Sed – Introducción a SED – Parte II

Junio 2014

SED - The Stream EDitor - Part II

Este articulo es una introducción al uso del editor de flujo "**SED**". El articulo intenta cubrir ciertas funcionalidades poco conocidas, por no decir "casi desconocidas", que hacen de "**SED**" una herramienta indispensable en la caja de herramientas de todo usuario de Linux que desea dominar el procesamiento de ficheros vía una consola y un shell.

Índice parte II

Los delimitadores

Delimitadores de comandos

De manera predeterminada, Sed utiliza como delimitador para su mecanismo de substitución la barra oblicua "/" (slash):

```
sed 's/patrón/reemplazo/' fichero
```

Para la mayoría de casos, esta opción por defecto resulta suficiente, pero puede resultar un problema si el patrón o la cadena de reemplazo también contienen uno o más slash(es) como en el caso de la ruta de un archivo. Evidentemente puedes proteger los slashes anteponiéndoles un "\" (anti-slash o barra oblicua invertida), pero esto resulta una operación muy fastidiosa si el patrón (o la cadena de reemplazo) contiene varios, volviendo la lectura del código bastante difícil:

```
sed 's/\/home\/jp\/Docs\/CCM\/SED/\/mnt\/servidor\/docs/' fichero
```

Hasta puede ser imposible si el patrón (o la cadena de reemplazo) es una variable que debe ser interpretada:

```
var="/home/jp/Documentos/CCM/SED/"
sed 's/'$var'/\/mnt\/servidor\/docs/' fichero
```

o (expresión entre doble apóstrofes)

```
sed "s/$var/\/mnt\/servidor\/docs/" fichero
```

Felizmente, Sed permite reemplazar el delimitador por defecto con el carácter de nuestra elección (#,|,!,§,etc.), siempre y cuando no forme parte del patrón (o la cadena de reemplazo):

```
sed 's#/home/jp/Docs/CCM/SED#/mnt/servidor/docs#' fichero
```

Este carácter puede ser cualquier letra siempre y cuando no este contenida en el patrón (o la cadena de reemplazo):

```
echo "hola" | sed 'sZbZBZ'
```


Si utilizas el carácter "!" (signo de exclamación) como separador, debes encerrar la expresión con apostrofes simples para que el shell no interprete el carácter "!" (empleado normalmente para gestionar el historial de los comandos).

Delimitador de patrón

Como lo hemos visto, Sed utiliza un patrón, encerrado entre "/" (slash), para buscar coincidencias en las líneas de un fichero. El slash utilizado por defecto, puede ser cambiado por cualquier otro carácter, simplemente anteponiendo a la 1ra ocurrencia de este carácter un "\" (backslash). Tomemos el ejemplo de una ruta como criterio de búsqueda. Bajo esta forma, vemos que la lectura del código no es fácil:

```
sed -n '/\/home\/jp\/Docs\/CCM\/SED/p' fichero
```

Sin embargo, si utilizamos el carácter almohadilla "#" como delimitador, tendremos:

```
sed -n '\#/home/jp/Docs/CCM/SED#p' fichero
```

El metacaracter &

A menudo, el mecanismo de substitución se limita a buscar un patrón a fin de substituirlo por si mismo y agregándole algo, por ejemplo, buscar la cadena "Sed the Stream EDitor" en un fichero y agregarle "Sed the Stream EDitor (Editor de flujo)". Podríamos escribir:

```
sed 's/Sed the Stream EDitor/Sed the Stream EDitor (Editor de flujo)/g' fichero
```

El metacaracter "&" (Ampersand) nos permite reemplazar todas las cadenas de caracteres que coinciden con el patrón (o la expresión regular) ingresado como 1er argumento. Aquí la noción de 1er argumento no es muy explicita, pero se vera su importancia con las expresiones regulares y en el capitulo de las “sub-expresiones”. Por lo tanto nuestro comando se escribirá de esta manera:

