Table of Contents

$ awk

Advanced

Pattern scanning and text processing language

awk is a powerful text processing language and command for pattern scanning, field extraction, and data transformation. It processes input line by line, splitting each line into fields, making it ideal for working with structured text like CSV files, log files, and command output.

awk programs c...

Options & Flags

`-F`	Set field separator (default: whitespace)
`-v`	Assign a variable before execution
`-f`	Read awk program from a file
`BEGIN`	Execute block before processing any input
`END`	Execute block after all input is processed
`NR`	Built-in variable: current record (line) number
`NF`	Built-in variable: number of fields in current record
`-OFS`	Output field separator

Practical Examples

Example: Print specific columns

```
$ awk '{print $1, $3}' data.txt
```

Extracts the first and third fields from each line.

Example: Process CSV files

```
$ awk -F',' '{print $2, $4}' users.csv
```

Uses comma as field separator to extract columns from a CSV file.

Example: Sum a column of numbers

```
$ awk '{sum += $1} END {print "Total:", sum}' sales.txt
Total: 15420
```

Adds up all values in the first column and prints the total.

Example: Filter lines by condition

```
$ awk '$3 > 1000' transactions.txt
```

Shows only lines where the third field exceeds 1000.

Example: Count lines matching a pattern

```
$ awk '/ERROR/ {count++} END {print count}' app.log
42
```

Counts lines containing ERROR in the log file.

Tips & Best Practices

Pro Tip: Print the last field: Use $NF to always get the last field regardless of how many fields exist: awk '{print $NF}' file.txt

Note: awk vs cut: For simple column extraction, cut is simpler and faster. Use awk when you need conditions, calculations, or complex formatting.

Warning: Quote carefully: Wrap awk programs in single quotes to prevent the shell from interpreting $1, $2, etc. as shell variables instead of awk field variables.

$ cat

Beginner

Concatenate and display file contents

The cat command (short for concatenate) reads, displays, and combines files. It is one of the most used commands for viewing file contents, creating files, and concatenating multiple files together.

cat reads files sequentially and writes their contents to standard output. When used with redirec...

Options & Flags

`-n`	Number all output lines
`-b`	Number only non-empty lines
`-s`	Squeeze multiple blank lines into one
`-A`	Show all non-printing characters (tabs as ^I, line ends a...
`-E`	Display $ at end of each line
`-T`	Display TAB characters as ^I
`-v`	Display non-printing characters

Practical Examples

Example: Display file contents

```
$ cat /etc/hostname
webserver01
```

Displays the contents of the hostname file.

Example: Concatenate multiple files

```
$ cat header.txt body.txt footer.txt > complete.txt
```

Combines three files into one new file.

Example: Display with line numbers

```
$ cat -n script.sh
     1 #!/bin/bash
     2 echo "Hello"
     3
```

Shows the file with numbered lines - useful for referencing specific line numbers.

Example: Create a file with content

```
$ cat > notes.txt << EOF\nLine 1\nLine 2\nEOF
```

Creates a file using a here-document. Type content and end with EOF.

Example: Append to an existing file

```
$ cat >> logfile.txt << EOF\nNew entry\nEOF
```

Appends content to the end of an existing file using >>.

Tips & Best Practices

Warning: Avoid useless use of cat: Instead of cat file | grep pattern, use grep pattern file directly. This avoids an unnecessary process (known as "UUOC" - Useless Use of Cat).

Pro Tip: Use tac for reverse: The tac command (cat reversed) displays files with the last line first. Great for viewing recent log entries: tac /var/log/syslog | head -20

Note: bat as a cat alternative: The bat command is a modern cat replacement with syntax highlighting, line numbers, and git integration built in.

$ cmp

Beginner

Compare two files byte by byte

The cmp command compares two files byte by byte. Unlike diff which compares text line by line, cmp works at the byte level, making it suitable for both text and binary file comparison.

cmp reports the first byte position and line number where the files differ. It is much faster than diff for sim...

Options & Flags

`-l`	Print all differing bytes with positions
`-s`	Silent mode - exit status only, no output
`-i`	Skip first N bytes
`-n`	Compare at most N bytes
`-b`	Print differing bytes
`exit codes`	0=identical, 1=different, 2=error

Practical Examples

Example: Compare two files

```
$ cmp original.bin backup.bin
original.bin backup.bin differ: byte 4521, line 23
```

Reports the first byte where the files differ, or nothing if identical.

Example: Silent comparison in script

```
$ if cmp -s config.yml config.yml.bak; then echo "No changes"; else echo "Modified"; fi
```

Uses exit code to check if files are identical without any output.

Example: List all differences

```
$ cmp -l file1.bin file2.bin
4521 141 142\n4522 157 160
```

Shows every byte position where the files differ with octal values.

