

Manual de introducción al comando awk**Introducción**

Algunas de las limitaciones de sed se remedian por medio del comando awk, pues aunque se parece mucho a sed, los detalles se basan más en el lenguaje de programación C que un solo editor de textos. Su uso es muy parecido:

```
awk 'programa' archivos
```

Sin embargo, la constitución del programa es distinta:

```
patron { acción }
patrón { acción }
patrón { acción }
...
```

A semejanza del comando sed, awk tampoco modifica los archivos de entrada, solamente la salida que imprime. El comando awk lee la entrada de un archivo un renglón a la vez, para el cual compara con cada patrón en orden, de esta manera, para cada patrón que concuerde se efectúa la acción correspondiente.

Los patrones son expresiones regulares como en el comando egrep o condiciones complicadas como en el lenguaje C. Por ejemplo:

```
awk '/expresión regular/' { print }' archivos
awk '/expresión regular/' archivos
```

Esta instrucción consiste en imprimir aquellas líneas que concuerden con la expresión regular, si se omite una acción, se imprime, por lo que ambas líneas muestran el mismo resultado. En caso de omitir el patrón, la parte de la acción se ejecuta a cada línea de entrada.

También es posible, al igual que sed, presentar el programa a awk en forma de archivo como sigue:

```
awk -f archprog archivos
```

A diferencia del shell, en awk, sólo los campos empiezan con un \$, las variables no tienen adornos, por ejemplo, awk reconoce los campos \$1, \$2 y así sucesivamente hasta \$NF, donde NF es una variable igual al total de campos. En el ejemplo de abajo, NF es igual a 5, pero \$1 representa a cada usuario en sesión, por línea procesada.

```
# who
root    pts/0    Dec 25 10:30
margo   pts/1    Dec 25 13:37
wendy   pts/2    Dec 25 16:36
```

De manera que podemos imprimir solo la hora en que entró en sesión cada usuario:

```
# who | awk '{ print $1,$5 }'
root    pts/0    Dec 25 10:30
margo   pts/1    Dec 25 13:37
wendy   pts/2    Dec 25 16:36
```

Awk supone que el espacio en blanco (ya sea un número de espacios o de tabuladores) separa los campos, a excepción del que se encuentra al principio. Los espacios al principio de línea se consideran elementos del primer campo. Pero el separador puede cambiarse por cualquier otro carácter por medio de la opción -F, donde c es el carácter que usará awk como separador. Por ejemplo:

```
#sed 3q /etc/passwd | awk -F: '{ print $1 }'
```

root
bin
daemon

Awk es más fácil de utilizar que sed en operaciones como la anterior y es mas lento al iniciar y ejecutar cuando existe mucho texto a la entrada.

Impresión

Además de reconocer los campos \$1, \$2, etc. awk reconoce \$0 como el conjunto de campos que se encuentran en la línea, es decir, la línea entera sin cambios. También la variable NR que indica el número de línea actual y NF el número de campos por cada línea. A la salida, cuando se imprimen varios campos, el separador por defecto que se utiliza es un espacio en blanco; sin embargo, este puede cambiar a través de otra proposición llamada printf que controla por completo la salida. Ejemplo:

```
awk '{ printf "%4d %s\n", NR, $0 }'
```

La instrucción anterior debería imprimir el número de línea dentro de un campo con 4 espacios. Los caracteres %4d especifican un entero decimal dentro de un campo de cuatro dígitos de ancho, mientras que %s especifica una cadena y por último \n representa una línea nueva al final de cada renglón, por lo que cada línea de entrada procesada se imprimirá en un renglón nuevo, de lo contrario se sobrescribiría cada línea.

También se puede redireccionar la salida de print o printf hacia archivos e interconexiones, pero debe estar acompañada de > y un nombre de archivo entre comillas simples; de igual manera, con >> se anexa en vez de sobrescribir en el archivo indicado entre comillas. Para las interconexiones se utiliza el símbolo pipe '|'.

