The Linux Command Line - Practical Session2

Course Overview

This comprehensive lesson plan covers part 2 of the essential Linux command in a progressive order suitable for beginners. Each lesson builds upon previous knowledge and includes detailed explanations, practical examples, and hands-on exercises.

Lesson 1: The `find` Command

Learning Objectives

- Understand file system navigation and searching
- Master basic and advanced `find` syntax
- Use `find` for system administration tasks
- Combine `find` with other commands effectively

1.1 Introduction to `find`

The `find` command is one of the most powerful tools for locating files and directories in Linux. It searches through directory hierarchies based on various criteria such as name, size, type, permissions, and modification time.

Basic Syntax:

```
find [starting-point] [expression]
```

1.2 Basic Usage Examples

Finding by Name

```
# Find files by exact namecd
find /home -name "config.txt"

# Find files with wildcards (always use quotes with wildcards!)
find . -name "*.txt"
find . -name "test*"
find . -name "*config*"

# Case-insensitive search
find . -iname "CONFIG.txt"
find . -iname "*.PDF"

# Find files NOT matching pattern
find . -not -name "*.tmp"
```

Finding by Type

```
# Find only regular files
find . -type f -name "*.log"

# Find only directories
find . -type d -name "backup*"

# Find symbolic links
find . -type l

# Find block devices
find /dev -type b

# Find character devices
find /dev -type c
```

Finding by Size

```
# Files larger than 100MB
find . -type f -size +100M

# Files smaller than 1KB
find . -type f -size -1k

# Files exactly 50 bytes
find . -type f -size 50c

# Size units: c (bytes), k (KB), M (MB), G (GB)
find . -type f -size +1G
```

1.3 Advanced `find` Operations

Time-based Searches

```
# Files modified in last 7 days
find . -type f -mtime -7

# Files accessed in last 24 hours
find . -type f -atime -1

# Files changed more than 30 days ago
find . -type f -ctime +30

# Files modified in last 60 minutes
find . -type f -mmin -60

# Files modified between 7 and 14 days ago
find . -type f -mtime +7 -mtime -14
```

Permission-based Searches

```
# Find files with specific permissions (exact match)
find . -type f -perm 644

# Find files with at least these permissions
find . -type f -perm -644

# Find files with any of these permissions
find . -type f -perm /u+x

# Find world-writable files (security check)
find . -type f -perm /o+w

# Find SUID files (security check)
find . -type f -perm /u+s
```

User and Group Searches

```
# Find files owned by specific user
find . -user john

# Find files owned by specific group
find . -group developers

# Find files owned by current user
find . -user $(whoami)

# Find files with no valid user (orphaned)
find . -nouser

# Find files with no valid group
find . -nogroup
```

1.4 Executing Commands on Found Files

Basic Execution

```
# Delete found files (be very careful!)
find . -name "*.tmp" -delete

# Execute command on each file (one by one)
find . -name "*.txt" -exec cat {} \;

# Execute command with confirmation prompt
find . -name "*.bak" -ok rm {} \;

# Execute command on multiple files at once
find . -name "*.txt" -exec cat {} +
```

Advanced Execution Examples

```
# Copy found files to backup directory
find . -name "*.conf" -exec cp {} /backup/ \;

# Change permissions on found files
find . -name "*.sh" -exec chmod +x {} \;

# Print detailed info about found files
find . -name "*.log" -exec ls -lh {} \;

# Find and archive files
find . -name "*.txt" -exec tar -czf text-files.tar.gz {} +
```

1.5 Using `find` with Pipes and `xargs`

```
# Use with xargs for efficiency
find . -name "*.log" | xargs grep "ERROR"

# Handle filenames with spaces
find . -name "*.txt" -print0 | xargs -0 grep "pattern"

# Count total size of found files
find . -name "*.jpg" -print0 | xargs -0 du -ch

# Process files in batches
find . -name "*.txt" | xargs -n 5 cat
```

1.6 Combining Multiple Criteria

```
# AND condition (both must be true)
find . -name "*.txt" -size +1M

# OR condition
find . \( -name "*.txt" -o -name "*.doc" \)

# Complex combinations
find . -type f \( -name "*.txt" -o -name "*.doc" \) -mtime -7 -size +1k

# NOT condition
find . -type f -not -name "*.tmp" -not -name "*.log"
```

Lesson 2: The `grep` Command

Learning Objectives

- Master pattern searching in files and streams
- Understand regular expressions and their power
- Combine `grep` with other commands effectively
- Use advanced `grep` options for complex searches

2.1 Introduction to `grep`

`grep` (Global Regular Expression Print) is a fundamental tool for searching text patterns. It's essential for log analysis, code searching, and data filtering. `grep` processes text line by line and prints lines that match specified patterns.

