Capítulo 4

Redireccionamientos y filtros

4.1. Introducción

En el capítulo 6 se explicará a fondo el significado de *shell* así como algunas características del *shell* bash. En este capítulo nos centraremos en algunas características de éste relacionadas con los redireccionamientos y los filtros:

1. Manejo de la entrada/salida estándar.
2. Listas de órdenes.
3. Agrupación de órdenes.

4.2. Entrada/Salida estándar

Por convenio, la mayor parte de los programas de GNU/Linux toman su entrada de la entrada estándar, envían la salida que producen a la salida estándar y los mensajes de error a la salida de errores estándar.

Todos los *shells* manejan la E/S estándar básicamente de la misma forma. Cada programa que llamemos tiene inicialmente los tres canales de E/S estándar asignados; así la entrada estándar está asignada al teclado, y la salida estándar y de errores a la pantalla o a la ventana donde se ejecuta la orden (en sistemas X Window).

Cuando es necesario, podemos redirigir la entrada y la salida estándar a un fichero. Es decir, podemos hacer que un programa tome la entrada que necesite de un fichero, o bien que escriba la salida que proporciona en un fichero. Esta es una de las funciones del *shell*.

Asimismo podemos concatenar varios programas mediante interconexiones; en este caso, la salida del primer programa alimenta la entrada estándar del segundo, y así sucesivamente. Esto hace posible emplear las utilidades de GNU/Linux para construir líneas de órdenes más complejas.

4.2.1. Redirección de la E/S

Una de las funciones del *shell* es gestionar la entrada y salida estándar. Las órdenes no necesitan saber de dónde les viene la entrada o a dónde va la salida, es el *shell* el que establece estas conexiones. Para ello el *shell* bash reconoce una serie de operadores, que podemos ver en el cuadro 4.1.

Símbolo	**Función**
orden `>` *fichero*	Escribe la salida de *orden* en *fichero*. Crea *fichero* si no existe, en otro caso lo reescribe.
orden `<` *fichero*	Toma la entrada para *orden* de *fichero*.
orden1 `|` *orden2*	Conecta la salida estándar de la *orden1* a la entrada estándar de la *orden2*.
orden `>>` *fichero*	Añade la salida de *orden* a *fichero*. Crea *fichero* si no existe.
orden `2>` *fichero*	Escribe la salida de error estándar en *fichero*. Crea *fichero* si no existe.
orden `2>>` *fichero*	Añade la salida de errores a *fichero*.
orden `&>` *fichero*	Redirecciona las salidas estándar y de errores al mismo *fichero*.
orden `>|` *fichero*	Redirecciona la salida a *fichero*, incluso si éste existe y la opción `noclobber` está activada.

Cuadro 4.1: Operadores de redirección

El operador > *fichero*

La salida estándar de una orden está asignada por omisión al terminal. Usando el símbolo >, la salida estándar de una orden puede ser redirigida a un fichero. El fichero se abre para escritura y se asigna a la salida estándar.

Ejemplo:

```
$ echo hola > p.out
```

```
$ cat p.out
hola
```

La salida de la orden `echo` *se redirecciona al fichero* `p.out`*, de forma que el contenido de éste es la cadena* `hola`.

Si el fichero no existe, se crea. Cuando el *shell* encuentra en una línea de órdenes un redireccionamiento de salida hacia un fichero que ya existe, lo primero que hace es abrir el fichero para escritura. Esto significa que su contenido se borra siempre, fuera el que fuese. Seguidamente, establece como contenido del fichero la información de salida del mandato que redirigimos. Evidentemente esta operación requiere que el fichero tenga los permisos adecuados además de tener en cuenta la opción `noclobber` del *shell*.

Ejemplo:

```
$ chmod 444 p.out
$ echo hola otra vez > p.out
bash: p.out: Cannot clobber existing file
```

El símbolo > también se puede utilizar para crear un fichero vacío.

