AWK Programming

OBJECTIVE: To understand basic concept of AWK Programming.

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 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 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 performs the associated actions.

Awk is abbreviated from the names of the developers – Aho, Weinberger, and Kernighan.

Some of the key features of Awk are:

- Awk views a text file as records and fields.
- Like common programming language, Awk has variables, conditionals and loops
- Awk has arithmetic and string operators.
- Awk can generate formatted reports
- Awk reads from a file or from its standard input, and outputs to its standard output.
- Awk does not get along with non-text files.

Syntax:

```
awk '/search pattern1/ {Actions} /search pattern2/ {Actions}' file_name
```

In the above awk syntax:

- search pattern is a regular expression.
- Actions – statement(s) to be performed.
- several patterns and actions are possible in Awk.
- Filename – Input file.
- Single quotes around program is to avoid shell not to interpret any of its special characters.

Awk Working Methodology

1. Awk reads the input files one line at a time.
2. For each line, it matches with given pattern in the given order, if matches performs the corresponding action.
3. If no pattern matches, no action will be performed.
4. In the above syntax, either search pattern or action are optional, But not both.
5. If the search pattern is not given, then Awk performs the given actions for each line of the input.
6. If the action is not given, print all that lines that matches with the given patterns which is the default action.
7. Empty braces without any action does nothing. It won't perform default printing operation.
8. Each statement in Actions should be delimited by semicolon.

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: NR command keeps a current count of the number of input records. Remember that records are usually lines. Awk command performs the pattern/action statements once for each record in a file.
NF: NF command keeps a count of the number of fields within the current input record.
FS: FS command contains the field separator character which is used to divide fields on the input line. The default is “white space”, meaning space and tab characters. FS can bereassigned to another character (typically in BEGIN) to change the field separator.
RS: RS command 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: OFS command 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: ORS command 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.
To print whole file then you can use below command,

Awk Conditional Operators

Awk has the following list of conditional operators which can be used with control

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

Awk Regular Expression Operator

Operator	Description
~	Match operator
!~	No Match operator

Awk Arithmetic Operators

The following operators are used for performing arithmetic calculations.

Operator	Description
+	Addition
–	Subtraction
*	Multiplicatio n
/	Division
%	Modulo Division

Awk String Operator

For string concatenation Awk has the following operators.

Operator	Description
(space)	String Concatenation

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

Count the total number of fields in a file.

The below awk script, matches all the lines and keeps adding the number of fields in each line,using shortcut addition assignment operator. The number of fields seen so far is kept in a variable named ‘total’. Once the input has been processed, special pattern ‘END {…}’ is executed, which prints the total number of fields.

```
$ cat /etc/passwd

$ awk –F ‘:’  ‘{ total +=  NF };  END {print total}’  /etc/passwd
```

Here we print the username and his home path from /etc/passwd, and surely the separator is specified with capital -F which is the colon.

Read AWK Scripts

To define an awk script, use braces surrounded by single quotation marks like this:

$ awk ‘{print “Welcome for awk lab session”}’

Using Variables

With awk, you can process text files. Awk assigns some variables for each data field found:

- $0 for the whole line.
- $1 for the first field.
- $2 for the second field.
- $n for the nth field.

You can create file using **cat or gedit** command as follows,

```
$cat > example1.txt
```

```
This is a test
This is a second test
This is a third test
This is a fourth test
This is a fifth test
```

To print whole field of above created file,

```
$ awk ‘{print $0}’
example1.txtor
```

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

```
$ awk ‘{print NR,$0}’ example1.txt
```

To print first field of above created file,

```
$ awk ‘{print $1}’ example1.txt
```

Adding Output Field Separators(OFS)

```
$ date
```

```
$ date |awk ' OFS="/" {print $2,$3,$6}'
```

Let us create **employee** file which has the following content, which will be used in theexamples mentioned below. Try awk commands to retrieve data from a file,

```
$gedit employe
```

```
100 Thomas     Manager        Sales         5,0000
200 Jason      Developer      Technology    5,5000
300 Sanjay     Sysadmin       Technology    7,0000
400 Nisha      Manager        Marketing     9,5000
500 Randy      DBA            Technology    6,0000
```

Awk Examples:

Default behaviour of Awk

Awk Example 1:By default Awk prints every line from the file.

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

In the above example pattern is not given. So the actions are applicable to all the lines. We can provide

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

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

AWK commands in a script file as shown ;

Input Field Separators

If you want awk to work with text that doesn't use whitespace to separate fields, you have to tell it which character the text uses as the field separator. For example, the /etc/passwd file uses a colon (:) to separate fields.

We will use that file and the **-F** (separator string) option to tell awk to use the colon (:) as the separator. We type the following to tell awk to print the name of the user account and the home folder:

```
$ awk -F: '{print $1,$6}' /etc/passwd
```

Adding Patterns

If all we're interested in are regular user accounts, we can include a pattern with our print action to filter out all other entries. Because **User ID** numbers are equal to, or greater than, 1,000, we can base our filter on that information.

We type the following to execute our print action only when the third field ($3) contains a value of 1,000 or greater:

```
$ awk -F: '$3 >= 1000 {print $1,$6}' /etc/passwd
```

Reading the script from a file

You can create your awk script in a file and specify that file use in -f option.
Create **testfile1** and include,

```
{
text=$1 " home at "
$6print text
}
```

```
$ awk -F: -f testfile1  /etc/passwd
```

We can use the **BEGIN rule** to provide a title for our little report. We type the following, using the (\n) notation to insert a newline character into the title string:

```
$ awk -F: 'BEGIN {print "User Accounts\n----------- "} $3 >= 1000 {print $1,$6}'
/etc/passwd
```

Awk Example 2. Print the lines which matches with the pattern

```
$ awk '/Thomas|Nisha/' employee.txt
```

Awk Example 3. Print only specific field.

