Bharati Vidyapeeth (Deemed to be) University)
College of Enginering,Pune.
Department of Computer Engineering

Course Name : Systems Programming

Course Coordinator: Dr. Mrunal Bewoor

UNIT :-V

Operating System
AWK Programming

Outline

- Introduction
- Types
- Uses
- Installation
- Operators
- Functions

Introduction

- The most important text-processing utility on GNU/Linux.
- It is very powerful .
- It uses simple programming language.
- It can solve complex text processing tasks with a few lines of code.
- Starting with an overview of AWK, its environment, and workflow,

Contd..

- It is an interpreted programming language.
- It is very powerful and specially designed for text processing.
- Its name is derived from the family names of its authors − **Alfred Aho, Peter Weinberger, and Brian Kernighan.**
- The version of AWK that GNU/Linux distributes is written and maintained by the Free Software Foundation (FSF); it is often referred to as **GNU AWK.**

Types of AWK

- **AWK** – Original AWK from AT & T Laboratory.
- **NAWK** – Newer and improved version of AWK from AT & T Laboratory.
- **GAWK** – It is GNU AWK. All GNU/Linux distributions ship GAWK.
- It is fully compatible with AWK and NAWK.

Typical Uses of AWK

- The number of tasks can be done with AWK.

  eg
- Process text
- Perform string operations
- Performing arithmetic operations
- Text formatting

What we need to study?

- Syntax
- Variables
- Operators
- Arrays
- Loops
- Functions used in AWK
- Redirection operators

Installation

- Generally, AWK is available by default on most GNU/Linux distributions.
- A command ‘ **which** ‘ is used to check whether it is present on the system or not.
- In case if AWK, is not available then install it on Debian based GNU/Linux using Advance Package Tool **(APT)** package manager as follows −
- [jeryy]$ sudo apt-get update [jeryy]$
- sudo apt-get install gawk
- Similarly, to install AWK on RPM based GNU/Linux, use following Updator Modifier **yum** package manager as follows −
- [root]# yum install gawk After installation, ensure that AWK is accessible via command line.
- [jerry]$ which awk
- On executing the above code, the following result can be see
- /usr/bin/awk

Installation from Source Code

- As GNU AWK is a part of the GNU project, its source code is available
- The source code is free download.
- The following installation is applicable to any GNU/Linux software, and for most other freely-available programs as well

Steps Contd..

- **Step 1** − Download the source code from an authentic place. The command-line utility **wget** serves this purpose.
- [jerry]$ wget http://ftp.gnu.org/gnu/gawk/gawk-4.1.1.tar.xz
- **Step 2** − Decompress and extract the downloaded source code.
- [jerry]$ tar xvf gawk-4.1.1.tar.xz
- **Step 3** − Change into the directory and run configure.
- [jerry]$ ./configure
- **Step 4** − Upon successful completion, the **configure** generates Makefile. To compile the source code, issue a **make** command.
- [jerry]$ make
- **Step 5** − You can run the test suite to ensure the build is clean. This is an optional step.
- [jerry]$ make check
- **Step 6** − Finally, install AWK. Make sure you have super-user privileges.
- [jerry]$ sudo make install

- **Program Structure**
- Let us now understand the program structure of AWK.
- **BEGIN block**
- The syntax of the BEGIN block is as follows –
- **Syntax**
- BEGIN {awk-commands}. The BEGIN block gets executed at program start-up. It executes only once. BEGIN is an AWK keyword and hence it must be in upper-case.

Contd..

- **Body Block**
- The syntax of the body block is as follows –
- **Syntax**
- /pattern/ {awk-commands} The body block applies AWK commands on every input line. By default, AWK executes commands on every line. We can restrict this by providing patterns. Note that there are no keywords for the Body block.

Contd..

- **END Block**
- The syntax of the END block is as follows –
- **Syntax**
- END {awk-commands} The END block executes at the end of the program. END is an AWK keyword and hence it must be in upper-case. Please note that this block is optional.
- Let us create a file **marks.txt** which contains the serial number, name of the student, subject name, and number of marks obtained.

