

# sed

Stream EDitor

## sintaxis

```
sed [-ns] '[direccion] instrucion argumentos'
```

donde:

- -n : no mostrar por STDOUT las líneas procesadas
- -s : tratar los ficheros entrantes como flujos separados
- dirección: número de línea, rango de números de línea o búsqueda por regexp. Si no se especifica, afecta a todas las líneas del fichero
- instrucción (sobre la línea actual o procesada):
  - i : insertar antes
  - a : insertar después
  - c : cambiar
  - d : borrar
  - p : imprimir (STDOUT)
  - s : sustituir cadena
  - r <fichero> : añadir contenido de fichero
  - w <fichero> : escribir salida a fichero
  - ! : negar la condición
  - q : finalizar procesamiento de fichero

```
sed 's/busca/reemplaza/g' /path/to/file(s)
```

## parámetros

- -i : actualiza el fichero «sujeto»
- s/cadena1/cadena2/g : sustituye cadena1 por cadena2 en todo el documento (parámetro g - global)
- s+cadena1+cadena+g : idem anterior (hemos cambiado el separador de la REGEXP, por si nos interesa)
- s/abc/xyz/gI : (parámetro I permite sustituir ignorando si van en mayúsculas o minúsculas)

## ejemplos

reemplaza «abc» por «xyz» en el archivo1 generando el archivo2:

```
sed s/abc/xyz/g archivo1 > archivo2
```

actualiza el fichero, misma sustitución anterior

```
sed -i s/abc/xyz/g archivo1
```

idem anterior, pero con variables:

```
sed "s|$var1|$var2|" archivo1 > archivo2''
```

realiza una sustitución solo si la línea contiene un valor determinado, si la linea contiene **requisito**, cambia el **texto1** por el **texto2** :

```
sed -e "/requisito/s/texto1/texto2/g"
```

listar archivos/carpetas de una ruta:

```
ls -l | awk '{ print $NF }' | sed 1d
```

cambio temporal de prompt (PS1):

```
export PS1="C:\$( pwd | sed 's:/:\\\\:g' )\\> "
```

eliminar línea en concreto (la 3)

```
sed "3d" distros-deb.txt > distros-deb-ok.txt
sed -i "3d" distros-deb.txt # reescribe el fichero
```

más eliminaciones aplicadas en un cierto rango

```
sed -i "3,5d" distros-deb.txt # de la linea 3 a la 5
sed -i "2,$d" distros-deb.txt # de la linea 2 al final
```

ejemplo docker

```
sed -i /etc/ssh/sshd_config \
-e 's/#PermitRootLogin.*/PermitRootLogin no/' \
-e 's/#RSAAuthentication.*/RSAAuthentication yes/' \
-e 's/#PasswordAuthentication.*/PasswordAuthentication no/'
```

## ejemplos avanzados

añadir texto a una línea, respetando el contenido existente (y haciendo match). Uso de subexpresión (backreference)

```
sed -i "1,12s/^(host: \)[^ ]*/\1localhost/" config/database.yml
```

- -i : sobrrescribe el fichero (config/database.yml en este caso)
- 1,12 : sobre ese rango de líneas
- (host: ) : busca esa cadena, pero además la guarda. En la instrucción ( y ) están escapados ← subexpresión
- [^ ]\* : cualquier número de caracteres que no sean espacios ??
- \1 <cadena> : recupera el texto guardado en la anterior expresión y le añade <cadena>.. el 1 está escapado!

buscar frases de un texto que tengas 6 palabras (y acaben en algo que no sea una letra):

```
grep -E '^([a-zA-Z]+[a-zA-Z]+){6}$' frases
```

- ^ inicio de frase (dentro de corchetes, es negación!)
- (...) : subexpresión
- [a-zA-Z]+ : palabras
- [^a-zA-Z]+ : y no palabras
- {6} : afecta a toda la subexpresión y hace que se repita 6 veces

- \$ : final de frase (línea)

buscar palíndromos de 7 letras<sup>1)</sup>

```
grep '^(\.\.)\(\.\.)\(\.\.)\.\3\2\1$' /usr/share/dict/words
```

- cada grupo de paréntesis hace que ese carácter se guarde para «futuras» referencias. El primer paréntesis se referencia como 1, el segundo...
- y se recogen esos valores con \3 \2 \1 respectivamente

<sup>1)</sup>

palabra capicúa

From:

<https://miguelangel.torresegea.es/wiki/> - miguel angel torres egea



Permanent link:

<https://miguelangel.torresegea.es/wiki/linux:bash:sed?rev=1540308771>

Last update: **23/10/2018 08:32**