Linux Assignment 4: Comprehensive Lab Manual

A Complete Educational Guide to Output Redirection, Directory Operations, and File Management

Table of Contents

Introduction & Key Concepts

What You'll Learn

- **Output Redirection**: How to save command output to files instead of displaying on screen
- **Directory Operations**: Creating, copying, and navigating complex directory structures
- **File Management**: Moving, copying, and manipulating files
- **Path Navigation**: Understanding absolute vs relative paths
- **Find Command**: Searching for files and directories recursively

Important Reminders

⚠️ **CRITICAL**: Directory and file names must be **EXACTLY** as specified (case-sensitive) ⚠️ **No Typos Allowed**: Wrong names = lost points ⚠️ **Screenshots Required**: Document every step as requested

---

Task 1: Output Redirection Basics

Learning Objective

Master the concept of output redirection using the `>` operator to save command output to files.

What is Output Redirection?

Output redirection allows you to save the output of a command to a file instead of displaying it on the terminal. The `>` operator creates a new file or overwrites an existing one.

Step 1.1: Execute Basic Commands

First, let's run each command to see their normal output:

```bash
# Display current date and time
$ date
```

What it does: Shows the current system date and time **Why useful**: Timestamps are essential for logging and file management

```bash
# Show logged-in users
$ users
```

What it does: Lists all users currently logged into the system **Why useful**: System monitoring and security

```bash
# Show detailed user information
$ who
```

What it does: Shows who is logged in, their terminal, and login time **Why useful**: More detailed than `users`, includes session information

```bash
# Echo your student number (replace with your actual student number)
$ echo 123456789
```

What it does: Simply prints the text you provide **Why useful**: Creating custom output, testing redirection

```bash
# Display calendar for September 2024
$ cal 9 2024
```

What it does: Shows a calendar for the specified month and year **Why useful**: Quick date reference without leaving terminal

```bash
# Show command history
$ history
```

What it does: Lists all previously executed commands **Why useful**: Reviewing past commands, debugging, learning

```bash
# List files with inode numbers and hidden files
$ ls -ia
```

What it does:

- `-i`: Shows inode numbers (unique file identifiers)
- `-a`: Shows all files including hidden ones (starting with .) **Why useful**: Understanding file system structure and hidden files

📷 **Screenshot Required**: Capture the terminal showing all these commands and their outputs

Step 1.2: Redirect Output to Files

Now we'll execute the same commands but save their output to files:

```bash
# Redirect date output to date.txt
$ date > date.txt

# Redirect users output to users.txt
$ users > users.txt

# Redirect who output to who.txt
$ who > who.txt

# Redirect echo output to echo.txt
$ echo 123456789 > echo.txt

# Redirect calendar output to cal.txt
$ cal 9 2024 > cal.txt

# Redirect history output to history.txt
$ history > history.txt

# Redirect ls output to ls.txt
$ ls -ia > ls.txt
```

Understanding the `>` Operator:

- Creates a new file if it doesn't exist
- **Overwrites** existing file content (be careful!)
- No output appears on screen when redirected

Step 1.3: View All Created Files

```bash
# View all .txt files and their contents in one command
$ cat *.txt
```

What this does:

- `cat`: Displays file contents
- `*.txt`: Wildcard that matches all files ending in .txt
- Shows filename headers before each file's content

📷 **Screenshot Required**: Capture the command line showing all the redirection commands and the final `cat *.txt` output

Task 2: Directory Structure & Navigation

Learning Objective

Create and navigate complex directory structures while understanding working directories.

