Sed – Introducción a SED – Parte I

Junio 2014

SED - The Stream EDitor - Part I

Este articulo es una introducción a la práctica y uso del editor de flujo “**SED**”, el articulo intenta cubrir ciertas funciones poco conocidas, por no decir, casi desconocidas, que hacen de SED una herramienta indispensable en la caja de herramientas de cualquier usuario de Linux que desea dominar el manejo del procesamiento de ficheros mediante una consola y un shell.

Índice Parte I

Presentación

Sed significa "**S**tream **ED**itor" en español “editor de flujo” o “editor de flujo orientado a líneas”. Por su modo de funcionamiento y concepción, Sed es un editor no interactivo. Al igual que el

editor “**ed**” (del cual proviene y que lo encontramos aun en las distribuciones actuales), Sed opera sobre una sola línea a la vez, a diferencia de otros editores como vi, emacs, Nedit, Xedit, etc., que operan sobre una página completa de texto que aparece en pantalla. El editor “**ed**” estaba dotado de un comando que trabajaba sobre el flujo de entrada estándar en vez que sobre un archivo, y era capaz de mostrar las líneas que contenían una expresión regular. Este comando cuya sintaxis es "**g**/**re**/**p**" (global/regular expression/print) dio nacimiento a la utilidad “**grep**”. Luego aparecería una nueva implementación de una versión de **ed**, que trabajaba únicamente sobre el flujo de entrada estándar y recibía las instrucciones de un archivo de script. Esta versión fue bautizada como Stream EDitor, más conocida con el nombre de “**Sed**”. El editor de flujo Sed lee las líneas de uno o varios ficheros desde la entrada estándar, lee los comandos desde la entrada estándar, o desde un archivo texto (*script*), y los agrupa bajo forma de expresiones (*comandos de edición*), luego aplica estos comandos y escribe el resultado en la salida estándar. Podríamos resumir el mecanismo de funcionamiento de Sed de esta manera:

- Lectura de una línea desde el flujo de entrada (*las líneas están delimitada por un carácter de salto de línea*)
- La línea es procesada en función de los comandos leídos
- Muestra (o no) del resultado en la salida estándar (*pantalla*)
- Continúa con la línea siguiente.

Los comandos aceptan números de líneas, rangos, o expresiones regulares (*RE o regex*) para seleccionar la o las líneas sobre las que deben operar

Introducción

Sed recibe las instrucciones o comandos desde la línea de comandos o desde un fichero (*script*) y aplica cada instrucción, en el orden en que aparece, a cada línea en la entrada estándar. Una vez que todas las instrucción han sido aplicadas a la 1ra línea, la línea es mostrada (o no, dependiendo de lo que se indique) en la salida estándar (*la pantalla, o redirigida a un archivo*), luego Sed procede a la lectura y el procesamiento de la siguiente línea y así sucesivamente hasta el final del archivo de entrada (*a menos que encuentre una instrucción de salida*) Este mecanismo es llamado “**ciclo**”. Se entiende por ciclo a la aplicación de todos los comandos que componen el script a los datos presentes en el espacio de patrón. Por defecto un ciclo comprende:

- La copia de una línea de entrada al espacio de patrón (*la línea estando delimitada por el carácter fin de línea (\n)*)
- Normalmente el espacio de patrón está vacío, a menos que un comando “D” haya terminado el ciclo precedente (*en ese caso un nuevo ciclo comenzará con los datos sobrantes en el espacio de patrón*).
- Sed aplicará los comandos secuencialmente (*provenientes de un script o desde la línea de comandos*) a los datos presentes en el espacio de patrón, una vez llegado al final del script, enviará los datos procesados a la salida estándar, a menos que se indique lo contrario con la opción "-n", y borrará el espacio de patrón. Todos los datos enviados a la salida estándar o a un fichero, son seguidos por un carácter de fin de línea (\n).
- Copia de una nueva línea o salida si se ha llegado al final del fichero.

Veamos con la ayuda de un organigrama el funcionamiento de Sed mediante un sencillo script que borra las líneas vacías de un fichero y las líneas que contienen únicamente un carácter

almohadilla (#) al inicio de la línea. Para ello, aquí tenemos un fichero conteniendo algunas líneas vacías, algunas almohadillas solas, entre ellas una espaciada a la derecha, y dos líneas con varias almohadillas. El fichero:

```
#
#
 #
##
# Este es un comentario

# Este es otro comentario
#

#
#
###
# Y otro más



#

#
# Y un último comentario
#
```

El script es relativamente simple. Aquí lo tenemos en una sola línea:

```
sed -e '/^$/d;/^#$/d'
```

Y en un script:

```
#! /bin/sed -f

/^$/d   # borrar las líneas vacías
/^#$/d # borrar las líneas conteniendo solo un carácter almohadilla “#”
        #+ encontrándose al inicio de la línea y nada detrás
```