Patrones

Para preguntar si hay una similitud con un patrón, se hace uso, básicamente, de una sentencia de control (if en lenguaje C), esta permite hacer uso de operadores de comparación y concordancia para identificar un patrón dentro de una línea, además de otras funciones predefinidas como length() que obtiene la longitud de una cadena situada entre sus paréntesis.

Por ejemplo, para una concordancia se utiliza el signo '~', pero requiere que la expresión regular se encuentre encerrada entre diagonales. También a un patrón puede anteponérsele el símbolo ! para negarlo. Ejemplo:

!(\$2 == "")	Pregunta en c/línea si el campo 2 no está vacío
length(\$2) == 0	Pregunta si la longitud del campo 2 es cero
\$2 !~ ./	Pregunta si el campo 2 no concuerda con una línea con cualquier carácter

\$2 ~ /amor/ Pregunta si el campo 2 concuerda con el patron 'amor'

También se puede enviar a la salida, algunos comentarios más informativos que expresen cuando existió un error o simplemente notificar que algo sucedió, por ejemplo, para indicar que un renglón es demasiado largo:

```
awk 'length($0) > 60 { print "Linea", NR, "muy larga:", substr ($0,1,60) }'
```

Esta acción impide que se impriman líneas mayores a 60 caracteres, en lugar de eso, imprime un mensaje notificando que la línea es muy larga y también incluye la función substr que produce que solo se imprima la subcadena del caracter 1 al 60 de la actual que representa \$0, es decir, aquella que es muy larga. Si se hubiera omitido el 60, el cual marca el final de la subcadena, se imprimiría toda la línea.

Los patrones BEGIN y END

awk facilita los patrones especiales BEGIN y END, los cuales se refieren a las acciones a ejecutar antes de leer el primer renglón de entrada y después de leer el último. Generalmente son empleados para inicializar variables, imprimir encabezados o posicionar separadores de campo por medio de la variable FS:

```
awk 'BEGIN { FS = ":" }
>      $2 == "" ' /etc/passwd
ò
awk 'END { print NR }' /etc/passwd
```

Operaciones

La verdadera fuerza de awk se entra en su capacidad de hacer cálculos sobre los datos de entrada. Se facilita el conteo, sumar y promediar, entre otras. Por ejemplo, para sumar columnas de números:

```
      { s = s + $1 }
END    { print s }
```

Este programa debería encontrarse en un archivo (prog.txt) y en otro se debería encontrar una serie de números en la primer columna (num.txt). Al final, la instrucción desde el shell sería

```
awk -f prog.txt num.txt
```

Lo mejor de todo es que respeta los números decimales, a diferencia de las operaciones que se realizan directamente sobre el shell.

awk ofrece los mismos operadores aritméticos que el lenguaje C, de manera que en el ejemplo anterior $s = s + \$1$ será lo mismo que $s += \$1$. Todos los operadores aritméticos y lógicos se muestran a continuación por orden creciente de precedencia:

= += -= *= /= %=	Asignación
	Comparación OR
&&	Comparación AND
!	Negación

> >= < <= == != ~ !~	Operadores relacionales
nothing	Concatenación de cadenas
+ -	más, menos
* / %	multiplicar, dividir, residuo
++ --	incrementar, decrementar (prefijo ó posfijo)

Funciones predefinidas

Existen un conjunto de funciones predefinidas en awk, las cuales se listan a continuación y pueden ser utilizadas de una manera muy simple.

cos (expr)	Obtiene el coseno de expr
exp (expr)	Obtiene el exponencial, es decir e^{expr}
getline ()	Lee la siguiente línea de entrada, devuelve 0 si es fin de archivo y 1 si no.
index (s1, s2)	Devuelve la posición de la cadena s2 en s1, pero 0 si no la encuentra
int (expr)	Realiza un cast como en lenguaje C para que expr sea considerado entero
length (s)	Obtiene la longitud de la cadena s
log (expr)	Obtiene el logaritmo de expr
sin (expr)	Obtiene el seno de expr
split (s, a, c)	Divide s en elementos del arreglo a, utilizando el separador c
sprintf (fmt, ...)	Da formato a ... según la especificación fmt
substr (s, m, n)	Substrae una cadena dentro de otra s, iniciando en m y terminando en n

