awk Quick Ref

compiled by v.ledos
rel 1.0 feb-2010

Usage

```
awk [-v var=val] 'program' [file1 file2...]
awk [-v var=val] -f progfile [file1 file2...]
```

Structure of an awk program

```
# comments
pattern { action }          A sequence of
pattern { action }          pattern-action
...                         statements
```

For each file,
  For each input line,
    For each pattern,
      If pattern matches input line, do the action.

"pattern"
BEGIN : executes "action" before starting to view the input file
END : executes "action" after ending to view the input file
Other : regular, numeric or string expression or combination

"action" is executable code
if (expression) statement1 **else** statement2
while (expression) statement
for (expr1;expr2;expr3) statement
do statement **while** (expression)
break / **continue** : immediately leave / start next iteration of innermost enclosing loop
exit / **exit** expression : go immediately to the END action; if within the END action, exit program

Built-in variables

$0 $1, $2 … $NF	Whole line, first, second… last field
ARGC	Number of command line arguments
ARGV	Array of command line arguments
FILENAME	Name of current input file
FS, RS	Input field / record separator (def: one space, \n)
NF	Number of fields in current record
NR, FNR	Number of record read so far / in current file
OFMT	Output format for numbers (default: %.6g)
OFS, ORS	Output field / rec. separator (def: one space, \n)
RESTART, RLENGTH	Start / Length of string matched by match function (see below)
SUBSEP	Subscript separator (default: \034)

Main built-in functions

r: regex ; s,t: strings ; n,p: integers

int(n), **sqrt**(n), **exp**(n), **log**(n), **sin**(n), **cos**(n)	
rand()	Random number between 0 and 1
close(file or command)	
getline [var] **getline** [var] < file command | **getline** [var]	Read next line from input file, from a specific file, or from a pipe
	Return 1 (record found), 0 (end of file), -1 (error)
gsub(r,s) **gsub**(r,s,t)	Substitute s for r globally in $0 / string t; return # of subs made
index(s,t)	Return first position of string t in s, or 0 if t is not present
length(s)	Return number of characters in s
match(s,r)	Test whether s contains a substring matched by r; return index or 0; sets RSTART and RLENGTH
split(s,a) **split**(s,a,fs)	Split s into array a on FS / field separaror fs; return # of fields
sprintf(fmt,expr-list)	Return expr-list formatted according to format string fmt
sub(r,s) **sub**(r,s,t)	Substitute s for the leftmost longest substring of $0 / t matched by r; return # of subs made
substr(s,p) **substr**(s,p,n)	Return substring of s (of length n) starting at position p
tolower(s), **toupper**(s)	Lower and upper cases

Formatted output

```
{ printf ("FORMAT",value1,value2,value3,…) }
```

%c %s	Print as character, as string
%-8s	Print as 8 characters, left aligned
%f %6.2f	Print as float number, with 6 digits (4 as integer, 2 as decimal)
\n	Line feed and carriage return

Operators

&& || !	Logical operators. Ex: !($2<4 || $3<20)
< <= == != >= > ~ !~	Comparing operators matched by, not
selector?if-true-exp:if-false-exp	

Basic programs

{ **print** NR, $0 }	Precede each line by line #
{ $1 = NR; **print** }	Replace first field by line #
{ $2 = **log**($2); $3 ="" ; **print** }	Replace the 2nd field by its logarithm, zap field 3
NF > 0	Print non-empty lines
NF > 0 {**print** $1, $NF}	Print first field and last one of non-empty lines
NF > 4	Print records containing more than 4 fields
$NF > 4	Print if last field greater than 4
NR%2==0	Print even-numbered lines
NR==10, NR==20	Print lines 10 to 20
/start/, /end /	Print lines between patterns
/regex/, EOF	Print from pattern to end of file
/regex/ {**print** $1}	Print first field of lines matching regex
$1 ~ /regex/	Print lines where first field matches
ORS=NR%5?",":"\n"	Concatenate every 5 lines of input, using comma separator
/regex/ {x++} **END** {**print** x}	Count and print the number of lines matching /regex/
{ nc += **length**($0) + 1; nw += NF } **END** { **print** NR, "lines", nw, "words", nc, "characters" }	wc command
{ sum += $1 } **END** { **print** sum, sum/NR }	Print sum and average
{ x[NR] = $0 } **END** {**for** (i = NR; i > 0; i--) **print** x[i]}	Reverse a file
{ a[$1] += $2 } **END** { **for** (i **in** a) **print** (i,":",a[i]) }	Group by field 1, and sum field 2
function pwr(a,b) { **return** exp(b*log(a)) } NF >= 2 { **print** pwr($1,$2) }	User defined function
BEGIN { RS=""; FS="\n" } { **print** "Name: ",$1 **print** "Address: ",$2 }	Multi-line records. Leading and trailing newlines are ignored