EBook Gratis

APRENDIZAJE sed

Free unaffiliated eBook created from
Stack Overflow contributors.

#sed

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: sed

It is an unofficial and free sed ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official sed.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con sed

Observaciones

Referencias

- Página de manual de FreeBSD
- Página de manual de NetBSD
- Página de manual de OpenBSD
- Illumos sed man-page
- Página de manual de macOS (OS X)
- Página de manual de plan9
- Manual online de GNU sed

Versiones

Nombre	Versión inicial	Versión	Fecha de lanzamiento
Posix sed	1992	IEEE Std 1003.1, Edición 2013	2013-04-19
BSD sed	1992	FreeBSD 10.3 / NetBSD 7.0 / OpenBSD 5.9	2016-04-04
GNU sed	1989	4.2.2	2012-12-22

Examples

Hola Mundo

Uno de los usos más comunes de Sed es la sustitución de texto que se puede lograr con el comando `s` .

En una terminal, escriba `echo "Hello sed" | sed 's/sed/World/'` y presione `Entrar` :

```
$ echo "Hello sed" | sed 's/sed/World/'
Hello World
```

"Hello World" debe salir al terminal.

La cadena "Hello, sed" se envía a través de la tubería como entrada al comando sed que reemplaza la palabra `sed` con `World` .

La sintaxis de un comando básico de sustitución está `s` seguido de la cadena o patrón a ser buscado y el texto de sustitución. `s` comando `s` y las cadenas se separan con un delimitador `/` predeterminado.

Lea Empezando con sed en línea: https://riptutorial.com/es/sed/topic/934/empezando-con-sed

Capítulo 2: Agregar comando

Examples

Insertar línea después del primer partido

Dado un archivo file.txt con el siguiente contenido:

```
line 1
line 2
line 3
```

Puede añadir una nueva línea después de la primera línea coincidente con el `a` comando.

Para uso portátil del `a` comando debe ser seguida inmediatamente por una nueva línea escapado, con el texto-a-anexar en su propia línea o líneas.

```
sed '
/line 2/a\
new line 2.2
' file.txt
```

GNU sed

Algunas versiones de `sed` permiten que el texto-a-anexados para ser en línea con el `a` comando:

```
sed '/line 2/a new line 2.2' file.txt
```

Los comandos anteriores darán salida:

```
line 1
line 2
new line 2.2
line 3
```

Lea Agregar comando en línea: https://riptutorial.com/es/sed/topic/2835/agregar-comando

Capítulo 3: BSD / macOS Sed vs. GNU Sed vs. la especificación POSIX Sed

Introducción

Para citar la solicitud de creación de tema de @SnoringFrog:

"Uno de los errores más grandes que utilizan sed son los scripts que fallan (o tienen éxito de una manera inesperada) porque fueron escritos para uno y no para el otro. La reducción simple de las diferencias más importantes sería buena".

Observaciones

***macOS* usa la versión *BSD* de `sed` [1] , que difiere en muchos aspectos de la versión *GNU* `sed` que viene con *las* distribuciones de *Linux* .**

Su **denominador común** es la funcionalidad decretada por **POSIX** : consulte la especificación de POSIX `sed` .

El **enfoque más portátil** es **usar solo las características de POSIX** , lo que, sin embargo, **limita la funcionalidad** :

- En particular, POSIX especifica el **soporte solo para expresiones regulares *básicas*** , que tienen muchas limitaciones (por ejemplo, sin soporte para `|` (alternancia), sin soporte directo para `+` y `?`) Y diferentes requisitos de escape.
    - Advertencia: **GNU `sed` (sin `-r`), *soporta* `\|` , `\+` y `\?` , que NO es compatible con POSIX; use `--posix` para deshabilitar** (ver más abajo).
- **Para usar solo las características POSIX** :
    - (ambas versiones): use *solo* las opciones `-n -e` (en particular, no use `-E` o `-r` para activar el soporte para expresiones regulares *extendidas*)
    - GNU `sed` : añadir la opción `--posix` para asegurar la funcionalidad POSIX única (que no estrictamente necesario, pero sin ella podría terminar con inadvertidamente no cuenta con POSIX sin darse; *salvedad:* `--posix` *sí no* es compatible con POSIX)
    - El uso de las funciones exclusivas de POSIX implica requisitos de formato más estrictos (renunciando a muchas comodidades disponibles en GNU `sed`):
        - Secuencias de caracteres de control como `\n` `\t` generalmente NO son compatibles.
        - Las etiquetas y los comandos de bifurcación (por ejemplo, `b`) *deben* ir seguidos de una nueva línea *real* o continuación a través de una opción `-e` separada.
        - Vea a continuación para más detalles.

Sin embargo, *ambas* versiones implementan *extensiones* al estándar POSIX:

- ***Las* extensiones que implementan difieren** (GNU `sed` implementa más).
- incluso aquellas **extensiones que *ambos* implementan *difieren parcialmente en la sintaxis*** .

