El lenguaje de procesamiento awk

awk

Sintaxis: awk [op] [−Ffs] `ord' [−v var=val]

archivo(s)

awk [op] [−Ffs] −f f_ord [−v var=val]

archivo(s)

La primera versión de awk para Unix fue diseñada e implementada por Alfred Aho, Peter Weinberger y Brian Kerninghan, de Bell Labs AT&t. Brian Kernighan sigue aún trabajando en ella en laboratorios de mantenimiento y mejora . El propio nombre, **awk,** deriva de las iniciales de los tres apellidos de los autores. Actualmente existen varias versiones de este programa. Nosotros nos vamos a referir en concreto a la versión de la *Free Software Foundation* (FSF), por ser la más completa y, además, compartible con las versiones iniciales.

Como podemos observar, tenemos dos modos diferentes de invocar al programa. En el primer modo le damos las órdenes desde la propia línea de órdenes, y en el segundo (opción −f), le especificamos un archivo donde se encuentran las órdenes que **awk** tiene que ejecutar. Este segundo modo es más cómodo si el conjunto total de ordenes es amplio, **awk** puede trabajar con varios archivos a un tiempo. Si no se le especifica ningún archivo, **awk** leerá en la entrada estándar. **awk** procesa los archivos especificados línea por línea, a cada línea se le compara con un patrón, y si coincide, se lleva a cabo sobre ella las acciones que indiquemos.

awk admite las siguientes opciones, las cuales deben estar disponibles en cualquier versión del programa.

−Fs Con esta opción indicaremos que el separador de campos es el carácter s. Esto es lo mismo que activar la variable predefinida FS. Por defecto, los separadores de campo utilizados por **awk** son los espacios en blanco y los tabuladores. Cada uno de los campos de una línea del archivo que se procesa puede ser referenciado por las variables $1, $2,..., $NF. La variable NF indica el numero de campo de línea que esta procesando. La variable $0 se refiere a la línea completa.

−v var=val: Asigna el valor val a la variable var antes de que se comience la ejecución del programa. Esta asignación de variable también se puede llevar a cabo en el bloque **BEJÍN** de un programa **awk**.

−f f_ordenes: awk leerá las ordenes en el archivo, son secuenciales de patrones y acciones:

patrón { acción }

Tanto el patrón como la acción son opcionales. Si falta el patrón, la acción o procedimiento se aplica a las líneas. Si falta la acción, simplemente se visualiza la línea.

Vamos a ver un primer ejemplo de uso de awk. Para ello, vamos a procesar lo que la orden **date** envía a la pantalla, que es algo como lo siguiente:

$ date

vie jun 15 18:43:08 CEST 2001

$

Lo único que vamos a hacer es visualizar los campos primero (día), segundo (mes), y sexto (año). La forma de hacerlo es la siguiente:

$date | awk `{print $1; print $2; print $6}'

vie

jun

2001

$

Seguidamente vamos a visualizar las líneas del archivo /etc/ paswd que comienzan con el carácter d:

$ awk `/^d/' /etc/paswd

daemon: x: 2: 2: daemon: / sbin:

Como no hemos especificado una opción, awk simplemente visualiza la línea que cumple el patrón que hemos indicado. El patrón anterior es una expresión regular, pero, como veremos en el punto siguiente, awk permite utilizar otros tipos de patrones.

Patrones de awk

Los patrones que awk reconoce pueden ser cualquiera de los siguientes:

- BEGIN
- END
- / expresiones regulares/
- expresiones relacionales

BEGIN y END son dos tipos de patrones especiales. El patrón BEGIN permite especificar una serie de procedimientos que se ejecutarán antes de que ninguna línea de ningún archivo sea procesado. Generalmente, con este patrón se declaran las variables globales. El patrón END permite especificar los procedimientos que no queremos que se ejecuten hasta que se terminen de procesar todas y cada una de las líneas de un archivo.

Para los patrones **/expresiones regulares**/, la acción se ejecuta para cada línea que verifica su expresión regular. Estas expresiones regulares son las mismas que hemos visto anteriormente.

Las expresiones relacionales pueden utilizar cualesquiera de los operadores que definiremos más tarde en el punto dedicado a ellos. Estos operadores se utilizan para comprobar si algún campo verifica alguna condición. Por ejemplo, **NF** > 2 selecciona las líneas en las que el numero de campos es mayor que dos.

Las expresiones de coincidencia de patrones utilizan los operadores −(coincide) y ! −(no coincide) para determinar si se lleva a cabo la acción o no.

Excepto para los patrones **BEGIN y END**, todos los patrones pueden ser combinados con operadores de Boole. Estos operadores son el **AND** lógico, **&&**, el **OR** lógico, | | , y el **NOT** lógico, **!** .

