

SED, una introducción y tutorial

Por Bruce Barnett

Last modified: Mon Dec 7 10:13:59 2020

Copyright 1994, 1995 Bruce Barnett and General Electric Company

Copyright 2001,2005,2007,2011,2013 Bruce Barnett

All rights reserved

You are allowed to print copies of this tutorial for your personal use, and link to this page, but you are not allowed to make electronic copies, or redistribute this tutorial in any form without permission.

Original version written in 1994 and published in the Sun Observer

Introducción a Sed

Cómo usar sed, un editor especial para modificar archivos de forma automática. Si desea escribir un programa para realizar cambios en un archivo, sed es la herramienta que debe utilizar.

Hay algunos programas que son el auténtico caballo de batalla en la caja de herramientas de UNIX. Estos programas son simples de usar para aplicaciones simples, pero tienen un amplio conjunto de acciones para realizar tareas complejas. No deje que el complejo potencial de un programa le impida hacer uso de los aspectos más simples. Empezaré con los conceptos simples e introduciré los temas avanzados más adelante.

Cuando escribí esto por primera vez (en 1994), la mayoría de las versiones de sed no permitían colocar comentarios dentro del script. Las líneas que comienzan con los caracteres '#' son comentarios. Las versiones más recientes de sed pueden soportar comentarios al final de la línea también.

Una forma de pensar en esto es que la versión antigua, "clásica", fue la base de las versiones de sed de GNU, FreeBSD y Solaris. Y para ayudarle a entender con qué tenía que trabajar, aquí está la página del manual de sed[1] de Sun/Oracle.

La horrible verdad sobre sed

Sed es el editor de flujos por excelencia (**Stream EDitor**). Si eso suena extraño, imagínese un arroyo fluyendo a través de una tubería. Vale, no puedes ver un arroyo si está dentro de una tubería. Eso me pasa por intentar una analogía fluida. Si quieres literatura, lee a James Joyce.

En cualquier caso, sed es una utilidad maravillosa. Desafortunadamente, la mayoría de la gente nunca aprende su verdadero poder. El lenguaje es muy simple, pero la documentación es terrible. Las páginas del manual en línea de Solaris para sed tienen cinco páginas, y dos de ellas describen los 34 errores diferentes que puede obtener. Un programa que dedica tanto espacio a documentar los errores como a documentar el lenguaje tiene una importante curva de aprendizaje.

No te preocupes. No es tu culpa que no entiendas sed. Cubriré sed completamente. Pero describiré las características en el orden en que las aprendí. No aprendí todo a la vez. Tú tampoco necesitas hacerlo.

El comando esencial: s para la sustitución

Sed tiene varios comandos, pero la mayoría de la gente sólo aprende el comando de sustitución: **s**. El comando de sustitución cambia todas las ocurrencias de la expresión regular en un nuevo valor. Un ejemplo sencillo es cambiar **day** en el archivo **old** por **night** en el archivo **new**:

```
sed s/day/night/ old >new
```

O de otra manera (para los principiantes de UNIX)

```
sed s/day/night/ old >new
```

y para los que quieran probar esto

```
echo day | sed s/day/night/
```

Esto dará como resultado **night**.

No puse comillas alrededor del argumento porque este ejemplo no las necesitaba. Si lees mi anterior tutorial sobre las comillas, entenderás por qué no las necesita. Sin embargo, te recomiendo que uses las comillas. Si tiene meta-caracteres en el comando, las comillas son necesarias. Y si no estás seguro, es un buen hábito, y a partir de ahora citaré los futuros ejemplos para enfatizar la "mejor práctica". Usando el carácter fuerte [comillas simples], sería

```
sed 's/día/noche/' < antiguo > nuevo
```

Debo enfatizar que el editor sed cambia exactamente lo que usted le diga. Así que si ejecutas

```
echo domingo | sed 's/día/noche/'
```

Esto mostraría la palabra **Sunnight** porque sed encontró la cadena **day** en la entrada.

Otro concepto importante es que sed está orientado a trabajar con líneas. Suponga que tiene el archivo de entrada

```
one two three, one two three  
four three two one  
one hundred
```

y ha utilizado el comando

```
sed 's/one/ONE/' <file
```

La salida sería

```
ONE two three, one two three  
four three two ONE  
ONE hundred
```

Tenga en cuenta que esto cambió **one** a **ONE** una vez en cada línea. La primera línea tenía **uno** dos veces, pero sólo se cambió la primera ocurrencia. Este es el comportamiento por defecto. Si quiere algo diferente, tendrá que usar algunas de las opciones que están disponibles. Las explicaré más adelante.

Así que continuemos.

Hay cuatro partes en este comando de sustitución:

s/one/ONE/	
s	Comando de sustitución
/.../.../	Delimitadores
one	Patrón de Expresión Regular Patrón de Búsqueda
ONE	Cadena de sustitución

El patrón de búsqueda está en la parte izquierda y la cadena de sustitución está en la parte derecha.

Hemos cubierto las comillas y las expresiones regulares.. Eso es el 90% del esfuerzo necesario para aprender el comando de sustitución. Para decirlo de otra manera, usted ya sabe cómo manejar el 90% de los usos más frecuentes de sed. Hay unos ... pocos puntos finos que cualquier futuro experto en sed debería conocer. [Acaba de terminar la sección 1. Sólo hay 63 secciones más que cubrir. :-] Oh. Y puede que quieras marcar esta página, por si acaso no la terminas.

La barra inclinada como delimitador

El carácter que sigue a la **s** es el delimitador. Es convencionalmente una barra, porque es lo que usan **ed**, **more** y **vi**. Sin embargo, puede ser cualquier cosa que desee. Si quiere cambiar un nombre de ruta que contiene una barra - digamos **/usr/local/bin** a **/common/bin** - puede usar la barra invertida para citar la barra:

```
sed 's/\usr/local/bin/common/bin/' <old >new
```

Gulp. Algunos llaman a esto 'Picket Fence' y es feo. Es más fácil de leer si utiliza un subrayado en lugar de una barra como delimitador:

```
sed 's_usr/local/bin_common/bin_' <old >new
```

Algunos utilizan dos puntos:

```
sed 's:/usr/local/bin:/common/bin:' <old >new
```

Otros utilizan el carácter "|".

```
sed 's|/usr/local/bin|/common/bin|' <old >new
```

Elige uno que te guste. Mientras no esté en la cadena que buscas, todo vale. Y recuerda que necesitas tres delimitadores. Si obtienes un "comando `s` sin terminar" es porque te falta uno de ellos.

Usar & como cadena de coincidencia

A veces se quiere buscar un patrón y añadir algunos caracteres, como paréntesis, alrededor o cerca del patrón encontrado. Es fácil hacer esto si está buscando una cadena en particular:

```
sed 's/abc/(abc)/' <old >new
```

Esto no funcionará si no sabes exactamente lo que vas a encontrar. ¿Cómo puede poner la cadena que ha encontrado en la cadena de reemplazo si no sabe lo que es?

La solución requiere el carácter especial "&". Corresponde al patrón encontrado.

```
sed 's/[a-z]*/&/' <old >new
```

Puede tener cualquier número de "&" en la cadena de sustitución. También puede duplicar un patrón, por ejemplo, el primer número de una línea:

```
% echo "123 abc" | sed 's/[0-9]*/& &/'  
123 123 abc
```

Permítame modificar ligeramente este ejemplo. Sed coincidirá con la primera cadena, y la hará lo más codiciosa posible. Lo explicaré más adelante. Si no quiere que sea tan codicioso (es decir, que limite las coincidencias), necesita poner restricciones a la coincidencia.

La primera coincidencia para `[0-9]*` es el primer carácter de la línea, ya que coincide con cero o más números. Así que si la entrada fuera `abc 123` la salida no cambiaría (bueno,

excepto por un espacio antes de las letras]. Una mejor manera de duplicar el número es asegurarse de que coincide con un número:

```
% echo "123 abc" | sed 's/[0-9][0-9]*/& &/'  
123 123 abc
```

La cadena `abc` no se ha modificado, porque no ha sido igualada por la expresión regular. Si quiere eliminar `abc` de la salida, debe ampliar la expresión regular para que coincida con el resto de la línea y excluir explícitamente parte de la expresión utilizando `(,)` y `\1`, que es lo que viene a continuación.

Expresiones regulares extendidas

Quisiera añadir un comentario rápido aquí porque hay otra forma de escribir el script anterior.

`[0-9]*` coincide con cero o más números. `[0-9][0-9]*` coincide con uno o más números.

Otra forma de hacer esto es usar el metacarácter `+` y usar el patrón `[0-9]+` ya que el `+` es un carácter especial cuando se usan "expresiones regulares extendidas". Las expresiones regulares extendidas tienen más potencia, pero los scripts `sed` que trataran el `+` como un carácter normal se romperían. Por lo tanto, debe habilitar explícitamente esta extensión con una opción de línea de comandos.

GNU `sed` activa esta característica si se utiliza la opción de línea de comandos `-r`. Así que lo anterior también podría escribirse usando

```
% echo "123 abc" | sed -r 's/[0-9]+/& &/'  
123 123 abc
```

Mac OSX y FreeBSD utilizan `-E` en lugar de `-r`. Para más información sobre las expresiones regulares extendidas, consulte [Expresiones regulares](#) y la descripción del argumento `-r`.

Uso de `\1` para mantener parte del patrón

Ya he descrito el uso de `(,)` y `\1` en mi tutorial sobre expresiones regulares. Para repasar, los paréntesis escapados (*es decir, los paréntesis con barras invertidas antes de ellos*) recuerdan

una subcadena de los caracteres coincidentes con la expresión regular. Puede utilizarlo para excluir parte de los caracteres coincidentes con la expresión regular. El `\1` es el primer patrón recordado, y el `\2` es el segundo patrón recordado. Sed tiene hasta nueve patrones recordados.

Si quiere mantener la primera palabra de una línea, y borrar el resto de la línea, marque la parte importante con el paréntesis:

```
sed 's/\([a-z]*\) .*/\1/'
```

Debería explicar esto con más detalle. Las expresiones regulares son codiciosas, e intentan coincidir con todo lo posible. La expresión regular `[a-z]*` coincide con cero o más letras minúsculas, e intenta coincidir con el mayor número de caracteres posible. El `.*` coincide con cero o más caracteres después de la primera coincidencia. Dado que la primera coge todas las letras minúsculas contiguas, la segunda coincide con cualquier otra cosa. Por lo tanto, si escribe

```
echo abcd123 | sed 's/\([a-z]*\) .*/\1/'
```

Esto mostrará `abcd` y borrará los números.

Si quiere cambiar dos palabras, puede recordar dos patrones y cambiar el orden

```
sed 's/\([a-z]*\) \([a-z]*\) /\2 \1/'
```

Observe el espacio entre los dos patrones recordados. Esto se utiliza para asegurarse de que se encuentran dos palabras. Sin embargo, esto no hará nada si se encuentra una sola palabra, o cualquier línea sin letras. Puede insistir en que las palabras tengan al menos una letra utilizando

```
sed 's/\([a-z][a-z]*\) \([a-z][a-z]*\) /\2 \1/'
```

o utilizando expresiones regulares extendidas (tenga en cuenta que `(` y `)` ya no necesitan tener una barra invertida):

```
sed -r 's/([a-z]+) ([a-z]+)/\2 \1/' # Usando GNU sed  
sed -E 's/([a-z]+) ([a-z]+)/\2 \1/' # Usando Apple Mac OS X
```

El `\1` no tiene por qué estar en la cadena de sustitución (en la parte derecha). Puede estar en el patrón que está buscando (en la parte izquierda). Si quiere eliminar las palabras duplicadas, puede probar

```
sed 's/\([a-z]*\) \1/\1/'
```

Si quiere detectar palabras duplicadas, puede utilizar

```
sed -n '/\([a-z][a-z]*\) \1/p'
```

o con expresiones regulares extendidas

```
sed -rn '/([a-z]+) \1/p' # GNU sed  
sed -En '/([a-z]+) \1/p' # Mac OS X
```

Esto, cuando se usa como filtro, imprimirá las líneas con palabras duplicadas.

El valor numérico puede tener hasta nueve valores: `\1` a `\9`. Si desea invertir los tres primeros caracteres de una línea, puede utilizar

```
sed 's/^(.)\(.)\(.)/\3\2\1/'
```

Indicadores del patrón Sed

Puede añadir banderas adicionales después del último delimitador. Habrás notado que usé una `'p'` al final del comando sustituto anterior. También añadí la opción `'-n'`. Permítame primero cubrir la `'p'` y otras banderas de patrón. Estas banderas pueden especificar lo que sucede cuando se encuentra una coincidencia. Permítanme describirlas.

/g - Reemplazo global

La mayoría de las utilidades de UNIX trabajan sobre archivos, leyendo una línea a la vez. Sed, por defecto, es de la misma manera. Si le dice que cambie una palabra, sólo cambiará la primera ocurrencia de la palabra en una línea. Usted puede querer hacer el cambio en cada palabra de la línea en lugar de la primera. Por ejemplo, pongamos paréntesis alrededor de las palabras de una línea. En lugar de usar un patrón como `"[A-Za-z]*"` que no coincidirá con palabras como "no", usaremos un patrón, `"[^]*"`, que coincide con todo excepto con un espacio. Bueno, esto también coincidirá con cualquier cosa porque `"*"` significa cero o más. La versión actual del sed de Solaris (en el momento en que escribí esto) puede ponerse descontenta con patrones como este, y generar errores como "Línea de salida demasiado larga" o incluso ejecutarse

eternamente. Considero que esto es un error, y lo he reportado a Sun. Como solución, hay que evitar que la cadena nula coincida con la bandera "g" de sed. Un ejemplo de solución es: "[^]*[^\n]*". Lo siguiente pondrá paréntesis alrededor de la primera palabra:

```
sed 's/[^ ]*/(&)/' <antiguo >nuevo
```

Si quiere que haga cambios para cada palabra, añada una "g" después del último delimitador y utilice la solución

```
sed 's/[^ ]*[^\n]*(&)/g' <viejo >nuevo
```

¿Sed es recursivo?