Example: Compare download integrity

```
$ cmp -s downloaded.iso original.iso && echo "OK" || echo "CORRUPT"
```

Verifies a downloaded file matches the original.

Example: Compare first N bytes

```
$ cmp -n 512 disk1.img disk2.img
```

Compares only the first 512 bytes (useful for checking boot sectors).

Tips & Best Practices

Pro Tip: Use -s in scripts: cmp -s is the fastest way to check if two files are identical in shell scripts. It stops at the first difference and produces no output.

Note: cmp vs diff: Use cmp for binary files and simple identity checks. Use diff for text files where you need to see what changed.

Warning: Large files: For very large files, cmp -s is much faster than diff because it only needs to find the first difference, not all differences.

$ column

Beginner

Format text into columns for clean output

The column command formats text into aligned columns for readable tabular display. It automatically determines column widths based on content, producing neatly formatted output.

column is especially useful for formatting command output, CSV files, and any delimited data into human-readable table...

Options & Flags

`-t`	Create a table with aligned columns
`-s`	Set input delimiter (for -t mode)
`-n`	Do not merge multiple delimiters
`-o`	Set output delimiter
`-c`	Set output width for filling columns
`-J`	Output as JSON

Practical Examples

Example: Format mount output

```
$ mount | column -t
```

Aligns the mount command output into neatly formatted columns.

Example: Display CSV as table

```
$ column -t -s',' data.csv
Name    Age  City
Alice   30   London
Bob     25   Paris
```

Formats comma-separated data into an aligned table.

Example: Format /etc/fstab readably

```
$ column -t /etc/fstab
```

Aligns the filesystem table for easier reading.

Example: Display list in columns

```
$ ls /usr/bin | column
```

Arranges a long list into multiple columns that fit the terminal width.

Example: Format pipe-delimited data

```
$ column -t -s'|' -o' | ' report.txt
```

Reads pipe-delimited input and outputs an aligned table with pipe separators.

Tips & Best Practices

Pro Tip: Quick table formatting: Pipe any delimiter-separated output through column -t for instant readable tables: cat /etc/passwd | head -5 | column -t -s':'

Note: JSON output: Modern versions of column support -J for JSON output, which is useful for scripting and data processing.

Warning: Wide output: column fills the terminal width. For very wide data, pipe through less -S to enable horizontal scrolling.

$ comm

Intermediate

Compare two sorted files line by line

The comm command compares two sorted files line by line and produces three columns of output: lines unique to file 1, lines unique to file 2, and lines common to both files.

comm is the Unix equivalent of set operations: the three columns represent set difference (A-B), set difference (B-A), and...

Options & Flags

`-1`	Suppress column 1 (lines unique to file 1)
`-2`	Suppress column 2 (lines unique to file 2)
`-3`	Suppress column 3 (lines common to both)
`-12`	Show only common lines (intersection)
`-23`	Show lines only in file 1 (difference)
`-13`	Show lines only in file 2 (difference)

Practical Examples

Example: Compare two sorted lists

```
$ comm list_old.txt list_new.txt
```

Shows three columns: old-only, new-only, and common items.

Example: Find common lines

```
$ comm -12 users_server1.txt users_server2.txt
```

Shows only lines present in both files (set intersection).

Example: Find lines only in file 1

```
$ comm -23 old_packages.txt new_packages.txt
```

Shows packages that were in old but removed in new (set difference).

Example: Compare unsorted files

```
$ comm -12 <(sort ips_today.txt) <(sort ips_yesterday.txt)
```

Uses process substitution to sort files before comparing.

Example: Find new entries

```
$ comm -13 <(sort baseline.txt) <(sort current.txt)
```

Shows lines in current.txt that are not in baseline.txt - useful for finding new items.

Tips & Best Practices

Warning: Files must be sorted: comm produces incorrect results on unsorted input. Always sort first: comm <(sort file1) <(sort file2).

Pro Tip: Set operations: Think of comm flags as set operations: -12 = intersection, -23 = A minus B, -13 = B minus A, -3 = symmetric difference.

Note: comm vs diff: comm shows set-based differences (what is in A, B, or both). diff shows how to transform one file into another. Use comm for list comparison.

$ cut

Beginner

Remove sections from each line of files

The cut command extracts specific sections from each line of input - by character position, byte position, or delimited field number. It is one of the simplest and fastest text extraction tools in Linux.

cut is ideal for extracting columns from structured text like CSVs, TSVs, /etc/passwd, and c...