Con objeto de aclarar los conceptos mostrados, vamos a poner unos ejemplos de uso de patrones. En el primer ejemplo vamos a poner unos ejemplos de uso de patrones. En el primer ejemplo vamos a introducir todas las ordenes dirigidas a **awk** en un archivo, y a continuación lo procesaremos. El contenido del archivo es el siguiente:

```
$ cat f_awk

# Inicialización (se ejecuta al comenzar)

BEGIN { FS =  :  ; x = 0 }

# Si la línea comienza con P, se visualizara el primer campo

/ ^P / { print $1 }

# Si el numero de campos es mayor que tres

# visualizamos el campo cuatro

NF > 3 { print $4 }

#Si el cuarto campo es mayor que 10, incrementamos x

$ 4 > 10 { x++ }

# Finalizacion ( se ejecuta al finalizar)

END { print x }

$
```

Todas las líneas que comienzan con el carácter # serán ignoradas por awk en el procesamiento. Asi pues, podemos emplear este carácter como inicio de una línea de comentarios. En el caso anterior los comentarios son explicativos de lo que hace cada línea. El archivo anterior no tiene ninguna utilidad, se a empleado con el unico objeto de mostrar el uso de patrones.

A la hora de procesar este archivo, debemos emplear la siguiente sintaxis:

```
$ awk −f f_awk archivo (s)
```

Veamos otro ejemplo empleado primero tenemos que seleccionar las líneas que visualiza ls− l que comienza con el carácter d (directorios), y cuyo campo noveno (nombre del archivo) comience con letra mayúscula. Para especificar ambas condiciones, emplearemos el operador **&&** (**AND** lógico).

```
$ ls −l / usr | awk `$1 − /^d/ && $9 − /[A−Z] /'

drwx −xr − x 8 root root 4096 abr 3 21: 32 X 11R6

$
```

Según se puede apreciar, estamos empleando expresiones de coincidencia de patrones. La primera expresión regular indica si $1 coincide con el patrón especificado por la expresión regular /^d/. La segunda expresión indica si $9 coincide con el patrón especificado por la expresión regular /[A−Z] /'.

La forma que tiene awk de ejecutar los programas es el siguiente. Primero, awk compila el programa y genera un formato interno. A continuación, se realizan las asignaciones especificadas por medio de la opción −v . Seguidamente, awk ejecutara el código incluido en el bloque BENING, si es que existe tal bloque. Después, se procesa línea por línea el archivo o los archivos especificados en la línea de ordenes. Si no le especificamos ninguno, awk leerá en la entrada estándar. Una vez procesadas todas las líneas, se ejecuta el código incluido en el bloque END, si es que existe.

Operadores de asignación en awk

Ya hemos indicado previamente que con awk podemos ampliar distintos operadores. Éstos son los que se indican seguidamente:

= += −= *= /= %= ^= Operadores de asignación. Se admite tanto la asignación absoluta (variable = valor) como la que utiliza un operador (el resto de los modos). Con el ejemplo del primer tipo de asignación, podemos poner el siguiente:

datos = datos +$2

Esto podría haberse hecho de una forma más compacta usando el operador +=, tal y como se muestra a continuación:

datos += $2

Este segundo caso es idéntico al primero en cuanto a funcionalidad se refiere, pero es mas compacto. Los operadores +, −, *, /, % y ^significan suma, resta, multiplicación, división, resto de la división entera y exponensiacion, respectivamente.

? Es igual a la expresión condicional empleada en el lenguaje C. Su formato es el siguiente:

expr1 ? expr2 : expr3

Esto debe entenderse como sigue: si expr1 es cierto, el valor de la expresión es expr2; de otro modo, será expr3. Solo se evalúa expr2 o expr3.

|| OR lógico.

&& AND lógico.

− !− Coincidencia y no coincidencia de expresiones regulares

< > <= >= != == Operadores relacionales

blanco Concatenación de cadenas.

+− Suma resta

* / % Multiplicación, división y modulo (resto de la división entera)

+ − ! Más unario, menos unario y negación logica.

^ Exponenciciacion

++ −− Incremento y decremento, tanto en forma de prefijo como de sufijo

$ Referencia a campo.

Matrices con awk

awk nos permite trabajar con matrices. Si la matriz la denominamos datos, la forma de referenciar cada uno de los elementos consiste en utilizar el nombre de la matriz y a continuación , entre corchetes, el numero de elemento. De este modo, datos [34] es el elemento numero 34 de la matriz. Vamos a poner un ejemplo en el que almacenemos el campo numero nueve de cada línea del archivo de entrada en una matriz a para finalizar, visualizaremos toda la matriz. El programa que debemos emplear el siguiente:

$ cat matriz

almacena el campo nueve en una matriz

visualiza la matriz

{ a [NR] = $9}

END { for (i=1; i < NR; i++) print a [I] }

$

La forma de invocarlo será la indicada a continuación:

$ awk − f matriz archivo (s)

En el programa anterior hemos utilizado un bucle for. En un punto posterior mencionaremos de forma ampliada las sentencias de control de flujo y las menciones que podemos emplear con awk.

