

Redireccionamiento de entradas y salidas

FLUJO DE DATOS

www.sxc.hu

Trabajar con el shell tiene muchas ventajas. Tuberías, redireccionamientos y cadenas de comandos proporcionan a los usuarios casi un número infinito de opciones. **POR HEIKE JURZIK**

La combinación de los comandos del shell nos permite incrementar el poder de la línea de comandos. Esto no sólo incluye el enlace a programas individuales, sino también la transferencia de la salida de comandos. Por ejemplo, si la información que necesitamos desaparece de la pantalla tan rápidamente que nos impide leerla, podemos redireccionarla a un fichero o usar un paginador.

Para enlazar comandos individuales en bash, sólo son necesarias un par de cosas. En vez de decir

```
$ mkdir carpeta
$ cd carpeta
$ cp ../carpeta2/* .
```

podemos combinar estos pasos y usar punto y coma para decirle al intérprete que ejecute estos comandos en una sucesión rápida:

```
$ mkdir &
carpeta; cd carpeta; &
cp ../carpeta2/* .
```

Para ejercer mayor control sobre bash, podemos pasarle una condición. Por ejem-

plo, podríamos ejecutar un segundo comando (o tercero, o enésimo) sólo cuando el comando anterior se haya completado exitosamente o si ha fallado.

Para asegurarnos de que un fichero existe antes de borrarlo, añadiríamos una condición de prueba simple a la línea de comandos con *rm*:

```
test -w fichero && rm fichero
```

El programa *test* simplemente comprueba para ver si un fichero existe y, en este caso, si es de lectura-escritura (parámetro *-w*), en cuyo caso *rm* lo borraría. Si necesitas compilar software frecuentemente desde el código fuente usando el grupo normal de comandos *./configure; make; make install*, puedes combinar los tres usando *&&* para asegurarte de que cada comando sucesivo sólo se ejecuta si el previo no tenía ningún error:

```
./configure && make && &
make install
```

Además de la opción *-w*, el comando *test* dispone de otros prácticos usos. Por ejemplo, *-d* permite comprobar si un

directorio existe, si no fuera así lo podemos crear:

```
test -d carpeta || mkdir carpeta
```

Los dos caracteres tubería entre ambos comandos actúan como un OR lógico.

Para la entrada y salida de los comandos en el shell no existen más que tres canales: Los programas leen datos desde su entrada estándar (STDIN, canal 0) o desde un fichero, la salida del programa se envía a una salida estándar (STDOUT, canal 1) y los mensajes de error se escriben en la salida estándar de error (STDERR, canal 2):

```
comando < entrada > salida 2> &
error
```

Los operadores *<* y *>* indican la dirección: si la entrada estándar no tiene su origen en el teclado, *<* le dice a *comando* que lo lea desde un fichero. Para redireccionar la salida a un fichero necesitas el operador *>*. La salida de error también usa *>*; sin embargo, se debe especificar el canal añadiendo una descripción del fichero (*2>*).

Redireccionando la Salida

Como he mencionado antes, el operador `>` redirecciona la salida desde un programa a un fichero. En lugar de `>` también podríamos escribir `1>` porque la opción especifica primer canal; sin embargo, esto no es estrictamente necesario, porque el shell supondrá que tiene que enviar los datos a la salida estándar si no le dices lo contrario:

```
ls /etc > contenido.txt
```

Si el fichero detrás del operador ya existe, el shell lo sobrescribe. La condición `test` que examinamos antes previene que esto ocurra:

```
test -w etc_contenido.txt ||
ls /etc > etc_contenido.txt
```

O puedes utilizar el operador `>>`:

```
ls /etc >> etc_contenido.txt
```

Los dos signos “mayores que” (`>>`) significan que el shell ha de añadir la salida desde el comando `ls` al fichero `etc_contenido.txt` si este fichero ya existe. Si no, el shell crearía un nuevo fichero y escribiría la salida en dicho fichero.

Captura de Mensajes de Error

Para redireccionar el segundo canal usa el número 2 seguido del operador `>`. Este método es idóneo si un programa genera tantos mensajes de error que nos evita leer la salida actual del programa:

```
$ find /home -name "*.tex"
find: /home/lost+found:
No permissions
find: /home/petronella/data:
No permissions
/home/huhn/book/book.tex
/home/huhn/book/chap01.tex
...
```

El comando

```
find /home -name "*.text" 2>>
/dev/null
```

permite enviar mensajes de error a `/dev/null` de nuestra máquina, evitando que se sature la salida estándar.

