Procesamiento de texto.

Todos los sistemas operativos similares a Unix dependen en gran medida de archivos de texto para el almacenamiento de datos. Por tanto, tiene sentido que haya muchas herramientas para manipular texto actual. En este capítulo, veremos los programas que se utilizan para "cortar y cortar" texto. En el próximo capítulo, veremos más procesamiento de texto, enfocándonos en programas que se utilizan para formatear texto para impresión y otros tipos de consumo humano.

Este capítulo revisará a algunos viejos amigos y nos presentará a algunos nuevos:

> cat | Concatenar archivos e imprimir en la salida estándar

> sort | Ordenar líneas de archivos de texto

> uniq | Informar u omitir líneas repetidas

> cut | Eliminar secciones de cada línea de archivos

> paste | Fusionar líneas de archivos

> join | Unir líneas de dos archivos en un campo común

> comm | Compare dos archivos ordenados línea por línea

> diff | Comparar archivos línea por línea

> patch | Aplicar un archivo diff a un original

> tr | Traducir o eliminar personajes

> sed | Editor de secuencias para filtrar y transformar texto

> aspell | Corrector ortográfico interactivo

Aplicaciones de texto

Hasta ahora, hemos aprendido un par de editores de texto (nano y vim), hemos analizado un montón de archivos de configuración y hemos sido testigos de la salida de docenas de comandos, todos en texto. Pero, ¿para qué más se usa el texto? Por muchas cosas, resulta.

Documentos

Mucha gente escribe documentos utilizando formatos de texto sin formato. Si bien es fácil ver cómo un archivo de texto pequeño puede ser útil para guardar notas simples, también es posible escribir documentos grandes en formato de texto. Un enfoque popular es escribir un documento grande en formato de texto y luego incrustar un lenguaje de marcado para describir el formato del documento terminado. Muchos artículos científicos se escriben utilizando este método, ya que los sistemas de procesamiento de texto basados en Unix estuvieron entre los primeros sistemas que admitieron el diseño tipográfico avanzado que necesitan los escritores en disciplinas técnicas.

Páginas web

El tipo de documento electrónico más popular del mundo es probablemente la página web. Las páginas web son documentos de texto que utilizan lenguaje de marcado de hipertexto (HTML) o lenguaje de marcado extensible (XML) como lenguajes de marcado para describir el formato visual del documento.

Email

El correo electrónico es un medio intrínsecamente basado en texto. Incluso los adjuntos que no son de texto se convierten en una representación de texto para su transmisión. Podemos ver esto por nosotros mismos al descargar un mensaje de correo electrónico y luego verlo en menos. Veremos que el mensaje comienza con un encabezado que describe la fuente del mensaje y el procesamiento que recibió durante su recorrido, seguido del cuerpo del mensaje con su contenido.

Salida de impresora

En sistemas similares a Unix, la salida destinada a una impresora se envía como texto sin formato o, si la página contiene gráficos, se convierte a un lenguaje de descripción de página en formato de texto conocido como PostScript, que luego se envía a un programa que genera los puntos gráficos para ser impreso.

Código fuente del programa

Muchos de los programas de línea de comandos que se encuentran en sistemas similares a Unix se crearon para respaldar la administración del sistema y el desarrollo de software, y los programas de procesamiento de texto no son una excepción. Muchos de ellos están diseñados para resolver problemas de desarrollo de software. La razón por la que el procesamiento de texto es importante para los desarrolladores de software es que todo el software comienza como texto. El código fuente, la parte del programa que el programador escribe, está siempre en formato de texto.

Revisando a algunos viejos amigos

En el capítulo 6 aprendimos sobre algunos comandos que pueden aceptar entradas estándar además de argumentos de línea de comandos. Los abordamos solo brevemente entonces, pero ahora veremos más de cerca cómo se pueden usar para realizar el procesamiento de texto.

cat

El programa cat tiene varias opciones interesantes. Muchos de ellos se utilizan para ayudar a visualizar mejor el contenido del texto. Un ejemplo es la opción -A, que se utiliza para mostrar caracteres que no se imprimen en el texto. Hay ocasiones en las que queremos saber si los caracteres de control están incrustados en nuestro texto que de otro modo sería visible. Los más comunes son los caracteres de tabulación (a diferencia de los espacios) y los retornos de carro, a menudo presentes como caracteres de final de línea en archivos de texto de estilo MS-DOS. Otra situación común es un archivo que contiene líneas de texto con espacios al final.

Creemos un archivo de prueba usando cat como un procesador de texto primitivo. Para hacer esto, simplemente ingresaremos el comando cat (junto con la especificación de un archivo para la salida redirigida) y escribiremos nuestro texto, seguido de enter para finalizar correctamente la línea y luego ctrl -D para indicarle a cat que hemos llegado al final de archivo. En este ejemplo, ingresamos un carácter de tabulación inicial y seguimos la línea con algunos espacios finales:

```
[~]$ cat > foo.txt
The quick brown fox jumped over the lazy dog.
[~]$
```

A continuación, usamos cat con la opción -A para mostrar el texto.

```
[~]$ cat -A foo.txt
^IThe quick brown fox jumped over the lazy dog. $
[~]$
```

Como podemos ver en los resultados, el carácter de tabulación en nuestro texto está representado por ^I. Esta es una notación común que significa ctrl-I, que resulta que es lo mismo que un carácter de tabulación. También vemos que aparece un $ al final verdadero de la línea, lo que indica que nuestro texto contiene espacios al final.

MS-DOS Text vs. Unix Text

Una de las razones por las que puede querer usar cat para buscar caracteres que no se imprimen en el texto es para detectar retornos de carro ocultos. ¿De dónde proceden los retornos de carro ocultos? DOS y Windows! Unix y DOS no definen el final de una línea de la misma forma en los archivos de texto. Unix termina una línea con un carácter de salto de línea (ASCII 10), mientras que MS-DOS y sus derivados utilizan el retorno de carro de secuencia (ASCII 13) y el salto de línea para terminar cada línea de texto.

Hay varias formas de convertir archivos de formato DOS a Unix. En muchos sistemas Linux, existen programas llamados dos2unix y unix2dos, que pueden convertir archivos de texto desde y hacia el formato DOS.

Sin embargo, si no tiene dos2unix en su sistema, no se preocupe. El proceso de conversión de texto de formato DOS a Unix es simple; implica la eliminación de los retornos de carro infractores. Eso se logra fácilmente con un par de programas que se describen más adelante en este capítulo.

---

cat también tiene opciones que se utilizan para modificar texto. Los dos más destacados son -n, que numera las líneas, y -s, que suprime la salida de varias líneas en blanco. Podemos demostrar así:

```
[~]$ cat > foo.txt
```

The quick brown fox

–

jumped over the lazy dog.

```
[~]$ cat -ns foo.txt
```

1 The quick brown fox

2

3 jumped over the lazy dog.

```
[~]$
```

En este ejemplo, creamos una nueva versión de nuestro archivo de prueba foo.txt, que contiene dos líneas de texto separadas por dos líneas en blanco. Después de procesar por cat con las opciones -ns, se elimina la línea en blanco adicional y se numeran las líneas restantes. Si bien esto no es un gran proceso para realizar en texto, es un proceso.

sort (ordenar)

El programa sort ordena el contenido de la entrada estándar, o uno o más archivos especificados en la línea de comando, y envía los resultados a la salida estándar. Usando la misma técnica que usamos con cat, podemos demostrar el procesamiento de la entrada estándar directamente desde el teclado de la siguiente manera:

```
[~]$ sort > foo.txt
```

c

b

a

```
[~]$ cat foo.txt
```

a

b

c

Después de ingresar el comando, ingresamos las letras c, b y a, y luego presionamos ctrl-D para indicar el final del archivo. Luego vemos el archivo resultante y vemos que las líneas ahora aparecen ordenadas. Dado que sort puede aceptar varios archivos en la línea de comando como argumentos, es posible combinar varios archivos en un solo todo ordenado. Por ejemplo, si tuviéramos tres archivos de texto y quisiéramos combinarlos en un solo archivo ordenado, podríamos hacer algo como esto:

```
sort file1.txt file2.txt file3.txt > final_sorted_list.txt
```

---

sort tiene varias opciones interesantes. La Tabla siguiente, proporciona una lista parcial.

