Compiled by Aluizio using the book UNIX IN A NUTSHELL, Arnold Robbins, O'Reilly Ed., 4th edition, 2005, ISBN 0-596-10029-9.

UNIX – awk Programming Language

The awk programming language is often used for text and string manipulation within shell scripts, particularly when input data can be viewed as records and fields. It is also an elegant and capable programming language that allows you to accomplish a lot with very little work.

awk is a pattern-matching program for processing files, especially when each line has a simple field-oriented layout.

Command-Line Syntax:

awk [options] 'script' var=value file(s)
awk [options] -f filescript var=value file(s)

Options:

Options	***Meaning***
-Ffs	Set fs as the field separator. Example: awk -F: '{print $1; print $2; print $3}' /etc/passwd (print the first 3 (colon-separated) fields of each record on separate lines).
-v var = value	Assign a value to variable before the script begin execution.

Simple Pattern-Action Examples:

Pattern-Action Example	***Meaning***
{print $1}	Print first field of each line.
/pattern/	Print all lines that contain pattern.
/pattern/ {print $1}	Print first field of lines that contain pattern.
NF > 2	Select records conataining more than 2 fields.
$1 ~ /URGENT/ {print $3, $2}	Print fields 2 and 3 in switched order, but only on lines whose first field matches the string URGENT.
/pattern/ {x++} END {print x}	Count and print the number of lines matching /pattern/.

Built-in Variables:

Variable	*Description*
ARGC	Number of arguments on the command-line.
ARGV	An array containning the command-line arguments, indexed from 0 to ARGC-1.
FILENAME	Current filename.
FS	Field Separator (space)
NF	Number of fields in current record.
NR	Number of current record.
OFS	Output field separator (space).
ORS	Output record separator (newline).
$0	Entire input record.
$n	nth field in current record; fields are separated by FS.

Functions:

Functions	*Description*
#	Ignore all text that follows on the same line.
and (exp1, exp2)	Return the bitwise AND of exp1 and exp2.
break	Exit from a while, for, or do loops.
continue	Begin next iteration of while, for, or do loops.
for (init-exp; test-exp; incr-exp) statement	C-styling for loop structure. Start from init-exp, incrementing incr-exp value each loop. Executing process until test-exp fails.
getline getline [var] [<file] command | getline [var]	Read next line of input. Reads input from file. Reads the output from a command.
if (condition) statement1 [else statement2]	If condition is true, do statement1; otherwise, do statement2.
or (exp1, exp2)	Return the bitwise OR of exp1 and exp2.

Functions	***Description***
print [output-expr[,...] [dest-expr]	Evaluate the output-expr and direct it to standard output followed by the value of OFS (output field separator). With no output-expr, print $0 (entire line). The output may be redirected to a file or **pipe** via dest-expr.
printf (format [,expr-list]) [dest-expr]	Borrowed from the C-language. Allows a formatted output. Example: In line 1, $1 = 5 and $2 = 5. {printf (“Sum of line %d is %d. \n”, NR, $1+$2)} Sum of line 1 is 10.
tolower (str)	Translate uppercase letters to lowercase.
toupper (str)	Translate lowercase letters to uppercase.

Output Redirections:

Redirection	***Meaning***
> file	Directs the output to a file, overwriting its previous contents. BE CAREFUL using this redirection.
>> file	Appends the output to a file, preserving its previous contents.
| command	Directs the output as the input to a command.