PRÁCTICA 3

LA HERRAMIENTA AWK

Objetivos

Programación y automatización de tareas utilizando la herramienta *awk*.

La herramienta awk

Es una herramienta que se emplea para procesar ficheros de texto, línea a línea, realizando diversas acciones sobre ellos. La sintaxis básica de la llamada al programa *awk* es la siguiente:

```
awk [-Fc] [-v var=valor ...][-f fichero | 'programa'] [ficheros_texto]
```

donde:

-Fc	Asigna el carácter (c) separador de los campos de las líneas (por defecto es uno o mas espacios en blanco). $0 se utiliza para representar toda la línea actual y $1...$n cada uno de los campos de la línea actual.
-v var=valor	Asigna a la variable *var* el valor indicado. La opción -v solo puede establecer una variable, pero se puede usar más de una vez.
-f fichero	Hace que el programa que requiere *awk* se tome desde un fichero en lugar de un texto entrecomillado.
ficheros_texto	Ficheros sobre los que se aplica el programa *awk*. Puede sustituirse el nombre de los ficheros por la salida de una orden anterior y el uso de tuberias

Un programa *awk* consta de sentencias de la forma `patrón{acción}`. Cada línea del fichero de entrada se contrasta con cada *patron* que aparece en el programa *awk* y si encaja se realizan las acciones entre llaves. Una vez que la línea no encaja en más patrones se pasa a la siguiente línea. Si no se especifica ningún *patron* entonces todas las líneas encajan en él.

Para formar los patrones se utilizan expresiones regulares (en la forma /exp_reg/), expresiones relacionales y los operadores lógicos && (AND), || (OR) y ! (NOT). Las expresiones relacionales pueden ser de la forma siguiente

expresión operador_relacional_de_C expresión
expresión operador_mach expresión_regular
expresión in nombre_array
(expresión,expresión,...) in nombre_array

El "operador_relacional_de_C" es uno de los siguientes: <, >, ==, >=, <=, ! =. El "operador_mach" es la tilde (~) para indicar "debe estar contenido en", y su negación ! para indicar "no contenido en". Las expresiones pueden ser aritméticas, relacionales, expresiones de la forma "var in nombre_array", o una combinación de ellas.

Además *awk* proporciona dos patrones especiales: BEGIN y END que pueden emplearse para realizar acciones antes de que la primera línea sea leída y procesada, y después de que la última línea sea leída y procesada.

awk reconoce entre otras las siguientes variables:

ARGC	Número de argumentos (incluye el propio programa *awk* y los ficheros argumentos).
ARGV	Array de argumentos.
FILENAME	Nombre del fichero que está procesando actualmente.
FNR	Número de registro (línea) actual en el fichero que está siendo procesado.
NR	Número de registro (línea) global en el conjunto de todos los ficheros procesados. Si solo se procesa un fichero coincide con FNR
NF	Número de campos en un registro.
FS	Separador de campo a la entrada.
OFS	Separador de campo a la salida.
RS	Separador de registro a la entrada.
ORS	Separador de registro a la salida.
$0	Registro completo actual.
$1 . . . $n	Campos en el registro actual.

Las acciones que aparecen encerradas entre las llaves tienen una sintaxis similar a C:

if (expresión) sentencia [else sentencia]
while (expresión) sentencia
for (expresión;expresión;expresión) sentencia
for (var in nombre_array) sentencia
delete nombre_array[indice]
break
continue
{[sentencia]}
variable=expresión
print [lista de expresiones] [>fichero]
printf formato [,listade expresiones] [>fichero]
next (pasa al siguiente registro sin procesar el actual)
exit (se salta el resto de registros del fichero de entrada)
return expresión

Además *awk* soporta un conjunto de funciones predefinidas, entre las cuales se encuentran:

index(s,t) Si t es una subcadena de s, devuelve la posición de comienzo de t en s. En caso contrario devuelve 0.

length(s) Devuelve la longitud de la cadena s.

substr(s,m,n) Devuelve una subcadena de s tomando n caracteres desde la posición m.

getline Lee el siguiente registro de entrada y asigna a $0 dicho registro.

system(comando) Ejecuta el comando desde un *shell* y retorna el valor de *errorlevel* obtenido.