Opciones comunes de sort

Option	Long option	Description
-b	--ignore-leading-blanks	De forma predeterminada, la clasificación se realiza en toda la línea, comenzando con el primer carácter de la línea. Esta opción hace que la ordenación ignore los espacios iniciales en las líneas y calcula la ordenación en función del primer carácter que no sea un espacio en blanco de la línea.
-f	--ignore-case	Haga que la clasificación no distinga entre mayúsculas y minúsculas.
-n	--numeric-sort	Realice una clasificación basada en la evaluación numérica de una cadena. El uso de esta opción permite que la clasificación se realice en valores numéricos en lugar de valores alfabéticos.
-r	--reverse	Ordene en orden inverso. Los resultados están en orden descendente en lugar de ascendente.
-k	--key=field1[,field2]	Ordene según un campo clave ubicado de field1 a field2 en lugar de la línea completa. Vea la siguiente discusión.
-m	--merge	Trate cada argumento como el nombre de un archivo preordenado. Combine varios archivos en un único resultado clasificado sin realizar ninguna clasificación adicional.
-o	--output=file	Envíe la salida ordenada a un archivo en lugar de la salida estándar.
-t	--field-separator=char	Defina el carácter separador de campos. Por defecto, los campos están separados por espacios o tabulaciones.

Aunque la mayoría de estas opciones se explican por sí mismas, algunas no lo son. Primero, veamos la opción -n, usada para ordenar numéricamente. Con esta opción, es posible ordenar valores basándose en valores numéricos en lugar de lexográficamente. Podemos demostrar esto ordenando los resultados del comando du para determinar los usuarios más grandes de espacio en disco. Normalmente, el comando du enumera los resultados de un resumen en orden de nombre de ruta.

```
[~]$ du -s /usr/share/* | head
```

252 /usr/share/aclocal

...

En este ejemplo, canalizamos los resultados al encabezado para limitar los resultados a las primeras 10 líneas. Podemos producir una lista ordenada numéricamente para mostrar los 10 mayores consumidores de espacio de esta manera.

```
[~]$ du -s /usr/share/* | sort -nr | head
```

509940 /usr/share/locale-langpack

...

Al usar las opciones n y r, producimos una ordenación numérica inversa, con los valores más grandes apareciendo primero en los resultados. Esta clasificación funciona porque los valores numéricos se encuentran al principio de cada línea. Pero, ¿qué pasa si queremos ordenar una lista en función de algún valor encontrado dentro de la línea? Por ejemplo, aquí están los resultados de ls -l:

```
[~]$ ls -l /usr/bin | head
```

```
total 152948
-rwxr-xr-x 1
...
```

Ignorando, por el momento, que ls puede ordenar sus resultados por tamaño, también podríamos usar sort para ordenar esta lista por tamaño de archivo.

```
[~]$ ls -l /usr/bin | sort -nrk 5 | head
```

Muchos usos de sort implican el procesamiento de datos tabulares, como los resultados del comando ls anterior. Si aplicamos la terminología de la base de datos a la tabla anterior, diríamos que cada fila es un registro y que cada registro consta de varios campos, como los atributos del archivo, el número de enlaces, el nombre del archivo, el tamaño del archivo, etc. sort puede procesar campos individuales. En términos de base de datos, podemos especificar uno o más campos clave para usar como claves de clasificación. En el ejemplo anterior, especificamos las opciones nyr para realizar una ordenación numérica inversa y especificamos -k 5 para que la ordenación utilice el quinto campo como clave para la ordenación. La opción k es interesante y tiene muchas características, pero primero debemos hablar sobre cómo sort define los campos. Consideremos el siguiente archivo de texto simple que consta de una sola línea que contiene el nombre del autor:

```
William Shotts
```

De forma predeterminada, sort considera que esta línea tiene dos campos. El primer campo contiene estos caracteres: "William". El segundo campo contiene estos caracteres "Shotts".

Esto significa que los caracteres de espacio en blanco (espacios y tabulaciones) se utilizan como delimitadores entre campos y que los delimitadores se incluyen en el campo cuando se realiza la clasificación. Mirando nuevamente una línea de nuestra salida ls, como sigue, podemos ver que una línea contiene ocho campos y que el quinto campo es el tamaño del archivo:

```
-rwxr-xr-x 1 root root 8234216 2016-04-07 17:42 inkscape
```

Para nuestra próxima serie de experimentos, consideremos el siguiente archivo que contiene el historial de tres distribuciones populares de Linux lanzadas de 2006 a 2008. Cada línea del archivo tiene tres campos: el

nombre de la distribución, el número de versión y la fecha de lanzamiento en formato MM/DD/AA.

Nombre	Distribución	Lanzamiento
SUSE	10.2	12/07/2006
Fedora	10	11/25/2008
SUSE	11.0	06/19/2008
Ubuntu	8.04	04/24/2008
Fedora	8	11/08/2007
SUSE	10.3	10/04/2007
Ubuntu	6.10	10/26/2006
Fedora	7	05/31/2007
Ubuntu	7.10	10/18/2007
Ubuntu	7.04	04/19/2007
SUSE	10.1	05/11/2006
Fedora	6	10/24/2006
Fedora	9	05/13/2008
Ubuntu	6.06	06/01/2006
Fedora	8.10	10/30/2008
Ubuntu	5	03/20/2006

Usando un editor de texto (quizás vim), ingresaremos estos datos y nombraremos el archivo resultante distros.txt. A continuación, intentaremos ordenar el archivo y observar estos resultados:

```
[~]$ sort distros.txt
```

- Fedora 10 11/25/2008
- Fedora 5 03/20/2006
- Fedora 6 10/24/2006
- Fedora 7 05/31/2007
- Fedora 8 11/08/2007
- Fedora 9 05/13/2008
- SUSE 10.1 05/11/2006
- SUSE 10.2 12/07/2006
- SUSE 10.3 10/04/2007
- SUSE 11.0 06/19/2008
- Ubuntu 6.06 06/01/2006
- Ubuntu 6.10 10/26/2006

- Ubuntu 7.04 04/19/2007
- Ubuntu 7.10 10/18/2007
- Ubuntu 8.04 04/24/2008
- Ubuntu 8.10 10/30/2008

Bueno, sobre todo funcionó. El problema ocurre en la clasificación de los números de versión de Fedora. Debido a que 1 viene antes que 5 en el conjunto de caracteres, la versión 10 termina en la parte superior mientras que la versión 9 cae en la parte inferior.

Para solucionar este problema, tendremos que ordenar por varias claves. Queremos realizar una ordenación alfabética en el primer campo y luego una ordenación numérica en el segundo campo. sort permite múltiples instancias de la opción -k para que se puedan especificar múltiples claves de ordenación. De hecho, una clave puede incluir una variedad de campos. Si no se especifica ningún rango (como ha sido el caso con nuestros ejemplos anteriores), sort usa una clave que comienza con el campo especificado y se extiende hasta el final de la línea. Aquí está la sintaxis para nuestro tipo de múltiples claves:

```
[~]$ sort --key=1,1 --key=2n distros.txt
```

Aunque usamos la forma larga de la opción para mayor claridad, -k 1,1 -k 2n sería exactamente equivalente. En la primera instancia de la opción de clave, especificamos un rango de campos para incluir en la primera clave. Como queríamos limitar la clasificación al primer campo, especificamos 1,1, que significa "empezar en el campo 1 y terminar en el campo 1". En la segunda instancia, especificamos 2n, lo que significa que el campo 2 es la clave de clasificación y que la clasificación debe ser numérica. Se puede incluir una letra de opción al final de un especificador de clave para indicar el tipo de clasificación que se realizará. Estas letras de opción son las mismas que las opciones globales para el programa de clasificación: b (ignora los espacios en blanco a la izquierda), n (ordenación numérica), r (ordenación inversa), etc.

El tercer campo de nuestra lista contiene una fecha en un formato inconveniente para ordenar. En las computadoras, las fechas generalmente se formatean en el orden Y Y Y Y-MM-DD para facilitar la clasificación cronológica, pero las nuestras están en el formato americano de MM/DD/YYYY. ¿Cómo podemos ordenar esta lista en orden cronológico?