Si necesita soportar AMBAS plataformas (discusión de diferencias):

- Características **incompatibles** :
  - El uso de la **opción `-i` *sin* un argumento** (actualización in situ sin copia de seguridad) es incompatible:
    - BSD `sed` : DEBE utilizar `-i ''`
    - GNU `sed` : DEBE usar solo `-i` (equivalente: `-i''`) - usar `-i ''` NO funciona.
  - **`-i` sensiblemente activa la numeración de línea por *archivo* de *entrada*** en *GNU* `sed` y *las* versiones *recientes* de *BSD* `sed` (por ejemplo, en FreeBSD 10), pero **NO en macOS a partir de 10.12** .
    Tenga en cuenta que, en ausencia de `-i` *todas las* versiones numeran las líneas de forma *acumulativa* en los archivos de entrada.
  - Si la ***última* línea de entrada *no* tiene una nueva línea final** (y se imprime):
    - BSD `sed` : *siempre agrega una nueva línea* en la salida, incluso si la línea de entrada no termina en una.
    - GNU `sed` : *conserva el estado de nueva línea final* , es decir, agrega una nueva línea solo si la línea de entrada terminó en una.
- Características **comunes** :
  - Si restringe sus scripts `sed` a lo que soporta BSD `sed` , por lo general, también funcionarán en GNU `sed` , con la notable excepción de usar características de expresiones regulares *extendidas* específicas de la plataforma con `-E` . Obviamente, también renunciará a las extensiones que son específicas de la versión GNU. Ver la siguiente sección.

Pautas para el *soporte multiplataforma* (OS X / BSD, Linux), impulsadas por los requisitos más estrictos de la versión BSD :

Tenga en cuenta que las abreviaturas que *MacOS* y *Linux* se utilizan de vez en cuando a continuación para referirse a las versiones BSD y GNU de `sed` , respectivamente, debido a que son las versiones *de valores* en cada plataforma. Sin embargo, es posible instalar GNU `sed` en macOS, por ejemplo, utilizando Homebrew con `brew install gnu-sed` .

Nota : ***Excepto cuando se usan los indicadores `-r` y `-E`*** (expresiones regulares *extendidas*), las instrucciones a continuación equivalen a escribir scripts `sed` **compatibles con POSIX** .

- Para cumplir con POSIX, debe restringirse a POSIX BRE (expresiones regulares *básicas*) , que, desafortunadamente, como su nombre indica, son bastante básicas.

Advertencia : no asuma que `\|` , `\+` y `\?` son compatibles: mientras que GNU `sed` admite (a menos que se use `--posix`), BSD `sed` no lo hace - estas características *no* son compatibles con POSIX.
Mientras que **`\+` y `\?` *Se* puede emular** de manera compatible con POSIX:
`\{1,\}` para `\+` ,
`\{0,1\}` para `\?` ,
`\|` (alternancia) *no puede* , por desgracia.

- Para expresiones regulares más potentes, **use `-E`** (en lugar de `-r`) para admitir **ERE (expresiones regulares *extendidas*)** (GNU `sed` no documenta `-E` , pero funciona como un alias de `-r` ; versión *más* reciente de BSD `sed` , como en FreeBSD 10, ahora también es compatible con `-r` , pero la versión macOS a partir de 10.12 no *lo* hace).
Advertencia : aunque el uso de `-r` / `-E` significa que su comando *no es* compatible con POSIX por definición, aún debe **restringirse a los ERE de POSIX (expresiones regulares extendidas)** . Lamentablemente, esto significa que no podrá usar varias construcciones útiles, en particular:

    - aserciones de límite de palabra, porque son *específicas de la plataforma* (por ejemplo, `\<` en Linux, `[[:<]]` en OS X).
    - Las referencias *inversas dentro de las expresiones regulares* (a diferencia de las "referencias inversas" a las coincidencias del grupo de captura en la cadena de reemplazo de `s` llamadas de función `s`), porque BSD `sed` no las admite en expresiones regulares *extendidas* (pero, curiosamente, lo hace en *Los básicos* , donde están obligados por POSIX).

- **Secuencias de escape de caracteres de control como `\n` y `\t` :**

    - En las **expresiones regulares** (tanto en los patrones para la selección de líneas como en el primer argumento de la función `s`), suponga que solo `\n` se reconoce como una secuencia de escape (que se usa raramente, ya que el espacio del patrón suele ser una *sola* línea (sin terminar `\n`), pero no dentro de una *clase de caracteres* , por lo que, por ejemplo, `[^\n]` no funciona; (si su entrada no contiene caracteres de control. aparte de `\t` , puede emular `[^\n]` con `[[:print:][:blank:]]` ; de lo contrario, empalme los caracteres de control en como *literales* [2]) - **generalmente, incluye los caracteres de control como *literales* , ya sea a través de *cadenas citadas con ANSI C* empalmadas (por ejemplo, `$'\t'`) en shells que lo soportan (`bash`, ksh, `zsh`), o mediante *sustituciones de comandos usando* `printf` (por ejemplo, `"$(printf '\t')"`) .**

        - Sólo Linux:
        ```
        sed 's/\t/-/' <<<$'a\tb' # -> 'a-b'
        ```
        - OSX *y* Linux:
        ```
        sed 's/'$'\t''/-/' <<<$'a\tb' # ANSI C-quoted string
        sed 's/'"$(printf '\t')"'/-/' <<<$'a\tb' # command subst. with printf
        ```

    - En las **cadenas de reemplazo** utilizadas con el comando `s` , **suponga que NO se admiten las secuencias de escape de control-caracteres** , por lo que, nuevamente, incluyen caracteres de control. Como *literales* , como los anteriores.

