Introducción a Sistemas Operativos: Usando el shell

Clips xxx
Francisco J Ballesteros

1. Editando streams

Ya hemos utilizado *sed(1)* para escribir las primeras líneas de un fichero. Pero puede hacer mucho más. Se trata de un editor similar a *ed(1)* (que fue el editor estándar en UNIX durante mucho mucho, pero hace ya mucho). Sus comandos están también disponibles (en general) en *vi(1)*, un editor "*visual*" que puede utilizarse en terminales de texto que no sean gráficos. La diferencia radica en que `sed` está hecho para procesar su entrada estándar, principalmente. De ahí el nombre *stream ed*.

`Sed` trabaja procesando su entrada línea a línea. En general, para cada línea, aplica los comandos de edición que se indican y después se escribe la línea en la salida.

Un comando en `sed` es una o dos *direcciones* seguidas de una función y, quizá, argumentos para la función. Las direcciones determinan en qué líneas son aplicables los comandos.

Por ejemplo,

```
sed 5q
```

Significa "*en la línea 5*" ejecuta "*quit*". Dado que no se han ejecutado comandos que alteren las líneas, `sed` las escribe tal cual hasta llegar a la línea 5, momento en que termina. Por ello este comando escribe las 5 primeras líneas. La primera línea es la `1` para `sed`.

Sabiendo esto, podemos utilizar `sed` para escribir un rango de líneas. Por ejemplo

```
unix$ seq 15 | sed -n 5,7p
5
6
7
unix$
```

Aquí el flag `-n` hace que `sed` no imprima las líneas tras editarlas y utilizamos el comando `p` para imprimir las líneas entre la `5` y la `7`. Como puedes ver, dos direcciones separadas por una "`,`" delimitan un rango de líneas.

Veamos otro ejemplo, sabemos que `ps` escribe una primera línea con la cabecera que indica qué es cada campo. Por ejemplo:

```
unix$ ps
  PID TTY          TIME CMD
  448 ttys000    0:00.01 -bash
  519 ttys000    0:00.01 acme
...
```

Podemos utilizar

```
unix$ ps | tail +1
  448 ttys000    0:00.01 -bash
  519 ttys000    0:00.01 acme
...
```

como ya vimos o bien

```
unix$ ps | sed -n '2,$p'
  448 ttys000    0:00.01 -bash
  519 ttys000    0:00.01 acme
...
```

Como puedes ver, "`$`" equivale al número de la última línea cuando se utiliza como dirección en `sed`.

De un modo similar podemos eliminar un rango de lineas con el comando `d`:

```
unix$ seq 15 | sed 2,12d
1
13
14
15
unix$
```

¡O varios rangos! El flag `-e` permite utilizar como argumento un comando de edición, y podemos repetirlo el número de veces que haga falta. Nos permite aplicar más de un comando en el mismo `sed`.

```
unix$ seq 15 | sed -e 1,3d -e 5,12d
4
13
14
15
unix$
```

Pero recuerda... ¡`sed` trabaja línea a línea sobre un stream!

```
unix$ seq 15 | sed -e 2,12d -e 1,3d
unix$
```

¡Aquí la hemos liado parda! Este comando no tiene mucho sentido dado que hay varios comandos que se aplican a líneas del mismo rango.

Las direcciones pueden ser expresiones regulares escritas entre "`/`". Esto nos permite seleccionar líneas entre ciertas líneas de interés. Por ejemplo, suponiendo que tenemos un fichero fuente en C con la función `run`, este comando escribe la cabecera y el cuerpo de dicha función:

```
unix$ sed -n  '/^run/,/^}/p <run2.c
run(char *cmd, char *argv[])
{
    ...
}
unix$
```

Para entender por qué no se escriben todas las líneas hasta la última que contenga un "`}`" al principio, piensa cómo evalúa `sed` las direcciones:

- Primero esperará hasta tener una línea que encaja con "`^run`"
- Una vez encontrada, seguirá hasta una línea que encaja con "`^}`".

En cada una de esas líneas ejecutará "`p`" (que imprime la línea) y una vez alcanza la línea de la segunda

dirección, el comando termina por lo que no se vuelve a ejecutar para las líneas que siguen.

Otro comando de los más utilizados es el de sustituir una expresión por otra. Por ejemplo, este comando aumenta la tabulación del texto a que se aplica:

```
sed 's/^/ /'
```

Lo que hace es reemplazar el principio de línea por un tabulador (que hemos escrito como " ".

