http://lml.ls.fi.upm.es/ep/1112/awk.html#toc11
http://rm-rf.es/iniciacion-a-awk/

Introducción

El comando awk proporciona un lenguaje de scripting para el procesamiento de texto. Con el lenguaje de scripting awk se puede:

- Definir variables.
- Utilizar cadenas y operadores aritméticos.
- Utilizar control de flujo y ciclos.
- Generar reportes con formato.

Sintaxis del comando awk

Sintaxis de AWK:

$ awk options program file

Awk puede tomar las siguientes opciones:

- **-F fs** Para especificar un separador de archivos.
- **-f file** Para especificar un archivo que contenga un script awk.
- **-v var=value** Para declarar una variable.

Primer script AWK

Para definir un script awk utilizamos llaves rodeadas por unas comillas simples como se muestra a continuación:

$ awk '{print "Hola, ¿qué tal? "}'

```
alumno@alumno-VirtualBox:~$ awk '{print "Hola, ¿qué tal?"}'
^C
alumno@alumno-VirtualBox:~$ awk
alumno@alumno-VirtualBox:~$ awk '{print "Hola, ¿qué tal?"}'

Hola, ¿qué tal?

Hola, ¿qué tal?
alumno@alumno-VirtualBox:~$
```

Pulsa Enter para mostrar el mensaje y CTRL+D para terminar.

Variables en AWK

Con awk se puede procesar archivos de texto. Awk asigna algunas variables para cada campo de datos encontrado:

- $0 para toda la línea.
- $1 para el primer campo.
- $2 para el segundo campo.
- $n para el campo enésimo campo.

Los caracteres de espacio en blanco como el espacio o la tabulación es el separador por defecto entre los campos en el awk.

Por ejemplo:

$ awk '{print $1}' fichero-prueba`

Muestra la primera columna del fichero

```
alumno@alumno-VirtualBox:~$ cat fichero-prueba
Esto es una prueba
Esto es una segunda prueba
Esto es una tercera prueba
Esto es una cuarta prueba
alumno@alumno-VirtualBox:~$ awk '{print $1}' fichero-prueba
Esto
Esto
Esto
Esto
alumno@alumno-VirtualBox:~$
```

Algunas veces el separador en algunos archivos no es el espacio ni la tabulación, sino algo más. Puedes especificarlo utilizando la opción – F:

$ awk -F: '{print $1}' /etc/passwd

```
alumno@alumno-VirtualBox:~$ awk -F: '{print $1}' /etc/passwd
root
daemon
bin
sys
sync
games
man
lp
mail
news
```

Utilizando varios comandos en AWK

Para ejecutar múltiples comandos, sepáralos con un símbolo de punto y coma de la siguiente forma:

$ echo "Hola" | awk '{$2="Pepe"; print $0}'

El primer comando hace que el campo $2 sea equivalente a Pepe. El segundo comando muestra la línea completa.

```
alumno@alumno-VirtualBox:~$ echo "Hola"|awk '{$2="Pepe"; print $0}'
Hola Pepe
alumno@alumno-VirtualBox:~$
```

Leyendo el Script Desde un Archivo

Puedes escribir tu script awk en un archivo y especificar ese archivo utilizando la opción –f.

#/bin/bash

{print $1 " directorio personal en " $6}

Luego escribe en la terminal

$ awk -F: -f script-awk /etc/passwd

```
alumno@alumno-VirtualBox:~$ awk -F: -f script-awk /etc/passwd
root directorio personal en /root
daemon directorio personal en /usr/sbin
bin directorio personal en /bin
sys directorio personal en /dev
sync directorio personal en /bin
games directorio personal en /usr/games
man directorio personal en /var/cache/man
lp directorio personal en /var/spool/lpd
mail directorio personal en /var/mail
news directorio personal en /var/spool/news
uucp directorio personal en /var/spool/uucp
proxy directorio personal en /bin
www-data directorio personal en /var/www
backup directorio personal en /var/backups
list directorio personal en /var/list
irc directorio personal en /var/run/ircd
gnats directorio personal en /var/lib/gnats
nobody directorio personal en /nonexistent
systemd-timesync directorio personal en /run/systemd
systemd-network directorio personal en /run/systemd/netif
systemd-resolve directorio personal en /run/systemd/resolve
systemd-bus-proxy directorio personal en /run/systemd
```

Se muestra el nombre de usuario y su ruta inicial desde /etc/passwd, y en este caso el separador es especificado con la letra mayúscula -F el cual es los dos puntos.

