Comandos Esenciales para Procesamiento

En este capítulo veremos alguno de los comandos de Linux que resultan ser de gran utilidad en el área de la Bioinformática para el procesamiento inicial de archivos. Entre los comandos que revisaremos tenemos: cut, tr, wc, sort, uniq, grep, sed, awk.

Comando cut

Este comando permite seleccionar o cortar secciones de cada línea de un archivo y escribir el resultado en la salida estándar. Se puede utilizar para seleccionar partes de una línea por posición de byte, carácter y delimitador. También se puede utilizar para extraer datos de archivos csv, tsv.

Para obtener una sección de una línea especificando una posición de byte se utiliza la opción -b

```
>echo 'prueba' | cut -b 2
r
>echo 'prueba' | cut -b 1-2
pr
>echo 'prueba' | cut -b 1,3
pu
```

Para seleccionar por carácter se usa la opción -c. Cuando el flujo de entrada de datos se basa en caracteres, -c puede ser una mejor opción que seleccionar por bytes, ya que a menudo encontramos caracteres que son representados por más de un byte.

En el siguiente ejemplo, el carácter "♣" tiene tres bytes. Al usar la opción -c, el carácter se puede seleccionar correctamente junto con cualquier otro carácter que nos interese.

```
>echo '♣foobar' | cut -c 1,6
♣a
>echo '♣foobar' | cut -c 1-3
♣fo
```

También podemos seleccionar secciones del flujo de datos usando un delimitador. Para ello usaremos la opción -d. Esto se usa normalmente junto con la opción -f para especificar el campo que se debe seleccionar.

Por ejemplo, el archivo CSV (nombres.csv) almacena datos separados por el carácter coma:

```
John,Smith,34,London
Arthur,Evans,21,Newport
George,Jones,32,Truro
```

El delimitador se puede indicar utilizando la opción -d junto al delimitador de interés entre comillas

```
-d ','
```

El comando cut puede extraer los campos de interés con la opción -f.

```
>cut -d ',' -f 1 names.csv
John
Arthur
George
```

Se pueden extraer varios campos pasando una lista separada por comas.

```
>cut -d ',' -f 1,4 names.csv
John,London
Arthur,Newport
George,Truro
```

Comando tr

El comando *tr* (translate) se usa principalmente para traducir y eliminar caracteres de una secuencia o archivo. Puede usarse para convertir mayúsculas a minúsculas y borrar caracteres. Es muy importante tener en cuenta que *tr* no es compatible con codificaciones de caracteres que utilicen más de 1 byte para su representación, por ejemplo, Unicode y todas aquellas que usen esta (UTF-8). Esta limitación se debe a que el comando trabaja byte a byte en las operaciones que realiza (eliminación/reemplazo) por lo tanto, el uso de dichas codificaciones hace que *tr* genere resultados inesperados.

```
tr [opciones] SET1 [SET2]
```

- -c: complementa el conjunto de caracteres en la cadena, es decir, las operaciones se aplican a caracteres que no están en el conjunto dado
- -d: elimina los caracteres del primer conjunto de la salida.
- -s: reemplaza los caracteres repetidos enumerados en el set1 con una sola ocurrencia
- -t: trunca set1

Para convertir de minúsculas a mayúsculas, se pueden utilizar los conjuntos predefinidos en *tr*:

```
cat archivo | tr “[a-z]” “[A-Z]” > nuevo
cat archivo | tr “[:lower:]” “[:upper:]” > nuevo
```

Para convertir espacios en blanco a tabuladores usamos el siguiente comando:

```
cat archivo | tr “[:blank:]” “\t” > nuevo
cat archivo | tr “[:space:]” “\t” > nuevo
cat archivo | tr -s “[:space:]” “\t” > nuevo
```

A continuación, se muestran otros ejemplos del uso del comando tr:

```
cat archivo | tr “{}” “()” > nuevo     # cambia {} por ()
echo “Esta es     una prueba   con muchos blancos” | tr -s “[:space:]”
```

Para eliminar ciertos caracteres podemos usar la opción -d y especificando los caracteres en SET1.

```
echo “De este texto quiero 345 eliminar los dígitos 64” | tr -d “[:digit:]”
```

También podemos indicar con cuáles caracteres deseamos quedarnos y eliminar el resto usando la opción de complemento -c.

```
echo “Mi numero de identificación es 12345” | tr -cd “[:digit:]”
12345
```

Algunos de los conjuntos predefinidos que pueden ser usados con el comando *tr* son:

