Filtros y herramientas

Un **filtro** (o un comando filtro) es un programa que sabe escribir y leer datos por los canales estándares de entrada y salida. Modifica o trata si es preciso el contenido. **wc** es un filtro. Las herramientas no siempre se comportan como filtros. Permiten un determinado número de acciones en archivos y su contenido, como, por ejemplo, dar formato o imprimir.

1. Extracción de los nombres y rutas

El comando **basename** permite extraer el nombre del archivo en una ruta.

```
$ basename /tmp/seb/lista
lista
```

El comando **dirname** efectúa lo contrario, extrae la ruta.

```
$ dirname /tmp/seb/lista
/tmp/seb
```

2. Búsqueda de líneas

Se trata de extraer líneas de un archivo según varios criterios. Para ello, dispone de tres comandos: **grep**, **egrep** y **fgrep**, que leen los datos o bien desde un archivo de entrada, o bien desde el canal de entrada estándar.

a. grep

La sintaxis del comando **grep** es:

```
grep [Opciones] modelo [Archivo1...].
```

El modelo se compone de criterios de búsqueda que se parecen mucho a los criterios ya expuestos para vi, por ejemplo. No hay que olvidar que se debe interpretar estos criterios con el comando **grep**, y no con el shell. Por lo tanto, hace falta cerrar todos los caracteres.

```
$ cat fic4
Cerdo
Ternera
Buey
rata
Rata
buey
$ grep "^[bB]" fic4
Buey
buey
```

El comando **grep** también puede tomar algunas opciones interesantes.

- **-v** efectúa la búsqueda inversa: se visualizan todas las líneas que no corresponden a los criterios.
- **-c** sólo devuelve el número de líneas encontradas, sin mostrarlas.
- **-i** no diferencia las mayúsculas de las minúsculas.
- **-n** indica el número de línea para cada línea encontrada.
- **-l** en el caso de archivos múltiples, indica en qué archivo se ha encontrado la línea.

```
$ grep -i "^b" fic4
Buey
buey
```

b. egrep

El comando **egrep** extiende los criterios de búsqueda y puede aceptar un archivo de

criterios en entrada. Equivale a un grep -E. Emplea como criterios expresiones regulares.

```
egrep -f archivo_criterio archivo_búsqueda
```

Carácter especial	Significado
|	O lógico, la expresión colocada antes o después debe desaparecer.
(...)	Agrupación de caracteres.
[...]	Un carácter tiene esta posición entre los indicados.
. (punto)	Cualquier carácter.
+	Repetición, el carácter colocado antes debe aparecer al menos una vez.
*	Repetición, el carácter colocado antes debe aparecer de cero a n veces.
?	El carácter colocado antes debe aparecer una vez como máximo.
{n}	El carácter colocado antes debe aparecer exactamente n veces.
{n,}	Aparece n veces o más.
{n,m}	Aparece entre n y m veces.
^	En principio de cadena.
$	En final de cadena.

Únicamente «buenas tardes» y «buenas noches» empezarán por una mayúscula o una

minúscula si están solos en una línea:

```
^[bB]uenas(tardes|noches)$
```

Verificación muy escueta de la validez de una dirección IP:

```
echo $IP | egrep '([0-9]{1,3}\.){3}[0-9]{1,3}'
```

Esta línea se descompone de la manera siguiente:

- ([0-9]{1,3}\.){3}: www.xxx.yyy.
  - [0-9]: un carácter entre 0 y 9
  - {1,3}: repetido entre una y tres veces, por lo tanto: x, xx o xxx
  - \.: seguido de un punto
  - {3}: el conjunto tres veces
- Luego [0-9]{1,3}: .zzz
  - [0-9]: un carácter entre 0 y 9
  - {1,3}: repetido entre una y tres veces

c. fgrep

El comando **fgrep** es un grep simplificado y rápido (fast grep) y equivale a un grep -F. Acepta también un archivo de criterios de búsqueda, pero debe tratarse de criterios simples, sin caracteres especiales. Introduzca en el archivo de criterios líneas sencillas (texto y cifras), una búsqueda por línea. Fgrep va a buscar en un archivo meta o un flujo en entrada las líneas que corresponden a cada uno de los criterios.

d. sed

El aprendizaje de sed requeriría todo un libro. Sed es un editor de flujo (Stream Editor) que permite filtrar y transformar texto. Es un poco como un editor que permite modificar texto vía comandos de scripts, pero en un paso y sin edición interactiva. Utiliza un juego extendido de comandos procedente del editor ed. Su sintaxis básica es:

```
sed -e '<cmd>' arch
```

Para utilizar sed, hay que aprender y entender las expresiones regulares. El cuadro del comando **egrep** retoma la sintaxis básica de las expresiones. Cualquier libro sobre sed parte de estas expresiones y recíprocamente.