Understanding Directory Concepts

- **HOME directory** (`~`): Your personal user directory
- **Current working directory** (`pwd`): Where you are right now
- **Absolute path**: Full path from root (starts with `/`)
- **Relative path**: Path from current location

Step 2.1: Create Required Directory Structure

```bash
# Navigate to your HOME directory
$ cd ~

# Create the main Assignment4 directory
$ mkdir Assignment4

# Navigate into Assignment4
$ cd Assignment4

# Create the Assignments subdirectory
$ mkdir Assignments
```

Directory Structure Created:

```
HOME/
└── Assignment4/
    └── Assignments/
```

Step 2.2: Execute the Specified Commands

```bash
# Make Assignment4 your current working directory (if not already there)
$ cd ~/Assignment4

# Execute the echo command with output redirection
$ echo "CSN09112 Assignment4" > first-argument.txt
```

Understanding this command:

- `echo`: Outputs the text
- Arguments: "CSN09112" (first) and "Assignment4" (second)
- `>`: Redirects output to file
- Creates `first-argument.txt` with the content

Step 2.3: Extract the Second Argument

```bash
# Create file with only the second argument
$ echo Assignment4 > second-argument.txt
```

Why this works: In the original command `echo "CSN09112 Assignment4"`, the second argument is "Assignment4"

```bash
# View the content to verify
$ cat second-argument.txt
```

📷 **Screenshot Required**: Show the content of `second-argument.txt`

Step 2.4: Navigate and Create cddir.txt

```bash
# Change to Assignments directory
$ cd Assignments

# Check current location and save to file
$ pwd > cddir.txt

# Verify the content
$ cat cddir.txt
```

Understanding `pwd`:

- Stands for "Print Working Directory"
- Shows the absolute path of your current location
- Essential for understanding where you are in the file system

📷 **Screenshot Required**: Show the content of `cddir.txt`

Task 3: Complex Directory Operations

Learning Objective

Master directory creation with confusing names and learn recursive copying operations.

⚠️ CRITICAL: Character Recognition

This task tests your ability to distinguish between similar-looking characters:

- `1` (number one) vs `l` (lowercase L) vs `I` (uppercase i)
- `0` (zero) vs `O` (uppercase o) vs `o` (lowercase o)

Step 3.1: Create the Complex Directory Structure

```bash
# Ensure you're in Assignment4 directory
$ cd ~/Assignment4

# Create the main directory (1ldIr = 1, l, d, I, r)
$ mkdir 1ldIr
```

Character Breakdown for `1ldIr`:

- `1`: Number one
- `l`: Lowercase L
- `d`: Lowercase d
- `I`: Uppercase i
- `r`: Lowercase r

```bash
# Navigate into 1ldIr
$ cd 1ldIr

# Create two subdirectories with confusing names
$ mkdir a0oO    # a + zero + lowercase o + uppercase O
$ mkdir bOo0    # b + uppercase O + lowercase o + zero
```

Character Breakdown:

- `a0oO`: a, 0 (zero), o (lowercase), O (uppercase)
- `bOo0`: b, O (uppercase), o (lowercase), 0 (zero)

```bash
# Create subdirectories under a0oO
$ cd a0oO
$ mkdir O0osub  # uppercase O + zero + lowercase o + sub

# Create subdirectories under bOo0
$ cd ../bOo0
$ mkdir 0Oosub  # zero + uppercase O + lowercase o + sub
```

Final Structure:

```
Assignment4/
├── 1ldIr/
│   ├── a0oO/
│   │   └── O0osub/
│   └── bOo0/
│       └── 0Oosub/
└── Assignments/
```

Step 3.2: Copy Directory Content

```bash
# Navigate back to Assignment4
$ cd ~/Assignment4

# Copy CONTENT of 1ldIr to new directory lId1r
$ cp -r 1ldIr lId1r
```

Understanding `cp -r`:

- `cp`: Copy command
- `-r`: Recursive flag (copies directories and all contents)
- Source: `1ldlr` (existing directory)
- Destination: `lld1r` (new directory, will be created)

⚠️ **Important**: The destination directory `lld1r` must NOT exist before this command!

