Awk Programming

Anuj Singh Tomar

2008

Name: - Anuj Singh Tomar
Address: - Gola Ka Mandir
Mobile: - +919926604345
Landline: - 07512361477

Complete reference to awk programming
Plus a few UNIX concepts which will be necessary in a long run.

Table of Contents

AWK PROGRAMMING

One of the most powerful data manipulation utilities is awk, a program that incorporates a wide range of data matching, modifying, and programming features. The awk is the first letters of the last names of its three developers, **A**ho, **W**einberger, and **K**ernighan. The awk utility like grep is a pattern matching tool, but with the added ability to perform specified, often complex, operations on records or on specific fields in records after a pattern is matched.
In addition awk is fully programmable- capable of supporting the loops, conditional statements, and variables expected in a programming language.

One of the most important differences b/w awk and grep is awk's ability to select records on the basis of the location of values within a record. In addition awk can select pieces of a record for processing. This can only be accomplished when the data is organized in a structured manner, as in a database.

Ways in which to use Awk

the awk utility reads data files or input that is the output of another utility. In this section several introductory forms of the awk utility are used to manipulate data read from files.

- **awk '/anuj/ {print}' records**
  All lines containing anuj in file records are displayed

  ```
  $ awk '/anuj/ {print}' records
  anuj singh tomar:+919926604345:06:02:1987
  ```

- **awk '/anuj/' records**
  all lines that contain the target anuj are displayed.

  ```
  anuj singh tomar:+919926604345:06:02:1987
  ```

- **Not specifying a pattern**
  awk '{print}' records
  Every record of the entire file is displayed.

- **awk '{print $1' records**
  The first field of the file records is displayed.
  ```
  $ awk '{print $1}' records
  anuj
  satyendra
  raj
  ```

- **awk 'print $3 $2 $1' records**
  The three fields of the file records are displayed without spaces b/w the fields.

tomar:+919926604345:06:02:1987singhanuj
narwariya:+919926547924:04:07:1987satyendra
gurjar:+919977186990:08:07:1987kishoreraj

- **awk 'print $3, $2, $1' records**

the three fields of the file records are displayed with spaces b/w the fields.

$ awk -F: '{print $3, $2,$1}' records
06 +919926604345 anuj singh tomar
04 +919926547924 satyendra narwariya
08 +919977186990 raj kishore gurjar
05 +919893110089 navdeep rajput

- **$awk '{print $0}' records**

Prints all the records from the records file.

$ awk -F: '{print $0}' records
anuj singh tomar:+919926604345:06:02:1987
satyendra narwariya:+919926547924:04:07:1987
raj kishore gurjar:+919977186990:08:07:1987
navdeep rajput:+919893110089:05:06:1987

Identifying variables and strings of characters

The $1 is a variable, awk interprets every variable as instruction to replace it with its value. we can create variable and use it with awk.

$ awk -v item='Name = ' '{print item,$1}' records
Name = anuj
Name = satyendra
Name = raj
Name = navdeep

the **-v option tells awk that the first argument passed to awk tells awk that a variable definition follows.**

' ' Are used to assign value to a variable
“ ” To print a string

- **$ awk -F: '{print "Name = ",$1,"Number = ",$2}' records**

Name = anuj singh tomar Number = +919926604345
Name = satyendra narwariya Number = +919926547924
Name = raj kishore gurjar Number = +919977186990

Selecting with regular expressions

- **$ awk '/anuj/ {print $0}' records**

anuj singh tomar:+919926604345:06:02:1987

$ awk '/[Aa]nuj/ {print $0}' records
anuj singh tomar:+919926604345:06:02:1987
Anuj singh tomar:+919926604345:06:02:1987

○ *Specifying beginning of lines*

- **$ awk -F: '/^v/ {print $1,$2}' records**
  All lines in the records file that start with a character v are displayed.
  vidyasagar yadav +919349736772

  The output will be all lines starting with other than a through m
  $ awk -F: '/^[^a-m]/ {print "Name = ", $1}' records
  Name = satyendra narwariya
  Name = raj kishore gurjar
  Name = navdeep rajput
  Name = ravi poddar
  Name = vidyasagar yadav

