AWK SENCILLO Y FÁCIL.

NÉSTOR MANUEL GARZÓN TORRES[1]

Resumen

Lenguajes de programación existen muchos y muy variados en cuanto a su semántica y su sintaxis; unos son fáciles de aprender, otros tienen un grado mayor para su aprendizaje, pero todos tienen en común algo: se tratan de lenguajes para escribir programas para computadoras, que permitan realizar procesos que reciben una serie de datos de entrada, se procesan mediante la lógica (instrucciones) y producen una salida o información para el usuario. En general, estos lenguajes pueden ser compilados (necesitan de un compilador para crear el ejecutable) o simplemente interpretados (no necesitan ser compilados para ejecutar).

AWK –tema del presente artículo- es un lenguaje de programación interpretado, cuyo propósito es la manipulación de datos en formato texto, mediante una serie de instrucciones muy fáciles de aprender y de escribir; simplemente se deben escribir unas reglas compuestas por una serie de patrones y unas acciones a ejecutar. Se muestra en este artículo, las bases fundamentales sobre las características del lenguaje, su escritura, sus diferentes formas de ejecutarlo y una serie de ejemplos muy cortos (una o dos líneas de código) para dar una idea de la facilidad y potencialidad de este lenguaje.

Abstract

Programming languages exist many and very varied as for their semantics and their syntax; some are easy to learn, others have a bigger grade for their learning, but all have in common something: they are languages to write programs for computers that allow to carry out processes that receive a series of entrance data, they are processed by means of the logic (instructions) and they produce an exit or information for the user. In general, these languages can be compiled (they need of a compiler to create the executable one) or simply interpreted (they don't need to be compiled to execute).

AWK - he/she fears of the present article - it is an interpreted programming language whose purpose is the manipulation of data in format text, by means of a series of instructions very easy of learning and of writing; some rules should simply be written composed by a series of patterns and some actions to execute. It is shown in this article, the fundamental bases on the characteristics of the language, their writing, their different forms of executing it and a series of very short examples (an or two code lines) to give an idea of the easiness and potentiality of this language.

Introducción

Los usuarios de computadores, desarrolladores y programadores gastan a menudo mucho tiempo manipulando datos tipo texto de una manera simple y mecánica – cambiando el formato de datos, comprobando su validez, buscando ítems en un registro que cumpla con alguna propiedad, realizando cálculos numéricos con items, imprimiendo reportes, y cosas por el estilo. Todos estos trabajos se deben mecanizar de alguna manera, razón por la cual se deben escribir programas de propósito general en un lenguaje estándar como C/C+ +, java, VisualBasic, etc., para ser ejecutados cada vez que sea necesaria la tarea específica.

Awk es un lenguaje de programación interpretado que hace posible manipular tareas de este tipo, con programas muy cortos, a menudo sólo una o dos líneas de longitud. Un programa awk es una secuencia de acciones y de patrones que dicen qué buscar en los datos de entrada (registros) y qué debe hacer cuando los encuentre. Awk busca en un conjunto de líneas componentes de un archivo, ítems que coincidan con cualquiera de los patrones. Cuando encuentra dicha correspondencia en una línea, la acción correspondiente es realizada. Un patrón puede

[1] Docente programa de Ingeniería de Sistemas, Universidad Autónoma de Colombia

seleccionar líneas por combinación de expresiones regulares y operaciones de comparación sobre cadenas, números, campos, variables, y arreglos de elementos.

Awk busca en los archivos de entrada y divide cada línea de entrada automáticamente en campos. De esta manera, un uso común de awk es justamente para la de manipulación de datos mencionada antes. Los programas, una línea o dos, son escritos desde el teclado, ejecutados de una vez, y luego descartados. En efecto, awk es una herramienta de programación de propósito general que puede reemplazar buena parte de herramientas especializadas o programas .

Este artículo es una breve introducción a unas de las herramientas (comando/programa AWK) disponible en los sistemas UNIX y GNU/LINUX . Aunque no tan popular como el shell o muchos otros lenguajes para escribir scripts, es una herramienta muy potente cuando hay que tratar con información tipo texto agrupada en tablas o archivos planos. Para usuarios con buen conocimiento de estos sistemas, administrador del sistema, desarrolladores de aplicaciones, esta herramienta es la equivalente a una navaja de las tipo suizo (victorinox), que sirven para muchas cosas, es, simplemente, una herramienta casi imprescindible.