`[:alnum:]`	Todas las letras y dígitos
`[:alpha:]`	Todas las letras
`[:blank:]`	Todos los espacios horizontales
`[:cntrl:]`	Todos los caracteres de control
`[:digit:]`	Todos los dígitos
`[:graph:]`	Todos los caracteres imprimibles, no incluye el espacio
`[:lower:]`	Todas las letras minúsculas
`[:print:]`	Todos los caracteres imprimibles incluyendo el espacio
`[:punct:]`	Todos los caracteres de puntuación
`[:space:]`	Todos los espacios horizontales y verticales
`[:upper:]`	Todos los caracteres mayúsculos

Comando wc

wc (*word count*) es un comando muy sencillo que lo utilizamos para contar palabras, caracteres, bytes o lineas de un archivo o de la entrada estándar.

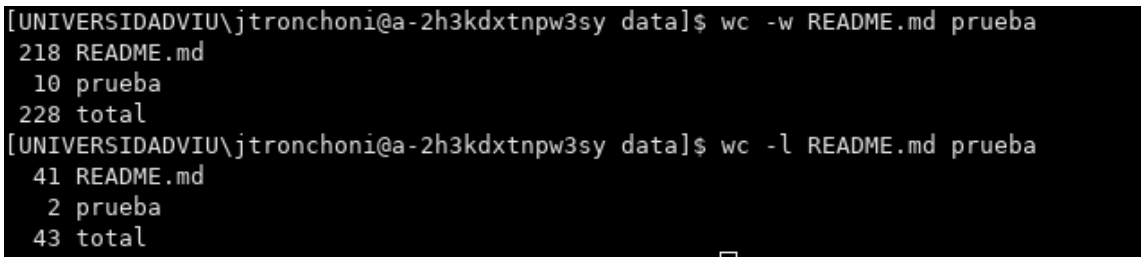
```
wc [opciones] [archivos]
```

Opciones:

- -c: cuenta el número de bytes (recuerde que dependiendo del tipo de codificación que se use un carácter puede ocupar mas de un byte)
- -m: cuenta el número de caracteres
- -l: cuenta el número de líneas

En la línea de comando se puede especificar uno o más archivos y nos mostrará el total general junto con el conteo para cada uno de los archivos. También se pueden usar metacaracteres para redirigir la entrada.

```
wc -m archivo
wc -m < archivo
wc -l archivo1 archivo 2
```



Comando sort

sort permite ordenar líneas de los archivos de entrada a partir de criterios de ordenación. Los espacios en blanco son tomados por defecto como separadores de campo.

```
sort [opciones] [archivo]
```

Opciones:

- -b ignora espacios en blanco precedentes

-d	ordena ignorando todos los caracteres salvo caracteres letras, números y espacios
-f	considera iguales las mayúsculas y minúsculas.
-n	ordena por valor numérico
-r	invierte el orden
-k n1,[n2]	especifica un campo como clave de ordenación, comienza en n1 y acaba en n2; los números de campo empiezan en 1
-o sal.txt	escribe el resultado en sal.txt

cat prueba

```
[UNIVERSIDADVIU\jtronchoni@a-2h3kdxtnpw3sy data]$ cat prueba
50 windows
10 www
 9 actualizacion
21 linux
 1 unix
[UNIVERSIDADVIU\jtronchoni@a-2h3kdxtnpw3sy data]$
```

sort prueba

```
[UNIVERSIDADVIU\jtronchoni@a-2h3kdxtnpw3sy data]$ sort prueba
10 www
 1 unix
21 linux
50 windows
 9 actualizacion
```

sort -n prueba

```
[UNIVERSIDADVIU\jtronchoni@a-2h3kdxtnpw3sy data]$ sort -n prueba
 1 unix
 9 actualizacion
10 www
21 linux
50 windows
```

sort -k 2 prueba

```
[UNIVERSIDADVIU\jtronchoni@a-2h3kdxtnpw3sy data]$ sort -k 2 prueba
 9 actualizacion
21 linux
 1 unix
50 windows
10 www
```

Comando uniq

Uniq se utiliza para informar u omitir cadenas o líneas repetidas. El comando *uniq* no detecta entradas duplicadas que no se encuentren en líneas adyacentes. Por tanto, primero necesitamos ordenar el archivo y luego podemos encontrar duplicados.

```
uniq [opciones] [entrada] [salida]
```

Algunas opciones:

-d	muestra solo las líneas que están repetidas
-c	muestra un informe en donde la columna de la izquierda contiene el número de repeticiones y en la siguiente columna la línea en si
-s N	ignora los primeros N caracteres
-f N	ignora los primeros N campos

Comando grep

grep (g/re/p, *globally search for a regular expression and print matching lines*) es una poderosa herramienta de búsqueda a nivel de línea de comandos que se incluye como parte de la mayoría de los sistemas Unix y Linux. Es una de las herramientas más útiles en bioinformática. En el nivel más básico, grep busca una cadena de caracteres que coincida con un patrón e imprimirá líneas que contengan una coincidencia.

