Linux Command Piping and Redirection

1. Introduction

Command piping and redirection are fundamental Linux concepts that allow you to control where command input comes from and where output goes. These techniques enable you to chain commands together and create powerful one-liners that would otherwise require complex scripts.

2. Output Redirection (> and >>)

Basic Output Redirection (>)

The `>` operator redirects stdout to a file, **overwriting** any existing content.

Syntax:

```
command > filename
```

Step-by-Step Example 1: Creating a File

```
# Step 1: Create a file with simple text
echo "Hello, World!" > greeting.txt

# Step 2: Verify the file was created
ls -l greeting.txt

# Step 3: View the contents
cat greeting.txt
# Output: Hello, World!
```

Step-by-Step Example 2: Saving Command Output

```
# Step 1: List all files and save to a file
ls -la > directory_listing.txt

# Step 2: Check the file was created
ls -l directory_listing.txt

# Step 3: View the saved listing
cat directory_listing.txt
```

Step-by-Step Example 3: Overwriting Behavior

```
# Step 1: Create initial content
echo "First content" > test.txt
cat test.txt
# Output: First content

# Step 2: Overwrite with new content
echo "Second content" > test.txt
cat test.txt
# Output: Second content (first content is gone!)
```

Warning: Using `>` will completely erase any existing file content! Be careful when using it.

Append Output Redirection (>>)

The `>>` operator redirects stdout to a file, **appending** to existing content without erasing it.

Syntax:

```
command >> filename
```

Step-by-Step Example 1: Building a Log File

```bash
# Step 1: Create initial log entry
echo "Log started at $(date)" > activity.log

# Step 2: View current content
cat activity.log

# Step 3: Add more entries using append
echo "User logged in" >> activity.log
echo "File uploaded" >> activity.log
echo "User logged out" >> activity.log

# Step 4: View all accumulated entries
cat activity.log
# Output:
# Log started at [date/time]
# User logged in
# File uploaded
# User logged out
```

Step-by-Step Example 2: Collecting Data Over Time

```bash
# Step 1: Create a file to track disk usage
df -h > disk_usage.txt

# Step 2: Add a separator and timestamp
echo "---" >> disk_usage.txt
echo "Checked at: $(date)" >> disk_usage.txt

# Step 3: Later, add another check
echo "---" >> disk_usage.txt
df -h >> disk_usage.txt
echo "Checked at: $(date)" >> disk_usage.txt

# Step 4: View the complete history
cat disk_usage.txt
```

Step-by-Step Example 3: Building Lists

```
# Step 1: Create a task list
echo "1. Complete report" > tasks.txt

# Step 2: Add more tasks
echo "2. Send emails" >> tasks.txt
echo "3. Review code" >> tasks.txt
echo "4. Update documentation" >> tasks.txt

# Step 3: View the complete list
cat tasks.txt
```

Choosing Between > and >>

Use `>` when:

- Starting a new file from scratch
- Replacing old data completely

Use `>>` when:

- Adding to existing files
- Building logs over time
- Accumulating results
- Preserving previous content

Step-by-Step Example: Practical Usage

```
# Step 1: Start a daily report (overwrite yesterday's)
echo "=== Daily Report ===" > daily_report.txt
echo "Date: $(date)" >> daily_report.txt

# Step 2: Add sections by appending
echo "" >> daily_report.txt
echo "Tasks Completed:" >> daily_report.txt
echo "- Task 1" >> daily_report.txt
echo "- Task 2" >> daily_report.txt

# Step 3: Add more sections
echo "" >> daily_report.txt
echo "Issues Found:" >> daily_report.txt
echo "- Issue 1" >> daily_report.txt

# Step 4: View the complete report
cat daily_report.txt
```

Redirecting to Special Files

---

4. Input Redirection (<)

Basic Input Redirection

The `<` operator redirects stdin to read from a file instead of the keyboard.