Sed sólo opera sobre los patrones encontrados en los datos de entrada. Es decir, se lee la línea de entrada, y cuando se encuentra un patrón, se genera la salida modificada, y se escanea el resto de la línea de entrada. El comando **s** no escanea la salida recién creada. Es decir, no tiene que preocuparse por expresiones como

```
sed 's/loop/loop the loop/g' <old >new
```

Esto no causará un bucle infinito. Si se ejecuta un segundo comando **s**, podría modificar los resultados de un comando anterior. Más adelante le mostraré cómo ejecutar múltiples comandos.

/1, /2, etc. Especificación de la ocurrencia

Sin banderas, se modifica la primera sustitución coincidente. Con la opción "g", se modifican todas las coincidencias. Si quiere modificar un patrón en particular que no es el primero de la línea, puede usar **\(** (y **\)** para marcar cada patrón, y usar **\1** para poner el primer patrón sin cambios. El siguiente ejemplo mantiene la primera palabra en la línea pero borra la segunda:

```
sed 's/\([a-zA-Z]*\) \([a-zA-Z]*\) /\1 /' <old >new
```

Qué asco. Hay una manera más fácil de hacer esto. Puedes añadir un número después del comando de sustitución para indicar que sólo quieres que coincida con ese patrón en particular. Por ejemplo:

```
sed 's/[a-zA-Z]* //2' <old >new
```

Puedes combinar un número con la bandera **g** (global). Por ejemplo, si quiere dejar la primera palabra, pero cambiar la segunda, tercera, etc. para que sean BORRADAS, utilice **/2g**:

```
sed 's/[a-zA-Z]* /DELETED /2g' <old >new
```

He oído que la combinación del número con el comando **g** no funciona en MacOS, y quizás también en la versión FreeBSD de sed.

No confundas **/2** y **\2**. El **/2** se utiliza al final. El **\2** se usa dentro del campo de reemplazo.

Observe el espacio después del carácter *****. Sin el espacio, sed se ejecutará mucho, mucho tiempo. [Nota: este error probablemente ya esté arreglado.] Esto se debe a que la bandera numérica y la bandera **g** tienen el mismo error. También debería poder utilizar el patrón

```
sed 's/[^ ]*/2' <old >new
```

pero esto también se come CPU. Si esto funciona en tu ordenador, y lo hace en algunos sistemas UNIX, podrías eliminar la contraseña encriptada del archivo de contraseñas:

```
sed 's/[^:]*//2' </etc/passwd >/etc/password.new
```

Pero esto no me funcionó la vez que escribí esto. Usar **[^:] [^:] *** como solución no ayuda porque no coincidirá con una contraseña inexistente, y en su lugar borrará el tercer campo, ¡que es el ID del usuario! En su lugar, tiene que utilizar el feo paréntesis

```
sed 's/^\([^:]*\):[^:]:/\1:/' </etc/passwd >/etc/password.new
```

También puedes añadir un carácter al primer patrón para que no coincida con el patrón nulo:

```
sed 's/[^:]*:/:/2' </etc/passwd >/etc/password.new
```

El indicador de número no está restringido a un solo dígito. Puede ser cualquier número del 1 al 512. Si quieres añadir dos puntos después del carácter 80 en cada línea, puedes escribir

```
sed 's/./&:/80' <file >new
```

También puede hacerlo de la manera más difícil, utilizando 80 puntos:

```
sed  
's/^.....  
...../&:/' <file >new
```

/p - imprimir

Por defecto, sed imprime cada línea. Si realiza una sustitución, el nuevo texto se imprime en lugar del anterior. Si utiliza un argumento opcional para sed, `sed -n`, no imprimirá, por defecto, ninguna línea nueva. Cubriré esta y otras opciones más adelante. Cuando se utiliza la opción `-n`, la bandera `p` hará que se imprima la línea modificada. Esta es una forma de duplicar la función de grep con sed:

```
sed -n 's/pattern/&/p' <file
```

Pero más adelante se describe una versión más sencilla

Escribir en un archivo con /w nombre

Hay una bandera más que puede seguir al tercer delimitador. Con ella, se puede especificar un archivo que recibirá los datos modificados. Un ejemplo es el siguiente, que escribirá todas las líneas que comiencen con un número par, seguido de un espacio, en el archivo `even`:

```
sed -n 's/^[0-9]*[02468] /&/w even' <file
```

En este ejemplo, el archivo de salida no es necesario, ya que la entrada no fue modificada. Debe haber exactamente un espacio entre la `w` y el nombre del archivo. También puede tener diez archivos abiertos con una sola instancia de sed. Esto le permite dividir un flujo de datos en archivos separados. Utilizando el ejemplo anterior, combinado con los comandos de sustitución múltiple que se describen más adelante, podría dividir un archivo en diez partes dependiendo del último dígito del primer número. También puede utilizar este método para registrar información de error o depuración en un archivo especial.

/I - Ignorar Mayusculas / Minusculas (case)

GNU ha añadido otra bandera de patrón - `/I`

Esta bandera hace que el patrón coincida con las mayúsculas y minúsculas. Esto coincidirá con `abc`, `aBc`, `ABC`, `AbC`, etc:

```
sed -n '/abc/I p' <old >new
```

Tenga en cuenta que un espacio después de `/I` y el comando `p` (print) enfatiza que `p` no es un modificador del proceso de coincidencia de patrones, sino un comando a ejecutar después de la coincidencia de patrones.

Combinación de indicadores de sustitución

Puede combinar banderas cuando tenga sentido. Tenga en cuenta que la `w` tiene que ser la última bandera. Por ejemplo, el siguiente comando funciona:

```
sed -n 's/a/A/2pw /tmp/file' <old >new
```

A continuación discutiré las opciones de sed, y las diferentes formas de invocar sed.

Argumentos e invocación de sed

Anteriormente, sólo he utilizado un comando de sustitución. Si necesitas hacer dos cambios, y no quieres leer el manual, puedes juntar varios comandos sed:

```
sed 's/BEGIN/begin/' <old | sed 's/END/end/' >new
```

Esto utiliza dos procesos en lugar de uno. Un gurú de sed nunca usa dos procesos cuando uno puede hacerlo.

Comandos múltiples con el comando -e

Un método para combinar múltiples comandos es utilizar una `-e` antes de cada comando:

```
sed -e 's/a/A/' -e 's/b/B/' <old >new
```

La `-e` no es necesaria en los ejemplos anteriores porque sed sabe que siempre debe haber un solo comando. Si le da a sed un argumento, debe ser un comando, y sed editará los datos leídos de la entrada estándar.

La versión con argumentos largos es

```
sed --expression='s/a/A/' --expression='s/b/B/' <old >new
```

Véase también Citar varias líneas sed en el shell Bourne

Nombres de archivos en la línea de comandos

Si lo desea, puede especificar archivos en la línea de comandos. Si hay más de un argumento para sed que no comienza con una opción, debe ser un nombre de archivo. El siguiente ejemplo contará el número de líneas en tres archivos que no comienzan con un `#`:

```
sed 's/^#.*//' f1 f2 f3 | grep -v '^$' | wc -l
```

Vamos a dividir esto en partes. El comando de sustitución sed cambia cada línea que comienza con `#` por una línea en blanco. Grep se utilizó para filtrar (eliminar) las líneas vacías. Wc cuenta el número de líneas que quedan. Sed tiene más comandos que hacen que grep sea innecesario. Y `grep -c` puede reemplazar a `wc -l`. Discutiré cómo puedes duplicar algunas de las funcionalidades de grep más adelante.

Por supuesto, puedes escribir el último ejemplo usando la opción `-e`:

```
sed -e 's/^#.*//' f1 f2 f3 | grep -v '^$' | wc -l
```

Hay otras dos opciones para sed.

sed -n: no imprimir

La opción `-n` no imprimirá nada a menos que se encuentre una solicitud explícita de impresión. He mencionado la bandera `/p` en el comando de sustitución como una forma de activar la impresión. Permítame aclarar esto. El comando

```
sed 's/PATTERN/&/p' file
```

actúa como el programa cat si `PATTERN` no está en el archivo: por ejemplo, no se cambia nada. Si `PATTERN` está en el archivo, entonces cada línea que lo tiene se imprime dos veces. Añada la opción `-n` y el ejemplo actúa como grep:

```
sed -n 's/PATTERN/&/p' file
```

No se imprime nada, excepto las líneas con **PATTERN** incluido.

La forma larga del argumento para el comando **-n** es

```
sed --quiet 's/PATTERN/&/p' file
```

o

```
sed --silent 's/PATTERN/&/p' file
```

Uso de 'sed /pattern/'

Sed tiene la capacidad de especificar qué líneas deben ser examinadas y/o modificadas, especificando direcciones antes del comando. Por ahora sólo describiré la versión más sencilla: la dirección **/PATTERN/**. Cuando se utiliza, sólo las líneas que coinciden con el patrón reciben el comando después de la dirección. Brevemente, cuando se usa con la bandera **/p**, las líneas que coinciden se imprimen dos veces:

```
sed '/PATTERN/p' file
```

Y, por supuesto, **PATTERN** es cualquier expresión regular.

Tenga en cuenta que si no incluye un comando, como la **p** para imprimir, obtendrá un error. Cuando escribo

```
echo abc | sed '/a/'
```

obtengo el error

```
sed: -e expression #1, char 3: missing command
```

Además, no es necesario, pero le recomiendo que coloque un espacio después del patrón y del comando. Esto le ayudará a distinguir entre las banderas que modifican la coincidencia del patrón, y los comandos que se ejecutan después de la coincidencia del patrón. Por lo tanto, recomiendo este estilo:

```
sed '/PATTERN/ p' archivo
```

Usando 'sed -n /pattern/p' para duplicar la función de grep

Si desea duplicar la función de grep, combine la opción `-n` (no imprimir) con la bandera de impresión `/p`:

```
sed -n '/PATTERN/p' file
```

sed -f nombre del script

Si tiene un gran número de comandos sed, puede ponerlos en un archivo y utilizar

```
sed -f sedscrip <old >new
```

donde `sedscrip` podría tener el siguiente aspecto

```
# Esto es un comentario - Este script cambia las vocales minúsculas a  
mayúsculas  
s/a/A/g  
s/e/E/g  
s/i/I/g  
s/o/O/g  
s/u/U/g
```

Cuando hay varios comandos en un archivo, cada comando debe estar en una línea separada.

La versión con argumentos largos es

```
sed --file=sedscrip <old >new
```

Vea también aquí cómo escribir un script que ejecute sed directamente

sed en scripts de shell

Si tiene muchos comandos y no caben limpiamente en una línea, puede dividir la línea usando una barra invertida:

```
sed -e 's/a/A/g' \  
    -e 's/e/E/g' \
```

```
-e 's/i/I/g' \  
-e 's/o/O/g' \  
-e 's/u/U/g' <old >new
```

Citando múltiples líneas sed en el shell C (csh)

Puede tener un script sed grande y de varias líneas en el shell de C, pero debe indicarle al shell de C que el entrecomillado continúa a través de varias líneas. Esto se hace colocando una barra invertida al final de cada línea:

```
#!/bin/csh -f  
sed 's/a/A/g \  
s/e/E/g \  
s/i/I/g \  
s/o/O/g \  
s/u/U/g' <old >new
```

Citando múltiples líneas sed en el shell Bourne (bash)

El shell Bourne facilita esta tarea, ya que una cita puede abarcar varias líneas:

```
#!/bin/bash  
sed '  
s/a/A/g  
s/e/E/g  
s/i/I/g  
s/o/O/g  
s/u/U/g' <old >new
```

sed -V

La opción **-V** imprimirá la versión de sed que está utilizando. El argumento largo del comando es

```
sed --versión
```

sed -h

La opción **-h** imprimirá un resumen de los comandos de sed. El argumento largo del comando es


```
sed --help
```

Un script de interpretación de sed

Otra forma de ejecutar sed es utilizarlo como intérprete script. Cree un archivo que contenga

```
#!/bin/sed -f
s/a/A/g
s/e/E/g
s/i/I/g
s/o/O/g
s/u/U/g
```

Haga clic aquí para obtener el archivo: [CapVowel.sed](#)

Si este script estuviera almacenado en un archivo con el nombre "CapVowel" y fuera ejecutable, podría utilizarlo con el simple comando.

```
CapVowel <antiguo >nuevo
```

Comentarios

Los comentarios de sed son líneas donde el primer carácter no blanco es un "#". En muchos sistemas, sed sólo puede tener un comentario, y debe ser la primera línea del script. En el Sun [1988 cuando escribí esto], puedes tener varias líneas de comentarios en cualquier parte del script. Las versiones modernas de Sed soportan esto. Si la primera línea contiene exactamente `#n`, esto hace lo mismo que la opción `-n`: desactivar la impresión por defecto. Esto no se puede hacer con un script intérprete de Sed, porque la primera línea debe comenzar con `#!/bin/sed -f` ya que creo que `#!/bin/sed -nf` genera un error. Funcionaba cuando escribí esto por primera vez [2008]. Tenga en cuenta que `#!/bin/sed -fn` no funciona porque sed piensa que el nombre de archivo del script es `n`. Sin embargo,

```
"#!/bin/sed -nf"
```

sí funciona.

Pasar argumentos a un script sed

Pasar una palabra a un script de la shell que llama a sed es fácil si recuerda mi tutorial sobre el mecanismo de comillas de UNIX. Para repasar, se utilizan las comillas simples para activar y desactivar las comillas. Un simple script de shell que utiliza sed para emular grep es:

```
#!/bin/sh
sed -n 's/'$1'&/p'
```

Sin embargo - hay un problema sutil con este script. Si tiene un espacio como argumento, el script causaría un error de sintaxis, como

```
sed: -e expression #1, char 4: unterminated `s' command
```

Una versión mejor evitaría que esto ocurriera:

```
#!/bin/sh
sed -n 's/"$1"/&/p'
```

Haga clic aquí para obtener el archivo: [sedgrep.sed](#)

Si esto estuviera almacenado en un archivo llamado `sedgrep`, podría escribir

```
sedgrep '[A-Z][A-Z]' <file
```

Esto permitiría a sed actuar como el comando grep.