También se puede utilizar para realizar las funciones que realizan muchas otras herramientas del sistema GNU/LINUX – buscar patrones, como **egrep**[2], **find**[3],o modificar archivos, como **tr o sed**[4] -. Pero, además, como es un lenguaje de programación interpretado, que maneja estructuras de control, funciones, variables, arreglos (muy parecido en su estructura al lenguaje C) resulta más potente y flexible que cualquiera de estos comandos. Programas en awk de una sola línea, pueden ser el equivalente a una herramienta regular del sistema operativo; por ejemplo, con un programa de una línea se puede contar el número de líneas de un archivo (comando wc), imprimir el primer campo de una tabla (comando cut), imprimir todas las líneas que tengan la palabra <autónoma> (comando grep), intercambiar la posición de los campos 1 y 2 de cada línea en un archivo (comando join y paste), comandos estos que sabemos están escritos en C/C++ y se componen de buena cantidad de líneas (instrucciones).

AWK le permite al desarrollador ejecutar los programas desde la línea de comandos, en lugar de usar los bonitos GUI que consiguen que el usuario no sepa cómo se hacen las cosas.

AWK fue escrito por **A**lfred V. Aho, Meter J. **W**einberger y Brian W. **K**ernighan, cuya primera versión apareció en 1977 en los laboratorios AT&T. En 1985 una nueva versión lo hizo más potente, introduciendo funciones definidas por el usuario, múltiples streams de entrada y evaluación de expresiones regulares. El release 4 de UNIX System V añadío algunas características nuevas y también corrigió algunos de los agujeros que presentaba el lenguaje.

La implementación para GNU, gawk, fue escrita en 1986 por Paul Rubin y Jay Fenlason, atendiendo consejos de Richard Stallman. En 1988 y 1989, David Trueman, con ayuda de Arnold Robbins, trabajaron duramente para hacer a gawk compatible con el nuevo awk.

Empecemos de una vez

Una de las funciones básicas de awk es buscar líneas en archivos (tipo texto) que cumplan con ciertos patrones. Cuando encuentra un patrón determinado en una línea, awk realiza las acciones especificadas para dicho patrón en la línea respectiva. El proceso se continúa línea por línea hasta que se encuentra el final del archivo.

Para mostrar algunos ejemplos de cómo trabaja AWK, vamos a suponer que tenemos un archivo que contiene información sobre estudiantes de la Universidad, con la siguiente información:

código	nombre	carrera	semestre	jornada	vlr.semestre
1234	PEDRO	SISTEMAS	5	D	1850000
2345	MARIA	DERECHO	3	D	1900000
4534	FRANCISCO	SISTEMAS	8	N	1850000
4567	MAURICIO	INDUSTRIAL	3	D	2000000
6787	CLAUDIA	DISEÑO	7	D	2100000
3344	YADIRA	DERECHO	9	D	1900000
5541	CONSUELO	SISTEMAS	8	N	1850000

Este archivo se llamará alumnos.txt, para todos los ejemplos siguientes.

[2] grep, egrep, fgrep: familia de comandos para buscar una cadena en un(os) archivo(s).
[3] find: comando para buscar archivos en la estructura de archivos del sistema.
[4] sed: comando para modificar un archivo simplemente desde la línea de comandos.

Ahora, queremos calcular e imprimir el valor total recaudado por concepto de matrículas por semestre. Muy fácil hacerlo con AWK:

$ awk '{s+= $6} END {print "Valor total recaudado por matrículas = ", s}' < alumnos.txt

Sencillamente, awk divide cada línea de entrada en campos –cada campo está delimitado por espacios en este caso-, de ahí que $6 es el campo vlr.semestre de cada línea de entrada y nos referimos a él como $6. Textualmente, el programa hace lo siguiente: por cada línea en el archivo de entrada, sume el vlr. semestre ($6) y cuando termine de leer todas las líneas (END) imprima el valor total calculado. Facil!!!!