Afortunadamente, sort proporciona una forma. La opción clave permite la especificación de compensaciones (offsets) dentro de los campos (fiels), por lo que podemos definir claves dentro de los campos.

```
[~]$ sort -k 3.7nbr -k 3.1nbr -k 3.4nbr distros.txt
```

Al especificar -k 3.7, le indicamos a sort que use una clave de ordenación que comience en el séptimo carácter dentro del tercer campo, que corresponde al comienzo del año. Asimismo, especificamos -k 3.1 y -k 3.4 para aislar las porciones de mes y día de la fecha. También agregamos las opciones nyr para lograr una ordenación numérica inversa. La opción b se incluye para suprimir los espacios iniciales (cuyos números varían de una línea a otra, lo que afecta el resultado de la clasificación) en el campo de fecha.

Algunos archivos no utilizan tabulaciones ni espacios como delimitadores de campo; por ejemplo, aquí está el archivo /etc/passwd:

```
[~]$ head /etc/passwd
```

Los campos de este archivo están delimitados con dos puntos (😃, entonces, ¿cómo ordenaríamos este archivo usando un campo clave? sort proporciona la opción -t para definir el carácter separador de campo. Para ordenar el archivo passwd en el séptimo campo (el shell predeterminado de la cuenta), podríamos hacer esto:

```
[~]$ sort -t ':' -k 7 /etc/passwd | head
```

Al especificar el carácter de dos puntos como separador de campo, podemos ordenar en el séptimo campo.

uniq

En comparación con la clasificación, el programa uniq es liviano. uniq realiza una tarea aparentemente trivial. Cuando se le da un archivo ordenado (o entrada estándar), elimina las líneas duplicadas y envía los resultados a la salida estándar. A menudo se usa junto con sort para limpiar la salida de duplicados.

> consejo:
>
> > Si bien uniq es una herramienta tradicional de Unix que se usa a menudo con sort, la versión GNU de sort admite una opción -u, que elimina los duplicados de la salida ordenada.

Hagamos un archivo de texto para probar esto.

```
[~]$ cat > foo.txt
```

```
a
b
c
a
b
c
```

Recuerde escribir ctrl-D para terminar la entrada estándar. Ahora, si ejecutamos uniq en nuestro archivo de texto, obtenemos esto:

```
[~]$ uniq foo.txt
```

```
a
b
```

```
c
a
b
c
```

Los resultados no son diferentes de nuestro archivo original; los duplicados no se eliminaron. Para que uniq haga su trabajo, la entrada debe ordenarse primero.

```
[~]$ sort foo.txt | uniq
```

```
a
b
c
```

Esto se debe a que uniq solo elimina las líneas duplicadas adyacentes entre sí. uniq tiene varias opciones.

La tabla siguiente enumera los más comunes.

> Opciones comunes de uniq

Opción	Opción Larga	Descripción
-c	--count	Genere una lista de líneas duplicadas precedidas por el número de veces que aparece la línea.
-d	--repeated	Genere solo líneas repetidas, en lugar de líneas únicas.
-f n	--skip-fields=n	Ignore n campos iniciales en cada línea. Los campos están separados por espacios en blanco, ya que están ordenados; sin embargo, a diferencia de sort, uniq no tiene ninguna opción para establecer un separador de campo alternativo.
-i	--ignore-case	Ignore el caso durante las comparaciones de línea.
-s n	--skip-chars=n	Omita (ignore) los n caracteres iniciales de cada línea.
-u	--unique	Salida solo líneas únicas. Las líneas con duplicados se ignoran.

Aquí vemos uniq usado para informar el número de duplicados encontrados en nuestro archivo de texto, usando la opción -c:

```
[~]$ sort foo.txt | uniq -c
```

```
2 a
2 b
2 c
```

Rebanando y cortando en cubitos (slicing and dicing)

Los siguientes tres programas que analizaremos se utilizan para extraer columnas de texto de archivos y recombinarlos de manera útil.

cut: eliminar secciones de cada línea de archivos

El programa **cut** se utiliza para extraer una sección de texto de una línea y enviar la sección extraída a la salida estándar. Puede aceptar múltiples argumentos de archivo de entrada desde entrada estándar. Especificar la sección de la línea que se extraerá es algo incómodo y se especifica mediante las opciones enumeradas en la Tabla siguiente.

> **cut:** Opciones de selección

Opción	Opción Larga	Descripción
-c list	--characters=list	Extraiga la parte de la línea definida por lista. La lista puede constar de uno o más rangos numéricos separados por comas.
-f list	--fields=list	Extraiga uno o más campos de la línea según lo definido por la lista. La lista puede contener uno o más campos o rangos de campos separados por comas.
-d delim	--delimiter=delim	Cuando se especifica -f, use delim como el carácter delimitador de campo. De forma predeterminada, los campos deben estar separados por un único carácter de tabulación.
-d delim	--complement	Extraiga toda la línea de texto, excepto las partes especificadas por -c y/o -f.

Como podemos ver, la forma en que Cut extrae el texto es bastante inflexible. cut se utiliza mejor para extraer texto de archivos producidos por otros programas, en lugar de texto escrito directamente por humanos. Echaremos un vistazo a nuestro archivo distros.txt para ver si está lo suficientemente "limpio" como para ser una buena muestra para nuestros ejemplos de corte. Si usamos cat con la opción -A, podemos ver si el archivo cumple con nuestros requisitos de campos separados por tabulaciones.

```
[~]$ cat -A distros.txt
```

```
SUSE^I10.2^I12/07/2006$
Fedora^I10^I11/25/2008$
SUSE^I11.0^I06/19/2008$
```

```
Ubuntu^I8.04^I04/24/2008$
Fedora^I8^I11/08/2007$
SUSE^I10.3^I10/04/2007$
Ubuntu^I6.10^I10/26/2006$
Fedora^I7^I05/31/2007$
Ubuntu^I7.10^I10/18/2007$
Ubuntu^I7.04^I04/19/2007$
SUSE^I10.1^I05/11/2006$
Fedora^I6^I10/24/2006$
Fedora^I9^I05/13/2008$
...
```

Se ve bien. No hay espacios incrustados, solo caracteres de tabulación entre los campos. Debido a que el archivo usa tabulaciones en lugar de espacios, usaremos la opción -f para extraer un campo.

```
[~]$ cut -f 3 distros.txt
```

```
12/07/2006
11/25/2008
...
```

Debido a que nuestro archivo de distribución está delimitado por tabulaciones, es mejor usar cortar para extraer campos en lugar de caracteres. Esto se debe a que cuando se elimina la pestaña de un archivo, es poco probable que cada línea contenga el mismo número de caracteres, lo que dificulta o imposibilita el cálculo de las posiciones de los caracteres dentro de la línea. Sin embargo, en nuestro ejemplo anterior, ahora hemos extraído un campo que, afortunadamente, contiene datos de idéntica longitud, por lo que podemos mostrar cómo funciona la extracción de caracteres extrayendo el año de cada línea.

```
[~]$ cut -f 3 distros.txt | cut -c 7-10
```

```
2006
2008
2008
```

Al ejecutar cut por segunda vez en nuestra lista, podemos extraer las posiciones de los caracteres 7 a 10, que corresponden al año en nuestro campo de fecha. La notación 7-10 es un ejemplo de rango. La página de manual de cut contiene una descripción completa de cómo se pueden especificar los rangos. Al trabajar con campos, es posible especificar un delimitador de campo diferente en lugar del carácter de tabulación. Aquí extraeremos el primer campo del archivo /etc/passwd:

```
[~]$ cut -d ':' -f 1 /etc/passwd | head
```

```
root
daemon
bin
sys
...
```

Usando la opción -d, podemos especificar el carácter de dos puntos como delimitador de campo.