Ejemplo:

```
$ > vacio
$ ls -s vacio
0 vacio
```

La opción `noclobber`

Impide que se destruya la información al redirigir la salida estándar a un fichero existente. La opción se establece con `set -o noclobber`.

Ejemplo:

```
$ ls > fichero.salida
$ chmod 777 fichero.salida
$ set -o noclobber
$ ls > fichero.salida
bash: fichero.salida: Cannot clobber existing file
```

El operador >| *fichero*

El operador `>|` se usa para forzar la sobreescritura de un fichero aunque está establecida la opción `noclobber`.

Ejemplo:

```
$ ls >| fichero.salida
```

En este caso no tiene en cuenta si la opción `noclobber` *está establecida.*

El operador >> *fichero*

El operador `>>` *fichero* añade a *fichero* la salida estándar. Si no existe *fichero* lo crea. Opera de modo similar al operador `>`, excepto en que los datos se escriben al final del contenido actual de *fichero*. El operador `>>` es útil para acumular la salida de varias ejecuciones consecutivas de una orden en un mismo fichero.

Ejemplos:

1.
```
$ ls -l > lista
```
2.
```
$ cat lista
$ ls -l >> lista
$ cat lista
```

El operador < *fichero*

Para redirigir la entrada estándar de una orden se utiliza el operador `<`. El fichero se abre para lectura y se asigna a la entrada estándar.

Ejemplo:

```
$ mail juan < carta
```

Hace que el programa `mail` *tome su entrada del fichero* `carta`.

Interconexiones: el operador |

También es posible redirigir la salida de un programa a la entrada estándar de otro en vez de a un fichero. Esto lo podemos hacer mediante una **interconexión** o tubo (*pipe*) y para ello se utiliza el símbolo `|`.

orden1 `|` *orden2* `|` ...

El *shell* arranca todas las órdenes simultáneamente. La salida estándar de cada orden, excepto la última, se conecta con la entrada estándar de la siguiente. A la secuencia de una o más órdenes simples separadas por el carácter `|`, se le denomina **tubería** (*pipeline*).

La información escrita en una interconexión se mantiene en la memoria del sistema hasta que otro programa la lee. Normalmente ésta es abierta por dos programas diferentes al mismo tiempo, uno la abre para escritura y el otro para lectura.

Ejemplos:

1. `$ find . -user boabdil | wc -l`
   Muestra en la salida estándar el número de ficheros del directorio de trabajo de los que es propietario el usuario `boabdil`.
2. `$ ls -l | more`
   Muestra en la salida estándar un listado largo del directorio de trabajo de forma paginada.
3. `$ who | wc -l`
   Muestra en la salida estándar el número de usuarios conectados al sistema.

Mediante la interconexión de órdenes podemos construir líneas de órdenes con funciones nuevas.

La orden `tee` La orden `tee` nos suministra un modo de capturar el contenido de una interconexión en un fichero sin interrumpir el flujo de información de la interconexión. Su formato es:

orden1 `|` `tee` [`-a`] [*fichero* ...] `|` *orden2*

La orden `tee` hace que la entrada estándar, proveniente de la salida estándar de *orden1*, se copie en la salida estándar, convirtiéndose en la entrada estándar de *orden2*, y también en *fichero*. Si se le pasa la opción `-a` añade la información a éste.

Ejemplos:

1. `$ who | tee usuarios | wc -l`
   La salida de la orden `who` *se copia en el fichero* `usuarios` *y además pasa por la interconexión a la siguiente orden (*`wc -l`*) que contará el número de líneas de la entrada. Es decir, la entrada estándar se desvía al fichero especificado y continúa a la salida estándar.*
2. `$ cat modelo | tee uno dos tres | lpr`
   Crea tres copias del fichero `modelo` *con los nombres* `uno`, `dos` *y* `tres`*, y lo imprime.*

Descriptores de ficheros

Un descriptor de fichero es un número entero que utiliza el núcleo para identificar los distintos flujos de datos asociados a un proceso. Cuando un proceso arranca, usualmente tiene tres descriptores de ficheros abiertos, los correspondientes a la entrada estándar (0), salida estándar (1), y salida de errores estándar (2). Si un proceso abre ficheros adicionales para entrada o salida, a éstos se les asignan los siguientes descriptores disponibles, empezando por el 3.