Uso de sed en un documento de shell here-is

Puede usar sed para pedir al usuario algunos parámetros y luego crear un archivo con esos parámetros rellenos. Puede crear un archivo con valores ficticios en su interior, y utilizar sed para cambiar esos valores ficticios. Una forma más sencilla es utilizar el documento "here is", que utiliza parte del script de la shell como si fuera la entrada estándar:

```
#!/bin/sh
echo -n '¿cuál es el valor? '
read value
sed 's/XYZ/'$value'/' <<EOF
El valor es XYZ
EOF
```

Cuando se ejecuta, el script dice:

```
¿cuál es el valor?
```

Si escribes "123", la siguiente línea será:

```
El valor es 123
```

Admito que este es un ejemplo artificioso. Los documentos "Here is" pueden tener valores evaluados sin el uso de sed. Este ejemplo hace lo mismo:

```
#!/bin/sh
echo -n 'what is the value? '
read value
cat <<EOF
The value is $value
EOF
```

Sin embargo, la combinación de documentos "here is" con sed puede ser útil para algunos casos complejos.

Tenga en cuenta que

```
sed 's/XYZ/'$value'/' <<EOF
```

dará un error de sintaxis si el usuario escribe una respuesta que contiene un espacio, como "a b c". Una forma mejor sería poner comillas dobles alrededor de la evaluación del valor:

```
#!/bin/sh
echo -n 'what is the value? '
read value
sed 's/XYZ/'"$value"'/ ' <<EOF
The value is XYZ
EOF
```

Cubrí esto en mi tutorial sobre las comillas.

Haga clic aquí para obtener el archivo: [sed_hereis.sed](#)

Múltiples comandos y orden de ejecución

A medida que exploremos más comandos de sed, los comandos se volverán complejos, y la secuencia real puede ser confusa. En realidad es bastante simple. Se lee cada línea. Cada comando, en el orden especificado por el usuario, tiene una oportunidad de operar en la línea de entrada. Una vez realizadas las sustituciones, el siguiente comando tiene la oportunidad de operar en la misma línea, que puede haber sido modificada por los comandos anteriores. Si alguna vez tiene una pregunta, la mejor manera de aprender lo que sucederá es crear un pequeño ejemplo. Si un comando complejo no funciona, hazlo más simple. Si tiene problemas para hacer funcionar un script complejo, divídalo en dos scripts más pequeños y canalice los dos scripts juntos.

Direcciones y rangos de texto

Sólo ha aprendido un comando, y puede ver lo poderoso que es sed. Sin embargo, todo lo que está haciendo es un grep y un sustituto. Es decir, el comando sustituto está tratando cada línea por sí misma, sin preocuparse de las líneas cercanas. Lo que sería útil es la capacidad de restringir la operación a ciertas líneas. Algunas restricciones útiles podrían ser

- Especificar una línea por su número.
- Especificar un rango de líneas por su número.
- Todas las líneas que contengan un patrón.
- Todas las líneas desde el principio de un fichero hasta una expresión regular.
- Todas las líneas desde una expresión regular hasta el final del archivo.
- Todas las líneas entre dos expresiones regulares.

Sed puede hacer todo eso y más. Cada comando en sed puede ser precedido por una dirección, rango o restricción como los ejemplos anteriores. La restricción o dirección precede inmediatamente al comando:

- comando de restricción

Restricción a un número de línea

La restricción más sencilla es un número de línea. Si quiere eliminar el primer número de la línea 3, sólo tiene que añadir un **3** antes del comando

```
sed '3 s/[0-9][0-9]*//' <file >new
```

Patrones

Muchas utilidades de UNIX, como vi y otras, utilizan una barra para buscar una expresión regular. Sed utiliza la misma convención, siempre que termine la expresión con una barra. Para eliminar el primer número de todas las líneas que comienzan con "#", utilice

```
sed '/^#/ s/[0-9][0-9]*//'
```

He puesto un espacio después de la **/expresión/** para que sea más fácil de leer. No es necesario, pero sin él el comando es más difícil de entender. Sed proporciona algunas opciones adicionales al especificar expresiones regulares. Pero hablaré de ellas más adelante. Si la expresión comienza con una barra invertida, el siguiente carácter es el delimitador. Para utilizar una coma en lugar de una barra invertida, utilice

```
sed '\,^#, s/[0-9][0-9]*//'
```

La principal ventaja de esta función es la búsqueda de barras. Suponga que quiere buscar la cadena **/usr/local/bin** y quiere cambiarla por **/common/all/bin**. Podrías utilizar la barra invertida para escapar de la barra:

```
sed '/\usr\local\bin/ s/\usr\local\/common/all/'
```

Sería más fácil de seguir si se utiliza un subrayado en lugar de una barra como búsqueda. Este ejemplo utiliza el subrayado tanto en el comando de búsqueda como en el de sustitución:

```
sed '_usr/local/bin_ s_usr/local_/common/all_'
```

Esto ilustra por qué los scripts sed tienen fama de oscuros. Podría ser perverso y mostrarle el ejemplo que buscará todas las líneas que empiecen por una **g**, y cambiará cada **g** de esa línea por una **s**:

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

Añadir un espacio y usar un guión bajo después del comando de sustitución hace que esto sea mucho más fácil de leer:

```
sed '/^g/ s_g_s_g'
```

Eh, me retracto. No tiene remedio. Hay una lección aquí: Utilice los comentarios libremente en un script sed. Puede que tengas que eliminar los comentarios para ejecutar el script en un sistema operativo diferente [más antiguo], pero ahora sabes cómo escribir un script sed para hacerlo muy fácilmente. Los comentarios son algo bueno. Puede que hayas entendido el script perfectamente cuando lo escribiste. Pero dentro de seis meses podría parecer un ruido de módem. Y si no entiendes esa referencia, imagina a un niño de 8 meses escribiendo en un ordenador.

Rangos por número de línea

Puede especificar un rango en los números de línea insertando una coma entre los números. Para restringir una sustitución a las 100 primeras líneas, puede utilizar

```
sed '1,100 s/A/a/ '
```

Si sabe exactamente cuántas líneas hay en un archivo, puede indicar explícitamente ese número para realizar la sustitución en el resto del archivo. En este caso, suponga que utilizó `wc` para averiguar que hay 532 líneas en el archivo:

```
sed '101,532 s/A/a/ '
```

Una forma más sencilla es utilizar el carácter especial `$`, que significa la última línea del archivo.

```
sed '101,$ s/A/a/ '
```

El `$` es una de esas convenciones que significan "último" en utilidades como `cat -e`, `vi` y `ed`. "`cat -e`" Los números de línea son acumulativos si se editan varios archivos. Es decir

```
sed '200,300 s/A/a/ ' f1 f2 f3 >new
```

es lo mismo que

```
cat f1 f2 f3 | sed '200,300 s/A/a/ ' >new
```

Rangos por patrones

Puede especificar dos expresiones regulares como rango. Suponiendo que un `#` inicia un comentario, puede buscar una palabra clave, eliminar todos los comentarios hasta que vea la segunda palabra clave. En este caso las dos palabras clave son `start` y `stop`:

```
sed '/start/,/stop/ s/#.*//'
```

El primer patrón activa una bandera que le dice a sed que realice el comando de sustitución en cada línea. El segundo patrón desactiva la bandera. Si los patrones `start` y `stop` aparecen dos veces, la sustitución se realiza ambas veces. Si el patrón `stop` no aparece, la bandera nunca se desactiva, y la sustitución se realizará en cada línea hasta el final del archivo.

Debe saber que si se encuentra el patrón `start`, la sustitución se realiza en la misma línea que contiene `start`. Esto activa un interruptor, que está orientado a la línea. Es decir, se lee la siguiente línea y se comprueba el comando de sustitución. Si contiene `stop`, el interruptor se desactiva. Los interruptores están orientados a la línea, y no a la palabra.

Puede combinar números de línea y expresiones regulares. Este ejemplo eliminará los comentarios del principio del archivo hasta que encuentre la palabra clave `start`

```
sed -e '1,/start/ s/#.*//'
```

Este ejemplo eliminará los comentarios en todas partes excepto en las líneas entre las dos palabras clave:

```
sed -e '1,/start/ s/#.*//' -e '/stop/, $ s/#.*//'
```

El último ejemplo tiene un rango que se solapa con el rango `/start/,/stop/`, ya que ambos rangos operan sobre las líneas que contienen las palabras clave. Más adelante le mostraré cómo restringir un comando hasta, pero sin incluir, la línea que contiene el patrón especificado. Está en Operando en un rango de patrones excepto los patrones Pero tengo que cubrir algunos principios más básicos.

Antes de empezar a hablar de los distintos comandos, debo explicar que algunos comandos no pueden operar en un rango de líneas. Se lo haré saber cuando mencione los comandos. En la siguiente sección describiré tres comandos, uno de los cuales no puede operar en un rango.

Borrar con d

El uso de rangos puede ser confuso, por lo que debe esperar hacer alguna experimentación cuando esté probando un nuevo script. Un comando útil borra todas las líneas que coinciden con la restricción "d." Si quiere ver las primeras 10 líneas de un archivo, puede usar

```
sed '11,$ d' <file
```

cuyo funcionamiento es similar al del comando **head**. Si quiere cortar la cabecera de un mensaje de correo, que es todo lo que hay hasta la primera línea en blanco, utilice

```
sed '1,/^\$/ d' <file
```

Puede duplicar la función del comando **tail**, asumiendo que conoce la longitud de un archivo. **Wc** puede contar las líneas, y **expr** puede restar 10 del número de líneas. Un script de Bash para ver las últimas 10 líneas de un archivo podría ser así

```
#!/bin/bash
#print last 10 lines of file
# First argument is the filename
lines=$(wc -l "$1" | awk '{print $1}' )
start=$(( lines - 10))
sed "1,$start d" "$1"
```

Haga clic aquí para obtener el archivo: [sed_tail.sh](#)

El rango para las eliminaciones puede ser pares de expresiones regulares para marcar el inicio y el final de la operación. O puede ser una sola expresión regular. Borrar todas las líneas que comienzan con un **#** es fácil:


```
sed '/^#/ d'
```

Eliminar los comentarios y las líneas en blanco requiere dos comandos. El primero elimina todos los caracteres desde el **#** hasta el final de la línea, y el segundo borra todas las líneas en blanco:

```
sed -e 's/#.*//' -e '/^$/ d'
```


Hay que añadir una tercera para eliminar todos los espacios en blanco y los tabuladores inmediatamente antes del final de línea

```
sed -e 's/#.*//' -e 's/[ ^I]*$//' -e '/^$/ d'
```

El carácter **^I** es el carácter **Ctrl** **i** o de tabulación (). Tendrías que escribir explícitamente el tabulador. Observe el orden de las operaciones anteriores, que está en ese orden por una muy buena razón. Los comentarios pueden empezar en medio de una línea, con caracteres de espacio en blanco antes de ellos. Por lo tanto, los comentarios se eliminan primero de una línea, dejando potencialmente los caracteres de espacio en blanco que estaban antes del comentario. El segundo comando elimina todos los espacios en blanco al final, de modo que las líneas que ahora están en blanco se convierten en líneas vacías. El último comando elimina las líneas vacías. Juntos, los tres comandos eliminan todas las líneas que sólo contienen comentarios, tabulaciones o espacios.

Esto demuestra el patrón de espacio que sed utiliza para operar en una línea. La operación real que utiliza sed es

- Copiar la línea de entrada en el espacio patrón.
- Aplicar el primer
- sed en el espacio patrón, si la restricción de dirección es verdadera.
- Repita con la siguiente expresión sed, de nuevo
- operando en el espacio patrón.
- Cuando se realice la última operación, escriba el espacio patrón
- y lea la siguiente línea del archivo de entrada.

Imprimir con p

Otro comando útil es el comando de impresión: **p**. Si sed no se inició con la opción **-n**, el comando **p** duplicará la entrada. El comando

```
sed 'p'
```

duplicará cada línea. Si quiere duplicar cada línea vacía, utilice

```
sed '/^$/ p'
```

Añadiendo la opción `-n` se desactiva la impresión a menos que lo solicite. Otra forma de duplicar la funcionalidad de `head` es imprimir sólo las líneas que desee. Este ejemplo imprime las 10 primeras líneas:

```
sed -n '1,10 p' <file
```

Sed puede actuar como grep combinando el operador de impresión para funcionar en todas las líneas que coincidan con una expresión regular:

```
sed -n '/match/ p'
```

que es lo mismo que

```
grep match
```

Invirtiendo la restricción con **!**

A veces es necesario realizar una acción en todas las líneas excepto en las que coinciden con una expresión regular, o en las que están fuera de un rango de direcciones. El carácter **!**, que suele significar *no* en las utilidades UNIX, invierte la restricción de direcciones. Recuerde que

```
sed -n '/match/ p'
```

actúa como el comando grep. La opción `-v` de grep imprime todas las líneas que no contienen el patrón. Sed puede hacer esto con

```
sed -n '/match/ !p' </tmp/b
```

La relación entre **d**, **p** y **!**

Como habrá notado, a menudo hay varias formas de resolver el mismo problema con sed. Esto se debe a que `print` y `delete` son funciones opuestas, y parece que `!p` es similar a `d`, mientras que `!d` es similar a `p`. Quería probar esto, así que creé un archivo de prueba de 20

líneas, y probé cada combinación diferente. La siguiente tabla, que muestra los resultados de cada prueba, demuestra la diferencia:

sed	Rango	Comando	Resultado
sed -n	1,10	p	Imprimir las 10 primeras líneas
sed -n	11,\$!p	Imprimir las 10 primeras líneas
sed	1,10	!d	Imprimir las 10 primeras líneas
sed	11,\$	d	Imprimir las 10 primeras líneas

sed	Rango	Comando	Resultado
sed -n	1,10	!p	Imprimir las 10 últimas líneas de mi archivo de 20 líneas
sed -n	11,\$	p	Imprimir las 10 últimas líneas de mi archivo de 20 líneas
sed	1,10	d	Imprimir las 10 últimas líneas de mi archivo de 20 líneas
sed	11,\$!d	Imprimir las 10 últimas líneas de mi archivo de 20 líneas

sed	Rango	Comando	Resultado
sed -n	1,10	d	No se imprime nada
sed -n	1,10	!d	No se imprime nada
sed -n	11,\$	d	No se imprime nada
sed -n	11,\$!d	No se imprime nada

sed	Rango	Comando	Resultado
sed	1,10	p	Imprimir las primeras 10 líneas dos veces, luego las siguientes 10 líneas una vez
sed	11,\$!p	Imprimir las primeras 10 líneas dos veces, luego las últimas 10 líneas una vez
sed	Rango	Comando	Resultado
sed	1,10	!p	Imprimir las primeras 10 líneas una vez, luego las últimas 10 líneas dos veces
sed	11,\$	p	Imprimir las primeras 10 líneas una vez, luego las últimas 10 líneas dos veces

Esta tabla muestra que utilizando mi archivo de prueba de 20 líneas, los siguientes comandos son idénticos:

```
sed -n '1,10 p'
sed -n '11,$ !p'
sed '1,10 !d'
sed '11,$ d'
```

También muestra que el comando **!** "invierte" el rango de direcciones, operando en las otras líneas.

