School of Computer Science and Engineering

Program- BTech-3 rd 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

Lab Assignment 11.1

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, frawk 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 process 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][0-9]/ { 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":

```
dra/ { 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 pattens:

  ```
  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
```

- **Built-in Variables**

  Awk has a number of built-in variables that contain useful information and allows you to control how the program is processed. Below are some of the most common built-in Variables:

NF	The number of fields in the record.
NR	The number of the current record.
FILENAME	The name of the input file that is currently processed.
FS	Field Separator
RS	Record Separator
OFS	Output field separator

ORS	Output record separator

Here is an example showing how to print the file name and the number of lines (records):

```
awk 'END { print "File", FILENAME, "contains", NR, "lines." }' primeMinisters.txt
```

Variables in AWK can be set at any line in the program. To define a variable for the entire program, put it in a BEGIN pattern.

- **Changing the Field and Record Separator**
  The default value of the field separator is any number of space or tab characters. It can be changed by setting in the FS variable.

  For example, to set the field separator to **"."** you would use:

  ```
  awk 'BEGIN { FS = "." } { print $1 }' primeMinisters.txt
  ```

  The field separator can also be set to more than one characters:

  ```
  awk 'BEGIN { FS = ".." } { print $1 }' primeMinisters.txt
  ```

  When running awk one-liners on the command-line, you can also use the -F option to change the field separator:

  ```
  awk -F "." '{ print $1 }' primeMinisters.txt
  ```

  By default, the record separator is a newline character and can be changed using the RS variable.

  Here is an example showing how to change the record separator to ".”

  ```
  awk 'BEGIN { RS = "." } { print $1 }' primeMinisters.txt
  ```

- **Awk Actions**
  Awk actions are enclosed in braces ({}) and executed when the pattern matches. An action can have zero or more statements. Multiple statements are executed in the order they appear and must be separated by newline or semi-colons (;).

  There are several types of action statements that are supported in awk:

  - Expressions, such as variable assignment, arithmetic operators, increment, and decrement operators.
  - Control statements, used to control the flow of the program (if, for, while, switch, and more)
  - Output statements, such as print and printf.
  - Compound statements, to group other statements.
  - Input statements, to control the processing of the input.
  - Deletion statements, to remove array elements.

  The print statement is probably the most used awk statement. It prints a formatted output of text, records, fields, and variables.

  When printing multiple items, you need to separate them with commas. Here is an example:

```
awk '{ print $1, $3, $5 }' primeMinisters.txt
```

Note: The printed items are separated by single spaces.

If you don't use commas, there will be no space between the items:

```
awk '{ print $1 $3 $5 }' primeMinisters.txt
```

Note: The printed items will be concatenated.

When print is used without an argument, it defaults to print $0. The current record is printed.

To print a custom text, you must quote the text with double-quote characters:

```
awk '{ print "The first field:", $1}' primeMinisters.txt
```

You can also print special characters such as newline:

```
awk 'BEGIN { print "Greetings\nHello\nHow are you?" }'
```

The printf statement gives you more control over the output format. Here is an example that inserts line numbers:

```
awk '{ printf "%3d. %s\n", NR, $0 }' primeMinisters.txt
```

Note: printf doesn't create a newline after each record, so we are using **\n**

The below command calculates the sum of the values stored in the third field in each line:

```
awk '{ sum += $3 } END { printf "%d\n", sum }' primeMinisters.txt
```

Here is another example showing how to use expressions and control statements to print the squares of numbers from 2 to 9:

```
awk 'BEGIN { i = 2; while (i < 10) { print "Square of", i, "is", i*i; ++i } }'
```

One-line commands such as the one above is harder to understand and maintain. When writing longer programs, you should create a separate program file like example.awk and the code is mentioned below:

```
BEGIN {
  i = 2
  while (i < 10) {
    print "Square of", i, "is", i*i;
    ++i
  }
}
```

Run the program by passing the file name to the awk interpreter:

```
awk -f example.awk
```

You can also run an awk program as an executable by using the shebang directive and setting the awk interpreter:

```
#!/usr/bin/awk -f

BEGIN {
  i = 2
  while (i < 10) {
    print "Square of", i, "is", i*i;
    ++i
  }
}
```

Save the file and make it executable :

```
chmod +x example.awk
```

You can now run the program by entering:

```
./example.awk
```

- **Using Shell Variables in Awk Programs**

  If you are using the awk command in shell scripts, the chances are that you'll need to pass a shell variable to the awk program. One option is to enclose the program with double instead of single quotes and substitute the variable in the program. However, this option will make your awk program more complex as you'll need to escape the awk variables.

  The recommended way to use shell variables in awk programs is to assign the shell variable to an awk variable. Here is an example:

```
awk -v n="$num" 'BEGIN {print n}'
```

Problems for Assessment (35 Minutes)

1. Develop an awk command which can: (15 Minutes)
   a. Print the lines and line number in the given input file
   b. Count the number of lines in a file that do not contain vowels
   c. Demonstrate user defined functions and system command
   d. Print first field and second field only if third field value is >=50 in the given input file. (Input field separator is ":" and output field separator is ",")

2. Consider the studentMidtermMarks.csv is a file that contains one record per line(comma separate fields) of the student data in the form of Enrollment Number, Student Name, Information Management Systems marks, Data Structures using C++ marks, Microprocessors and Computer Architecture Marks, Probability and Statistics marks, Software Engineering Marks, Linux and Shell Programming Marks. Implement an awk script to generate result for every student in the form of Enrollment_Number, Student_Name, Total Marks and Result. Result is PASS if marks is >=30 in Linux and shell Programming and Data Structures using C++, and if

marks>=40 in other subjects. Otherwise, the result is failed. (20 Minutes)

Submission Instructions:

1. Submission requires the screen shots of all the incurred steps to execute a shell script.
2. All these files are in single document preferably in pdf format.
3. Use the naming convention: Prog_CourseCode_RollNo_LabNo.docx
4. Submission is through LMS only