La sintaxis básica es:

```
grep PATRON ARCHIVO
```

Al igual que otros comandos de Linux, hay varias opciones disponibles para este comando. Las opciones más útiles incluyen:

Argumento	Función
-v	Invierte la coincidencia o encuentra líneas que NO contengan el patrón
--color	Colorea el texto que coincide para facilitar su visualización
-F	Interpreta el patrón como una cadena literal
-H ,-h	Imprime o no imprime el nombre de archivo que coincida
-i	Ignora casos que coincidan con el patrón
-l	Lista los nombres de archivo que contienen el patrón (en lugar de coincidir)
-n	Imprime el número de líneas que contienen el patrón (en lugar de coincidir)

Argumento	Función
-c	Cuenta el número de coincidencias para un patrón
-o	Solo imprime el patrón que coincide
-w	Fuerza a que el patrón coincida con una palabra entera
-x	Fuerza patrones para que coincidan con toda la línea
-f	Obtiene patrones desde un archivo
-E	Interpreta el patrón como una expresión regular extendida

Con opciones, la sintaxis del comando es:

```
grep [OPCIONES] PATRON ARCHIVO
```

Algunos de los escenarios típicos en donde podemos utilizar grep son:

- Contar el número de secuencias en un archivo de secuencia fasta.
- Obtener las líneas de encabezado del archivo de secuencia fasta.
- Encontrar una coincidencia en un archivo de secuencia.
- Obtener todas las identificaciones de genes en archivos de secuencia y mucho más.

Veamos algunos ejemplos en donde aplicaremos el comando grep.

Secuencias de conteo

Según la definición del formato FASTA, sabemos que el número de secuencias en un archivo debe ser igual al número de líneas descriptivas. Entonces, si contamos el número de veces que aparece el carácter **>** en el archivo, obtendremos el número de secuencias. Esto se puede hacer usando la opción de conteo del grep con su opción -c.

```
grep -c ">" ARCHIVO
```

Sin embargo, hay que considerar si las definiciones tienen el carácter > más de una vez. Para estar seguro indicaremos que el carácter > debe aparecer al inicio de una línea usando el metacarácter ^ visto en la sección de expresiones regulares.

```
grep -c "^>" ARCHIVO
```

Contar el número de secuencias en *rrnDB-5.6_16S_rRNA.fasta*

```
grep -c "^>" rrnDB-5.6_16S_rRNA.fasta
```

Buscar secuencias

Si está buscando información sobre las secuencias, puede listar todos los encabezados (líneas de descripción) de las secuencias usando grep. Simplemente hay que buscar el carácter >.

```
grep ">" ARCHIVO
grep ">" rrnDB-5.6_16S_rRNA.fasta
```

Esta información puede ser enviada a un archivo si desea usarlo más adelante o puede simplemente canalizarlo al comando ***less*** o ***more*** para desplazarse línea por línea o página por página.

```
grep ">" rrnDB-5.6_16S_rRNA.fasta > secuencias.txt
grep ">" rrnDB-5.6_16S_rRNA.fasta | less
grep ">" rrnDB-5.6_16S_rRNA.fasta | more
```

Eliminar una lista de otra

Si hay una pequeña lista de genes que desea eliminar de una lista más grande, podemos usar la función grep con las siguientes opciones:

```
grep -Fvw -f SubLista.txt ListaCompleta.txt
grep -Fvw -f SubLista.txt ListaCompleta.txt > ListaReducida.txt
```

aquí -F y -w se asegurarán de que la palabra completa se use como cadena literal, -v NO imprimirá los patrones coincidentes y -f SubLista.txt es para decir que los patrones de entrada a eliminar se encuentran en el archivo.