  The output will be all the lines starting with a through m
  $ awk -F: '/^[a-m]/ {print "Name = ", $1}' records
  Name = anuj singh tomar
  Name = Anuj singh tomar
  Name = madhuraj tomar
  Name = amit kumar gupta

Selecting records by specific database components

- **$ awk -F: '$1=="Anuj singh tomar" {print $0}' records**
  It shows those records which have anuj in their first field
  Anuj singh tomar:+919926604345:06:02:1987

- This command will display all those records which have +919926604345 in its 2nd field.
  $ awk -F: '$2==+919926604345' records
  anuj singh tomar:+919926604345:06:02:1987
  Anuj singh tomar:+919926604345:06:02:1987

- This command shows those files which have either gurjar or tomar in it.
  $ awk '/gurjar/ || /tomar/' records
  anuj singh tomar:+919926604345:06:02:1987
  Anuj singh tomar:+919926604345:06:02:1987
  raj kishore gurjar:+919977186990:08:07:1987
  madhuraj tomar:+919977859602:04:01:1987

- **$ awk -F: '$1=="Anuj singh tomar" && $2=="+919926604345"' records**
  Anuj singh tomar:+919926604345:06:02:1987

○ *Finding patterns within the fields*

- This will search for '6' in 3rd field

  $ awk -F: '$3 ~ /6/' records

anuj singh tomar:+919926604345:06:02:1987

Anuj singh tomar:+919926604345:06:02:1987

- **It** shows those lines which have 6 in it.

  $ awk '/6/' records

  anuj singh tomar:+919926604345:06:02:1987

  Anuj singh tomar:+919926604345:06:02:1987

  satyendra narwariya:+919926547924:04:07:1987

  raj kishore gurjar:+919977186990:08:07:1987

  navdeep rajput:+919893110089:05:06:1987

  madhuraj tomar:+919977859602:04:01:1987

  vidyasagar yadav:+919349736772:05:12:1987

- Shows those lines which have 5 fields

  $ awk -F: ' (NF == 5)' records

  anuj singh tomar:+919926604345:06:02:1987

  Anuj singh tomar:+919926604345:06:02:1987

  $ awk -F: ' (NF == 4)' records

  no output since no line has 4 fields all have 5 fields.

Creating and Using awk command files

when we place the complex awk commands in a separate file and then associate these files on the command line then we reduce both complexity and the potential for errors.

Example

In a separate file place the following code.

/Anuj/ {print $1, $2}

On command line enter foll.

awk -F: -f ex1.ak records

The resulting output will be the first and second fields of the records in file records that contain the string Anuj. **-F: should be used before -f otherwise it will show an error we can also specify field separator in the ex1.ak file and we will see it later.**

$ awk -F: -f ex1.ak records
Anuj singh tomar +919926604345

- **$ cat ex2.ak**

  BEGIN {

```
FS=":"
OFS="-----"
ORS="\n"
}
{
print "Record no. is " NR,$1,$2
}
```

$ awk -f ex2.ak records

```
Record no. is 1-----anuj singh tomar-----+919926604345
Record no. is 2-----Anuj singh tomar-----+919926604345
Record no. is 3-----satyendra narwariya-----+919926547924
Record no. is 4-----raj kishore gurjar-----+919977186990
Record no. is 5-----navdeep rajput-----+919893110089
Record no. is 6-----madhuraj tomar-----+919977859602
```

In **FS** we specify the record **Field Separator**.
In **OFS** we specify the Output **Field Separator**.
In **ORS** we specify the Output **Record Separator**.

The BEGIN's opening curly brace starts at the same line and not on the new line.

Employing Variables

User defined variables are also supported by awk and they work when you are trying to improve the readability of the code.

- Create a file ex3.ak with the following awk code.

$ vi ex3.ak

```
BEGIN {
FS=":"
}
/Anuj/ {
name=$1
number=$2
print name, price
}
```

$ awk -f ex3.ak records

```
Anuj singh tomar +919926604345
```

Using variable names as words

in awk, literals are always enclosed in quotation marks, variables on the other hand are not quoted.
Ex.

