent group groupname              # Show group members
groups username                     # Show user's groups
```

Deleting Groups:

```
sudo groupdel groupname             # Delete group
```

3.5 Switching Users

su (Switch User)

Switches to another user account.

```
su username                     # Switch user (keeps current environment)
su - username                   # Switch user (loads user's environment)
su -                            # Switch to root
exit                            # Return to previous user
```

sudo (Superuser Do)

Executes a single command with root privileges.

```
sudo command                    # Run command as root
sudo -u username command        # Run command as another user
sudo -i                         # Start root shell
```

Part 4: Process Management

4.1 Understanding Processes

A process is a running instance of a program. Every process has:

- A unique Process ID (PID)
- A Parent Process ID (PPID)
- Owner (user who started it)
- Priority level
- Resource usage (CPU, memory)
- State (running, sleeping, stopped, zombie)

4.2 Viewing Processes

ps (Process Status)

Shows currently running processes.

```
ps                    # Your processes
ps aux                # All processes, detailed
ps -ef                # All processes, full format
ps -u username        # Processes by user
ps -p PID             # Specific process
```

Common columns:

- PID: Process ID
- PPID: Parent Process ID
- USER: Process owner
- %CPU: CPU usage percentage
- %MEM: Memory usage percentage
- STAT: Process state
- COMMAND: Command that started process

top

Real-time, dynamic view of processes.

```
top
```

Useful keys while in top:

- q: Quit
- k: Kill a process (prompts for PID)
- M: Sort by memory usage
- P: Sort by CPU usage
- h: Help
- 1: Show individual CPU cores

htop

Enhanced version of top (may need installation).

```
sudo apt install htop          # Install on Ubuntu
htop                           # Run htop
```

pgrep

Search for processes by name.

```
pgrep firefox                  # Find PIDs of firefox processes
pgrep -u username              # Processes by user
pgrep -l firefox               # Show PID and name
```

4.3 Process States

- **R (Running)**: Currently executing
- **S (Sleeping)**: Waiting for an event
- **D (Uninterruptible Sleep)**: Usually waiting for I/O
- **T (Stopped)**: Suspended
- **Z (Zombie)**: Finished but not cleaned up by parent

4.4 Managing Processes

kill

Sends signals to processes to control them.

```
kill PID                       # Send TERM signal (graceful shutdown)
kill -9 PID                    # Send KILL signal (force kill)
kill -STOP PID                 # Pause process
kill -CONT PID                 # Resume process
```

Common signals:

- 1 (HUP): Hangup, reload configuration
- 9 (KILL): Force kill (cannot be ignored)
- 15 (TERM): Graceful termination (default)
- 19 (STOP): Pause process
- 18 (CONT): Continue process

killall

Kills processes by name.

```
killall firefox                 # Kill all firefox processes
killall -9 unresponsive_app     # Force kill by name
```

pkill

Similar to killall but more flexible.

```
pkill firefox                   # Kill processes by name
pkill -u username               # Kill all processes by user
```

4.5 Background and Foreground Jobs

Running Commands in Background:

```
command &                       # Run in background
```

Job Control:

```
jobs                        # List background jobs
fg %1                       # Bring job 1 to foreground
bg %1                       # Resume job 1 in background
Ctrl+Z                      # Suspend current foreground job
```

nohup (No Hangup)

Runs command immune to hangups (continues after logout).

```
nohup command &                 # Run in background, ignore hangup
nohup python script.py > output.log & # Redirect output to file
```

disown

Removes job from shell's job table (makes it independent).

```
command &
disown                      # Current job won't receive hangup signal
```

4.6 Process Priority

nice

Starts a process with modified scheduling priority (-20 highest, 19 lowest).

```
nice -n 10 command              # Start with lower priority
nice -n -10 command             # Start with higher priority (needs sudo)
```

renice

Changes priority of running process.

```
renice -n 5 -p PID              # Change priority of PID
sudo renice -n -5 -p PID        # Increase priority (needs sudo)
```

Part 5: System Resource Management

5.1 Disk Usage

df (Disk Free)

Shows filesystem disk space usage.

```
df                      # Show disk usage
df -h                   # Human-readable format
df -T                   # Show filesystem type
df /home                # Show specific filesystem
```

du (Disk Usage)

Shows directory space usage.

```
du                      # Current directory
du -h                   # Human-readable
du -sh                  # Summary only
du -sh *                # Size of each item
du -h --max-depth=1             # Limit subdirectory depth
```

5.2 Memory Usage

free

Shows memory usage.

```
free                    # Show memory in KB
free -h                 # Human-readable
free -m                 # Show in MB
free -g                 # Show in GB
```

Output shows:

- total: Total installed memory
- used: Used memory
- free: Completely unused memory
- shared: Memory used by tmpfs
- buff/cache: Memory used for buffers and cache
- available: Estimation of memory available for new applications

5.3 System Information

uname

Shows system information.

```
uname -a                # All information
uname -r                # Kernel version
uname -m                # Machine hardware name
```

uptime

Shows how long system has been running.

```
uptime
```

lscpu

Shows CPU architecture information.

```
lscpu
```

lsblk

Lists block devices (disks and partitions).

```
lsblk
lsblk -f                # Show filesystem type
```

dmesg

Shows kernel ring buffer messages (useful for hardware issues).