Syntax:

```
command < input_file
```

Step-by-Step Example 1: Reading from a File

```
# Step 1: Create a file with unsorted names
echo "Zebra" > animals.txt
echo "Elephant" >> animals.txt
echo "Ant" >> animals.txt
echo "Bear" >> animals.txt

# Step 2: Sort the names using input redirection
sort < animals.txt
# Output: Ant, Bear, Elephant, Zebra

# Step 3: Save sorted output to a new file
sort < animals.txt > sorted_animals.txt

# Step 4: Verify the sorted file
cat sorted_animals.txt
```

Step-by-Step Example 2: Counting Words in a File

```
# Step 1: Create a file with multiple lines
echo "The quick brown fox jumps over the lazy dog" > sentence.txt

# Step 2: Count words using input redirection
wc -w < sentence.txt
# Output: 9

# Step 3: Get multiple statistics
wc < sentence.txt
# Output: lines, words, characters
```

5. Piping Commands (|)

Understanding Pipes

Definition: A **pipe** connects the standard output of one command directly to the standard input of another command, creating a data flow.

Syntax:

```
command1 | command2
```

The output of command1 becomes the input of command2.

Basic Piping Examples

Step-by-Step Example 1: List and Count

```
# Step 1: List all files in current directory
ls

# Step 2: Count how many files there are
ls | wc -l

# Step 3: See what's happening
# ls produces a list of files (output)
# | sends that output as input to wc
# wc -l counts the lines (number of files)
```

Step-by-Step Example 2: Search Through Output

```
# Step 1: View all running processes
ps aux

# Step 2: Filter to find specific processes
ps aux | grep "firefox"

# Step 3: Understanding the flow
# ps aux lists all processes
# | pipes that list to grep
# grep filters only lines containing "firefox"
```

Step-by-Step Example 3: View Large Output Page by Page

```
# Step 1: Generate large output
ls -la /usr/bin

# Step 2: Pipe to less for easy viewing
ls -la /usr/bin | less

# Step 3: Navigate using
# Space = next page
# b = previous page
# q = quit
```

Chaining Multiple Pipes

Step-by-Step Example 1: Three-Command Pipeline

```
# Step 1: List files, filter for .txt, count them
ls -la | grep ".txt" | wc -l

# Flow explanation:
# ls -la → produces file listing
# | → pipes to grep
# grep ".txt" → filters only .txt files
# | → pipes to wc
# wc -l → counts the lines (number of .txt files)
```

Step-by-Step Example 2: Complex Data Processing

```
# Step 1: Show all users on the system
cat /etc/passwd

# Step 2: Extract just usernames (first field)
cat /etc/passwd | cut -d':' -f1

# Step 3: Sort them alphabetically
cat /etc/passwd | cut -d':' -f1 | sort

# Step 4: Show only first 10
cat /etc/passwd | cut -d':' -f1 | sort | head -n 10
```

Step-by-Step Example 3: Finding Largest Files

```
# Step 1: List all files with sizes
ls -lh

# Step 2: Sort by size (5th column, human-readable, reverse)
ls -lh | sort -k5 -hr

# Step 3: Show only top 5 largest
ls -lh | sort -k5 -hr | head -n 5

# Step 4: Save results to a file
ls -lh | sort -k5 -hr | head -n 5 > largest_files.txt
```

6. Practical Examples

Example 1: Log File Analysis

Task: Analyze a web server access log to find the most frequent visitors.

Step-by-Step Solution:

```bash
# Step 1: Create a sample log file
echo "192.168.1.100 - GET /index.html" > access.log
echo "192.168.1.101 - GET /about.html" >> access.log
echo "192.168.1.100 - GET /contact.html" >> access.log
echo "192.168.1.102 - GET /index.html" >> access.log
echo "192.168.1.100 - GET /services.html" >> access.log

# Step 2: Extract IP addresses (first field)
cat access.log | cut -d' ' -f1 > ips.txt

# Step 3: Sort the IP addresses
sort < ips.txt > sorted_ips.txt

# Step 4: Count unique occurrences
sort < ips.txt | uniq -c > ip_counts.txt

# Step 5: Sort by frequency (highest first)
sort < ips.txt | uniq -c | sort -rn > top_visitors.txt

# Step 6: View the results
cat top_visitors.txt
```

Example 2: System Information Report

Task: Create a comprehensive system report.