📷 **Screenshot Required**: Show the copy command line

Step 3.3: Verify Directory Structure

```bash
# Display tree structure (if tree command available)
$ tree Assignment4

# Alternative if tree not available:
$ find Assignment4 -type d | sort
```

📷 **Screenshot Required**: Show the tree structure output

Step 3.4: Generate Recursive Pathname List

```bash
# Navigate to Assignment4 directory
$ cd ~/Assignment4

# Generate recursive list starting from 1ldlr
$ find 1ldlr
```

Understanding `find`:

- Recursively searches directories
- Without options, lists all files and directories
- Output should be exactly 5 lines
- One line should be: `1ldlr/a0oO/O0osub`

📷 **Screenshot Required**: Show the 5-line recursive output

Step 3.5: Redirect Output to File

```bash
# Redirect the find output to myPaths.txt in the correct location
$ find 1ldlr > lld1r/a0oO/O0osub/myPaths.txt
```

Path Breakdown:

- `lld1r/a0oO/O0osub/`: The destination directory path
- `myPaths.txt`: The filename
- Note: Using `lld1r` (copied directory), not `1ldlr` (original)

📷 **Screenshot Required**: Show this command line

Step 3.6: Verify File Creation

```bash
# Navigate to the O0osub directory
$ cd lld1r/a0oO/O0osub

# List files in current directory
$ ls -la

# Display file contents
$ cat myPaths.txt
```

📷 **Screenshot Required**: Show the content of `myPaths.txt`

Step 3.7: Copy File with Relative Paths

```bash
# From O0osub, copy to 0Oosub using relative paths
$ cp myPaths.txt ../../bOo0/0Oosub/myPaths.txt.copy
```

Understanding the Relative Path:

- `../../`: Go up two directories (O0osub → a0oO → lld1r)
- `bOo0/0Oosub/`: Navigate down to target directory
- `myPaths.txt.copy`: New filename for the copy

📷 **Screenshot Required**: Show this command line

Step 3.8: Save Command History

This exercise teaches you to manipulate command history:

Step a-c: Use arrow keys to retrieve and modify the copy command:

```bash
# Use Up-Arrow to get: cp myPaths.txt ../../bOo0/0Oosub/myPaths.txt.copy
# Add "echo " at the beginning:
$ echo cp myPaths.txt ../../bOo0/0Oosub/myPaths.txt.copy
```

Step d-e: Redirect the echo output:

```bash
# Use Up-Arrow again to get the echo command, then add redirection:
$ echo cp myPaths.txt ../../bOo0/0Oosub/myPaths.txt.copy > ../../../Copycmnd.txt
```

Path Analysis for Redirection:

- `../../../`: Go up 3 levels (O0osub → a0oO → Ild1r → Assignment4)
- `Copycmnd.txt`: File in Assignment4 directory

📷 **Screenshots Required**:

1. Command line showing the echo redirection
2. Content of `Copycmnd.txt` file

Task 4: Command Review Exercise

Learning Objective

Execute a series of commands to understand file operations and track errors.

Understanding the Exercise

You'll execute 16 commands, some will produce errors (intentionally). This teaches you to recognize and understand common Linux errors.

Step 4.1: Execute All Commands

Execute these commands **exactly as shown**:

```bash
1. cd ; rm -rf ~/lab4.8
2. mkdir ~/lab4.8
3. cd ~/lab4.8
4. mkdir ./orchard
5. touch apple orange
6. mv orange orchard/lemon
7. rm orange
8. touch lettuce tomato cucumber
9. cp tomato lettuce garden
10. mkdir jardin forest
11. mkdir garden/flower
12. rmdir jardin
13. touch lab4
14. cd orchard
15. cd ../../lab4.8/forest
16. mv ../lab4 ../tomato
```

Step 4.2: Answer Questions

Question 1: Commands that generated errors:

- **Command 7**: `rm orange` - Error: "rm: cannot remove 'orange': No such file or directory"
  - **Why**: orange was moved to orchard/lemon in command 6
- **Command 9**: `cp tomato lettuce garden` - Error: "cp: cannot create regular file 'garden': No such file or directory"
  - **Why**: garden directory doesn't exist yet
- **Command 11**: `mkdir garden/flower` - Error: "mkdir: cannot create directory 'garden/flower': No such file or directory"
  - **Why**: garden directory still doesn't exist