> Pestañas desplegables

> Nuestro archivo distros.txt tiene el formato ideal para extraer campos utilizando cut. Pero, ¿y si quisiéramos un archivo que pudiera manipularse por completo con cortes por caracteres, en lugar de campos? Esto requeriría que reemplacemos los caracteres de tabulación dentro del archivo con el número correspondiente de espacios. Afortunadamente, el paquete GNU Coreutils incluye una herramienta para eso. Este programa, denominado expandir, acepta uno o más argumentos de archivo o entrada estándar y envía el texto modificado a la salida estándar. Si procesamos nuestro archivo distros.txt con expand, podemos usar cut -c para extraer cualquier rango de caracteres del archivo. Por ejemplo, podríamos usar el siguiente comando para extraer el año de publicación de nuestra lista expandiendo el archivo y usando cortar para extraer cada carácter desde la posición 23 hasta el final de la línea:

> ---

> ```
> [~]$ expand distros.txt | cut -c 23-
> ```

> ---

> Coreutils también proporciona el programa no expandido para sustituir espacios por tabulaciones.

---

paste (pegar): fusionar líneas de archivos

El comando paste hace lo contrario de cortar. En lugar de extraer una columna de texto de un archivo, agrega una o más columnas de texto a un archivo. Para ello, lee varios archivos y combina los campos que se encuentran en cada archivo en una única secuencia en la salida estándar. Al igual que cortar, pegar acepta múltiples argumentos de de archivos y/o entrada estándar. Para demostrar cómo funciona el pegado, realizaremos algunas operaciones en nuestro archivo **distros.txt** para producir una lista cronológica de lanzamientos. De nuestro trabajo anterior con sort, primero produciremos una lista de distribuciones ordenadas por fecha y almacenaremos el resultado en un archivo llamado **distros-by-date.txt.**

```
[~]$ sort -k 3.7nbr -k 3.1nbr -k 3.4nbr distros.txt > distros-by-date.txt
```

A continuación, usaremos cut para extraer los dos primeros campos del archivo (el nombre de la distribución y la versión) y almacenaremos ese resultado en un archivo llamado ***distro-versions.txt.***

```
[~]$ cut -f 1,2 distros-by-date.txt > distros-versions.txt
```

```
[~]$ head distros-versions.txt
```

```
Fedora 10
Ubuntu 8.10
SUSE 11.
....
```

La última pieza de preparación es extraer las fechas de lanzamiento y almacenarlas en un archivo llamado ***distro-fechas.txt.***

```
[~]$ cut -f 3 distros-by-date.txt > distros-dates.txt

[~]$ head distros-dates.txt
```

```
11/25/2008
10/30/2008
06/19/2008
...
```

Ahora tenemos las piezas que necesitamos. Para completar el proceso, use ***paste*** para poner la columna de fechas antes de los nombres y versiones de la distribución, creando así una lista cronológica. Esto se hace simplemente usando pegar y ordenando sus argumentos en la disposición deseada.

```
[~]$ paste distros-dates.txt distros-versions.txt
```

```
11/25/2008 Fedora 10
10/30/2008 Ubuntu 8.10
...
```

---

join: une líneas de dos archivos en un campo común

De alguna manera, unir es como pegar en el sentido de que agrega columnas a un archivo, pero utiliza una forma única de hacerlo. Una unión **join** es una operación generalmente asociada con bases de datos relacionales donde los datos de varias tablas con un campo de clave compartido se combinan para formar un resultado deseado. El programa de combinación realiza la misma operación. Une datos de varios archivos basándose en un campo de clave compartida. Para ver cómo se usa una operación de unión en una base de datos relacional, imaginemos una pequeña base de datos que consta de dos tablas, cada una de las cuales contiene un solo registro. La primera tabla, denominada CLIENTES --CUSTOMERS--, tiene tres campos: un número de cliente (CUSTNUM), el nombre del cliente (FNAME) y el apellido del cliente (LNAME):

```
    CUSTOMERS
CUSTNUM FNAME LNAME
======== ===== ======
4681934 John Smith
```

La segunda tabla se llama PEDIDOS - ORDERS - y contiene cuatro campos: un número de pedido (ORDERNUM), el número de cliente (CUSTNUM), la cantidad (QUAN) y el artículo pedido (ITEM).

```
              ORDERS
ORDERNUM     CUSTNUM    QUAN    ITEM
========     =======    ====    ====
3014953305   4681934     1      Blue Widget
```

Tenga en cuenta que ambas tablas comparten el campo CUSTNUM. Esto es importante porque permite una relación entre las tablas. Realizar una operación de unión nos permitiría combinar los campos en las dos tablas para lograr un resultado útil, como preparar una factura. Usando los valores coincidentes en los campos CUSTNUM de ambas tablas, una operación de combinación podría producir lo siguiente:

```
FNAME LNAME QUAN ITEM
===== ===== ==== ====
John  Smith  1   Blue Widget
```

Para demostrar el programa de unión, necesitaremos crear un par de archivos con una clave compartida. Para hacer esto, usaremos nuestro archivo distros-by-date.txt. A partir de este archivo, construiremos dos archivos adicionales. Uno contiene las fechas de lanzamiento (que será nuestra clave compartida para esta demostración) y los nombres de lanzamiento, como se muestra aquí:

```
[~]$ cut -f 1,1 distros-by-date.txt > distros-names.txt

[~]$ paste distros-dates.txt distros-names.txt > distros-key-names.txt

[~]$ head distros-key-names.txt
```

```
11/25/2008 Fedora
10/30/2008 Ubuntu
....
```

El segundo archivo contiene las fechas de lanzamiento y los números de versión, como se muestra aquí.

```
[~]$ cut -f 2,2 distros-by-date.txt > distros-vernums.txt

[~]$ paste distros-dates.txt distros-vernums.txt > distros-key-vernums.txt

[~]$ head distros-key-vernums.txt
```

```
11/25/2008 10
10/30/2008 8.10
....
```

Ahora tenemos dos archivos con una clave compartida (el campo "fecha de publicación"). Es importante señalar que los archivos deben estar ordenados en el campo clave para que la combinación funcione correctamente.

```
[~]$ join distros-key-names.txt distros-key-vernums.txt | head
```

```
11/25/2008 Fedora 10
10/30/2008 Ubuntu 8.10
....
```

Tenga en cuenta también que, de forma predeterminada, join usa espacios en blanco como delimitador del campo de entrada y un solo espacio como delimitador del campo de salida. Este comportamiento se puede modificar especificando opciones. Consulte la página del manual de unión para obtener más detalles.

Comparar texto

Suele ser útil comparar versiones de archivos de texto. Para los administradores de sistemas y desarrolladores de software, esto es particularmente importante. Un administrador del sistema puede, por ejemplo, necesitar comparar un archivo de configuración existente con una versión anterior para diagnosticar un problema del sistema. Del mismo modo, un programador con frecuencia necesita ver qué cambios se han realizado en los programas a lo largo del tiempo.

comm: comparar dos archivos ordenados línea por línea

El programa de comunicaciones compara dos archivos de texto y muestra las líneas que son únicas para cada uno y las líneas que tienen en común. Para demostrarlo, crearemos dos archivos de texto casi idénticos usando cat.

```
[~]$ cat > file1.txt
```

```
a
b
c
```

```
[~]$ cat > file2.txt
```

```
b
c
d
```

Ahora compramos los dos ficheros con comm.

```
[~]$ comm file1.txt file2.txt
```

Como podemos ver, comm produce tres columnas de salida. La primera columna contiene líneas exclusivas del primer argumento de archivo, la segunda columna contiene las líneas exclusivas del segundo argumento de archivo y la tercera columna contiene las líneas compartidas por ambos archivos. comm admite opciones en la forma -n, donde n es 1, 2 o 3. Cuando se utilizan, estas opciones especifican qué columnas suprimir. Por ejemplo, si quisiéramos generar solo las líneas compartidas por ambos archivos, suprimiremos la salida de la primera y la segunda columna.

```
[~]$ comm -12 file1.txt file2.txt
```

---

diff — comparar archivos linea por linea.

Al igual que el programa comm, **diff** se utiliza para detectar las diferencias entre archivos. Sin embargo, **diff** es una herramienta mucho más compleja, que admite muchos formatos de salida y la capacidad de procesar grandes colecciones de archivos de texto a la vez. Los desarrolladores de software suelen utilizar **diff** para examinar cambios entre diferentes versiones del código fuente del programa y, por lo tanto, tiene la capacidad de examinar de forma recursiva directorios de código fuente, a menudo denominados árboles de fuentes. Un uso común de **diff** es la creación de archivos **diff** o **patches** que son utilizados por programas como **patch** (que discutiremos en breve) para convertir una versión de un archivo (o archivos) en otra versión.

