Linux Assignment 5: Comprehensive Lab Manual

A Complete Educational Guide to Aliases, Variables, Arrays, and File Permissions

Table of Contents

Introduction & Setup

What You'll Learn

- **Aliases**: Create custom commands for efficiency
- **Variables & Arrays**: Store and manipulate data in shell
- **Advanced Redirection**: Master output and error handling
- **File Permissions**: Understand and modify access controls
- **Log Analysis**: Extract meaningful data from system logs
- **Shell Customization**: Personalize your Linux environment

Initial Setup: Create Directory Structure

First, create the required directory structure in your HOME directory:

```bash
# Navigate to HOME directory
cd ~

# Create main Assignment5 directory
mkdir Assignment5

# Navigate into Assignment5
cd Assignment5

# Create subdirectories a, b, c
mkdir a b c

# Create files Week5c and Week5a
touch Week5c Week5a

# Verify structure
tree Assignment5
# or if tree not available:
find Assignment5 -type f -o -type d | sort
```

Expected Structure:

```
Assignment5/
├── a/
├── b/
├── c/
├── Week5c
└── Week5a
```

Task 1: Aliases and Find Operations

Learning Objective

Master creating custom commands (aliases) and advanced file searching techniques.

Task 1(a): Creating and Managing Aliases

Step 1.1: Compare `ls` vs `ls -ail`

```bash
# Basic listing
ls

# Advanced listing with all options
ls -ail
```

Explanation of `ls -ail` options:

- `-a`: Shows **all** files, including hidden ones (starting with `.`)
- `-i`: Shows **inode** numbers (unique file identifiers)
- `-l`: **Long format** - shows permissions, owner, size, date, etc.

Why this matters: The advanced version gives complete file information essential for system administration.

📸 **Screenshot Required**: Show both outputs side by side

Step 1.2: Create "list" Alias

```bash
# Create alias for ls -ail
alias list='ls -ail'

# Test the new alias
list
```

Understanding Aliases:

- **Purpose**: Create shortcuts for frequently used commands
- **Syntax**: `alias name='command'`
- **Scope**: Only active in current session (unless saved)

📸 **Screenshot Required**: Show the alias creation and execution

Step 1.3: View All Existing Aliases

```bash
# Display all current aliases
alias
```

What you'll see: System-defined and user-defined aliases. Look for your `list` alias in the output.

📸 **Screenshot Required**: Show all aliases including your new `list` alias

Step 1.4: Create "2day" Alias for Weekday

```bash
# Create alias to show current weekday
alias 2day='date +%A'

# Test the alias
2day
```

Understanding `date +%A`:

- `date`: System date command
- `+%A`: Format specifier for full weekday name
- **Output**: Monday, Tuesday, etc.

📸 **Screenshot Required**: Show `2day` command working

Step 1.5: Create "files" Alias for Recursive File Listing

```bash
# Create alias to find all files under HOME directory
alias files='find /home/$USER -type f'

# Test the alias
files
```

Understanding this alias:

- `find /home/$USER`: Searches your home directory
- `-type f`: Only finds regular files (not directories)
- `$USER`: Environment variable containing your username

📸 **Screenshot Required**: Show `files` command execution

Task 1(b): Finding Program Locations

Step 1.6: Locate the `cat` Command

```bash
# Ensure you're in HOME directory
cd ~

# Find cat command location, suppress errors
find / -name "cat" -type f 2>/dev/null
```

Understanding this command:

- `find /`: Search from root directory
- `-name "cat"`: Look for files named exactly "cat"
- `-type f`: Only regular files
- `2>/dev/null`: Redirect error messages to nowhere

Why `/dev/null`?: It's a special "black hole" file that discards everything sent to it. This hides "Permission denied" errors from directories you can't access.