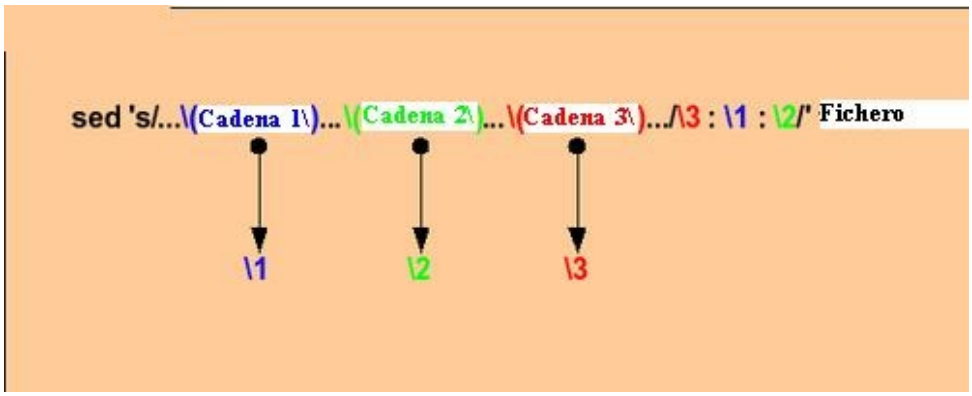
```
sed 's/Sed the Stream EDitor/& (Editor de flujo)/g' fichero
```

Supongamos que tenemos que buscar todas las cadenas numéricas (1 o varias cifras consecutivas) en un fichero y deseamos anteponer a cada una de estas cadenas "n° " (observa el espacio después de el "°"). El comando seria:

```
sed 's/[0-9][0-9]*/n° &/g' fichero
```

Necesariamente se debe utilizar la expresión "\&" para obtener un "&" literal en una cadena de reemplazo, si no se generarán errores.

Las subexpresiones y las referencias hacia atrás



Las subexpresiones

\(...\) Una subexpresión es una parte de una expresión regular, encerrada entre paréntesis, que posteriormente podemos utilizar en la substitución. Los paréntesis deben ser protegidos con backslashes, a menos que la opción “-r” haya sido empleada.

Las referencias hacia atrás

\1 \2 \5 Para referirnos a cada subexpresión en la cadena de reemplazo, utilizamos un número que corresponde a la posición que ocupa en la expresión regular. Este número debe estar protegido por un backslash. Únicamente podemos referirnos a 9 subexpresiones numeradas del \1 à \9. Estas referencias también son llamadas “referencias hacia atrás”. A continuación algunos ejemplo con nombre de ciudades francesas y su código postal correspondiente: Archivo de referencia:

```
$ cat plop
31000   Toulouse
34000   Montpellier
66000   Perpignan
```

En este ejemplo, la subexpresión "\([0-9]*\)" coincidirá con cualquier cadena numérica y es referenciada por la referencia hacia atrás "\1":

```
$ sed 's/\([0-9]*\).*/\1/' plop
31000
34000
66000
```

Esta vez, buscamos las coincidencias de 2 subexpresiones, una numérica y la otra conteniendo el resto de la línea, pero sin incluir el carácter de tabulación que separa el código postal del nombre de la ciudad. Luego utilizamos la referencia hacia atrás "\1" (código postal) y "\2" (nombre de la ciudad) para mostrar el resultado bajo la forma: nombre de la ciudad > código postal

```
$ sed 's/\([0-9]*\)\t\(.*\)/\2 > \1/' plop
Toulouse > 31000
Montpellier > 34000
Perpignan > 66000
```

En este 3er y último ejemplo, buscamos las coincidencias de una subexpresión con cada parte de la línea, es decir el código postal (\1), la tabulación (\2) y el nombre de la ciudad (\3)

```
$ sed 's/\([0-9]*\)\(\t\)\(.*\)/\3\2\1/' plop
Toulouse        31000
Montpellier     34000
Perpignan       66000
```