        - Sólo Linux:
          ```
          sed 's/-/\t/' <<<$'ab' # -> 'a<tab>b'
          ```
        - macOS *y* Linux:
          ```
          sed 's/-/'$'\t''/' <<<'a-b'
          sed 's/-/'"$(printf '\t')"'/' <<<'a-b'
          ```
    - Lo mismo ocurre con los **argumentos de texto** a la `i` **y** `a` **funciones: no utilizar secuencias de control de caracteres** - véase más abajo.

- **Etiquetas y bifurcaciones** : las etiquetas, así como el *argumento de* nombre de etiqueta para las funciones `b` y `t` , **deben ir seguidas de una línea nueva *literal* o de** `$'\n'` **empalmados** . Alternativamente, use múltiples opciones `-e` y termine cada una justo después del nombre de la etiqueta.

    - Sólo Linux:
      ```
      sed -n '/a/ bLBL; d; :LBL p' <<<$'a\nb' # -> 'a'
      ```
    - macOS *y* Linux:
        - O (nuevas líneas reales):
          ```
          sed -n '/a/ bLBL d; :LBL p' <<<$'a\nb'
          ```
        - O (instancias empalmadas en `$\n`):
          ```
          sed -n '/a/ bLBL'$'\n''d; :LBL'$'\n''p' <<<$'a\nb'
          ```
        - O (múltiples opciones `-e`):
          ```
          sed -n -e '/a/ bLBL' -e 'd; :LBL' -e 'p' <<<$'a\nb'
          ```

- Funciones `i` **y** `a` **para insertar / agregar texto** : **siga el nombre de la función por** `\` **, seguido de una nueva línea *literal* o un empalmado** `$'\n'` antes de especificar el argumento del texto.

    - Sólo Linux:
      ```
      sed '1 i new first line' <<<$'a\nb' # -> 'new first line<nl>a<nl>b'
      ```
    - OSX *y* Linux:
      ```
      sed -e '1 i\'$'\n''new first line' <<<$'a\nb'
      ```
    - Nota:
        - Sin `-e` , el argumento de texto no es inexplicablemente terminado en una nueva línea en la salida en macOS (bug?).
        - **No utilice escapes de caracteres de control** como `\n \t` en el argumento de texto, ya que solo son compatibles con Linux.
        - Si el argumento de texto, por lo tanto, tiene nuevas líneas interiores reales, `\` - escape ellas.
        - Si desea colocar comandos adicionales después del argumento de texto, debe terminarlo con una nueva línea (no escapada) (ya sea literal o empalmada), o continuar con una opción `-e` separada (este es un requisito general que se aplica a todas las versiones) .

- Dentro de **las *listas de* funciones** (múltiples llamadas a función encerradas en `{...}`), **asegúrese de terminar también la *última* función, antes del cierre** `}` **, con** `;` .

    - Sólo Linux:
        - `sed -n '1 {p;q}' <<<$'a\nb' # -> 'a'`
    - macOS *y* Linux:
        - `sed -n '1 {p;q;}' <<<$'a\nb'`

Características específicas de GNU `sed` que faltan en BSD `sed` completo:

Las características de GNU que se perderán si necesita soportar ambas plataformas:

- Varias **opciones de coincidencia de expresiones regulares y sustitución** (tanto en los patrones de selección de líneas como en el primer argumento de la función `s`):
    - **La opción `I` para la coincidencia de expresiones regulares con mayúsculas y minúsculas (increíblemente, BSD `sed` no admite esto en absoluto).**
    - La opción `M` para la coincidencia de varias líneas (donde `^` / `$` coincide con el inicio / final de *cada línea*)
    - Para opciones adicionales que son específicas de la función `s` , consulte https://www.gnu.org/software/sed/manual/sed.html#The-_0022s_0022-Command
- **Secuencias de escape**
    - Secuencias de escape relacionadas con la sustitución como `\u` en el argumento de reemplazo de la función `s///` que permite la *manipulación de subcadenas* , dentro de los límites; por ejemplo, `sed 's/^./\u&/' <<<'dog' # -> 'Dog'` - vea http://www.gnu.org/software/sed/manual/sed.html#The-_0022s_0022 -Mando
    - Secuencias de escape de caracteres de control: además de `\n` , `\t` , ..., escapes basados en puntos de código; por ejemplo, todos los escapes siguientes (hex., octal, decimal) representan una comilla simple (`'`): `\x27` , `\o047` , `\d039` - vea https://www.gnu.org/software/sed/manual/ sed.html # Escapes
- **Extensiones de dirección** , como `first~step` para que coincida con cada línea de paso, `addr, +N` para que coincida con N líneas después de `addr` , ... - consulte http://www.gnu.org/software/sed/manual/sed. html # Direcciones

---

[1] La versión macOS `sed` es *más antigua* que la versión en otros sistemas similares a BSD como FreeBSD y PC-BSD. Desafortunadamente, esto significa que no puede asumir que las funciones que funcionan en FreeBSD, por ejemplo, funcionarán [las mismas] en macOS.