Sed se utiliza muy a menudo para sustituir valores por otros (sustitución) o suprimir líneas particulares (aunque se podría utilizar grep en este caso). La sintaxis básica de sustitución es la siguiente:

```
s/<antiguo>/<nuevo>/[g]
```

La g final permite realizar una sustitución en toda la línea en caso de haya varias coincidencias. Aquí tiene un ejemplo que sustituye `__NOMBRE__` por `Pepito`:

```
$ echo "Me llamo __NOMBRE__. ¿Te llamas __NOMBRE__?" | sed -e 's/__
NOMBRE__/Pepito/'
Me llamo Pepito. ¿Te llamas __NOMBRE?
$ echo "Me llamo __NOMBRE__. ¿Te llamas __NOMBRE__?" | sed -e 's/__
NOMBRE__/Pepito/g'
Me llamo Pepito. ¿Te llamas Pepito?
```

Puede colocar un valor numérico en el campo nuevo para precisar, si la búsqueda consta de varios elementos agrupados por paréntesis, en qué elemento de los buscados debe trabajar. Aquí tenemos un ejemplo sencillo que añade asteriscos alrededor del nombre pepito:

```
$ echo pepito | sed -e "s/\(pepito\)/**\1**/"
**pepito**
```

Para suprimir todas las líneas vacías o que contienen únicamente espacios:

```
$ sed -e '/^ *$/d'  archivo
```

e. Expresiones regulares

Las expresiones regulares son un campo práctico pero complejo. Ya vimos que la siguiente expresión le permite validar una dirección IP: ([0-9]{1,3}\.){3}[0-9]{1,3}. Sin embargo no es completamente exacta. Observe:

```
$ echo 300.400.500.600 | egrep '([0-9]{1,3}\.){3}[0-9]{1,3}'
300.400.500.600
```

La dirección es inválida. Hay que detallar que solamente se autorizarán los valores comprendidos entre 0 y 255:

- 25[0-5]: los valores entre 250 y 255
- 2[0-4][0-9]: los valores entre 200 y 249
- [0-1]?[0-9][0-9]? : los valores entre 0 y 199
- .|$: el punto o el final de la línea
- {4}: repetido cuatro veces
- \b(…)\b: indica que lo que se encuentra en medio tiene que ser una palabra completa

Obtenemos (035 y 35 son válidos en IPv4):

```
$ echo 300.400.500.600 | egrep '\b((25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)(\.|$)){4}\b'
$ echo 127.0.0.1 | egrep '\b((25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)(\.|$)){4}\b'
127.0.0.1
$ echo 192.168.15.37 | egrep '\b((25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)(\.|$)){4}\b'
192.168.15.37
$ echo 192.168.015.037 | egrep '\b((25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)(\.|$)){4}\b'
192.168.015.037
```

3. Columnas y campos

El comando **cut** permite seleccionar columnas y campos en un archivo.

a. Columnas

La sintaxis es la siguiente:

```
cut -cColumnas [fic1...]
```

Una columna es la posición de un carácter en la línea. El primer carácter es la columna 1; el segundo, la columna 2, y así sucesivamente. Una línea de 80 caracteres dispone de 80 columnas. La numeración empieza en 1. Es el método ideal para archivos planos y con formato fijo, donde cada campo empieza y termina con posiciones dadas.

El formato de selección de columna es el siguiente:

- una columna sola (p. ej. -c2 para la columna 2);
- un intervalo (p. ej. -c2-4 para las columnas 2, 3 y 4);
- una lista de columnas (p. ej. -c1,3,6 para las columnas 1, 3 y 6);
- los tres a la vez (p. ej. -c1-3,5,6,12-).

```
$ cat lista
Producto  precio  cantidades
ratón     30      15
disco     100     30
pantalla  300     20
teclado   45      30

$ cut -c1-5 lista
Produ
ratón
disco
panta
tecla
```

```
$ cut -c1-3,10-12,15
Prorx   cantidades
rat0    15
dis0    30
pan0    20
tec530
```

b. Campos

El comando **cut** también permite seleccionar campos. Se deben delimitar estos campos por defecto por una tabulación, pero el parámetro `-d` permite seleccionar otro carácter (espacio, ;). La selección de los campos es idéntica a la de las columnas.