Si usamos **diff** para ver nuestro archivos del ejemplo anterior:

```
[~]$ diff file1.txt file2.txt
```

```
1d0
< a
```

```
4a4
> e
```

---

vemos su estilo de salida predeterminado: una descripción concisa de las diferencias entre los dos archivos. En el formato predeterminado, cada grupo de cambios está precedido por un comando de cambio en forma de rango de operación de rango para describir las posiciones y los tipos de cambios necesarios para convertir el primer archivo en el segundo archivo, como se describe en la Tabla siguiente.

> Comandos cambiar de diff

Cambio	Descripción
r1ar2	Agregue **- Add -** las líneas en la posición r2 en el segundo archivo a la posición r1 en el primer archivo.
r1cr2	Cambie **- Change -** (reemplace) las líneas en la posición r1 con las líneas en la posición r2 en el segundo archivo.
r1dr2	Elimine **- Delete -** las líneas en el primer archivo en la posición r1, que habrían aparecido en el rango r2 en el segundo archivo

En este formato, un rango es una lista separada por comas de la línea inicial y la línea final. Si bien este formato es el predeterminado (principalmente para el cumplimiento de POSIX y la compatibilidad con versiones anteriores de diff de Unix tradicionales), no se usa tan ampliamente como otros formatos opcionales. Dos de los formatos más populares son el formato de contexto y el formato unificado.

Cuando se ve usando el formato de contexto (la opción -c), veremos esto:

```
[~]$ diff -c file1.txt file2.txt
```

```
*** file1.txt
2008-12-23 06:40:13.000000000 -0500
--- file2.txt
2008-12-23 06:40:34.000000000 -0500
***************
*** 1,4 ****
- a
b
c
d
--- 1,4 ----
b
c
d
+e
```

La salida comienza con los nombres de los dos archivos y sus marcas de tiempo. El primer archivo está marcado con asteriscos y el segundo archivo está marcado con guiones. En el resto de la lista, estos marcadores significarán sus respectivos archivos. A continuación, vemos grupos de cambios, incluido el número predeterminado de líneas de contexto circundantes. En el primer grupo, vemos esto:

```
*** 1,4 ***
```

que indica las líneas 1 a 4 en el primer archivo. Más tarde vemos esto:

```
--- 1,4 ---
```

que indica las líneas 1 a 4 en el segundo archivo. Dentro de un grupo de cambio, las líneas comienzan con uno de los cuatro indicadores descritos en la Tabla siguiente

> Indicadores de cambio de formato de contexto **diff**

Indicator	Meaning
blank	Se muestra una línea para el contexto. No indica una diferencia entre los dos archivos.
-	Una línea eliminada. Esta línea aparecerá en el primer archivo pero no en el segundo.
+	Añadida una línea. Esta línea aparecerá en el segundo archivo pero no en el primer archivo.
!	Una línea cambió. Se mostrarán las dos versiones de la línea, cada una en su sección respectiva del grupo de cambios.

El formato unificado es similar al formato de contexto pero es más conciso. Se especifica con la opción -u.

```
[~]$ diff -u file1.txt file2.txt
```

```
--- file1.txt
2008-12-23 06:40:13.000000000 -0500
+++ file2.txt
2008-12-23 06:40:34.000000000 -0500
@@ -1,4 +1,4 @@
-a
b
c
d
+e
```

La diferencia más notable entre el contexto y los formatos unificados es la eliminación de las líneas de contexto duplicadas, lo que hace que los resultados del formato unificado sean más cortos que los del formato de contexto. En nuestro ejemplo anterior, vemos marcas de tiempo de archivos como las del formato de contexto, seguidas de la cadena @@ -1,4 +1,4 @@. Esto indica las líneas en el primer archivo y las líneas en el segundo archivo descritas en el grupo de cambios. A continuación se encuentran las líneas en sí, con las tres líneas de contexto predeterminadas. Cada línea comienza con uno de los tres caracteres posibles descritos en la Tabla 20-6.

> indicadores de cambios de formato unificado ***diff**

Caracter	significado
blank	Esta línea es compartida por ambos archivos.
-	Esta línea se eliminó del primer archivo.
+	Esta línea se agregó al primer archivo.

patch: aplique una diferencia a un original

El programa de patch se utiliza para aplicar cambios a archivos de texto. Acepta la salida de diff y generalmente se usa para convertir archivos de versiones anteriores en versiones más nuevas. Consideremos un ejemplo famoso. El kernel de Linux es desarrollado por un equipo grande y poco organizado de colaboradores que envían un flujo constante de pequeños cambios en el código fuente. El kernel de Linux consta de varios millones de líneas de código, mientras que los cambios que realiza un colaborador a la vez son bastante pequeños. No tiene sentido que un colaborador envíe a cada desarrollador un árbol completo de fuentes del kernel cada vez que se realiza un pequeño cambio.

En su lugar, se envía un archivo de diferencias. El ***archivo diff*** contiene el cambio de la versión anterior del kernel a la nueva versión con los cambios del colaborador. El receptor luego utiliza el programa de ***patch*** para aplicar el cambio a su propio árbol de fuentes. El uso de ***diff/patch*** ofrece dos ventajas importantes.

> El ***archivo diff*** es pequeño, comparado con el tamaño completo del árbol de origen.

> El ***archivo diff*** muestra de forma concisa el cambio que se está realizando, lo que permite a los revisores del parche evaluarlo rápidamente.

Por supuesto, ***diff/patch*** funcionará en cualquier archivo de texto, no solo en el código fuente. Sería igualmente aplicable a archivos de configuración o cualquier otro texto.

Para preparar un ***archivo diff*** para usar con el parche, la documentación de GNU, sugiere usar diff de la siguiente manera:

```
diff -Naur old_file new_file > diff_file
```

donde old_file y new_file son archivos individuales o directorios que contienen archivos. La opción r admite la recursividad de un árbol de directorios.

Una vez que se ha creado el archivo diff, podemos aplicarlo para parchear el archivo antiguo en el nuevo archivo.

```
patch < diff_file
```

Lo demostraremos con nuestro archivo de prueba.

```
[~]$ diff -Naur file1.txt file2.txt > patchfile.txt
```

```
[~]$ patch < patchfile.txt
```

```
patching file file1.txt
```

```
[~]$ cat file1.txt
```

```
b
c
d
e
```

En este ejemplo, creamos un archivo diff llamado patchfile.txt y luego usamos el programa de parche para aplicar el parche. Tenga en cuenta que no tuvimos que especificar un archivo de destino para parchear, ya que el archivo diff (en formato unificado) ya contiene los nombres de archivo en el encabezado. Una vez que se aplica el parche, podemos ver que file1.txt ahora coincide con file2.txt.

patch tiene una gran cantidad de opciones, y hay programas de utilidad adicionales que se pueden usar para analizar y editar parches.

Edición sobre la marcha

Nuestra experiencia con los editores de texto ha sido en gran parte interactiva, lo que significa que movemos manualmente un cursor y luego escribimos nuestros cambios. Sin embargo, también existen formas no interactivas de editar texto. Es posible, por ejemplo, aplicar un conjunto de cambios a varios archivos con un solo comando.

tr: transliterar o eliminar caracteres

El programa tr se utiliza para transliterar caracteres. Podemos pensar en esto como una especie de operación de búsqueda y reemplazo basada en caracteres. La transliteración es el proceso de cambiar caracteres de un alfabeto a otro. Por ejemplo, convertir caracteres de minúsculas a mayúsculas es transliteración. Podemos realizar dicha conversión con tr de la siguiente manera:

```
[~]$ echo "lowercase letters" | tr a-z A-Z
```

```
LOWERCASE LETTERS
```

Como podemos ver, tr opera en una entrada estándar y genera sus resultados en una salida estándar. tr acepta dos argumentos: un conjunto de caracteres para convertir y un conjunto correspondiente de caracteres para convertir. Los juegos de caracteres se pueden expresar de tres formas.

> Una lista enumerada. Por ejemplo, ABCDEFGHIJKLMNOPQRSTUVWXYZ.