Una referencia hacia atrás puede llamar a una subexpresián las veces que se desee en la cadena de reemplazo. Retomando el último ejemplo, veamos la demostración con la repetición de la tabulación en diversos lugares:

```
$ sed 's/\([0-9]*\)\(\t\)\(.*\)/\2\3\2\2\1/' plop
        Toulouse                31000
        Montpellier             34000
        Perpignan               66000
```

Expresión regular precedente

Cuando encuentra una coincidencia entre una cadena y una expresión regular, Sed pone en su buffer

dicha cadena por lo que es posible referirse a esta cadena en la 1ra parte del comando de substitución "s" (LHS) sin mencionarla literalmente. En otras palabras, un comando del tipo 's//cadena_de_reemplazo/' substituirá la ultima expresión regular encontrada por Sed con la cadena_de_reemplazo. Para ilustrar esto, retomemos nuestro fichero con las ciudades y los códigos postales. Vamos a buscar las líneas que contengan el patrón “Montpellier”, las otras serán borradas (d) y substituiremos "Montpellier" por "Béziers":

```
$ cat plop
31000   Toulouse
34000   Montpellier
64000   Perpignan

$ sed '/Montpellier/!d; s//Béziers/' plop
34000   Béziers
```

Para tratar lo relacionado a las subexpresiones, referencias hacia atrás y expresión regular precedente, el ejemplo precedente muestra otra faceta de las posibilidades que ofrece sed. Si el patrón buscado es encerrado entre paréntesis, entonces se convierte en una subexpresión que luego puede ser invocada en la parte derecha del comando “s”:

```
sed '/\(Montpellier\)/!d;s//Béziers-\1/' plop
34000   Béziers-Montpellier

$ sed '/\(Mon\)tpellier/!d;s//Béziers-\1blanc/' plop
34000   Béziers-Monblanc
```

La negación

A veces puede ser útil excluir una línea que coincide con un patrón (*o un rango de lineas*) para que no sea procesada. Para ello, Sed utiliza el carácter "**!**" (signo de exclamación) que como en la mayoría de herramientas derivadas de Unix expresa la negación, exactamente como lo hace el comando "grep -v". Para ello, tan solo hay que poner a continuación del patrón (*o el numero de línea o rango de lineas*) el carácter "!":

```
sed -n '3 !p' fich.txt
sed -n '3,8 !p' fich.txt
sed -n '/indice/! p' fich3.txt
```

A menudo encontramos en los scripts Sed, la expresión "$!" que significa “mientras no se alcance la ultima línea” y permite efectuar uno o varios comandos mientras esta condición sea verdadera. Podemos comparar diferentes maneras de escribir la sintaxis de un comando, pero obteniendo un mismo resultado.

```
echo -e 'a\nb\n\nc\nd\ne\n\nf\ng' | sed '/./! d'
echo -e 'a\nb\n\nc\nd\ne\n\nf\ng' | sed '/^$/ d'
echo -e 'a\nb\n\nc\nd\ne\n\nf\ng' | sed -n '/^$/! p'
echo -e 'a\nb\n\nc\nd\ne\n\nf\ng' | sed -n '/./ p'
```

La agrupación de comandos

/**dirección**/**{...}** (*acepta un rango de direcciones*)

- Las llaves permiten agrupar comandos que serán ejecutados en una dirección o en un rango de direcciones. Dentro de la agrupación de estos comandos podemos encontrar otras direcciones así como otros comandos agrupados también entre llaves.

