

Program- BTech 3<sup>rd</sup> Semester  
 Course Code- CSET213  
 Year- 2024  
 Date- 11/11/2024

Type- Sp. Core-I  
 Course Name-Linux and Shell Programming  
 Semester- Odd  
 Batch- Cyber Security (B9-16)

## Lab Assignment 11

Exp No	Name	CO1	CO2	CO3
11	AWK commands and its application	✓	✓	-

**Objective:** To learn and use awk commands for data processing.

**Outcomes:** After hands-on on awk commands, the student can understand that without using any specialized GUI analytical tool like Microsoft BI for analyzing data and reporting, the same can be done using awk commands.

### Hands-on Learning on awk commands (1HR 35 MINUTES)

- **Why awk?**

In contrast to other programming languages which are procedural, awk is data-driven, which means that you define a set of actions to be performed against the input data. It takes the input data, transforms it according to the specified action by you and sends the result to the standard output. Awk is one of the most powerful tools for text manipulation.

- **What is awk?**

Awk is a general-purpose scripting language especially designed for advanced text processing. It is generally used as a reporting and analysis tool.

- **How awk functions?**

There are several different implementations of awk like mawk, awka, pawk, fawk etc. Here, we will use the GNU implementation of awk called gawk. In most of the Linux systems, the awk interpreter is just a symlink to gawk.

- **Records and fields:** Awk can process textual data files and streams. The input data is divided into records and fields. Awk operates on one record at a time until the end of the input is reached. Records are separated by a character called the record separator. The default record separator is the newline character, which means that each line in the text data is a record. A new record separator can be set using the **RS** variable.

Records consist of fields which are separated by the field separator. By default, fields are separated by a whitespace, including one or more tab, space, and newline characters.

The fields in each record are referenced by the dollar sign (\$) followed by field number, beginning with 1. The first field is represented with \$1, the second with \$2, and so on. The last field can also be referenced with the special variable \$NF. The entire record can be referenced with \$0.

Below figure illustrated how to reference records and fields:

```

tmpfs      788M  1.8M  786M   1% /run/lock
/dev/sda1  234G  191G   31G  87% /
|-----| |--| |--| |--| |-| |-----|
$1        $2    $3     $4    $5   $6 ($NF) --> fields
|-----|                               |
          $0                           --> record
  
```

- Awk Program

To process a text with awk, you write a program that tells the command what to do. The program consists of a series of rules and user defined functions. Each rule contains one pattern and action pair. Rules are separated by newline or semi-colons (;). Typically, an awk program looks like this:

```
pattern { action }
pattern { action }
...
...
```

When awk processes data, if the pattern matches the record, it performs the specified action on that record. When the rule has no pattern, all records (lines) are matched.

An awk action is enclosed in braces ({} ) and consists of statements. Each statement specifies the operation to be performed. An action can have more than one statement separated by newline or semi-colons (;). If the rule has no action, it defaults to printing the whole record.

Awk supports different types of statements, including expressions, conditionals, input, output statements, and more. The most common awk statements are:

- exit - Stops the execution of the whole program and exits.
- next - Stops processing the current record and moves to the next record in the input data.
- print - Print records, fields, variables, and custom text.
- printf - Gives you more control over the output format, similar to C and bash

When writing awk programs, everything after the hash mark (#) and until the end of the line is considered to be a comment. Long lines can be broken into multiple lines using the continuation character, backslash (\).

- **Executing awk programs**

An awk program can be run in several ways. If the program is short and simple, it can be passed directly to the awk interpreter on the command-line:

```
awk 'program' input-file...
```

When running the program on the command-line, it should be enclosed in single quotes ("), so the shell doesn't interpret the program.

If the program is large and complex, it is best to put it in a file and use the **-f** option to pass the file to the awk command:

```
awk -f program-file input-file...
```

In the examples below, we will use a file named “primeMinisters.txt” that looks like the one below:

AtalBihari	Vajpayee	60	22	0.732
HDDeve	Gowda	58	24	0.707
InderKumar	Gujral	51	31	0.622
AtalBihari	Vajpayee	49	33	0.598
Manmohan	Singh	48	34	0.585
Narendra	Modi	23	21	0.897

- **Awk Patterns**

Patterns in awk control whether the associated action should be executed or not. Awk supports different types of patterns, including regular expression, relation expression, range, and special expression patterns. When the rule has no pattern, each input record is matched. Here is an example of a rule containing only an action:

```
awk '{ print $3 }' primeMinisters.txt
```

The program will print the third field of each record:

60
58
51
49
48
23

- **Regular expression patterns**

A regular expression or regex is a pattern that matches a set of strings. Awk regular expression patterns are enclosed in slashes (/):

```
/regex pattern/ { action }
```

The most basic example is a literal character or string matching. For example, to display the first field of each record that contains “0.5” you would run the following command:

```
awk '/0.5/ { print $1 }' primeMinisters.txt
```

The program will print:

AtalBihari
Manmohan

The pattern can be any type of extended regular expression. Here is an example that prints the first field if the record starts with two or more digits:

```
awk '/^([0-9]{2,})/ { print $1 }' test.txt
```

- **Relational expressions patterns**

The relational expressions patterns are generally used to match the content of a specific field or variable.

