awk programming

- The Awk text-processing language is useful for such tasks as:
  - ★ Tallying information from text files and creating reports from the results.
  - ★ Adding additional functions to text editors like "vi".
  - ★ Translating files from one format to another.
  - ★ Creating small databases.
  - ★ Performing mathematical operations on files of numeric data.
- Awk has two faces:
  - ★ it is a utility for performing simple text-processing tasks, and
  - ★ it is a programming language for performing complex text-processing tasks.
- awk comes in three variations

  awk : Original AWK by A. Aho, B. W. Kernighnan and P. Weinberger
  nawk : New AWK, AT&T's version of AWK
  gawk : GNU AWK, all linux distributions come with gawk. In some distros, awk is a symbolic link to gawk.
- Simplest form of using awk
  - ♦ **awk** *pattern* {`action`}
  - ♦ Most common action: `print`
  - ♦ Print file dosum.sh: `awk '{print $0}' dosum.sh`
  - ♦ Print line matching bash in all files in current directory:
    `awk '/bash/{print $0}' *.sh`

- awk patterns may be one of the following

  BEGIN : special pattern which is not tested against input.
  Mostly used for preprocessing, setting constants, etc. before input is read.

  END : special pattern which is not tested against input.
  Mostly used for postprocessing after input has been read.

  /regular expression/ : the associated regular expression is matched to each input line that is read

  relational expression : used with the if, while relational operators

  && : logical AND operator used as pattern1 && pattern2.
  Execute action if pattern1 and pattern2 are true

  || : logical OR operator used as pattern1 —— pattern2.
  Execute action if either pattern1 or pattern2 is true

  ! : logical NOT operator used as !pattern.
  Execute action if pattern is not matched

  ?: : Used as pattern1 ? pattern2 : pattern3.
  If pattern1 is true use pattern2 for testing else use pattern3

  pattern1, pattern2 : Range pattern, match all records starting with record that matches pattern1 continuing until a record has been reached that matches pattern2

- *print expression* is the most common action in the awk statement. If formatted output is required, use the *printf format, expression* action.

- Format specifiers are similar to the C-programming language

  %d,%i : decimal number

  %e,%E : floating point number of the form [-]d.dddddd.e[±]dd. The %E format uses E instead of e.

  %f : floating point number of the form [-]ddd.dddddd

%g,%G : Use %e or %f conversion with nonsignificant zeros truncated. The %G format uses %E instead of %e

%s : character string

- Format specifiers have additional parameter which may lie between the % and the control letter

0 : A leading 0 (zero) acts as a flag, that indicates output should be padded with zeroes instead of spaces.

width : The field should be padded to this width. The field is normally padded with spaces. If the 0 flag has been used, it is padded with zeroes.

.prec : A number that specifies the precision to use when printing.

- string constants supported by awk

\\ : Literal backslash

\n : newline

\r : carriage-return

\t : horizontal tab

\v : vertical tab

```
~/Tutorials/BASH/scripts/day1/examples> echo hello 0.2485 5 | awk '{printf ``%s \t %f \n %d \v %0.5d\
      n'',$1,$2,$3,$3}'
hello    0.248500
 5
    00005
```

- The print command puts an explicit newline character at the end while the printf command does not.

- awk has in-built support for arithmetic operations

Operation	Operator
Addition	+
Subtraction	-
Multiplication	*
Division	/
Exponentiation	**
Modulo	%

Assignment Operation	Operator
Autoincrement	++
Autodecrement	–
Add result to varibale	+=
Subtract result from variable	-=
Multiple variable by result	*=
Divide variable by result	/=

```
~/Tutorials/BASH/scripts/day1/examples> echo | awk '{print 10%3}'
1
~/Tutorials/BASH/scripts/day1/examples> echo | awk '{a=10;print a/=5}'
2
```

- awk also supports trignometric functions such as sin(expr) and cos(expr) where expr is in radians and atan2(y/x) where y/x is in radians

```
~/Tutorials/BASH/scripts/day1/examples> echo | awk '{pi=atan2(1,1)*4;print pi,sin(pi),cos(pi)}'
3.14159 1.22465e-16 -1
```

- Other Arithmetic operations supported are

exp(expr)	: The exponential function
int(expr)	: Truncates to an integer
log(expr)	: The natural Logarithm function
sqrt(expr)	: The square root function
rand()	: Returns a random number N between 0 and 1 such that $0 \leq N < 1$
srand(expr)	: Uses expr as a new seed for random number generator. If expr is not provided, time of day is used.