Variables y arreglos

Las variables se definen al utilizarse, se inicializan a cero por defecto pero no se declaran previamente. Las variables en awk también guardan tanto números como cadenas de caracteres, la diferencia depende del contexto a tratar. En termino generales, una expresión aritmético como `s+=$1` permitirá almacenar un valor numérico en s, mientras que otro contexto como `s="abc"`, almacenará una cadena en la misma variable.

En casos ambiguos como `x>y` se utiliza el valor de cadena a menos que los operandos sean totalmente numéricos. Las variables de cadenas se inicializan a la cadena vacía.

En awk existen también, variables predefinidas que utiliza el comando para su funcionamiento general, estas se listan a continuación:

FILENAME	Nombre del archivo de entrada actual
FS	Caracter separador de campo
NF	Numero de campos en cada registro o línea
NR	Numero de línea actual
OFMT	Formato de salida para números (%g por defecto, según printf())
OFS	Cadena separadora de campo de salida (defecto: espacio)
ORS	Cadena separadora de línea (defecto: nueva línea)
RS	Caracter separador de línea de entrada (defecto: nueva línea)

Los arreglos, mientras tanto, son parecidos al lenguaje C y otros lenguajes de programación. Como las variables, no necesitan ser declarados; el tamaño de un arreglo está limitado por la memoria de la máquina. Por ello, si un archivo muy extenso se está transfiriendo a un arreglo, probablemente se agotaría la memoria rápidamente.

De la misma manera en que el procesamiento normal de la entrada divide cada renglón de entrada en campos, es posible efectuar la misma operación para la separación de campos y almacenar el resultado en un arreglo, por medio de la función split (s, arr, sep), donde s representa la cadena original, arr el arreglo donde se almacenará cada campo, sep el separador de campos, el cual es opcional, pues el defecto es el espacio. Por último, el valor que devuelve la función split es el total de elementos almacenados en arr. Ejemplo:

```
arr[1] = "lola"
arr[2] = "beltran"
ó
s = "lola beltrán"
n = split ( s, arr )

#sed 1q /etc/passwd | awk '{split ($0,a,":"); print a[1]}'
root
```

También existen los arreglos asociativos, puesto que uno de los problemas ordinarios consiste en acumular un conjunto de valores para un conjunto de nombres. Esto es, que en lugar de índices numéricos, también se pueden utilizar cadenas aleatorias. Ejemplo:

```
honor[susana] = 10
honor[felipe] = 9.8
honor[marco] = 9.5
honor [patricia] = 9.2
```

awk ofrece una manera adecuada de manipular este tipo de arreglos, por medio de una variación del for, que se puede ver en el apartado de control de flujo.

La realización de la memoria asociativa se vale de un esquema de hashing para garantizar que el acceso a cualquier elemento tarde el mismo tiempo aproximado que las demás y que por lo menos, el tiempo no depende de cuántos elementos contenga el arreglo. La memoria asociativa es eficiente en tareas como contar todas las palabras de una entrada.

Cadenas

La diferencia entre sed y awk, es que sed se utiliza mayormente para editar o filtrar algún texto de entrada, es decir, es un sistema simple de entrada-proceso-salida; mientras que awk se utiliza en tareas que de verdad requieren programación. Es por eso que hace uso de sentencias de control de flujo.

En awk no hay operadores explícitos de concatenación de cadenas; éstas se concatenan cuando están contiguas.