1) Amit Physics 80

2) Rahul Maths 90

3) Shyam Biology 87

4) Kedar English 85

5) Hari History 89

Let us now display the file contents with header by using AWK script.

- **Example**
- [jerry]$ awk 'BEGIN{printf "Sr No\tName\tSub\tMarks\n"} {print}' marks.txt
  After execution of the code , it gives the following

- **Output**
- Sr No Name Sub Marks
- 1) Amit Physics 80
- 2) Rahul Maths 90
- 3) Shyam Biology 87
- 4) Kedar English 85
- 5) Hari History 89

Contd..

- At the start, AWK prints the header from the BEGIN block. Then in the body block, it reads a line from a file and executes AWK's print command which just prints the contents on the standard output stream. This process repeats until file reaches the end.

Standard Variables

ARGC

- It implies the number of arguments provided at the command line.

- [jerry]$ awk 'BEGIN {print "Arguments =", ARGC}' One Two Three Four
- After execution of this code the result will be
- arguments will be displayed as 5

Contd..

- ARGV- Array that stores the command line arguments
- **CONVFMT -**It represents the conversion format for numbers. Its default value is **%.6g**.
- **ENVIRON -**It is an associative array of environment variables.
- **FILENAME -**It represents the current file name.

Contd..

- **FS -**It represents the (input) field separator and its default value is space. You can also change this by using **-F** command line option.
- **NF -**It represents the number of fields in the current record. For instance, the following example prints only those lines that contain more than two fields.
- **NR -**It represents the number of the current record. For instance, the following example prints the record if the current record number is less than three.

Contd..

- **FNR -**It is similar to NR, but relative to the current file. It is useful when AWK is operating on multiple files. Value of FNR resets with new file.
- **OFMT -**It represents the output format number and its default value is **%.6g**.
- **OFS -** It represents the output field separator and its default value is space.

Contd..

- **RLENGTH -I**t represents the length of the string matched by **match** function. AWK's match function searches for a given string in the input-string.
- **RS -**It represents (input) record separator and its default value is newline.
- **RSTART -**It represents the first position in the string matched by **match** function.

Contd..

- **SUBSEP -**It represents the separator character for array subscripts and its default value is **\034**.
- **$0 -**It represents the entire input record
- **$n -**It represents the n^{th} field in the current record where the fields are separated by FS.

Operators

- Arithmetic
- Logical
- Assignment
- Increment/Decrement
- Ternary
- Unary
- Exponential
- String catenation
- Array membership
- Regular Expression

Control Statments

- If
- If Else
- If –Else-If-Ladder
- For Loop
- While Loop
- Do While Loop
- Break
- Continue
- Exit

Built in Functions

- Arithematic
- Bit
- Time
- String manipulation
- Miscellaneous

User Defined Functions

- Functions are basic building blocks of a program. AWK allows us to define our own functions. A large program can be divided into functions and each function can be written/tested independently. It provides re-usability of code.
- **Syntax**
- function function_name(argument1, argument2, ...) { function body }

Contd..

- In this syntax, the **function_name** is the name of the user-defined function. Function name should begin with a letter and the rest of the characters can be any combination of numbers, alphabetic characters, or underscore. AWK's reserve words cannot be used as function names.
- Functions can accept multiple arguments separated by comma. Arguments are not mandatory. You can also create a user-defined function without any argument.
- **function body** consists of one or more AWK statements.

Redirection Operator

- The syntax of the redirection operator is −

  print DATA > output-file

- It writes the data into the **output-file**. If the output-file does not exist, then it creates one. When this type of redirection is used, the output-file is erased before the first output is written to it. Subsequent write operations to the same output-file do not erase the output-file, but append to it.

Contd..

- Append- It appends the data into the **output-file**. If the output-file does not exist, then it creates one. When this type of redirection is used, new contents are appended at the end of file.
- Pipe- It is possible to send output to another program through a pipe instead of using a file. This redirection opens a pipe to command, and writes the values of items through this pipe to another process to execute the command. The redirection argument command is actually an AWK expression.

Thank You

Questions?