A cualquiera de los operadores de redireccionamiento (de entrada/salida) se le puede anteponer como prefijo un número de descriptor de fichero (0-9). Si se especifica éste, es ese descriptor de fichero el que se abre (de lectura/escritura) en vez de la entrada o salida estándar. Si no especificamos descriptor de ficheros alguno, los operadores de redireccionamiento actúan sobre la entrada/salida estándar.

Los operadores `2>` *fichero* y `2>>` *fichero*

La mayoría de las órdenes escriben sus mensajes de error en la salida estándar de errores. Los mensajes de error de los mandatos se pueden redirigir usando el descriptor 2 con los símbolos de redireccionamiento ya comentados.

Ejemplos:

1. `$ ls tmp 2> error` *Si se produce algún error al ejecutar la orden, éste se escribiré en el fichero* `error`*. Es decir, se ha redirigido la salida de errores al fichero mencionado.*
2. `$ ls -R / 2> /dev/null`
   Cuando no nos interesa ver los mensajes de error, ni queremos almacenarlos en un fichero podemos redireccionar la salida de errores al dispositivo `/dev/null`.

El operador `&>` *fichero*

Redirige la salida estándar y de errores al mismo fichero. También se puede utilizar el operador `>&`, aunque de las dos formas se prefiere la primera.

Ejemplo:

```
$ programa &> salida
```

Redirige tanto la salida estándar como la de errores al mismo fichero.

Ejercicios:

1. ¿Qué hacen las siguientes órdenes?
   a) `$ ls -l lpr | lpr`
   b) `$ cat < cat`
   c) `$ find . -name "*.txt" >> more 2> /dev/null`
   d) `$ more less 2> cat | cat > lpr`
   e) `$ cat fichero | tee contenido | wc -l`

4.3. Listas de órdenes

Una **lista** es una secuencia de una o más tuberías separadas por unos de los operadores siguientes: `;`, `&`, `&&` o `||`, y terminada opcionalmente en uno de los siguientes: `;`, `&`, o `Nueva-Línea`.

A continuación veremos qué significado tienen los operadores `;`, `&&`, `||`. El estudio del operador `&` se deja para el apartado **??**.

4.3.1. órdenes múltiples

El *shell* bash permite escribir varias órdenes en una misma línea separándolas con el carácter `;`. El resultado es equivalente a dar las órdenes de forma independiente, cada una en una línea.

Ejemplo:

Compare las salidas que producen:

1. `$ pwd ; ls texto ; echo hola`
2. ```
   $ pwd
   $ ls texto
   $ echo hola
   ```

4.3.2. Ejecución condicional

Cuando un programa termina su ejecución devuelve un valor de salida (status) que indica si se ha ejecutado correctamente o no. Un valor de salida cero significa que su ejecución ha sido correcta, y uno distinto de cero indica una ejecución incorrecta o que se ha producido algún error. Este valor de salida es usado por el *shell* bash mediante dos operadores condicionales simples, `&&` y `||`.

Si se separan dos órdenes mediante el operador `&&`, la segunda sólo se ejecutará si la primera se ejecuta correctamente.

Ejemplos:

1. ```
   $ ls temp && echo fichero temp existe
   temp
   fichero temp existe
   ```
2. ```
   $ rm temp
   $ ls temp && echo fichero temp existe
   ls: temp: No such file or directory
   ```

Si dos órdenes se separan por el operador `||`, la segunda se ejecutará sólo si la ejecución de la primera devuelve un valor de salida distinto de cero, es decir, si no se ejecuta correctamente.

Ejemplo:
   ```

```
$ ls temp || echo fichero temp no existe
ls: temp: No such file or directory
fichero temp no existe
```

4.4. Agrupación de órdenes

El *shell* bash proporciona dos formas de agrupar una lista de órdenes para que sea ejecutada como una unidad, mediante el uso de `{}`, o de `()`.