Awk has number of built in variables. 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. NF is a built in variable which represents totalnumber of fields in a record.

1. `$ awk '/Thomas |Nisha/{print $4;}' employee.txt`
2. **`$ awk '{print $2,$5;}' employee.txt`**
3. `$ awk '{print $2,$NF;}' employee.txt`

In the above example $2 ,$4 and $5 represents Name , Department and Salary respectively. We canget the Salary using $NF also, where $NF represents last field. In the print statement „,‟ is a concatenator.

Awk Example 4. Initialization and Final Action

Awk has two important patterns which are specified by the keyword called **BEGIN** and **END**.

Syntax:

```
BEGIN {awk-commands} /pattern/ {awk-commands} END {awk-commands}
```

- The BEGIN block gets executed at program start-up. It executes only once. This is good place to initialize variables. BEGIN is an AWK keyword and hence it must be in upper-case.Note that this block is optional.

- The body block applies AWK commands on every input line. By default, AWK executes commands on every line. We can restrict this by providing patterns. Note that there are nokeywords for the Body block.

- The END block executes at the end of the program. END is an AWK keyword and hence it must be in upper-case. Note that this block is optional.

Examples:

```
$ awk ' BEGIN {print "This is my first lession"} {print $1,$2,$6}' exlab5.txt
```

```
$ awk ' BEGIN {print "This is my first lession"} {print $1,$2,$6} END {print "This is end"}'
exlab5.txt
```

```
$ awk 'BEGIN {print "Name\tDesignation\tDepartment\tSalary";} {print
$2,"\t",$3,"\t",$4,"\t",$NF;} END{print "Report Generated\n------------------ "; }'
employee.txt
```

Awk Example 5. Find the employees who has employee id greater than 200

```
$ awk '$1 >200' employee.txt
```

Awk Example 6. Print the list of employees in Technology Department

```
$ awk '$4 ~/Technology/' employee.txt
```

Operator ~ is for comparing with the regular expressions. If it matches the default action i.e print whole line will be performed.

Awk Example 7. Print number of employees in Technology Department

```
$ awk 'BEGIN { count=0;} $4 ~ /Technology/ { count++; } END { print "Number ofemployees
in Technology Dept =",count;}' employee.txt
```

Using Multiple Commands

To run multiple commands, separate them with a semicolon like this:

```
$ echo "Hello John" | awk ' {$2="Andrew" ; print $0}'
```

Structured Commands

The awk scripting language supports if conditional

statement.The **testfile** contains the following,

```
10

15

6

33

45
```

```
$ awk '{if($1>30) print S1}' testfile
```

You can use else statements like this:

```
$ awk '{ if($1>30) { x=$1 * 3 print x} else { x=$1/2 print x }}' testfile
```

There are some examples are given in below. Try these examples.

1. $**awk** '/a/ {print $2 "\t" $3}' **employee.txt**
$**awk** '/a/{print $0}' **employee.txt**
$**awk** '/a/' **employee.txt**

2. $**awk** 'length($2) > 5' **employee.txt**
AWK provides a built-in **length** function that returns the length of the string. If any value of $2 field has more than 5 characters, then the comparison results true and the line gets printed.

3. echo -e "One Two\nOne Two Three\nOne Two Three Four" **| awk** 'NF > 2'
NF represents the number of fields in the current record. For instance, this example prints only thoselines that contain more than two fields.

4. $ awk 'BEGIN { if (match("One Two Three", "Thre")) { print RSTART } }'
RSTART represents the first position in the string matched by **match** function

5. echo -e "cat\nbat\nfun\nfin\nfan" **| awk** '/f.n/'
It matches any single character except the end of line character. For instance, the above examplematches **fin, fun, fan** etc.

6. echo -e "This\nThat\nThere\nTheir\nthese" **| awk** '/^The/'
It matches the start of line. For instance, this example prints all the lines that start with pattern **The**

7. echo -e "knife\nknow\nfun\nfin\nfan\nnine" **| awk** '/n$/'
It matches the end of line. For instance, the above example prints the lines that end with the letter **n**.

8. echo -e "Call\nTall\nBall" **| awk** '/[CT]all/'
It is used to match only one out of several characters. For instance, the above example matches pattern
Call and **Tall** but not **Ball**.

9. echo -e "Call\nTall\nBall" | **awk** '/[^CT]all/'
In exclusive set, the caret operator (^) negates the set of characters in the square brackets. For instance, the above example prints only **Ball**.

10. $ echo -e "Call\nTall\nBall\nSmall\nShall" | **awk** '/Call|Ball/'
A vertical bar allows regular expressions to be logically OR . For instance, this example prints **Ball** and **Call**.

11. echo -e "Colour\nColor" | **awk** '/Colou?r/'

It matches zero or one occurrence of the preceding character. For instance, the above examplematches **Colour** as well as **Color**. We have made **u** as an optional character by using **?**.

12. $ echo -e "ca\ncat\ncatt" | **awk** '/cat*/'
It matches zero or more occurrences of the preceding character. For instance, the above examplematches **ca, cat, catt,** and so on.

13. echo -e "Apple Juice\nApple Pie\nApple Tart\nApple Cake" | **awk** '/Apple (Juice|Cake)/'

Parentheses () are used for grouping and the character | is used for alternatives. For instance,this regular expression matches the lines containing either **Apple Juice or Apple Cake**

14. index(in,find)

awk 'BEGIN{print index("peanut","an")}'

This searches the string ***in*** for the first occurrence of the string ***find***, and returns the position in characters where that occurrence begins in the sting ***in***. If ***find*** is not found , index returns 0.