awk

awk is a utility that enables a programmer to write tiny but effective programs in the form of statements that define text patterns that are to be searched for in each line of a document and the action that is to be taken when a match is found within a line.

Awk is a scripting language used for manipulating data and generating reports.

The **awk** command programming language requires no compiling, and allows the user to use variables, numeric functions, string functions, and logical operators.

Awk is create by:

Aho, Weinberger, and Kernighan

The Awk is mostly used for pattern scanning and processing. It searches one or more files to see if they contain lines that matches with the specified patterns and then perform associated actions.

What can you do with awk?

1. **awk operation:**

   (a) scans a file line by line

   (b) splits each input line into fields

   (c) compares input line/fields to pattern

   (d) performs action(s) on matched lines

2. **Useful for:**

   (a) transform data files

   (b) produce formatted reports

3. **Programming constructs:**

(a) format output lines

(b) arithmetic and string operations

(c) conditionals and loops

Basic awk syntax

awk options 'selection _criteria {action }' files(s)

Option	Description
-f *program-file*	Read the AWK program source from the file *program-file*, instead of from the first command line argument.
-F *fs*	Use *fs* for the input field separator

Simple Awk Filtering

1. Let us create employee.txt file which has the following content, which will be used in the examples .

```
$cat > employee.txt

ajay manager account 45000
```

```
sunil clerk account 25000
varun manager sales 50000
amit manager account 47000
tarun peon sales 15000
deepak clerk sales 23000
sunil peon sales 13000
satvik director purchase 80000
```

Example

1. Default behavior of Awk

By default Awk prints every line from the file.

```
$ awk '{print}' employee.txt
```

Output will be

```
ajay manager account 45000
sunil clerk account 25000
varun manager sales 50000
amit manager account 47000
tarun peon sales 15000
deepak clerk sales 23000
sunil peon sales 13000
satvik director purchase 80000
```

In the above example pattern is not given. So the actions are applicable to all the lines. Action print with out any argument prints the whole line by default. So it prints all the lines of the file with out fail. Actions has to be enclosed with in the braces.

2.Print the lines which matches with the pattern.

```
$ awk '/manager/ {print}' employee.txt
```

Output will be

```
ajay manager account 45000
varun manager sales 50000
amit manager account 47000
```

In the above example it prints all the line which matches with the 'manager'

```
$ awk '/manager/||/clerk/ {print}' employee.txt
```

Output will be

```
ajay manager account 45000
sunil clerk account 25000
varun manager sales 50000
amit manager account 47000
deepak clerk sales 23000
```

```
$ awk '/manager/
>/clerk/ {print}'
employee.txt
```

In the above example it prints all the line which matches with the ‘manager’ or ‘clerk’. It has two patterns. Awk accepts any number of patterns, but each set (patterns and its corresponding actions) has to be separated by newline.

3. Spliting a Line Into Filed

For each record i.e line, it splits the record delimited by whitespace character by default and stores it in the $n variables. If the line has 4 words, it will be stored in $1, $2, $3 and $4. $0 represents whole line.

```
$ awk '{print $1,$4}' employee.txt
```

Output will be

```
ajay 45000
sunil 25000
varun 50000
amit 47000
tarun 15000
deepak 23000
sunil 13000
satvik 80000
```

In the above example $1 and $4 represents Name and Salary respectively.

Use -F option to specify delimiter

1. Let us create employee.txt file which has the following content, which will be used in the examples .

```
$cat>employee.txt

Ajay|manager|account|45000
Sunil|clerk|account|25000
Varun|manager|sales|50000
Amit|manager|account|47000
Tarun|peon|sales|15000
Deepak|clerk|sales|23000
Sunil|peon|sales|13000
Satvik|director|purchase|80000
```

```
$ awk -F "|" '{print $1,$4}' employee.txt
```

```
$ awk -F "|" '/manager/ {print $1,$4}' employee.txt
```

Use regular expression in awk

```
$ awk -F "|" '/Sa[kx]s*ena/' employee.txt
```

4.Built In Variables In Awk

Awk's built-in variables include the field variables—`$1`, `$2`, `$3`, and so on (`$0` is the entire line)—that break a line of text into individual words or pieces called *fields*

NR:

Keeps a current count of the number of input records. Remember that *records* are usually lines; Awk performs the pattern/action statements once for each record in a file.

NF:

Keeps a count of the number of fields within the current input record.

FILENAME:

Contains the name of the current input file.

FS:

Contains the *field separator* character used to divide fields on the input line. The default is "white space", meaning space and tab characters. `FS` can be reassigned to another character (typically in `BEGIN`) to change the field separator.

RS:

Stores the current *record separator* character. Since, by default, an input line is the input record, the default record separator character is a newline.

