**AWK Commands in Linux**

Reference: <https://www.geeksforgeeks.org/awk-command-unixlinux-examples/>

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

WHAT CAN WE DO WITH AWK?

1. AWK Operations:

(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

Syntax:

awk options 'selection \_criteria {action }' input-file

Options:

-f program-file : Reads 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

**Sample Commands :-**

Example:

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.

**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 be reassigned 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.

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

**8)** To print the squares of first numbers from 1 to n say 6:

$ awk 'BEGIN { for(i=1;i<=6;i++) print "square of", i, "is",i\*i; }'

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

**Some other examples:**

Awk is a powerful programming language and command-line utility in Unix and Linux environments. It is primarily used for text processing and manipulation. Here are some commonly used Awk commands and features:

1. Basic Syntax:

```bash

awk 'pattern { action }' filename

```

- `pattern`: Specifies a condition to match lines.

- `{ action }`: Specifies the action to be performed if the pattern is matched.

2. Printing Columns:

- Print the first column:

```bash

awk '{ print $1 }' filename

```

- Print multiple columns:

```bash

awk '{ print $1, $3 }' filename

```

3. Setting Field Separator (FS):

- By default, Awk uses whitespace as the field separator. You can change it using the `-F` option:

```bash

awk -F',' '{ print $1 }' filename

```

4. Conditional Statements:

- Use `if` statements for conditional processing:

```bash

awk '{ if ($3 > 50) print $1 }' filename

```

5. Pattern Matching:

- Match lines containing a specific pattern:

```bash

awk '/pattern/ { print $0 }' filename

```

- Negate the match:

```bash

awk '!/pattern/ { print $0 }' filename

```

6. Built-in Variables:

- `NF`: Number of fields in the current line.

- `NR`: Current record (line) number.

- `FNR`: Current record number in the current file.

Example:

```bash

awk '{ print NF, NR, FNR }' filename

```

7. Mathematical Operations:

- Perform arithmetic operations on columns:

```bash

awk '{ total = $2 + $3; print total }' filename

```

8. Formatting Output:

- Use `printf` for formatted printing:

```bash

awk '{ printf "Name: %-10s Score: %d\n", $1, $2 }' filename

```

9. User-Defined Functions:

- Define and use functions within Awk:

```bash

awk 'function myFunc(x) { return x 2 } { print myFunc($2) }' filename

```

10. Examples with Input Redirection:

- Read input from a pipeline or a file:

```bash

cat data.txt | awk '{ print $1 }'

awk '{ print $1 }' < data.txt

```

These are just some basic examples of what Awk can do. Awk is a versatile tool with many advanced features and functions, making it an excellent choice for text processing tasks in the Unix/Linux command line.