Basic Syntax:

```
grep [options] pattern [file...]
grep [options] -e pattern1 -e pattern2 [file...]
```

2.2 Basic `grep` Usage

Simple Text Searches

```
# Search for a word in a file
grep "error" /var/log/syslog

# Search in multiple files
grep "TODO" *.txt *.py *.sh

# Search recursively in directories
grep -r "password" /etc/

# Case-insensitive search
grep -i "Error" logfile.txt
grep -i "warning" *.log
```

Essential Options

```
# Show line numbers
grep -n "pattern" file.txt

# Show only the matching part
grep -o "pattern" file.txt

# Show lines NOT matching pattern (invert)
grep -v "pattern" file.txt

# Count matches (lines containing pattern)
grep -c "pattern" file.txt

# Show only filenames containing pattern
grep -l "pattern" *.txt

# Show filenames NOT containing pattern
grep -L "pattern" *.txt
```

2.3 Advanced Options and Techniques

Context and Formatting

```
# Show 3 lines after match
grep -A 3 "error" logfile.txt

# Show 3 lines before match
grep -B 3 "error" logfile.txt

# Show 3 lines before and after (context)
grep -C 3 "error" logfile.txt

# Highlight matches in color
grep --color=always "pattern" file.txt

# Suppress filename in output
grep -h "pattern" *.txt

# Show filename for single file
grep -H "pattern" file.txt
```

Multiple Patterns and Files

```
# Multiple patterns (OR logic)
grep -e "error" -e "warning" -e "critical" logfile.txt

# Patterns from file
echo "error" > patterns.txt
echo "warning" >> patterns.txt
grep -f patterns.txt logfile.txt

# Fixed strings (no regex interpretation)
grep -F "3.14159" data.txt

# Whole word matching
grep -w "test" file.txt  # won't match "testing" or "contest"
```

2.4 Regular Expressions with `grep`

Basic Regular Expressions

```
# Beginning of line anchor
grep "^Error" logfile.txt

# End of line anchor
grep "completed$" logfile.txt

# Any single character
grep "c.t" file.txt  # matches cat, cot, cut, c@t

# Zero or more of previous character
grep "colou*r" file.txt  # matches color, colour, colouur

# Character classes
grep "[0-9]" file.txt      # any digit
grep "[a-zA-Z]" file.txt   # any letter
grep "[aeiou]" file.txt    # any vowel
grep "[^0-9]" file.txt     # any non-digit

# Ranges and specific sets
grep "[A-Z][0-9][0-9]" file.txt  # uppercase letter + 2 digits
grep "[13579]" file.txt           # odd digits only
```

Extended Regular Expressions (-E flag)

```
# One or more of previous character
grep -E "colou+r" file.txt

# Zero or one of previous character (optional)
grep -E "colou?r" file.txt

# Multiple patterns with OR
grep -E "(error|warning|critical)" logfile.txt

# Grouping with parentheses
grep -E "^(http|https|ftp)://" urls.txt

# Repetition quantifiers
grep -E "[0-9]{3}" file.txt     # exactly 3 digits
grep -E "[0-9]{3,5}" file.txt   # 3 to 5 digits
grep -E "[0-9]{3,}" file.txt    # 3 or more digits

# Word boundaries
grep -E "\btest\b" file.txt     # whole word only
```

Practical Regular Expression Examples

```
# IP addresses
grep -E "([0-9]{1,3}\.){3}[0-9]{1,3}" logfile.txt

# Email addresses (basic)
grep -E "[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}" file.txt

# URLs
grep -E "https?://[a-zA-Z0-9./?=_%:-]*" file.txt

# Phone numbers (US format)
grep -E "\([0-9]{3}\) [0-9]{3}-[0-9]{4}" contacts.txt

# Date formats (YYYY-MM-DD)
grep -E "[0-9]{4}-[0-9]{2}-[0-9]{2}" file.txt
```