El *shell* bash trata sintácticamente un grupo como si fuera una orden simple a efectos de interconexiones, redirecciones u orden de evaluación en mandatos condicionales. Es útil cuando se quiere combinar la salida de varias órdenes.

Los caracteres `{ }` tienen que escribirse aislados, es decir, precedidos y seguidos de un espacio. Las órdenes que se incluyan dentro de `{ }`, si se dan en una sola línea deben terminar en `;`.

Ejercicios:

1. Explique la diferencia entre la salida que producen estas órdenes:

```
$ echo El fichero temp: ; cat temp | wc -l
$ { echo El fichero temp: ; cat temp ; } | wc -l
```

La agrupación de órdenes se puede usar también con los operadores de ejecución condicional.

Ejemplo:

```
$ ls temp && { echo Existe ... borrando ; rm temp ; }
```

La utilización de `( )` es similar, pero la lista se ejecuta en un *subshell*, concepto que se estudiará en el apartado **??**.

4.5. Concepto de filtro

Los filtros son programas que hacen una selección o transformación del flujo de datos de la entrada estándar o de un fichero, en la salida estándar.

Los filtros están diseñados para leer la entrada estándar, procesarla de alguna manera y escribir el resultado en la salida estándar. Puede pensarse

que la información en bruto entra por un extremo de la orden, obteniéndose en el otro la información filtrada. Una ventaja importante de los filtros es que pueden usarse con redireccionamientos e interconexiones.

4.6. Ordenación de ficheros

La orden `sort` ordena las líneas de un fichero de texto. Como unidad de ordenación utiliza la línea, a no ser que se le indique otra cosa; `sort` ordena según distintos criterios, dependiendo de las opciones elegidas. Su formato es:

`sort` [*opciones*] [*fichero* ...]

Si no se especifica *fichero* se ordena la entrada estándar. Si se especifica un fichero, lo ordena. Si se especifican varios, primero los concatena y después ordena el resultado. Si incluimos en la lista de ficheros el carácter `-`, se tratará la entrada estándar como un fichero.

Ejemplo:

```
$ ls -lR > listavieja
$ who | sort listavieja - > nuevowho
```

`sort` *toma como entrada el fichero* `listavieja` *y la entrada estándar, que es la salida de* `who`*. La salida se almacena en el fichero* `nuevowho`.

Opciones:

`-o` *fichero* Escribe la salida en *fichero*. Puede ser el mismo que uno de los de entrada.

Ejemplo:

```
$ sort -o notas notas
```

Sustituye el fichero original por el fichero ordenado.

`-f` Indica que ignore las diferencias entre las mayúsculas y las minúsculas a la hora de ordenar.

`-n` Indica a `sort` que ordene los números por su valor numérico.

Ejemplo:
`$ sort -n datos`
Ordena el fichero `datos` *según el orden numérico normal.*

`-r` Invierte el orden de clasificación.

`-u` Elimina líneas repetidas.

`-d` Especifica ordenación tipo diccionario (letras, dígitos y blancos sólo).

`-b` Ignora los blancos iniciales.

4.7. Selección de caracteres o campos de un fichero

La orden `cut`, selecciona secciones específicas de cada línea de la entrada y las muestra. Su formato es:

`cut` [*opciones*] [*fichero ...*]

Si no se especifica *fichero*, toma su entrada de la entrada estándar; si se especifican uno o varios *ficheros*, tomará su entrada de éstos. Si se da el carácter `-` como *fichero*, éste especifica la entrada estándar.