> Una gama de caracteres. Por ejemplo, A-Z. Tenga en cuenta que este método a veces está sujeto a los mismos problemas que otros comandos debido al orden de clasificación de la configuración regional y, por lo tanto, debe usarse con precaución.

> Clases de caracteres POSIX. Por ejemplo, [:upper:].

En la mayoría de los casos, ambos juegos de caracteres deben tener la misma longitud; sin embargo, es posible que el primer conjunto sea más grande que el segundo, especialmente si queremos convertir varios caracteres en un solo carácter.

```
[~]$ echo "lowercase letters" | tr [:lower:] A
```

```
AAAAAAAAA AAAAAAA
```

Además de la transliteración, tr permite que los caracteres simplemente se eliminen del flujo de entrada. Anteriormente en este capítulo, discutimos el problema de convertir archivos de texto MS-DOS a texto estilo Unix. Para realizar esta conversión, los caracteres de retorno de carro deben eliminarse del final de cada línea.

Esto se puede realizar con tr de la siguiente manera:

```
tr -d '\r' < dos_file > unix_file
```

---

donde dos_file es el archivo a convertir y unix_file es el resultado. Esta forma del comando usa la secuencia de escape \r para representar el carácter de retorno de carro. Para ver una lista completa de las secuencias y clases de caracteres que admite tr, intente lo siguiente:

```
[~]$ tr --help
```

---

tr también puede realizar otro truco. Usando la opción -s, tr puede “apretar” (borrar) instancias repetidas de un carácter.

```
[~]$ echo "aaabbbccc" | tr -s ab
```

```
abccc
```

Aquí tenemos una cadena que contiene caracteres repetidos. Al especificar el conjunto ab a tr, eliminamos las instancias repetidas de las letras en el conjunto, mientras dejamos el carácter que falta en el conjunto (c) sin cambios.

Tenga en cuenta que los caracteres que se repiten deben ser contiguos. Si no es así, la compresión no tendrá ningún efecto.

```
[~]$ echo "abcabcabc" | tr -s ab
```

```
abcabcabc
```

ROT13: The Not-So-Secret Decoder Ring

Un uso divertido de tr es realizar la codificación de texto ROT13. ROT13 es un tipo de cifrado trivial basado en un cifrado de sustitución simple. Llamar al cifrado ROT13 está siendo generoso; la ofuscación del texto es más precisa. A veces se utiliza en texto para ocultar contenido potencialmente ofensivo. El método simplemente mueve cada carácter 13 lugares hacia arriba en el alfabeto. Debido a que está a la mitad de los 26 caracteres posibles, al realizar el algoritmo por segunda vez en el texto, se restaura a su forma original. Utilice lo siguiente para realizar esta codificación con tr:

```
echo "secret text" | tr a-zA-Z n-za-mN-ZA-M
```

```
frperg grkg
```

Realizar el mismo procedimiento por segunda vez da como resultado la siguiente traducción:

```
echo "frperg grkg" | tr a-zA-Z n-za-mN-ZA-M
```

```
secret text
```

Varios programas de correo electrónico y lectores de noticias de Usenet admiten la codificación ROT13. Wikipedia contiene un buen artículo sobre el tema en http://en.wikipedia.org/wiki/ROT13.

---

sed — Stream Editor para filtrar y transformar texto

El nombre sed es la abreviatura de editor de secuencias. Realiza la edición de texto en una secuencia de texto, ya sea un conjunto de archivos especificados o una entrada estándar. sed es un programa poderoso y algo complejo (hay libros completos al respecto), por lo que no lo cubriremos completamente aquí.

En general, la forma en que funciona sed es que se le da un solo comando de edición (en la línea de comando) o el nombre de un archivo de secuencia de comandos que contiene varios comandos, y luego ejecuta estos comandos en cada línea del flujo de texto. Aquí hay un ejemplo simple de sed en acción:

```
[~]$ echo "front" | sed 's/front/back/'
```

```
back
```

En este ejemplo, producimos un flujo de texto de una palabra usando echo y lo canalizamos a sed. sed, a su vez, lleva a cabo la instrucción s/front/back/ sobre el texto en la secuencia y produce la salida como resultado. También podemos reconocer este comando como parecido al comando de “sustitución” (buscar y reemplazar) en vi.

Los comandos en sed comienzan con una sola letra. En el ejemplo anterior, el comando de sustitución está representado por la letra s y va seguido de las cadenas de búsqueda y reemplazo, separadas por el carácter de barra (slash) como delimitador. La elección del carácter delimitador es arbitraria. Por convención, el carácter de barra se utiliza a menudo, pero sed aceptará cualquier carácter que siga inmediatamente al comando como delimitador. Podríamos realizar el mismo comando de esta manera:

```
[~]$ echo "front" | sed 's_front_back_'
```

```
back
```

Al usar el carácter de subrayado inmediatamente después del comando, se convierte en el delimitador. La capacidad de establecer el delimitador se puede utilizar para hacer que los comandos sean más legibles, como veremos.

La mayoría de los comandos en sed pueden ir precedidos de una dirección, que especifica qué línea (s) del flujo de entrada se editarán. Si se omite la dirección, el comando de edición se ejecuta en cada línea del flujo de entrada.

La forma más simple de dirección es un número de línea. Podemos agregar uno a nuestro ejemplo.

```
[~]$ echo "front" | sed '1s/front/back/'
```

```
back
```

Agregar la dirección 1 a nuestro comando hace que nuestra sustitución se realice en la primera línea de nuestro flujo de entrada de una línea. Si especificamos otro número, vemos que no se realiza la edición ya que nuestro flujo de entrada no tiene línea 2.

```
[~]$ echo "front" | sed '2s/front/back/'
```

```
front
```

Las direcciones pueden expresarse de muchas formas. La tabla siguiente enumera los más comunes

Tabla 20-7: Notación de dirección sed

Direccion (Address)	Descripción
n	Un número de línea donde n es un número entero positivo.
$	La última línea.
/regexp/	Líneas que coinciden con una expresión regular básica POSIX. Tenga en cuenta que la expresión regular está delimitada por caracteres de barra. Opcionalmente, la expresión regular puede estar delimitada por un carácter alternativo, especificando la expresión con \ cregexpc, donde c es el carácter alternativo.
addr1,addr2	Un rango de líneas desde addr1 a addr2, inclusive. Las direcciones pueden ser cualquiera de los formularios de direcciones individuales enumerados anteriormente.
first~step	Haga coincidir la línea representada por el número primero y luego cada línea subsiguiente a intervalos de paso. Por ejemplo, 1 ~ 2 se refiere a cada línea impar y 5 ~ 5 se refiere a la quinta línea y cada quinta línea a partir de entonces.
addr1,+n	Haga coincidir addr1 y las siguientes n líneas.
addr!	Coincidir con todas las líneas excepto addr, que puede ser cualquiera de los formularios enumerados anteriormente.

Demostraremos diferentes tipos de direcciones usando el archivo distros.txt de antes en este capítulo. Primero, aquí hay un rango de números de línea:

```
[~]$ sed -n '1,5p' distros.txt
```

```
SUSE 10.2 12/07/2006
Fedora 10 11/25/2008
SUSE 11.0 06/19/2008
Ubuntu 8.04 04/24/2008
Fedora 8 11/08/2007
```

En este ejemplo, imprimimos un rango de líneas, comenzando con la línea 1 y continuando hasta la línea 5. Para hacer esto, usamos el comando p, que simplemente causa que se imprima una línea coincidente. Sin embargo, para que esto sea efectivo, debemos incluir la opción -n (la opción “no auto-print”) para hacer que sed no imprima todas las líneas por defecto.

A continuación, probaremos una expresión regular.

```
[~]$ sed -n '/SUSE/p' distros.txt
```

```
SUSE 10.2 12/07/2006
SUSE 11.0 06/19/2008
SUSE 10.3 10/04/2007
SUSE 10.1 05/11/2006
```

Al incluir la expresión regular delimitada por barra inclinada / SUSE /, podemos aislar las líneas que la contienen de la misma manera que grep.