2.5 Combining `grep` with Other Commands

Pipeline Usage

```
# Grep with process list
ps aux | grep "apache"
ps aux | grep -v grep | grep "mysql"

# Grep with command history
history | grep "ssh"
history | grep -E "(git|svn)"

# Multiple greps in pipeline
grep "error" logfile.txt | grep -v "timeout" | grep "database"

# Grep with find
find . -name "*.log" -exec grep -l "error" {} \;
find . -name "*.txt" | xargs grep "TODO"
```

Data Processing Combinations

```
# Count unique error types
grep "ERROR" logfile.txt | cut -d: -f3 | sort | uniq -c

# Find most common error messages
grep "ERROR" logfile.txt | sort | uniq -c | sort -nr | head -10

# Extract and process IP addresses
grep -oE "([0-9]{1,3}\.){3}[0-9]{1,3}" access.log | sort | uniq -c | sort -nr

# Monitor log files in real-time
tail -f /var/log/syslog | grep --color=always -i "error"
```

2.6 Performance and Efficiency Tips

```
# Use fixed strings when possible (faster)
grep -F "literal.string" hugefile.txt

# Limit search depth with find + grep
find . -maxdepth 2 -name "*.txt" -exec grep -l "pattern" {} \;

# Use -q for silent mode (just exit status)
if grep -q "pattern" file.txt; then
    echo "Pattern found"
fi

# Stop after first match
grep -m 1 "pattern" file.txt
```

Lesson 3: File Compression and Archives with `tar`, `gzip`, and Extraction

Learning Objectives

- Understand the difference between archiving and compression
- Master `tar` for creating, listing, and extracting archives
- Use `gzip`, `bzip2`, and other compression tools effectively
- Handle various archive formats and extraction scenarios
- Implement backup strategies using archives

3.1 Understanding Archives vs Compression

Archiving combines multiple files and directories into a single file, preserving directory structure and metadata. Think of it as putting files in a box.

Compression reduces file size by encoding data more efficiently. Think of it as vacuum-sealing the box to make it smaller.

Archive + Compression = Most efficient storage and transfer method.

3.2 The `tar` Command (Tape ARchiver)

`tar` is the standard archiving tool in Unix/Linux systems. Originally designed for tape backups, it's now used for all types of archives.

Basic `tar` Operations

```
# Create archive (c = create, f = file)
tar -cf archive.tar file1 file2 directory/

# Extract archive (x = extract, f = file)
tar -xf archive.tar

# List contents of archive (t = list, f = file)
tar -tf archive.tar

# Verbose output (v = verbose) - shows what's being processed
tar -cvf archive.tar directory/
tar -xvf archive.tar
tar -tvf archive.tar  # detailed listing like 'ls -l'
```

Essential `tar` Options

- `c` : create new archive
- `x` : extract files from archive
- `t` : list contents of archive
- `f` : specify archive filename (must be followed by filename)
- `v` : verbose output
- `z` : compress/decompress with gzip
- `j` : compress/decompress with bzip2
- `J` : compress/decompress with xz

3.3 Creating Archives with `tar`

Basic Archive Creation

```
# Archive a single directory
tar -cvf backup.tar /home/user/documents/

# Archive multiple items
tar -cvf project.tar file1.txt file2.txt src/ docs/

# Archive with absolute paths preserved
tar -cvf system-backup.tar /etc/ /var/log/

# Archive current directory contents
tar -cvf current-dir.tar .
```

Advanced Archive Options

```
# Exclude specific files or patterns
tar -cvf backup.tar --exclude="*.tmp" --exclude="*.log" project/

# Exclude multiple patterns
tar -cvf backup.tar --exclude="*.tmp" --exclude="node_modules" --exclude=".git"
project/

# Use exclude file
echo "*.tmp" > .exclude
echo "*.log" >> .exclude
echo ".git" >> .exclude
tar -cvf backup.tar -X .exclude project/

# Archive only newer files (incremental)
tar -cvf backup.tar --newer-mtime="2023-01-01" /home/user/

# Archive with compression on-the-fly
tar -czvf backup.tar.gz project/  # gzip compression
tar -cjvf backup.tar.bz2 project/ # bzip2 compression
tar -cJvf backup.tar.xz project/  # xz compression
```