Options & Flags

`-d`	Set field delimiter (default: TAB)
`-f`	Select fields by number
`-c`	Select characters by position
`-b`	Select bytes by position
`--complement`	Output everything except selected fields
`--output-delimiter`	Set output delimiter (can differ from input)

Practical Examples

Example: Extract username and shell from /etc/passwd

```
$ cut -d':' -f1,7 /etc/passwd
root:/bin/bash
www-data:/usr/sbin/nologin
```

Shows username (field 1) and login shell (field 7) from the password file.

Example: Extract CSV columns

```
$ cut -d',' -f2,4 employees.csv
```

Extracts the 2nd and 4th columns from a CSV file.

Example: Extract first 20 characters

```
$ cut -c1-20 longlines.txt
```

Displays only the first 20 characters of each line.

Example: Extract IP addresses from log

```
$ cut -d' ' -f1 access.log | sort -u
```

Extracts the first field (IP address) and shows unique values.

Example: Remove a specific column

```
$ cut -d',' --complement -f3 data.csv
```

Outputs all columns except the 3rd one.

Tips & Best Practices

Note: Single-character delimiter only: cut only supports single-character delimiters. For multi-character delimiters or regex-based splitting, use awk instead.

Pro Tip: Field ranges: Use ranges with -f: -f1-3 (fields 1 to 3), -f3- (field 3 onward), -f-5 (fields 1 to 5).

Warning: Spaces in CSV: If CSV fields contain the delimiter character, cut will split incorrectly. Use a proper CSV parser (awk or csvtool) for complex CSV files.

$ diff

Intermediate

Compare files line by line

The diff command compares two files line by line and outputs the differences between them. It is essential for code review, configuration management, patch creation, and troubleshooting.

diff supports several output formats: normal (default), unified (-u, most commonly used), context (-c), and s...

Options & Flags

`-u`	Unified diff format (most readable, standard for patches)
`-r`	Recursively compare directories
`-y`	Side-by-side output
`-q`	Report only whether files differ
`-i`	Ignore case differences
`-w`	Ignore all whitespace
`-B`	Ignore blank line changes
`--color`	Colorize output (GNU diff)

Practical Examples

Example: Unified diff of two files

```
$ diff -u config.old config.new
--- config.old
+++ config.new
@@ -1,3 +1,3 @@
-port=80
```

Shows differences in unified format with + for additions and - for removals.

Example: Compare directories

```
$ diff -rq project-v1/ project-v2/
```

Recursively compares two directory trees, reporting only which files differ.

Example: Side-by-side comparison

```
$ diff -y --width=120 file1.txt file2.txt
```

Shows both files side by side with differences marked.

Example: Create a patch file

```
$ diff -u original.c modified.c > fix.patch
```

Creates a patch file that can be applied with the patch command.

Example: Apply a patch

```
$ patch < fix.patch
```

Applies the differences from a patch file to transform the original.

Tips & Best Practices

Pro Tip: Use colordiff for better visibility: Install colordiff for automatic colorization: diff -u file1 file2 | colordiff. Or pipe to less -R for paged colored output.

Note: Exit codes: diff returns 0 if files are identical, 1 if they differ, and 2 on error. Useful in scripts: if diff -q file1 file2 > /dev/null; then echo "Same"; fi

Warning: Binary files: diff is designed for text files. For binary files, use cmp for byte-level comparison or xxd for hex dump comparison.

$ envsubst

Beginner

Substitute environment variables in text

The envsubst command substitutes environment variable references in text with their values. It reads from stdin (or files) and replaces $VARIABLE and ${VARIABLE} patterns with the corresponding environment variable values.

envsubst is essential for configuration management and container deployme...

Options & Flags

`(basic usage)`	Substitute all environment variables
`' '`	Substitute only specified variables
`-v`	Print version and exit
`-V`	List all recognized variable references

Practical Examples

Example: Template a config file

```
$ export DB_HOST=10.0.0.5 DB_PORT=5432 DB_NAME=myapp && envsubst < db.conf.template > db.conf
```

Replace $DB_HOST, $DB_PORT, $DB_NAME in template with actual values.

Example: Nginx config template

```
$ export SERVER_NAME=example.com BACKEND_PORT=3000 && envsubst '$SERVER_NAME $BACKEND_PORT' < /etc/nginx/templates/default.conf.ter
```

Generate Nginx config from template. Only substitute specified variables (leave other $ signs untouched).

Example: Docker entrypoint pattern

```
$ #!/bin/sh\nenvsubst < /app/config.template.js > /app/config.js\nexec "$@"
```

Common Docker entrypoint pattern: generate config from template using runtime env vars, then start the app.

Example: Inline template

```
$ export NAME=World && echo 'Hello $NAME, today is $TODAY' | envsubst
Hello World, today is
```

Use with echo for simple templating. Note: single quotes prevent shell expansion, envsubst does the substitution.

