Opensource.com: GNU awk Cheat Sheet BY JIM HALL

Use this handy quick reference guide to the most commonly used features of GNU awk (gawk).

COMMAND-LINE USAGE

Run a gawk script using **-f** or include a short script right on the command line.

gawk -f file.awk file1 file2…

or:

gawk 'pattern {action}' file1 file2…

also: set the field separator using **-F**

gawk -F: …

PATTERNS

All program lines are some combination of a pattern and actions:

pattern {action}

where **pattern** can be:

- **BEGIN** (matches start of input)
- **END** (matches end of input)
- a regular expression (act only on matching lines)
- a comparison (act only when true)
- empty (act on all lines)

ACTIONS

Actions are very similar to C programming.

Actions can span multiple lines.

End statements with a semicolon (;)

For example:

```
BEGIN { FS = ":"; }

{ print "Hello world"; }

{
  print;
  i = i + 1;
}
```

FIELDS

Gawk does the work for you and splits input lines so you can reference them by field. Use **-F** on the command line or set **FS** to set the field separator.

- Reference fields using **$**
- **$1** for the first string, and so on
- Use **$0** for the entire line

For example:

gawk '{print "1st word:", $1;}' file.txt

or:

gawk -F: '{print "uid", $3;}' /etc/passwd

REGULAR EXPRESSIONS

Common regular expression patterns include:

^	Matches start of a line
$	Matches end of a line
.	Matches any character, including newline
a	Matches a single letter **a**
a+	Matches one or more **a**'s
a*	Matches zero or more **a**'s
a?	Matches zero or one **a**'s
[abc]	Matches any of the characters **a**, **b**, or **c**
[^abc]	Negation; matches any character except **a**, **b**, or **c**
\.	Use backslash (\) to match a special character (like **.**)

You can also use character classes, including:

[:alpha:]	Any alphabetic character
[:lower:]	Any lowercase letter
[:upper:]	Any uppercase letter
[:digit:]	Any numeric character
[:alnum:]	Any alphanumeric character
[:cntrl:]	Any control character
[:blank:]	Spaces or tabs
[:space:]	Spaces, tabs, and other white space (such as linefeed)

OPERATORS

(…)	Grouping
++ --	Increment and decrement
^	Exponents
+ - !	Unary plus, minus, and negation
*** / %**	Multiply, divide, and modulo
+ -	Add and subtract
< > <= >= == !=	Relations
~ !~	Regular expression match or negated match
&&	Logical AND
||	Logical OR
= += -= *= /= %= ^=	Assignment

FLOW CONTROL

You can use many common flow control and loop structures, including if, while, do-while, for, and switch.

```
if (i < 10) { print; }

while (i < 10) { print; i++; }

do {
  print;
  i++;
} while (i < 10);

for (i = 1; i < 10; i++) { print i; }

switch (n) {
  case 1: print "yes";
  ⋮
  default: print "no";
}
```

FUNCTIONS

Frequently-used string functions include:

```
print "hello world"
print "user:" $1
print $1, $2
print i
print
```

Print a value or string. If you don't give a value, outputs **$0** instead.

Use commas (,) to put space between the values.

Use spaces ( ) to combine the output.

printf(fmt, values…)

The standard C **printf** function.

sprintf(fmt, values…)

Similar to the standard C **sprintf** function, returns the new string.

index(str, sub)

Return the index of the substring **sub** in the string **str**, or zero if not found.

length([str])

Return the length of the string **$0**.

If you include the string **str**, give that length instead.

FUNCTIONS (CONTINUED)

substr(str, pos [, n])

Return the next **n** characters of the string **str**, starting at position **pos**.

If **n** is omitted, return the rest of the string **str**.

tolower(str)

Return the string **str**, converted to all lowercase.

toupper(str)

Return the string **str**, converted to all uppercase.

Other common string functions include:

match(str, regex)

Return the position of the first occurrence of the regular expression **regex** in the string **str**.

sub(sub, repl [, str])

For the first matching substring **sub** in the string **$0**, replace it with **repl**.

If you include the optional string **str**, operate on that string instead.

gsub(sub, repl [, str])

Same as **sub()**, but replaces all matching substrings.

split(str, arr [, del])

Splits up the string **str** into the array **arr**, according to spaces and tabs.

If you include the optional string **del**, use that as the field delimiter characters.

strtonum(str)

Return the numeric value of the string **str**. Works with decimal, octal, and hexadecimal values.

USER-DEFINED FUNCTIONS

You can define your own functions to add new functionality, or to make frequently-used code easier to reference.

Define a function using the **function** keyword:

```
function name(parameters) {
  statements
}
```