El organigrama: Graforganigr

Sintaxis

Sintaxis general

```
sed [-opciones] [comando] [<fichero(s)>]
sed [-n [-e comando] [-f script] [-i[.extension]] [l [corte]] rsu] [<comando>] [<fichero(s)>]
```

Sintaxis de un comando

El direccionamiento de una o varias líneas es opcional en todos los comandos

```
[dirección[,dirección]][!]comando[argumentos]
```

Poniendo entre llaves un conjunto de comandos, pueden ser aplicados a una línea o rango de líneas.

```
[dirección[,dirección]]{
comando1
comando2
comando3
}
```

Estos comandos pueden estar puestos en una sola línea, pero deben ser separados por un punto y coma (;).

```
[dirección[,dirección]]{comando1; comando2; comando3}
```

Direccionamiento

Sed puede direccionar directamente una línea (o un rango) por su número de línea o por la coincidencia con una expresión regular haciendo de patrón. Un signo de exclamación (!) después de un numero de línea, un patrón o una expresión regular evita que la línea (o rango) sea procesada. El direccionamiento se puede efectuar de la siguientes forma: **num**

- El número de la línea
    - `sed -n 3p fich.txt`

inicio~salto

- Todas las n líneas (salto) comenzando desde el inicio.
    - `sed -n 1~2p fich.txt`

$

- La última línea del último fichero leído en la entrada, o de cada fichero si las opciones “-i” o “-s” han sido especificadas.
    - `sed -n '$ p' fich.txt`
      `*sed -ns '$ p' fich*`

/exp/

- Todas las líneas que coinciden con la expresión regular exp
    - `sed -n '/est/p' fich2.txt`

\#exp#

- Todas las líneas que coinciden con la expresión regular exp precisando utilizar como delimitador el carácter "#" (almohadilla) en lugar del delimitador predeterminado.
    - `sed -n '\#est#p' fich2.txt`

num1,num2

- Todas las líneas comprendidas entre num1 y num2. Si num2 es inferior a num1, solo num1 es mostrado.
    - `sed -n '3,6 p' fich.txt`

/exp1/,/exp2/

- Todas las líneas comprendidas entre exp1 y exp2, comprendidas las líneas conteniendo exp1 y exp2. Si el intervalo conteniendo las 2 expresiones se repite varias veces, Sed aplicará las instrucciones a cada intervalo sucesivamente. No obstante si exp2 no es encontrado, las instrucciones son aplicadas a cada línea comenzando por exp1 hasta el final del fichero.
    - `sed -n '/comentario1/,/comentario2/ p' fich2.txt`

num,/exp/ /exp/,num

- Todas las líneas comprendidas entre un número de línea y una expresión regular (o a la inversa). En "num,/exp/", no obstante si exp no es encontrado, las instrucciones son aplicadas a cada línea comenzando por num hasta el final del fichero. En "/exp/,num", si num es inferior al numero de línea correspondiente a exp, solo la línea que contiene exp es mostrada.
    - `sed -n '2,/comentario2/ p' fich2.txt`
    - `sed -n '/comentario1/,8 p' fich2.txt`

Las opciones (argumentos)

Sed acepta opciones (argumentos), pero no demasiadas. Las mas utilizadas son: "**-n**", "**-e**" e "**-i**".
-n, --quiet, --silent

- Solicitud implícita para no mostrar el estado de la memoria principal (*buffer*). En un script la notación se hará de esta manera "#n" (*un signo almohadilla seguido del carácter "n"*) y se deberá encontrar en la 1ra línea del script.

-e *script*, --**expresión**=*script*

- Permite encadenar varios comandos

-f *fichero-script*, ***--file**=*fichero-script*

- Lectura de comandos desde el fichero indicado

-i[*SUFIJO*], **--in-place**[=*SUFIJO*]

- Edita archivos en el lugar. También da la posibilidad de hacer una copia de respaldo añadiendo la extensión (-i.BAK)

--posix

- Desactiva todas las extensiones de GNU

-r, --regexp-extended

- Utiliza expresiones regulares extendidas (ERE)

-s, *--separate*

- Si varios ficheros son ingresados en la entrada, los procesa uno a uno en vez que como uno solo

-u, *--unbuffered*

- Carga cantidades mínimas de datos desde los ficheros de entrada y libera los almacenamientos temporales de salida con mayor frecuencia

--help

- Muestra esta ayuda y termina

--version

- Muestra información acerca de la versión del programa y termina.

Los comandos

En los capítulos que siguen veremos los comandos utilizados por sed. Mientras que el uso de algunos de ellos puede parecer sencillo, el uso y la implementación de otros dentro de scripts puede ser un poco más difícil debido a su sintaxis. Algunos comandos admiten un rango de direcciones mientras que otros solo admiten una e incluso ninguna en una minoría de comandos.

