Ejemplos con awk: Una breve introducción

por Javier Palacios Bermejo

Sobre el Author:

Está realizando su tesis doctoral en Astronomía en una universidad española, donde está encargado de la administración del cluster . El trabajo diario se lleva a cabo con máquinas unix y, tras unos primeros intentos infructuosos, se consiguió instalar linux-slackware. Varias actualizaciones después sigue funcionando, mejor que algunos otros de los unix propietarios que se corren en las máquinas del cluster.

Resumen:

Este artículo pretende ser una breve introducción al viejo comando/programa de unix `awk`. Aunque no tan popular como el shell o muchos otros lenguajes de scripting, es una herramienta muy potente cuando hay que tratar con información agrupada en tablas. No está enfocado como un tutorial, sino que se desarrollan algunos ejemplos de uso reales, para mostrar con cierto detalle sus capacidades.

Contenidos:

La idea de escribir este texto surgió por la lectura de un par de artículos aparecidos en LinuxFocus y escritos por Guido Socher. Uno de ellos versaba sobre find y comandos relacionados y me hizo ver que al parecer no era el único que usaba todavía la línea de comandos, en lugar de bonitos GUI que consiguen que no sepas como se hacen las cosas (que es el camino por el que tiró Windows hace mucho). El otro de los artículos trataba sobre expresiones regulares que, aunque apenas mencionadas en este artículo, resulta muy conveniente conocer para sacar el mayor partido posible a `awk` y algún otro de los comandos sobre los que pensaba hablar inicialmente en este artículo (`sed` y `grep` principalmente). algunos otros comandos.

La pregunta clave es si es realmente útil este comando,

y la respuesta es que sí. Le puede resultar útil a un usuario para procesar ficheros de texto, reformatearlos, etc... Para un administrador, awk es, simplemente, una utilidad casi imprescindible. Basta con pasear por `/var/yp/Makefile`, para darse cuenta de ello.

Introducción al `awk`

Supe de su existencia hace tanto que no lo recuerdo bien. Fue cuando un compañero tenía que trabajar con unos output impresionantes en un pequeño Cray y estuvo investigando muchas posibilidades de clasificación. La página man del `awk` en el Cray era de lo más exigua, pero el decía que parecía muy bueno para esa tarea aunque no había forma de hincarle el diente.
Mucho tiempo después, se volvió a cruzar en mi vida, mediante una especie de comentario casual (otro sitio, otro compañero), que lo usaba para extraer la primera columna de una tabla:

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

Fácil, verdad ? Esta tarea tan simple requeriría una cierta dosis de programación en C o cualquier otro lenguaje compilado o interpretado.

Una vez aprendida la lección *extrayendo una columna* ya podemos hacer algunas cosillas como añadir una extensión a una serie de ficheros con secuencias como

```
ls -1 pattern | awk '{print "mv "$1" "$1".nuevo"}' | sh
```

Y más aún ...

1. renombrando el interior del nombre
   `ls -1 *viejo* | awk '{print "mv "$1" "$1}' | sed s/viejo/nuevo/2 | sh`
   (aunque en algunos casos fallará, como con fichero_viejo_y_viejo)
2. borrar solo ficheros (puede hacerse con rm nada más, pero qué pasa con alias como 'rm -r')
   `ls -l * | grep -v drwx | awk '{print "rm "$9}' | sh`
   Cuidado al probar esto en un directorio. Borramos ficheros !
3. borrar solo directorios
   `ls -l | grep '^d' | awk '{print "rm -r "$9}' | sh`

o

```
ls -p | grep /$ | awk '{print "rm -r "$1}' | sh
```

Feedback de los lectores: Como Dan Debertin me hizo notar, algunos de los ejemplos anteriores se pueden realizar sin usar el comando grep, solo con las capacidades de *matcheo*del awk que se mencionan unas líneas más abajo.

```
ls -l *|grep -v drwx|awk '{print "rm "$9}'|sh

sería mas ilustrativo de la potencia de AWK en la forma:
ls -l|awk '$1!~/^drwx/{print $9}'|xargs rm

también,
ls -l|grep '^d'|awk '{print "rm -r "$9}'|sh

podría escribirse como
ls -l|awk '$1~/^d.*x/{print $9}'|xargs rm -r

Uso constantemente la siguiente línea para matar procesos:
(digamos que el proceso se llama 'sleep')
ps -ef|awk '$1~/'$LOGNAME'/&&$8~/sleep/&&$8!~/awk/{print $2}'|xargs kill -9

(hay que ajustarlo para la forma que adopte el comando ps en el sistema en
que trabajes. En algunas ocasiones será ps -aux, el número de campos
variará, etc.) Básicamente es "Si el dueño del proceso ($1) soy yo, y si
se llama ($8) "sleep", y no se llama "awk" (en ese caso el comando awk se
mataría a sí mismo), enviar el PID correspondiente ($2) al comando kill -9.".
Y sin usar grep!
```