Dos Pájaro de un Tiro

Una combinación inteligente de operadores nos permite redireccionar dos canales al

mismo tiempo. Si deseas escribir la salida estándar a un fichero desde el comando `find` en el ejemplo anterior, aunque sin que se registren todos los mensajes de error, usa:

```
find /home -name
"*.text" >
salidafind 2>
/dev/null
```

El doble operador `>>`, que creará ficheros que no existen o los añadirá a ficheros que si existen, también es útil en este caso. El ejemplo anterior muestra cómo hacer esto para una salida estándar. Para el caso de salidas estándar de errores podemos usar la doble flecha con el mismo efecto si necesitamos escribir mensajes de error a un fichero:

```
find /home -name "*.text" >
salidafind 2>> error
```

Fontanería

Las tuberías a menudo nos evitan pasos al direccionar la salida directamente desde un programa a otro sin que sea necesario el desvío a través de un fichero. El carácter pipe (`|`) separa los comandos individuales, como se demuestra en el siguiente ejemplo.

```
ls /etc | less
```

Esto envía la salida desde el comando `ls` al paginador `less` o directamente al terminal. El paginador muestra la salida pantalla por pantalla y soporta el desplazamiento con las teclas de cursores. La tubería es muy común en combinación con salidas de `grep` para flujos específicos; por ejemplo,

```
find debian -name "*.png" |
grep --color apt
```

busca en el directorio `debian` todos los ficheros que acaban en `.png` y pasa la salida a `grep`. `grep` busca en el flujo la secuencia de caracteres `apt` y destaca los que encuentra en rojo gracias a la opción `--color` (Figura 1).

Puedes usar distintas tuberías. El comando siguiente lista el contenido de tu directorio de inicio línea a línea, pasa los resultados a la herramienta `grep`, busca en el flujo `.jpg` y cuenta las coincidencias (a efectos prácticos: cuenta el número de ficheros de imagen JPEG en tu directorio de inicio):

```
$ ls -l ~ | grep .jpg | wc -l
12
```

Cuando usé este comando encontré, 12 ficheros JPEG en mi directorio de inicio sin tener que hacer nada muy enrevesado y utilizando herramientas estándar de la línea de comandos.

La Hora del Tee

Para desviar la salida de un comando a más de un sitio usando el programa Tee. El comando espera datos desde la entrada estándar y los puede escribir simultáneamente tanto a un fichero como en pantalla.

Podemos colocar una “Tee junction” (o intersección Tee) al final de la línea de comandos o entre comandos individuales:

```
comando1 | tee salida.txt | && comando2
```

Para buscar los ficheros PNG, comenzando por el directorio actual, registrando a continuación esta salida en el fichero `imagenes.txt` mientras lo presenta en la pantalla, mientras se busca con `grep` el cadena `libro` en el nombre de las imágenes, usaremos:

```
find . -name "*.png" |
tee imagenes.txt | grep libro
```

Por defecto, Tee sobrescribirá el fichero si existe. Para añadir la salida a un fichero existente, simplemente añadimos el parámetro `-a`:

```
find . -name "*.png" | tee
-a imagenes.txt | grep libro
```

Los distintos operadores, junto con la tubería y los comandos Tee soportan combinaciones de comandos extremadamente flexibles. No vale la pena crear un script simplemente para búsquedas rápidas, por ejemplo.

Pronto descubrirás incluso nuevos usos y nuevas combinaciones.

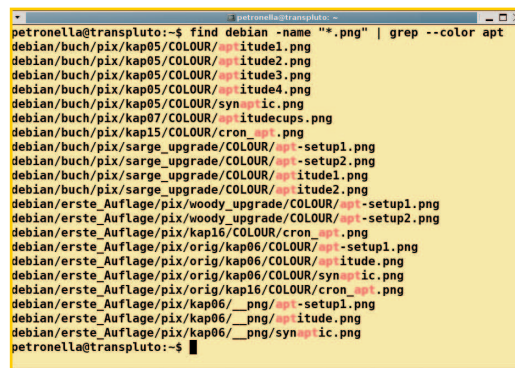


Figura 1: La salida del comando `find` se envía a la herramienta `grep` sin desviarla a través del shell.