Shell

Como ya se vió anteriormente y al igual que la creación de scripts de shell, en awk se pueden identificar los campos de una entrada (salida de algún comando del shell actual), por medio del número de campo, precedido por '\$', es decir, para el campo uno, será \$1, para el segundo, \$2 y así sucesivamente. Sin embargo, en el caso de que se deseen pasar parámetros del shell a un programa awk, se puede realizar por medio de un uso correcto de las comillas. Por ejemplo:

Suponiendo que field es un programa de awk que consiste en imprimir solo una columna de cada registro, para ello, se le necesita indicar con un parámetro, el número de columna. El comando tecleado en el shell, es de la siguiente forma:

```
# who | field 1
```

Para pasar el argumento como 1 al comando awk, se hace lo siguiente:

```
awk '{ print $1 }'
```

De esta manera el \$1 que se refiere al parámetro del shell, se mantiene fuera de las comillas simples y se reemplaza su valor correcto. Otra manera es de la siguiente forma:

```
# awk "{ print \$$1 }"
```

El problema no estriba en llevar el control de si uno está dentro o no de las comillas, sino en el hecho de que, en su redacción actual, tales programas no pueden leer más que su entrada estándar; no hay manera de pasarles el parámetro n y una lista arbitrariamente larga de nombres de archivo. Esto requiere programación con shell.

Control de flujo

Awk permite controlar el flujo de la programación para tratar una entrada, básicamente son los mismos ciclos de cualquier lenguaje de programación, pero se asemeja mas al lenguaje C, como en muchas otras cosas. A continuación se presenta cada sentencia:

Sentencia IF-ELSE

```
if (condición)
    proposición 1
else
    proposición 2
```

Sentencia FOR

```
for (expresión1; condición; expresión2)
    proposición
```

Sentencia FOR para arreglos asociativos

```
for (var in array)
    proposición
```

Sentencia WHILE-DO

```
expresión 1
while (condición){
    proposición
    expresión 2
}
```

Por ejemplo:

```
awk ' BEGIN { NR = 1 }
```

```
while ( NR != 5 ){
    if ( i == NF )
        print "Se llegó al límite"
    else
        print i, $0
    NR+=1
}' $*
```

Este pequeño programa de awk, lee las primeras 5 líneas de una entrada dada, y cuando llega a la fila o línea 5, indica que se ha llegado al límite.

```
awk ' { line[ NR ] = $0}
END { for (i = NR; i > 0; i--) print [ i ] } ' $*
```

Ahora, este programa de awk, imprime las líneas de una entrada en orden inverso.

Usos de awk

El uso común de patrones se refiere a la validación de datos, puesto que la mayoría de ellas consiste en buscar los renglones que cumplen o no con un criterio en específico.

Awk tiene la convención de que los comentarios inician después de un '#' y toma el resto de la línea como comentario.

Los usos de awk consisten en programas de una o dos líneas que actúan como filtros dentro de una interconexión mayor. De esta manera, la mayoría de problemas se resuelve aplicando un solo filtro para cada interconexión, es decir, subdividiendo el problema mayor. Sin embargo, el concepto resulta restricto.

La salida producida por los programas escritos en UNIX, está en un formato reconocido como entrada por otros programas. Los archivos filtrables contienen líneas de texto sin encabezados decorativos, terminadores, ni líneas en blanco,. Cada renglón es un objeto de interés, de ahí que programas como wc y grep puedan contar elementos interesantes o buscarlos por nombre. Al momento de que se cuenta con mas información para cada objeto, el archivo permanece en su formato pero ahora agregando la división por columnas con campos separados mediante blancos o tabuladores, de esa manera programas como awk pueden seleccionar, procesar o reorganizar la información de manera sencilla.

Cada filtro escribe sobre su salida estándar el resultado de procesar los archivos dados como argumentos o la entrada estándar si no se dan argumentos. Estos especifican la entrada pero nunca la salida, de esta manera, la salida de un comando puede introducirse en una interconexión. Los argumentos opcionales preceden a cualquier nombre de archivo y por último, los mensajes de error se escriben sobre el error estándar, por lo que no desaparecen a lo largo de las interconexiones.

Los comandos individuales, difícilmente, pueden hacer uso de tan sencilla estructura.