Unit 5
AWK

Introduction

- Awk is a filter program developed in 1977 by Aho Weinberger and Kernighan
- It is pattern scanning language
- It is programming language with C-like control structures, functions and variables
- It was designed to work with structured files and text patterns
- It operates at file level

Awk command line Syntax

- The general format of an awk command line is
  - $ awk options ‘program’ filelist
  - Use of options is optional
  - Filelist will have zero or more input filenames
  - Program will have one or more statements having the following general format
    - pattern {action}
- The pattern component of a program statement indicates the basis for a line or record selection and manipulation.
- The action part of every program is surrounded by a pair of curly brackets

- The action part is made up of C-like statements, which performs actions on the lines or records selected based upon pattern component.
- The pattern can be simple words or regular expressions as in egrep or they can be more complicated conditions like in C language.
- Awk has only two options –F and –f . –F input field separator, -f program is on a separate file.

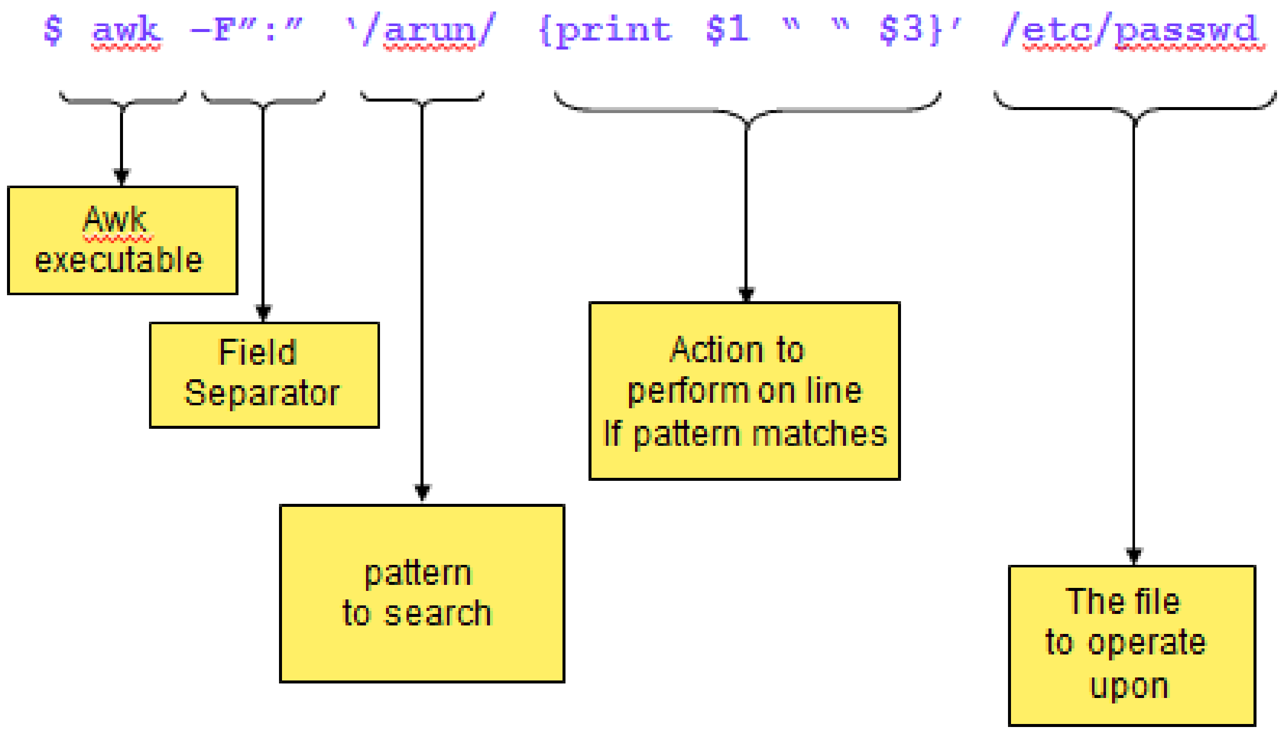
Simple Example

- Problem : Get the userid of user "arun" from the /etc/passwd file.
- Suppose /etc/passwd file contains the following entries

```
arun:x:504:504::/home/arun:/bin/bash
try:x:500:500::/home/try:/bin/bash
optima:x:501:501::/home/optima:/bin/bash
optimal:x:502:502::/home/optimal:/bin/bash
```

- awk will see this file as follows
  - 1 line = 1 record (by default) so in total there are 4 records in the file.
  - 1 record = 7 fields separated by ":" (Not by default)

  Note : Default field separator is space.



```
$ awk -F":" '/arun/ {print $1 " " $3}' /etc/passwd
```

Structure of an awk script

BEGIN { ……
Program Statements
END { ….}

- An awk script can have one or more of these sections.
- The presence of any of these sections is not mandatory
- Awk must contain at least one of these sections

The BEGIN section

- Recognized by the keyword BEGIN
- All the instructions present in this section are executed once before the awk actually starts reading and executing program statements from the body
- Instructions such as initializing variables, generating headings and other similar tasks can be done here
- The input field separator can be specified here
  - 'BEGIN { FS ":"}'