[2] La cadena de cotización ANSI `$'\001\002\003\004\005\006\007\010\011\013\014\015\016\017\020\021\022\023\024\025\026\027\030\031\032\0` contiene todos los caracteres de control ASCII excepto `\n` (y NUL), por lo que puede usarlo en combinación con `[:print:]` para una emulación bastante robusta de `[^\n]` :

```
'[[:print:]'$'\001\002\003\004\005\006\007\010\011\013\014\015\016\017\020\021\022\023\024\025\026\027\0
```

Examples

Reemplazar todas las nuevas líneas con pestañas

Nota: para mayor brevedad, los comandos utilizan aquí cadenas (`<<<`) y cadenas ANSI entre comillas (`$'...'`) . Estas dos funciones de shell funcionan en `bash` , `ksh` y `zsh` .

```
# GNU Sed
$ sed ':a;$!{N;ba}; s/\n/\t/g' <<<$'line_1\nline_2\nline_3'
```

```
line_1 line_2 line_3

# BSD Sed equivalent (multi-line form)
sed <<<$'line_1\nline_2\nline_3' '
:a
$!{N;ba
}; s/\n/'$'\t''/g'

# BSD Sed equivalent (single-line form, via separate -e options)
sed -e ':a' -e '$!{N;ba' -e '}; s/\n/'$'\t''/g' <<<$'line 1\nline 2\nline 3'
```

Notas de BSD Sed:

- Tenga en cuenta la necesidad de terminar las etiquetas (`:a`) y los comandos de bifurcación (`ba`) con nuevas líneas reales o con opciones `-e` separadas.
- Dado que las secuencias de escape de caracteres de control como `\t` no son compatibles con la cadena de reemplazo, se empalma un *literal de* tabulación ANSI con comillas en la cadena de reemplazo.
  (En la parte de *expresiones regulares* , BSD Sed *solo* reconoce `\n` como una secuencia de escape).

Agregar texto literal a una línea con la función 'a'

Nota: para mayor brevedad, los comandos utilizan aquí cadenas (`<<<`) y cadenas ANSI entre comillas (`$'...'`) .
Estas dos funciones de shell funcionan en `bash` , `ksh` y `zsh` .

```
# GNU Sed
$ sed '1 a appended text' <<<'line 1'
line 1
appended text

# BSD Sed (multi-line form)
sed '1 a\
appended text' <<<'line 1'

# BSD Sed (single-line form via a Bash/Ksh/Zsh ANSI C-quoted string)
sed $'1 a\\\nappended text' <<<'line 1'
```

Observe cómo BSD Seed requiere un `\` seguido de una *nueva línea real* para pasar el texto para agregar.
Lo mismo se aplica a las funciones relacionadas `i` (insertar) `c` (eliminar e insertar).

Lea BSD / macOS Sed vs. GNU Sed vs. la especificación POSIX Sed en línea: https://riptutorial.com/es/sed/topic/9436/bsd---macos-sed-vs--gnu-sed-vs--la-especificacion-posix-sed

Capítulo 4: Comandos avanzados sed

Examples

Inserte una nueva línea antes de hacer coincidir el patrón - usando eXchange

Dado un archivo file.txt con el siguiente contenido:

```
line 1
line 2
line 3
```

Puedes agregar una nueva línea usando el siguiente comando

```
sed '/line 2/{x;p;x;}' file.txt
```

El comando anterior dará salida

```
line 1

line 2
line 3
```

Explicación:

`x` comando `x` es eXchange. sed tiene un búfer que puedes usar para almacenar algunas líneas. Este comando intercambia este búfer con la línea actual (por lo que la línea actual va a este búfer y el contenido del búfer se convierte en la línea actual).

`p` comando `p` imprime la línea actual.

Lea Comandos avanzados sed en línea: https://riptutorial.com/es/sed/topic/3946/comandos-avanzados-sed

Capítulo 5: Dirección y rango de direcciones

Introducción

Los comandos sed se pueden especificar para actuar solo en ciertas líneas usando *direcciones* o *rangos de direcciones* .

Examples

Linea especifica

```
$ cat ip.txt
address
range
substitution
pattern
sample
```

- Línea *n* th

  ```
  $ sed -n '2p' ip.txt
  range

  $ sed '3d' ip.txt
  address
  range
  pattern
  sample
  ```

- Última línea

  ```
  $ sed -n '$p' ip.txt
  sample
  ```

Gama específica de líneas

```
$ cat ip.txt
address
range
substitution
pattern
sample
```

- El rango especificado incluye esos números de línea

  ```
  $ sed -n '2,4p' ip.txt
  range
  substitution
  ```

```
pattern
```

- `$` se puede utilizar para especificar la última línea. Se puede utilizar el espacio entre la dirección y el comando para mayor claridad.

```
$ sed -n '3,$ s/[aeiou]//gp' ip.txt
sbstttn
pttrn
smpl
```

GNU sed

- *i* th line to *i* + *j* th line

```
$ sed '2,+2d' ip.txt
address
sample
```

- *i* th line e *i* + *j* , *i* +2 *j* , *i* +3 *j* , etc.

```
$ sed -n '1~2p' ip.txt
address
substitution
sample
```

Líneas que coinciden con el patrón de expresión regular

```
$ cat ip.txt
address
range
substitution
pattern
sample
Add Sub Mul Div
```

- Líneas que coinciden con un patrón

```
$ sed '/add/d' ip.txt
range
substitution
pattern
sample
Add Sub Mul Div

$ sed -n '/t/p' ip.txt
substitution
pattern

$ sed -n '/[A-Z]/ s| |/|gp' ip.txt
Add/Sub/Mul/Div
```

- Gama de patrones

```
$ sed -n '/add/,/sub/p' ip.txt
address
range
substitution

$ sed -n '/a/,/e/p' ip.txt
address
range
pattern
sample
```

Nota

- En el segundo ejemplo, combinó dos rangos: líneas `1,2` y líneas `4,5`
- Consulte Uso de diferentes delimitadores sobre cómo usar otros caracteres en lugar de `/` para especificar el patrón.