Finalmente, intentaremos la negación agregando un signo de exclamación (!) A la dirección.

```
[~]$ sed -n '/SUSE/!p' distros.txt
```

```
Fedora 10 11/25/2008
Ubuntu 8.04 04/24/2008
Fedora 8 11/08/2007
Ubuntu 6.10 10/26/2006
Fedora 7    05/31/2007
Ubuntu 7.10 10/18/2007
Ubuntu 7.04 04/19/2007
Fedora 6    10/24/2006
Fedora 9    05/13/2008
Ubuntu 6.06 06/01/2006
Ubuntu 8.10 10/30/2008
Fedora 5    03/20/2006
```

Aquí vemos el resultado esperado: todas las líneas del archivo excepto las que coinciden con la expresión regular. Hasta ahora, hemos analizado dos de los comandos de edición sed, s y p.

La Tabla 20-8 proporciona una lista más completa de los comandos de edición básicos.

> Tabla 20-8: Comandos de edición básicos de sed

Comandos	Descripción
=	Muestra el número de línea actual.
a	Agrega texto después de la línea actual.
d	Elimina la línea actual.
i	Inserta texto delante de la línea actual.
p	Imprime la línea actual. De forma predeterminada, sed imprime todas las líneas y solo edita las líneas que coinciden con una dirección específica dentro del archivo. El comportamiento predeterminado se puede anular especificando la opción -n.
q	Salga de sed sin procesar más líneas. Si no se especifica la opción -n, genera la línea actual.
Q	Salga de sed sin procesar más líneas.
s/regexp/replacement/	Sustituya el contenido de reemplazo dondequiera que se encuentre regexp. el reemplazo puede incluir el carácter especial &, que es equivalente al texto que coincide con regexp. Además, el reemplazo puede incluir las secuencias \1 a \9, que son el contenido de las subexpresiones correspondientes en regexp. Para obtener más información sobre esto, consulte la siguiente discusión sobre referencias anteriores. Después de la barra diagonal posterior al reemplazo, se puede especificar un indicador opcional para modificar el comportamiento del comando s.
y/set1/set2	Realice la transliteración convirtiendo los caracteres de set1 en los caracteres correspondientes en set2. Tenga en cuenta que, a diferencia de tr, sed requiere que ambos conjuntos tengan la misma longitud.

El comando s es, con mucho, el comando de edición más utilizado.

Demostraremos solo parte de su poder al realizar una edición en nuestro archivo distros.txt. Hablamos anteriormente sobre cómo el campo de fecha en distros.txt no estaba en un formato "compatible con la computadora". Si bien la fecha tiene el formato MM/DD/AAAA, sería mejor (para facilitar la clasificación) si el formato fuera AAAA-MM-DD.

Realizar este cambio en el archivo a mano llevaría mucho tiempo y sería propenso a errores, pero con sed, este cambio se puede realizar en un solo paso.

```
[~]$ sed 's/\([0-9]\{2\}\)\/\([0-9]\{2\}\)\/\([0-9]\{4\}\ )$/\3-\1-\2/'
distros.txt
```

```
SUSE 10.2 2006-12-07
Fedora 10 2008-11-25
SUSE 11.0 2008-06-19
Ubuntu 8.04 2008-04-24
```

```
Fedora 8    2007-11-08
SUSE 10.3   2007-10-04
Ubuntu 6.10 2006-10-26
Fedora 7    2007-05-31
....
```

¡Guauu! Ahora que es un comando de aspecto feo. Pero funciona. En un solo paso, hemos cambiado el formato de fecha en nuestro archivo. También es un ejemplo perfecto de por qué las expresiones regulares a veces se refieren en broma como un medio de solo escritura. Podemos escribirlos, pero a veces no podemos leerlos. Antes de que estemos tentados a huir aterrorizados de este comando, veamos cómo fue construido. Primero, sabemos que el comando tendrá esta estructura básica:

```
sed 's/regexp/replacement/' distros.txt
```

Nuestro siguiente paso es encontrar una expresión regular que aísle la fecha. Debido a que está en formato MM / DD / Y Y Y Y y aparece al final de la línea, podemos usar una expresión como esta:

```
[0-9]{2}/[0-9]{2}/[0-9]{4}$
```

Esto coincide con dos dígitos, una barra, dos dígitos, una barra, cuatro dígitos y el final de la línea. Eso se encarga de las expresiones regulares, pero ¿qué pasa con el reemplazo? Para manejar eso, debemos introducir una nueva función de expresión regular que aparece en algunas aplicaciones que usan BRE. Esta característica se llama referencias retrospectivas y funciona así: si la secuencia \ n aparece en reemplazo donde n es un número del 1 al 9, la secuencia se referirá a la subexpresión correspondiente en la expresión regular anterior. Para crear las subexpresiones, simplemente las encerramos entre paréntesis así:

```
([0-9]{2})/([0-9]{2})/([0-9]{4})$
```

Ahora tenemos tres subexpresiones. El primero contiene el mes, el segundo contiene el día del mes y el tercero contiene el año. Ahora podemos construir el reemplazo de la siguiente manera:

```
\3-\1-\2
```

Esto nos da el año, un guión, el mes, un guión y el día. Ahora, nuestro comando se ve así:

```
sed 's/([0-9]{2})/([0-9]{2})/([0-9]{4})$/\3-\1-\2/' distros.txt
```

Tenemos dos problemas pendientes. La primera es que las barras diagonales adicionales en nuestra expresión regular confundirán a sed cuando intente interpretar el comando s. La segunda es que debido a que sed, por defecto, acepta solo expresiones regulares básicas, varios de los caracteres de nuestra

expresión regular se tomarán como literales, en lugar de como metacaracteres. Podemos resolver ambos problemas con una aplicación liberal de barras invertidas para escapar de los personajes ofensivos.

```
sed 's/\([0-9]\{2\}\)\/\([0-9]\{2\}\)\/\([0-9]\{4\}\)$/\3-\1-\2/' distros.txt
```

¡Y ahí lo tienes!

Otra característica del comando s es el uso de indicadores opcionales que pueden seguir a la cadena de reemplazo. El más importante de ellos es el indicador g, que le indica a sed que aplique la búsqueda y reemplazo globalmente a una línea, no solo a la primera instancia, que es la predeterminada. Aquí hay un ejemplo:

```
[~]$ echo "aaabbbccc" | sed 's/b/B/'
```

```
aaaBbbccc
```

Vemos que se realizó el reemplazo, pero solo a la primera instancia de la letra b, mientras que las instancias restantes se dejaron sin cambios. Al agregar la bandera g, podemos cambiar todas las instancias.

```
[~]$ echo "aaabbbccc" | sed 's/b/B/g'
```

```
aaaBBBccc
```

Hasta ahora, solo hemos dado comandos individuales sed a través de la línea de comandos. También es posible construir comandos más complejos en un archivo de script usando la opción -f. Para demostrarlo, usaremos sed con nuestro archivo distros.txt para crear un informe. Nuestro informe incluirá un título en la parte superior, nuestras fechas de modificación y todos los nombres de distribución convertidos a mayúsculas. Para hacer esto, necesitaremos escribir una secuencia de comandos, por lo que activaremos nuestro editor de texto e ingresaremos lo siguiente:

```
# sed script to produce Linux distributions report

1 i\
\
Linux Distributions Report\

s/\([0-9]\{2\}\)\/\([0-9]\{2\}\)\/\([0-9]\{4\}\)$/\3-\1-\2/
y/abcdefghijklmnopqrstuvwxyz/ABCDEFGHIJKLMNOPQRSTUVWXYZ/
```

Guardaremos nuestro script sed como distros.sed y lo ejecutaremos así:

```
[~]$ sed -f distros.sed distros.txt
```

```
Linux Distributions Report
SUSE 10.2   2006-12-07
FEDORA 10   2008-11-25
SUSE 11.0   2008-06-19
UBUNTU 8.04 2008-04-24
FEDORA 8    2007-11-08
......
```

Como podemos ver, nuestro script produce los resultados deseados, pero ¿cómo lo hace? Echemos otro vistazo a nuestro guión. Usaremos gato para numerar las líneas.

```
[~]$ cat -n distros.sed
```

```
1 # sed script to produce Linux distributions report
2
3 1 i\
4 \
5 Linux Distributions Report\
6
7 s/\([0-9]\{2\}\)\/\([0-9]\{2\}\)\/\([0-9]\{4\}\)$/\3-\1-\2/
8 y/abcdefghijklmnopqrstuvwxyz/ABCDEFGHIJKLMNOPQRSTUVWXYZ/
```