Contar una palabra

Si la palabra o patrón a buscar aparece más de una vez en una línea, solo se contará una vez. Para evitar esto, debe utilizar la opción o.

```
grep -o "PATRON" ARCHIVO
grep -o que temporal
```

```
[UNIVERSIDADVIU\jtronchoni@a-2h3kdxtnpw3sy raw]$ cat temporal
a veces sucede que no sabemos lo que se quiere
asi que hay que estar seguro antes de actuar
[UNIVERSIDADVIU\jtronchoni@a-2h3kdxtnpw3sy raw]$ grep  que temporal
a veces sucede que no sabemos lo que se quiere
asi que hay que estar seguro antes de actuar
[UNIVERSIDADVIU\jtronchoni@a-2h3kdxtnpw3sy raw]$ grep -on que temporal
1:que
1:que
2:que
2:que
```

Ahora, se puede obtener todo tipo de información útil simplemente imprimiendo el patrón, en lugar de la línea completa. Por ejemplo, ¿cuántas veces aparece una palabra en cada línea?

```
grep -on que temporal | cut -f 1 -d ":" | sort | uniq -c
```

Esto imprimirá el número de la línea seguido del número de veces que aparece el patrón (que) en dicha línea.

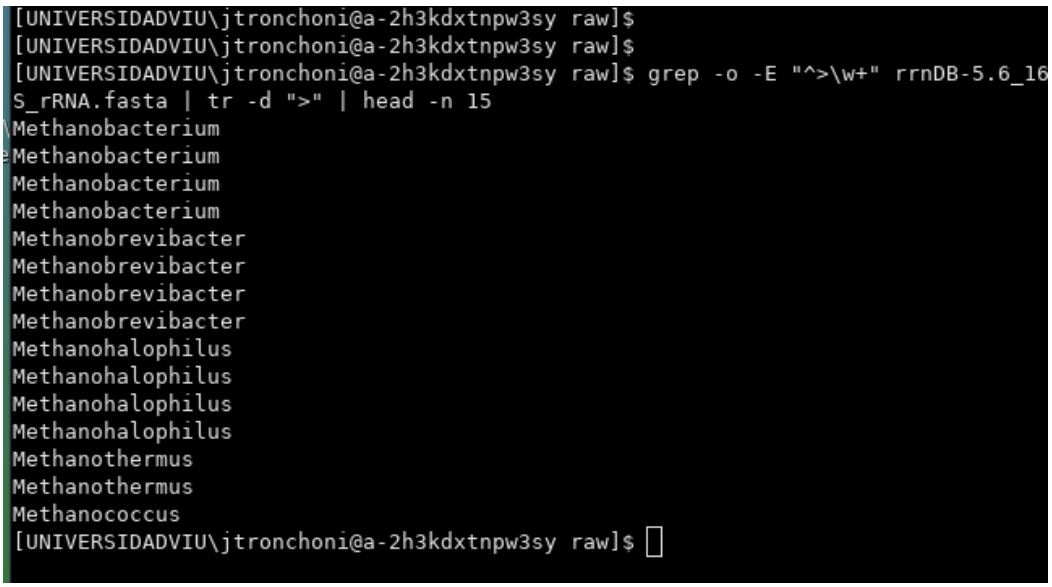
```
[UNIVERSIDADVIU\jtronchoni@a-2h3kdxtnpw3sy raw]$ grep -on que temporal
1:que
1:que
2:que
2:que
[UNIVERSIDADVIU\jtronchoni@a-2h3kdxtnpw3sy raw]$ grep -on que  temporal | cut -f
 1 -d ":" | sort | uniq -c
      2 1
      2 2
```

```
grep -o -E "^>\w+" rrnDB-5.6_16S_rRNA.fasta | head -n 15
```

```
[UNIVERSIDADVIU\jtronchoni@a-2h3kdxtnpw3sy raw]$ grep -o -E "^>\w+" rrnDB-5.6_16
S_rRNA.fasta | head -n 15
>Methanobacterium
>Methanobacterium
>Methanobacterium
>Methanobacterium
>Methanobrevibacter
>Methanobrevibacter
>Methanobrevibacter
>Methanobrevibacter
>Methanohalophilus
>Methanohalophilus
>Methanohalophilus
>Methanohalophilus
>Methanothermus
>Methanothermus
>Methanococcus
```

Para eliminar el carácter > que se encuentra al inicio de cada línea, usamos el comando *tr*.

```
grep -o -E "^>\w+" rrnDB-5.6_16S_rRNA.fasta | tr -d ">" | head -n 15
```