El carácter separador debe ser único. No es posible poner ni dos, ni tres, ni una cadena de separadores. Para eliminar los caracteres múltiples, utilice tr. Asimismo, el separador por defecto es la tabulación. Ahora bien, por defecto se sustituyen las tabulaciones con espacios dentro de los editores...

```
cut -dc -fCampos [fic1...]
```

Le presentamos unos ejemplos. El archivo lista contiene campos separados por tabulaciones.

```
$ cat lista
Producto  precio  cantidades
ratón     30      15
duro      100     30
disco     100     30
pantalla  300     20
```

```
teclado   45     30
tarjeta   45     30

$ cut -f1 lista
Producto
ratón
duro
disco
pantalla
teclado
tarjeta

$ cut -f1,3 lista
Producto cantidades
ratón    15
duro     30
disco    30
pantalla 20
teclado  30
tarjeta  30
```

Observe que, si invierte el orden de los campos (-f3,1), no obtendrá el efecto deseado: los campos salen siempre en el sentido 1,3. Habrá que usar métodos más complejos, como por ejemplo con awk.

A continuación vemos cómo aislar los nombres de un grupo y sus identificadores respectivos:

```
$ cat /etc/group
seb@slyserver:~> cat /etc/group
at:!:25:
audio:x:17:
avahi:!:106:
beagleindex:!:107:
```

```
bin:x:1:daemon
cdrom:x:20:
console:x:21:
daemon:x:2:
dialout:x:16:seb,esteban,enrique,public
disk:x:6:

$ cut -d: -f1,3 /etc/group
at:25
audio:17
avahi:106
beagleindex:107
bin:1
cdrom:20
console:21
daemon:2
dialout:16
disk:6
```

Si no hay delimitador (tabulación u otro) en una línea, **cut** muestra toda la línea.

4. Recuento de líneas

El comando **wc** (*word count*) permite contar las líneas, las palabras y los caracteres.

```
wc [-l] [-c] [-w] [-w] fic1
```

- `-l`: cuenta el número de líneas
- `-c`: cuenta el número de bytes
- `-w`: cuenta el número de palabras

- `-m`: cuenta el número de caracteres

```
$ wc lista
      12      48     234 lista
```

El archivo lista contiene 12 líneas, 48 palabras y 234 caracteres.

5. Ordenación de líneas

El comando **sort** permite ordenar las líneas. Por defecto, la ordenación se hace sobre toda la tabla en orden creciente. La ordenación es posible a partir de uno o varios campos. El separador de campos por defecto es la tabulación o, al menos, un espacio. Si hay varios campos, el primero es el separador; los demás son caracteres del campo.

La sintaxis de sort ha evolucionado desde hace varios años y Linux ha aplicado un estándar. Además, ya no utiliza la antigua sintaxis basada en +/-. En su lugar, hay que utilizar el parámetro `-k`. La numeración de los campos empieza con 1.

```
sort [opciones] [-k pos1[,pos2]] [fic1...]

$ cat lista
ratón     óptico  30     15
duro      30giga  100    30
duro      70giga  150    30
disco     zip     12     30
disco     blando  10     30
pantalla  15      150    20
pantalla  17      300    20
pantalla  19      500    20
teclado   105     45     30
teclado   115     55     30
tarjeta   sonido  45     30
tarjeta   vídeo   145    30
```

A continuación vemos cómo ordenar por orden alfabético a partir de la primera columna:

```
$ sort -k 1 lista
disco     blando  10     30
disco     zip     12     30
duro      30giga  100    30
duro      70giga  150    30
pantalla  15      150    20
pantalla  17      300    20
pantalla  19      500    20
ratón     óptico  30     15
tarjeta   sonido  45     30
tarjeta   vídeo   145    30
teclado   105     45     30
teclado   115     55     30
```

Algunos parámetros

Opción	Función
-d	Dictionnary sort (ordenación de diccionario). Sólo toma como criterio de ordenación las letras, las cifras y los espacios.
-n	Ordenación numérica, ideal para las columnas de cifras.
-b	Ignora los espacios al principio del campo.
-f	No hay diferencias entre mayúsculas y minúsculas (conversión en minúsculas y luego ordenación).
-r	Reverse, ordenación en orden decreciente.
-tc	Nuevo delimitador de campo c.

Ejemplo: ordenación numérica a partir de los precios por productos en orden decreciente:

```
$ sort -n -r -k 3 lista
pantalla   19     500    20
pantalla   17     300    20
pantalla   15     150    20
duro       70giga  150    30
tarjeta    vídeo   145    30
duro       30giga  100    30
teclado    115     55     30
teclado    105     45     30
tarjeta    sonido  45     30
ratón      óptico  30     15
disco      zip     12     30
disco      blando  10     30
```

También es posible ejecutar la ordenación desde un determinado carácter de un campo. Para ello, debe especificar el «.pos»: -k1.3 empezará la ordenación a partir del tercer carácter del campo 1.