3.4 Extracting Archives

Basic Extraction

```
# Extract to current directory
tar -xvf archive.tar

# Extract to specific directory
tar -xvf archive.tar -C /destination/path/

# Extract specific files only
tar -xvf archive.tar path/to/specific/file.txt
tar -xvf archive.tar "*.txt"  # extract all .txt files

# Extract with original permissions preserved
tar -xpvf archive.tar
```

Smart Extraction (auto-detect compression)

```
# Modern tar can auto-detect compression
tar -xvf archive.tar.gz    # auto-detects gzip
tar -xvf archive.tar.bz2   # auto-detects bzip2
tar -xvf archive.tar.xz    # auto-detects xz

# Explicit compression flags (older systems)
tar -xzvf archive.tar.gz   # gzip
tar -xjvf archive.tar.bz2  # bzip2
tar -xJvf archive.tar.xz   # xz
```

Extraction Verification and Safety

```
# List contents before extracting
tar -tvf archive.tar | head -20

# Test archive integrity
tar -tvf archive.tar > /dev/null && echo "Archive is valid"

# Extract and verify
tar -xvf archive.tar && echo "Extraction completed successfully"

# Extract with progress indicator (for large archives)
tar -xvf largearchive.tar | pv -l > /dev/null
```

3.5 Compression Tools

`gzip` - Most Common Compression

```
# Compress a file (replaces original)
gzip largefile.txt
# Creates largefile.txt.gz

# Compress keeping original
gzip -k largefile.txt

# Decompress
gunzip largefile.txt.gz
# or
gzip -d largefile.txt.gz

# Compression levels (1=fast, 9=best compression)
gzip -1 file.txt  # fast compression
gzip -9 file.txt  # maximum compression
gzip -6 file.txt  # default level

# View compressed file without extracting
zcat file.txt.gz
zless file.txt.gz
zmore file.txt.gz
zgrep "pattern" file.txt.gz
```

`bzip2` - Better Compression Ratio

```
# Compress with bzip2
bzip2 largefile.txt
# Creates largefile.txt.bz2

# Compress keeping original
bzip2 -k largefile.txt

# Decompress
bunzip2 largefile.txt.bz2
# or
bzip2 -d largefile.txt.bz2

# View compressed file
bzcat file.txt.bz2
bzless file.txt.bz2
bzgrep "pattern" file.txt.bz2
```

`xz` - Modern High-Efficiency Compression

```
# Compress with xz (highest compression)
xz largefile.txt
# Creates largefile.txt.xz

# Compress keeping original
xz -k largefile.txt

# Decompress
unxz largefile.txt.xz
# or
xz -d largefile.txt.xz

# View compressed file
xzcat file.txt.xz
xzless file.txt.xz
```

3.6 Working with Different Archive Formats

Common Archive Types and Extraction

```
# TAR archives (uncompressed)
tar -xvf archive.tar

# TAR.GZ (tar + gzip) - most common
tar -xzvf archive.tar.gz
tar -xf archive.tar.gz  # auto-detection

# TAR.BZ2 (tar + bzip2)
tar -xjvf archive.tar.bz2
tar -xf archive.tar.bz2  # auto-detection

# TAR.XZ (tar + xz)
tar -xJvf archive.tar.xz
tar -xf archive.tar.xz   # auto-detection

# ZIP files (if zip/unzip installed)
unzip archive.zip
unzip -l archive.zip     # list contents
unzip archive.zip -d /destination/

# RAR files (if unrar installed)
unrar x archive.rar
unrar l archive.rar      # list contents
```

Creating Different Archive Formats

```
# Standard combinations
tar -czf project.tar.gz project/    # tar + gzip
tar -cjf project.tar.bz2 project/   # tar + bzip2
tar -cJf project.tar.xz project/    # tar + xz

# ZIP archives (cross-platform compatible)
zip -r project.zip project/
zip -r -9 project.zip project/      # maximum compression

# Create archive with date stamp
tar -czf "backup-$(date +%Y%m%d-%H%M%S).tar.gz" project/
```