Su principal función, indicar una línea (o rango de líneas) y aplicar sucesivamente uno o varios comandos. El procesamiento es realizado sobre el contenido del buffer y no sobre la línea original, por lo que las modificaciones realizadas pueden condicionar los criterios de selección posteriores.

```
sed '
/a/{    # solo la línea que contiene "a"
s/a/c/g  # reemplazar todas las "a" por "c"
/c/{    # solo la línea que contiene "c" de la línea que coincide
s/c/A/g  # reemplazar todas las "c" por "A"
}
}
' < <(echo -e "aaa\nbbb\nccc\nddd")


AAA
bbb
ccc
ddd


$ echo -e "aaa\nbbb\nccc\nddd" | sed '/a/{s/a/c/g;/c/{s/c/A/g}}'

AAA
bbb
ccc
ddd
```

En un grupo de comandos (en un script), cada comando debe iniciar sobre su propia línea, las llaves y el conjunto de comandos deben estar en líneas separadas. La llave que abre debe encontrarse al final de la línea, mientras que la llave que cierra debe necesariamente encontrarse sola en una línea. Atención, no debe haber ningún espacio después de las llaves.

El reemplazo de variables

Puede darse el caso que dentro de un script Sed necesitemos pasar una variable como argumento, ya sea como patrón o en una u otra de las partes (LHS o RHS) durante una substitución. Como lo mencione anteriormente, habrá que prestar atención a los caracteres presentes en la variable a fin de adaptar los apóstrofes a utilizar. En efecto, por defecto Sed utiliza apóstrofes simples para encerrar las expresiones, pero este mecanismo en shell bloquea justamente la expansión de las variables; por lo que el empleo de apóstrofes simples impedirá una buena interpretación de la variable.

```
var=A; echo 'azerty' | sed 's/a/$var/'
$varzerty
```

Inicialmente, es posible reemplazar estos apóstrofes simples por apóstrofes dobles, lo que en la mayoría de casos será suficiente para permitir la interpretación correcta de la variable.

```
var=A; echo 'azerty' | sed "s/a/$var/"
Azerty
```

Sin embargo puede ser más conveniente utilizar esta sintaxis:

```
var=A; echo 'azerty' | sed 's/a/"$var"'/'
Azerty
```

En este caso, no es muy fácil comprender por qué es preferible utilizar una mezcla de apóstrofes simples y dobles. En realidad, la palabra “mezcla” no es el término mas apropiado, es más apropiado emplear la palabra “excluir”, ya que con esta sintaxis, excluimos la variable de la expresión, lo que permite que sea interpretada por el shell, pero protege dentro de la expresión la interpretación de posibles caracteres propios al shell. Para comprender mejor esto, veamos un ejemplo: La visualización por el shell de 6 letras del alfabeto, cada una en una línea, en la que la letra A es repetida dos veces. Después de haber asignado esta letra (A) a una variable ($var), vamos a pedir a Sed que no muestre las líneas conteniendo esta variable, empleando la sintaxis incluyendo la negación, es decir el signo de exclamación (!). Primero, escribamos la expresión con apóstrofes simples:

```
var=A; echo -e "A\nB\nC\nA\nD\nE" | sed -n '/$var/!p'
A
B
C
A
D
E
```

Observamos que todas las letras son mostradas y es normal ya que la variable no ha sido interpretada por el shell debido a los apóstrofes simples. Ahora, pongamos la expresión entre apóstrofes dobles:

```
var=A; echo -e "A\nB\nC\nA\nD\nE" | sed -n "/$var/!p"
-l: !p": event not found
```

El shell nos devuelve un mensaje de error ! Efectivamente, el "!" es un carácter reservado del shell que sirve para el historial de los comandos y no ha podido ser interpretado como tal. Separemos la variable de la expresión propia a Sed encerrando la expresión justo antes de la variable y volviéndola a abrir justo después utilizando apostrofes simples (observa que los apóstrofos dobles alrededor de la variable son opcionales, pero que es preferible dejarlos, ya que es una buena costumbre a considerar para prevenir posibles errores debidos a espacios dentro de la variable).

```
$ var=A; echo -e "A\nB\nC\nA\nD\nE" | sed -n '/"$var"'/!p'
B
C
D
E
```