6. Eliminación de las líneas repetidas

El comando **uniq** permite suprimir las líneas repetidas en flujos de entrada o archivos ordenados. Por ejemplo, a continuación se muestra cómo sacar únicamente la lista de los GID realmente utilizados como grupo principal de usuarios:

```
$ cut -d: -f4 /etc/passwd | sort -n | uniq
0
1
2
7
8
12
13
14
25
49
51
```

```
62
...
```

7. Unión de dos archivos

a. En los campos comunes

El comando **join** permite efectuar la unión de dos archivos en función de un campo común. Se deben ordenar los dos archivos en los campos especificados en la unión.

```
join [-tc] [-1 n] [-2 m] fic1 fic2
```

La opción `-t` indica los separadores, `-1` el campo del primer archivo y `-2` el campo del segundo archivo, en los cuales efectuar la unión. Observe que `join` gestiona mal los duplicados y puede que se detenga como consecuencia de ello.

Es posible que el comando **join** no le proporcione el resultado esperado. La razón es que se detiene en cuanto no encuentra correspondencia entre dos líneas.

b. Línea a línea

El comando **paste** agrupa n archivos en uno. Para ello, concatena las líneas de cada uno de los archivos en una sola línea: línea1 de fic1 con línea2 de fic2, línea3 de fic 3, y así sucesivamente. Es un poco lo contrario de cut. El separador por defecto es la tabulación, pero puede precisar un delimitador con `-d`.

```
$ cat fic1
lista_a
```

```
lista_b
lista_c

$ cat fic2
lista_a2
lista_b2
lista_c2

$ paste -d: fic1 fic2
lista_a:lista_a2
lista_b:lista_b2
lista_c:lista_c2
```

8. División de un archivo en partes

a. Recortar

Aquí tenemos un comando muy práctico, **split**, que permite recortar un gran archivo en varios trozos con un tamaño determinado. Los sistemas de archivos no son todos iguales frente al tamaño máximo de un archivo. En Linux, el problema no es habitual, ya que un sistema de archivos de tipo ext4 puede soportar archivos de varias decenas de TB. Pero las bandas magnéticas, o en menor medida los discos removibles, no disponen de esta posibilidad.

Se suelen formatear los pendrives o un disco externo con un sistema de archivos de tipo VFAT, procedente del mundo de Microsoft. Este sistema de archivos, que procede de DOS y luego de Windows 9x, garantiza una compatibilidad entre todos los sistemas (Unix, Windows, Mac OS), ya que quien puede lo más, puede lo menos. VFAT (o más bien FAT16 o FAT32) soporta únicamente archivos de un tamaño máximo de 4 GB. Una imagen ISO de DVD o una carpeta de copia de seguridad no puede entrar en un solo bloque. Por lo tanto, hace falta dividir el archivo en varias partes.

```
split [-l n] [-b n[bkm] [archivo [prefijo]]
```

El comando puede funcionar según dos modos:

- recorte por líneas con `-l`: los archivos de salida tendrán todos n líneas de texto (salvo el último si se da el caso);
- recorte a tamaño fijo con `-b`: los archivos tendrán todos un tamaño fijo de n bytes. El sufijo b indica un tamaño de n bloques (512 bytes), k indica n kB (1024 bytes) y m indica n MB (1024 kB).

Como cualquier filtro, **split** puede coger un flujo de entrada, lo que ocurre si no se indica ningún archivo, o si hay un guión. Un prefijo define el nombre de los archivos en salida. Aquí tenemos un archivo de 1 GB a cortar en partes de 150 MB. El prefijo es fic. Cada archivo de salida se llama ficaa, ficab, ficac, ficad, y así sucesivamente.

```
$ ls -l granarchivo
-rw-r--r-- 1 seb users 1073741824 mar 12 19:47 granarchivo
$ split -b 150m granarchivo fic
$ ls -l fic*
-rw-r--r-- 1 seb users 157286400 mar 12 20:15 ficaa
-rw-r--r-- 1 seb users 157286400 mar 12 20:15 ficab
-rw-r--r-- 1 seb users 157286400 mar 12 20:15 ficac
-rw-r--r-- 1 seb users 157286400 mar 12 20:16 ficad
-rw-r--r-- 1 seb users 157286400 mar 12 20:16 ficae
-rw-r--r-- 1 seb users 157286400 mar 12 20:16 ficaf
-rw-r--r-- 1 seb users 130023424 mar 12 20:16 ficag
```

b. Reconstruir

Una línea basta para reconstruir un archivo dividido con la ayuda de las redirecciones:

```
$ cat fic* > newfic
$ ls -l newfic
-rw-r--r-- 1 seb users 1073741824 mar 12 20:47 newfic
```

9. Sustitución de caracteres

a. Lista de caracteres

El comando **tr** permite sustituir unos caracteres con otros y sólo acepta datos que provengan del canal de entrada estándar, no de los archivos.

```
tr [opciones] original destino
```

El original y el destino representan uno o varios caracteres. Se sustituyen los caracteres originales con los de destino en el orden indicado. Los corchetes permiten definir intervalos.