📸 **Screenshot Required**: Show the find command and cat location results

Task 1(c): Pattern Matching with Find

Step 1.7: Find Files Starting with "Week" (6 characters total)

```bash
# Find files starting with "Week" and exactly 6 characters long
find ~ -name "Week??" -type f
```

Understanding the Pattern:

- `Week??`: "Week" followed by exactly 2 more characters
- `?`: Wildcard matching exactly one character
- This finds files like: Week5c, Week5a (both 6 characters)

📸 **Screenshot Required**: Show command and results with absolute paths

Step 1.8: Count the Results Using Pipes

```bash
# Count how many files match the pattern
find ~ -name "Week??" -type f | wc -l
```

Understanding Pipes (`|`):

- Takes output from first command
- Sends it as input to second command
- `wc -l`: Counts lines of input
- **Result**: Number of matching files

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

Task 2: Output Redirection Mastery

Learning Objective

Master different types of output redirection and understand when to use each.

Step 2.1: Basic Output Redirection

```bash
# Redirect /etc/passwd to dump.txt in Assignment5 directory
cat /etc/passwd > ~/Assignment5/dump.txt

# Verify the file was created
ls -l ~/Assignment5/dump.txt
```

📸 **Screenshot Required**: Show the redirection command

Step 2.2: Understanding Overwrite vs Append

```bash
# This will OVERWRITE dump.txt completely
ls /home/user1 > ~/Assignment5/dump.txt

# View the new content (old content is gone!)
cat ~/Assignment5/dump.txt
```

Why content disappeared: The `>` operator **overwrites** the entire file with new content.

Answer: The `>` operator truncates (empties) the file before writing new content, so previous data is lost.

Step 2.3: Append Operations

```bash
# Use >> to APPEND instead of overwrite
ls /etc/passwd >> ~/Assignment5/dump.txt

# View the combined content
cat ~/Assignment5/dump.txt
```

Difference between `>` and `>>`:

- `>`: **Overwrites** - replaces all file content
- `>>`: **Appends** - adds to end of existing content

Step 2.4: Error Redirection

```bash
# Command that produces both output and error
ls W5 Assignment5
```

Expected result: Error for W5 (doesn't exist), success for Assignment5

```bash
# Redirect ONLY errors to file
ls W5 Assignment5 2> ~/Assignment5/dirlist

# Check what was captured
cat ~/Assignment5/dirlist
```

Understanding `2>`:

- `2`: File descriptor for stderr (error messages)
- Standard output still appears on screen
- Only errors go to file

📸 **Screenshot Required**: Show error redirection command

Step 2.5: Redirect Both Output and Errors

```bash
# Redirect both standard output and errors to same file
ls W5 Assignment5 > ~/Assignment5/Both.txt 2>&1

# View the combined output
cat ~/Assignment5/Both.txt
```

Understanding `2>&1`:

- `2>&1`: Redirect errors to wherever output is going
- Both success and error messages end up in the file
- Nothing appears on screen

📸 **Screenshot Required**: Show combined redirection command

Step 2.6: Piping Operations

Count Pathnames in /etc

```bash
# Count all pathnames under /etc
find /etc | wc -l
```

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

Show First 5 Lines

```bash
# Show only first 5 pathnames
find /etc | head -5
```

Count Only Files (not directories)

```bash
# Count only regular files under /etc
find /etc -type f | wc -l
```

📸 **Screenshot Required**: Show command and file count

Extract Specific Line Range (50-100)

```bash
# Get lines 50 through 100 from find results
find /etc -type f | head -100 | tail -51
```

Understanding the Pipeline:

1. `find /etc -type f`: Generate all file paths
2. `head -100`: Take first 100 lines
3. `tail -51`: From those 100, take last 51 (lines 50-100)

📸 **Screenshot Required**: Show command and line range output

Paginated Output

```bash
# View /etc contents page by page
ls /etc | more
```

Use: Space to go forward, 'q' to quit

Task 3: Variables and Arrays

Learning Objective

Understand shell variables and array operations for data storage and manipulation.