Las expresiones regulares

El conocimiento de expresiones regulares (*regexp*) es un plus en la práctica de Sed (*y de muchos otros lenguajes*) Una expresión regular es un patrón (*pattern en inglés*) para buscar coincidencias con cadenas de caracteres. La potencia de las expresiones regulares radica en su capacidad de incluir alternativas y repeticiones en la elaboración del patrón. Estas particularidades son elaboradas utilizando caracteres especiales, que no son interpretados literalmente, sino de una manera especifica. A continuación una breve descripción de la sintaxis de las expresiones regulares utilizadas en Sed. **carácter** (*cualquier carácter*)

- Coincide con un carácter único

*****

- Coincide si el carácter o conjunto de caracteres que el precede está presente 0 o más veces. Puede ser un carácter ordinario, un carácter especial protegido por un \, un punto (.), un grupo de expresión regular o una subexpresión.

\+

- Idéntico a *, pero indica que la expresión anterior puede ocurrir una o varias veces.

\?

- Idéntico a *, pero indica que la expresión anterior puede ocurrir 0 ó 1 vez.

\{i\}

- Idéntico a *, pero coincide con exactamente i secuencia de la ocurrencia de la expresión precedente (i representa un entero).

\{i,j\}

- Coincide con una secuencia comprendida entre i y j incluyendolas.

\{i,\}

- Coincide con una secuencia mayor o igual a i

\(regexp\)

- Coincide con un conjunto de expresiones regulares, llamado también subexpresión y que puede ser direccionada por referencia hacia atrás (back referente)

. *un punto*

- Cualquier carácter, excepto una nueva línea.

^

- Coincide con una cadena nula al inicio de la línea, es decir lo que se encuentra después del acento circunflejo debe aparecer al inicio de línea o al inicio de una subexpresión.

$

- Idéntico a ^ pero la coincidencia es al final de la línea o subexpresión.

[lista]

- Coincide con cualquier carácter de la lista. Una lista puede estar constituida por una secuencia de letras como [a-z] lo que equivale a una coincidencia con cualquier carácter comprendido en a y z incluidos. Para incluir un] hazlo figurar en primera posición en la lista. Para incluir un – ponlo en la primera o ultima posición de la lista. Para incluir un ^ ponlo después del primer carácter de la lista.

Por lo general, los caracteres $, *, ., [, y \ no son considerados como caracteres especiales dentro de una lista. De este modo la expresión [*] corresponderá tanto al carácter \ como al carácter *, el carácter \ no es considerado como un carácter de escape para proteger el carácter *. **[^lista]**

- A la inversa, un ^ al inicio de lista, hará coincidir con cualquier carácter excepto los de la lista. Para incluir un] hazlo figurar en la primera posición en la lista justo después de el signo ^.

regexp1\|regexp2

- Coincidencia con regexp1 o regexp2. La búsqueda de coincidencias se hace con cada una de las alternativas, de izquierda a derecha y la primera que es encontrada es utilizada. Observa el carácter (\) para proteger el carácter (|).

\número

- Coincide con la enésima subexpresión \(...\) utilizada en la expresión regular. Las subexpresiones, llamadas también “referencias hacia a tras”, son implícitamente numeradas de izquierda a derecha contando el numero de ocurrencias de "\(", con un máximo de 9.

\n

- Coincide con el carácter nueva línea (LF)

\métacaracter

- Coincide con un metacaracter entre $, *, ., [, \, o ^, debiendo ser protegido para una interpretación literal.

Nota: en la búsqueda de una coincidencia, ésta es efectuada de izquierda a derecha, y si dos o más expresiones tienen el mismo carácter inicial, la coincidencia será hecha con la expresión más larga.