Por supuesto, para archivos de más de 20 líneas, obtendrá más de 10 líneas para los dos últimos casos.

El comando q o quit

Hay un comando más simple que puede restringir los cambios a un conjunto de líneas. Es el comando **q**: *quit*. la tercera forma de duplicar el comando **head** es

```
sed '11 q'
```

que se cierra cuando se alcanza la undécima línea. Este comando es muy útil cuando se desea abortar la edición después de alcanzar alguna condición.

El comando "q" es el único que no toma un rango de direcciones. Obviamente, el comando

```
sed '1,10 q'
```

no puede salir 10 veces. En su lugar

```
sed '1 q'
```

0

```
sed '10 q'
```

es correcto.

Agrupando con { y }

Las llaves, { y }, se utilizan para agrupar los comandos.

Apenas merece la pena la acumulación. Toda esa prosa y la solución no es más que hacer coincidir los garabatos. Bueno, hay una complicación. Como cada comando sed debe comenzar en su propia línea, las llaves y los comandos sed anidados deben estar en líneas separadas.

Anteriormente, le mostré cómo eliminar los comentarios que comienzan con un "#". Si quieres restringir la eliminación a las líneas entre las palabras clave especiales "begin" y "end", puedes usar

```
#!/bin/sh
# Este es un script del shell Bourne que elimina los comentarios de tipo #
# entre las palabras "begin" y "end".
sed -n '
    /begin/, /end/ {
        s/#.*//
        s/[ ^I]*$//
        /^$/ d
        p
    }
'
```

Haga clic aquí para obtener el archivo: [sed_begin_end.sh](#)

Estos corchetes pueden anidarse, lo que permite combinar rangos de direcciones. Podría realizar la misma acción que antes, pero limitando el cambio a las primeras 100 líneas:

```
#!/bin/sh
# Este es un script Bash que elimina los comentarios # de tipo
# entre las palabras "comienzo" y "final".
sed -n '
    1,100 {
        /begin/,/end/ {
            s/#.*//
            s/[ ^I]*$//
            /^$/ d
            p
        }
    }
'
```

Haga clic aquí para obtener el archivo: [sed_begin_end1.sh](#)

Puede colocar un **!** antes de un conjunto de llaves. Esto invierte la dirección, lo que elimina los comentarios de todas las líneas excepto las que están entre las dos palabras reservadas:

```
#!/bin/sh
sed '
    /begin/,/end/ !{
        s/#.*//
        s/[ ^I]*$//
        /^$/ d
        p
    }
'
```

Haga clic aquí para obtener el archivo: [sed_begin_end2.sh](#)

Operating in a pattern range except for the patterns

Tal vez recuerdes que mencioné que puedes hacer una sustitución en un rango de patrones, como cambiar "old" por "new" entre un patrón de begin/end:

```
#!/bin/sh
sed '
    /begin/,/end/ s/old/new/
'
```

Otra forma de escribir esto es utilizar las llaves para agrupar:

```
#!/bin/sh
sed '
    /begin/,/end/ {
        s/old/new/
    }
'
```

Creo que esto hace que el código sea más claro de entender, y más fácil de modificar, como verás a continuación.

Si no quiere hacer ningún cambio en el lugar donde se encuentra la palabra "begin", puede simplemente añadir una nueva condición para saltarse esa línea:

```
#!/bin/sh
sed '
    /begin/,/end/ {
        /begin/n # skip over the line that has "begin" on it
        s/old/new/
    }
'
```

Sin embargo, saltarse la línea que tiene end es más complicado. Si utiliza el mismo método que utilizó para begin, el motor sed no verá el end para detener el rango - también se lo salta. La solución es hacer una sustitución en todas las líneas que no tienen el end usando

```
#!/bin/sh
sed '
    /begin/,/end/ {
        /begin/n # skip over the line that has "begin" on it
        /end/ !{
            s/old/new/
        }
    }
'
```

Escribir un archivo con el comando 'w'

Puede recordar que el comando sustituto puede escribir en un archivo. Aquí está de nuevo el ejemplo que sólo escribirá las líneas que comienzan con un número par [y seguido de un espacio]:

```
sed -n 's/^[0-9]*[02468] /&/w even' <file
```

Utilicé el `&` en la parte de reemplazo del comando de sustitución para que la línea no se modificara. Un ejemplo más sencillo es utilizar el comando `w`, que tiene la misma sintaxis que la bandera `w` en el comando de sustitución:

```
sed -n '/^[0-9]*[02468]/ w even' <file
```

Recuerde: sólo un espacio debe seguir al comando. Cualquier otra cosa se considerará parte del nombre del archivo. El comando `w` también tiene la misma limitación que la bandera `w`: sólo se pueden abrir 10 archivos en sed.

Reading in a file with the 'r' command

También hay un comando para leer archivos. El comando

```
sed '$r end' <in>out
```

añadirá el archivo `end` al final del archivo [dirección `$`]. Lo siguiente insertará un archivo después de la línea con la palabra `"INCLUDE:"`:

```
sed '/INCLUDE/ r file' <in >out
```

Puede utilizar las llaves para eliminar la línea que tiene el comando `INCLUDE:`:

```
#!/bin/sh
sed '/INCLUDE/ {
    r archivo
    d
}'
```

Haga clic aquí para obtener el archivo: [sed_include.sh](#)

El orden del comando de borrado `d` y del comando de lectura de archivos `r` es importante. Cambia el orden y no funcionará. Hay dos acciones sutiles que impiden que esto funcione. La primera es que el comando `r` escribe el archivo en el flujo de salida. El archivo no se inserta en

el espacio del patrón, y por lo tanto no puede ser modificado por ningún comando. Por lo tanto, el comando de borrado no afecta a los datos leídos del archivo.

La otra sutileza es que el comando `d` borra los datos actuales en el espacio del patrón. Una vez que se borran todos los datos, tiene sentido que no se intente ninguna otra acción. Por lo tanto, un comando `d` ejecutado en una llave rizada también aborta todas las acciones posteriores. Como ejemplo, el comando sustituto de abajo nunca se ejecuta:

```
#!/bin/sh
# este ejemplo es INCORRECTO
sed -e '1 {
    d
    s/.*/ /
}'
```

Haga clic aquí para obtener el archivo: [sed_bad_example.sh](#)

El ejemplo anterior es una versión rudimentaria del programa preprocesador C. El archivo que se incluye tiene un nombre predeterminado. Estaría bien que sed permitiera una variable (por ejemplo, `"\1"`) en lugar de un nombre de archivo fijo. Por desgracia, sed no tiene esta capacidad. Puede evitar esta limitación creando comandos sed sobre la marcha, o utilizando comillas de shell para pasar variables al script sed. Suponga que quiere crear un comando que incluya un archivo como `cpp`, pero el nombre del archivo es un argumento del script. Un ejemplo de este script es:

```
% include 'sys/param.h' <file.c >file.c.new
```

Un script de shell para hacer esto sería:

```
#!/bin/sh
# watch out for a '/' in the parameter
# use alternate search delimiter
sed -e '\_#INCLUDE <"$1">_{
    r "$1"
    d
}'
```

Permítanme que me explaye. Si usted tiene un archivo que contiene

```
Test first file
```

```
#INCLUDE <file1>
Test second file
#INCLUDE <file2>
```

puede utilizar el comando

```
sed_include1.sh file1<input|sed_include1.sh file2
```

para incluir los archivos especificados.

Haga clic aquí para obtener el archivo: [sed_include1.sh](#)

El comando **#** comentario

A medida que profundizamos en sed, los comentarios harán que los comandos sean más fáciles de seguir. Las versiones más antiguas de sed sólo permiten una línea como comentario, y debe ser la primera. SunOS [y sed de GNU] permite más de un comentario, y estos comentarios no tienen que ser los primeros. El último ejemplo podría ser:

```
#!/bin/sh
# watch out for a '/' in the parameter
# use alternate search delimiter
sed -e '\_#INCLUDE <'"$1"'>_{

    # read the file
    r '"$1"'

    # delete any characters in the pattern space
    # and read the next line in
    d
}'
```

Haga clic aquí para obtener el archivo: [sed_include2.sh](#)

Añadir, modificar, insertar nuevas líneas

Sed tiene tres comandos utilizados para añadir nuevas líneas al flujo de salida. Como se añade una línea entera, la nueva línea está en una línea por sí misma para enfatizar esto. No hay

opción, se usa una línea entera, y debe estar en su propia línea. Si está familiarizado con muchas utilidades de UNIX, esperaría que sed utilizara una convención similar: las líneas se continúan terminando la línea anterior con un `\`. La sintaxis de estos comandos es delicada, como la de los comandos `r` y `w`.

Añadir una línea con 'a'

El comando "a" añade una línea después del rango o patrón. Este ejemplo añadirá una línea después de cada línea con "WORD:"

```
#!/bin/sh
sed '
/WORD/ a\
Add this line after every line with WORD
'
```

Haga clic aquí para obtener el archivo: [sed_add_line_after_word.sh](#)

Si lo desea, puede eliminar dos líneas en el script de shell:

```
#!/bin/sh
sed '/WORD/ a\
Add this line after every line with WORD'
```

Haga clic aquí para obtener el archivo: [sed_add_line_after_word1.sh](#)

Prefiero la primera forma porque es más fácil añadir un nuevo comando añadiendo una nueva línea y porque la intención es más clara. No debe haber un espacio después de `\`.

Insertar una línea con 'i'

Puede insertar una nueva línea antes del patrón con el comando "i":

```
#!/bin/sh
sed '
/WORD/ i\
Add this line before every line with WORD
'
```

Haga clic aquí para obtener el archivo: [sed_add_line_before_word.sh](#)

Cambiar una línea con c

Puede cambiar la línea actual por una nueva línea.

```
#!/bin/sh
sed '
/WORD/ c\
Replace the current line with the line
'
```

Haga clic aquí para obtener el archivo: [sed_change_line.sh](#)

Un comando **d** seguido de un comando **a** no funcionará, como he comentado antes. El comando **d** terminaría las acciones actuales. Puedes combinar las tres acciones usando llaves:

```
#!/bin/sh
sed '
/WORD/ {
i\
Add this line before
a\
Add this line after
c\
Change the line to this one
}'
```

Haga clic aquí para obtener el archivo: [sed_insert_append_change.sh](#)

Leading tabs and spaces in a sed script

Sed ignora los tabuladores y espacios iniciales en todos los comandos. Sin embargo, estos caracteres de espacio en blanco pueden o no ser ignorados si comienzan el texto después de un comando **"a"**, **"c"** o **"i"**. En SunOS, ambas "características" están disponibles. El sed estilo Berkeley (y Linux) está en `/usr/bin`, y la versión AT&T (System V) está en `/usr/5bin/`.

Para explicarlo mejor, el comando `/usr/bin/sed` conserva los espacios en blanco, mientras que `/usr/5bin/sed` elimina los espacios iniciales. Si quiere mantener los espacios a la izquierda, y no le importa la versión de sed que esté utilizando, ponga un **"\N"** como primer carácter de la línea:

```
#!/bin/sh
```

```
sed '
    a\
\    This line starts with a tab
'
```

Agregando mas de una linea

Los tres comandos le permitirán añadir más de una línea. Sólo tiene que terminar cada línea con un "\"

```
#!/bin/sh
sed '
/WORD/ a\
Add this line\
This line\
And this line
'
```

Añadir líneas y el espacio de patrones

Ya he mencionado antes el espacio de patrones. La mayoría de los comandos operan en el espacio patrón, y los comandos posteriores pueden actuar sobre los resultados de la última modificación. Los tres comandos anteriores, al igual que el comando de lectura de archivos, añaden las nuevas líneas al flujo de salida, saltándose el espacio de patrones.

Rangos de direcciones y los comandos anteriores

Tal vez recuerdes que antes te advertí que algunos comandos pueden tomar un rango de líneas, y otros no. Para ser precisos, los comandos "a", "i", "r" y "q" no tomarán un rango como "1,100" o "/begin/,/end/". La documentación indica que el comando "read" puede tomar un rango, pero obtuve un error cuando lo intenté. El comando "c" o **change** lo permite, y te permitirá cambiar varias líneas en una sola:

```
#!/bin/sh
sed '
/begin/,/end/ c\
***DELETED***
'
```

Si necesita hacer esto, puede utilizar las llaves, ya que eso le permitirá realizar la operación en cada línea:

```
#!/bin/sh
# add a blank line after every line
sed '1,$ {
    a\

}'
```

Patrones de varias líneas

La mayoría de las utilidades de UNIX están orientadas a la línea. Las expresiones regulares están orientadas a la línea. Buscar patrones que abarquen más de una línea no es una tarea fácil. [Pista: lo será muy pronto].