Para realizar esta tarea, simplemente desde el prompt del sistema, se escribe tal cual aparece y pulsamos Enter; el resultado será:

Valor total recaudado por matrículas = 13450000

Ahora queremos imprimir la información de los estudiantes de la carrera sistemas; procedemos así:

$ awk '$3 == "SISTEMAS" {print $0}' < alumnos.txt [5]

La salida es:

1234	PEDRO	SISTEMAS	5	D	1850000
4534	FRANCISCO	SISTEMAS	8	N	1850000
5541	CONSUELO	SISTEMAS	8	N	1850000

Aquí, le estamos diciendo a awk que busque en cada línea y si el tercer campo ($3) es igual a la cadena SISTEMAS, imprima toda la línea ($0).

Como se puede ver, un programa awk le dice a awk que tiene que hacer. El programa consiste de una serie de reglas. Cada regla especifica un patrón a buscar, y una acción a realizar cuando se encuentre dicho patrón en la línea de entrada.

Sintacticamente, una regla consiste en un patrón seguido por una acción. La acción se debe encerrar entre llaves {} para separarlas de los patrones. Un programa awk, se ve de la siguiente forma:

patrón { acción }
patrón { acción }
….
En una regla awk, o el patrón o la acción puede ser omitida, pero no ambas. Si el patrón se omite, entonces la acción se realiza para cada línea de entrada. Si se omite la acción por defecto es imprimir todas las líneas que concuerdan con el patrón.

awk '{print $0}' … se omite el patrón, se realiza la acción por cada línea.
awk '$3 == "SISTEMAS" {}' … se omite la acción, por defecto imprimir todas las líneas que cumplan con el patrón.

Ejecutando un programa AWK

Hay diferentes maneras de ejecutar (run) un programa awk. Se puede digitar desde la línea de comandos, de la forma:

awk 'programa' archivos_entrada

Para ejecutar un programa que realice las acciones sobre cada uno de los archivos de entrada. Por ejemplo, se puede escribir

$ awk '$3 == 0 {print $1 }' file1, file2

Para imprimir el primer campo ($1) de cada archivo (file1, file2) si el campo 3 ($3) es igual a 0.

Se puede omitir el archivo de entrada en la línea de comando, como esto

awk 'programa'

En este caso, awk aplica las acciones a realizar, a cada una de las entradas digitadas desde el teclado, hasta cuando se pulsen las teclas CTRL-D. Un ejemplo:

```
$ awk '$2 == 0 { print $1}'
Pedro          2
Teresa         5
Carlos         0
Julian         0
Enrique        4
….
```

La salida será:

```
Carlos         0
Julian         0
```

Estos métodos son apropiados cuando el programa es corto (una o dos líneas). Si el programa es un poco más largo, es conveniente crearlo como un archivo separado, con un nombre específico, y entonces desde la línea de comando escribir:

awk –f nombre_programa [lista_de_archivos_de_entrada]

donde la opción –f le dice a awk que ejecute el programa que está en nombre_programa.

Salidas simples

Se presentan a continuación una serie de programas muy cortos, donde se muestra la funcionalidad de la herramienta para la manipulación de archivos tipo texto (alumnos.txt), con su respectiva explicación para entender un poco más la semántica del programa.

[5] El signo $ al inicio de la línea indica el prompt del sistema; este puede ser diferente en su máquina.

- **Mostrar un archivo**

$ awk '{print $0}' < alumos.txt … reemplaza el comando cat
La salida es:

1234	PEDRO	SISTEMAS	5	D	1850000
2345	MARIA	DERECHO	3	D	1900000
4534	FRANCISCO	SISTEMAS	8	N	1850000
4567	MAURICIO	INDUSTRIAL	3	D	2000000
6787	CLAUDIA	DISEÑO	7	D	2100000
3344	YADIRA	DERECHO	9	D	1900000
5541	CONSUELO	SISTEMAS	8	N	1850000

- **Imprimir algunos campos del archivo de entrada**

$ awk '{print $1, $2 }' < alumnos.txt … muestra código y nombre
La salida es:
1234 PEDRO
2345 MARIA
4534 FRANCISCO
4567 MAURICIO
6787 CLAUDIA
3344 YADIRA
5541 CONSUELO