The END section

- Recognized by the keyword END
- All the instructions present in this section are executed once after the last line of the input has been processed.
- Instructions in this section are used for generating summary reports
- 'END {print NR}'
- NR represents the number of records

The BODY section

- This section contains one or more actual program statements
- Operational mechanism of awk
  - Awk picks up records or lines from the input file one by one and applies all the program statements present on the program file to each line.
  - The pattern is checked on the picked up line one by one
  - When the pattern matches the action mentioned in the action portion is carried out on the present line.
  - The awk statement might be made up of operators, decision making statements, loop statements, regular expressions.

variables

- Variables can be of User defined variables & Built-in variables
- User defined variables
  - Storage location can hold string or numbers
  - Alphanumeric and underscores are allowed
  - Names begin with a letter
  - Variables defined will get set to zero or a null string automatically

- Built-in variables
  - Upper case letters
  - Some variables have default values
  - Others can be controlled by the awk program

Name	Meaning
FILENAME	Name of the current Input File
FS	Input Field Separator (def: blank & tab)
NF	Number of fields in input record
NR	Number of current record
OFS	Output Field Separator (def: blank & tab)
ORS	Output Record Separator(def: newline)
RS	Input Record Separator (def: newline)
ARGC	Number of command line arguments
ARGV	Command line arguments array

Records, Fields and Special Variables

- Every line of the input file is treated as a record
- Input file can be a text file or database file
- Each unit or word is considered to a field. Hence record is made up of fields
- These fields are separated by blank or tab character. FS default value is blank or tab
- If required the FS can be changed
- Awk automatically splits lines/record into fields. Contents of the fields are stroed in $1,$2…
- The total number of fields information is stored in NF
- $0 - The current line or record that is currently processed is stored in a varaible $0

Understanding Special Variables

- Eg: Suppose awk is reading the 14th line of a file emp.lst and the line is thenmozhi 4424 11/14/77 54321

Variable	Value
$0	thenmozhi 4424 11/14/77 54321
$1	thenmozhi
$2	4424
$3	11/14/77
$4	54321
NR	1
FS	blank
NF	4
FILENAME	emp.lst

Addressing: Line and Context

- Pattern is to be matched against the file
- Pattern can be record numbers or field or any portion of a record.
- Selecting by record numbers – Line addressing
  - Specified range can be selected by starting address and ending address separated by comma in the pattern
- Selecting by content – context addressing

Patterns

- Awk can have a pattern which is matched against each line of the input file
- There may be occasions there is no pattern. Such patterns are called no pattern case
- In such cases action in taken on all the records or lines of the input file
- Pattern is made up of expression
- It can be a arithmetic expression, relational expression, logical expression or regular expression

Awk Operators

- Arithmetic operators : +,-,/,* ,^ , %
- Logical Operators: ||,&&,!
- Relational Operators: >,>=,<,<=,==,!=
- Assignment Operators(shorthand): =,+=,-=,*=,/=,%=
- Increment and Decrement Operators: ++,--
- Match Operators: ~ , !~

What this statement will give?

- `awk '{print NR, $1, $2, $5}' emp.lst`
- `awk -F:'/Raksha/{print $1, $2}’ emp.lst`
- `awk -F: '/00$/ {print $0}' emp.lst`
- `awk '($4 > 40000)&&($4 <= 60000) {print $0}' emp.lst`
- `awk '$4 == 54321 || $4 > 50000 {print $0}' emp.lst`

awk script

Writing awk script – sample1.awk

BEGIN { print “Record No” “ “ “Last Name”}

{print “NR” “ “ $2}

END { print “Number of records processed are NR in the file FILENAME}

Execution

Awk –f sample1.awk emp.lst

- cat marks.pu

```
Radhika     72    67    96
Darshana    86    97    93
Anil        88    96    91
Prasanna    75    86    79
Vinay       45    99    88
```

Write simple awk programs

- Print first and second field of marks.pu file
- Print name column where marks in Subject1 is greater than 80 in marks.pu file
- Print name and total, where total is greater than 240 in marks.pu file
- Print the names of students who have secured marks between 60 to 80 in Subject1
- Print the names of students whose name start with either D or V
- Print the names of students whose name is Vinay or Anil

awk with pipe

- Being a filter program, awk can take its input from the output of another program.
- It can lie in either side of the pipe.
- Eg:

```
date|awk '{print "The day is",$1
          print "The month is",$2
          print "The year is",$6}‘
```

```
awk '{print "%-9s %5d \n",$1,$2+$3+$4}' marks.pu | sort -r -
t" " -k 2 >result
```

Awk control structures

- If ---else

```
if (expression) {
   statement1 }
else {
    statement2 }
```

- Entry controlled - While

```
while(expression) {
Statements}
```

- Exit controlled – do

```
do
    statements
while(expression)
```

- For

```
{for (expr1;condition;expr2)
 statements }
```