Ejemplos de awk

En los siguientes ejemplos de *awk* se utiliza el fichero */etc/passwd*. Cada línea del fichero se compone de 7 campos separados por el carácter *dos puntos* (:). Por ejemplo, en la siguiente línea:

```
usuario:XXXXX:1000:1001:nombre usuario:/home/usuario:/bin/bash
```

el significado de cada campo es el siguiente:

1	usuario	Nombre de la cuenta
2	XXXXX	Clave de acceso encriptada
3	1000	UID de la cuenta
4	1001	GID del grupo principal al que pertenece la cuenta
5	nombre usuario	Nombre del usuario
6	`/home/usuario`	Directorio de trabajo del usuario
7	`/bin/bash`	Interprete de comando (shell) del usuario

Ejemplo 1

```
#!/bin/bash
# a) imprime passwd
# b) imprime el resultado de ls -l

echo "Fichero passwd"
awk '{print $0}' /etc/passwd
echo "Listado de ficheros"
ls -l | awk '{print $0}'
```

Ejemplo 2

```
#!/bin/bash
#
# imprime los 2 ficheros indicados numerando las lineas, primero globalmente
# y luego de cada fichero

echo -e "\nNumeración global"  # con salto de linea previo
awk '{print NR,$0}' /etc/passwd /etc/group
echo -e "\nNumeración por cada fichero"
awk '{print FNR,$0}' /etc/passwd /etc/group
```

Ejemplo 3

```
#!/bin/bash
#
# imprime el primer campo
#(separador por defecto es " ")
#
awk '{print $1}' /etc/passwd
```

Ejemplo 4

```
#!/bin/bash
#
# Imprime (con salida formateada)
# los campos 1 y 5 (usuario y nombre).
# El campo 1 tiene una anchura de 20 caracteres y se justifica
# a la izquierda.
# El separador es ":"
#
awk -F: '{printf "%-20s %s\n",$1,$5}' /etc/passwd
```

Ejemplo 5

```
#!/bin/bash
#
# imprime las lineas 3 a 12 del fichero, numerando las lineas
#
# FNR es el contador de registros de un fichero
# NR es el contador de registros global
#
awk 'NR==3,NR==12{print NR,$0}' /etc/passwd
# o tambien
awk 'FNR>=3 && FNR<=12{print NR,$0}' /etc/passwd
```

Ejemplo 6

```
#!/bin/bash
#
# Imprime los campos 1 y 5 (usuario y nombre) de los usuarios # que empiezan por "r" o "d"
# El separador es ":"
awk -F: '/^r/{printf "%-10s %s\n",$1,$5}
         /^d/{printf "%-10s %s\n",$1,$5} ' /etc/passwd
```

Ejercicio 1 Crear un comando que realice las siguientes acciones:

- Del listado de procesos activos en el sistema muestre únicamente el usuario propietario del proceso (UID), el identificador del proceso (PID) y el comando ejecutado (CMD).
- Modifica el comando anterior de manera que solo se muestren los procesos de los usuarios cuyo nombre empieza por "r".

Ejemplo 7

```
#!/bin/bash
#
# Imprime los campos 1, 5 y 7 (usuario, nombre y shell) de los
# usuarios que utilizan la shell bash.
# El separador es ":"
#
awk -F: '$7=="/bin/bash"{printf "%-10s %-10s %s\n",$1,$5,$7}' /etc/passwd
```

Ejemplo 8

```
#!/bin/bash
#
# Imprime los campos 1, 5 y 7 (usuario, nombre y shell) de los
# usuarios que utilizan la shell bash y empiezan por "r".
# El separador es ":"
#
awk -F: '$7~/^\/bin\/bash$/ && /^r/{printf "%-10s %-10s %s\n",$1,$5,$7}' /etc/passwd
```