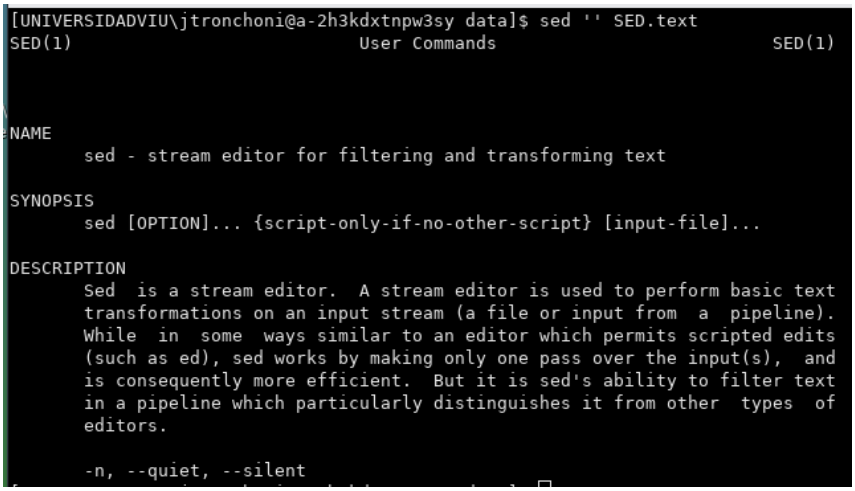
Comando sed

sed (*stream editor*) es un potente editor de flujo de texto que permite filtrar y transformar texto. Un editor de flujo se utiliza para realizar transformaciones de texto básicas en un flujo de entrada que puede ser a partir de un archivo o como entrada de una tubería (pipe). Esta capacidad de sed para filtrar texto usando tuberías es lo que lo distingue particularmente de otros tipos de editores.

```
sed [opciones] commandos [archivo-a-editar]
```

Usemos sed para ver el contenido de un archivo. sed envía sus resultados por defecto a la salida estándar, lo que significa que se puede usar como un lector de archivos sin pasarle comandos de edición.