- **Imprimir número de campos de cada línea, campo primero y último**

$ awk '{ print NF, $1, $NF}' < alumnos.txt … NF es Number Fields, $NF es el último campo de la línea de entrada
La salida es:
6 1234 1850000
6 2345 1900000
6 4534 1850000
6 4567 2000000
6 6787 2100000
6 3344 1900000
6 5541 1850000

- **Imprimir el número de la línea de entrada (número de registro)**

$ awk '{print NR, $0}' < alumnos.txt … NR es Number Record.
La salida es:

1	1234	PEDRO	SISTEMAS	5	D	1850000
2	2345	MARIA	DERECHO	3	D	1900000
3	4534	FRANCISCO	SISTEMAS	8	N	1850000
4	4567	MAURICIO	INDUSTRIAL	3	D	2000000
5	6787	CLAUDIA	DISEÑO	7	D	2100000
6	3344	YADIRA	DERECHO	9	D	1900000
7	5541	CONSUELO	SISTEMAS	8	N	1850000

- **Imprimir solamente los registros pares**

$ awk 'NR % 2 == 0 {print NR, $0}' … NR % 2 == 0, similar al lenguaje C
La salida es:

2	2345	MARIA	DERECHO	3	D	1900000
4	4567	MAURICIO	INDUSTRIAL	3	D	2000000
6	3344	YADIRA	DERECHO	9	D	1900000

- **Imprimir utilzando formato de salida**

$ awk '{printf("Valor semestre para %s es $%.2f\n", $2, $6}' < alumnos.txt
La salida es:

```
Valor semestre para PEDRO      es $1850000
Valor semestre para MARIA      es $1900000
Valor semestre para FRANCISCO  es $1850000
Valor semestre para MAURICIO   es$2000000
Valor semestre para CLAUDIA    es $2100000
Valor semestre para YADIRA     es $1900000
Valor semestre para CONSUELO   es $1850000
```

Como se puede ver en el ejemplo anterior, la instrucción printf es similar a como se usa con el lenguaje C

- **Contando Líneas, Palabras y Caracteres de un archivo**

Este programa usa length, NF, y NR para contar el número de líneas, palabras y caracteres del archivo de entrada. Reemplaza el comando **wc**[6] de UNIX, GNU/LINUX.

```
#cuenta el número de lineas, palabras y caracters de un archivo...
#contar.sh
{
  nc = nc + length($0) + 1
  nw = nw + NF
}
END { print NR, "lines,", nw, "words,", nc, "characters"}
```

Se ejecuta desde la línea de comandos así:

$ awk –f contar.sh contar.sh <Enter>
Mostrará por pantalla:
7 lines, 35 words, 184 characters

Realizar este programa, por ejemplo en Turbo C, necesita algo más de 30 líneas de buen código; bastante diferencia entre los dos.

- **Calcular el total de bytes que ocupan los archivos de un directorio..**

$ ls –l | awk '{s += $5} END {print = "Total bytes directorio = ", s}'

Este ejemplo utiliza el comando ***ls –l*** que muestra un listado en formato largo de los archivos de un directorio, filtra la salida hacia la entrada para el programa awk, que lee cada línea de entrada y va acumulando en la variable ***s***, el tamaño en bytes del archivo (campo 5, $5); cuando termina de leer todas las líneas, imprime el total de bytes que ocupa el directorio. En un lenguaje tradicional, hacer esto lleva buen número de líneas de código.

El comando *ls –l* produce el siguiente listado:

```
total 45
drwxr-xr-x   2 root root  376 Oct 16 16:45 .
drwx------  30 root root 1392 Oct 16 16:43 ..
-rw-r--r--   1 root root  519 Oct 16 16:35 abc.txt
-rw-r--r--   1 root root   75 Oct 13 21:43 abc1
-rw-r--r--   1 root root  174 Oct 16 15:56 contar.sh
-rw-r--r--   1 root root    0 Oct 16 16:45 listado
-rw-r--r--   1 root root  932 Oct 16 16:04 passwd
-rw-r--r--   1 root root  932 Oct 16 16:04 passwd.rev
-rw-r--r--   1 root root  950 Oct 16 16:22 passwd1
-rw-r--r--   1 root root  207 Oct 16 16:01 reverse.sh
-rw-r--r--   1 root root  166 Oct 13 21:49 reverse1.sh
-rw-r--r--   1 root root  188 Oct 16 16:26 reverse2.sh
-rw-r--r--   1 root root  187 Oct 13 22:16 reverse3
-rw-r-----   1 root root  799 May 22 09:01 shadow
```

