AWK Command

The **awk command** is used for **text processing** in Linux. Although, the sed command is also used for text processing, but it has some limitations, so the awk command becomes a handy option for text processing. It provides powerful control to the data.

The Awk is a powerful scripting language used for **text scripting**. It searches and replaces the texts and sorts, validates, and indexes the database.

It is one of the most widely used tools for the programmer, as they write the scaled-down effective program in the form of a statement to define the text patterns and designs.

It acts as a **filter in Linux**. It is also referred as **gawk (GNU awk)** In Linux.

How is it named as AWK?

This command is named by using the first letter of the name of three people who wrote the original version of this command in 1977. Their names are **Alfred Aho, Peter Weinberger,** and **Brian Kernighan** and they were from **AT & T Bell Laboratories.**

Features of AWK command

Various features of the Awk command are as follows:

- It scans a file line by line.
- It splits a file into multiple fields.
- It compares the input text or a segment of a text file.
- It performs various actions on a file like searching a specified text and more.
- It formats the output lines.
- It performs arithmetic and string operations.
- It applies the conditions and loops on output.
- It transforms the files and data on a specified structure.
- It produces the format reports.

Syntax:

awk options 'selection _criteria {action }' input-file > output-file

The options can be:

- **-f program files:** It reads the source code of the script written on the awk command
- **-F fs:** It is used as the input field separator.
-

general syntax of a running an **Awk** command is:

```
# awk 'script' filenames
```

And in the syntax above, the **Awk** script has the form:

```
/pattern/ { actions }
```

When you consider the pattern in the script, it is normally a regular expression, additionally, you can also think of pattern as special patterns `BEGIN` and `END`. Therefore, we can also write an **Awk** command in the form below:

```
awk '

          BEGIN { actions }

          /pattern/ { actions }

          /pattern/ { actions }

         ……….

           END { actions }

' filenames
```

In the event that you use the special patterns: `BEGIN` and `END` in an **Awk** script, this is what each of them means:

- **BEGIN pattern**: means that Awk will execute the action(s) specified in **BEGIN** once before any input lines are read.

- **END pattern**: means that Awk will execute the action(s) specified in **END** before it actually exits.

And the flow of execution of the an **Awk** command script which contains these special patterns is as follows:

- When the `BEGIN` pattern is used in a script, all the actions for **BEGIN** are executed once before any input line is read.
- Then an input line is read and parsed into the different fields.

- Next, each of the non-special patterns specified is compared with the input line for a match, when a match is found, the action(s) for that pattern are then executed. This stage will be repeated for all the patterns you have specified.
- Next, stage 2 and 3 are repeated for all input lines.
- When all input lines have been read and dealt with, in case you specify the END pattern, the action(s) will be executed.

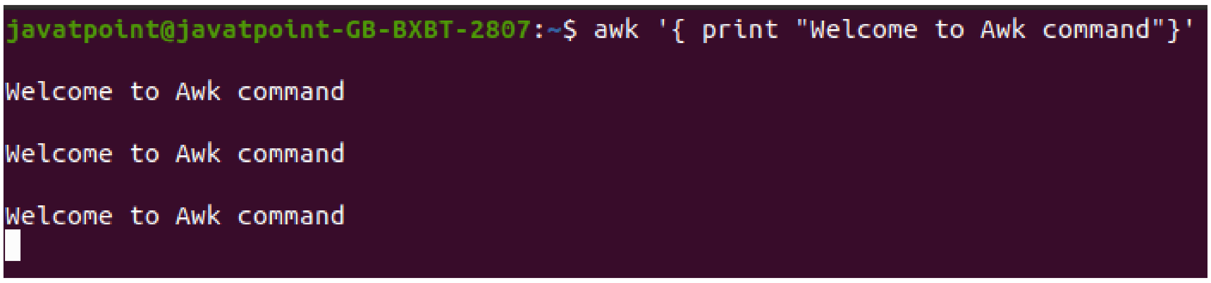
You should always remember this sequence of execution when working with the special patterns to achieve the best results in an **Awk** operation.

How to define AWK Script?