Los comandos básicos 1

En este capitulo veremos los comandos más conocidos de Sed cuyo uso es relativamente sencillo. **#** Comentario (*no acepta ninguna dirección*)

- El carácter **#** (almohadilla) inicia un comentario que se extiende hasta el final de la línea. Se puede encontrar en la misma línea de un comando.


Si los dos primeros caracteres de un script Sed son "**#n**", la opción "**-n**" (no-autoprint) es forzada. Por lo tanto si tu script debe empezar necesariamente con una línea de comentario comenzando con la letra “n” minúscula, utiliza una “N” mayúscula o inserta un espacio entre la almohadilla (#) y la "n". **q** quit Abandonar (*acepta una dirección*)

- Abandona sed sin ejecutar ningún otro comando ni evaluar otra entrada. La línea actual contenida en la memoria principal es mostrada a menos que la opción "-n" haya sido empleada.
    - `sed '3q' fich.txt`

d delete Borrar (*acepta un rango de direcciones*)

- Borra el espacio de patrón y pasa a la siguiente línea de entrada.
    - `sed '3d' fich.txt`

p print Mostrar (*acepta un rango de direcciones*)

- Muestra en pantalla el espacio de patrón actual. No borra el espacio de patrón ni modifica la ejecución del script. Este comando siempre es empleado conjuntamente con la opción "**-n**", sino la línea aparecería duplicada. (*Utilizar mejor la segunda forma que es mas adaptada ya que el script se termina en cuanto el patrón o la línea encontrada es mostrada en la salida estándar y no continua recorriendo el resto del fichero*)
    - `sed -n '3p' fich.txt`
    - `sed -n '3{p;q}' fich.txt`

n next-line Siguiente línea (*acepta un rango de direcciones*)

- Reemplaza el espacio de patrón actual con la siguiente línea sin comenzar un nuevo ciclo. La línea reemplazada es enviada a la salida estándar.
    - `echo -e "AAA\nBBB\nCCC\nDDD" | sed -n '/BBB/ {n;p;q}'`

{ ... } Agrupación de comandos (*acepta un rango de direcciones*)

- El empleo de llaves permite agrupar ciertos comandos que serán ejecutados sobre una dirección o un rango de direcciones. No es necesario protegerlos con un backslash como en el caso del empleo de expresiones regulares indicando un número de repeticiones.
- `echo -e "AAA\nBBB\nCCC\nDDD" | sed -n '/BBB/ {n;s/C/Z/2p}'`

s *substitución* Comando de substitución (*acepta un rango de direcciones*)

- El comando de substitución “**s**” es sin ninguna duda el comando mas utilizado del filtro Sed. Su sintaxis es muy simple:
    - `'s/patrón/reemplazo/flag(x)'`
- Su funcionamiento también es muy simple: si encuentra una cadena que coincide con el

patrón o la expresión regular, la cadena es substituida por la cadena de reemplazo, teniendo en cuenta los posibles flags.
- En los mecanismos de substitución hay que distinguir dos partes: **LHS** (*Left Hand Side = lado izquierdo*) que corresponde a la cadena buscada y **RHS** (*Right Hand Side = lado derecho*) correspondiente a la cadena de reemplazo.
- Mientras que la parte izquierda acepta la sintaxis de las **BRE** (*Basic Regular Expression, expresiones regulares básicas*), la parte de la derecha (*reemplazo*) acepta únicamente tres valores que pueden ser interpolados:
  - el carácter & (*amperstand*)
  - las referencias hacia atrás \1 (*de 1 al 9*)
  - las opciones \U,\u,\L,\l y \E


Para interpretar literalmente un amperstand (&) o un anti-slash (\) es necesario hacerlas preceder de un anti-slash: \& o \\

Flags

Los flags o atributos El comando de substitución (s) puede ser seguid de varios flags o atributos. Ciertas combinaciones no pueden ser hechas como el atributo “g” (*global*) y una enésima ocurrencia (*N*) lo que seria una total incoherencia. Siguiendo la misma lógica, el atributo “w” debe ser el último de la lista. **g** *global*

- Efectúa el reemplazo de todas las ocurrencias encontradas correspondientes al patrón o a expresión regular.

```
echo "AAAAA" | sed 's/A/B/'
BAAAA
echo "AAAAA" | sed 's/A/B/g'
BBBBB
```

N *enésima ocurrencia*

- Reemplaza únicamente la enésima ocurrencia encontrada correspondiente al patrón o expresión regular.

```
echo "AAAAA" | sed 's/A/B/3'
AABAA
```

p print (visualización)