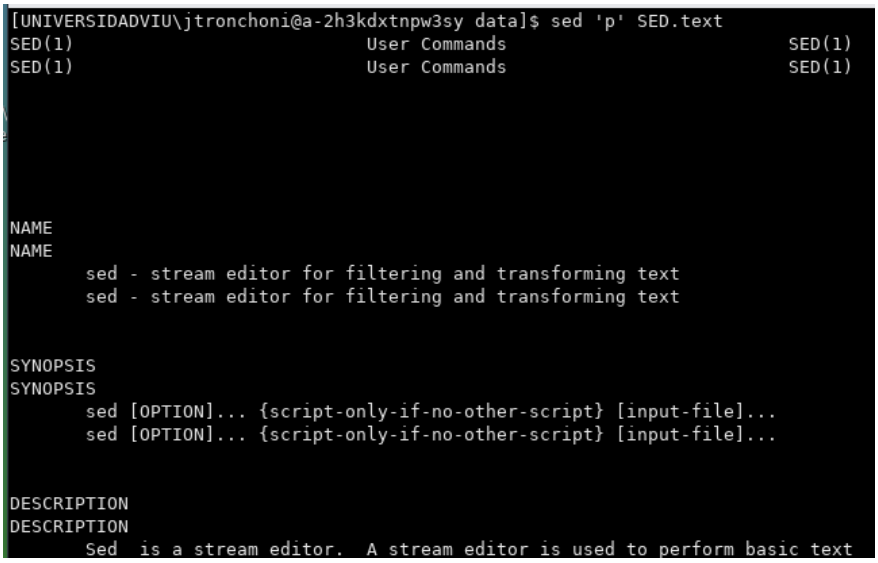
```
sed '' SED.txt
```



Veamos que sucede cuando ejecutamos el siguiente comando

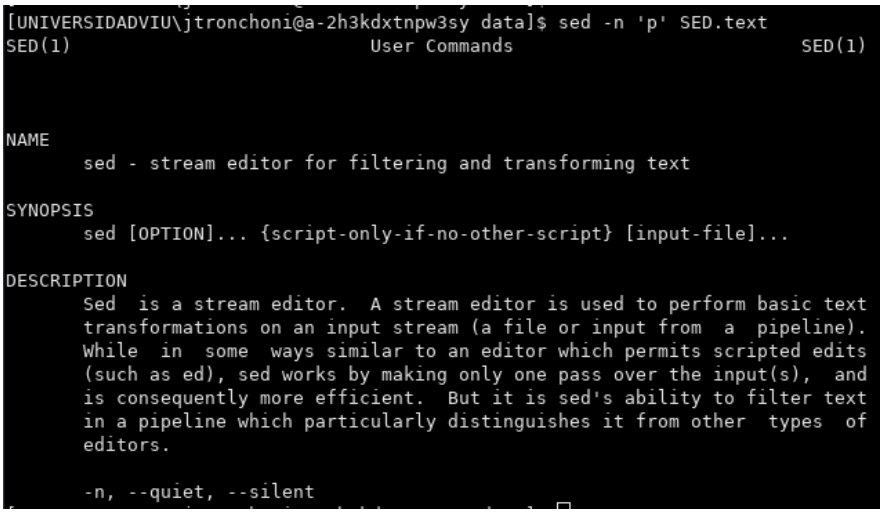
```
sed ‘p’ SED.txt
```

sed imprime automáticamente cada línea de forma predeterminada, y luego le hemos indicado que imprima líneas explícitamente con el comando "p", de modo que cada línea se imprime dos veces. sed opera los datos línea por línea. Lee una línea, opera en ella y genera el texto resultante antes de repetir el proceso en la siguiente línea.



Con la opción -n le indicamos al comando que no imprima de forma automática.

```
sed -n ‘p’ SED.txt
```



Impresión de un rango de líneas

sed nos permite imprimir rangos de líneas de un archivo:

```
sed -n '1p' SED.txt      # imprime la primera línea
sed -n '1,5p' SED.txt    # imprime las primeras 5 líneas
sed -n '1,+4p' SED.txt   # imprime las primeras 5 líneas
sed -n '1,~2p' SED.txt   # imprime cada 2 líneas
```

Eliminar texto

Para eliminar líneas de texto simplemente se cambia el comando p por el comando d. En este caso, ya no es necesario la opción -n porque sed imprimirá todo lo que no se elimine.

```
sed '1d' SED.txt      # elimina la primera línea
sed '1,5d' SED.txt    # elimina las primeras 5 líneas
sed '1,+4d' SED.txt   # elimina las primeras 5 líneas
sed '1,~2d' SED.txt   # elimina cada 2 líneas
```

Es importante tener en cuenta que nuestro archivo fuente no se ve afectado. Todavía está intacto. Las ediciones se envían a la salida estándar. Si queremos guardar nuestras ediciones, podemos redirigir la salida estándar a un archivo usando el metacarácter > o usar la opción -i. Esto alterará el archivo fuente.

```
sed -i '1~2d' SED.txt      # elimina cada dos líneas y modifica SED.txt
```

La opción -i puede ser peligrosa. Afortunadamente, sed le brinda la posibilidad de crear un archivo de respaldo antes de editarlo. Para crear un archivo de respaldo antes de editarlo, agregue la extensión de respaldo directamente después de la opción "-i":

```
sed -i.bak '1~2d' SED.txt    # crea una copia del archivo fuente, elimina cada dos líneas y
                             # modifica SED.txt
```

Sustitución de texto

El uso más conocido de sed es para la sustitución de texto. sed puede buscar patrones de texto usando expresiones regulares y luego reemplazar el texto encontrado con otra cosa. En su forma más básica, puede cambiar una palabra por otra usando la siguiente sintaxis:

```
's/palabra_a_sustituir/nueva_palabra/'
```

La **s** es el comando de sustitución. Las tres barras (/) se utilizan para separar los diferentes campos de texto. Se puede utilizar otro carácter para delimitar los campos si resulta más útil. Por ejemplo, si intenta cambiar el nombre de un sitio web, sería útil utilizar otro delimitador, ya que las URL contienen barras.

```
echo "http://www.example.com/index.html" | sed 's_com/index_org/home_'
```

sed reemplaza patrones, no palabras. De forma predeterminada, el comando **s** opera en la primera coincidencia de una línea y luego pasa a la siguiente línea. Para hacer que ***sed*** reemplace cada instancia de un patrón en lugar de solo la primera instancia en cada línea, debe pasar la bandera opcional **g** al comando **s**.

```
sed 's/on/forward/g' song.txt
```

Si solo desea cambiar la segunda instancia de "on" que ***sed*** encuentra en cada línea, entonces usaría el número 2 en lugar de la g:

```
sed 's/on/forward/2' song.txt
```

Si solo desea ver qué líneas fueron sustituidas, use la opción -n nuevamente para suprimir la impresión automática. A continuación, puede pasar la opción p al comando **s** para imprimir las líneas donde tuvo lugar la sustitución.

```
sed -n 's/on/forward/2p' song.txt
```

Se pueden combinar las banderas al final del comando, así por ejemplo, si deseamos que el proceso de búsqueda ignore las mayúsculas y minúsculas se puede pasar la bandera *i*.

```
sed 's/on/forward/i' song.txt
```