OFS:

Stores the *output field separator*, which separates the fields when Awk prints them. The default is a blank space. Whenever `print` has several parameters separated with commas, it will print the value of `OFS` in between each parameter.

ORS:

Stores the *output record separator*, which separates the output lines when Awk prints them. The default is a newline character. `print` automatically outputs the contents of `ORS` at the end of whatever it is given to print.

ARGC:

The number of command-line arguments present.

ARGV:

The list of command-line arguments.

Example1

Use of NF built-in variables (Display Last Field)

```
$ awk '{print $1,$NF}' employee.txt
```

Output will be

```
ajay 45000
sunil 25000
varun 50000
amit 47000
tarun 15000
deepak 23000
sunil 13000
satvik 80000
```

In the above example $1 represents Name and $NF represents Salary. We can get the Salary using $NF , where $NF represents last field.

Example2

Use of NR built-in variables (Display Line Number)

```
$ awk '{print NR,$0}' employee.txt
```

Output will be

```
1 ajay manager account 45000
2 sunil clerk account 25000
3 varun manager sales 50000
4 amit manager account 47000
5 tarun peon sales 15000
6 deepak clerk sales 23000
7 sunil peon sales 13000
8 satvik director purchase 80000
```

In the above example it prints all the line along with line number.

Example3

```
$ awk '{print NR,$1,$4}' employee.txt
```

Output will be

```
1 ajay 45000
2 sunil 25000
3 varun 50000
4 amit 47000
5 tarun 15000
6 deepak 23000
7 sunil 13000
8 satvik 80000
```

In the above example it prints name and salary along with line number.

Example4

Another use of NR built-in variables (Display Line From 3 to 6)

```
$ awk 'NR==3, NR==6 {print NR,$0}' employee.txt
```

Output will be

```
3 varun manager sales 50000
4 amit manager account 47000
```

```
5 tarun peon sales 15000
6 deepak clerk sales 23000
```

Example4

Use of FS and OFS built-in variables

1. Let us create employee.txt file which has the following content, which will be used in the example.(use ,(comma) between each fileds)

```
$cat > employee.txt

ajay,manager,account,45000
sunil,clerk,account,25000
varun,manager,sales,50000
amit,manager,account,47000
tarun,peon,sales,15000
deepak,clerk,sales,23000
sunil,peon,sales,13000
satvik,director,purchase,80000


$awk
'BEGIN {FS="|"}{print $1 OFS  "," $4 }' employee.txt
```

Output will be

```
Ajay|45000
Sunil|25000
Varun|50000
Amit|47000
Tarun|15000
Deepak|23000
Sunil|13000
Satvik|80000
```

In the above example it prints Filed separator between Name And Salary(FS is for Input Field Separator and OFS is for Output Field Separator)

5.Opertaors in Awk

1.Awk Arithmetic Opertors

The following operators are used for performing arithmetic calculations.

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo Division

2.Awk Assignment Operators

Awk has Assignment operator and Shortcut assignment operator as listed below.

Operator	Description
=	Assignment
+=	Shortcut addition assignment
-=	Shortcut subtraction assignment
*=	Shortcut multiplication assignment
/=	Shortcut division assignment

%=	**Shortcut modulo division assignment**

3.Awk Conditional Operators

Awk has the following list of conditional operators which can be used with control structures and looping statement

Operator	Description
>	**Is greater than**
>=	**Is greater than or equal to**
<	**Is less than**
<=	**Is less than or equal to**
<=	**Is less than or equal to**
==	**Is equal to**
!=	**Is not equal to**

4. Awk Logical Operators

&&	**Both the conditional expression should be true**
||	**Any one of the conditional expression should be true**

5. The Regular Expression Operators

Operator	Description
~	Match operator
!~	No Match operator

Example1

```
$ awk '$2=="manager" || $2=="clerk" {print $0}'
employee.txt
```

Output will be

```
ajay manager account 45000
sunil clerk account 25000
varun manager sales 50000
amit manager account 47000
deepak clerk sales 23000
```

In the above example it prints all the line which matches with the ‘manager’ or ‘clerk’

```
$ awk '$2=="manager" || $2=="clerk" ' employee.txt
```

Example2

```
$ awk '$2=="manager" && $4>=50000' employee.txt
```

Output will be

```
varun manager sales 50000
```

Example3

```
$ awk '$2!="manager"' employee.txt
```

Output will be

```
sunil clerk account 25000
tarun peon sales 15000
deepak clerk sales 23000
sunil peon sales 13000
satvik director purchase 80000
```

Use –f option

-f *program-file*	Read the AWK program source from the file *program-file*, instead of from the first command line argument.