La línea 1 de nuestro guión es un comentario. Como muchos archivos de configuración y lenguajes de programación en sistemas Linux, los comentarios comienzan con el carácter # y van seguidos de texto legible por humanos. Los comentarios se pueden colocar en cualquier lugar del script (aunque no dentro de los comandos en sí) y son útiles para cualquier persona que necesite identificar y / o mantener el script. La línea 2 es una línea en blanco. Al igual que los comentarios, se pueden agregar líneas en blanco para mejorar la legibilidad. Muchos comandos sed admiten direcciones de línea. Se utilizan para especificar sobre qué líneas de la entrada se actuará. Las direcciones de línea se pueden expresar como números de una sola línea, rangos de números de línea y el número de línea especial $, que indica la última línea de entrada. Las líneas 3, 4, 5 y 6 contienen texto que se insertará en la dirección 1, la primera línea de la entrada. El comando i va seguido de la secuencia de una barra invertida y luego un retorno de carro para producir un retorno de carro de escape, o lo que se llama un carácter de continuación de línea. Esta secuencia, que se puede utilizar en muchas circunstancias, incluidos los scripts de shell, permite incrustar un retorno de carro en un flujo de texto sin indicar al intérprete (en este caso sed) que se ha alcanzado el final de la línea. Los comandos i y a (que agrega texto, en lugar de insertarlo) yc (que reemplaza el texto) permiten múltiples líneas de texto siempre que cada línea, excepto la última, termine con un carácter de continuación de línea. La sexta línea de nuestro script es en realidad el final de nuestro texto insertado y

termina con un retorno de carro simple en lugar de un carácter de continuación de línea, lo que indica el final del comando i.

> ***Nota:*** Un carácter de continuación de línea está formado por una barra invertida seguida inmediatamente por un retorno de carro. No se permiten espacios intermedios.

La línea 7 es nuestro comando de búsqueda y reemplazo. Dado que no está precedido por una dirección, cada línea del flujo de entrada está sujeta a su acción.

La línea 8 realiza la transliteración de las letras minúsculas a letras mayúsculas. Tenga en cuenta que, a diferencia de tr, el comando y en sed no admite rangos de caracteres (por ejemplo, [a-z]), ni admite POSIX. Nuevamente, debido a que el comando y no está precedido por una dirección, se aplica a todas las líneas del flujo de entrada.

Gente a la que le gusta sed tanto como ...

> sed es un programa capaz, capaz de realizar tareas de edición bastante complejas en flujos de texto. Se utiliza con mayor frecuencia para tareas simples de una línea en lugar de scripts largos. Muchos usuarios prefieren otras herramientas para tareas más grandes. Los más populares son awk y perl. Estos van más allá de meras herramientas como los programas cubiertos aquí y se extienden al ámbito de los lenguajes de programación completos. Perl, en particular, se usa a menudo en lugar de scripts de shell para muchas tareas de administración y administración del sistema, además de ser un medio popular para el desarrollo web. awk es un poco más especializado. Su fortaleza específica es su capacidad para manipular datos tabulares. Se parece a sed en que los programas awk normalmente procesan archivos de texto línea por línea, utilizando un esquema similar al concepto sed de una dirección seguida de una acción. Aunque tanto awk como perl están fuera del alcance de este libro, son buenas habilidades para que las aprenda el usuario de la línea de comandos de Linux.

aspell — Corrector Ortográfico Interactivo (Interactive Spellchecker)

La última herramienta que veremos es aspell, un corrector ortográfico interactivo. El programa aspell es el sucesor de un programa anterior llamado ispell y se puede utilizar, en su mayor parte, como un reemplazo directo. Si bien el programa aspell es utilizado principalmente por otros programas que requieren la capacidad de corrección ortográfica, también se puede usar de manera efectiva como una herramienta independiente desde la línea de comandos. Tiene la capacidad de verificar de manera inteligente varios tipos de archivos de texto, incluidos documentos HTML, programas C o C ++, mensajes de correo electrónico y otros tipos de textos especializados.

Para revisar la ortografía de un archivo de texto que contiene prosa simple, podría usarse así:

```
aspell check textfile
```

donde archivo de texto es el nombre del archivo a verificar. Como ejemplo práctico, creemos un archivo de texto simple llamado foo.txt que contiene algunos errores ortográficos deliberados.

```
[~]$ cat > foo.txt
```

```
The quick brown fox jimped over the laxy dog.
```

A continuación, comprobaremos el archivo con aspell.

```
[~]$ aspell check foo.txt
```

Como aspell es interactivo en el modo de verificación, veremos una pantalla como esta.

```
The quick brown fox **jimped** over the laxy dog.

1) jumped  6) wimped
2) gimped  7) camped
3) comped  8) humped
4) limped  9) impede
5) pimped  0) umped
i) Ignore  I) Ignore all
r) Replace R) Replace all
a) Add     l) Add Lower
b) Abort   x) Exit

?
```

En la parte superior de la pantalla, vemos nuestro texto con una palabra deletreada sospechosamente resaltada. En el medio, vemos 10 sugerencias de ortografía numeradas del 0 al 9, seguidas de una lista de otras acciones posibles. Finalmente, en la parte inferior, vemos un mensaje listo para aceptar nuestra elección.

Si presionamos la tecla 1, aspell reemplaza la palabra ofrecida con la palabra resaltada y pasa a la siguiente palabra mal escrita, que es laxa. Si seleccionamos el reemplazo perezoso, aspell lo reemplaza y termina. Una vez que ha terminado un hechizo, podemos examinar nuestro archivo y ver que los errores ortográficos han sido corregidos.

```
[~]$ cat foo.txt
```

El veloz zorro marrón saltó sobre el perro perezoso. A menos que se indique lo contrario a través de la opción de línea de comando --dont-backup, aspell crea un archivo de respaldo que contiene el texto original agregando la extensión .bak al nombre del archivo.

Mostrando nuestra destreza en la edición de sed, volveremos a colocar nuestros errores de ortografía para que podamos reutilizar nuestro archivo.

```
[~]$ sed -i 's/lazy/laxy/; s/jumped/jimped/' foo.txt
```

La opción sed -i le dice a sed que edite el archivo "en el lugar", "in-place", lo que significa que en lugar de enviar la salida editada a la salida estándar, reescribirá el archivo con los cambios aplicados. También vemos la capacidad de colocar más de un comando de edición en la línea separándolos con un punto y coma.

A continuación, veremos cómo aspell puede manejar diferentes tipos de archivos de texto. Usando un editor de texto como vim (los aventureros pueden querer probar sed), agregaremos algunas marcas HTML a nuestro archivo.

```
<html>
    <head>
        <title>Mispelled HTML file</title>
    </head>
    <body>
        <p>The quick brown fox jimped over the laxy dog.</p>
    </body>
</html>
```

---

aspell verá el contenido de las etiquetas HTML como mal escrito. Este problema se puede solucionar incluyendo la opción de modo de comprobación -H (HTML), como esta:

```
[~]$ aspell -H check foo.txt
```

---

que resultará en esto:

```
<html>
    <head>
    <title> Mispelled HTML file</title>
    </head>
    <body>
        <p>The quick brown fox jimped over the laxy dog.</p>
    </body>
</html>

1) Mi spelled 6) Misapplied
2) Mi-spelled 7) Miscalled
3) Misspelled 8) Respelled
4) Dispelled  9) Misspell
5) Spelled    0) Misled
i) Ignore     I) Ignore all
r) Replace    R) Replace all
```

```
a) Add          l) Add Lower
b) Abort        x) Exit

?
```

El HTML se ignora y solo se verifican las partes del archivo que no están marcadas. En este modo, el contenido de las etiquetas HTML se ignora y no revisado para la ortografía. Sin embargo, el contenido de las etiquetas ALT, que se benefician de la verificación, se verifica en este modo.

> **Nota:**

> De forma predeterminada, aspell ignorará las URL y las direcciones de correo electrónico en el texto. Este comportamiento se puede anular con las opciones de la línea de comandos. También es posible especificar qué etiquetas de marcado se verifican y se omiten. Consulte la página de manual de aspell para obtener más detalles.