AWK Cheatsheet

A concise cheat sheet covering essential AWK syntax, patterns, actions, and built-in functions, designed to help you quickly write and understand AWK scripts.

AWK Basics

Syntax

`awk 'pattern { action }' file`

AWK scripts consist of patterns and actions. For each line in the input `file`, AWK checks if the `pattern` matches. If it does, the `action` is executed. If no pattern is given, the action is performed for every input line. If no action is given, the matching line is printed.

`awk '{ print $1 }' file`

Prints the first field of each line in `file`. Fields are separated by whitespace by default.

`awk -F',' '{ print $1, $2 }' file`

Uses `,` as the field separator and prints the first and second fields of each line.

`awk 'BEGIN { print "Start" } { print $0 } END { print "End" }' file`

`BEGIN` block is executed before processing any input. `END` block is executed after processing all input. The `{ print $0 }` action prints each line of the input file.

Patterns

`BEGIN`	Executed before any input is read.
`END`	Executed after all input is read.
`expression`	A boolean expression that determines whether the action is executed. Example: `$1 > 10`
`pattern1, pattern2`	A range pattern that matches all lines from a line matching `pattern1` to a line matching `pattern2`.
`!pattern`	Negates the pattern. The action is executed if the line does *not* match the pattern.

Actions

`print`: Prints the current line or specified fields.
Example: `print $1, $3`

`printf`: Formatted printing, similar to C's `printf`.
Example: `printf "%s: %d\n", $1, $2`

`next`: Skips the current line and proceeds to the next input line.

`exit`: Terminates the AWK script.

`delete array[index]`: Deletes an element from an array.

Variables and Operators

Built-in Variables

`$0`	The entire current line.
`$1`, `$2`, `...`	The first, second, ... field of the current line.
`NF`	The number of fields in the current line.
`NR`	The number of the current line.
`FILENAME`	The name of the current input file.
`FS`	The field separator (default is whitespace). Can be changed with `-F` option or by assigning a value to `FS`.
`RS`	The record separator (default is newline).
`OFS`	The output field separator (default is whitespace).
`ORS`	The output record separator (default is newline).

Operators

`=`	Assignment operator.
`==`, `!=`	Equality and inequality operators.
`>`, `<`, `>=`, `<=``	Comparison operators.
`~`, `!~`	Regular expression match and non-match operators.
`&&`, `||`, `!`	Logical AND, OR, and NOT operators.
`+`, `-`, `*`, `/`, `^`, `%`	Arithmetic operators: addition, subtraction, multiplication, division, exponentiation, modulus.
`++`, `--`	Increment and decrement operators.
`+=`, `-=`, `*=`, `/=`, `%=`, `^=`	Compound assignment operators.

User-defined Variables

Variables can be defined and used within AWK scripts.

Example:

```
BEGIN { count = 0 }
{ count++ }
END { print "Total lines:", count }
```

Variables are initialized to zero or the empty string if not explicitly initialized.

Functions

Built-in Functions

`length(string)`	Returns the length of the string.
`substr(string, start, length)`	Returns a substring of the string starting at `start` with the given `length`.
`index(string, substring)`	Returns the starting position of `substring` in `string`, or 0 if not found.
`split(string, array, separator)`	Splits the string into elements of the `array` using `separator` as the delimiter. Returns the number of elements.
`match(string, regex)`	Returns the starting position of the regular expression `regex` in `string`, or 0 if not found. Sets `RSTART` and `RLENGTH`.
`gsub(regex, replacement, string)`	Globally substitutes all matches of the regular expression `regex` in `string` with `replacement`. Returns the number of substitutions made.
`tolower(string)`	Converts the string to lowercase.
`toupper(string)`	Converts the string to uppercase.
`sprintf(format, expr1, expr2, ...)`	Formats expressions `expr1`, `expr2`, ... according to the format string `format` (similar to C's `sprintf`).

User-Defined Functions

You can define your own functions in AWK.

Syntax:

```
function function_name(parameter1, parameter2, ...) {
  # Function body
  return value
}
```

Example:

```
function max(x, y) {
  return (x > y ? x : y)
}

{ print max($1, $2) }
```

Examples

Simple Examples

Print lines longer than 80 characters:
`awk 'length($0) > 80 { print }' file`

Print the total number of fields in the input:
`awk '{ total += NF } END { print "Total fields:", total }' file`

Print lines containing the word 'error':
`awk '/error/ { print }' file`

Print the last field of each line:
`awk '{ print $NF }' file`

Advanced Examples

Calculate the average of the values in the first field:
`awk '{ sum += $1; count++ } END { if (count > 0) print "Average:", sum / count }' file`

Print unique lines in a file:
`awk '!seen[$0]++' file`

Sum values in a specific column based on a condition:
`awk '$2 == "active" { sum += $1 } END { print "Sum of active values:", sum }' file`