- Si se ha producido una substitución, entonces muestra el espacio de patrón actual. Necesita la presencia de la opción "-n".

```
$ var="línea1\nlínea2\nlínea3\nlínea4\nlínea5"
 $ echo -e "$var"
 línea1
 línea2
 línea3
 línea4
 línea5

 $ echo -e "$var" | sed '3 s/e3/e n° 3/'
 línea1
 línea2
 línea n° 3
 línea4
 línea5

 $ echo -e "$var" | sed -n '3 s/e3/e n° 3/'

 $ echo -e "$var" | sed '3 s/e3/e n° 3/p'
 línea1
 línea2
 línea n° 3
 línea n° 3
 línea4
 línea5

 $ echo -e "$var" | sed -n '3 s/e3/e n° 3/p'
 línea n° 3
```

w *fichero - Write (escritura en un fichero)*

- Si se ha producido una substitución, entonces escribe el espacio patrón en el fichero especificado. Solo es aceptado un espacio entre el atributo “w” y el nombre del fichero.

```
$ var="línea1\nlínea2\nlínea3\nlínea4\nlínea5"
$ echo -e "$var" | sed -n '3 s/e3/e n° 3/pw fich.out'
```

e *evaluate (evaluación)*

- Permite ejecutar un comando mediante el shell y substituir el resultado con el del patrón, únicamente si se ha encontrado una ocurrencia.

Ejemplo 1:

```
$ echo $var
línea1\nlínea2\nlínea3\nlínea4\nlínea5\nlínea6\nlínea7\nlínea8\nlínea9
$ echo $A
Bonjour
$ echo -e "$var" | sed 's/.*5/echo '$A'/e'
línea1
línea2
línea3
línea4
Bonjour
línea6
línea7
línea8
línea9
$
```

Ejemplo 2:

```
$ cat plop
0x00000000 0         root     777
0x00000000 65537     user1  600
0x00000000 98306     user1  600

$ echo -e "$var" | sed 's/.*5/cat plop/e'
línea1
línea2
línea3
línea4
0x00000000 0         root     777
0x00000000 65537     user1  600
0x00000000 98306     user1  600
línea6
línea7
línea8
línea9
$
```

I *Ignorar diferencia entre mayúsculas y minúsculas*

- Permite ignorar la diferencia entre mayúsculas y minúsculas en la búsqueda de una coincidencia con el patrón.

```
$ echo "BonJouR" | sed 's/bONjOUr/Salut/'
BonJouR

$ echo "BonJouR" | sed 's/bONjOUr/Salut/I'
Salut
```

M

- El modificador M para la búsqueda de coincidencias con expresiones regulares es una nueva extensión de GNU Sed que permite que coincidan el carácter ^ y el carácter $ con una cadena vacía después de una nueva línea y una cadena vacía antes de una nueva línea respectivamente. Ya existían los caracteres especiales \` y ' (*en modo basico o extendido de expresiones regulares*) que hacían coincidir el inicio y el fin del buffer. M siendo multilínea.

Para explicarlo más claramente, el espacio de patrón contiene una línea leída desde la entrada y pueden ser agregadas otras líneas utilizando comandos como N,G, x, etc. Todas estas líneas en el espacio de patrón son separadas por el carácter de fin de línea "\n" pero son vistas por sed como una sola línea cuyo inicio empieza antes de la 1ra línea y termina al final de la ultima línea. Con el flag “M” cada carácter representando el inicio (^) y el final ($) de línea retoma su sentido inicial y hace coincidir el inicio y el final de la línea con cada línea que se encuentra en el espacio de patrón. A continuación un ejemplo que muestra el empleo del flag “M”: 1er caso:

```
$ echo -e "foo\nbar" | sed 'N;s/^.*$//'
 $
```

En este caso, ^ y $ apuntan al inicio y al final del buffer que después de la aplicación del comando “N” contiene "foo\nbar$", y la expresión regular coincide con todo lo que se encuentra entre los dos caracteres que indican el inicio (^) y el final ($) sin tener en cuenta el carácter que representa el final de la línea "\n". 2do caso:

```
$ echo -e "foo\nbar" | sed 'N;s/^.*$//M'
 bar
 $
```

En este caso, ^y $ apuntan al inicio y el final de la primera línea en el buffer, que como en el caso anterior después de la aplicación del comando “N” contiene "foo\nbar$", pero con la diferencia que la expresión regular coincide únicamente con los caracteres que se encuentran antes del carácter fin de línea "\n". 3er caso:

```
$ echo -e "foo\nbar\nfoobar\nbarfoo" | sed -e ':boucle; N; $! b boucle; s/^.*$//M3'
 foo
 bar

 barfoo
 $
```

En este 3er caso, el buffer después de la ejecución del comando “N” (*dentro de un bucle que tiene por efecto cargar la totalidad de las líneas en el buffer*), se parece a "foo\nbar\nfoobar\nbarfoo$" y la substitución se aplica únicamente a la 3era línea materializada por el carácter "\n". A continuación otros 2 ejemplos: El 1ro:

```
$  echo -e "foo\nfoo\nfoo\nbar\nfoo" |  sed 'N;/bar$/s/^/>/Mg;P;D'
 foo
 foo
 >foo
 >bar
 foo
 $
```