GNU sed

- Partido insensible a mayúsculas

```
$ sed -n '/add/Ip' ip.txt
address
Add Sub Mul Div

$ sed -n '/add/I,/sub/p' ip.txt
address
range
substitution
Add Sub Mul Div
```

Especificando rango usando tanto números como patrones

```
$ cat ip.txt
address
range
substitution
pattern
sample
Add Sub Mul Div
```

- Número de línea para alinear el patrón

```
$ sed -n '2,/pat/p' ip.txt
range
substitution
pattern
```

- Patrón de coincidencia de línea a número de línea

```
$ sed '/pat/,$d' ip.txt
address
range
substitution
```

GNU sed

- Patrón de coincidencia de líneas más el número de líneas que lo siguen

```
$ sed -n '/add/I,+1p' ip.txt
address
range
Add Sub Mul Div
```

- Se puede usar `0` como número de línea de inicio para señalar el final del rango cuando el patrón coincide con la primera línea de entrada

```
$ sed -n '0,/r/p' ip.txt
address

$ sed -n '1,/r/p' ip.txt
address
range

$ sed -n '0,/u/p' ip.txt
address
range
substitution
```

Rango de direcciones negativas

```
$ cat ip.txt
address
range
substitution
1234
search pattern
sample
Add Sub Mul Div
```

- Borrado de líneas distintas a la dirección especificada

```
$ sed '/[0-9]/!d' ip.txt
1234

$ sed -n '/[0-9]/p' ip.txt
1234

$ sed '$!d' ip.txt
Add Sub Mul Div
```

```
$ sed -n '$p' ip.txt
Add Sub Mul Div
```

- Buscar y reemplazar en líneas que no coinciden con un patrón

```
$ sed '/ /! s/^/#/' ip.txt
#address
#range
#substitution
#1234
search pattern
#sample
Add Sub Mul Div

$ sed '/add/,/sub/! s/[aeiou]//gi' ip.txt
address
range
substitution
1234
srch pttrn
smpl
dd Sb Ml Dv
```

Lea Dirección y rango de direcciones en línea: https://riptutorial.com/es/sed/topic/3120/direccion-y-rango-de-direcciones

Capítulo 6: Edición in situ

Sintaxis

- sed -I *extension* - FreeBSD sed (contador de linea continua)
- sed -I [*extensión*] - NetBSD e Illumos sed (contador de línea continua)
- *extensión* sed -i - FreeBSD sed
- sed -i [*extensión*] - NetBSD, OpenBSD, Illumos, BusyBox y GNU sed
- sed --in-place [= *extension*] - Illumos, BusyBox y GNU sed

Parámetros

Parámetro	Detalles
`extension`	Guarde un archivo de respaldo con la extensión especificada, o ningún archivo de respaldo cuando la `extension` sea una cadena de longitud cero.

Observaciones

La edición in situ es una extensión común pero no estándar presente en la mayoría de los sistemas recientes.

De un manual BSD `sed`

(Una sección como esta aparece en todos los manuales actuales de BSD `sed` , y los de sus derivados)

> No se recomienda dar una extensión de longitud cero cuando esté en su lugar editando archivos, ya que se corre el riesgo de corrupción o contenido parcial en situaciones donde se agota el espacio en disco, etc.

No olvides la poderosa `ed`

Definitivamente, hay un uso para `sed` y para las funciones de edición in situ de `sed` , pero cuando se extiende el estándar UNIX, siempre debemos preguntarnos por qué el antiguo estándar UNIX no incluyó esa característica. Aunque UNIX no es perfecto, la ortogonalidad y la integridad de las herramientas se han desarrollado para que se encuentren cerca de la perfección, al menos para los fines que *eran* visibles alrededor de 1970: la *edición de texto y la edición automatizada de texto seguramente eran visibles alrededor de ese tiempo.*

En realidad, la idea de `sed` no es editar un *archivo* en su lugar, sino editar una *secuencia* . Es por eso que el nombre `sed` es una forma corta de *editor* de *secuencias* . Quita la `s` , y obtienes la herramienta que realmente fue diseñada para *la* edición de *archivos* : `ed` :

```
printf 'g/what to replace/s//with what to replace/g\nw\nq\n' | ed file
```

o `cat file_edit_commands | ed file` .

Examples

Reemplazo de cadenas en un archivo en el lugar

```
sed -i s/"what to replace"/"with what to replace"/g $file
```

Usamos `-i` para seleccionar la edición local en el `$file` file. En algunos sistemas se requiere agregar el sufijo después de la `-i` que se usará para crear una copia de seguridad del archivo original. Puede agregar una cadena vacía como `-i ''` para omitir la creación de la copia de seguridad. Mira los *comentarios* en este tema sobre la opción `-i` .