Cuando, por ejemplo, se repiten los mismos cálculos una y otra vez, estas herramientas resultan una gran ayuda. Y, además, es mucho más divertido escribir un programa de awk que repetir manualmente lo mismo veinte veces.

Aunque nos referimos a él con ese nombre, el awk no es en realidad un comando, de igual forma que el gcc tampoco lo es. Awk es en realidad un lenguaje de programación, con una sintaxis con aspectos similares al C, y cuyo intérprete se invocacon la instrucción awk.

En cuanto a la sintaxis del comando, casi todo está dicho ya:

```
# gawk --help
Usage: gawk [POSIX or GNU style options] -f progfile [--] file ...
        gawk [POSIX or GNU style options] [--] 'program' file ...
POSIX options:          GNU long options:
        -f progfile             --file=progfile
        -F fs                   --field-separator=fs
        -v var=val              --assign=var=val
        -m[fr] val
        -W compat               --compat
        -W copyleft             --copyleft
        -W copyright            --copyright
```

```
        -W help                 --help
        -W lint                 --lint
        -W lint-old             --lint-old
        -W posix                --posix
        -W re-interval          --re-interval
        -W source=program-text  --source=program-text
        -W traditional          --traditional
        -W usage                --usage
        -W version              --version

Report bugs to bug-gnu-utils@prep.ai.mit.edu,
with a Cc: to arnold@gnu.ai.mit.edu
```

Baste destacar que, además de incluir los programas entre comillas sencillas (') en la línea de comandos, se pueden escribir en un fichero que invocamos con la opción `-f`, y que definiendo variables en la línea de comandos `-v var=`val, podemos dotar de cierta versatilidad a los programas que escribamos.

Awk es, básicamente, un lenguaje orientado al manejo de tablas, en el sentido de información susceptible de clasificarse en forma de campos y registros, al estilo de las bases de datos más tradicionales. Con la ventaja de que la definición del registro (e incluso del campo) es sumamente flexible.

Pero `awk` es mucho más potente. Está pensado para trabajar con registros de una línea, pero esa necesidad se puede relajar. Para profundizar un poco en algunos aspectos, vamos a echar un vistazo a algunos ejemplos ilustrativos (y reales).

- Imprimir tablas un poco más bonitas
  Es posible que alguna vez hayamos tenido que imprimir alguna tabla ASCII obtenida de alguna parte como, por ejemplo, las asociaciones de números ethernet, IP y nombres de hosts. Cuando las tablas son realmente grandes, su lectura se hace realmente difícil, y empezamos a echar de menos lo bien que se lee una tabla impresa con LaTeX o, al menos, formateada algo mejor. Si la tabla es sencilla (y/o sabemos usar bien el `awk`), no resulta demasiado difícil, aunque puede hacerse un poco tedioso:

```
BEGIN {
    printf "preambulo LaTeX"
    printf "\\begin{tabular}"
    printf "{|c|c|...|c|}"
    }


{
    printf $1" & "
    printf $2" & "
    .
    .
    .
```

```
    printf $n" \\\\ "
    printf "\\hline"
    }


END {
    print "\\end{document}"
    }
```

Ciertamente no es un programa lo que se dice *genérico*, pero estamos empezando ...
(los \ dobles son necesarios por tratarse del carácter de escape del shell)

- Troceando ficheros de output
  SIMBAD es una base de datos de objetos astronómicos que, entre otras cosas, incluye las posiciones en el cielo de los mismos. En cierta ocasión, necesité hacer búsquedas para construir mapas alrededor de algunos objetos. Como el interface de dicha base de datos permite guardar los resultados en ficheros de texto, podía hacer dos cosas: generar un fichero para cada objeto o darle como input la lista completa, obteniendo un único y enorme *log* con la consulta. Como decidí hacer lo segundo, use `awk` para trocearlo. Obviamente, para ello tuve que aprovechar ciertas características del output.