Al ejecutar el commando ls –l | awk ... produce la siguiente salida:

Total bytes directorio = 6873

Manejo de arreglos

AWK provee arreglos (arrays) para almacenar grupos de valores relacionados. Estos le dan a awk una considerable fuerza y capacidad. Se muestra el uso sencillo del manejo de un arreglo, en un ejemplo que lee todas las líneas de entrada de un archivo, las almacena en un arreglo y luego las imprime (por pantalla) en orden inverso: última de primera, antepenúltima de segunda y así sucesivamente.

```
#invierte las lineas de un archivo: primero la última, segundo
antepenúltima , ....
#invertir.sh
   { line[NR] = $0 }
END {
    i = NR
    while (i > 0) {
       print line[i]
       i = i - 1
    }
  }
```

$ awk –f invertir.sh prueba.txt <Enter>

[6] wc (word counter) : comando UNIX, GNU/LINUX para contar el número de líneas, palabras y caracteres de un archivo.

Archivo de prueba....
El separador de campos está representado por la variable implícita FS. Que tomen nota los programadores de la Shell!. Awk no usa el nombre IFS el cual es usado por la Shell. Puedes cambiar el valor de FS en el programa awk con el operador asignación, '='. A menudo el momento adecuado para hacer esto es al principio de la ejecución, antes de que se procese alguna entrada, de forma que el primer registro se lea con el separador adecuado. Para hacer esto, use el patrón especial BEGIN.

Al ejecutar este programa, y teniendo como archivo de entrada el texto mostrado, la salida producida es:

adecuado. Para hacer esto, use el patrón especial BEGIN.
entrada, de forma que el primer registro se lea con el separador principio de la ejecución, antes de que se procese alguna
ción, '='. A menudo el momento adecuado para hacer esto es al cambiar el valor de FS en el programa awk con el operador asignano usa el nombre IFS el cual es usado por la Shell. Puedes
implícita FS. Que tomen nota los programadores de la Shell!. Awk
El separador de campos está representado por la variable
Archivo de prueba....

El lector que tenga una formación en lenguaje de programación, puede observar la gran facilidad presentada con estas pocas líneas y lo que permite hacer. Se puede guardar la salida estándard en un archivo, de la siguiente manera:

$ awk –f invertir.sh prueba.txt > prueba_inv.txt <Enter>

Aquí se direcciona la salida estandard (monitor) hacia un archivo llamado prueba_inv.txt.

Invirtiendo el orden de los campos de una línea

Otro ejemplo de la gran utilidad de la herramienta, es un pequeño programa que lea cada línea de entrada de un archivo y la imprima en orden inverso con respecto a cada uno de sus campos (último de primero, antepenúltimo de segundo y así en adelante). Es una especie de encriptador de un archivo.Este es el programa:

```
#escribe cada linea invertida (último campo de primero, penúltimo de segundo, ....
#invertir_linea.sh
{
  for (i = NF; i > 0; i--) {
     printf ("%s ",  $i)
  }
  printf ("\n")
}
```

En este ejemplo se observa el uso de la instrucción for similar a como se utiliza en lenguaje C/C++. Al ejecutar este programa con archivo de entrada el mismo programa, muestra la siguiente salida:

$ awk –f invertir_linea.sh invertir_linea.sh <Enter>

```
… segundo, de penúltimo primero, de campo (último invertida
linea cada #escribe
#invertir_linea.sh
{
{ i--) 0; > i NF;  = (i for
   $i)   ("%s" ,  printf
}
("\n") printf
}
```

Otro ejemplo más interesante consiste en leer cada línea de entrada e invertirla, no campo por campo, sino, más bien carácter a carácter, es decir, el último carácter de la línea se imprime de primero, el antepenúltimo se imprime de segundo y así sucesivamente. Este programa se puede usar para tratar de proteger los programas fuentes, convirtiéndolos de esta manera, púes difícilmente otro usuario puede interpretar el contenido del archivo invertido.

```
#escribe cada linea invertida (último caracter de primero, penúltimo de segundo, ....
#invertir_car.sh
{
  for ( i = length($0); i > 0; i--) {
     printf ( "%c", substr($0, i, 1))
  }
  printf ("\n")
}
```