By default, regular expressions patterns are matched against the records. To match a regex against a field, specify the field and use the “contain” comparison operator (~) against the pattern.

For example, to print the first field of each record whose second field contains “ee” you would type:

```
awk '$2 ~ /ee/ { print $1 }' primeMinisters.txt
```

Output:

```
AtalBihari
AtalBihari
```

To match fields that do not contain a given pattern use the !~ operator:

```
awk '$2 !~ /ee/ { print $1 }' primeMinisters.txt
```

Output:

```
HDDeve
InderKumar
Manmohan
Narendra
```

You can compare strings or numbers for relationships such as, greater than, less than, equal, and so on. The following command prints the first field of all records whose third field is greater than 50:

```
awk '$3 > 50 { print $1 }' primeMinisters.txt
```

- **Range Patterns**

Range patterns consist of two patterns separated by a comma:

```
pattern1, pattern2
```

All records starting with a record that matches the first pattern until a record that matches the second pattern are matched.

Here is an example that will print the first field of all records starting from the record including “HDDeve” until the record including “Narendra”:

```
awk '/HDDeve/,/Narendra/ { print $1 }' primeMinisters.txt
```

Output:

HDDeve  
InderKumar  
AtalBihari  
Manmohan  
Narendra

The patterns can also be relation expressions. The command below will print all records starting from the one whose fourth field is equal to 32 until the one whose fourth field is equal to 33:

```
awk '$4 == 32, $4 == 33 { print $0 }' primeMinisters.txt
```

**Note:** Range patterns cannot be combined with other pattern expressions.

- **Special expression patterns**

Awk includes the following special patterns:

BEGIN - Used to perform actions before records are processed.  
END - Used to perform actions after records are processed.

The BEGIN pattern is generally used to set variables and the END pattern to process data from the records such as calculation.

The following example will print “Start Processing.”, then print the third field of each record and finally “End Processing.”:

```
awk 'BEGIN { print "Start Processing." }; { print $3 }; END { print "End Processing." }' primeMinisters.txt
```

If a program has only a BEGIN pattern, actions are executed, and the input is not processed. If a program has only an END pattern, the input is processed before performing the rule actions.

**Note:** The Gnu version of awk also includes two more special patterns BEGINFILE and ENDFILE, which allows you to perform actions when processing files.

- **Combining Patterns**

Awk allows you to combine two or more patterns using the logical AND operator (`&&`) and logical OR operator (`||`).

Here is an example that uses the `&&` operator to print the first field of those record whose third field is greater than 50 and the fourth field is less than 30:

```
awk '$3 > 50 && $4 < 30 { print $1 }' primeMinisters.txt
```

## Scripting Problems for

### Assessment (40 Minutes)

Use the given file primeMinisters.txt and write the awk commands for the following:

1. Print the first field of records that contain the substring 0.7.

2. Print all records that start with a digit.
3. Print only the lines containing “Narendra” anywhere in the record.
4. Print the second field of records containing the string ee in any field.
5. Print the first and third fields for all records that have decimal values (e.g., 0.732, 0.707, etc.) in any field.

Use comparison operators (>, <, ==, !=, etc.) in awk:

1. Print the first field of all records whose third field > 50.
2. Print the complete record where the third field < 25.
3. Print the name (first field) if the third field equals 22.
4. Print the first and second fields of records where fourth field <= 30.
5. Print only those records where the third field is between 20 and 50 (inclusive).

1. Print all records from “HDDeve” to “Narendra”.
2. Print all records between those whose third field equals 22 and 33.
3. Display only the first field for records from “AtalBihari” up to “Manmohan”.

### Submission Instructions:

1. Submission requires the screen shots of all the incurred steps to execute a shell script or a video showing the whole process.
2. All these files are in single zip folder.
3. Use the naming convention: Prog\_CourseCode\_RollNo\_LabNo.docx (Example: BCA3rdSem\_CBCA221\_E21BCA002\_Lab1.1)
4. Submission is through LMS only