Sed lee una línea de texto, ejecuta comandos que pueden modificar la línea y emite la modificación si se desea. El bucle principal de un script sed tiene el siguiente aspecto:

- Se lee la siguiente línea del archivo de entrada y se coloca en el espacio de patrones. Si se encuentra el final del archivo, y si hay archivos adicionales para leer, se cierra el archivo actual, se abre el siguiente archivo y se coloca la primera línea del nuevo archivo en el espacio del patrón.
- El número de líneas se incrementa en uno. La apertura de un nuevo archivo no restablece este número.
- Cada comando sed es examinado. Si hay una restricción en el comando, y la línea actual en el espacio del patrón cumple con esa restricción, el comando se ejecuta. Algunos comandos, como "n" o "d" hacen que sed vaya al principio del bucle. El comando "q" hace que sed se detenga. En caso contrario, se examina el siguiente comando.
- Una vez examinados todos los comandos, se emite el espacio del patrón a menos que sed tenga el argumento opcional "-n".

La restricción que precede al comando determina si éste se ejecuta. Si la restricción es un patrón, y la operación es el comando de borrado, entonces lo siguiente borrará todas las líneas que tengan el patrón:

```
/PATTERN/ d
```

Si la restricción es un par de números, la eliminación se producirá si el número de la línea es igual al primer número o mayor que el primer número y menor o igual que el último:

```
10,20 d
```

Si la restricción es un par de patrones, hay una variable que se mantiene para cada uno de estos pares. Si la variable es falsa y se encuentra el primer patrón, la variable se convierte en verdadera. Si la variable es verdadera, se ejecuta el comando. Si la variable es verdadera, y el último patrón está en la línea, después de ejecutar el comando la variable se apaga:

```
/begin/,/end/ d
```

¡Uf! Eso fue un bocado. Si has leído con atención hasta aquí, deberías haber pasado rápido por esto. Quizá quiera volver a leerlo, porque he tratado varios puntos sutiles. Mi elección de palabras fue deliberada. Cubre algunos casos inusuales, como:

```
# qué pasa si el segundo número
# es menor que el primer número?
sed -n '20,1 p' file
```

y

```
# generate a 10 line file with line numbers
# and see what happens when two patterns overlap
yes | head -10 | cat -n | \
sed -n -e '/1/,/7/ p' -e '/5/,/9/ p'
```

Suficiente castigo mental. He aquí otra revisión, esta vez en formato de tabla. Supongamos que el archivo de entrada contiene las siguientes líneas:

```
AB
CD
EF
GH
IJ
```

Cuando el sed se pone en marcha, la primera línea se coloca en el espacio del patrón. La siguiente línea es "CD". Las operaciones de los comandos "n", "d" y "p" pueden resumirse como

Espacio de los	Siguiente	Comando	Salida	Nuevo espacio del	Nueva entrada de
----------------	-----------	---------	--------	-------------------	------------------

patrones	entrada			patrón	texto
AB	CD	n	<por defecto>	CD	EF
AB	CD	d	-	CD	EF
AB	CD	p	AB	CD	EF

El comando "n" puede o no generar salida dependiendo de la existencia de la bandera "-n".

Esta revisión es un poco más fácil de seguir, ¿no? Antes de saltar a los patrones multilínea, quería cubrir tres comandos más:

Imprimir el número de línea con =

El comando = imprime el número de línea actual en la salida estándar. Una forma de averiguar los números de línea que contienen un patrón es utilizar:

```
# add line numbers first,
# then use grep,
# then just print the number
cat -n file | grep 'PATTERN' | awk '{print $1}'
```

La solución sed es:

```
sed -n '/PATTERN/ =' file
```

Anteriormente utilicé lo siguiente para encontrar el número de líneas en un archivo

```
#!/bin/sh
lines=$(wc -l file | awk '{print $1}' )
```

El uso del comando = puede simplificar esto:

```
#!/bin/sh
lines=$(sed -n '$=' file )
```

El comando = sólo acepta una dirección, por lo que si desea imprimir el número de un rango de líneas, debe utilizar las llaves:


```
#!/bin/sh
# Just print the line numbers
sed -n '/begin/,/end/ {
=
d
}' file
```

Como el comando `=` sólo imprime en la salida estándar, no puede imprimir el número de línea en la misma línea que el patrón. Para ello, es necesario editar los patrones de varias líneas.

Transformar con `y`

Si quiere cambiar una palabra de minúsculas a mayúsculas, podría escribir 26 sustituciones de caracteres, convirtiendo "a" en "A", etc. Sed tiene un comando que opera como el programa `tr`. Se llama comando "`y`". Por ejemplo, para cambiar las letras "a" a "f" a su forma mayúscula, utilice

```
sed 'y/abcdef/ABCDEF/' file
```

Este es un ejemplo de sed que convierte todas las letras mayúsculas en minúsculas, como el comando `tr`:

```
sed 'y/ABCDEFGHIJKLMNOPQRSTUVWXYZ/abcdefghijklmnopqrstuvwxyz/' <uppercase
>lowercase
```

Si quieres convertir una línea que contenga un número hexadecimal (por ejemplo, `0x1aff`) a mayúsculas (`0x1AFF`), puedes utilizar

```
sed '/0x[0-9a-zA-Z]*/ y/abcdef/ABCDEF' file
```

Esto funciona bien si sólo hay números en el archivo. Si quieres cambiar la segunda palabra de una línea a mayúsculas, y estás usando sed clásico, no tienes suerte - a menos que uses la edición multilínea. ¡Eh, creo que hay algún tipo de tema aquí!

Sin embargo, GNU sed tiene una extensión de mayúsculas y minúsculas.

Mostrar caracteres de control con **l**

El comando "**l**" imprime el espacio del patrón actual. Por lo tanto, es útil para depurar los scripts de sed. También convierte los caracteres no imprimibles en caracteres imprimibles, emitiendo el valor en octal precedido por un carácter "****". Lo encontré útil para imprimir el espacio de patrones actual, mientras sondeaba las sutilezas de sed.

Trabajar con varias líneas

Hay tres nuevos comandos utilizados en los patrones de líneas múltiples: "**N**", "**D**" y "**P**". Explicaré su relación con los comandos de línea única "**n**", "**d**" y "**p**" que coinciden.

El comando "**n**" imprimirá el espacio de patrones actual (a menos que se utilice la bandera "**-n**"), vaciará el espacio de patrones actual y leerá la siguiente línea de entrada. El comando "**N**" no imprime el espacio de patrones actual y no vacía el espacio de patrones. Lee la siguiente línea, pero añade un nuevo carácter de línea junto con la propia línea de entrada al espacio del patrón.

El comando "**d**" borra el espacio del patrón actual, lee la siguiente línea, coloca la nueva línea en el espacio del patrón, y aborta el comando actual, y comienza la ejecución en el primer comando sed. Esto se llama iniciar un nuevo "ciclo". El comando "**D**" borra la primera porción del espacio del patrón, hasta el carácter de la nueva línea, dejando el resto del patrón solo. Al igual que "**d**", detiene el comando actual y comienza el ciclo de comandos de nuevo. Sin embargo, no imprimirá el espacio del patrón actual. Debe imprimirlo usted mismo, un paso antes. Si el comando "**D**" se ejecuta con un grupo de otros comandos en una llave rizada, los comandos después del comando "**D**" se ignoran. Se ejecuta el siguiente grupo de comandos sed, a menos que se vacíe el espacio del patrón. Si esto ocurre, el ciclo se inicia desde el principio y se lee una nueva línea.

El comando "**p**" imprime todo el espacio del patrón. El comando "**P**" sólo imprime la primera parte del espacio del patrón, hasta el carácter **NEWLINE**. Ni el comando "**p**" ni el "**P**" cambian el espacio de patrones.

Algunos ejemplos podrían demostrar que "**N**" por sí solo no es muy útil. el filtro

```
sed -e 'N'
```

no modifica el flujo de entrada. En cambio, combina la primera y la segunda línea, luego las imprime, combina la tercera y la cuarta línea, y las imprime, etc. Le permite utilizar un nuevo carácter de "anclaje": "\n". Esto coincide con el nuevo carácter de línea que separa varias líneas en el espacio del patrón. Si quiere buscar una línea que termine con el carácter "#" y añadirle la siguiente línea, puede utilizar

```
#!/bin/sh
sed '
# look for a "#" at the end of the line
/##$/ {
# Found one - now read in the next line
    N
# delete the "#" and the new line character,
    s/##\n//
}' file
```

Podría buscar dos líneas que contengan "ONE" y "TWO" y sólo imprimir las dos líneas consecutivas:

```
#!/bin/sh
sed -n '
/ONE/ {
# found "ONE" - read in next line
    N
# look for "TWO" on the second line
# and print if there.
    /\n.*TWO/ p
}' file
```

El siguiente ejemplo eliminaría todo lo que hay entre "ONE" y "TWO":

```
#!/bin/sh
sed '
/ONE/ {
# append a line
    N
# search for TWO on the second line
    /\n.*TWO/ {
# found it - now edit making one line
        s/ONE.*\n.*TWO/ONE TWO/
    }
}'
```

```
} ' file
```

Hacer coincidir tres líneas con sed

Puede hacer coincidir varias líneas en las búsquedas.

Esta es una forma de buscar la cadena "skip3", y si se encuentra, borrar esa línea y las dos siguientes.

```
#!/bin/sh
sed '/skip3/ {
    N
    N
    s/skip3\n.*\n.*/# 3 lines deleted/
}'
```

Tenga en cuenta que no importa cuáles son las dos líneas siguientes. Si quisieras hacer coincidir 3 líneas en particular, es un poco más de trabajo.

Este script busca tres líneas, donde la primera línea contenga "one", la segunda contenga "two" y la tercera contenga "three", y si las encuentra, las sustituye por la cadena "1+2+3":

```
#!/bin/sh
sed '
/one/ {
    N
    /two/ {
        N
        /three/ {
            N
            s/one\ntwo\nthree/1+2+3/
        }
    }
}
```

Comparación de patrones que abarcan varias líneas

Puede buscar un patrón concreto en dos líneas consecutivas, o bien puede buscar dos palabras consecutivas que pueden estar divididas en un límite de línea. El siguiente ejemplo buscará dos palabras que estén en la misma línea o que una esté al final de una línea y la segunda al principio de la siguiente. Si se encuentran, se elimina la primera palabra:

```
#!/bin/sh
sed '
/ONE/ {
# append a line
    N
# "ONE TWO" on same line
    s/ONE TWO/TWO/
# "ONE
# TWO" on two consecutive lines
    s/ONE\nTWO/TWO/
}' file
```

Utilicemos el comando **"D"**, y si encontramos una línea que contenga **"ONE"** inmediatamente después de una línea que contenga **"TWO"**, entonces borremos la primera línea:

```
#!/bin/sh
sed '
/ONE/ {
# append a line
    N
# if TWO found, delete the first line
    /\n.*TWO/ D
}' file
```

Haga clic aquí para obtener el archivo: [sed_delete_line_after_word.sh](#)

Si quisiéramos imprimir la primera línea en lugar de borrarla, y no imprimir cada dos líneas, cambie la **"D"** por una **"P"** y añada un **"-n"** como argumento a sed:

```
#!/bin/sh
sed -n '
# by default - do not print anything
/ONE/ {
# append a line
```

```
        N
# if TWO found, print the first line
    /\n.*TWO/ P
}' file
```

Haga clic aquí para obtener el archivo: [sed_print_line_after_word.sh](#)

Es muy común combinar los tres comandos de varias líneas. El orden típico es "N", "P" y por último "D". Éste borrará todo lo que haya entre "UNO" y "DOS" si están en una o dos líneas consecutivas:

```
#!/bin/sh
sed '
/ONE/ {
# append the next line
    N
# look for "ONE" followed by "TWO"
    /ONE.*TWO/ {
#       delete everything between
        s/ONE.*TWO/ONE TWO/
#       print
        P
#       then delete the first line
        D
    }
}' file
```

Haga clic aquí para obtener el archivo: [sed_delete_between_two_words.sh](#)

Anteriormente hablé del comando "=", y de su uso para añadir números de línea a un archivo. Puede usar dos invocaciones de sed para hacer esto (aunque es posible hacerlo con una, pero eso debe esperar hasta la próxima sección). La primera orden de sed mostrará un número de línea en una línea, y luego imprimirá la línea en la siguiente. La segunda invocación de sed unirá las dos líneas:

```
#!/bin/sh
sed '=' file | \
sed '{
    N
    s/\n/ /
}'
```

Haga clic aquí para obtener el archivo: [sed_merge_two_lines.sh](#)

Si lo considera necesario, puede dividir una línea en dos, editarlas y volver a unir las. Por ejemplo, si tiene un archivo con un número hexadecimal seguido de una palabra, y quiere convertir la primera palabra en mayúsculas, puede utilizar el comando "y", pero primero debe dividir la línea en dos, cambiar una de las dos y unir las. Es decir, una línea que contenga

```
0x1fff table2
```

se convertirá en dos líneas:

```
0x1fff  
tabla2
```

y la primera línea se convertirá en mayúsculas. Usaré `tr` para convertir el espacio en una nueva línea, y luego usaré `sed` para hacer el resto. El comando sería

```
./sed_split <file
```

and `sed_split` would be:

```
#!/bin/sh  
tr ' ' '\012' |  
sed ' {  
    y/abcdef/ABCDEF/  
    N  
    s/\n/ /  
}'
```

Haga clic aquí para obtener el archivo: [sed_split.sh](#)