Reemplazo y referencia de texto coincidente

Si deseamos encontrar patrones más complejos con expresiones regulares, tenemos varios métodos diferentes para hacer referencia al patrón que queremos reemplazar. Por ejemplo, para hacer coincidir desde el principio de la línea hasta la cadena *at*, usamos el siguiente comando:

Texto original

```
this is the song that never ends
yes, it goes on and on, my friend
some people started saying it
not knowing what it was
and they'll continue saying it forever
just because...
```

```
sed 's/^.*at/REPLACED/' song.txt
```

Salida

```
REPLACED never ends
yes, it goes on and on, my friend
some people started singing it
REPLACED it was
and they'll continue singing it forever
just because...
```

La expresión * (wildcard) coincide desde el principio de la línea hasta la última instancia de la cadena at. Dado que no conocemos la frase exacta que coincidirá en la cadena de búsqueda, podemos usar el carácter & para representar el texto coincidente en la cadena de reemplazo.

Texto original

```
this is the song that never ends
yes, it goes on and on, my friend
some people started saying it
not knowing what it was
and they'll continue saying it forever
just because...
```

```
sed 's/^.*at/(&)/' song.txt
```

```
Salida
(this is the song that) never ends
yes, it goes on and on, my friend
some people started singing it
(not knowing what) it was
and they'll continue singing it forever
just because...
```

Una forma más flexible de hacer referencia al texto coincidente es utilizar paréntesis de escape para agrupar secciones de texto coincidente. Se puede hacer referencia a cada grupo de texto de búsqueda marcado entre paréntesis mediante un número de referencia de escape. Por ejemplo, se puede hacer referencia al primer grupo de paréntesis con \1, al segundo con \2 y así sucesivamente.

En este ejemplo, intercambiaremos las dos primeras palabras de cada línea:

```
sed 's/\([a-zA-Z0-9][a-zA-Z0-9]*\) \([a-zA-Z0-9][a-zA-Z0-9]*\)/\2 \1/' song.txt
```

```
Texto original
this is the song that never ends
yes, it goes on and on, my friend
some people started saying it
not knowing what it was
and they'll continue saying it forever
just because...
```

```
sed 's/\([a-zA-Z0-9][a-zA-Z0-9]*\)\([a-zA-Z0-9][a-zA-Z0-9]*\)/\2 \1/' song.txt
```

```
Output
is this the song that never ends
yes, goes it on and on, my friend
people some started singing it
knowing not what it was
they and'll continue singing it forever
because just...
```

El resultado obtenido no es perfecto. La segunda línea omite la primera palabra porque tiene un carácter que no figura en nuestro conjunto de caracteres. De manera similar, hay un problema con la quinta línea. Podemos mejorar la expresión regular para que sea más precisa.

```
Texto original
this is the song that never ends
yes, it goes on and on, my friend
```

```
some people started saying it
not knowing what it was
and they'll continue saying it forever
just because...
```

```
sed 's/\([^ ][^ ]*\) \([^ ][^ ]*\)/\2 \1/'  song.txt
```

Salida

```
is this the song that never ends
it yes, goes on and on, my friend
people some started singing it
knowing not what it was
they'll and continue singing it forever
because... just
```

Comando awk

awk es un lenguaje de programación y un procesador de texto que se puede utilizar para manipular datos de texto. Es más útil cuando se manejan archivos de texto formateados de forma predecible. Por ejemplo, es excelente para analizar y manipular datos tabulares. Opera línea por línea y recorre todo el archivo. Por defecto, utiliza espacios en blanco (espacios, tabulaciones, etc.) para separar campos.

El formato básico de un comando awk es:

```
awk '/patron_busqueda/{accion_a_realizar_coincidencia;otra_accion} '  archivo
```

Se puede omitir la parte de búsqueda o la parte de acción de cualquier comando awk. Por defecto, la acción que se toma si no se proporciona la parte de "acción" es imprimir. Es decir, imprime todas las líneas que coinciden. Si no se proporciona la parte de búsqueda, awk realiza la acción enumerada en cada una de las líneas del archivo.