Por ejemplo, para sustituir la o por la e y la i por la a:

```
$ cat lista | tr "oi" "ea"
Predut   ebjete  precie  cantadades
raten    eptaque 30     15
dure     30gaga  100    30
dure     70gaga  150    30
dasce    zap     12     30
dasce    blande  10     30
pantalla 15      150    20
pantalla 17      300    20
pantalla 19      500    20
teclade  105     45     30
teclade  115     55     30
tarjeta  senade  45     30
tarjeta  vadee   145    30
```

Con este comando, puede convertir una cadena en mayúsculas o en minúsculas.

```
$ cat lista | tr "[a-z]" "[A-Z]"
PRODUCTO  OBJETO  PRECIO  CANTIDADES
RATÓN     ÓPTICO  30      15
DURO      30GIGA  100     30
DURO      70GIGA  150     30
DISCO     ZIP     12      30
DISCO     BLANDO  10      30
```

```
PANTALLA  15     150    20
PANTALLA  17     300    20
PANTALLA  19     500    20
TECLADO   105    45     30
TECLADO   115    55     30
TARJETA   SONIDO  45     30
TARJETA   VÍDEO   145    30
```

Eliminar las repeticiones

Sobre todo, `tr` admite dos parámetros, `-s` (squeeze) y `-d` (delete), que permiten suprimir caracteres, duplicados o no. Es perfecto en el caso de separadores múltiples. A continuación damos un ejemplo práctico en el cual se busca aislar la dirección IP de una máquina.

```
$ /sbin/ifconfig eth0
eth0      Vínculo encap:Ethernet  HWaddr 00:13:D3:D7:A4:6C
          inet adr:10.9.238.170  Bcast:10.9.239.255  Máscara:255.255.252.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500   Metric:1
          RX packets:15054381 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4991811 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:1000
          RX bytes:4157389034 (3964.7 Mb)  TX bytes:374974072 (357.6 Mb)
          Interrupción:22 Dirección básica:0xcc00
```

Sólo le interesa la línea que contiene `inet`:

```
$ /sbin/ifconfig eth0| grep "inet "
          inet dir:10.9.238.170 Bcast:10.9.239.255  Máscara:255.255.252.0
```

Para aislar la dirección IP colocada después de "`inet dir:` " el separador «`:`» puede parecer interesante, pero en este caso un cut nos devolvería "`10.9.238.170 Bcast` ", lo que no conviene. La artimaña consiste en sustituir todos los espacios por un solo «:». El parámetro `-s` sustituye una cadena de n caracteres idénticos por uno solo. Si no se precisa, es el mismo carácter; en caso contrario, se trata de un carácter de sustitución determinado.

```
$ /sbin/ifconfig eth0| grep "inet " | tr -s " " ":"
:inet:dir:10.9.238.170:Bcast:10.9.239.255:Máscara:255.255.252.0
```

Ya sólo falta contar: la dirección IP está en cuarta posición (el primer campo antes del primer «:» está vacío).

```
$ /sbin/ifconfig eth0| grep "inet " | tr -s " " ":" | cut -d: -f4
10.9.238.170
```

Lo mismo pasará con el comando **ip** (ifconfig será obsoleto y ya no se encontrará por defecto en muchos sistemas) :

```
$ ip addr show dev enp0s8 | grep inet | grep -v inet6 | tr -s " " ":" |
cut -d: -f3 | cut -d/ -f1
192.168.0.10
```

b. Tabulaciones y espacios

La mayoría de los editores sustituyen las tabulaciones por espacios. Ahora bien, algunos comandos esperan a obtener tabulaciones como delimitadores de campos (es el caso de cut). Si no puede apañarse con `tr`, tiene a su disposición dos comandos para este caso específico.

El comando **expand** convierte las tabulaciones en espacios. El comando **unexpand** convierte los espacios en tabulaciones. O sea, el archivo lista según el modelo anterior, en el cual se separan las columnas por espacios en lugar de tabulaciones. En el primer caso, el resultado no es el esperado para nada. El comando **cut** intenta sacar el tercer campo de un archivo tabulado. Como no hay tabulaciones, muestra toda la línea.

```
$ cut -f1 lista
Producto    objeto    precio   cantidades
ratón       óptico    30       15
duro        30giga    100      30
duro        70giga    150      30
```

```
disco      zip      12      30
disco      blando   10      30
...
```

El comando **unexpand** con el parámetro `-a` sustituye todas las secuencias de al menos dos espacios por el número necesario de tabulaciones. Esta vez el resultado es correcto.

```
$ unexpand -a lista | cut -f1
Producto
ratón
duro
duro
disco
disco
...
```

10. xargs

El comando **xargs** permite leer los elementos de la entrada estándar (pipe, redirección), delimitados por defecto por un espacio o un retorno de línea, y luego ejecutar un comando, por defecto **echo**, con esos mismos elementos, uno por uno o reformateados.