Let us create calgrosss.awk file to calculate hra,da gross and use it by using –f option

```
$cat > calgross.awk
{hra=1500;da=2500;gross=hra+da+$4;print $0,hra,da,gross}
```

```
$ awk –f calgross.awk employee.txt
```

Output will be

```
ajay manager account 45000 1500 2500 49000
sunil clerk account 25000 1500 2500 29000
varun manager sales 50000 1500 2500 54000
amit manager account 47000 1500 2500 51000
tarun peon sales 15000 1500 2500 19000
deepak clerk sales 23000 1500 2500 27000
sunil peon sales 13000 1500 2500 17000
satvik director purchase 80000 1500 2500 84000
```

Operator	Description
~	Match operator
!~	No Match operator

```
$ awk –F "|" '/Sa[kx]s*ena/' employee.txt
```

This matches the pattern anywhere in the line and not in a specific field. For matching a regular expression with in a specific field , awk offers the ~ and !~ operators to match and negate a match, respectively.

Example3

```
$ awk –F "|" '$1 ~/Sa[kx]s*ena/' employee.txt

$ awk –F "|" '$1 ~/kukreti/ {print}' employee.txt
```

Redirecting print output and Piping use

```
$ awk '{print $1,S4 > "newfile" }' employee.txt

$ cat newfile
```

Output will be

```
1 ajay 45000
2 sunil 25000
3 varun 50000
4 amit 47000
5 tarun 15000
6 deepak 23000
7 sunil 13000
8 satvik 80000
```

Sort according to name

```
$ awk '{print $0 | "sort"}' employee.txt
```

Output will be

```
ajay manager account 45000
amit manager account 47000
deepak clerk sales 23000
satvik director purchase 80000
sunil clerk account 25000
sunil peon sales 13000
tarun peon sales 15000
varun manager sales 50000
```

```
$ awk '{print $1,$4 | "sort"}' employee.txt
```

Output will be

```
ajay 45000
amit 47000
deepak 23000
satvik 80000
sunil 13000
sunil 25000
tarun 15000
varun 50000
```

Another example Of Pipe In Awk

$date

```
Mon Dec  9 10:27:11 EST 2013
```

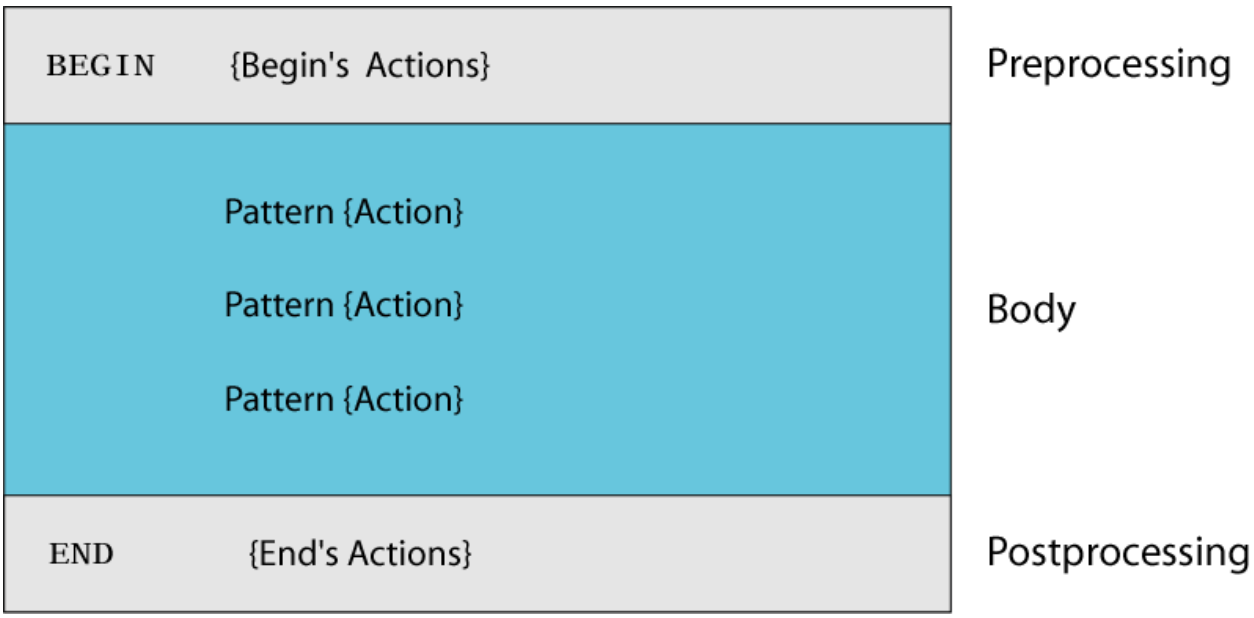
```
$date | awk '{print "month:" $2 "\nYear:" $6}'
```

Output will be

Month:Dec
Year:2013

The BEGIN and END Sections

awk scripts are divided into three major parts



BEGIN: pre-processing

Performs processing that must be completed before the file processing starts (i.e., before awk starts reading records from the input file)

Useful for initialization tasks such as to initialize variables and to create report headings

BODY: Processing

Contains main processing logic to be applied to input records

END: post-processing

Contains logic to be executed after all input data have been processed.