Example: Selective substitution

```
$ export APP_PORT=8080 && envsubst '$APP_PORT' < template.yml > output.yml
```

Only replace $APP_PORT, leaving other $variables unchanged. Essential when template contains literal $ signs.

Tips & Best Practices

Pro Tip: Specify variables to substitute: Always specify which variables to substitute when your template contains literal $ signs: envsubst '$VAR1 $VAR2' < template. Otherwise ALL $ references are replaced.

Warning: Missing variables become empty: Undefined environment variables are replaced with empty strings. Use set -u in scripts to catch missing variables, or validate required vars before envsubst.

Note: Part of gettext: envsubst is part of the gettext package. Install: apt install gettext (Debian/Ubuntu), dnf install gettext (Fedora/RHEL).

Pro Tip: Docker best practice: In Dockerfiles, copy templates and use envsubst in the entrypoint script. This keeps images environment-agnostic - the same image works in dev, staging, and production.

$ grep

Beginner

Search text using patterns (regular expressions)

The grep command searches for text patterns in files and input streams using regular expressions. It is one of the most powerful and frequently used text processing tools in Linux, essential for log analysis, code searching, and data filtering.

grep supports basic regular expressions (BRE) by de...

Options & Flags

`-i`	Case-insensitive search
`-r, -R`	Search recursively through directories
`-n`	Show line numbers with matches
`-v`	Invert match - show lines that do NOT match
`-c`	Count the number of matching lines
`-l`	Show only filenames containing matches
`-E`	Use extended regular expressions (same as egrep)
`-w`	Match whole words only
`-A n`	Show n lines after each match
`-B n`	Show n lines before each match
`-C n`	Show n lines of context around each match
`--include`	Search only files matching a pattern

Practical Examples

Example: Search for a string in a file

```
$ grep "error" /var/log/syslog
Mar 14 09:23:01 server error: connection refused
```

Finds all lines containing "error" in the syslog file.

Example: Case-insensitive recursive search

```
$ grep -ri "password" /etc/
```

Searches all files under /etc/ for "password" regardless of case.

Example: Find lines NOT matching a pattern

```
$ grep -v "^#" /etc/ssh/sshd_config | grep -v "^$"
```

Shows active configuration lines by removing comments and empty lines.

Example: Count error occurrences

```
$ grep -c "404" access.log
247
```

Counts how many 404 errors appear in the access log.

Example: Search with context

```
$ grep -B 2 -A 5 "Exception" /var/log/app.log
```

Shows 2 lines before and 5 lines after each Exception match - essential for debugging.

Tips & Best Practices

Pro Tip: Use --color for highlighted matches: Add --color=auto (or alias grep="grep --color=auto" in .bashrc) to highlight matching text in the output.

Note: grep vs ripgrep: For large codebases, consider ripgrep (rg) which is significantly faster and respects .gitignore by default. Syntax is very similar to grep.

Warning: Avoid grep grep in ps output: When using ps aux | grep process, the grep command itself shows up. Use ps aux | grep [p]rocess or pgrep instead.

$ head

Beginner

Output the first part of files

The head command outputs the first part of files. By default, it prints the first 10 lines, but this can be customized. It is essential for quickly previewing file contents, examining log files, and processing data streams.

head works with both files and piped input, making it versatile in shell...

Options & Flags

`-n`	Output the first N lines (default 10)
`-c`	Output the first N bytes
`-n -N`	Output all lines except the last N
`-q`	Quiet mode - never print headers (for multiple files)
`-v`	Always print headers with filenames
`N (shorthand)`	Short form for -n N

Practical Examples

Example: View first 10 lines (default)

```
$ head /var/log/syslog
```

Shows the first 10 lines of the syslog file.

Example: View first N lines

```
$ head -n 50 access.log
```

Shows the first 50 lines of the access log.

Example: View first N bytes

```
$ head -c 256 /dev/urandom | base64
```

Reads the first 256 bytes from urandom and converts to base64.

Example: Preview multiple files

```
$ head -n 3 *.conf
==> nginx.conf <==
worker_processes auto;
```

Shows the first 3 lines of each .conf file with headers.

Example: Exclude last N lines

```
$ head -n -2 data.csv
```

Shows everything except the last 2 lines. Useful for removing footers.

Tips & Best Practices

Pro Tip: Combine with tail for range: To extract lines 100-110: head -n 110 file | tail -n 11. Or use sed -n "100,110p" file for the same result.

Note: head vs less: head is better for scripts and pipelines (non-interactive). less is better for interactive browsing with search and scrolling.

Warning: Binary files: Using head on binary files can mess up your terminal. If that happens, run the reset command to restore your terminal.