3.7 Practical Archive Scenarios

System Backup Strategies

```
# Daily home directory backup
tar -czf "/backup/home-backup-$(date +%Y%m%d).tar.gz" \
    --exclude="*.tmp" --exclude="Cache" --exclude=".git" /home/user/

# System configuration backup
sudo tar -czf "/backup/etc-backup-$(date +%Y%m%d).tar.gz" /etc/

# Database backup with compression
mysqldump database_name | gzip > "db-backup-$(date +%Y%m%d).sql.gz"

# Log rotation backup
tar -czf "/backup/logs-$(date +%Y%m%d).tar.gz" --remove-files /var/log/*.log
```

Remote Archive Operations

```
# Extract remote archive directly
curl -s http://example.com/file.tar.gz | tar -xzf -

# Create and send archive remotely
tar -czf - project/ | ssh user@remote 'cat > /backup/project.tar.gz'

# Sync and archive
rsync -av /source/ /backup/ && tar -czf archive.tar.gz /backup/

# Archive over network with progress
tar -czf - largedir/ | pv | ssh user@remote 'cat > backup.tar.gz'
```

Archive Maintenance

```
# Compare archive contents with filesystem
tar -df archive.tar.gz

# Update existing archive (add new files)
tar -uvf archive.tar newfile.txt

# Append to existing archive
tar -rvf archive.tar additionalfile.txt

# Remove files from archive (create new archive)
tar --delete -f archive.tar unwanted.txt
```

Lesson 4: Text Processing with `sed`

Learning Objectives

- Understand stream editing concepts and philosophy
- Master basic `sed` commands for text transformation
- Use advanced `sed` features for complex text processing
- Apply `sed` in real-world scenarios and automation
- Combine `sed` with other tools for powerful text processing pipelines

4.1 Introduction to `sed` (Stream Editor)

`sed` is a powerful stream editor that performs text transformations on input streams (files or pipelines). Unlike interactive text editors, `sed` processes text automatically according to a script

of commands. It's particularly useful for:

- Automated text processing
- Configuration file modifications
- Log file analysis and cleanup
- Data format conversions
- Search and replace operations across multiple files

Philosophy: `sed` reads input line by line, applies commands, and outputs the result. It doesn't modify original files unless explicitly told to do so.

Basic Syntax:

```
sed 'command' file
sed 'address command' file
sed -e 'command1' -e 'command2' file
```

4.2 Basic `sed` Commands

The Substitution Command (s)

The most commonly used `sed` command.

```
# Basic substitution - replace first occurrence per line
sed 's/old/new/' file.txt

# Global substitution - replace all occurrences per line
sed 's/old/new/g' file.txt

# Case-insensitive substitution
sed 's/old/new/gi' file.txt

# Replace only 2nd occurrence per line
sed 's/old/new/2' file.txt

# Replace from 2nd occurrence onward
sed 's/old/new/2g' file.txt
```

Using Different Delimiters

```
# When dealing with paths, use different delimiters
sed 's|/old/path|/new/path|g' file.txt
sed 's#old#new#g' file.txt
sed 's@old@new@g' file.txt

# Useful for URLs
sed 's|http://|https://|g' urls.txt
```

Print Commands (p)

```
# Print specific line (suppress default output with -n)
sed -n '5p' file.txt

# Print range of lines
sed -n '5,10p' file.txt
sed -n '5,$p' file.txt    # from line 5 to end

# Print lines matching pattern
sed -n '/pattern/p' file.txt

# Print line numbers
sed -n '=' file.txt
sed -n '5=' file.txt      # print line number of line 5
```

Delete Commands (d)

```
# Delete specific line
sed '5d' file.txt

# Delete range of lines
sed '5,10d' file.txt
sed '5,$d' file.txt       # from line 5 to end

# Delete lines matching pattern
sed '/pattern/d' file.txt

# Delete empty lines
sed '/^$/d' file.txt

# Delete lines NOT matching pattern
sed '/pattern/!d' file.txt
```

4.3 Addressing in `sed`

Addressing determines which lines `sed` commands operate on.

