awk Column and text processing

START HERE

`awk '{print $1}' <file>`
Print first column

`awk -F: '{print $1}' <file>`
Set field separator

`awk '/<regex>/' <file>`
Lines matching regex

`awk '$3 > 10' <file>`
Filter on a field

`awk '{s+=$2} END {print s}' <file>`
Sum a column

`awk 'NR==1' <file>`
Print line 1

`awk -v n=<val> '$1==n' <file>`
Pass a variable in

`awk '{print NF}' <file>`
Fields per line

BEGIN once, before input → `each line` → **Split line** $0, $1 … $NF → `pattern true?` → **Action** { … } per record → `after last line` → **END** once, totals

CONCEPTS

A program is pattern { action } pairs. awk reads each line, splits it into $1, $2 …, and runs the action when the pattern is true. No pattern means every line; no action means print.

Single-quote the program so the shell leaves $1 and braces alone. Features marked gawk (FPAT, gensub, asort, BEGINFILE, -i) are missing in mawk and BSD awk.

RUNNING

`awk '<prog>' <file>`
Run a program on a file

`awk -f <script.awk> <file>`
Load the program from a file

`awk -F<sep> '<prog>'`
Set the input field separator
A single space is the default and splits on runs of blanks; -F, splits on every comma; -F'\t' on tabs.

`awk -v <var>=<val> '<prog>'`
Set a variable before BEGIN
Use this instead of splicing shell variables into the program text.

`<cmd> | awk '<prog>'`
Process piped input

`awk -F, -v OFS= '<prog>'`
Set output separator

`awk -i inplace '<prog>' <file>`
Edit file in place (gawk)
Needs gawk 4.1+; otherwise write to a temp file and mv it.

`awk --csv '<prog>' <file>`
Parse real CSV with quotes (gawk)
New in gawk 5.3; before that use FPAT.

FIELDS & VARIABLES

`$0`
Whole current line

`$1, $2, $NF`
Field by number; last field
$(NF-1) is the second to last; $NF needs no parentheses.

`NR`
Line number across all files

`NF`
Number of fields on this line

`FNR`
Line number within current file
Compare NR==FNR to detect the first of two files.

FIELDS & VARIABLES (cont.)

`FS / OFS`
Input / output field separator

`RS / ORS`
Input / output record separator
RS="" reads blank-line-separated paragraphs.

`FILENAME`
Name of the current input file

`$1=$1`
Rebuild line with OFS
Assigning any field makes awk re-join $0 using OFS.

`ENVIRON["<name>"]`
Read an environment variable

`ARGV[<n>] / ARGC`
Command-line arguments

PATTERNS

`BEGIN { ... }`
Run once before reading input

`END { ... }`
Run once after the last line

`/<regex>/`
Lines matching a regex

`!/<regex>/`
Lines not matching

`$<n> ~ /<regex>/`
Field matches regex

`$<n> !~ /<regex>/`
Field does not match

`<expr> && <expr>`
Combine conditions (also || and !)

`NR==<a>, NR==<b>`
Range between two conditions
Starts at the first match of the left side, ends at the next match of the right side.

`NR % 2`
Every other line

`BEGINFILE { ... }`
Run at the start of each file (gawk)

ACTIONS & CONTROL

`print $1, $3`
Print fields joined by OFS

`{ s += $2 }`
Accumulate in a variable
Variables start empty, which counts as 0 in arithmetic.

`if (<cond>) ... else ...`
Conditional

ACTIONS & CONTROL (cont.)

`for (i = 1; i <= NF; i++)`
Loop over fields

`for (<key> in <arr>)`
Loop over array keys
Order is unspecified.

`while (<cond>) { ... }`
Loop while true

`next`
Skip to the next line

`exit [<code>]`
Stop; END still runs

`getline <var> < <file>`
Read a line from a file

`system("<cmd>")`
Run a shell command

FUNCTIONS

`length(<s>)`
String length (no arg: $0)

`substr(<s>, <start>, <len>)`
Substring, 1-based

`index(<s>, <t>)`
Position of t in s, 0 if none

`split(<s>, <arr>, <sep>)`
Split into array; returns count

`sub(/<re>/, "<new>" [, <t>])`
Replace first match
Works on $0 unless a target is given.

`gsub(/<re>/, "<new>" [, <t>])`
Replace every match
In the replacement, & stands for the matched text.

`match(<s>, /<re>/)`
Find regex; sets RSTART, RLENGTH

`tolower(<s>) / toupper(<s>)`
Change case

`sprintf("<fmt>", ...)`
Format into a string

`int(<x>), sqrt(<x>), rand()`
Basic math

`gensub(/<re>/, "<new>", "g", <s>)`
Replace with \1 groups (gawk)

`asort(<arr>) / asorti(<arr>)`
Sort values / indices (gawk)

`systime() / strftime("<fmt>")`
Epoch time and formatting (gawk)

OUTPUT

`printf "%s\n", $1`
Formatted output; no auto newline

`printf "%-10s %5.2f\n", $1, $2`
Pad text, round numbers
%d integer, %s string, %f float; a minus sign left-aligns.

`print > "<file>"`
Write to a file
The file is truncated once, then kept open: later prints append.

`print >> "<file>"`
Append to a file

`print | "<cmd>"`
Pipe output into a command
Call close("<cmd>") before END to flush it.

`print > "/dev/stderr"`
Write to stderr

ARRAYS

Array idioms

Idiom	Meaning
`a[$1]++`	count each key
`a[$1] += $2`	sum per key
`!seen[$0]++`	first occurrence only
`(k in a)`	key exists, no create
`delete a[k]`	remove one element
`delete a`	empty the whole array

Arrays are associative: keys are strings, so a[1] and a["1"] are the same slot. Merely reading a[k] creates it; test with (k in a).

RECIPES

Sum and average a column

```
awk '{ s += $3 } END { print s, s/NR }' <file>
```

Count per key, biggest first

```
awk '{ n[$1]++ } END { for (k in n) print n[k], k }' <file> | sort -rn
```

Drop duplicate lines, keep order

```
awk '!seen[$0]++' <file>
```

Comma-separated to tab, skip header

```
awk -F, -v OFS='\t' 'NR > 1 { $1 = $1; print }' <file>
```

Join two files on first column

```
# lookup table first, data second
awk 'NR==FNR { v[$1]=$2; next } $1 in v { print $0, v[$1] }' <a> <b>
```