```
dmesg                   # Show all messages
dmesg | tail            # Show recent messages
dmesg | grep -i error   # Show errors
```

5.4 Monitoring System Logs

journalctl (systemd systems)

Views systemd logs.

```
journalctl                    # All logs
journalctl -f                 # Follow logs (like tail -f)
journalctl -u servicename     # Logs for specific service
journalctl --since today      # Today's logs
journalctl --since "2026-01-20"   # Logs from specific date
journalctl -p err             # Error level logs only
```

Traditional log files:

```
tail -f /var/log/syslog       # System log
tail -f /var/log/auth.log     # Authentication log
tail -f /var/log/kern.log     # Kernel log
```

Part 6: Practical Exercises

Exercise 1: File Permissions Practice

1. Create a directory called `permissions_lab` in your home directory
2. Navigate into it and create three files: `public.txt`, `private.txt`, `executable.sh`
3. View the default permissions using `ls -l`
4. Set permissions on `public.txt` to 644 (rw-r--r--)
5. Set permissions on `private.txt` to 600 (rw-------)
6. Set permissions on `executable.sh` to 755 (rwxr-xr-x)
7. Verify the changes with `ls -l`
8. Try to create a file that only you can read and write: `secret.txt` with permissions 600
9. Create a shell script that prints "Hello DevOps" and make it executable
10. Run your script: `./executable.sh`

Exercise 2: User and Group Management

Note: These commands require sudo privileges. Your instructor may provide a virtual machine or special account.

1. Create a new group called `devteam`
2. Create a new user called `developer1`
3. Add `developer1` to the `devteam` group
4. Create a directory called `team_project` with group ownership set to `devteam`
5. Set permissions on `team_project` so group members can read, write, and execute
6. Verify the group membership with `groups developer1`
7. Use `id developer1` to see all user information

Exercise 3: Process Management

1. Open the terminal and run `top` to see running processes
2. Press 'q' to quit top
3. Start a long-running process in the background: `sleep 300 &`
4. Use `jobs` to see your background jobs
5. Use `ps aux | grep sleep` to find the PID of your sleep command
6. Kill the sleep process using `kill PID` (replace PID with actual number)
7. Verify it's gone with `ps aux | grep sleep`
8. Start another sleep process: `sleep 200 &`
9. Use `killall sleep` to kill it
10. Try running: `htop` to see an enhanced process viewer (if available)

Exercise 4: Finding and Searching Files

1. Create a directory structure:

```
search_lab/
├── documents/
│   ├── report.txt
│   └── notes.txt
├── scripts/
│   ├── backup.sh
│   └── deploy.sh
└── logs/
    ├── error.log
    └── access.log
```

2. Add the word "ERROR" to error.log
3. Add the word "SUCCESS" to access.log
4. Use `find` to locate all .log files: `find search_lab -name "*.log"`
5. Use `find` to locate all .sh files
6. Use `grep` to find which log file contains "ERROR": `grep -r "ERROR" search_lab/logs/`
7. Use `grep` to count how many times "ERROR" appears: `grep -c "ERROR" search_lab/logs/error.log`

Exercise 5: System Monitoring

1. Check disk usage: `df -h`
2. Check memory usage: `free -h`
3. Check your home directory size: `du -sh ~`
4. Find the top 5 largest directories in your home: `du -h ~ | sort -rh | head -5`
5. Check system uptime: `uptime`
6. View CPU information: `lscpu`

7. List all block devices: `lsblk`
8. Check which processes are using the most CPU: `ps aux --sort=-%cpu | head -10`

Exercise 6: File Compression

1. Create a directory called `backup_test` with several files inside
2. Create a tar archive: `tar -cvf backup.tar backup_test/`
3. Create a compressed tar archive: `tar -czvf backup.tar.gz backup_test/`
4. Compare the sizes: `ls -lh backup.tar backup.tar.gz`
5. Delete the `backup_test` directory
6. Extract the tar.gz archive: `tar -xzvf backup.tar.gz`
7. Verify the files were restored
8. Create a zip archive: `zip -r backup.zip backup_test/`
9. List the contents without extracting: `unzip -l backup.zip`

Lab Assessment Tasks

Complete the following tasks and document your work:

Task 1: Create a shell script named `system_info.sh` that displays:

- Current user
- Current directory
- Disk usage of home directory
- Memory usage
- Top 5 processes by CPU usage

Make the script executable and run it.

Task 2: Create a directory structure for a web project:

```
webapp/
├── public/
├── src/
├── config/
└── logs/
```

Set the following permissions:

- `public/`: 755
- `src/`: 750
- `config/`: 700
- `logs/`: 775

Task 3: Create three files:

- `readme.txt` with permissions 644
- `deploy.sh` with permissions 755
- `secrets.conf` with permissions 600

Write a command that lists only files with executable permissions.

Task 4: Start a long-running process (e.g., `sleep 500`) in the background. Find its PID using three different methods:

- Using `ps` with grep
- Using `pgrep`
- Using `jobs` and `ps`

Then kill the process.

Task 5: Find all `.txt` files in your home directory that were modified in the last 7 days. Save the output to a file called `recent_files.txt`.
Task 6: Create a compressed backup of your `devops_course` directory (from Lab 1) named `devops_backup_YYYYMMDD.tar.gz` (use today's date).