Line Number Addressing

```
# Single line
sed '3s/old/new/' file.txt

# Range of lines
sed '3,7s/old/new/' file.txt

# From line to end of file
sed '3,$s/old/new/' file.txt

# Every nth line
sed '0~3s/old/new/' file.txt    # every 3rd line starting from 3
```

Pattern Addressing

```
# Lines matching pattern
sed '/ERROR/s/old/new/' file.txt

# Range between patterns
sed '/START/,/END/s/old/new/' file.txt

# From pattern to line number
sed '/pattern/,10s/old/new/' file.txt

# From line number to pattern
sed '5,/pattern/s/old/new/' file.txt
```

Negation (!)

```
# Apply command to lines NOT matching
sed '/pattern/!s/old/new/' file.txt

# Delete all lines except those matching pattern
sed '/pattern/!d' file.txt
```

4.4 Advanced `sed` Operations

Multiple Commands

```
# Multiple commands with -e
sed -e 's/old/new/g' -e '/pattern/d' file.txt

# Multiple commands with semicolons
sed 's/old/new/g; /pattern/d' file.txt

# Multiple commands with newlines
sed '
s/old/new/g
/pattern/d
' file.txt
```

Append (a), Insert (i), and Change (c) Commands

```
# Append line after pattern
sed '/pattern/a\New line to append' file.txt

# Insert line before pattern
sed '/pattern/i\New line to insert' file.txt

# Change/replace entire line
sed '/pattern/c\Replacement line' file.txt

# Append multiple lines
sed '/pattern/a\
Line 1\
Line 2\
Line 3' file.txt
```

Advanced Substitution Features

```
# Reference the matched pattern with &
sed 's/[0-9]*/Number: &/' file.txt

# Use parentheses for grouping (need escaping in basic regex)
sed 's/\([0-9]*\)-\([0-9]*\)/\2-\1/' file.txt

# Multiple groups
sed 's/\([a-zA-Z]*\) \([0-9]*\)/\2 \1/' file.txt

# Case conversion (GNU sed)
sed 's/[a-z]/\U&/g' file.txt    # lowercase to uppercase
sed 's/[A-Z]/\L&/g' file.txt    # uppercase to lowercase
```

Hold Space and Pattern Space

```
# Copy line to hold space
sed 'h' file.txt

# Copy hold space to pattern space
sed 'g' file.txt

# Append pattern space to hold space
sed 'H' file.txt

# Append hold space to pattern space
sed 'G' file.txt

# Exchange pattern and hold spaces
sed 'x' file.txt
```

4.5 Practical `sed` Applications

Configuration File Management

```
# Comment out lines
sed 's/^/#&/' config.conf

# Uncomment lines
sed 's/^#//' config.conf

# Change configuration values
sed 's/^DEBUG=.*/DEBUG=true/' config.conf
sed 's/^PORT=.*/PORT=8080/' config.conf

# Enable/disable services in config
sed '/^#.*service_name/s/^#//' config.conf

# Add configuration if not exists
sed '$a\
new_config=value' config.conf
```

Data Processing and Cleanup

```
# Remove Windows line endings (carriage return)
sed 's/\r$//' file.txt

# Convert spaces to tabs
sed 's/ /\t/g' file.txt

# Convert tabs to spaces (4 spaces)
sed 's/\t/    /g' file.txt

# Remove trailing whitespace
sed 's/[[:space:]]*$//' file.txt

# Remove leading whitespace
sed 's/^[[:space:]]*//' file.txt

# Remove both leading and trailing whitespace
sed 's/^[[:space:]]*\|[[:space:]]*$//g' file.txt
```

Log File Processing

```
# Extract specific time range
sed -n '/2023-12-01 09:00/,/2023-12-01 10:00/p' logfile.log

# Remove debug messages
sed '/DEBUG/d' logfile.log

# Convert log format
sed 's/\[\([^]]*\)\] \([A-Z]*\) \(.*\)/\1|\2|\3/' logfile.log

# Add line numbers to log entries
sed = logfile.log | sed 'N;s/\n/: /'

# Filter and format IP addresses
sed -n 's/.*\([0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\).*/\1/p'
access.log
```