Pre procesamiento Awk

Si se necesita crear un título o cabecera para el resultado. Se utiliza la palabra clave BEGIN para lograr esto. Esta se ejecutará antes de procesar los datos:

$ awk 'BEGIN {print "Título"}'

También se puede aplicar a algo más:

$ awk 'BEGIN {print "Título:"}

{print $0}' fichero-prueba

Muestra el título y la columna 0 del fichero-prueba

```
alumno@alumno-VirtualBox:~$ awk 'BEGIN {print "Título"}'
Título
alumno@alumno-VirtualBox:~$ awk 'BEGIN {print "Título"}
> {print $0}' fichero-prueba
Título
Esto es una prueba
Esto es una segunda prueba
Esto es una tercera prueba
Esto es una cuarta prueba
```

Post procesamiento Awk

Para ejecutar un script después de procesar los datos, utiliza la palabra clave END

$ awk 'BEGIN {print "Título:"}

{print $0}

END {print “Fin del fichero”}' fichero-prueba

```
alumno@alumno-VirtualBox:~$ awk 'BEGIN {print "Título"}
{print $0}
> END {print "Fin del fichero"}' fichero-prueba
Título
Esto es una prueba
Esto es una segunda prueba
Esto es una tercera prueba
Esto es una cuarta prueba
Fin del fichero
alumno@alumno-VirtualBox:~$
```

A continuación, combinemos ambos en un archivo de script. Escribimos el siguiente script:

```
  GNU nano 2.7.4            File: script-awk

#/bin/bash
BEGIN {
print "Usuarios y su directorio personal"
print "Nombre de usuario \t Directorio personal"
print "-----------------\t -------------------"
FS=":"
}
{
print $1 "\t \t \t" $6
}
END {
print "Final"
}
```

Primero, la sección superior es creada utilizando BEGIN. Luego definimos FS e imprimimos el footer al final.

Ejecutamos el script:

```
alumno@alumno-VirtualBox:~$ awk -f script-awk /etc/passwd
Usuarios y su directorio personal
Nombre de usuario        Directorio personal
-----------------       -------------------
root            /root
daemon          /usr/sbin
bin             /bin
sys             /dev
sync            /bin
games           /usr/games
man             /var/cache/man
lp              /var/spool/lpd
mail            /var/mail
news            /var/spool/news
```

Variables Integradas

Hemos visto que las variables de campo de datos $1, $2 ,$3, etc son utilizadas para extraer datos del campo, además debemos lidiar con el separador FS.

Existen más variables integradas. Son:

- **FIELDWIDTHS** Especifica el ancho del campo.
- **RS** Especifica el separador de registros.
- **FS** Especifica un separador de campos.
- **OFS** Especifica un separador de Salidas.
- **ORS** Especifica el separador de Salidas.
- **NF** Cuenta los campos de la línea que está siendo procesada.
- **NR** Retorna el total de registros procesados.
- **FNR** Con esta variable puedes acceder al registro que está siendo procesado.
- **IGNORECASE** Le dice al programa que ignore las diferencias entre mayúsculas y minúsculas.

Por defecto, la variable OFS es el espacio, puedes establecer esta variable para que especifique el separador que necesitas.

Por ejemplo:

$ awk 'BEGIN{FS=":"; OFS="-"} {print $1,$6,$7}'/etc/passwd

```
alumno@alumno-VirtualBox:~$ awk 'BEGIN{FS=":";OFS="-"}{print $1,$6,
$7}' /etc/passwd
root-/root-/bin/bash
daemon-/usr/sbin-/usr/sbin/nologin
bin-/bin-/usr/sbin/nologin
sys-/dev-/usr/sbin/nologin
sync-/bin-/bin/sync
```

Como siguiente ejercicio supongamos que tus datos son distribuidos en líneas diferentes como se muestra a continuación:

```
  GNU nano 2.7.4                File: fich-direcciones

Nombre
10123 C/ Nueva
(927) 543-345

Otro nombre
10876 C/ Ancha
(927) 765-234
```

En el ejemplo anterior, awk falla al procesar los campos debido a que estos son separados por el carácter de nueva línea y no el de espacio.

Se necesita establecer la variable FS con el carácter de nueva línea (\n) y la RS a texto en blanco, de manera que las líneas vacías sean consideradas separadores.

$ awk 'BEGIN{FS="\n"; RS=""} {print $1,$3}' fich-direcciones

```
alumno@alumno-VirtualBox:~$ awk 'BEGIN{FS="\n";RS=""}{print $1,$3}' fich-direc
ciones
Nombre (927) 543-345
Otro nombre (927) 765-234
```

La variable NF especifica el último campo en el registro sin conocer su posición:

$ awk 'BEGIN{FS=":"; OFS=":"} {print $1,$NF}' /etc/passwd

```
alumno@alumno-VirtualBox:~$ awk 'BEGIN{FS=":";OFS=":"}{print $1,$NF}'  /etc/pa
sswd
root:/bin/bash
daemon:/usr/sbin/nologin
bin:/usr/sbin/nologin
sys:/usr/sbin/nologin
sync:/bin/sync
games:/usr/sbin/nologin
```

Esta variable NF puede ser utilizada como una variable de campo de datos si se escribe con esta sintaxis: $NF.