Aquí son cargadas 2 líneas al espacio de patrón, si el final del buffer no termina en “bar”, entonces la 1ra línea del buffer es mostrada (P), luego borrada (D), y se retoma la ejecución del script con la carga de la línea siguiente donde se comprueba nuevamente la coincidencia con la expresión regular. Si se encuentra una ocurrencia, se agrega un (>) al inicio de la línea, luego la 1ra línea del buffer es mostrada (P), luego borrada (D) y se retoma la ejecución del script… El 2do:

```
$ echo -e "foo\nfoo" |  sed 'N;s/^/>/;s/\n/\n>/g'
 >foo
 >foo

 $ echo -e "foo\nfoo" |  sed 'N;s/^/>/Mg'
 >foo
 >foo
 $
```

En este ejemplo se muestra la utilidad del flag “M” utilizando únicamente una expresión para agregar un (>) al inicio de cada línea contenida en el espacio de patrón después de llamar al comando “N”.

Los comandos básicos 2

y Transposición de caracteres (*acepta un rango de direcciones*)

- El comando “y” permite convertir cualquier carácter enumerado en la cadena carácter-origen por su homologo, en su lugar, que se encuentra en la cadena carácter-destino.

El empleo más común de este comando es sin dudas el reemplazo de caracteres acentuados. Veamos un ejemplo:

```
sed '
 y/àâéèêëîïôöùûü/aaeeeeiiooouuu/
 y/ÀÂÉÈÊËÎÏÔÖÙÛÜ/AAEEEEIIOOUUU/
 ' fichero.txt
```

a\
text Agregar (*acepta una dirección*)

- Agrega el texto “text” después de la línea que coincide con el número de línea, patrón o expresión regular, y antes de la lectura de la línea siguiente. “text” corresponde a una sola línea de texto, que sin embargo puede contener saltos de línea precedidos de "\" (backslash).

```
sed '/Línea n° 5/ a\
 Bonjour
' fich.txt
```

i
text Inserción (*acepta una dirección*)

- Inserta el texto “text” antes de la línea que coincide con el número de línea, patrón o expresión regular. “text” corresponde a una sola línea de texto, que sin embargo puede contener saltos de líneas precedidos de "\" (backslash).

```
sed '/Línea n° 4/ i\
 Bonjour
' fich.txt
```

c
text Intercambio (*acepta un rango de direcciones*)

- Cambia la línea que coincide con el número de línea, patrón o expresión regular con “text”. “text” corresponde a una sola línea de texto, que sin embargo puede contener saltos de líneas precedidos de "\" (backslash).

```
sed '/Línea n° 2/,/Línea n° 6/ c\
 Anulado\
 A causa\
 de trabajos
' fich.txt
```

r fichero *read* Lectura (*acepta una dirección*)

- Lee el contenido de “fichero” en el espacio de patrón luego de la dirección especificada. Únicamente debe haber un espacio entre el comando y el nombre del fichero. Lo que sigue después del espacio, hasta el final de la línea, es considerado el nombre del fichero. De aquí que cualquier espacio (incluido la tabulación) será considerado como parte del nombre. Si el fichero no existe, ningún mensaje de advertencia aparecerá en la salida estándar ni en otro lugar. Si el fichero no se encuentra en el mismo directorio en el que es ejecutado el comando, se debe especificar la ruta completa del fichero.

Por ejemplo lo podemos utilizar para agregar una firma al final de una serie de ficheros. Para ilustrar esto, vamos a agregar el contenido del fichero “firma.txt” al final de todos los ficheros correspondientes al patrón “fich*.txt) (observa bien el siguiente ejemplo y observa el empleo del switch “-s”):

```
sed -s '$ r firma.txt' fich*.txt
```

w fichero *write* Escribir (*acepta una sola dirección*)

- Escribe la línea que esta siendo procesando en el fichero indicado luego del comando “w”. Al igual que el comando “r” (lectura), únicamente debe haber un espacio entre el comando y el nombre del fichero. Todo lo que esta después de este espacio, hasta el final de la línea, es considerado como el nombre del fichero. De aquí que cualquier espacio (incluida

una tabulación) será considerado como parte del nombre. Si un fichero con el mismo nombre ya existe, será aplastado sin ninguna advertencia ni confirmación en cada invocación del script. En cambio, si varias instrucciones del comando “w” deben ser escritas en un mismo fichero desde un script, cada escritura es agregada al final del fichero.