Variables mantenidas por awk

En algún ejemplo anterior ya hemos utilizado alguna de estas variables, por ejemplo NF, FS, $0, etc. A continuación vamos a dar un listado mas completo de estas variables (no se incluyen todas).

FILENAME Es el nombre del archivo que esta siendo procesado. Si no se ha especificado ningún archivo desde la línea de ordenes, el valor de esta variable será − (entrada estándar).

FNR Es el numero de la línea del archivo que esta siendo procesado.

FS Indica cual es el carácter separador de campos (por defecto es el espacio en blanco)

NF Es el numero de campos presentes en la línea que esta siendo procesado.

NR Indica el numero total de líneas que han sido procesadas.

OFS Es el separador de campos para la salida. Por defecto es el espacio en blanco.

ORS Es el separador de líneas de salida. Por defecto es el carácter de nueva línea.

RS Es el separador de líneas de entrada. Por defecto es el carácter de nueva línea.

$0 Representa la línea que se está procesando.

$n Representa el campo n de la línea que se está procesando.

Sentencias de control de flujo

awk es un autentico lenguaje de programación, y como tal es capaz de trabajar con sentencias de control de flujo. Este tipo de sentencias vamos a describirlas a continuación.

if

if (condición)

orden

[else]

[orden]

Si la condición que se evalúa es cierta, se ejecuta la orden u ordenes colocadas después del if. Si la condición no es cierta, se ejecutan las colocadas después del else (si es que existe). La condición puede ser cualquier expresión que utilice operadores relacionales, así como operadores de correspondencia de patrones. Si se deben ejecutar varias órdenes , tanto después del if como después del else, estas deben ser colocadas entre llaves.

While

While (condición)

Orden

Si verifica la condición, se ejecuta la orden. Las posibles condiciones son las indicadas anteriormente al hablar de if. Si se deben ejecutar varias ordenes dentro del bucle, estas deben ir entre llaves.

do

do

orden

while (condición)

En este caso se ejecuta la orden indicada dentro del cuerpo do while. Si al evaluar la condición esta se verifica, se vuelve a ejecutar la orden. En el caso de que queramos ejecutar varias ordenes en el cuerpo del bucle estas deben de ir entre llaves.

for

Esta orden tiene dos modos de operar. Las sintaxis del primer modo es la siguiente:

For (i= mínimo; i < máximo; i ++)

Orden

En este caso, mientras el valor de la variable i este comprendido entre mínimo y máximo, se ejecuta la orden indicada. En el caso de especificar varias ordenes, estas deben de ir entre llaves. Para la condición de finalización del bucle (i < máximo), se pueden emplear otros operadores relaciónales. En el campo de progreso del bucle (i++) se pueden emplear ++ y −−, tanto en forma pre como post.

El segundo modo se muestra a continuación:

For (elemento in matriz)

Orden

En este caso, para cada elemento de la matriz se ejecuta la orden indicada. En caso de especificar varias ordenes, estas deben de ir entre llaves. Para referirnos a cada elemento de la matriz utilizaremos la expresión matriz [elemento], donde elemento es el numero de ítem dentro de la matriz.

break

break

Esta sentencia se emplea para salir de un bucle while o for. Con ella podemos evitar iteraciones en caso de detectar que un bucle no tiene sentido que continue su repetición.

continue

continue

Esta sentencia nos permite pasar a procesar la siguiente iteración dentro de un bucle while o for, saltando todas las posibles ordenes posteriores dentro del bucle.

exit

exit

con esta sentencia se dejan de ejecutar instrucciones y no se procesas mas archivos. Solo se ejecutaran los procedimientos indicados en el patron END. Así pues, exit sirve para finalizar el procedimiento de archivos por parte de awk.

Ordenes de entrada − salida

print

print [argumentos] [destino]

Con esta orden podemos imprimir los argumentos especificados en la salida. Los argumentos son normalmente campos, aunque también pueden ser cualesquiera de las variables de **awk.** Para visualizar cadenas literales, debemos ponerlas entre dobles comillas. Si los argumentos de print son separados por

comas, en la salida serán separados por el carácter indicado en la variable OFS. Si los argumentos son separados por espacios en blanco, la salida será la concatenación de los argumentos. El parámetro destino puede ser una expresión de redirección o entubamiento. De este modo, podemos redirigir la salida por defecto.

printf

printf [formato [, expresión (es)]]

Esta orden se utiliza para visualizar con formato las expresiones que le indiquemos. Su sintaxis es muy similar ala empleada en la función printf descrita en el lenguaje C. Esta orden también es capaz de interpretar secuencias de escape como el carácter de nueva línea \ n o el tabulador \t . los espacios y el texto literal que deseamos visualizar deben ir entre comillas dobles. Por cada expresión que deseamos visualizar, debe existir su correspondiente formato.