Veamos estos ejemplos para conocer la diferencia entre las variables FNR y NR:

$ awk 'BEGIN{FS=","}{print $1,"FNR="FNR}' fichero-prueba fich-direcciones

```
alumno@alumno-VirtualBox:~$ awk 'BEGIN{FS=","}{print $1,"FNR="FNR}' fichero-pr
ueba fich-direcciones
12345.9876 FNR=1
927-8.654 FNR=2
34564.87456 FNR=3
Nombre FNR=1
10123 C/ Nueva FNR=2
(927) 543-345 FNR=3
 FNR=4
Otro nombre FNR=5
10876 C/ Ancha FNR=6
(927) 765-234 FNR=7
```

Como se puede observar, FNR muestra el número de registro procesado en cada fichero.

Ahora escribimos:

$ awk 'BEGIN{FS=","}{print $1,"FNR="FNR}' fichero-prueba fichero-prueba

```
alumno@alumno-VirtualBox:~$ awk 'BEGIN{FS=","}{print $1,"FNR="FNR}' fichero-pr
ueba fichero-prueba
12345.9876 FNR=1
927-8.654 FNR=2
34564.87456 FNR=3
12345.9876 FNR=1
927-8.654 FNR=2
34564.87456 FNR=3
```

Usamos la variable NR:

$ awk 'BEGIN{FS=","}{print $1,"FNR="FNR,"NR="NR}END{print "Total",NR,"líneas procesadas"}' fichero-prueba fichero-prueba

```
alumno@alumno-VirtualBox:~$ awk 'BEGIN{FS=","}{print $1,"FNR="FNR}' fichero-pr
ueba fichero-prueba
12345.9876 FNR=1
927-8.654 FNR=2
34564.87456 FNR=3
12345.9876 FNR=1
927-8.654 FNR=2
34564.87456 FNR=3
alumno@alumno-VirtualBox:~$ awk 'BEGIN{FS=","}{print $1,"FNR="FNR, "NR="NR}END
{print "Total",NR,"líneas procesadas"}' fichero-prueba fichero-prueba
12345.9876 FNR=1 NR=1
927-8.654 FNR=2 NR=2
34564.87456 FNR=3 NR=3
12345.9876 FNR=1 NR=4
927-8.654 FNR=2 NR=5
34564.87456 FNR=3 NR=6
Total 6 líneas procesadas
```

La variable FNR se convierte en 1 cuando viene al segundo archivo, pero la variable NR mantiene su valor.

Dándole Formato a una Impresión

El comando printf en awk permite imprimir con un formato de salida utilizando especificadores de formatos.

Los especificadores de formatos son escritos de la siguiente forma:

%letradecontrol

Esta lista muestra cómo se puede utilizar los especificadores de formato con printf:

- c Imprime salidas numéricas como una cadena de caracteres (string).
- d Imprime un valor entero.
- e Imprime números con notación científica.
- f Imprime valores numéricos con decimales (float).
- o Imprime valores en notación octal.
- s Imprime una cadena de texto.

Operadores aritméticos

Suma: x+y
Resta: x-y
Negación: -x
Multiplicación: x*y
División: x/y
Resto: x%y
Exponente: x**y

Comparaciones: x<y , x<=y, x>y, x>=y, x==y, x!=y, x~y (‘y’ debe ser una expresión regular)

Expresiones booleanas: &&, ||, ! (negación)

En esta sentencia utilizamos printf para darle formato a la salida:

```
alumno@alumno-VirtualBox:~$ awk 'BEGIN{
> x=100 * 100
> printf "El resultado es: %e \n", x
> }'
El resultado es: 1.000000e+04
alumno@alumno-VirtualBox:~$
```

Ahora se mostrará como valor entero:

```
alumno@alumno-VirtualBox:~$ awk 'BEGIN{
x=100 * 100
printf "El resultado es: %d \n", x
}'
El resultado es: 10000
```

Funciones Integradas

Awk proporciona múltiples funciones integradas como:

Funciones Matemáticas

sin(x) | cos(x) | sqrt(x) | exp(x) | log(x) | rand()

Y estas pueden ser usadas normalmente así:

$ awk 'BEGIN{x=sqrt(5); print x}'

```
alumno@alumno-VirtualBox:~$ awk 'BEGIN{x=sqrt(5);print x}'
2.23607
```

Funciones de String (cadenas de caracteres)

Existen muchas funciones de string:

- lenght(string)
- substr(string,pos,len)
- tolower(string)
- toupper(string)

$ awk 'BEGIN{x = "hola"; print toupper(x)}'

```
alumno@alumno-VirtualBox:~$ awk 'BEGIN{x="hola";print toupper(x)}'
HOLA
```

La función toupper convierte los caracteres del string pasado a mayúsculas.