Si el fichero no existe será creado, incluso si el proceso es nulo en la salida (ninguna escritura enviada). A continuación un pequeño escenario para poner este comando en aplicación. Desde un fichero “direcciones.txt” agrupando nombres de servicios postales asociados a un código postal y su ciudad de referencia, extraer el nombre del servicio postal y la ciudad asociada y enviarla a un nuevo fichero que lleva el nombre del departamento. Este script llamado "foo.sed" será invocado de la manera siguiente:

```
sed -f foo.sed < direccións.txt
```

Contenido del fichero "foo.sed":

```
#n

 /\b31/{
 s/[0-9][0-9]*//
 w Haute-Garonne
 }
 /\b34/{
 s/[0-9][0-9]*//
 w Hérault
 }
 /\b66/{
 s/[0-9][0-9]*//
 w Pyrénées-Orientales
 }
```

=

- Muestra el número de la línea actual

```
sed -n '/patrón/=' fichero
```

l [N] --line-length=N Corte (*acepta un rango de direcciones*)

- Muestra caracteres no imprimibles – *N* permite especificar la longitud de corte de línea deseada.

```
sed -n l fichero # Muestra caracteres no imprimibles
 sed -n 'l 8' fichero # lo mismo pero con un retorno a la línea cada 8 caracteres
```

Los comandos avanzados

Además de los comandos que acabamos de ver, Sed posee otros comandos, poco utilizados y

para algunos no muy fácil de utilizar, pero que permiten realizar ciertas tareas. Los comandos precedentes utilizan principalmente el siguiente mecanismo: Lectura de una línea del fichero de entrada en el espacio de patrón a la cual se le aplica cada uno de los comandos del script. Cuando se alcanza el final del script, la línea es enviada a la salida estándar, el espacio patrón es borrado, una nueva línea es leída desde la entrada y el control es pasado nuevamente al inicio del script. Con los comandos que siguen, veremos cómo podemos intervenir en el desarrollo del script: modificar el flujo de entrada bajo ciertas condiciones, almacenar partes de líneas, probar condiciones, etc. Estos comandos pueden ser clasificados en 3 grupos:

- Los comandos multi-líneas (N,D,P)
- Los comandos que utilizan la memoria secundaria (h,H,g,G,x)
- Los comandos de test que utilizan etiquetas (:,b,t,T)

Los comandos multi-líneas

N Next Siguiente (*acepta un rango de direcciones*)

- El comando “N” posiciona el carácter “nueva línea” (\n) al final del contenido del espacio patrón y agrega la línea siguiente del flujo de entrada en el espacio patrón. Si el final del fichero de entrada es alcanzado, sed termina la ejecución sin proceder a la ejecución de un nuevo comando. El carácter “nueva línea” incorporado en el espacio patrón puede ser asociado a la secuencia de escape “\n”. en un espacio patrón multi-líneas, los meta-caracteres "^" y "$" concuerdan con el inicio y final del espacio patrón y no los inicios y finales de las líneas precedentes o siguientes al carácter nueva línea incorporada.

El ejemplo que sigue busca una línea que contiene el patrón “C”. Si este es encontrado, agrega la siguiente línea al espacio patrón y substituye el carácter final de línea “\n” por un guión rodeado de un espacio en ambos lados:

```
echo -e "A\nB\nC\nD\nE" | sed  '/C/{N;s/\n/ - /}'
```

D Delete Borrar (*acepta un rango de direcciones*)

- El comando “D” borra el contenido del espacio patron hasta el 1er carácter delimitando una nueva línea (\n). si aun quedan datos en el espacio patron, un nuevo ciclo es iniciado con este contenido (*sin leer una nueva línea desde la entrada*), si no un nuevo ciclo es iniciado con la línea siguiente.

Para mostrar el uso del comando “D”, tomaré un ejemplo dado en el excelente libro publicado por O'Reilly (sed & awk, Second Edition) y que resume muy bien el mecanismo de este comando similar al comando “d”. El fichero de referencia (comando_D.txt):

```
Esta línea es seguida de una línea vacía

Esta línea es seguida de 2 líneas vacías


Esta línea es seguida de 3 líneas vacías



Esta línea es seguida de 4 líneas vacías



Fin del archivo
```

Por razones de presentación inherentes a esta sección, en el fichero solo hay 3 líneas vacías debajo de la línea *"Esta línea es seguida de 4 líneas vacías"*. Deberás agregar una 4ta línea si deseas comprobar este ejemplo. Ya que el objetivo es agrupar las líneas vacías consecutivas en una sola. El comando "**d**" parece ser el apropiado para esta tarea. Veamos un 1er script utilizando este comando:

```
sed '
/^$/{
N
/^\n$/d
}' comando_D.txt
```

Para ello utilizaremos un patrón que nos permita coincidir con una línea vacía "/^$/". En cuanto una línea vacía sea encontrada pediremos que sea cargada la línea siguiente al espacio de patrón con el comando "N". Una vez esta línea cargada, comprobaremos que el patrón presente en el espacio de patrón coincida con el patrón "/^\n$/", y si es así, lo borramos (comando "d"). Pero esta sintaxis funciona únicamente cuando el número de líneas es impar. Esto se explica por el hecho de que el comando "d" borra la totalidad del contenido del espacio de patrón. Efectivamente, cuando una línea vacía es encontrada la línea siguiente es cargada (N), si esta línea es vacía, el espacio patrón es borrado (d) y comienza un nuevo ciclo con una nueva línea. Así, si esta nueva línea (3ra) es vacía, y la siguiente no, entonces el comando de borrado (d) no se aplica y se muestra la línea vacía. En cambio si reemplazamos el comando "**d**" por "**D**":

```
sed '
/^$/{
N
/^\n$/D
}' comando_D.txt
```