El terminador `g` significa hacer una búsqueda / reemplazo global en cada línea.

```
$ cat example
one
two
three
total
$ sed -i s/"t"/"g"/g example
$ cat example
one
gwo
ghree
gogal
```

Uso Portátil

La edición en contexto, aunque es común, es una característica no estándar. Una alternativa viable sería utilizar un archivo intermedio para almacenar el original o la salida.

```
sed 'sed commands' > file.out && mv file.out file
# or
mv file file.orig && sed 'sed commands' file.orig > file
```

Para usar la opción `-i` con la sintaxis de GNU y FreeBSD, se debe especificar una extensión y adjuntarla a la opción `-i` . Lo siguiente será aceptado por ambos, y producirá dos archivos, la versión original en `file.orig` y la versión editada en el `file` :

```
sed -i.orig 'sed commands' file
```

Ver un ejemplo básico dado un archivo de `file` :

```
$ cat file
one
two
```

```
three
$ sed -i.orig 's/one/XX/' file
$ cat file                      # the original file has changed its content
XX
two
three
$ cat file.orig                 # the original content is now in file.orig
one
two
three
```

Un ejemplo más complejo, reemplazando cada línea con un número de línea:

```
$ printf 'one\ntwo\n' | tee file1 | tr a-z A-Z > file2
$ sed -ni.orig = file1 file2
$ cat file1.orig file2.orig
one
two
ONE
TWO
$ cat file1 file2
1
2
1
2
```

¿Por qué se requiere un archivo de copia de seguridad?

Para utilizar la edición in situ sin un archivo de copia de seguridad, `-i` se debe dar un argumento de longitud cero y FreeBSD `sed` requiere un argumento a `-i` , ya sea anexado o por separado, mientras que la extensión argumento opcional GNU requiere el argumento se anexará a `-i` Ambos admiten la adición del argumento a `-i` , pero sin que sea necesario `-i'' command` es indistinguible de la `-i extension` , por lo que no se puede agregar un argumento de longitud cero a `-i` .

La edición en contexto sin especificar un archivo de copia de seguridad anula los permisos de solo lectura

`sed -i -e cmd file` modificará el `file` incluso si sus permisos están configurados como de solo lectura.

Este comando se comporta de manera similar a

```
sed -e cmd file > tmp; mv -f tmp file
```

más bien que

```
sed -e cmd file > tmp; cat tmp > file; rm tmp
```

El siguiente ejemplo usa gnu `sed` :

```
$ echo 'Extremely important data' > input
$ chmod 400 input  # Protect that data by removing write access
```

```
$ echo 'data destroyed' > input
-bash: input: Permission denied
$ cat input
Extremely important data (#phew! Data is intact)
$ sed -i s/important/destroyed/ input
$ cat input
Extremely destroyed data (#see, data changed)
```

Esto se puede mitigar creando una copia de seguridad especificando un `SUFFIX` con la opción `i` :

```
$ sed -i.bak s/important/destroyed/ input
$ cat input
Extremely destroyed data
$ cat input.bak
Extremely important data
```

Lea Edición in situ en línea: https://riptutorial.com/es/sed/topic/3640/edicion-in-situ

Capítulo 7: Eliminar comando

Examples

Eliminar una línea que contiene un patrón

Dado un archivo **file.txt** con el siguiente contenido:

```
line 1
line 2
line 3
```

Puede eliminar una línea del contenido del archivo con el comando `d` .

El patrón a coincidir está rodeado de predeterminado `/` delimitador y el comando `d` sigue el patrón:

```
sed '/line 2/d' file.txt
```

El comando anterior dará salida:

```
line 1
line 3
```

Para editar el archivo *en su lugar* , use la opción `-i` :

```
sed -i '/line 2/d' file.txt
```

Lea Eliminar comando en línea: https://riptutorial.com/es/sed/topic/2177/eliminar-comando

Capítulo 8: Expresiones regulares

Examples

Usando diferentes delimitadores.

Dado un archivo como este:

```
$ cat file
hello/how/are/you
i am fine
```

Puede usar `/pattern/` para hacer coincidir líneas específicas:

```
$ sed -n '/hello/p' file
hello/how/are/you
```

Si el patrón contiene barras inclinadas, puede usar otro delimitador usando `\cBREc` :

```
$ sed -n '\#hello/how#p' file
hello/how/are/you
$ sed -n '\_hello/how_p' file
hello/how/are/you
```

Según lo definido por POSIX en:

> **Expresiones regulares en sed**
> En una dirección de contexto, la construcción `\cBREc` , donde c es cualquier carácter que no sea barra invertida o, será idéntica a `/BRE/` . Si el carácter designado por c aparece después de una barra invertida, se considerará que es ese carácter literal, que no terminará el BRE. Por ejemplo, en la dirección de contexto "\ xabc \ xdefx", la segunda x se representa a sí misma, de modo que el BRE es "abcxdef".