1. cada solicitud generaba una línea de cabecera, con un formato del tipo
`====> nombre : nlines <====`
El primer campo me permitia saber cuando empezaba un objeto nuevo y el cuarto cuantas entradas correspondían al mismo (aunque ese dato no es imprescindible)
2. el caracter que separaba las columnas dentro de las listas de output era '|'. Eso requería dos líneas de código adicional para poder enviar al output sólo los campos de mi interés.
	`( $1 == "====>" ) {`
	`  NomObj = $2`
	`  TotObj = $4`
	`  if ( TotObj > 0 ) {`
	`  FS = "|"`
	`  for ( cont=0 ; cont<TotObj ; cont++ ) {`
	`  getline`
	`  print $2 $4 $5 $3  >> NomObj`
	`  }`
	`  FS = " "`
	`  }`
	`  }`

NOTA: Como en realidad no daba el nombre del objeto, era un poco más complicado, pero pretende ser un ejemplo ilustrativo.	

- Jugeteando con el spool del mail

```
BEGIN {
  BEGIN_MSG  = "From"
  BEGIN_BDY  = "Precedence:"
  MAIN_KEY   = "Subject:"
  VALIDATION = "[RESUMEN MENSUAL]"

  HEAD = "NO"; BODY = "NO"; PRINT="NO"
  OUT_FILE = "Resumenes_Mensuales"
  }

  {

  if ( $1 == BEGIN_MSG ) {
    HEAD = "YES"; BODY = "NO"; PRINT="NO"
    }

  if ( $1 == MAIN_KEY ) {
    if ( $2 == VALIDATION ) {
      PRINT = "YES"
      $1 = ""; $2 = ""
      print "\n\n"$0"\n" > OUT_FILE
      }
    }

  if ( $1 == BEGIN_BDY ) {
    getline
    if ( $0 == "" ) {
      HEAD = "NO"; BODY = "YES"
    } else {
      HEAD = "NO"; BODY = "NO"; PRINT="NO"
      }
    }

  if ( BODY == "YES" && PRINT == "YES" ) {
    print $0 >> OUT_FILE
    }
  }
```

Tal vez administramos una lista de correo. Tal vez, de vez en cuando, se envían a la lista mensajes especiales (p.e. resúmenes mensuales) con algún formato determinado (p.e. un subject tipo '[RESUMEN MENSUAL] mes , dept'). Y de repente, se nos ocurre a fin de año recopilar todos los resúmenes, separándolos de los demás mensajes.
Esto podemos hacerlo usando el awk con el spool del mail y el programa que tenemos a la izquierda

Hacer que cada resumen vaya a un fichero requiere tres líneas adicionales, y hacer también que, por ejemplo, cada departamento vaya a un fichero diferente supone unos pocos caracteres más.

NOTA: Todo este ejemplo está basado en cómo creo yo que estan estructurados los mails en el spool. Realmente no se como lo hacen, aunque me funciona (de nuevo, en algunos casos fallará, como siempre).

Programas como éstos sólo necesitan 5 minutos pensando y 5 escribiendo (o más de 20 minutos sin pensar, mediante ensayo y error que es como resulta más divertido).
Si hay alguna forma de hacerlo en menos tiempo, quiero saberla.

He usado el awk para muchas otras cosas (como generación automática de páginas web con información obtenida de una base de datos) y se lo suficiente de programación como para estar seguro de que se pueden hacer con él cosas que ni siquiera se me han ocurrido.

Sólo hay que dejar volar la imaginación.

Un problema

El único problema del awk es que necesita información tabular perfecta, sin huecos: no puede trabajar con columnas de anchura fija, que son de lo mas común. Si el input del awk lo generamos nosotros mismos, no es muy problemático: elegimos algo realmente raro para separar los campos, lo fijamos luego con la variable FS y ya está. Pero si solo tenemos el input, esto puede ser más problemático. Por ejemplo, una tabla tipo

```
1234  HD 13324  22:40:54 ....
1235  HD12223   22:43:12 ....
```

no se podria tratar con el awk. Entradas como esta a veces son necesarias, aparte de ser bastante comunes. Aún así, rizando el rizo, si sólo tenemos una columna con esas características no todo está perdido (si alguien sabe manejarse con más de una en un caso general, soy todo oídos).
En una ocasión tuve que enfrentarme a una de esas tablas, similar a la descrita más arriba. La segunda columna era un nombre e incluía un número variable de espacios. Como suele pasar yo necesitaba ordenarla por una columna posterior a ella. Hice varios intentos con el `sort +/-n.m` que tenía el mismo problema de los espacios embebidos.