Obtenemos el resultado esperado. En efecto, el comando "D" borra únicamente la parte del espacio patrón comprendido antes del 1er carácter "/n" (salto de línea), de aquí si 2 líneas vacías se encuentran en el espacio patrón, solo la 1ra línea es borrada y el script reinicia con el contenido del espacio patrón (una línea vacía), entonces una nueva línea es cargada, si no es vacía, las 2 líneas contenidas en el espacio patrón son enviadas a la salida estándar, si no la

1ra parte es borrada y el escenario se repite… En otras palabras, si dos líneas vacías se encuentran en el espacio patrón, solo la 1ra línea es borrada, si es una línea vacía seguida de texto, son enviadas a la salida estándar. **P** Print Visualización (*acepta un rango de direcciones*)

- Al igual que su similar en minúscula que muestra el contenido del espacio patrón, el comando “P” muestra el contenido del espacio patrón hasta el 1er carácter delimitando una nueva línea (\n). Cuando el ultimo comando del script es alcanzado, el contenido del espacio patrón es mostrado automáticamente en la salida estándar (*a menos que la opcion “-n” o "#n" haya sido empleada*).

Los buffers

El editor de flujo Sed dispone de dos buffers que permiten almacenar la (las) línea(s) que se están procesando. Estas memorias son generalmente denominadas *"pattern space"* para la memoria principal, que la podemos traducir por “espacio patrón”, y "hold space" para la memoria secundaria. El espacio de patrón (pattern space) es el espacio de memoria en el que son mantenidos los datos (la o las líneas) seleccionados mientras sean procesados. El hold space es el espacio de memoria en el que los datos (la o las líneas) son almacenados temporalmente. Existen 5 comandos que permiten pasar de un espacio a otro, a continuación un breve resumen:

- **h** Copia el contenido del espacio patrón en el hold space
- **H** Agrega el contenido del espacio patrón en el hold space
- **g** Copia el contenido del hold space en el espacio patrón
- **G** Agrega el contenido del hold space en el espacio patrón
- **x** Intercambie el contenido de las 2 memorias

Salvo el comando “x”, los otros comandos funcionan en parejas y actuan para cada binomio a la manera de redirecciones (>, >>, <,<<) interpretes de comandos en el shell y en el “bash” o “ksh”. Su rol se podria traducir de este modo:

- **h** > Aplasta el contenido
- **H** >> Agrega contenido
- **g** < Aplasta el contenido
- **G** << Agrega contenido

A continuación una breve definición de cada comando que puede afectar el espacio patrón: **h** hold pattern space (*acepta un rango de direcciones*)

- El comando h copia el contenido del espacio patrón en la memoria secundaria, destruyendo el contenido existente.

H Hold pattern space (*acepta un rango de direcciones*)

- El comando H agrega el contenido de espacio patrón al contenido de la memoria secundaria. El antiguo contenido y el nuevo son separados por una nueva línea representada por el carácter "\n". Una nueva línea (\n) es agregada al espacio patrón, incluso si éste esté vacío.

g get contents Copia el contenido (*acepta un rango de direcciones*)

- El comando g copia el contenido de la memoria secundaria en el espacio patrón, destruyendo el contenido existente.

G Get contents Agrega contenido (*acepta un rango de direcciones*)

- El comando G agrega el contenido de la memoria secundaria al espacio patrón. El contenido que existia y el nuevo son separados por una nueva línea representada por el carácter "\n".

x eXchange Intercambio (*acepta un rango de direcciones*)

- El comando x intercambia el contenido de las dos memorias (principal y secundaria). La memoria secundaria inicia su ciclo con una línea vacía. Si aplicamos el comando “x” a la 1ra línea de un fichero, está línea es colocada en la memoria secundaria y será reemplazada por el contenido de esta memoria secundaria, es decir una línea vacía. También debemos saber, que según este principio, la última línea de un fichero es colocada en la memoria secundaria pero nunca es restituida en el espacio patrón y de aquí que no será nunca mostrada a menos que se haga una solicitud implícita.