He aquí algunos ejemplos de operaciones posibles con un archivo llamado input que contiene una lista de líneas, del 1 al 20.

```
$ cat input
1
2
...
20
```

Todas las líneas y palabras reagrupadas:

```
$ cat input | xargs
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

En columnas de dos:

```
$ cat input | xargs -n 2
1 2
3 4
5 6
7 8
9 10
11 12
...
```

Todos los usuarios de /etc/passwd en una sola línea, en orden creciente:

```
$ cut -d: -f1 < /etc/passwd | sort | xargs echo
abrt adm avahi avahi-autoipd bin chrony colord daemon dbus dockerroot ftp
games gdm geoclue gnome-initial-setup hacluster halt haproxy libstoragemg
...
```

Eliminar todos los archivos encontrados por un comando **find**:

```
$ find -name "core*" -size +1m -print | xargs rm -f
```

11. Visualización de texto

a. En pantalla completa

Nada impide desviar cualquier flujo para visualizarlo en la pantalla o por impresora. Aquí presentamos algunos comandos.

- página por página: **pg**, **more**, **less**
- en bloque: **cat**
- al revés: **tac**
- en dump hexadecimal: **hexdump**
- en dump octal (por defecto): **od**
- creación de un banner: **banner**
- formateo para impresión: **pr**
- formateo de párrafo: **fmt**
- numerar las líneas: **cat -n** o **nl**

He aquí algunos ejemplos utilizando siempre el mismo archivo input:

Salida en hexadecimal:

```
$ hexdump input
0000000 0a31 0a32 0a33 0a34 0a35 0a36 0a37 0a38
0000010 0a39 3031 310a 0a31 3231 310a 0a33 3431
0000020 310a 0a35 3631 310a 0a37 3831 310a 0a39
0000030 3032 000a
0000033
$ od -Ax -x input
000000 0a31 0a32 0a33 0a34 0a35 0a36 0a37 0a38
000010 0a39 3031 310a 0a31 3231 310a 0a33 3431
000020 310a 0a35 3631 310a 0a37 3831 310a 0a39
000030 3032 000a
000033
```

Salida en octal:

```
$ od input
0000000 005061 005062 005063 005064 005065 005066 005067 005070
0000020 005071 030061 030412 005061 031061 030412 005063 032061
0000040 030412 005065 033061 030412 005067 034061 030412 005071
0000060 030062 000012
0000063
```

Numeración de las líneas eliminando los ceros:

```
$ nl -nrz input
000001  1
000002  2
000003  3
000004  4
000005  5
000006  6
```

Tomemos el archivo siguiente:

```
$ cat lorem.txt

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,
quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse
cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat
non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
```

El comando **fmt** reformateará los parrafos para que sean más legibles (o para su impresión). Por ejemplo, en columnas de 60 caracteres:

```
$ fmt -t -w 60  lorem.txt
Lorem ipsum dolor sit amet, consectetur adipiscing
  elit, sed do eiusmod tempor incididunt ut labore et
  dolore magna aliqua. Ut enim ad minim veniam, quis
  nostrud exercitation ullamco laboris nisi ut aliquip
  ex ea commodo consequat. Duis aute irure dolor in
  reprehenderit in voluptate velit esse cillum dolore eu
  fugiat nulla pariatur. Excepteur sint occaecat cupidatat
  non proident, sunt in culpa qui officia deserunt mollit
  anim id est laborum.
```

b. El principio de un archivo

Para ver el principio del contenido de un archivo, utilice el comando **head**.

```
head [-c nbcar] [-n nblíneas] [fic1...]
```

El parámetro `-c` permite precisar el número de bytes de encabezamiento que visualizar. Por defecto se visualizan diez líneas. El parámetro `-n` permite indicar el número de líneas que visualizar. Es posible indicar directamente el número de líneas:

```
head [-nblíneas] [fic1...]

$ head -3 lista
Producto objeto   precio  cantidades
ratón    óptico   30     15
duro     30giga   100    30
```

c. Fin y modo de espera de archivo

Para ver las últimas líneas de un archivo, utilice el comando **tail**.

```
tail [+/-valor[b/c]] [-f] [fic1...]
```

Al igual que para head, por defecto se visualizan las diez últimas líneas. El valor `-numlíneas` permite modificar este estado. Use c para indicar un número de caracteres. Una b indica un número de bloques (512 bytes por bloque).

Finalmente, la opción `-f` deja el archivo abierto. Si se inserta más información en él (por ejemplo, un archivo de registros), se visualizará su contenido en tiempo real en la pantalla hasta que el usuario lo interrumpa de manera voluntaria ([Ctrl] **C**).

