Notes 9 - Handling Text Files 2

grep

Definition:

Searches for patterns in files and prints matching lines. The name stands for Global Regular Expression Print.

- Usage: grep [OPTIONS] PATTERN [FILE...]
- Common Options:
- -i : Case-insensitive search
- -v : Invert match (show non-matching lines)
- -n : Show line numbers
- -c : Count matching lines only
- -r or -R : Recursive search in directories
- -l : Show only filenames with matches
- -E : Extended regular expressions (or use egrep)
- -w : Match whole words only
- -A NUM : Show NUM lines after match
- -B NUM : Show NUM lines before match
- -C NUM : Show NUM lines before and after match

Examples:

Basic search for a word

grep "error" logfile.txt

Case-insensitive search

grep -i "ERROR" logfile.txt

Search with line numbers

grep -n "warning" logfile.txt

Count occurrences

```
grep -c "success" logfile.txt
```

Search for whole words only

```
grep -w "test" file.txt # Matches "test" but not "testing" or "contest"
```

Recursive search in directory

```
grep -r "TODO" /home/user/project/
```

Show lines before and after match

```
grep -C 2 "critical" logfile.txt
```

Search with extended regex

```
grep -E "(error|warning|critical)" logfile.txt
```

Inverse search (lines NOT containing pattern)

```
grep -v "debug" logfile.txt
```

Search in compressed files

```
zgrep "pattern" *.gz
```

awk

Definition:

A powerful programming language for text processing and data extraction. Named after its creators: Aho, Weinberger, and Kernighan.

- Usage: awk [OPTIONS] 'PROGRAM' [FILE...]
- or awk [OPTIONS] -f PROGRAM_FILE [FILE...]
- Basic Structure: awk 'pattern { action }' file

Built-in Variables:

- $0 : Entire line

- $1, $2, ... : Field 1, field 2, etc.
- NF : Number of fields in current line
- NR : Current line number
- FS : Field separator (default: whitespace)
- OFS : Output field separator (default: space)

Common Options:

- -F : Set input field separator
- -v : Assign a variable value
- -f : Read program from file

Examples:

Print first field of each line (whitespace separated)

awk '{print $1}' file.txt

Print specific fields

awk '{print $1, $3, $NF}' file.txt # First, third, and last field

Use custom field separator (comma for CSV)

awk -F ',' '{print $1, $3}' data.csv

Print lines where condition is true

awk '$3 > 100 {print $0}' data.txt # Lines where 3rd field > 100

Print line numbers with content

awk '{print NR, $0}' file.txt

Calculate sum of second column

awk '{sum += $2} END {print "Total:", sum}' data.txt

Count lines with pattern

awk '/error/ {count++} END {print count}' logfile.txt

Multiple commands

awk '{print NR ": " $0; total += $3} END {print "Sum:", total}' data.txt

Using BEGIN for initialization

awk 'BEGIN {FS=":"; OFS="\t"} {print $1, $3}' /etc/passwd

sed

Definition:

Stream EDitor for filtering and transforming text.

*Usage: sed [OPTIONS] 'COMMAND' [FILE...]

Common Options:

- -n : Suppress automatic printing
- -i : Edit files in-place (with backup: -i.bak)
- -e : Add multiple commands
- -r or -E : Use extended regular expressions

Common Commands:

- s/pattern/replacement/ : Substitute
- p : Print
- d : Delete
- a\text : Append text after line
- i\text : Insert text before line
- y/abc/ABC/ : Translate characters

Examples:

Simple substitution (first occurrence per line)

sed 's/old/new/' file.txt

Global substitution (all occurrences)

```
sed 's/old/new/g' file.txt
```

Delete lines containing pattern

```
sed '/pattern/d' file.txt
```

Delete lines 1-5

```
sed '1,5d' file.txt
```

Print only lines matching pattern (with -n)

```
sed -n '/pattern/p' file.txt
```

In-place editing with backup

```
sed -i.bak 's/old/new/g' file.txt
```

Multiple commands

```
sed -e 's/foo/bar/g' -e 's/baz/qux/g' file.txt
```

Insert text before line 5

```
sed '5i\This is inserted text' file.txt
```

Append text after line containing pattern

```
sed '/pattern/a\This is appended text' file.txt
```

Change line 10 specifically

```
sed '10c\This is new line 10' file.txt
```

Extract specific lines (10-20)

```
sed -n '10,20p' file.txt
```

Transform characters (like tr command)

sed 'y/abc/ABC/' file.txt

Redirection and Piping

The Pipe (|) Operator

- Definition: The pipe (|) redirects the output (stdout) of one command to become the input (stdin) of another command. It allows chaining multiple commands together.
- Basic Syntax: command1 | command2 | command3 ...

Examples:

Example 1: Search for a pattern in the output of another command

ls -la | grep ".txt" # List files and filter for .txt files

Example 2: Count lines in filtered output

ps aux | grep "chrome" | wc -l # Count how many Chrome processes are running

Example 3: Sort and paginate output

cat access.log | cut -d' ' -f1 | sort | uniq -c | sort -nr | head -10

- This pipeline:

- 1. Reads log file
- 2. Extracts first field (IP addresses)
- 3. Sorts IPs
- 4. Counts unique IPs
- 5. Sorts by count (descending)
- 6. Shows top 10

- Example 4: Process substitution with tee ls -l | tee filelist.txt | grep "Dec"

- Saves output to file AND pipes to grep

Output Redirection to File (>)

- Definition: The > operator redirects command output (stdout) to a file, overwriting the file if it exists.
- Basic Syntax: command > filename

Examples:

- Example 1: Save directory listing to file ls -la > directory_listing.txt

- Example 2: Save command output for later use date > current_date.txt whoami > current_user.txt
- Example 3: Save filtered output grep "ERROR" /var/log/syslog > errors_only.log
- Example 4: Save to multiple locations (with tee) ls -l | tee listing.txt > listing_copy.txt
- Example 5: Discard output (send to /dev/null) noisy_command > /dev/null # Silences output

Append Output to File (>>)

- Definition: The >> operator redirects command output (stdout) to a file, appending to the end of the file if it exists.
- Basic Syntax: command >> filename

Examples:

- Example 1: Append to log file echo "Process started at $(date)" >> process.log
- ... some time later ... echo "Process completed at $(date)" >> process.log
- Example 2: Build a file incrementally echo "Header: Report" >> report.txt echo "==============" >> report.txt ls -la >> report.txt
- Example 3: Collect results from multiple runs for i in {1..5}; do echo "Run $i: $(some_command)" >> results.txt done
- Example 4: Append error output (stderr) command 2>> error_log.txt # Appends stderr to file
- Example 5: Create a simple log rotation echo "$(date): User $(whoami) logged in" >> /var/log/myapp.log