Lea Expresiones regulares en línea: https://riptutorial.com/es/sed/topic/7720/expresiones-regulares

Capítulo 9: Opciones adicionales

Sintaxis

- -a - (BSD sed) Crea / Trunca todos los archivos escritos antes de procesar
- -E | -r - Usa expresiones regulares extendidas
- -i | -I - Consulte el tema sobre la edición en el lugar
- -l - (BSD sed) Usa la salida con buffer de línea
- -l longitud - (GNU sed) Especifique la longitud para `l` comando de ajuste de línea
- -s - (GNU sed) Trata los archivos como flujos separados
- -u - No búfer la salida
- -z - (GNU sed) Usa el carácter NUL para separar los registros
- --quieto | --silent - (GNU sed) Sinónimos para `-n`
- --expression = command - (GNU sed) Sinónimo para `-e`
- --file = command_file - (GNU sed) Sinónimo para `-f`
- --seguir-enlaces simbólicos - (GNU sed) Seguir enlaces simbólicos
- --in-place [= extension] - (GNU sed) Sinónimo para `-i`
- --line-length = length - (GNU sed) Sinónimo para `-l`
- --separate - (GNU sed) Sinónimo para `-s`
- --unbuffered - (GNU sed) Sinónimo de `-u`
- --null-data - (GNU sed) Sinónimo de `-z`
- --help - (GNU sed) Uso de impresión
- --version - (GNU sed) Versión para imprimir

Observaciones

La opción `-E` se estandarizará en la próxima versión principal, vea el problema relevante .

Examples

Retraso en la creación / truncamiento de archivos

Los archivos escritos con el comando `w` se crean / truncan antes de ejecutar cualquier comando.

```
$ sed 'w created-file' < /dev/null && ls created-file && rm created-file
created-file
```

De la norma:

> Cada archivo se creará antes de que comience el procesamiento. Las implementaciones admitirán al menos diez argumentos wfile en el script; el número real (mayor o igual a 10) que es compatible con la implementación no está especificado. El uso del parámetro wfile hará que ese archivo se cree inicialmente, si no existe, o reemplazará el contenido de un archivo existente.

BSD `sed` proporciona la opción `-a` para retrasar la creación / truncado de archivos hasta que se escriben con el comando `w` .

```
$ if sed -a 'w created-file' < /dev/null && [ ! -e created-file ]; then
>       echo The file was not created
> fi
The file was not created
```

'l' envoltura de línea

La longitud del ajuste de línea cuando se usa el comando `l` se define por la implementación.

De la norma:

> Las líneas largas se plegarán, con el punto de plegado indicado escribiendo a seguido de a; la longitud a la que se produce el plegado no está especificada, pero debe ser apropiada para el dispositivo de salida.

GNU `sed` proporciona la opción `-l` para especificar la longitud en la que se dividirán las líneas largas al imprimir con el comando `l` , con un valor predeterminado de setenta caracteres.

```
$ yes | head -c100 | tr '\n' ' ' | sed -n l | head -n1 | wc -c
      71
$ yes | head -c100 | tr '\n' ' ' | sed -nl50 l | head -n1 | wc -c
      51
```

BSD `sed` divide líneas largas en el número proporcionado por la variable de entorno `COLUMNS` , si `COLUMNS` no se proporciona, entonces se divide en el ancho del terminal, y si `COULMNS` no se proporciona y la salida no es un terminal, por defecto tiene sesenta caracteres.

```
$ yes | head -c100 | tr '\n' ' ' | sed -n l | head -n1 | wc -c
      61
$ yes | head -c100 | tr '\n' ' ' | COLUMNS=50 sed -n l | head -n1 | wc -c
      51
$ yes | head -c100 | tr '\n' ' ' | sed -n l | head -n1
y y y y y y y y y y y y y y y y y y y y y y y y y y y y y y y y y y y\
 y y y y y y y y y y $
```

Lea Opciones adicionales en línea: https://riptutorial.com/es/sed/topic/7922/opciones-adicionales

Capítulo 10: Operación de ramificación

Introducción

La operación de bifurcación de `sed` puede ayudar a controlar el flujo del programa.

Examples

Repetición de expresiones múltiples de líneas múltiples con ramificación incondicional

Supongamos que tengo un archivo llamado `in.txt` :

```
$ cat in.txt
a
b
a
c
a
d
```

Solo quiero reemplazar el `a\nc` por `deleted` , pero no `a\nb` o `a\nd` .

```
$ sed -e ':loop          # create a branch/label named `loop`
 $!{
 N                       # append the next line of input into the pattern space
 /\n$/!b loop            # If it is not the last line go to the `loop` branch again
 }
 s/a\nc/"deleted"/' in.txt    # do replacing in the pattern space

a
b
"deleted"    # see! succeed
a
d
```

Lea Operación de ramificación en línea: https://riptutorial.com/es/sed/topic/8821/operacion-de-ramificacion

Capítulo 11: Sustitución

Examples

Sustitución usando variables de shell

Las variables dentro de comillas simples `'` no se expanden con shells compatibles con POSIX, por lo que usar una variable de shell en una sustitución `sed` requiere el uso de comillas dobles `"` lugar de comillas simples `'` :

```
$ var="he"
$ echo "hello" | sed "s/$var/XX/"
XXllo

$ var="he"
$ echo "hello" | sed 's/$var/XX/'
hello
```

Tenga cuidado con la inyección de comandos al evaluar las variables:

```
$ var='./&/;x;w/etc/passwd
> x;s/he'
$ echo "hello" | sed "s/$var/XX/"
sed: /etc/passwd: Permission denied
```

Si se ejecutara lo anterior como raíz, la salida no se habría podido distinguir del primer ejemplo y el contenido de `/etc/passwd` se destruiría.