```
$ tail -5 lista
pantalla  19     500    20
teclado   105    45     30
```

```
teclado   115    55     30
tarjeta   sonido  45     30
tarjeta   vídeo   145    30

$ tail -10c lista
eo       145    30
```

d. Dar formato a una salida

El comando **column** permite dar formato de tabla a la salida de un comando. La opción `-t` determina cuántas columnas se mostrarán en la salida y añade espacios para alinearlas. La opción `-s` permite indicar cuál es el separador.

```
$ column -s: -t /etc/group
root          x  0
daemon        x  1
bin           x  2
sys           x  3
adm           x  4  seb
tty           x  5
disk          x  6
lp            x  7
```

12. Duplicación del canal de salida estándar

En algunos casos, como por ejemplo en la generación de archivos de registros, puede ser necesario colocar el resultado de un comando en un archivo y a la vez filtrar este mismo resultado con otro comando. Para ello, utilice el comando **tee**, que permite duplicar el flujo de datos. Este comando lee el flujo de datos que proviene de otro comando por el canal de entrada, lo escribe en un archivo y restituye este flujo de forma idéntica por el canal de salida. Por defecto, el archivo generado sobrescribe el antiguo si existe.

```
tee [-a] nombre_archivo
```

El parámetro `-a` significa append. En este caso, no se sobrescribe el archivo, sino que se inserta la información al final. Por ejemplo, supongamos que quiere obtener en un archivo la lista de los nombres de usuario y que al mismo tiempo se visualice su número en la pantalla.

```
$ cat /etc/passwd | cut -d: -f1 | tee users | wc -l
      65
$ cat users
root
nobody
nobodyV
daemon
bin
uucp
uucpa
auth
cron
lp
tcb
...
```

13. Comparación de archivos

Los dos comandos que permiten comparar el contenido de dos archivos, o de un archivo y de un flujo, son los comandos **diff** y **cmp**.

a. diff

El comando **diff** indica las modificaciones que hay que aportar a los dos archivos en entrada para que su contenido sea idéntico.

```
diff [-b] [-e] fic1 fic2
```

La opción `-b` permite ignorar los espacios (blank), y la opción `-e` permite generar un

script ed (no lo utilizaremos). Este comando devuelve tres tipos de mensajes:

- `APPEND: línea1 a línea3,línea4, ex 5 a 6,8` significa: en la línea 5 de fic1 hay que insertar las líneas 6 a 8 de fic2 para que sus contenidos sean idénticos.
- `DELETE: línea1,línea2 d línea3, ex 7,9 d 6` significa: se deben suprimir las líneas 7 a 9 de fic1, no existen detrás de la línea 6 de fic2.
- `CHANGE: línea1,línea2 c línea3,línea4, ex 8,12 c 9,13` significa: se debe intercambiar las líneas 8 a 12 de fic1 contra las líneas 9 a 13 de fic2.

En cualquier caso, el signo "<" indica las líneas de fic1 concernientes y el signo ">" las líneas de fic2 concernientes.

```
$ cat lista
Producto  objeto  precio  cantidades
ratón    óptico  30     15
duro     30giga  100    30
duro     70giga  150    30
disco    zip     12     30
disco    blando  10     30
pantalla  15     150    20
pantalla  17     300    20
pantalla  19     500    20
teclado   105    45     30
teclado   115    55     30
tarjeta   sonido  45     30
tarjeta   vídeo   145    30

$ cat lista2
Producto  objeto  precio  cantidades
ratón    botones 30     15
duro     30giga  100    30
duro     70giga  150    30
disco    zip     12     30
disco    blando  10     30
pantalla  15     150    20
pantalla  17     300    20
```

```
pantalla  19    500   20
pantalla  21    500   20
teclado   105   45    30
teclado   115   55    30
```

El archivo lista es el original. En lista2 se ha modificado la segunda línea, se ha añadido una línea pantalla y se han suprimido las últimas dos líneas.

```
$ diff lista lista2
2c2
< ratón         óptico  30     15
---
> ratón         botones 30     15
9a10
> pantalla 21    500    20
12,13d12
< tarjeta sonido  45     30
< tarjeta vídeo   145    30
```

- `2c2`: se deben intercambiar las líneas 2 de lista y lista2 (deben concordar o en óptico o en botones).
- `9a10`: después de la línea 9 de lista (pantalla 19), hay que añadir la línea 10 (pantalla 21) de lista2.
- `12,13d12` : se deben suprimir las líneas 12 y 13 de lista (tarjeta de sonido y vídeo), ya que no existen después de la línea 12 de lista2.

b. cmp

El comando **cmp** compara los archivos carácter por carácter. Por defecto, el comando se para en cuanto encuentra la primera diferencia e indica la posición.