Los caracteres de escape

\a

- alarma (campana bip) (BEL, Ctrl-G, 0x07)

\b

- borrado hacia atrás (BS, Ctrl-H, 0x08)

\f

- fin de página (FF, Ctrl-L, 0x0C)

\n

- fin de línea (LF, Ctrl-J, 0x0A)

\r

- retorno de carro (CR, Ctrl-M, 0x0D)

\t

- tabulación horizontal (HT, Ctrl-I, 0x09)

\v

- tabulación vertical (VT, Ctrl-K, 0x0B)

\o000

- carácter cuyo valor en el sistema octal es 000 (de una a tres cifras) [0-7]

\dDDD

- carácter cuyo valor en el sistema decimal es DDD (de una a tres cifras) [0-9]

\xHH

- carácter cuyo valor en el sistema hexadecimal es HH [0-9A-F]

Otros

\`

- coincide con una cadena nula al inicio de la línea (idéntico a ^)

'

- coincide con una cadena nula al final de la línea (idéntico a $)

\b

- coincide con una cadena vacía al extremo de una palabra. Denota el límite entre una palabra y un carácter que no sea una palabra

\B

- coincide con una cadena vacía que no se encuentre al extremo de una palabra. Denota el límite entre un carácter, salvo una palabra, y una palabra

\w

- cualquier palabra de la clase: [A-Za-z0-9_] (underscore _ incluido)

\W

- cualquier palabra que no pertenezca a la clase: [^A-Za-z0-9_] (underscore _ incluido)

\s

- cualquier carácter de espaciado: espacio, tabulación horizontal o vertical

\S

- uno o varios caracteres de espaciado

\<

- coincide con una cadena vacía al inicio de una palabra

\>

- coincide con una cadena vacía al final de una palabra

\e

- fin de conversión de mayúsculas o minúsculas

\l

- conversión del próximo carácter en minúscula

\L

- conversión de los caracteres restantes en minúscula

\u

- conversión del próximo carácter en mayúscula

\U

- conversión de los caracteres restantes en mayúscula

Las clases de caracteres

[:alnum:]

- caracteres alfanuméricos [A-Za-z0-9]

[:alpha:]

- caracteres alfabéticos [A-Za-z]

[:digit:]

- cifras [0-9]

[:lower:]

- caracteres en minúsculas [a-z]

[:upper:]

- caracteres en mayúsculas [A-Z]

[:print:]

- caracteres imprimibles [-~]

[:punct:]

- caracteres de puntuación [!-/:-@[-`{-~]

[:space:]

- espacios, tabulaciones y cualquier carácter vacío [\t\v\f]

[:blank:]

- espacio y tabulación [\x09]

[:graph:]

- cualquier carácter imprimible [!-~] (excepto los espacios vacíos)

[:cntrl:]

- caracteres de control [\x00-\x19\x7F]

[:xdigit:]

- números hexadecimales [0-9a-fA-F]

Las diferentes versiones

Unix

- GNU sed v4.0.5 (Gsed) Descarga
  - Ultima versión oficial de GNU Sed ofreciendo la edición actual (-i)
- Ssed v3.60 Descarga
  - (Small/Stupid Stream EDitor) Versión recomendada
- BSD multi-byte sed Descarga
  - (Japonesa) Basada en la ultima versión de GNU Sed

Windows

- GnuWin32-Sed v4.1.5 Descargar
  - La versión ejecutable para MS Windows 95 / 98 / ME / NT / 2000 y XP

Depuradores

A continuación 2 depuradores que te permitirán comprender mejor el funcionamiento de Sed y, en algunos casos, te ahorrarán enormes horas de trabajo.

- sd.ksh y sd.sh.txt son dos pequeños scripts escritos en Korn shell y Bourne shell