- **awk** supports the if and while conditional and for loops
- if and while conditionals work similar to that in C-programming

```
if ( condition ) {
  command1 ;
  command2
}
```

```
while ( condition ) {
  command1 ;
  command2
}
```

- awk supports if ... else if .. else conditionals.

```
if (condition1) {
  command1 ;
  command2
} else if (condition2 ) {
  command3
}  else {
  command4
}
```

- Relational operators supported by if and while

==	:	Is equal to
!=	:	Is not equal to
>	:	Is greater than
>=	:	Is greater than or equal to
<	:	Is less than
<=	:	Is less than or equal to
~	:	String Matches to
!~	:	Doesn't Match

```
~/Tutorials/BASH/scripts/day1/examples> awk '{if (NR > 0 ){print NR,'':'', $0}}' hello.sh
1 : #!/bin/bash
2 :
3 : # My First Script
4 :
5 : echo ``Hello World!''
```

- The for command can be used for processing the various columns of each line

```
~/Tutorials/BASH/scripts/day1/examples> echo $(seq 1 10) | awk 'BEGIN{a=6}{for (i=1;i<=NF;i++){a+=$i
      }}END{print a}'
61
```

- Like all progamming languages, awk supports the use of variables. Like Shell, variable types do not have to be defined.
- awk variables can be user defined or could be one of the columns of the file being processed.

```
~/Tutorials/BASH/scripts/day1/examples> awk '{print $1}' hello.sh
#!/bin/bash
#
echo
~/Tutorials/BASH/scripts/day1/examples> awk '{col=$1;print col,$2}' hello.sh
#!/bin/bash

# My

echo ``Hello
```

- Unlike Shell, awk variables are referenced as is i.e. no $ prepended to variable name.
- awk one-liners: `http://www.pement.org/awk/awk1line.txt`

- awk can also be used as a programming language.
- The first line in awk scripts is the shebang line (#!) which indicates the location of the awk binary. Use **`which awk`** to find the exact location
- On my Linux desktop, the location is /usr/bin/awk.
- If unsure, just use /usr/bin/env awk

hello.awk

```
#!/usr/bin/awk -f

BEGIN {
  print "Hello World!"
}
```

```
~/Tutorials/BASH/scripts/day2/examples>./hello.awk
Hello World!
```

- To support scripting, awk has several built-in variables, which can also be used in one line commands

ARGC	:	number of command line arguments
ARGV	:	array of command line arguments
FILENAME	:	name of current input file
FS	:	field separator
OFS	:	output field separator
ORS	:	output record separator, default is newline

- awk permits the use of arrays
- arrays are subscripted with an expression between square brackets ([· · ·])

hello1.awk

```
#!/usr/bin/awk -f

BEGIN {
  x[1] = "Hello,"
  x[2] = "World!"
  x[3] = "\n"
  for (i=1;i<=3;i++)
    printf " %s", x[i]
}
```

```
~/Tutorials/BASH/scripts/day2/examples>./hello1.awk
 Hello, World!
```

- Use the delete command to delete an array element
- awk has in-built functions to aid writing of scripts

length	: length() function calculates the length of a string.
toupper	: toupper() converts string to uppercase (GNU awk only)
tolower	: tolower() converts to lower case (GNU awk only)
split	: used to split a string. Takes three arguments: the string, an array and a separator
gsub	: add primitive sed like functionality. Usage gsub(/pattern/,"replacement pattern",string)

getline : force reading of new line

- Similar to bash, GNU awk also supports user defined function

```
#!/usr/bin/gawk -f
{
    if (NF != 4) {
        error(``Expected 4 fields'');
    } else {
        print;
    }
}
function error ( message ) {
    if (FILENAME != ``-'') {
        printf(``%s: ``, FILENAME) > ``/dev/tty'';
    }
    printf(``line # %d, %s, line: %s\n'', NR, message, $0) >>
          ``/dev/tty'';
}
```