Text Format Conversions

```
# CSV to pipe-separated
sed 's/,/|/g' data.csv

# Add quotes around fields
sed 's/\([^,]*\)/"\1"/g' data.csv

# Remove HTML tags (basic)
sed 's/<[^>]*>//g' file.html

# Convert markdown headers to HTML
sed 's/^# \(.*\)/<h1>\1<\/h1>/' file.md
sed 's/^## \(.*\)/<h2>\1<\/h2>/' file.md

# Format phone numbers
sed 's/\([0-9]\{3\}\)\([0-9]\{3\}\)\([0-9]\{4\}\)/(\1) \2-\3/' contacts.txt
```

4.6 In-Place Editing and Backups

Modifying Files Directly

```
# Edit file in place (GNU sed)
sed -i 's/old/new/g' file.txt

# Edit with backup (creates file.txt.bak)
sed -i.bak 's/old/new/g' file.txt

# Edit with custom backup extension
sed -i.backup-$(date +%Y%m%d) 's/old/new/g' file.txt

# Edit multiple files
sed -i 's/old/new/g' *.txt

# Portable in-place editing (works on macOS/BSD)
sed -i '' 's/old/new/g' file.txt  # macOS
```

4.7 `sed` Scripts and Complex Operations

Using `sed` Script Files

```
# Create a sed script file
cat > cleanup.sed << 'EOF'
# Remove comments
/^#/d
# Remove empty lines
/^$/d
# Convert tabs to spaces
s/\t/    /g
# Remove trailing whitespace
s/[[:space:]]*$//
EOF

# Run the script
sed -f cleanup.sed input.txt
```

Complex Multi-Line Operations

```
# Join lines ending with backslash
sed ':a;/\\$/N;s/\\\n//;ta' file.txt

# Print paragraph containing pattern
sed -n '/pattern/{:a;N;/^$/!ba;p;}' file.txt

# Remove C-style comments (/* ... */)
sed ':a;s|/\*[^*]*\*/||g;t;s|/\*
[^*]*$||;t;:b;n;s|^[^*]*\*/||;t;s|^[^/]*$||;t;bb;ba' file.c

# Number non-empty lines
sed '/./=' file.txt | sed '/./N; s/\n/: /'
```

4.8 Combining `sed` with Other Tools

Pipeline Examples

```
# Count occurrences after substitution
sed 's/old/new/g' file.txt | grep -c "new"

# Process and sort
sed 's/^[[:space:]]*//' file.txt | sort | uniq

# Extract and process data
sed -n 's/.*Error: \(.*\)/\1/p' logfile.log | sort | uniq -c | sort -nr

# Combine with awk for advanced processing
sed 's/,/ /g' data.csv | awk '{print $1, $3}'

# Use with find for bulk operations
find . -name "*.txt" -exec sed -i 's/old/new/g' {} \;
```

Real-World Automation Examples

```
# Update copyright year in all source files
find . -name "*.c" -o -name "*.h" | xargs sed -i 's/Copyright 2023/Copyright
2024/g'

# Convert DOS line endings to Unix
find . -name "*.txt" -exec sed -i 's/\r$//' {} \;

# Batch rename references in code
sed -i 's/oldFunctionName/newFunctionName/g' *.c *.h

# Clean up log files
sed -i '/DEBUG/d; /^$/d; s/[[:space:]]*$//' *.log
```

Lesson 5: Advanced Text Processing with `awk`

Learning Objectives

- Understand `awk` as a pattern-action programming language
- Master field processing and built-in variables
- Use `awk` for data extraction, calculation, and reporting
- Write complex `awk` programs for real-world data processing
- Combine `awk` with other command-line tools effectively

5.1 Introduction to `awk`

`awk` is a powerful pattern-scanning and data extraction language. Unlike `grep` (which finds patterns) and `sed` (which edits streams), `awk` is a complete programming language designed for text processing and report generation.

