© 2014 IJIRT | Volume 1 Issue 5 | ISSN : 2349-6002

AWK Utility

Ravi Sangwan, Pankaj Gupta
Dronacharya College of Engineering

Abstract: **AWK is an interpreted programming language designed for text processing and typically used as a data extraction and reporting tool. It is a standard feature of most Unix-like operating systems. The term AWK refers to a particular program, and to the language you use to tell this program what to do. When we need to be careful, we call the program "the AWK utility" and the language "the AWK language". The term gawk refers to a version of AWK developed as part the GNU project. The purpose of this paper is to explain both the AWK language and how to run the AWK utility.**

INDEXED TERMS: *One-Short, Throw-Away, AWK Programs, Long, comments.*

I. INTRODUCTION

AWK is an interpreted programming language designed for text processing and typically used as a data extraction and reporting tool. It is a standard feature of most Unix-like operating systems. AWK (also written as Awk and AWK) is a utility that enables a programmer to write small 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 comes with most UNIX-based operating systems such as Linux, and also with some other operating systems, such as Windows 95/98/NT. The basic function of AWK is to search files for lines (or other units or text) that contains certain patterns. When a line matches one of the patterns, AWK performs specified actions on that line. AWK keeps processing input lines in this way until the end of the input files are reached.

Program in AWK are different from programs in most other languages, because AWK programs are data-driven; i.e., you describe the data you wish to work with, and then what to do when you find it. Most other languages are procedural; you have to describe, in great detail, every step the program is to take. When working with procedural languages, it is usually much harder to clearly describe the data your program will process. For this reason, AWK programs are often refreshingly easy to both writer and reader.
When you run AWK, you specify an AWK program that tells AWK what to do. The program consists of a series of rules. Syntactically, a rule consists of a pattern followed by an action. The action is enclosed in curly braces to separate it from the pattern. Rules are usually separated by new lines.

Therefore, an AWK programs looks like this:

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

where *pattern* is typically an expression and *action* is a series of commands. The input is split into records, where by default records are separated by newline characters so that the input is split into lines. The program tests each record against each of the patterns in turn, and executes the *action* for each expression that is true. Either the *pattern* or the *action* may be omitted. The *pattern* defaults to matching every record. The default *action* is to print the record. This is the same pattern-action structure as sed.

II. HOW TO RUN AWK PROGRAMS

There are several ways to run an AWK program. If the program is short it is easy to include in the command that runs AWK, like this:

```
awk 'program' input-file1 input-file2 . . .
```

where program consists of a series of patterns and actions, as described earlier. (The reason for the single quotes is described below; in section One-short throw-away AWK programs.)

When the program is long it is usually more convenient to put in a file and run it with a command like this:

```
awk –f program-file input-file1 input-file2 . . .
```

- **One-Short:** Running a short throw-away AWK program.
- **Read Terminal :** Using no input files (input from terminal instead)
- **Long:** Putting permanent AWK program in files.
- **Executable scripts:** Making self contained AWK programs.

- **Comments:** Adding documentation to gawk programs.

2.1 One-Short Throw-Away AWK Programs

Once you are familiar with AWK, you will often type in simple programs the moment you want to use them. Then you can write the program as the first argument of the AWK command, like this:

awk 'program' input-file1 input-file2 . . .

when program consist of a series of *patterns* and *actions*, as described earlier.

This command format instructs the shell, or command interpreter, to start AWK and use the program to process records in the input files. There are single quotes around *program* so that the shell doesn't interpret any AWK characters as special shell characters. They also cause the shell to treat all of *program* as a single argument for AWK and allow *program* to be more than one line long.

This format is also useful for running short or medium-sized AWK programs for shell scripts, because it avoids the need for separate file for the AWK program. A shell contain shell script is more reliable since there are no other files to be misplaced.

As an interesting side point, the command

awk '/foo/' files . . .

is essentially the same as

egrep foo files . . .

2.2 Running AWK without Input Files

You can run AWK without any input files. If you type the command line:

awk 'programs'

then AWK applies the program to the standard input, which usually means whatever you type on the terminal. This continues until you indicate end of file by typing control-d. (On other operating systems, the end of file character may be different. For example, on OS/2 and MS-DOS, it is control-z.)

For example, the following program prints a friendly piece of advice (from 'Douglas Adams' *The Hitchhiker's Guide to Galaxy*), to keep you from worrying about the complexities of computer programming ('BEGIN' is a feature that we haven't discussed yet).

```
$ awk "BEGIN {print \"Don't Panic! \"}"
-| Don't Panic!
```

This program doesn't read any input. The '\' before each of the inner double quotes is necessary because of the shell's quoting rules, in particular because it mixes both single quotes and double quotes.

This next simple AWK program emulates the cat utility; it copies whatever you type at the keyboard to its standard output.

```
$awk '{ print }'
Now is the time for all good men
-| Now is the time for all good men
to come to the aid of their country
-| to come to the aid of their country
Four score and seven year ago, . . .
-| Four score and seven year ago, . . .
What, me worry?
-| What, me worry?
Control-d
```

2.3 Running Long Programs

Sometimes your AWK programs can be very long. In this case it is more convenient to put the program into a separate file. To tell AWK to use that file for its program, you type:

awk –f source-file input-file1 input-file2 . . .

The '-f' instructs the AWK utility to get the AWK program from the file source file. Any filename can be used for source file. For example, you could put the program:

BEGIN { print "Don't Panic!" }

into the file 'advice'. Then this command:

awk –f advice

does the same thing as this one:

awk "BEGIN {print \"Don't Panic! \" }"

this was explained earlier. Note that you don't usually need single quotes around the file name that you specify with '-f', because most file names don't any of the shell's special character. Notice that in 'advice', the AWK program does not have single quotes around it. The quotes are only needed for programs that are provided on the AWK program line.