Logic such as printing report grand total should be performed in this part of the script .

Examples

1.Write a Awk script to count the total number of Lines in a given Filename

```
$ awk 'BEGIN{c=0}{c++} END{print "Total Number
Of Lines= ",c}' employee.txt
```

Output will be

```
Total Number Of Lines= 8
```

2.Write a Awk script to count the total number of employees in account department Filename

```
$ awk 'BEGIN{c=0;}$4 ~ /account/ {c++;}
END{print "Total Number Of Employee in Account
Department= ",c}' employee.txt
```

Output will be

```
Total Number Of Employee in Account Department= 3
```

Another Example Of Awk Script

Billing for Item

In this example, the input file itemdata.txt contains records with fields — item name, Item Quantity and Rate per Item.

```
$cat > itemdata.txt
pencil 50 5
eraser 100 4
scale 25 15
color 100 25
notebook 50 15
```

Now the following Awk script, reads and processes the above itemdata.txt file, and generates report that displays — rate of each item sold, and total amount for all the item sold.

Now Create Awk script for billing calculation for item is given below.

```
$cat > bill.awk
BEGIN{
```

```
tamount=0;
}
{
iqty=$2;
iprice=$3;
total=iqty*iprice;
tamount=tamount+total;
print $1 "\t" total
}
END{print "Total Amount= " tamount;}
```

Now execute the bill.awk script to generate the report .

```
$ awk –f bill.awk itemdata.txt
```

Output will be

```
Pencil 250
Eraser 400
scale    375
color    2500
notebook 750
Total Amount= 4275
```

Awk If Statement

Syntax:

```
if (conditional-expression)
     action1
else
     action2
```

Example

Now let us create the sample input file which has the student marks.

```
$cat>student.txt
1 ajay 50 60 90
2 sunil 20 30 30
3 varun 80 80 70
4 amit 23 67 88
5 tarun 67 68 78
6 deepak 10 30 45
```

Now Create Awk script

```
$cat > studres.awk

{if ($3>=40 && $4>=40 && $5>=40)
{
print $0,"=>","Pass";
}
else
{
print $0,"=>","Fail";
}
}
```

Now execute the studres.awk script

```
$ awk -f studres.awk student.txt
```

Output will be

```
1 ajay 50 60 90 => Pass
2 sunil 20 30 30 => Fail
3 varun 80 80 70 => Pass
4 amit 23 67 88 => Fail
5 tarun 67 68 78 => Pass
```

```
6 deepak 10 30 45 => Fail
```

Awk If Else If ladder

```
if(conditional-expression1)
     action1;
else if(conditional-expression2)
     action2;
else if(conditional-expression3)
     action3;
     .
     .
else
     action n;
```

Create Awk script

```
$cat > studgrade.awk

{total=$3+$4+$5;
avg=total/3
if (avg>=80)
{
grade="A";
}
else if(avg>=70)
{
grade="B";
}
else if(avg>=60)
{
grade="C";
```

```
}
else if(avg>=50)
{
grade="D";
}
else
{
grade="F";
}
print $0,"===>",grade;
}
```

Now execute the studgrade.awk script

```
$ awk -f studgrade.awk student.txt
```

Output will be

```
1 ajay 50 60 90 ===> C
2 sunil 20 30 30 ===> F
3 varun 80 80 70 ===> B
4 amit 23 67 88 ===> D
5 tarun 67 68 78 ===> B
6 deepak 10 30 45 ===> F
```

While loop in awk

Syntax:

```
while(condition)
{
      actions
```

```
}
```

Example

```
$cat > loopdemo.awk
BEGIN{c=1;
While(c<=10)
{
print c;
C++;
}
}


$ awk –f loopdemo.awk
```

Awk For Loop Statement

Awk for statement is same as awk while loop, but it is syntax is much easier to use.

```
Syntax:

for(initialization;condition;increment/decrement)
{
        Actions
}
```

How it works? — **Awk for** statement starts by executing initialization, then checks the condition, if the condition is true, it executes the actions, then increment or decrement.Then as long as the condition is true, it repeatedly executes action and then increment/decrement.

Example

```
$cat > fordemo.awk
BEGIN{
for(i=1;i<=10;i++)
{
```

```
print c;
}
}
```

```
$ awk –f fordemo.awk
```