Al ejecutarlo, como sigue:
$ awk –f invertir_car.sh invertir_car.sh <Enter>, la salida es:

```
.... ,odnuges ed omitlúnep ,oremirp ed retcarac omitlú( aditrevni aenil adac ebircse#
hs.rac_ritrevni#
{
{ )--i ;0 > i ;)0$(htgnel = i ( rof
  ))1 ,i ,0$(rtsbus ,"c%" ( ftnirp
}
)"n\"( ftnirp
}
```

Habiendo ya presentado una serie de programas cortos para producir diferentes resultados, miremos a continuación otra serie de programas más cortos, que pueden sernos de gran utilidad en un momento dado.

- **Imprimir en número total de líneas del archivo de entrada:**

```
END {print NR}
```

- **Imprimir la décima línea de entrada:**

```
NR == 10
```

- **Imprimir el último campo de cada línea:**

```
{ print $NF}
```

- **Imprimir el último campo de la última línea de entrada:**

```
    { field = $NF }
END {print field }
```

- **Imprimir las líneas que tengan más de 4 campos:**

```
NF > 4
```

- **Imprimir cada línea de entrada en la cual el último campo es mayor de 4:**

```
$NF > 4
```

- **Imprimir el número total de campos de todas las líneas de entrada:**

```
    { nf = nf + NF }
END {print nf }
```

- **Imprimir el número total de líneas que contengan la cadena AN:**

```
/AN/   { nlines = nlines + 1 }
END    { print nlines }
```

- **Imprimir el primer campo más largo y la línea que lo contiene:**

```
$1 > max  { max = $1
            maxline = $0 }
    END { print max, maxline }
```

- **Imprimir cada línea que tenga al menos un campo:**

```
NF > 0
```

- **Imprimir cada línea mayor de 80 caracteres:**

```
Length($0) > 80
```

- **Imprimir el número de campos de cada línea, y el contenido de la línea:**

```
{ print NF, $0}
```

- **Imprimir los dos primeros campos en orden invertido:**

```
{ print $2, $1 }
```

- **Intercambiar los dos primeros campos y luego imprimir la línea:**

```
{ temp = $1; $1 = $2; $2 = temp; print }
```

- **Imprimir cada línea, reemplazando el primer campo por el número de registro:**

```
{ $1 = NR; print }
```

- **Imprimir cada línea después de borrar el segundo campo:**

```
{ $2 = "" ; print }
```

- **Imprimir la suma de los campos de cada línea (campos deben ser numéricos):**

```
{ sum=0
  for ( i = 1; i <= NF; i++ ) sum = sum + $i
  print sum
}
```

- **Imprimir cada línea después de reemplazar cada campo por su valor absoluto:**

```
{ for (i = 1; i <= NF; i++) if ($1 < 0) $i = -$i
  print
}
```

Los programas mostrados en este artículo, son simplemente la parte esencial del lenguaje AWK. Cada uno de estos programas, ha sido una secuencia de instrucciones patrón-acción; se observó cómo awk prueba cada línea de entrada contra el patrón respectivo, y cuando hay concordancia de este en la línea, se realiza la acción correspondiente. Son muchas las aplicaciones que se pueden realizar con este lenguaje, lo mostrado acá es simplemente un "aperitivo" de todo lo interesante que resulta ser awk. Para el lector interesado en el aprendizaje y uso de esta herramienta, en la bibliografía se menciona un muy buen libro sobre el lenguaje; con este material, y por supuesto, con buenos conocimientos sobre el Sistema Operativo GNU/LINUX se aprende a programar en AWK.

Bibliografía

Alfred V. Aho, Brian W. Kernighan, Peter J. Weinberger, The AWK programming Language, AT&T Bell Labotatories

Jesús Alberto Vidal Cortés, El lenguaje de Programación AWK/GAWK. Una guía de Usuario, http://inicia.es/de/chube

Manual en línea de AWK ($ man awk) sistema operativo UNIX – GNU/LINUX, Free Software Foundation, Junio 2003

Mohr James, LINUX Recursos para el usuario, Prentice may, México, 1999.