Supongamos que queremos renombrar todos los ficheros "`.c`" a ficheros "`.C`" por alguna extraña razón. Podemos utilizar `sed` para ello:

```
unix$ echo foo.c | sed 's/\.c$/.C/'
foo.C
```

Así pues

```
unix$ for f in *.c ; do
>    nf=‘echo $f | sed 's/\.c$/.C/’‘
>    mv $f $nf
>    done
unix$
```

hace el trabajo. Cuidado aquí. El comando de `sed` está entre comillas simples, pero todo lo que hay tras el signo "`=`" está entre comillas invertidas. ¡No son iguales!

Por ejemplo, podemos tener ficheros llamados `ch1.w`, `ch2.w`, etc. y llegar al `ch10.w`. En ese momento quizá queramos renombrar `ch1.w` para que sea `ch01.w`, y así hasta el `9`. ¿Por qué? Bueno, de ese modo `ls` los lista en el orden adecuado y otros programas los verán en el orden en que deberían estar. De otro modo `ch11.w` estará listado antes que `ch2.w`.

Una vez más `sed` nos puede ayudar. Para probarlo, vamos a crear esos ficheros, y vamos a utilizar `seq` para crearlos.

```
unix$ for n in ‘seq 12‘; do touch ch$n.w ; done
unix$ ls
ch1.w    ch11.w    ch2.w    ch4.w    ch6.w    ch8.w
ch10.w    ch12.w    ch3.w    ch5.w    ch7.w    ch9.w
unix$
```

Y ahora podemos probar...

```
unix$ for ch in ch[0-9].w ; do
>    echo mv $ch ‘echo $ch | sed 's/\([0-9]\)/0\1/’ ‘
>    done
mv ch1.w ch01.w
mv ch2.w ch02.w
mv ch3.w ch03.w
mv ch4.w ch04.w
mv ch5.w ch05.w
mv ch6.w ch06.w
mv ch7.w ch07.w
mv ch8.w ch08.w
mv ch9.w ch09.w
unix$
```

En lugar de ejecutar `mv` directamente, lo hemos precedido de `echo` para ver lo que va a ejecutar. Si nos convence el resultado podemos añadir "`|sh`" para ejecutarlo o quitar el `echo`.

Pero la parte interesante es "`s/\([0-9]\)/0\1/`". Esto quiere decir que hay que sustituir el texto que encaja con "`[0-9]`" en cada línea por "`0\1`". El "`\1`" significa *el texto que encaja en la primera expresión entre* "`\(...\)`", que en este caso es "`[0-9]`". Así pues, "`\1`" es capaz de recuperar parte del texto que ha encajado en la expresión para utilizarlo al reemplazar. Vuelve a leer la sesión anterior y notarás el efecto de "`\1`".

Observa que esta facilidad te permite reordenar el texto que hay en la línea. Si contamos de 10 a 15 de 1 en 1 usando `seq`

```
unix$ seq 10 1 15
10
11
12
13
14
15
```

podemos ver el efecto de estas expresiones en otro ejemplo:

```
unix$ seq 10 1 15 | sed 's/\([0-9]\)\([0-9]\)/\2\1 and not \1\2/'
01 and not 10
11 and not 11
21 and not 12
31 and not 13
41 and not 14
51 and not 15
unix$
```

Ahora tenemos dos subexpresiones entre "`\(...\)`", y reemplazamos toda la expresión por texto consistente en los dos caracteres con el orden cambiado, luego "` and not `" y luego los dos caracteres en el orden original. Ni que decir tiene que puedes utilizar cualquier subexpresión y no una que opere sobre un sólo carácter.

Para borrar una expresión, podemos reemplazarla por *nada*. Por ejemplo, este comando elimina un nivel de tabulación de su entrada:

```
sed 's/^ //'
```

Debes tener cuidado con las sustituciones en `sed`. Se aplican al primer trozo de la línea que encaja con la expresión, pero no se repiten más adelante en la misma línea. Esto es, puedes reemplazar texto una sóla vez. Si deseas reemplazar todas las veces que sea posible puedes añadir un flag "`g`" al comando. Por ejemplo, dados los ficheros `chXX.w`", este comando nos imprime los números para esos ficheros:

```
unix$ ls
ch1.w    ch11.w    ch2.w    ch4.w    ch6.w    ch8.w
ch10.w    ch12.w    ch3.w    ch5.w    ch7.w    ch9.w
unix$ ls | sed 's/[.a-z]//g'
    1
    10
    11
    12
    2
    ...
    9
unix$
```

Compara con

```
unix$ ls | sed 's/[.a-z]//'
h1.w
h10.w
...
unix$
```