Step 3.1: Declare Variables

```bash
# Declare your variables (replace with your actual information)
Name="John Doe Smith"
City="Ottawa"
Student_Number="041234567"
Username="smit1234"
```

Variable Rules:

- No spaces around `=`
- Use quotes for values with spaces
- Case-sensitive names
- No special characters in names

Step 3.2: View Variables with set and grep

```bash
# View all variables (very long output)
set

# Filter to show only your variables
set | grep -E "(Name|City|Student_Number|Username)="
```

Understanding the Filter:

- `set`: Shows all shell variables
- `grep -E`: Extended regex matching
- `(Name|City|Student_Number|Username)`: Match any of these names
- `=`: Ensure we match variable assignments

📸 **Screenshot Required**: Show filtered variable display

Step 3.3: Display Variable Values

```bash
# Display all variable values in one line
echo "Name: $Name, City: $City, Student Number: $Student_Number, Username: $Username"
```

Variable Access:

- `$variablename`: Gets the value stored in the variable
- Variables must be prefixed with `$` when accessing values

📸 **Screenshot Required**: Show echo command output

Step 3.4: Working with Arrays

Understanding Arrays

- **Definition**: Variables that hold multiple values
- **Index**: Position of each value (starts at 0)
- **Access**: Use index to get specific values

Create and Use Arrays

```bash
# Create array with your information
My_Info=("John Doe Smith" "Ottawa" "041234567" "smit1234")

# Display array creation
echo "Array created: ${My_Info[@]}"
```

📸 **Screenshot Required**: Show array creation

Access Specific Array Elements

```bash
# Get student number (index 2) and username (index 3)
echo "Student Number: ${My_Info[2]}"
echo "Username: ${My_Info[3]}"

# Alternative: both in one command
echo "Student Number: ${My_Info[2]}, Username: ${My_Info[3]}"
```

Array Syntax:

- `${arrayname[index]}`: Access specific element
- `${arrayname[@]}`: Access all elements
- Indexes start at 0

📸 **Screenshot Required**: Show specific element access

Task 4: Custom Welcome Banner

Learning Objective

Customize your shell environment by modifying the `.bashrc` file to display a personalized welcome message.

Understanding .bashrc

- **Purpose**: Configuration file executed when you open a new terminal
- **Location**: `~/.bashrc` (hidden file in HOME directory)
- **Function**: Sets up your shell environment, aliases, and custom features

Step 4.1: Edit .bashrc File

```bash
# Navigate to HOME directory
cd ~

# Backup the original .bashrc (safety measure)
cp .bashrc .bashrc.backup

# Edit .bashrc (use nano, vim, or gedit)
nano .bashrc
```

Step 4.2: Add Welcome Banner Code

Add these lines at the **end** of the `.bashrc` file:

```bash
# Custom Welcome Banner - Assignment 5
echo "============================================"
echo "          WELCOME TO LINUX"
echo "============================================"
echo
echo "Username is    : $USER"
echo "Current Date   : $(date '+%A, %B %d, %Y')"
echo "Current Time   : $(date '+%H:%M:%S')"
echo "Hostname       : Your Full Name Here"  # Replace with your actual name
echo "Home Directory: $HOME"
echo "Current Path   : $PWD"
echo
echo "============================================"
echo "       Have a productive session!"
echo "============================================"
echo
```

Understanding the Variables:

- `$USER`: Current username
- `$(date '+%A, %B %d, %Y')`: Current date in readable format
- `$(date '+%H:%M:%S')`: Current time
- `$HOME`: Path to home directory
- `$PWD`: Current working directory

Step 4.3: Save and Test

```bash
# Save the file (Ctrl+X, then Y, then Enter in nano)

# Close and reopen terminal to see the banner
# OR reload .bashrc:
source ~/.bashrc
```

📸 **Screenshot Required**: Show your custom welcome banner when terminal opens

---

Task 5: Log File Analysis

Learning Objective

Learn to analyze system logs using command-line tools to extract security-relevant information.

Step 5.1: Download and Locate Log File

```bash
# Find where your logfile was downloaded (usually Downloads folder)
find ~ -name "*log*" -type f 2>/dev/null

# Or check common download locations
ls ~/Downloads/
ls ~/Desktop/
```

📸 **Screenshot Required**: Show find command locating the log file

Step 5.2: Copy Log File to Assignment Directory

```bash
# Create logs directory
mkdir -p ~/Assignment5/logs

# Copy the log file (adjust path as needed)
cp ~/Downloads/[logfilename] ~/Assignment5/logs/

# Verify the copy
ls -l ~/Assignment5/logs/
```