Step-by-Step Solution:

```
# Step 1: Start the report with a header
echo "=== System Report ===" > system_report.txt
echo "Generated on: $(date)" >> system_report.txt
echo "" >> system_report.txt

# Step 2: Add system uptime
echo "=== System Uptime ===" >> system_report.txt
uptime >> system_report.txt
echo "" >> system_report.txt

# Step 3: Add disk usage
echo "=== Disk Usage ===" >> system_report.txt
df -h >> system_report.txt
echo "" >> system_report.txt

# Step 4: Add memory information
echo "=== Memory Usage ===" >> system_report.txt
free -h >> system_report.txt
echo "" >> system_report.txt

# Step 5: Add top processes
echo "=== Top 5 Processes by CPU ===" >> system_report.txt
ps aux | sort -k3 -rn | head -n 6 >> system_report.txt

# Step 6: View the complete report
cat system_report.txt
```

Example 3: Text Processing and Cleanup

Task: Process a messy data file and clean it up.

Step-by-Step Solution:

```
# Step 1: Create a messy data file
echo "  apple  " > fruits_messy.txt
echo "BANANA" >> fruits_messy.txt
echo "  Cherry  " >> fruits_messy.txt
echo "apple" >> fruits_messy.txt
echo "banana" >> fruits_messy.txt

# Step 2: Convert to lowercase
tr 'A-Z' 'a-z' < fruits_messy.txt > fruits_lower.txt

# Step 3: Remove leading/trailing spaces and sort
cat fruits_lower.txt | sed 's/^[[:space:]]*//;s/[[:space:]]*$//' | sort >
fruits_clean.txt

# Step 4: Remove duplicates
sort < fruits_clean.txt | uniq > fruits_final.txt

# Step 5: View the cleaned result
cat fruits_final.txt
```

Example 4: Backup File List

Task: Create a list of files to backup with timestamps.

Step-by-Step Solution:

```bash
# Step 1: Create the backup list header
echo "Backup List Created: $(date)" > backup_list.txt
echo "========================================" >> backup_list.txt
echo "" >> backup_list.txt

# Step 2: Find all important files (example: .txt and .pdf)
find . -name "*.txt" > temp_files.txt
find . -name "*.pdf" >> temp_files.txt

# Step 3: Sort the file list
sort < temp_files.txt >> backup_list.txt

# Step 4: Add file count
echo "" >> backup_list.txt
echo "Total files to backup:" >> backup_list.txt
cat temp_files.txt | wc -l >> backup_list.txt

# Step 5: Clean up temporary file
rm temp_files.txt

# Step 6: View the backup list
cat backup_list.txt
```

Example 5: User Activity Monitor

Task: Monitor and log user activity.

Step-by-Step Solution:

```
# Step 1: Create log header
echo "User Activity Log" > user_activity.log
echo "Started: $(date)" >> user_activity.log
echo "==================" >> user_activity.log

# Step 2: Log currently logged-in users
echo "" >> user_activity.log
echo "Currently logged in:" >> user_activity.log
who | tee -a user_activity.log

# Step 3: Count active sessions
echo "" >> user_activity.log
echo "Active sessions:" >> user_activity.log
who | wc -l >> user_activity.log

# Step 4: Show recent logins
echo "" >> user_activity.log
echo "Recent logins:" >> user_activity.log
last -n 10 >> user_activity.log

# Step 5: Display the log
cat user_activity.log
```

Example 6: Quick Statistics Generator

Task: Generate statistics from a numbers file.

Step-by-Step Solution:

```
# Step 1: Create a numbers file
echo "45" > numbers.txt
echo "23" >> numbers.txt
echo "67" >> numbers.txt
echo "12" >> numbers.txt
echo "89" >> numbers.txt
echo "34" >> numbers.txt

# Step 2: Sort the numbers
sort -n < numbers.txt > numbers_sorted.txt

# Step 3: Find the smallest number
head -n 1 < numbers_sorted.txt > stats.txt

# Step 4: Label it
echo "Minimum:" > stats_report.txt
cat stats.txt >> stats_report.txt

# Step 5: Find the largest number
echo "Maximum:" >> stats_report.txt
tail -n 1 < numbers_sorted.txt >> stats_report.txt

# Step 6: Count how many numbers
echo "Count:" >> stats_report.txt
cat numbers.txt | wc -l >> stats_report.txt

# Step 7: View the statistics
cat stats_report.txt
```