Referencia inversa

Usando corchetes escapados, puede definir un grupo de captura en un patrón que puede ser referenciado en la cadena de sustitución con `\1` :

```
$ echo Hello world! | sed 's/\(Hello\) world!/\1 sed/'
Hello sed
```

Con múltiples grupos:

```
$ echo one two three | sed 's/\(one\) \(two\) \(three\)/\3 \2 \1/'
three two one
```

BSD sed GNU sed

Cuando se usan expresiones regulares extendidas (ver Opciones adicionales), los paréntesis realizan la agrupación de forma predeterminada y no tienen que escaparse:

```
$ echo one two three | sed -E 's/(one) (two) (three)/\3 \2 \1/'
three two one
```

Las palabras que consisten en letras, dígitos y guiones bajos pueden combinarse utilizando la expresión `[[:alnum:]_]\{1,\}` :

```
$ echo Hello 123 reg_exp | sed 's/\([[:alnum:]_]\{1,\}\) \([[:alnum:]_]\{1,\}\)
\([[:alnum:]_]\{1,\}\)/\3 \2 \1/'
reg_exp 123 Hello
```

GNU sed

La secuencia `\w` es equivalente a `[[:alnum:]_]`

```
$ echo Hello 123 reg_exp | sed 's/\(\w\w*\) \(\w\w*\) \(\w\w*\)/\3 \2 \1/'
reg_exp 123 Hello
```

Usando diferentes delimitadores.

La especificación de base de grupo abierto POSIX / IEEE dice :

> **[2addr] s / BRE / replacement / flags**
>
> Sustituya la cadena de reemplazo por instancias de BRE en el espacio del patrón. **Se** puede usar **cualquier carácter que no sea barra invertida o nueva línea en** lugar de una barra inclinada para delimitar el BRE y el reemplazo. Dentro del BRE y el reemplazo, el delimitador BRE se puede usar como un carácter literal si está precedido por una barra invertida.

Hay casos en que el delimitador `/` para el reemplazo de `sed` está en el BRE o el reemplazo, lo que desencadena errores como:

```
$ echo "2/3/4" | sed "s/2/3/X/"
sed: -e expression #1, char 7: unknown option to `s'
```

Para esto, podemos usar diferentes delimitadores como `#` o _ o incluso un espacio:

```
$ echo "2/3/4" | sed "s#2/3#X#"
X/4
$ echo "2/3/4" | sed "s_2/3_X_"
X/4
$ echo "2/3/4" | sed "s 2/3 X "
X/4
```

Banderas de patrón - reemplazo de ocurrencia

Si queremos reemplazar solo la primera aparición en una línea, usamos `sed` como siempre:

```
$ cat example
aaaaabbbbb
aaaaaccccc
aaaaaddddd
$ sed 's/a/x/' example
xaaaabbbbb
```

```
xaaaaccccc
xaaaaddddd
```

Pero ¿y si queremos reemplazar todas las ocurrencias?

Acabamos de añadir la bandera del patrón `g` al final:

```
$ sed 's/a/x/g' example
xxxxxbbbbb
xxxxxccccc
xxxxxddddd
```

Y si queremos reemplazar una ocurrencia específica, podemos especificar cuál:

```
$ sed 's/a/x/3' example
aaxaabbbbb
aaxaaccccc
aaxaaddddd
```

`/3` siendo la tercera ocurrencia.

GNU sed

De `info sed` , vea el manual de GNU sed para la versión en línea

> El estándar POSIX no especifica lo que debería suceder cuando mezcla los modificadores `g` NUMBER, y actualmente no existe un significado ampliamente aceptado en todas las implementaciones `sed` . Para GNU `sed` , la interacción se define como: ignorar las coincidencias antes del NUMBERth, y luego coincidir y reemplazar todas las coincidencias desde el NUMBERth en adelante.

```
$ sed 's/b/y/2g' example
aaaaabyyyy
aaaaaccccc
aaaaaddddd

$ sed 's/c/z/g3' example
aaaaabbbbb
aaaaacczzz
aaaaaddddd
```

Lea Sustitución en línea: https://riptutorial.com/es/sed/topic/1096/sustitucion

Creditos

S. No	Capítulos	Contributors
1	Empezando con sed	Community, fedorqui, kdhp, SLePort
2	Agregar comando	kdhp, Slawomir Jaranowski
3	BSD / macOS Sed vs. GNU Sed vs. la especificación POSIX Sed	mklement0
4	Comandos avanzados sed	Raju
5	Dirección y rango de direcciones	Benjamin W., Sundeep
6	Edición in situ	AstraSerg, Ekeyme Mo, Emil Burzo, fedorqui, ghostarbeiter, ikrabbe, kdhp, mklement0, Oleg Arkhipov, William Pursell
7	Eliminar comando	SLePort
8	Expresiones regulares	fedorqui
9	Opciones adicionales	kdhp, mklement0
10	Operación de ramificación	Ekeyme Mo
11	Sustitución	Emil Burzo, fedorqui, kdhp, SLePort, Sundeep, thanasisp