Al final de este documento encontraras algunos ejemplos comentados acerca del uso de la memoria principal y secundaria de Sed, y para comenzar veremos un pequeño ejemplo (sacado de la obra de las ediciones O'Reilly), muy fácil de comprender, pero que nos advierte acerca de unos de los errores a veces incomprensibles que puede ocurrir. Uno de estos errores tiene que ver con la memoria secundaria. Cuando enviamos el contenido del espacio patrón, y luego procedemos a diversas operaciones, y estas operaciones restituyen el contenido de la memoria secundaria únicamente bajo ciertas condiciones, puede ocurrir que este contenido no sea nunca restituido en el espacio patrón y de aquí, jamás será enviado a la salida estándar… Veamos la demostración. Vamos a mostrar la siguiente variable:

```
$ A="1\n2\n11\n22\n111\n222"
$ echo -e "$A"
1
2
11
22
111
222
```

Y pedir a Sed que invierta las líneas comenzando por “1” con las comenzando por “2”. Para ello comenzaremos por emparejar las líneas comenzando por “1”, copiar el contenido en la memoria secundaria con el comando “h”, luego vaciar el espacio patrón, utilizando el comando “d”. el control es enviado al inicio del script, donde una nueva línea es cargada (con un “2”), la primera comparación (/1/) fracasa, pero la segunda (/2/) es verdadera, por lo que el contenido de la memoria secundaria es agregado al espacio patrón, que contiene "2\n1$". Como hemos llegado al final del script, el contenido del espacio patrón es mostrado y reemplazado por la siguiente entrada (11) y el script vuelve a comenzar, y así sucesivamente… Este es el script:

```
echo -e "$A" | sed '
/1/{  # si el patrón está presente
h  # copiarlo en la memoria secundaria
d  # borrar el contenido de la memoria principal
}
/2/{  # si el patrón está presente
G  # agregar el contenido de la memoria secundaria
}'
```

La visualización final:

```
2
1
22
11
222
111
```

Como lo hemos visto, todo marcha bien. Pero que pasaría si pusiéramos un “333” en lugar de “22”. Esto es justamente lo que vamos a ver. En primer lugar, la visualización de la nueva variable:

```
$ A="1\n2\n11\n33\n111\n222"
$ echo -e "$A"
1
2
11
33
111
222
```

Y el filtrado hecho por "sed" :

```
$ echo -e "$A" | sed '/1/{h;d};/2/{G}'
2
1
33
222
111
```

Y bien como podemos ver, la visualización del “11” quedo atrás. Y por qué? Como lo hemos dicho al inicio de este ejemplo, simplemente porque el contenido de la memoria secundaria es restituida únicamente en el espacio patrón si y solamente si un patrón conteniendo un “2” es encontrado. En caso contrario, el script continúa su camino, dicho de otro modo, muestra la línea presente en el espacio patrón (33) y pasa el control al inicio del script que carga la línea siguiente (111), línea que satisface la condición del 1er motivo (/1/) y envía su contenido al espacio secundario, destruyendo los datos presentes (11). Por lo tanto, hay que tener cuidado durante la elaboración de ciertos scripts en restituir el contenido de la memoria secundaria.

Etiquetas

Las etiquetas (*label*) permiten saltar a una ubicación precisa dentro del script. Sed posee tres comandos especialmente para esto. Un comando incondicional “b” y dos comandos condicionales "t" et "T" de "[tT]est". La sintaxis para designar una etiqueta se limita a colocar al inicio de línea (por un script) dos puntos seguidos de una letra (*o cadena de letras a fin de formar una palabra, ésta ultima es recomendado para una mejor lectura del código*).

```
:etiqueta
```

Esta etiqueta será llamada en el script utilizando los comandos "b", "t" o "T". Simplemente anteponiendo su nombre con el comando deseado.

```
b etiqueta
t etiqueta
T etiqueta
```

Emplame condicional

b branch (*acepta rango de direcciones*)

- El comando b permite transferir incondicionalmente la ejecución del script a la posición indicada por la etiqueta pasada como argumento. Si no es pasado ningún argumento, el comando envía al final del script. El comando que se esta ejecutando es mostrado a menos que la opción “-n” estaba activa y el script retoma su ejecución con la próxima línea del flujo de entrada.

[Ejemplo de empalme incondicional]

Empalme condicional

t test (*acepta rango de direcciones*)

- El comando t permite transferir condicionalmente la ejecución del script a la posición indicada por la etiqueta pasada como argumento si un comando de substitución ha tenido éxito en la línea que se esta procesando o en el ultimo empalme condicional. Si ningún argumento es pasado, el comando envía al final del script.

[Ejemplo de empalme condicional] **T** test (*acepta rango de direcciones*)

- El comando T permite transferir condicionalmente la ejecución del script a la posición indicada por la etiqueta pasada como argumento si un comando de substitución ha fracasado en la línea que se está procesando o en el ultimo empalme condicional si ningún argumento es pasado, el comando envía al final del script.

[Ejemplo de empalme condicional 2]

SED - The Stream EDitor - Part II

Continuación => [SED - The Stream EDitor – Part] PD: El artículo original fue escrito por jipicy, contribuidor de CommentCaMarche