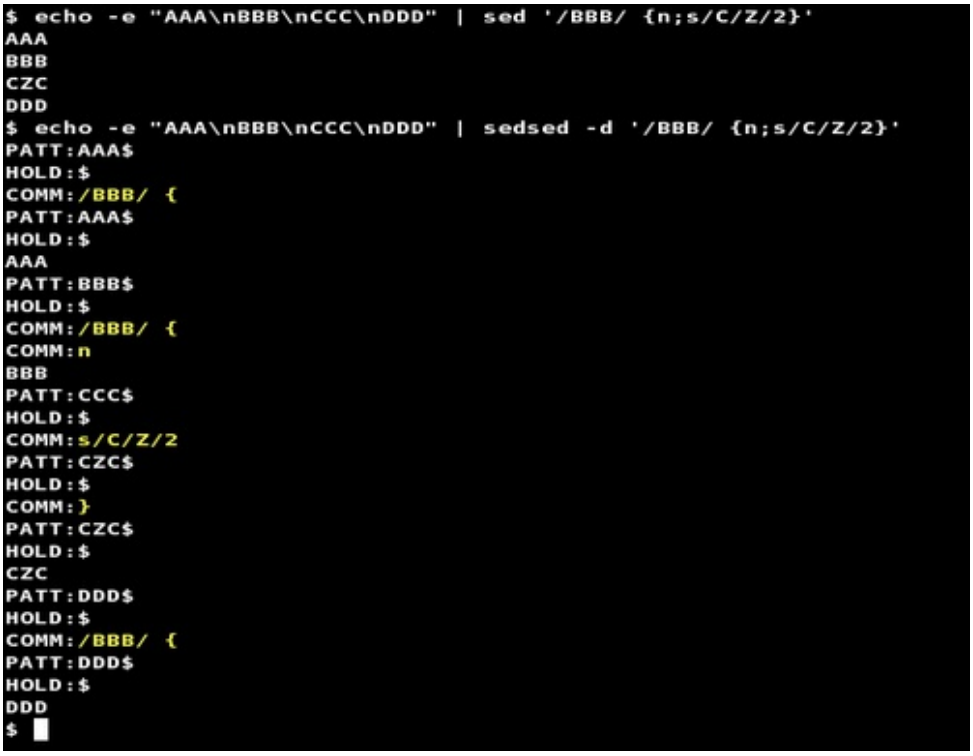
respectivamente que incluyen un manual de uso al final del fichero.
- sedsed está escrito en Python (por lo que debe ser instalado en tu sistema) y te permite visualizar el estado del buffer y los comandos interpretados.

A continuación un mini tutorial para el uso de “sedsed”.

```
Usage : sedsed OPTION [-e sedscript] [-f sedscriptfile] [inputfile]

OPCIONES:

   -f, --file         lectura de los comandos desde el fichero indicado
   -e, --expression     permite encadenar varios comandos
   -n, --quiet        solicitud implicita para no mostrar el estado de la memoria principal
       --silent       equivalente a --quiet

   -d, --debug       activa el modo debug
       --hide         oculta ciertas opciones de depuración (opciones: PATT,HOLD,COMM)
       --color        activa la salida del depurador coloreado (activado por defecto)
       --nocolor       desactiva la salida del depurador coloreado
       --dump-debug     listado del depurador en la pantalla

   -i, --indent        indentación del script, un comando por línea
       --prefix       indentación precedida por espacios o tabulaciones (4 espacios por defecto)

   -t, --tokenize      modo verbosa, muestra cada comando con más información
   -H, --htmlize      convierte un script sed en una página HTML coloreada

   -V, --version       muestra la versión del programa y termina
   -h, --help         muestra una página de ayuda y termina
```

Significado a la salida: **PATT**: Muestra el contenido del espacio patrón (memoria principal) **HOLD**: Muestra el contenido de la memoria secundaria **COMM**: El comando SED debe ser ejecutado **$** Delimita el contenido de PATT y HOLD La sintaxis más común es el modo debug simple (-d):

```
echo -e "AAA\nBBB\nCCC\nDDD" | sed '/BBB/ {n;s/C/Z/2}'
echo -e "AAA\nBBB\nCCC\nDDD" | sedsed -d '/BBB/ {n;s/C/Z/2}'
```

 Esquema: debogage_-d.png En modo indentación:

```
sedsed -i -n ':notag;/tag/!{1!H;1h;x;s/\n/ /g;x;$b lastline;d;};/tag/{x;/^$/!p;$b lastline;d;b notag;};:lastline;x;p;'
```

Esquema: debogage_indent.png Ocultar la visualización de la memoria secundaria:

```
echo -e "AAA\nBBB\nCCC\nDDD" | sedsed -d --hide=HOLD -n '/BBB/ {n;s/C/Z/2p}'
```

¿Cuando no debo utilizar Sed?