📸 **Screenshot Required**: Show copy operation and verification

Step 5.3: Analyze SSH Break-in Attempts

This complex command chain demonstrates advanced Linux text processing:

```bash
# Navigate to the logs directory
cd ~/Assignment5/logs

# Complex pipeline to analyze SSH attacks
grep "refused connect" [logfilename] | grep "Jan " | awk '{print $NF}' | sort | uniq -c | sort -nr
```

Breaking Down the Pipeline:

1. `grep "refused connect" [logfilename]`
   - Finds all lines containing "refused connect"
   - These indicate failed SSH connection attempts
2. `| grep "Jan "`
   - From those lines, only keep ones containing "Jan "
   - Filters to January attacks only
3. `| awk '{print $NF}'`
   - `awk`: Text processing tool
   - `$NF`: Last field in each line (the IP address)
   - Extracts just the attacking IP addresses
4. `| sort`
   - Sorts IP addresses alphabetically
   - Required for `uniq` to work properly
5. `| uniq -c`
   - Counts consecutive identical lines
   - `-c`: Prefixes each unique line with its count
6. `| sort -nr`
   - `sort`: Sort the results
   - `-n`: Numerical sort (by count)
   - `-r`: Reverse order (highest first)

Expected Output Format:

```
23 192.168.1.100
15 10.0.0.50
8 172.16.0.25
```

📸 **Screenshot Required**: Show the complete command and its output

Understanding the Security Context

What this analysis reveals:

- **Attack patterns**: Which IPs are most aggressive
- **Timing**: Attacks concentrated in January
- **Scale**: Total number of attack attempts
- **Sources**: Geographic distribution of attackers

Real-world application: System administrators use similar commands to:

- Identify security threats
- Block malicious IPs
- Generate security reports
- Monitor system health

Task 6: File Permissions Deep Dive

Learning Objective

Master Linux file permissions in both symbolic and octal formats, and understand their practical effects.

Permission System Overview

Three Permission Types:

- **r (read)**: View file content or list directory contents
- **w (write)**: Modify file content or create/delete files in directory
- **x (execute)**: Run file as program or enter directory

Three User Categories:

- **User/Owner**: Creator of the file
- **Group**: Users in the same group
- **Other**: Everyone else

Step 6.1: Symbolic to Octal Conversion

Conversion Method:

- **r** = 4, **w** = 2, **x** = 1
- Add values for each permission set

Symbolic Mode	Octal Mode	User/Owner	Group	Other
rwxrw-r-x	765	7 (4+2+1)	6 (4+2)	5 (4+1)
r--wx-w-	432	4 (4+0+0)	3 (0+2+1)	2 (0+2+0)
--x------	100	1 (0+0+1)	0 (0+0+0)	0 (0+0+0)

Step 6.2: Octal to Symbolic Conversion

Octal Mode	Symbolic Mode	User/Owner	Group	Other
001	------x	--- (0)	--- (0)	--x (1)
421	r---w---x	r-- (4)	-w- (2)	--x (1)
300	-wx------	-wx (3)	--- (0)	--- (0)
504	r-x---r--	r-x (5)	--- (0)	r-- (4)
756	rwxr-xrw-	rwx (7)	r-x (5)	rw- (6)

Step 6.3: Practical Permission Testing

Create Test Structure

```bash
# Navigate to Assignment5
cd ~/Assignment5

# Create task6 directory and subdirectories
mkdir task6
cd task6
mkdir dir1 dir2 dir3

# Create files in each directory
echo "This is f1.txt content" > dir1/f1.txt
echo "This is f2.txt content" > dir2/f2.txt
echo "This is f3.txt content" > dir3/f3.txt
```

View All Files Together

```bash
# Display all files with their names
cat dir1/f1.txt dir2/f2.txt dir3/f3.txt
```

📸 **Screenshot Required**: Show all file contents

Set Specific File Permissions

```bash
# Set specific permissions for each file (owner only)
chmod u=x,go= dir1/f1.txt     # Execute only for owner
chmod u=w,go= dir2/f2.txt     # Write only for owner
chmod u=r,go= dir3/f3.txt     # Read only for owner

# Verify permissions
ls -l dir1/f1.txt dir2/f2.txt dir3/f3.txt
```

📸 **Screenshot Required**: Show file permissions

Test File Access

```bash
# Try to read all files
cat dir1/f1.txt dir2/f2.txt dir3/f3.txt
```

Expected behavior:

- **f1.txt**: Can't read (no read permission)
- **f2.txt**: Can't read (no read permission)
- **f3.txt**: Can read (has read permission)

Answer: You cannot see the content of f1.txt and f2.txt because they don't have read permission for the owner.