If you want to identify your AWK files clearly as such, you can add the extension '.awk' to the file name. This doesn't affect the execution of the AWK program but it does makes "housekeeping" easier.

2.3 Executable AWK Programs

Once you have learned AWK, you may want to write self contained AWK script, using the '#!' script mechanism. You can do this on many UNIX systems (and some day on the GNU systems).

For example, you could update the file 'advice' to look like this:

```
#! /bin/awk -f
BEGIN {print "Don't Panic!" }
```

You should not put more than one argument on the '#!' line after the path to awk. It does not work. The operating system treats the rest of the line as a single argument and passes it to awk. Doing this leads to confusing behavior—most likely a usage diagnostic of some sort from awk.
After making this file executable (with the chmod utility), simply type 'advice' at the shell and the system arranges to run AWK as if you had typed 'awk -f advice':

```
$ chmod +x advice
$ advice
-| Don't Panic!
```

Self-contained awk scripts are useful when you want to write a program that users can invoke without their having to know that the program is written in awk. Some systems limit the length of the interpreter name to 32 characters. Often, this can be dealt with by using a symbolic link. The '#!' mechanism works on GNU/Linux systems, BSD-based systems and commercial UNIX systems.

The colon ensures execution by the standard shell.

```
awk 'program' "&@"
```

Using this technique, it is vital to enclose the program in single quotes to protect it from interpretation by the shell. If you omit the quotes, only a shell wizard can predict the result.

2.4 Comments in AWK Programs

A *comment* is some text that is included in a program for the sake of human readers; it is not really an executable part of the program. Comments can explain what the program does and how it works. Nearly all programming languages have provisions for comments, as programs are typically hard to understand without them.
In the awk language, a comment starts with the sharp sign character ('#') and continues to the end of the line. The '#' does not have to be the first character on the line. The awk language ignores the rest of a line following a sharp sign. For example, we could have put the following into advice:

```
# This program prints a nice friendly message. It helps
# keep novice users from being afraid of the computer.
BEGIN    { print "Don't Panic!" }
```

You can put comment lines into keyboard-composed throwaway awk programs, but this usually isn't very useful; the purpose of a comment is to help you or another person understand the program when reading it at a later time.

CAUTION: As mentioned in One-shot, you can enclose small to medium programs in single quotes, in order to keep your shell scripts self-contained. When doing so, *don't* put an apostrophe (i.e., a single quote) into a comment (or anywhere else in your program). The shell interprets the quote as the closing quote for the entire program. As a result, usually the shell prints a message about mismatched quotes, and if awk actually runs, it will probably print strange messages about syntax errors. For example, look at the following:

```
$ awk '{ print "hello" } # let's be cute'
```

The shell sees that the first two quotes match, and that a new quoted object begins at the end of the command line. It therefore prompts with the secondary prompt, waiting for more input. With Unix awk, closing the quoted string produces this result:

```
$ awk '{ print "hello" } # let's be cute'
> '
error→ awk: can't open file be
error→ source line number 1
```

Putting a backslash before the single quote in 'let's' wouldn't help, since backslashes are not special inside single quotes. The next subsection describes the shell's quoting rules.

2.5 A Very Simple Example

The following command runs a simple awk program that searches the input file `BBS-list' for the string of characters: `foo'. (A string of characters is usually called a string. The term string is perhaps based on similar usage

in English, such as "a string of pearls," or, "a string of cars in a train.")

```
awk '/foo/ { print $0 }' BBS-list
```

When lines containing `foo' are found, they are printed, because `print $0' means print the current line. (Just `print' by itself means the same thing, so we could have written that instead). You will notice that slashes, `/', surround the string `foo' in the awk program. The slashes indicate that `foo' is a pattern to search for. This type of pattern is called a regular expression, and is covered in more detail later (see section Regular Expressions). The pattern is allowed to match parts of words. There are single-quotes around the awk program so that the shell won't interpret any of it as special shell characters.

Here is output of the program:

```
$ awk '/foo/ { print $0 }' BBS-list
-| fooey      555-1234    2400/1200/300    B
-| foot       555-6699    1200/300         B
-| macfoo     555-6480    1200/300         A
-| sabafoo    555-2127    1200/300         C
```

In an awk rule, either the pattern or the action can be omitted, but not both. If the pattern is omitted, then the action is performed for *every* input line. If the action is omitted, the default action is to print all lines that match the pattern. Thus, we could leave out the action (the print statement and the curly braces) in the above example, and the result would be the same: all lines matching the pattern `foo' would be printed. By comparison, omitting the print statement but retaining the curly braces makes an empty action that does nothing; then no lines would be printed.

III. CONCLUSIONS

The basic function of awk is to search files for lines (or other units of text) that contain certain patterns. When a line matches one of the patterns, awk performs specified actions on that line. awk keeps processing input lines in this way until the end of the input files are reached.

Programs in awk are different from programs in most other languages, because awk programs are data-driven; that is, you describe the data you wish to work with, and then what to do when you find it. Most other languages are procedural; you have to describe, in great detail, every step the program is to take. When working with procedural languages, it is usually much harder to clearly describe the data your program will process. For this reason, awk programs are often refreshingly easy to both write and read.

REFERENCES

[1] Pankaj Sharma, System Programming and System Administration, Ghaziabad, 2010.
[2]https://www.gnu.org/software/gawk/manual/html_node/Index.html#Index
[3] http://www.linuxjournal.com/article
[4] http://www.grymoire.com/Unix/Awk.html
[5]http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.31.1299
[6]Hamilton, Naomi (2008-05-27). "The A-Z of Programming Languages: AWK".