To define the awk script, use the awk command followed by curly braces {} surrounded by single quotation mark '' as follows:

```
awk '{ print "Welcome to Awk command"}'
```

The above command will print the inputted string every time we execute the command. Press **CTRL+D** key to terminate the program. Consider the below output:



```
javatpoint@javatpoint-GB-BXBT-2807:~$ awk '{ print "Welcome to Awk command"}'

Welcome to Awk command

Welcome to Awk command

Welcome to Awk command
```

AWK Command Examples

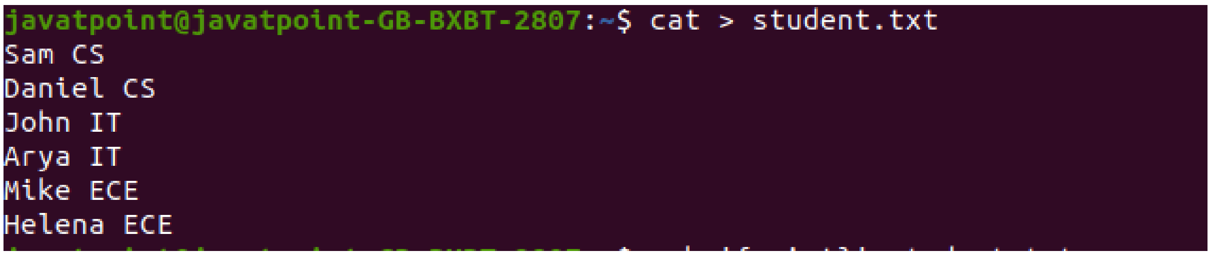
To better understand the Awk command, have a look at the below example:

Let's create a data to apply the various awk operations. Consider student data from different streams.

To create data, execute the cat command as follows:

```
cat > student.txt
Sam CS
Daniel IT
John IT
Arya IT
Mike ECE
Helena ECE
```

Press **CTRL + D** key to save the file and **ESC** key to exit from the command-line editor. It will create data. Consider the below output:


```
javatpoint@javatpoint-GB-BXBT-2807:~$ cat > student.txt
Sam CS
Daniel CS
John IT
Arya IT
Mike ECE
Helena ECE
```

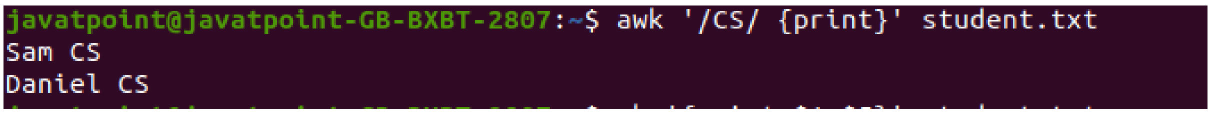
A student data has been created, and we will operate the awk command on this data.

Example1: List students with the specified pattern.

Consider the below command:

```
awk '/ CS/ {print} ' student.txt
```

Output:


```
javatpoint@javatpoint-GB-BXBT-2807:~$ awk '/CS/ {print}' student.txt
Sam CS
Daniel CS
```

Example2: Default behaviour of awk command.

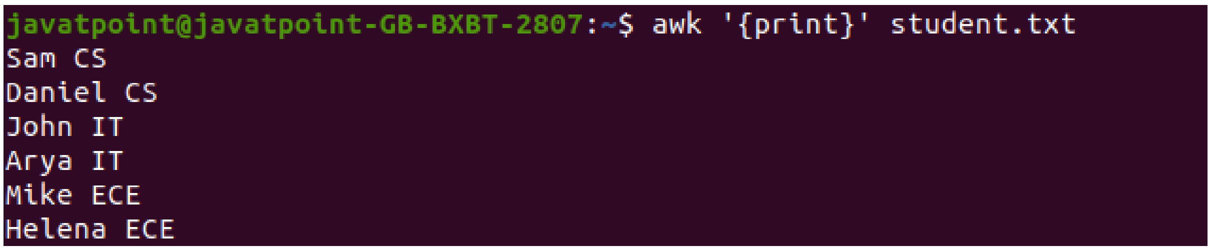
If we do not specify the pattern, it will show all of the content of the file.

Consider the below command:

```
awk '{print}' student.txt
```

We have not specified any pattern in the above command so, it will display all lines of the file.

Output:


```
javatpoint@javatpoint-GB-BXBT-2807:~$ awk '{print}' student.txt
Sam CS
Daniel CS
John IT
Arya IT
Mike ECE
Helena ECE
```

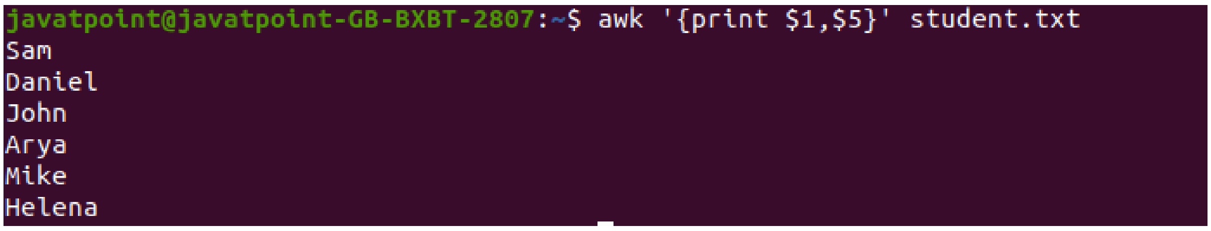
Example3: Print the specified column.