No es obvio, pero se puede usar `sed` en lugar de `tr`. Puede incrustar una nueva línea en un comando de sustitución, pero debe escapar con una barra invertida. Es desafortunado que deba usar "\n" en el lado izquierdo de un comando sustituto, y una nueva línea incrustada en el lado derecho. Suspiro pesado. Aquí está el ejemplo:

```
#!/bin/sh  
sed '  
s/ /\n  
' | \  
sed ' {  
    y/abcdef/ABCDEF/  
    N
```

```
s/\n/ /
{'
```

Haga clic aquí para obtener el archivo: [sed_split_merge.sh](#)

A veces añado un carácter especial como marcador, y busco ese carácter en el flujo de entrada. Cuando se encuentra, indica el lugar donde solía estar un espacio en blanco. Una barra invertida es un buen carácter, excepto que debe ser escapado con una barra invertida, y hace que el script sed sea oscuro. Guárdelo para ese tipo que sigue haciendo preguntas tontas. El script sed para cambiar un espacio en blanco por un "\N" seguido de una nueva línea sería

```
#!/bin/sh
sed 's/ /\n\
/' file
```

Haga clic aquí para obtener el archivo: [sed_addslash_before_blank.sh](#)

Sí. Ese es el ticket. O usa el shell C y confúndelo de verdad.

```
#!/bin/csh -f
sed '\
s/ /\n\
/' file
```

Haga clic aquí para obtener el archivo: [sed_addslash_before_blank.csh](#)

Un par de ejemplos más de eso, y nunca más te hará una pregunta. Creo que me estoy dejando llevar. Voy a resumir con un gráfico que cubre las características de las que hemos hablado:

Pattern Space	Siguiente Input	Comando	Salida	New Pattern Space	Nueva entrada
AB	CD	n	<default>	CD	EF
AB	CD	N	-	AB\nCD	EF
AB	CD	d	-	-	EF
AB	CD	D	-	-	EF
AB	CD	p	AB	AB	CD
AB	CD	P	AB	AB	CD

AB\nCD	EF	n	<default>	EF	GH
AB\nCD	EF	N	-	AB\nCD\nEF	GH
AB\nCD	EF	d	-	EF	GH
AB\nCD	EF	D	-	CD	EF
AB\nCD	EF	p	AB\nCD	AB\nCD	EF
AB\nCD	EF	P	AB	AB\nCD	EF

Uso de nuevas líneas en los scripts sed

Ocasionalmente uno desea utilizar un nuevo carácter de línea en un script sed. Esto tiene algunos problemas sutiles. Si uno quiere buscar una nueva línea, tiene que usar "\n". Aquí hay un ejemplo en el que se busca una frase, y se elimina el carácter de nueva línea después de esa frase - uniendo dos líneas.

```
(echo a;echo x;echo y) | sed '/x$/ {
N
s:x\n:x:
}'
```

que genera

```
a
xy
```

Sin embargo, si va a insertar una nueva línea, no utilice "\n", sino que inserte un carácter literal de nueva línea:

```
(echo a;echo x;echo y) | sed 's:x:X\
:'
```

genera

```
a
X

y
```

El búfer de retención

Hasta ahora hemos hablado de tres conceptos de sed:

- Primero.** El flujo de entrada o datos antes de ser modificados,
- Segundo.** El flujo de salida o datos después de ser modificados, y
- Tercero.** El espacio de patrones, o buffer que contiene caracteres que pueden ser modificados y enviados al flujo de salida.

Hay una "ubicación" más que cubrir: el búfer de retención o espacio de retención. Piensa en él como un buffer de patrones de repuesto. Se puede utilizar para "copiar" o "recordar" los datos en el espacio de patrones para más tarde. Hay cinco comandos que utilizan el buffer de retención.

Intercambio con x

El comando "x" intercambia el espacio del patrón con el buffer de retención. Por sí mismo, el comando no es útil. Ejecutar el comando sed

```
sed 'x'
```

como filtro añade una línea en blanco al principio y borra la última línea. Parece que no cambió el flujo de entrada significativamente, pero el comando sed está modificando cada línea.

El búfer de retención comienza conteniendo una línea en blanco. Cuando el comando "x" modifica la primera línea, la línea 1 se guarda en el búfer de retención, y la línea en blanco ocupa el lugar de la primera línea. El segundo comando "x" intercambia la segunda línea con el búfer de retención, que contiene la primera línea. Cada línea posterior se intercambia con la línea anterior. La última línea se coloca en el búfer de retención, y no se intercambia una segunda vez, por lo que permanece en el búfer de retención cuando el programa termina, y nunca se imprime. Esto ilustra que hay que tener cuidado cuando se almacenan datos en el búfer de retención, porque no se imprimirán a menos que se solicite explícitamente.

Ejemplo de Grep Contextual

Un uso del búfer de retención es recordar las líneas anteriores. Un ejemplo de esto es una utilidad que actúa como grep, ya que le muestra las líneas que coinciden con un patrón. Además, le muestra la línea anterior y posterior al patrón. Es decir, si la línea 8 contiene el patrón, esta utilidad imprimiría las líneas 7, 8 y 9.

Una forma de hacerlo es ver si la línea tiene el patrón. Si no tiene el patrón, ponga la línea actual en el búfer de retención. Si lo tiene, imprime la línea en el búfer de retención, luego la línea actual, y luego la siguiente línea. Después de cada conjunto, se imprimen tres guiones. El script comprueba la existencia de un argumento, y si falta, imprime un error. El paso del argumento al script sed se realiza desactivando el mecanismo de comillas simples, insertando el "\$1" en el script, y volviendo a poner en marcha las comillas simples:

```
#!/bin/sh
# grep3 - prints out three lines around pattern
# if there is only one argument, exit

case $# in
    1);;
    *) echo "Usage: $0 pattern";exit;;
esac;
# I hope the argument doesn't contain a /
# if it does, sed will complain

# use sed -n to disable printing
# unless we ask for it
sed -n '
"/"$1"/' !{
    #no match - put the current line in the hold buffer
    x
    # delete the old one, which is
    # now in the pattern buffer
    d
}
"/"$1"/' {
    # a match - get last line
    x
    # print it
    p
    # get the original line back
    x
}
```

```
# print it
p
# get the next line
n
# print it
p
# now add three dashes as a marker
a\
---
# now put this line into the hold buffer
x
}'
```

Haga clic aquí para obtener el archivo: [grep3.sh](#)

Puede utilizarlo para mostrar las tres líneas que rodean a una palabra clave, por ejemplo:

```
grep3 vt100 </etc/termcap
```

Retención con **h** o **H**

El comando "x" intercambia el búfer de retención y el búfer de patrón. Ambos se cambian. El comando "h" copia el búfer de patrones en el búfer de retención. La memoria intermedia del patrón no cambia. Un script idéntico al anterior utiliza los comandos de retención:

```
#!/bin/sh
# grep3 version b - another version using the hold commands
# if there is only one argument, exit

case $# in
  1);;
  *) echo "Usage: $0 pattern";exit;;
esac;

# again - I hope the argument doesn't contain a /

# use sed -n to disable printing

sed -n '
  /"$1"/ !{
    # put the non-matching line in the hold buffer
    h
```

```
}
'/"$1"/' {
    # found a line that matches
    # append it to the hold buffer
    H
    # the hold buffer contains 2 lines
    # get the next line
    n
    # and add it to the hold buffer
    H
    # now print it back to the pattern space
    x
    # and print it.
    p
    # add the three hyphens as a marker
    a\
---
}'
```

Click here to get file: [grep3a.sh](#)

Mantener más de una línea en el buffer de retención

El comando "H" permite combinar varias líneas en el buffer de retención. Actúa como el comando "N", ya que las líneas se añaden al búfer, con una "\n" entre las líneas. Puede guardar varias líneas en el búfer de retención, e imprimirlas sólo si se encuentra un patrón particular más tarde.

Como ejemplo, tomemos un archivo que utiliza espacios como primer carácter de una línea como carácter de continuación. Los ficheros `/etc/termcap`, `/etc/printcap`, `makefile` y los mensajes de correo utilizan espacios o tabulaciones para indicar la continuación de una entrada. Si quisieras imprimir la entrada antes de una palabra, podrías usar este script. Yo uso un "^I" para indicar un carácter de tabulación real:

```
#!/bin/sh
# print previous entry
sed -n '
/^[ ^I]/!{
    # line does not start with a space or tab,
    # does it have the pattern we are interested in?
```

```

        '/'"$1"/' {
            # yes it does. print three dashes
            i\
---
            # get hold buffer, save current line
            x
            # now print what was in the hold buffer
            p
            # get the original line back
            x
        }
        # store it in the hold buffer
        h
    }
    # what about lines that start
    # with a space or tab?
    /^[ ^I]/ {
        # append it to the hold buffer
        H
    }
}'

```

Haga clic aquí para obtener el archivo: [grep_previous.sh](#)

También puede utilizar la "H" para ampliar el contexto grep. En este ejemplo, el programa imprime las dos líneas anteriores al patrón, en lugar de una sola línea. El método para limitarlo a dos líneas es utilizar el comando "s" para mantener una nueva línea, y eliminar las líneas adicionales. Lo llamo **grep4**:

```

#!/bin/sh

# grep4: prints out 4 lines around pattern
# if there is only one argument, exit

case $# in
    1);;
    *) echo "Usage: $0 pattern";exit;;
esac;

sed -n '
/'"$1"/' !{
    # does not match - add this line to the hold space
    H
    # bring it back into the pattern space
    x
    # Two lines would look like .*\\n.*
}

```

```
# Three lines look like .*\\n.*\\n.*
# Delete extra lines - keep two
s/^..*\\n\\(.*\\n.*\\)$\\1/
# now put the two lines (at most) into
# the hold buffer again
x
}
'/"$1"/' {
    # matches - append the current line
    H
    # get the next line
    n
    # append that one also
    H
    # bring it back, but keep the current line in
    # the hold buffer. This is the line after the pattern,
    # and we want to place it in hold in case the next line
    # has the desired pattern
    x
    # print the 4 lines
    p
    # add the mark
    a\\
---
}'
```

Haga clic aquí para obtener el archivo: [grep4](#).

shPuede modificar esto para imprimir cualquier número de líneas alrededor de un patrón. Como puede ver, debe recordar lo que está en el espacio de retención, y lo que está en el espacio del patrón. Hay otras formas de escribir la misma rutina.

Obtener con **g** o **G**

En lugar de intercambiar el espacio de retención con el espacio del patrón, puede copiar el espacio de retención al espacio del patrón con el comando "**g**". Esto borra el espacio del patrón. Si quiere añadir algo al espacio del patrón, utilice el comando "**G**". Esto añade una nueva línea al espacio del patrón, y copia el espacio de retención después de la nueva línea.

Aquí hay otra versión del comando "**grep3**". Funciona igual que el anterior, pero está implementado de forma diferente. Esto ilustra que sed tiene más de una forma de resolver muchos problemas. Lo importante es que entiendas tu problema y documentes tu solución:

```
#!/bin/sh
# grep3 version c: use 'G' instead of H

# if there is only one argument, exit

case $# in
    1);;
    *) echo "Usage: $0 pattern";exit;;
esac;

# again - I hope the argument doesn't contain a /

sed -n '
"/"$1"/' !{
    # put the non-matching line in the hold buffer
    h
}
"/"$1"/' {
    # found a line that matches
    # add the next line to the pattern space
    N
    # exchange the previous line with the
    # 2 in pattern space
    x
    # now add the two lines back
    G
    # and print it.
    p
    # add the three hyphens as a marker
    a\
---
    # remove first 2 lines
    s/.*\n.*\n\(.*\)$/\1/
    # and place in the hold buffer for next time
    h
}'
```

Haga clic aquí para obtener el archivo: [grep3c.sh](#)

El comando "G" facilita tener dos copias de una línea. Supongamos que quiere convertir el primer número hexadecimal en mayúsculas, y no quiere usar el [sed_split.sh](#) script que describí antes. Ese script sólo funciona cuando hay exactamente 2 palabras por línea. Si quieres permitir más de una palabra en una línea y sólo convertir la primera palabra hexadecimal a mayúsculas, entonces este es un mejor enfoque:


```
#!/bin/sh
# change the first hex number to upper case format, leave the rest of the
line alone
# uses sed twice
# used as a filter
# convert2uc <in >out
sed '
s/ /\
/' | \
sed ' {
    y/abcdef/ABCDEF/
    N
    s/\n/ /
}'
```

Haga clic aquí para obtener el archivo: [convert2uc.sh](#)

He aquí una solución que no requiere dos invocaciones de sed porque utiliza el comando "h" y "G":

```
#!/bin/sh
# convert2uc version b
# change the first hex number to upper case format, leave the rest of the
line alone
# uses sed once
# used as a filter
# convert2uc <in >out
sed '
{
    # remember the line
    h
    #change the current line to upper case
    y/abcdef/ABCDEF/
    # add the old line back
    G
    # Keep the first word of the first line,
    # and second word of the second line
    # with one humongous regular expression
    s/^\([^ ]*\) .* \n\([^ ]*\) \(.*)/\1 \2/
}'
```

Haga clic aquí para obtener el archivo: [convert2uc1.shCarl](#)

Henrik Lunde sugirió una manera de hacer esto más simple, pero no como propósito general. Las dos versiones anteriores convertidas trabajan con líneas con múltiples palabras. Esta sólo convierte dos palabras - la primera y la última. Sin embargo, borra cualquier palabra entre ellas.

```
#!/bin/sh
# convert2uc version b
# change the first hex number to upper case format, and keeps the last word
# Note that it deletes the words in-between
# uses sed once
# used as a filter
# convert2uc <in >out
sed '
{
    # remember the line
    h
    #change the current line to upper case
    y/abcdef/ABCDEF/
    # add the old line back
    G
    # Keep the first word of the first line,
    # and last word of the second line
    # with one humongous regular expression
    s/ .* / / # delete all but the first and last word
}'
```

Haga clic aquí para obtener el archivo: [convert2uc2.sh](#) Este

ejemplo sólo convierte las letras "a" a "f" en mayúsculas. Esto se eligió para que el script fuera más fácil de imprimir en estas columnas estrechas. Puede modificar fácilmente el script para convertir todas las letras a mayúsculas, o para cambiar la primera letra, la segunda palabra, etc.