- **$ awk -F: '/Anuj/{name=$1; print "name",name}' records**
  name Anuj singh tomar

Performing arithmetic operations in awk

In addition to manipulating character strings, the awk utility can apply arithmetic operations to variables and data.
Ex.
Subtract one day from date of birth in file records and show the filtered records from 2 to 4.

$ awk -F: 'NR==2,NR==4{ print NR,$1,$3-1}' records
2 Anuj singh tomar 5
3 satyendra narwariya 3
4 raj kishore gurjar 7

Maintaining a running Total

The way in which awk creates and initializes variables can be used to maintain an updated or running total on items in a database.

$ cat ex4.ak

```
BEGIN {
FS=":"
}
{
name=$1
number=$2
total=$3 * $4
running=running+total
print name,total,running
}
```

$ awk -f ex4.ak records

```
anuj singh tomar 12 12
Anuj singh tomar 12 24
satyendra narwariya 28 52
raj kishore gurjar 56 108
```

Using the printf function to format output

the awk utility borrows some of its notation and functions from C language, in which the utility is written. May be Kernighan, who was an author of both, had something to do with it. the C function

printf, is commonly used in awk code to provide additional formating capabilities over basic print

Left and Right justifying the output

Modify the printf function as follows of the previous example:-

printf "%-20s %10s %10s\n", name,total,running

The newly added format specifiers -20 and 10 have altered the appearance of the output .These numerical specifiers create minimum field widths of 20 and 10,10 characters. Their respective variables are left and right,right

Output:

$ awk -f ex4.ak records

anuj singh tomar	12	12
Anuj singh tomar	12	24
satyendra narwariya	28	52
raj kishore gurjar	56	108
navdeep rajput	30	138

Aligning the decimal and truncating numbers

All decimal points in the output should be aligned to do this use following:

%10.2f = tells to right align a floating point number held to a precision of two decimal places rather than a string. This results in an improved alignment.

COMMAND SUMMARY

- **-Fcharacter**
  When used on command line the ***-F*** flag informs awk to use the specified ***character*** as the field separator.
- **-v variablename=value**
  Assign the value to variable before execution of the program . such variable values are available to the BEGIN block of an awk program.

- **;**
  Separates actions in a block
- **BEGIN**
  Instructs awk to perform the following block of actions **before** processing of the database.
- **END**
  Instructs awk to perform the following block of actions **after** processing of the database.

An Original Work By: - **Anuj Singh Tomar** *Mobile:* **+919926604345**

Summary of awk predefined variables

- **$#**

  the value of @# is the content of the #th field in the current record.

- **$0**

  the value of $0 is the content of all the fields in the current record.

- **NF**

  the value of NF is the number of fields in the current record.

- **NR**

  The value of NR is the Record number of the current record.

- **FS**

  The value of FS is the value of the field separator. Default separators (Delimiters) are one or more spaces, or a tab.

- **OFS**

  The output field separator, a space by default.

- **RS**

  The value of RS is the value of the record separator , the default separator is a newline character.

- **ORS**

  the output record separator , by default a newline.

Summary of awk printing commands

- **printf "string"**
  Prints the string enclosed by the double quotes.
- **printf "\tstring\n"**
  prints the string enclosed by double quotes, preceded by a tab and followed by a newline.
- **print "string %s \n", variable**
  print the string in "" replacing ***%s*** with the variable and starting at a newline.
- **printf "%ns" , variable**
  printf string variable ***right*** justified to ***n*** number of spaces
- **printf "%-ns" , variable**
  printf string variable ***left*** justified to ***n*** number of spaces
- **printf "%nf", variable**
  print the value of variable as a floating point number, right justified against the end space of a field **n** characters wide.
- **printf "%n.nf", variable**

print the value of variable as a floating point number, rounded to the ***nth*** decimal point, right justified to the ***nth*** space..

Summary of operators

TYPE OF OPERATOR	OPERATORS	FUNCTION
Logical	a || b	True if either a or b is true.
	a && b	True if both a and b are true.
	! a	true if a is not true
Assignment	a = b	assign value of b in a
	a += b	a=a+b
Arithmetic	+,-,*,/	same meaning as symbol
Relations	a==b	true if a matches b
	a < b	true if a < b
	a > b	true if a > b
	a ~ b	true if field a contains string b.