Si se especifica tanto el patron a buscar como la acción, awk usa la parte de búsqueda para decidir si la línea actual refleja el patrón y luego realiza las acciones en dichas coincidencias.

```
> echo "sandia maria
ensalada sandy
pasta luis
sandwich jose
pollo luisiana" > favoritos.txt
```

Usaremos el comando awk para imprimir el archivo:

```
awk '{print} '  favoritos.txt
```

```
sandia maria
ensalada sandy
pasta luis
sandwich jose
pollo luisiana
```

Filtremos el archivo buscando el texto sand:

```
awk '/sand/'  favoritos.txt
```

Y obtenemos como salida:

```
sandia maria
ensalada sandy
sandwich jose
```

Podemos usar expresiones regulares para encontrar solo el patr'on al inicio de la línea y que además solo muestre la primera columna:

```
awk '/^sand/ {print $1;}'  favoritos.txt
```

```
sandia
sandwich
```

Puede hacer referencia a cada columna mediante variables asociadas con su número de columna. Por ejemplo, la primera columna es $1, la segunda es $2 y puede hacer referencia a toda la línea con $0.

Variables internas y formato expandido

El comando awk usa algunas variables internas para asignar ciertas piezas de información mientras procesa un archivo. Las variables internas que usa awk son:

FILENAME: Hace referencia al archivo actual de entrada

FNR: Hace referencia al número del registro actual en relación con el archivo de entrada actual

FS: Separador de campo que se usa para denotar cada campo en un registro. Por defecto es el espacio en blanco

NF: Número de campos en el registro actual.

NR: Número del registro actual.

OFS: Separador de campo para los datos generados. Por defecto se establece en espacios en blanco

ORS: Separador de registros para los datos generados. Por defectoeste es un carácter de nueva línea

RS: Separador de registros utilizado para distinguir registros en el archivo de entrada. Por defecto es un carácter de nueva línea

Estos valores pueden ser cambiados para que se ajusten a las necesidades de sus archivos. Por lo general, esto se hace durante la fase de inicialización de su procesamiento.

Esto nos lleva a otro concepto importante. La sintaxis awk es un poco más compleja que la que hemos utilizado hasta ahora. También hay bloques BEGIN y END opcionales que pueden contener comandos para ejecutar antes y después del procesamiento del archivo, respectivamente.

La sintaxis ampliada del comando awk es:

```
awk 'BEGIN { accion; }'
/busqueda/{accion;}
END {accion;}  archivo_entrada
```

Las palabras claves BEGIN y END son conjuntos de condiciones específicos, al igual que los parámetros de búsqueda. Coinciden antes y después de que se haya procesado el documento. Esto significa que puede cambiar algunas de las variables internas en la sección BEGIN. Por ejemplo, el archivo /etc/passwd está delimitado con dos puntos (:) en lugar de espacios en blanco.

```
awk 'BEGIN { FS=":"; }
{print $1}
END {;}'  /etc/passwd
```

```
root
bin
daemon
```

```
adm
lp
sync
shutdown
halt
mail
operator
games
ftp
nobody
...
```

Puede utilizar los bloques BEGIN y END para imprimir encabezados de los campos que está imprimiendo. El siguiente ejemplo muestra el uso del comando awk para transformar los datos del archivo en una tabla espaciada con tabuladores.

```
awk 'BEGIN { FS=":"; print "Usuario\t\tId Usuario\t\tHome\n----------"; }
{print $1,"\t\t", $3,"\t\t",$6;}
END {print "----------\n"}'  /etc/passwd
```

Búsqueda de campos y expresiones compuestas

En un ejemplo anterior, imprimimos las líneas del archivo favoritos.txt que comenzaba con "sand". Esto fue fácil porque buscaba el comienzo de toda la línea. ¿Qué pasa si desea saber si un patrón de búsqueda coincide al principio de un campo? Supongamos que el archivo favoritos.txt contiene un número de artículo delante de la comida favorita de cada persona,

```
1 sandia maria
2 ensalada sandy
3 pasta luis
4 sandwich jose
5 pollo luisiana
```

Podemos indicarle al comando awk que busque el patrón al inicio del segundo campo o segunda columna del archivo

```
awk '$2 ~ /^sand/'  favoritos.txt
```

El campo_numerico ~ especifica al comando que solo debe buscar en la columna especificada en campo_numerico

Con la misma facilidad, podemos buscar elementos que no coincidan si incluye el carácter "!" antes de la tilde (~). Este comando devolverá todas las líneas que no tengan un alimento que comience con "sand":

```
awk '$2 !~ /^sand/’  favoritos.txt
```

Si solo le interesan las líneas que no comienzan con "sand" y el número de artículo es menor que 5, puede usar una expresión compuesta como:

```
awk '$2 !~ /^sand/ && $1 < 5’  favoritos.txt
```