Decisiones (control de flujo)

A medida que vas aprendiendo sobre sed te das cuenta de que tiene su propio lenguaje de programación. Es cierto que es un lenguaje muy especializado y sencillo. ¿Qué lenguaje estaría completo sin un método para cambiar el control de flujo? Hay tres comandos que sed utiliza para esto. Puedes especificar una etiqueta con una cadena de texto precedida por dos puntos. El comando "b" se ramifica hacia la etiqueta. La etiqueta sigue al comando. Si no hay etiqueta, se bifurca al final del script. El comando "t" se utiliza para probar condiciones. Antes de discutir el comando "t", le mostraré un ejemplo usando el comando "b".

Este ejemplo recuerda los párrafos, y si contiene el patrón (especificado por un argumento), el script imprime el párrafo completo.

```
#!/bin/sh
sed -n '
# if an empty line, check the paragraph
/^$/ b para
# else add it to the hold buffer
H
# at end of file, check paragraph
$ b para
# now branch to end of script
b
# this is where a paragraph is checked for the pattern
:para
# return the entire paragraph
# into the pattern space
x
# look for the pattern, if there - print
/'"$1"'/ p
'
```

Haga clic aquí para obtener el archivo: [grep_paragraph.sh](#)

Probar con t

Puede ejecutar una rama si se encuentra un patrón. Es posible que desee ejecutar una rama sólo si se realiza una sustitución. El comando "**t etiqueta**" se bifurcará a la etiqueta si el último comando de sustitución modificó el espacio del patrón.

Un uso de esto son los patrones recursivos. Suponga que quiere eliminar los espacios en blanco dentro de los paréntesis. Estos paréntesis podrían estar anidados. Es decir, querrías eliminar una cadena que se pareciera a "**(((())))**". Las expresiones **sed**

```
sed 's/([ ^I]*)/g'
```

sólo eliminaría el conjunto más interno. Tendría que pasar los datos por el script cuatro veces para eliminar cada conjunto o paréntesis. Podría utilizar la expresión regular

```
sed 's/([ ^I()]*)/g'
```

pero eso eliminaría los conjuntos de paréntesis no coincidentes. El comando "**t**" lo solucionaría:

```
#!/bin/sh
sed '
:again
    s/([ ^I]*)//
    t again
'
```

Una versión anterior tenía una "g" después de la expresión "s". Esto no es necesario.

Haga clic aquí para obtener el archivo: [delete_nested_parens.sh](#)

Debugging con 1

El comando '1' imprimirá el espacio del patrón de forma inequívoca. Los caracteres que no se imprimen se imprimen en un formato escapado al estilo C.

Esto puede ser útil al depurar un complejo script `sed` de varias líneas.

Una forma alternativa de añadir comentarios

Hay una forma de añadir comentarios en un script `sed` si no tienes una versión que lo soporte. Utilice el comando "a" con el número de línea de cero:

```
#!/bin/sh
sed '
/begin/ {
0i\
    This is a comment\
    It can cover several lines\
    It will work with any version of sed
}'
```

Haga clic aquí para obtener el archivo: [sed_add_comments.sh](#)

El pobremente documentado ;

Hay otro comando `sed` que no está bien documentado. Es el comando `;`. Se puede utilizar para combinar varios comandos `sed` en una línea. Aquí está el script `grep4` que describí antes, pero sin los comentarios o la comprobación de errores y con punto y coma entre los comandos:

```
#!/bin/sh
sed -n '
'/"$1"/' !{;H;x;s/^.*\n\(.*\n.*\)$/\1/;x;}
'/"$1"/' {;H;n;H;x;p;a\
---
}'
```

Haga clic aquí para obtener el archivo: [grep4a.sh](#)

¡Yessireebob! Definitivamente, la construcción del carácter. Creo que he dejado claro mi punto de vista. En lo que a mí respecta, el único momento en que el punto y coma es útil es cuando quieres escribir el script `sed` en la línea de comandos. Si vas a ponerlo en un script, dale formato para que sea legible. He mencionado antes que muchas versiones de `sed` no admiten comentarios excepto en la primera línea. Puede que quiera escribir sus scripts con comentarios en ellos, e instalarlos en forma "binaria" sin comentarios. Esto no debería ser difícil. Después de todo, ya te has convertido en un gurú de `sed`. Ni siquiera te diré cómo escribir un script para eliminar los comentarios. Eso sería un insulto a su inteligencia. Además, algunos sistemas operativos NO permiten el uso del punto y coma. Así que si ves un script con punto y coma, y no funciona en un sistema que no sea Linux, sustituye el punto y coma por un carácter de nueva línea. [Siempre y cuando no estés usando `csh`/`tcsh`, pero ese es otro tema.

Pasar expresiones regulares como argumentos

En los scripts anteriores, mencioné que tendrías problemas si pasabas un argumento al script que tuviera una barra. De hecho, la expresión regular podría causarle problemas. Un script como el siguiente está pidiendo que se rompa algún día:

```
#!/bin/sh
```

```
sed 's/\'"$1"\'//g'
```

Si el argumento contiene alguno de estos caracteres, puede obtener un script roto:

`"/\.*[]^$"` Por ejemplo, si alguien escribe una `/`, el comando de sustitución verá cuatro delimitadores en lugar de tres. También obtendrá errores de sintaxis si proporciona un `]` sin un `[`. Una solución es hacer que el usuario ponga una barra invertida antes de cualquiera de estos caracteres cuando lo pase como argumento. Sin embargo, el usuario tiene que saber qué caracteres son especiales.

He aquí otra solución: añadir una barra invertida antes de cada uno de esos caracteres especiales en el script.

```
#!/bin/sh
# put two backslashes before each of these characters: ][^$.*/
# Note that the first '[' doesn't need a backslash
arg=$(echo "$1" | sed 's:[][^\$\\.\\*\\/]:\\\\&:g')
# We need two backslashes because the shell converts each double backslash in
quotes to a single backslash
sed 's/\'"$arg"\'//g'
```

Haga clic aquí para obtener el archivo: [sed_with_regular_expressions1.sh](#) Si

estuviera buscando el patrón `^.. /`, el script lo convertiría en `\\^.\\ /` antes de pasarlo a `sed`.

Inserción de caracteres binarios

Tratar con caracteres binarios puede ser complicado, especialmente cuando se escriben scripts para que la gente los lea. Puedo insertar un carácter binario usando un editor como EMACS pero si muestro el carácter binario, el terminal puede cambiarlo para mostrarlo.

La forma más fácil que he encontrado para hacer esto en un script de forma portátil es utilizar el comando `tr(1)`. Entiende las notaciones octales, y puede ser la salida en una variable que se puede utilizar.

Este es un script que sustituirá la cadena "ding" por el carácter ASCII de la campana:

```
#!/bin/sh
BELL=$(echo x | tr 'x' '\\007')
```

```
sed "s/ding/$BELL/"
```

Tenga en cuenta que he utilizado comillas dobles. Como los caracteres especiales se interpretan, hay que tener cuidado cuando se utiliza este mecanismo.

Argumentos de la línea de comandos de GNU sed

Una de las convenciones de los sistemas UNIX es utilizar letras simples como argumentos en la línea de comandos. Esto hace que el tecleo sea más rápido, y más corto, lo que es una ventaja si estás en un concurso. La gente normal suele encontrar críptica la tersura de sed. Puede mejorar la legibilidad de los scripts de sed utilizando las opciones de equivalencia de palabras largas. Es decir, en lugar de escribir

```
sed -n 20p
```

Puede escribir la versión larga del argumento -n

```
sed --quiet 20p
```

O

```
sed --silent 20p
```

La forma larga de los argumentos de la línea de comandos de sed siempre tiene 2 guiones antes de sus nombres. GNU **sed** tiene los siguientes argumentos de línea de comandos de forma larga:

Argumentos de la línea de comandos de GNU

Forma corta

Forma larga

-n

--quiet

--silent

-e script

--expression=SCRIPT

-f SCRIPTFILE

--file=SCRIPTFILE

-i[SUFFIX]

--in-place[=SUFFIX]

<code>-l N</code>	<code>--line-length=N</code>
	<code>--posix</code>
<code>-b</code>	<code>--binary</code>
	<code>--follow-symlinks</code>
<code>-r</code>	<code>--regular-extended</code>
<code>-s</code>	<code>--separate</code>
<code>-u</code>	<code>--unbuffered</code>
	<code>--help</code>
	<code>--version</code>

Definamos cada una de ellas.

El argumento `-posix`

La versión GNU de sed tiene muchas características que no están disponibles en otras versiones. Cuando la portabilidad es importante, pruebe su script con la opción `-posix`. Si tuviera un ejemplo que utilizara una característica de GNU sed, como el comando `'v'` para probar el número de versión, como

```
#this is a sed command file
v 4.0.1
# print the number of lines
$=
```

Y lo ejecutó con el comando

```
sed -nf sedfile --posix <file
```

entonces la versión GNU del programa sed le daría una advertencia de que su script sed no es compatible. Informaría:

```
sed: -e expression #1, char 2: unknown command: `v'
```

El argumento `--version`

Puede determinar qué versión de sed está utilizando con el comando GNU `sed --version`. Esto es lo que muestra en mi ordenador


```
# sed --version
GNU sed version 4.2.1
Copyright (C) 2009 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE,
to the extent permitted by law.

GNU sed home page: <https://www.gnu.org/software/sed/>.
General help using GNU software: <https://www.gnu.org/gethelp/>.
E-mail bug reports to: <bug-gnu-utils@gnu.org>.
Be sure to include the word ``sed`` somewhere in the ``Subject:`` field.
Asegúrese de incluir la palabra ``sed`` en algún lugar del campo ``Asunto:``.
```

El argumento para ayuda -h

La opción **-h** imprimirá un resumen de los comandos sed. El argumento largo del comando es

```
sed --help
```

Proporciona un buen resumen de los argumentos de la línea de comandos.

El argumento longitud de línea -l

Ya he descrito el comando **l**. El ancho de línea por defecto del comando **l** es de 70 caracteres. Este valor por defecto se puede cambiar añadiendo la opción **-l N** y especificando la longitud máxima de la línea como el número que aparece después de **-l**.

```
sed -n -l 80 'l' <file
```

La versión larga de la línea de comandos es

```
sed -n --line-length=80 'l' <file
```

El argumento separado -s

Normalmente, cuando se especifican varios archivos en la línea de comandos, sed concatena los archivos en un solo flujo, y luego opera en ese único flujo. Si tuviera tres archivos, cada uno con 100 líneas, entonces el comando

```
sed -n '1,10 p' file1 file2 file3
```

sólo imprimiría las primeras 10 líneas del archivo `file1`. El comando `-s` le dice a GNU `sed` que trate los archivos como archivos independientes, y que imprima las primeras 10 líneas de cada archivo, lo que es similar al comando `head`. He aquí otro ejemplo: Si quieres imprimir el número de líneas de cada archivo, puedes usar `wc -l` que imprime el número de líneas, y el nombre del archivo, para cada archivo, y al final imprime el número total de líneas. Aquí hay un simple script de shell que hace algo similar, sólo que usando `sed`:

```
#!/bin/sh
FILES=$*
sed -s -n '$=' $FILES # print the number of lines for each file
sed -n '$=' $FILES # print the total number of lines.
```

El comando `wc -l` sí imprime los nombres de los archivos, a diferencia del script anterior. Una mejor emulación del comando `wc -l` ejecutaría el comando en un bucle, e imprimiría los nombres de los archivos. Aquí hay un script más avanzado que hace esto, pero no utiliza el comando `-s`:

```
#!/bin/sh
for F in "$@"
do
    NL=$(sed -n '$=' < "$F" ) && printf " %d %s\n" $NL "$F"
done
TOTAL=$(sed -n '$=' "$@")
printf " %d total\n" $TOTAL
```

El argumento en el lugar `-i`

Ya he descrito en Editar múltiples archivos la forma en que me gusta hacer esto. Para aquellos que quieren un método más simple, GNU Sed le permite hacer esto con una opción de línea de comandos `-i`. Supongamos que vamos a hacer el mismo cambio simple - añadir un tabulador antes de cada línea. Esta es una manera de hacer esto para todos los archivos en un directorio con la extensión `.txt` en el directorio actual:

```
sed -i 's/^\t/' *.txt
```

La versión del nombre del argumento largo es

```
sed --in-place 's/^\t/' *.txt
```

Esta versión elimina el archivo original. Si eres tan precavido como yo, tal vez prefieras especificar una extensión, que se utiliza para conservar una copia del original:

```
sed -i.tmp 's/^\t/' *.txt
```

Y la versión del nombre del argumento largo es

```
sed --in-place=.tmp 's/^\t/' *.txt
```

En las dos últimas versiones, la versión original del archivo "**a.txt**" tendría el nombre "**a.txt.tmp**". Entonces puedes borrar los archivos originales después de asegurarte de que todo ha funcionado como esperabas. Por favor, considera la opción de la copia de seguridad, y presta atención a mi advertencia. Puedes eliminar fácilmente el archivo original respaldado, siempre que la extensión sea única.