If we specify the column number on this command, it will print that line only. Consider the below output:

```
awk '{print $1,$5} student.txt
```

The above command will print the column number 1 and 5. If column 5 does not exist in the file system, it will only print column number 1.

Consider the below output:

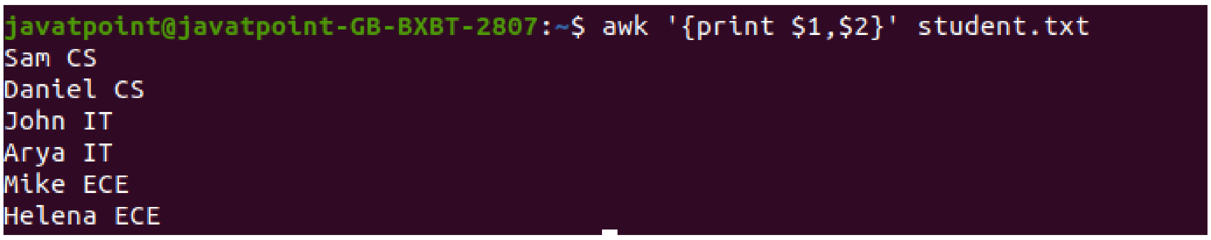


```
javatpoint@javatpoint-GB-BXBT-2807:~$ awk '{print $1,$5}' student.txt
Sam
Daniel
John
Arya
Mike
Helena
```

Consider the below command:

```
awk '{print $1,$2}' student.txt
```

The above command will list the column number 1 & 2. Consider the below output:



```
javatpoint@javatpoint-GB-BXBT-2807:~$ awk '{print $1,$2}' student.txt
Sam CS
Daniel CS
John IT
Arya IT
Mike ECE
Helena ECE
```

Built-in variables in AWK command

Awk command supports many built-in variables, which include $1, $2, and so on, that break the file content into individual segments.

NR: It is used to show the current count of the lines. The awk command performs action once for each line. These lines are said as records.

NF: It is used to count the number of fields within the current database.

FS: It is used to create a field separator character to divide fields into the input lines.

OFS: It is used to store the output field separator. It separates the output fields.

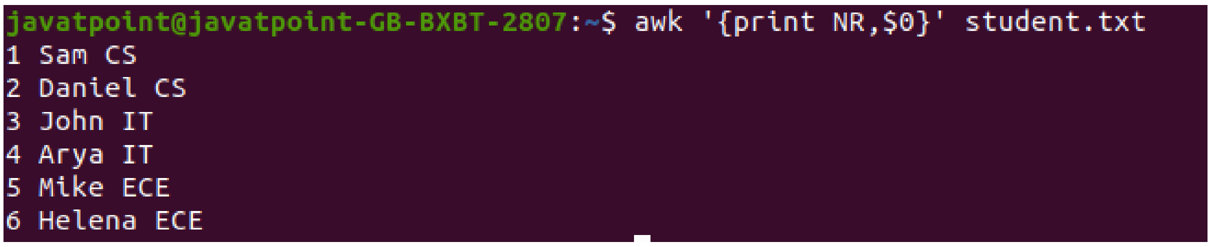
ORS: It is used to store the output record separator. It separates the output records. It prints the content of the ORS command automatically.

Example4: Print the output and display the line number.

To display the line number in output, use the NR variable with the Awk command as follows:

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

Consider the below output:



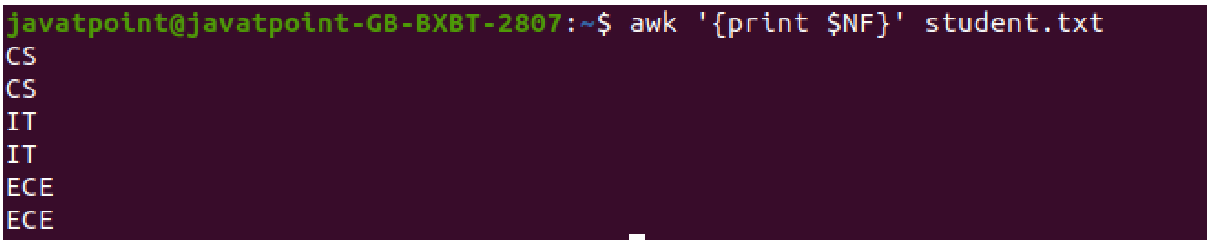
```
javatpoint@javatpoint-GB-BXBT-2807:~$ awk '{print NR,$0}' student.txt
1 Sam CS
2 Daniel CS
3 John IT
4 Arya IT
5 Mike ECE
6 Helena ECE
```

Example5: Print the last field of the file.

To display the last field of the file, execute the NF variable with the Awk command as follows:

```
awk '{print $NF}' student.txt
```

Consider the below output:



```
javatpoint@javatpoint-GB-BXBT-2807:~$ awk '{print $NF}' student.txt
CS
CS
IT
IT
ECE
ECE
```

Example6: Separate the output in the specified format.

To separate the output by a '-' **symbol or (:) semicolon**, specify it with ORS command as follows:

```
awk 'BEGIN { ORS ="-"} {print $0}' student.txt
```

The above command will separate the output by an underscore (_) symbol. Consider the below output:

```
Sam CS -Daniel CS-John IT-Arya IT-Mike ECE-
```

Example7: Print the square of the numbers from 1 to 8.

To print the numbers from 1 to 8, execute below command:

```
awk 'BEGIN { for(i=1;i<=8;i++) print "square of", i, "is",i*i; }'
```

The above command will print the square of 1 to 8. consider the below output:

```
square of 1 is 1
square of 2 is 4
square of 3 is 9
square of 4 is 16
square of 5 is 25
square of 6 is 36
square of 7 is 49
square of 8 is 64
```

Example8: Calculate the sum of a particular column.

Let's create a data to apply the sum operation on a column. To create students marks data, execute the cat command as follows:

```
 cat >  StudentMarks
Name, Marks, Max marks
Sam,75,100
```

Daniel,80,100
John,74,100
Arya,85,100
Mike,70,100
Helena,74,100

Press **CTRL+D** to save the file. We have successfully created StudentsMarks data. We can check it by executing the cat command as follows:

```
cat StudentMarks
```

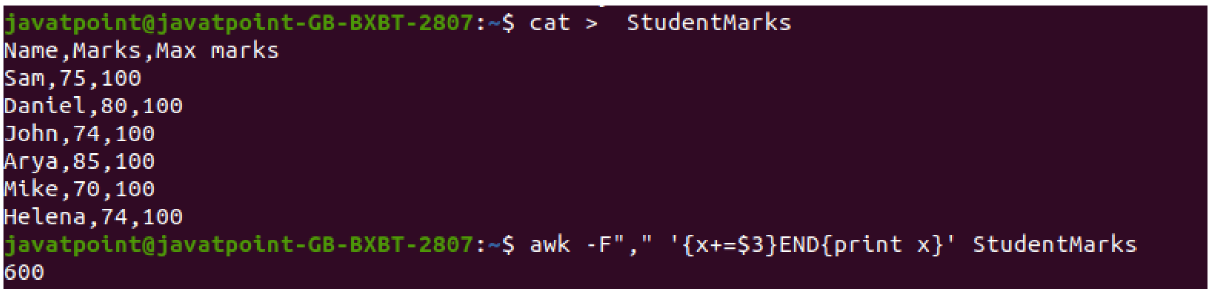
To calculate the third column of the created data, execute the below command:

```
awk -F"," '{x+=$3}END{print x}' StudentMarks
```

Output:

```
600
```

Consider the below output:



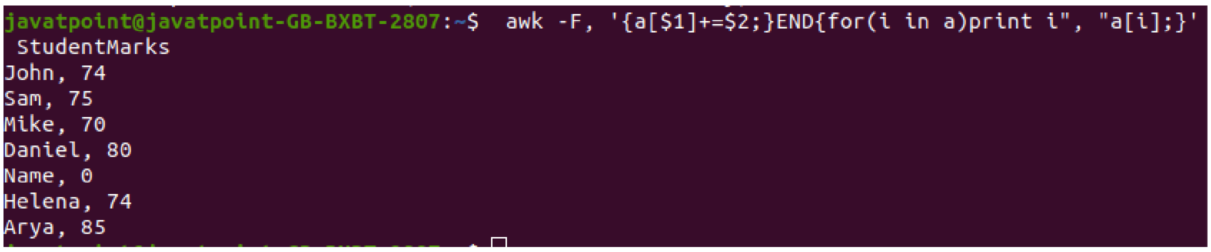
```
javatpoint@javatpoint-GB-BXBT-2807:~$ cat >  StudentMarks
Name,Marks,Max marks
Sam,75,100
Daniel,80,100
John,74,100
Arya,85,100
Mike,70,100
Helena,74,100
javatpoint@javatpoint-GB-BXBT-2807:~$ awk -F"," '{x+=$3}END{print x}' StudentMarks
600
```

Example9: Find some of the individual records.