Test File Writing

```bash
# Try to append to all files
echo "Additional text" >> dir1/f1.txt
echo "Additional text" >> dir2/f2.txt
echo "Additional text" >> dir3/f3.txt
```

Expected behavior:

- **f1.txt**: Can't write (no write permission)
- **f2.txt**: Can write (has write permission)
- **f3.txt**: Can't write (no write permission)

Answer: You can only append to f2.txt because it has write permission. The others fail due to lack of write permission.

📸 **Screenshot Required**: Show append attempts

Set Directory Permissions

```bash
# Set specific permissions for directories (owner only)
chmod u=x,go= dir1      # Execute only
chmod u=w,go= dir2      # Write only
chmod u=r,go= dir3      # Read only

# Verify directory permissions
ls -ld dir1 dir2 dir3
```

📸 **Screenshot Required**: Show directory permissions

Test Directory Access

```bash
# Ensure you're in task6 directory
cd ~/Assignment5/task6

# Try to enter each directory
cd dir1    # Should work
cd ..      # Return to task6
cd dir2    # Should fail
cd dir3    # Should fail
```

Answer: You can cd to dir1 because it has execute permission. Directories need execute permission to be entered. dir2 and dir3 fail because they lack execute permission.

Test Directory Listing

```bash
# Try to list contents of each directory
ls -l dir1
ls -l dir2
ls -l dir3
```

Answer: You can list dir3 content because it has read permission. Directories need read permission for listing contents.

Minimum Permissions for File Access

To read file contents, you need:

- **File**: Read permission (r)
- **Parent directory**: Execute permission (x) to traverse to the file

```bash
# Set minimum required permissions
chmod u=rx,go= dir1    # Read + execute for directory
chmod u=rx,go= dir2    # Read + execute for directory
chmod u=rx,go= dir3    # Read + execute for directory
chmod u=r,go= dir1/f1.txt  # Read for file
chmod u=r,go= dir2/f2.txt  # Read for file
chmod u=r,go= dir3/f3.txt  # Read for file
```

Minimum Permissions Table:

Item	Permissions	Reason
dir1	r-x (5)	Read to list + Execute to enter
dir2	r-x (5)	Read to list + Execute to enter
dir3	r-x (5)	Read to list + Execute to enter
f1.txt	r-- (4)	Read to view content
f2.txt	r-- (4)	Read to view content
f3.txt	r-- (4)	Read to view content

Summary & Key Takeaways

Skills Mastered

1. **Aliases**: Created custom commands for efficiency
2. **Variables**: Stored and manipulated shell data
3. **Arrays**: Managed multiple values in single variables
4. **Redirection**: Controlled command output and errors
5. **File Permissions**: Understood and modified access controls
6. **Log Analysis**: Extracted security information from system logs
7. **Shell Customization**: Personalized the Linux environment

Advanced Concepts Learned

- **Pipeline Processing**: Chaining commands for complex operations
- **Pattern Matching**: Using wildcards and regular expressions
- **Error Handling**: Managing and redirecting error output
- **Security Analysis**: Identifying patterns in log files
- **Environment Customization**: Modifying shell behavior

Real-world Applications

- **System Administration**: Managing user access and security
- **Security Monitoring**: Analyzing attack patterns
- **Automation**: Creating shortcuts for common tasks
- **Data Processing**: Extracting information from large files
- **Environment Setup**: Customizing user experiences

Commands Mastered

- `alias`, `find`, `grep`, `awk`, `sort`, `uniq`, `wc`
- `chmod`, `cat`, `echo`, `head`, `tail`, `more`
- Advanced redirection: `>`, `>>`, `2>`, `2>&1`, `|`
- Variables: declaration, access, arrays
- File permissions: symbolic and octal notation

🎉 **Congratulations!** You've completed an advanced Linux administration tutorial covering essential system management skills. These techniques form the backbone of professional Linux system administration and security analysis.