La versión GNU de sed permite utilizar "-i" sin argumento. La de FreeBSD/Mac OS X no lo hace. **Debe** proporcionar una extensión para la versión de FreeBSD/Mac OS X. Si quiere hacer una edición en el lugar sin crear una copia de seguridad, puede usar

```
sed -i '' 's/^\t/' *.txt
```

El argumento seguir enlaces **--follow-symlinks**

La función de edición en el lugar es muy útil. Pero, ¿qué sucede si el archivo que está editando es un enlace simbólico a otro archivo? Supongamos que tienes un archivo llamado "**b**" en un directorio llamado "**tmp**", con un enlace simbólico a este archivo:

```
$ ls -l b
lrwxrwxrwx 1 barnett adm 6 Mar 16 16:03 b.txt -> tmp/b.txt
```

Si ejecutó el comando anterior para hacer una edición en el lugar, habrá un nuevo archivo llamado "**b.txt**" en el directorio actual, y "**tmp/b.txt**" no será modificado. Ahora tiene dos versiones del archivo, una cambiada (en el directorio actual), y otra no (en el directorio "**tmp**"). Y donde tenías un enlace simbólico, ha sido reemplazado por una versión modificada del archivo

original. Si quieres editar el archivo real, y mantener el enlace simbólico en su lugar, utiliza la opción de línea de comandos `--follow-symlinks`:

```
sed -i --follow-symlinks 's/^\t/' *.txt
```

Esto sigue el enlace simbólico a la ubicación original, y modifica el archivo en el directorio `"tmp"`. Si usted especifica una extensión, el archivo original se encontrará con esa extensión en el mismo directorio que la fuente real. Sin la opción de línea de comandos `--follow-symlinks`, el archivo de "copia de seguridad" `"b.tmp"` estará en el mismo directorio que contenía el enlace simbólico, y seguirá siendo un enlace simbólico - sólo se le cambiará el nombre para darle una nueva extensión.

El argumento binario `-b`

Los sistemas Unix y Linux consideran que el carácter de nueva línea `"\n"` es el final de la línea. Sin embargo, los sistemas MS-DOS, Windows y Cygwin terminan cada línea con `"\r\n"` - Retorno de carro y salto de línea. Si está utilizando alguno de estos sistemas operativos, la opción de línea de comandos `-b` o `--binary` tratará la combinación de retorno de carro/salto de línea como el final de la línea. De lo contrario, el retorno de carro se trata como un carácter no imprimible inmediatamente antes del final de línea. Creo que. [Nota para mí: verifícalo].

El argumento expresión regular extendida `-r`

Cuando menciono patrones, como `"s/patrón/"`, el patrón es una expresión regular. Hay dos clases comunes de expresiones regulares, las expresiones "básicas" originales y las expresiones regulares "extendidas". Para más información sobre las diferencias, consulte Mi tutorial sobre expresiones regulares y la sección sobre expresiones regulares extendidas. Debido a que el significado de ciertos caracteres es diferente entre las expresiones regulares y las extendidas, se necesita un argumento en la línea de comandos para que `sed` pueda utilizar la extensión. Para habilitar esta extensión, utilice el comando `-r`, como se menciona en el ejemplo sobre la búsqueda de palabras duplicadas en una línea

```
sed -r -n '/\[a-z]+\)/ \1/p'
```

0

```
sed --regular-extended -quiet '/\[a-z]+\)' \1/p'
```

Ya he mencionado que Mac OS X y FreeBSD utilizan `-E` en lugar de `-r`.

El argumento sin búfer `-u`

Normalmente, los sistemas Unix y Linux aplican cierta inteligencia en el manejo de la salida estándar. Se asume que si estás enviando resultados a una terminal, quieres la salida tan pronto como esté disponible. Sin embargo, si estás enviando la salida a un archivo, se asume que quieres un mejor rendimiento, por lo que se almacena la salida en el búfer hasta que éste se llena, y entonces el contenido del búfer se escribe en el archivo. Permítanme explicar esto. Supongamos para este ejemplo que tiene un archivo muy grande, y que está usando sed para buscar una cadena, e imprimirla cuando la encuentre:

```
sed -n '/MATCH/p' <file
```

Como la salida es el terminal, en cuanto se encuentra una coincidencia, se imprime. Sin embargo, si sed canaliza su salida a otro programa, almacenará los resultados en un búfer. Pero hay ocasiones en las que se quieren resultados inmediatos. Esto es especialmente cierto cuando se trata de archivos grandes, o archivos que ocasionalmente generan datos. Para resumir, tienes muchos datos de entrada, y quieres que Sed los procese, y luego envíe esto a otro programa que procese los resultados, pero quieres los resultados cuando ocurran, y no con retraso. Permítame inventar un ejemplo sencillo. Es artificioso, pero explica cómo funciona esto. Aquí hay un programa llamado `SlowText` que imprime números del 1 al 60, una vez por segundo:

```
#!/bin/sh
for i in $(seq 1 60)
do
    echo $i
    sleep 1
done
```

Usemos sed para buscar líneas que tengan el carácter `'1'`, y hagamos que envíe los resultados a awk, que calculará el cuadrado de ese número. Este sería el script, ciertamente artificioso:

```
SlowText | sed -n '/1/p' | awk '{print $1*$1}'
```

Esto funciona, pero como sed almacena los resultados en el búfer, tenemos que esperar hasta que el búfer se llene, o hasta que el programa `SlowText` exista, antes de ver los resultados. Puede eliminar el almacenamiento en búfer, y ver los resultados tan pronto como `SlowText` los imprima, utilizando la opción "`-u`". Con esta opción, verá los cuadrados impresos tan pronto como sea posible:

```
SlowText | sed -un '/1/p' | awk '{print $1*$1}'
```

La forma larga del argumento es "`--unbuffered`".

Mac OS X y FreeBSD utilizan el argumento "`-l`".

GNU Sed 4.2.2 y posteriores también se desbufarán al leer archivos, no sólo al escribirlos.

El argumento datos nulos -z

Normalmente, sed lee una línea leyendo una cadena de caracteres hasta el carácter de fin de línea (nueva línea o retorno de carro). Vea el argumento de línea de comandos `-b` Binario La versión GNU de sed añadió una función en la versión 4.2.2 para utilizar el carácter "`NULL`" en su lugar. Esto puede ser útil si tiene archivos que utilizan el `NULL` como separador de registros. Algunas utilidades de GNU pueden generar una salida que utiliza un `NULL` en lugar de una nueva línea, como "`find . -print0`" o "`grep -lZ`". Esta característica es útil si está operando con nombres de archivos que pueden contener espacios o caracteres binarios.

Por ejemplo, si quieres usar "`find`" para buscar archivos y usas la opción "`-print0`" para imprimir un `NULL` al final de cada nombre de archivo, puedes usar sed para borrar la ruta del directorio:

```
find . -type f -print0 | sed -z 's:^.*/::' | xargs -0 echo
```

El ejemplo anterior no es terriblemente útil, ya que el uso de "`xargs`" de echo no conserva la capacidad de retener espacios como parte del nombre del archivo. Pero sí muestra cómo utilizar el comando sed "`-z`".

GNU grep también tiene una opción `-Z` para buscar cadenas en archivos, colocando un "`NULL`" al final de cada nombre de archivo en lugar de una nueva línea. Y con el comando `-l, grep`

imprimirá el nombre de archivo que contiene la cadena, conservando los caracteres no imprimibles y binarios:

```
grep -lZ STRING */** | sed -z 's:^.*/::' | xargs -0 echo
```

Esta función es muy útil cuando los usuarios tienen la posibilidad de crear sus propios nombres de archivo.

Extensiones FreeBSD

Apple utiliza la versión FreeBSD de sed para Mac OS X en lugar de la GNU sed. Sin embargo, la versión de FreeBSD tiene un par de adiciones.

El argumento para apertura diferida -a

Normalmente, tan pronto como sed se inicia, abre todos los ficheros a los que hace referencia la orden "w". La versión de FreeBSD de sed tiene una opción para retrasar esta acción hasta que se ejecute la orden "w".

El argumento para en el lugar -I

FreeBSD ha añadido una opción "-I" que es similar a la opción -i. La opción "-i" trata la edición de cada archivo como una instancia separada de sed. Si se utiliza la opción "-I", los números de línea no se reinician al principio de cada línea, y los rangos de direcciones continúan de un fichero al siguiente. Es decir, si se utiliza el rango '/BEGIN/, /END/' y se usa la opción "-I", se puede tener el "BEGIN" en el primer archivo, y el "END" en el segundo, y los comandos ejecutados dentro del rango abarcarían ambos archivos. Si utilizas "-i", los comandos no lo harán.

Y al igual que la opción -i, se debe especificar la extensión utilizada para almacenar el archivo de copia de seguridad.

El argumento para expresiones regulares extendidas -E

Anteriormente mencioné las expresiones regulares extendidas. FreeBSD (y Mac OS X) utiliza "-E" para activar esto. Sin embargo, FreeBSD añadió posteriormente el comando "-r" para ser compatible con GNU sed.

Utilizando delimitadores de palabra

Una vez, alguien me pidió que le ayudara a resolver un complicado problema de sedes que implicaba límites de palabras. Supongamos que tiene la siguiente entrada

```
/usr/bin /usr/local/bin /usr/local /usr/local/project/bin
```

y querías borrar '/usr/local' pero dejar las otras 3 rutas solas. Podrías usar el simple (e incorrecto) comando:

```
sed 's@/usr/local@'
```

lo que daría como resultado

```
/usr/bin /bin /usr/local /usr/local/project/bin
```

Es decir, cambiaría erróneamente '/usr/local/bin' por '/bin' y no borraría '/usr/local' que era la intención del programador. El mejor método es incluir espacios alrededor de la búsqueda:

```
sed 's@ /usr/local @ @'
```

Sin embargo, esto no funcionará si '/usr/local' está al principio o al final de la línea. Tampoco funcionará si '/usr/local' es la única ruta en la línea. Para manejar estos casos extremos, puedes simplemente describir todas estas condiciones como casos separados:

```
#!/bin/sh
sed '
s@ /usr/local @ @g
s@^/usr/local @@
s@ /usr/local$@@
```



```
s@^/usr/local$@@
'
```

Esto funciona bien si la cadena que se busca está rodeada por un espacio. ¿Pero qué ocurre si la cadena está rodeada por otros caracteres, que pueden ser uno de los varios posibles? Por ejemplo, si la cadena está formada por caracteres alfanuméricos y la barra oblicua, la clase de caracteres puede definirse como '[a-zA-Z0-9/]' o la más flexible '[:alnum:]/'. Podemos definir la clase de caracteres para que sean todos menos éstos, utilizando el signo de intercalación, es decir, '[^[:alnum:]/]'. Y a diferencia del espacio anterior, si vamos a utilizar clases de caracteres, es posible que tengamos que recordar cuáles son estos caracteres y no borrarlos. Así que podemos sustituir el espacio por '[^[:alnum:]/]' y luego cambiar el comando para que sea

```
#!/bin/sh
sed '
s@\[([[:alnum:]]/)\)/usr/local\[([[:alnum:]]/)\)@1\2@g
s@^/usr/local\[([[:alnum:]]/)\)@1@
s@\[([[:alnum:]]/)\)/usr/local$@1@
s@^/usr/local$@@
'
```

La primera versión reemplazaría '/usr/local' con un solo espacio. Este método reemplazaría ':usr/local:' con '::' - porque los delimitadores redundantes no se eliminan. Asegúrese de arreglar esto si lo necesita.

Este método siempre funciona, pero es poco elegante y propenso a errores. Hay otros métodos, pero **pueden no ser portables**. La versión de Solaris de sed utilizaba los caracteres especiales '\<' y '\>' como anclas que indicaban un límite de palabra. Así que podría usar

```
s@\</usr/local\>@@
```

Sin embargo, la versión GNU de sed dice que el uso de estos caracteres especiales no está definido. Según la página del manual:

```
Regex syntax clashes (problems with backslashes)
'sed' uses the POSIX basic regular expression syntax. According to
the standard, the meaning of some escape sequences is undefined in
this syntax; notable in the case of 'sed' are '\|', '\+', '\?',
'\'', '\'', '\<', '\>', '\b', '\B', '\w', and '\W'.
```

As in all GNU programs that use POSIX basic regular expressions, 'sed' interprets these escape sequences as special characters. So, 'x\+' matches one or more occurrences of 'x'. 'abc\|def' matches either 'abc' or 'def'.

En caso de duda, experimenta.

Resumen de comandos

Como prometí antes, aquí hay una tabla que resume los diferentes comandos. La segunda columna especifica si el comando puede tener un rango o par de direcciones o una sola dirección o patrón. Las siguientes cuatro columnas especifican cuál de los cuatro buffers o flujos es modificado por el comando. Algunos comandos sólo afectan al flujo de salida, otros sólo afectan al buffer de retención. Si recuerda que el espacio del patrón es de salida (a menos que se le haya dado un "-n" a `sed`), esta tabla debería ayudarle a seguir la pista de los distintos comandos.

Comando	Dirección o rango	Modificación de la corriente de entrada	Modificación del flujo de salida	Modificación del espacio del patrón	Modificación del buffer de retención
=	-	-	Y	-	-
a	Dirección	-	Y	-	-
b	Rango	-	-	-	-
c	Rango	-	Y	-	-
d	Rango	Y	-	Y	-
D	Rango	Y	-	Y	-
g	Rango	-	-	Y	-
G	Rango	-	-	Y	-
h	Rango	-	-	-	Y
H	Rango	-	-	-	Y
i	Dirección	-	Y	-	-
l	Dirección	-	Y	-	-
n	Rango	Y	*	-	-

N	Rango	Y	-	Y	-
p	Rango	-	Y	-	-
P	Rango	-	Y	-	-
q	Dirección	-	-	-	-
r	Dirección	-	Y	-	-
s	Rango	-	-	Y	-
t	Rango	-	-	-	-
w	Rango	-	Y	-	-
x	Rango	-	-	Y	Y
y	Rango	-	-	Y	-

El comando "n" puede o no generar salida, dependiendo de la opción "-n". El comando "r" sólo puede tener una dirección, a pesar de la documentación.

Consulte mi tabla de [referencia de Sed](#)

En conclusión

Con esto concluye mi tutorial sobre sed. Es posible encontrar formas más cortas de algunos de mis scripts. Sin embargo, elegí estos ejemplos para ilustrar algunas construcciones básicas. Quería claridad, no oscuridad. Espero que lo hayas disfrutado.