To print some of the individual student marks record, execute the below command:

```
awk -F, '{a[$1]+=$2;}END{for(i in a)print i", "a[i];}' StudentMarks
```

The above command will print the individual's name with his marks from the StudentMarks file. Consider the below output:



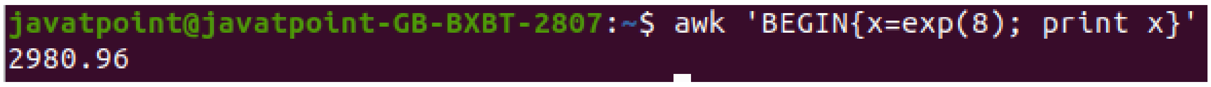
```
javatpoint@javatpoint-GB-BXBT-2807:~$  awk -F, '{a[$1]+=$2;}END{for(i in a)print i", "a[i];}'
 StudentMarks
John, 74
Sam, 75
Mike, 70
Daniel, 80
Name, 0
Helena, 74
Arya, 85
```

Example10: Find the value of exp 8.

To find the value of exp 8, execute the below command:

```
awk 'BEGIN{x=exp(8); print x}'
```

The above command will print the value of exp 8. consider the below output:



```
javatpoint@javatpoint-GB-BXBT-2807:~$ awk 'BEGIN{x=exp(8); print x}'
2980.96
```

Consider the following text file as the input file for all cases below:

$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

1. Default behavior of Awk: By default Awk prints every line of data from the specified file.
$ awk '{print}' employee.txt

Output:
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, no pattern is given. So the actions are applicable to all the lines. Action print without any argument prints the whole line by default, so it prints all the lines of the file without failure.

2. Print the lines which match the given pattern.
$ awk '/manager/ {print}' employee.txt

Output:
ajay manager account 45000

varun manager sales 50000

amit manager account 47000

In the above example, the awk command prints all the line which matches with the ‘manager’.

3. Splitting a Line Into Fields : For each record i.e line, the awk command 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 respectively. Also, $0 represents the whole line.

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

Output:

```
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 fields respectively.

Examples:

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

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

Output:

```
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, the awk command with NR prints all the lines along with the line number.

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

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

Output:

```
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.

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

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

Output:

```
3 varun manager sales 50000

4 amit manager account 47000

5 tarun peon sales 15000

6 deepak clerk sales 23000
```

More Examples

For the given text file:

```
$cat > geeksforgeeks.txt


A    B    C

Tarun    A12    1

Man    B6    2

Praveen    M42    3
```

1) To print the first item along with the row number(NR) separated with ” – “ from each line in geeksforgeeks.txt:

```
$ awk '{print NR "- " $1 }' geeksforgeeks.txt

1 - A

2 - Tarun

3 – Manav

4 - Praveen
```

2) To return the second column/item from geeksforgeeks.txt:

The question should be:- To return the second column/item from geeksforgeeks.txt:

```
$ awk '{print $2}' geeksforgeeks.txt

B

A12

B6

M42
```

3) To print any non empty line if present

```
$ awk 'NF < 0' geeksforgeeks.txt
```

here NF should be 0 not less than and the user have to print the line number also:

correct answer : awk ‘NF == 0 {print NR}’ geeksforgeeks.txt

OR

awk ‘NF <= 0 {print NR}’ geeksforgeeks.txt

```
0
```

4) To find the length of the longest line present in the file:

```
$ awk '{ if (length($0) > max) max = length($0) } END { print max }' geeksforgeeks.txt
```

```
13
```

5) To count the lines in a file:

```
$ awk 'END { print NR }' geeksforgeeks.txt
```

```
3
```

6) Printing lines with more than 10 characters:

```
$ awk 'length($0) > 10' geeksforgeeks.txt
```

```
Tarun    A12    1

Praveen    M42    3
```

7) To find/check for any string in any specific column:

```
$ awk '{ if($3 == "B6") print $0;}' geeksforgeeks.txt
```