Question 2: Absolute path after command 16:

```
/home/[username]/lab4.8/forest
```

Question 3: Absolute pathname of lemon file:

```
/home/[username]/lab4.8/orchard/lemon
```

Question 4: Relative path to lemon from forest directory:

```
../orchard/lemon
```

Question 5: Relative path to lemon from HOME directory:

```
lab4.8/orchard/lemon
```

Questions 6-10: Continue with similar path analysis...

Task 5: Advanced Find Operations

Learning Objective

Master the `find` command with various options and output redirection techniques.

Step 5.1: Command Compacting

Question: Can this be written more compactly?

```bash
cd /etc ; find . -name "*.log"
```

Answer: Yes! It can be written as:

```bash
find /etc -name "*.log"
```

Why this works:

- Instead of changing directory then using `.`, specify the full path directly
- More efficient and clearer
- Fewer commands to execute

Step 5.2: Understanding find -ls Option

```bash
$ man find
# Look for -ls option
```

Answer: The `-ls` option makes find display detailed information about each file found, similar to `ls -l` output, including permissions, size, date, etc.

Step 5.3: Find Files Ending in "log"

```bash
$ find /etc -name "*.log"
```

What this does:

- `find`: Search command
- `/etc`: Starting directory
- `-name "*.log"`: Pattern matching files ending in .log
- `*`: Wildcard matching any characters before .log

📷 **Screenshot Required**: Show command and output (including permission errors)

Step 5.4: Suppress Error Messages

```bash
$ find /etc -name "*.log" 2>/dev/null
```

Understanding Error Redirection:

- `2>`: Redirects error messages (stderr)
- `/dev/null`: Special file that discards all data sent to it
- Only successful results appear, errors are hidden

📷 **Screenshot Required**: Show command with clean output

Step 5.5: Save Results to File

```bash
$ find /etc -name "*.log" 2>/dev/null > ~/logfile
```

What this adds:

- `> ~/logfile`: Redirects successful output to file in HOME directory
- Combines error suppression with output saving

📷 **Screenshot Required**: Show this command line

Step 5.6: Count Results

```bash
$ wc -l ~/logfile
```

Understanding `wc -l`:

- `wc`: Word count command
- `-l`: Lines option (counts lines instead of words)
- Shows how many .log files were found

📷 **Screenshot Required**: Show command and count result

Summary & Key Takeaways

What You've Learned

1. **Output Redirection**:
   - `>` creates/overwrites files with command output
   - `2>` redirects error messages
   - `/dev/null` discards unwanted output
2. **Directory Operations**:
   - Character recognition is critical in Linux
   - Recursive copying preserves directory structure
   - Relative vs absolute paths have different use cases
3. **File Management**:
   - Commands can fail for logical reasons
   - Understanding error messages helps debugging
   - History manipulation saves time
4. **Find Command**:
   - Powerful tool for searching file systems
   - Can be combined with other commands
   - Options like `-name`, `-type`, `-ls` modify behavior

Best Practices Learned

- Always verify directory/file names carefully
- Use `pwd` to confirm your location
- Test commands before redirecting output
- Understand relative paths to navigate efficiently
- Read error messages to understand what went wrong

Commands Mastered

- `date`, `who`, `users`, `echo`, `cal`, `history`, `ls`
- `cd`, `mkdir`, `cp`, `mv`, `rm`, `rmdir`, `touch`
- `pwd`, `cat`, `find`, `tree`, `wc`
- Output redirection: `>`, `2>`, `2>/dev/null`
- Path navigation: absolute vs relative paths

🎉 **Congratulations!** You've completed a comprehensive Linux file system and command tutorial. These skills form the foundation for advanced Linux system administration.