(y una solución)

Y me di cuenta de que la columna que yo quería ordenar era la última. Y de que awk sabe cuantos campos hay en el registro actual, por lo que bastaba ser capaz de acceder al último (unas veces $9, otras $11, pero siempre el NF). Total, que un par de pruebas, arrojaron el resultado deseado:

```
{
    printf $NF
    $NF = ""
    printf " "$0"\n"
}
```

Y obtengo un output igual al input, pero con la última columna movida a la primera posición, y *sorteo* sin problemas. Obviamente, el método es facilmente ampliable al tercer campo empezando por el final, o al que va despues de un campo de control que siempre tiene el mismo valor, porque es el que usamos al extraer nuestra subtabla de la base de datos original ...
Sólo deja volar tu imaginación de nuevo.

Profundizando en el awk

Trabajando sobre lineas matcheadas

Hasta ahora, casi todos los ejemplos expuestos procesan todas las lineas del fichero de entrada. Pero, como claramente explica la página de manual, es posible hacer que un cierto grupo de comandos procese tan sólo unas ciertas líneas por el simple método de incluir la condición antes de los comandos, al modo del segundo de los ejemplos anteriores. La condición que debe satisfacer la línea puede llegar a ser bastante flexible, desde una expresión regular, hasta un test sobre los contenidos de alguno de los campos, pudiendo agruparse condiciones en base a operadores lógicos.

awk como lenguaje de programación

Como todo lenguaje de programación, awk implementa todas las estructuras de control necesarias, así como un conjunto de operadores y funciones predefinidas, para manejar números y cadenas. Su sintaxis es en general muy parecida a la del C, aunque difiere de él en algunos aspectos.

Y, por supuesto, también es posible incluir funciones definidas por el usuario, usando la palabra **function**, y escribiendo los comandos como si se tratara de procesar una línea normal del fichero de entrada. E, igualmente, aparte de las variables escalares habituales, tambíen es capaz de manejar arrays de variables.

Incluyendo librerías

Como suele pasar con todos los lenguajes, hay una cierta serie de funciones que son bastante comunes, y llega un momento en que cortar y pegar no es la mejor forma de hacer las cosas. Para eso se inventaron las librerías. Y, al menos con la versión GNU de awk, es posible incluirlas dentro del programa awk. Pero eso es usar awk como una herramienta de trabajo mucho más seria de lo que se pretende mostrar en este artículo, aunque deja claro el nivel de complejidad que puede llegar a alcanzar el awk.

Conclusiones

Ciertamente, puede no ser tan potente como numerosas herramientas que se pueden usar con la misma finalidad. Pero tiene la enorme ventaja de que, en un tiempo realmente corto, permite escribir programas que, aunque tal vez sean de un solo uso, están totalmente adaptados a nuestras necesidades, que en muchas ocasiones son sumamente sencillas.

`awk` es ideal para los propósitos con los que se diseño: leer ficheros línea por línea y procesar en base a los patterns y cadenas que encuentre en ellas.

Ficheros del sistema como el `/etc/password` y muchos otros, resultan sumamente fáciles de tratar mediante el `awk`, sin recurrir a nada más.

Y desde luego que `awk` no es el mejor. Hay varios lenguajes de *scripting* con capacidades mucho mayores. Pero `awk` sigue teniendo la ventaja de ser siempre accesible en cualquier instalación, por mínima que esta sea.

Información adicional

Este tipo de comandos tan básicos no suelen estar excesivamente documentados, pero siempre se puede encontrar algo buscando por ahí.

- la sintaxis del `awk` no es igual en todos los *nix, pero siempre hay una forma de saber exactamente qué podemos hacer con el del nuestro particular: `man awk`;
- Como no podía ser de otra forma, O'Reilly tiene un libro sobre el tema: *Sed & Awk (Nutshell handbook)* de Dale Dougherty.
- Una búsqueda en Amazon, nos proporciona algunos otros títulos como *Effective Awk Programming: A User's Guide*, bastante orientado al `gawk`, y media docena más.

En general, todos los libros y manuales de unix mencionan estos comandos. Pero sólo algunos de ellos profundizan un poco y dan información útil. Lo mejor, hojear todos aquellos que pasen por nuestras manos, pues nunca se sabe donde podemos encontrar información valiosa.

Contactar con el equipo de LinuFocus

LinuxFocus 1999

1999-06-05, generated by lfparser version 0.6