Ejemplo 9

```
#!/bin/bash
#
# Imprime el numero de lineas que tiene el fichero
#
awk 'BEGIN{num=0}
    {num++}
    END{print "Numero de lineas:",num}' /etc/passwd
```

Ejemplo 10

```
#!/bin/bash
#
# Imprime el campo 1 de la linea 5
#
awk -F: -v l=5 -v c=1 ' NR==l{print $c}
    END{print "Se imprimio el campo "c" de la linea "l}' /etc/passwd
```

Ejemplo 11

```
#!/bin/bash
#
# Imprime un campo de una linea (pasados como parametros)
#
awk -F: -v l=$1 -v c=$2 'NR==l{print $c}
    END{print "Se imprimio el campo "c" de la linea "l}' /etc/passwd
```

Ejemplo 12

```
#!/bin/bash
#
# Imprime los campos 1 y 5 (usuario y nombre) de los usuarios que
# empiezan por "r" siempre que el campo 5 no este vacio
# El separador es ":"
#
awk -F: '/^r/ && $5!=""{print $1"\n-->\t"$5}' /etc/passwd
# otra forma equivalente
awk -F: '/^r/{if ($5!="") print $1"\n-->\t"$5}' /etc/passwd
```

Ejemplo 13

```
#!/bin/bash
#
# Imprime el campo 1 y su longitud siempre que sea mayor que el
# parametro pasado en la llamada.
# El separador es ":"
#
awk -F: -v l=$1 '{if (length($1)>l) print length($1),$1}' /etc/passwd
```

Ejemplo 14

```
#!/bin/bash
#
# Imprime el campo 1 ordenado alfabeticamente
# El separador es ":"
#
awk -F: '{print $1}' /etc/passwd | sort
```

Ejemplo 15

```
#!/bin/bash
#
# redirecciona la salida a un fichero y ejecuta un comando del shell
#
awk '{print $0 >"borrar"} END{system("cat borrar | more")}' /etc/passwd
```

Ejercicio 2 Crear un shell-script al que se le pase como parámetro el nombre de un usuario (xxxx) y muestre la siguiente información:

- Para cada proceso activo del usuario: *El usuario xxxx tiene un proceso con PID xxxx*
- Si no tiene procesos activos: *El usuario xxxx no tiene procesos activos*

Ejercicio 3 Mostrar por pantalla el contenido de un fichero numerando sus líneas de la manera que se indica en cada apartado.

- Mostrando todas las líneas del texto (vacías y no vacías) con el siguiente formato: 4 caracteres para el número de línea, un espacio en blanco, y a continuación la línea de texto.

  Por ejemplo, para el siguiente fichero:

  línea uno
  línea dos

  línea cuatro

  línea seis

  debe obtenerse la siguiente salida:

  1 línea uno
  2 línea dos
  3
  4 línea cuatro
  5
  6 línea seis

- Mostrando únicamente las líneas de texto no vacías y respetando el número de línea original. Para el mismo fichero de ejemplo anterior la salida será:

  1 línea uno
  2 línea dos
  4 línea cuatro
  6 línea seis

- Numerando por separado y consecutivamente las líneas vacías y no vacías. Después de mostrar el fichero debe indicarse el número de líneas vacías y no vacías.

  Para el mismo fichero de ejemplo la salida será:

  1 línea uno
  2 línea dos
  1
  3 línea cuatro
  2
  4 línea seis
  Líneas vacías:2. Líneas no vacías:4

Ejercicio 4 Unir las tres alternativas del ejercicio anterior en un fichero, para ello crear una shell-script a la que se le pase como parámetros el nombre de un fichero y una opción (del 1 al 3).
Se deberá comprobar que el número de parámetros es correcto, el fichero existe y la opción está comprendida entre 1 y 3. Dar los mensajes pertinentes.