Key Concepts:

- **Records**: Lines of input (by default)
- **Fields**: Parts of each record separated by delimiters (whitespace by default)
- **Pattern-Action**: `awk` programs consist of patterns and actions
- **Built-in Variables**: Automatic variables for field processing

Basic Syntax:

```
awk 'pattern { action }' file
awk 'BEGIN { action } pattern { action } END { action }' file
```

5.2 Basic `awk` Structure and Variables

Built-in Variables

```
# $0 = entire record (line)
# $1, $2, $3... = individual fields
# NR = Number of Records (line number)
# NF = Number of Fields in current record
# FS = Field Separator (default: whitespace)
# OFS = Output Field Separator (default: space)
# RS = Record Separator (default: newline)
# ORS = Output Record Separator (default: newline)

# Print entire line
awk '{ print $0 }' file.txt

# Print first field
awk '{ print $1 }' file.txt

# Print multiple fields
awk '{ print $1, $3 }' file.txt

# Print with custom separator
awk '{ print $1 ":" $2 }' file.txt
```

Field and Record Information

```
# Print line number and content
awk '{ print NR, $0 }' file.txt

# Print number of fields per line
awk '{ print NF }' file.txt

# Print last field of each line
awk '{ print $NF }' file.txt

# Print second-to-last field
awk '{ print $(NF-1) }' file.txt
```

5.3 Field Separators and Data Processing

Custom Field Separators

```
# Use comma as field separator (CSV)
awk -F',' '{ print $1, $3 }' data.csv

# Use colon as separator (like /etc/passwd)
awk -F':' '{ print $1, $3 }' /etc/passwd

# Use multiple characters as separator
awk -F'::' '{ print $1, $2 }' file.txt

# Use regular expression as separator
awk -F'[,:]' '{ print $1, $2 }' file.txt

# Set field separator within program
awk 'BEGIN { FS="," } { print $1, $3 }' data.csv
```

Output Field Separation

```
# Change output field separator
awk 'BEGIN { OFS="|" } { print $1, $2, $3 }' file.txt

# Custom formatting
awk '{ printf "Name: %s, Age: %s\n", $1, $2 }' people.txt

# Tab-separated output
awk 'BEGIN { OFS="\t" } { print $1, $2, $3 }' file.txt
```

5.4 Pattern Matching in `awk`

Simple Patterns

```
# Print lines containing pattern
awk '/pattern/ { print }' file.txt
awk '/ERROR/' file.log

# Pattern with specific field
awk '$3 ~ /pattern/ { print }' file.txt

# Multiple patterns
awk '/pattern1/ || /pattern2/ { print }' file.txt

# Pattern negation
awk '!/pattern/ { print }' file.txt
awk '$1 !~ /pattern/ { print }' file.txt
```

Numeric and String Comparisons

```
# Numeric comparisons
awk '$3 > 100 { print }' numbers.txt
awk '$2 >= 50 && $2 <= 100 { print }' data.txt

# String comparisons
awk '$1 == "root" { print }' /etc/passwd
awk '$4 != "active" { print }' status.txt

# String matching with operators
awk '$1 ~ /^[Aa]/ { print }' names.txt    # starts with A or a
awk '$NF ~ /\.txt$/ { print }' files.txt  # ends with .txt
```

Range Patterns

```
# Print between patterns
awk '/START/,/END/ { print }' file.txt

# Print from line number to pattern
awk 'NR==10,/pattern/ { print }' file.txt

# Print specific line range
awk 'NR>=10 && NR<=20 { print }' file.txt
```

String Functions

```
# String length
awk '{ print length($1) }' file.txt

# Substring
awk '{ print substr($1, 2, 3) }' file.txt  # from position 2, length 3

# String position
awk '{ print index($1, "pattern") }' file.txt

# Case conversion
awk '{ print toupper($1), tolower($2) }' file.txt

# String substitution
awk '{ gsub(/old/, "new", $1); print }' file.txt

# Split string into array
awk '{ n=split($0, words, " ");
       for (i=1; i<=n; i++) print i, words[i] }' file.txt
```

Combining `awk` with Other Tools

Pipeline Examples

```
# Combine with sort and uniq
awk '{ print $1 }' file.txt | sort | uniq -c | sort -nr

# Process find results
find . -name "*.txt" -exec wc -l {} \; | awk '{ total += $1 } END { print total
}'

# Combine with grep
grep "ERROR" logfile.log | awk '{ print $1, $2, $NF }'

# Use with curl for web data processing
curl -s "http://api.example.com/data.csv" | awk -F',' '{ print $1, $3 }'

# Complex pipeline
ps aux | awk 'NR>1 { print $1, $3 }' | sort -k2 -nr | head -5
```