Cuando ya existen herramientas apropiadas y efectúan la tarea fácilmente y con mayor rapidez, por jemplo:

- Búsqueda de patrón simple:
  - grep "patrón" fichero # sed -n "/patrón/p' fichero
- Exclusión de un patrón simple
  - grep -v "patrón" fichero # sed "/patrón/!d' fichero
- Búsqueda de patrón y visualización de líneas de contexto antes/después

    - grep -A1 -B1 "patón" fichero # sed -n '/patrón/! {x;d;}; /patrón/{x;p;x;p;n;p;x;}' fichero
- Numeración de líneas
    - cat -n fichero # sed -e '=' fichero | sed 'N;s/\n/\t/'
    - nl fichero
- Eliminación de saltos de línea
    - tr '\n' ' ' < fichero# sed ':bucle; N; $! b bucle; s/\n//g' fichero
- Eliminación de caracteres individuales
    - tr -d "[w-z]" < fichero # sed 's/[w-z]//g' fichero
- Repetición de caracteres
    - echo "Booonjoouuur" | tr -s "ou" # echo "Booonjoouuur" | sed 's/oo*/o/g;s/uu*/u/'
- Transposición de caracteres
    - echo "ABCDEF" | tr "[A-F]" "[a-f]" # echo "ABCDEF" | sed 'y/ABCDEF/abcdef/'
- Formato de ficheros, mejor utilizar los comandos:
    - fold
    - fmt
    - par

Para ciertas tareas que “awk” y “perl” las realizan mucho mas fácil y rápido, como por ejemplo:

- Contar campos y caracteres
- Contar líneas de un bloque u objetos de un fichero
- Operaciones matemáticas
- Calcular la longitud de una cadena
- Manipular datos binarios
- Bucles en tablas o listas
- etc.

Limites conocidos de las diferentes versiones

A continuación algunos limites conocidos de las versiones distribuidas de Sed, dependiendo claro estas del hardware, la memoria, el sistema operativo y de las librerías C utilizadas durante la compilación de Sed.

- Longitud máxima de una línea
    - GNU sed: sin limitación
    - ssed: sin limitación
- Tamaño máximo de la memoria principal y secundaria
    - GNU sed: sin limitación
    - ssed: sin limitación
- Número máximo de ficheros que pueden ser leídos por el comando "r"
    - GNU sed v3+: sin limitación
    - ssed: sin limitación
    - GNU sed v2.05: el total de lecturas (r) y escrituras (w) no debe exceder 32
- Numero máximo de ficheros que pueden ser escritos por el comando “w”
    - GNU sed v3+: sin limitación
    - ssed: sin limitación
    - GNU sed v2.05: el total de lecturas (r) y escrituras (w) no debe exceder 32
- Tamaño máximo de un nombre de etiqueta
    - GNU sed: sin limitación
    - ssed: sin limitación

    - BSD sed: 8 caracteres
- Tamaño máximo de nombre de fichero en escritura
    - GNU sed: sin limitación
    - ssed: sin limitación
    - BSD sed: 8 caracteres
- Numero máximo de conexiones
    - GNU sed: sin limitación
    - ssed: sin limitación

Referencias

A continuación encontraras algunas obras así como los sitios Web que utilice para elaborar este artículo.

Libros

- sed & awk, Second Edition

Enlaces

Principiantes y conocedores

- info sed
- man sed
- Sed - An Introduction and Tutorial
- THE SED FAQ
- sed, a stream editor
- HANDY ONE-LINERS FOR SED

Avanzados

- scripts
- tutoriales

IRC

- Freenode
    - irc://irc.freenode.net/#sed

SED - The Stream EDitor - Part III

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