```
cmp [-l] [-s] fic1 fic2
```

El parámetro `-l` detalla todas las diferencias en tres columnas. La primera columna

representa el número de carácter; la segunda, el valor octal ASCII del carácter correspondiente de fic1, y la tercera, el valor octal ASCII del carácter correspondiente de fic2.

La opción `-s` devuelve únicamente el código de error (no visible), al que se puede acceder por echo $?.

```
$ cmp lista lista2
lista lista2 differ: char 38, line 2
$ cmp -l lista lista2
  38 157 142
  39 160 157
  40 164 165
  41 151 164
  42 161 157
  43 165 156
  44 145 163
 182 143 145
 183 154 143
 ...
```

14. Plazo de espera

El comando **sleep** permite esperar el número de segundos indicados. El script se interrumpe durante este tiempo: el número de segundos y un entero comprendido entre 0 y cuatro mil millones (136 años).

```
$ sleep 10
```

15. Controlar el flujo

El comando **pv**, a menudo desconocido, permite responder a una pregunta frecuente: ¿qué pasa en la tubería mientras que los datos pasan de un proceso a otro? ¿Cuántos datos?

¿Cuánto va a tardar? pv es un monitor de flujo. Se intercala generalmente entre uno y otro comando (como tee), y analiza el flujo que recibe antes de reenviarlo a su destino. De esta forma, sabe lo que se ha transferido, y puede mostrar una barra de progreso, por ejemplo. Veamos dos ejemplos:

Copiar un archivo

```
[jolivares@slyserver ~]$ pv test > test2
50MiB 0:00:02 [24,1MiB/s] [================>              ] 52% ETA
```

Duración de la compresión de un archivo

```
pv /boot/vmlinuz-3.14.7-200.fc20.x86_64 | gzip > test.gz
5,26MiB 0:00:00 [6,05MiB/s] [==================================>] 100%
```

16. Las sumas de control

Es habitual, cuando descarga por ejemplo una imagen ISO desde Internet, encontrar junto con la imagen un dato o un archivo de suma de control. En el caso que nos interesa, una suma de control o **checksum** es un cálculo efectuado sobre todo el contenido, o una parte, de un archivo para obtener une huella única. Existen distintos comandos para calcularla y todos tienen la misma sintaxis:

- **md5sum**: suma de control md5, función de hash criptográfica que fue usada durante muchos años. Esta ofrece una huella única. En 1996 se encontró que podrían obtenerse colisiones en los resultados de estas funciones y fue demostrado en 2004: se pueden generar archivos que presenten la misma huella y, por lo tanto, se podría forjar un archivo especial que permitiera que un malware se presentara como un archivo original.
- **sha1sum**: suma de control que usa las funciones de hash SHA-1. Desde 2011 ya no se consideran como seguras: Google consiguió forjar dos archivos PDF que presentaban la misma suma de control SHA-1 en 2017.
- **sha256sum**, **sha512sum**: suma de control que usa las funciones de hash SHA-2,

estas son consideradas como seguras.

Por ejemplo, si accede al servidor https://cdimage.debian.org/debian-cd/current/amd64/iso-cd/, encontrará las imágenes ISO pero también un archivo llamado SHA256SUMS con el contenido siguiente:

```
396553f005ad9f86a51e246b1540c60cef676f9c71b95e22b753faef59f89bee
debian-10.8.0-amd64-netinst.iso
30809f90e18cc501e88e615b45509fd128c2cf9a7f52742a528001898fd35a09
debian-10.8.0-amd64-xfce-CD-1.iso
fe6d8e3bbea8721b379f9cd23266073027316695e0d6ab375a476e26da66232b
debian-edu-10.8.0-amd64-netinst.iso
a5774f23aa4fe9ff6661cae39e2333f6848bc5c1fd63845652debfa16835712c
debian-mac-10.8.0-amd64-netinst.iso
```

Se trata de un archivo que contiene los resultados de la suma de control SHA256 de las imágenes ISO. He aquí el cálculo SHA-2 efectuado en la imagen ISO:

```
$ sha256sum debian-10.8.0-amd64-netinst.iso
396553f005ad9f86a51e246b1540c60cef676f9c71b95e22b753faef59f89bee
debian-10.8.0-amd64-netinst.iso
```

Puede observar que el resultado coincide con la suma de control que encontramos en el archivo. También se puede usar el archivo SHA256SUMS como entrada del comando:

```
$ sha256sum -c SHA256SUMS
debian-10.8.0-amd64-netinst.iso: OK
```