Opciones:

La orden `cut` nos permite seleccionar caracteres o campos. Para ello proporciona dos opciones `-c` y `-f` que van seguidas de la lista de caracteres o campos que deseamos seleccionar. La lista es una secuencia de números o un rango. El rango se indica con un par de números separados por el carácter `-`. Los números deberán especificarse en orden creciente.

- Cómo seleccionar caracteres:
  `cut -c`*lista* [*fichero ...*]

  Ejemplo:
  `$ cut -c1,3-6  /etc/group`
  Selecciona de cada línea del fichero `/etc/group` *el primer carácter y del tercero al sexto.*

- Cómo seleccionar campos:
  `cut -f`*lista* [`-d`*caracter*] [`-s`] [*fichero ...*]

Ejemplo:
`$ cut -f5,7-9  casa`
Selecciona los campos 5.º, 7.º, 8.º y 9.º del fichero `casa`.

Los campos son una serie de caracteres separados, si no se le indica otra cosa, por el carácter `<tab>`. Para especificar un carácter separador de campos diferente, tendremos que dar la opción `-d`.

`-d`*char* *char* indica el carácter a utilizar como separador de campos.

Ejemplos:
1. `$ cut -f2,4 -d: /etc/passwd`
2. `$ who | cut -d" " -f1`
3. `$ cut -f5-9 -d" " otrofichero`

`-s` No muestra aquellas líneas que no contienen el carácter separador de campos.

4.8. Eliminación de líneas duplicadas

La orden `uniq` suprime o informa de las líneas duplicadas consecutivas que contiene un fichero. Su forma de uso es:

`uniq` [`-udc`] [+*n*] [-*n*] [*fich-entrada* [*fich-salida*]]

Los datos los toma de *fich-entrada* y los escribe en *fich-salida*. Si sólo se indica un fichero, el resultado se manda a la salida estándar. Si no se indica ningún fichero lee de la entrada estándar y escribe en la salida estándar.

Ejemplo:
`$ uniq repetido unico`
Elimina las líneas repetidas consecutivas del fichero `repetido` *y envía la salida al fichero* `unico`.

Opciones:

`-u` Sólo muestra aquellas líneas que no están repetidas.

`-d` Sólo muestra las líneas que están repetidas.

`-c` Muestra el número de veces que aparece cada línea en la entrada junto con dicha línea.

`-`*n* Ignora los primeros *n* campos de una línea. Los campos están separados por espacios o por tabuladores.

`+`*n* Ignora los *n* primeros caracteres de una línea a la hora de comprobar si está duplicada.

4.9. Borrado de columnas

La orden `colrm` elimina columnas de la entrada. Su formato es:

`colrm` [*columna_inicial* [*columna_final*]]

Si sólo se especifica la *columna_inicial*, borrará desde ésta hasta la última. Para que lea de un fichero habrá que especificar el operador de redirección de la entrada estándar.

Ejemplo:

```
$ colrm 1 3 < datos
```

Muestra en la salida el resultado de eliminar las columnas primera, segunda y tercera del fichero `datos`. *Observe que hemos redirigido la entrada estándar para que lea del fichero.*

4.10. Sustitución caracteres

La orden `tr` copia la entrada estándar en la salida estándar sustituyendo o borrando los caracteres seleccionados. Su uso más común es la conversión de mayúsculas y minúsculas. Su formato es;

`tr` [`-d`] [*cadena1* [*cadena2*]]

Opciones:

`-d` Borra todos los caracteres que se especifican en la *cadena1* de la salida estándar.

cadena1 Caracteres a sustituir o borrar.

cadena2 Caracteres en que se van a convertir.

Ejemplos:

1. `$ tr "a-z" "A-Z" < fuente`
   Pasa a mayúsculas todas las letras minúsculas del fichero `fuente` *y lo muestra en la salida estándar.*
2. `$ tr "A-Z"  "a-z" < fuente > resultado`
   Pasa a minúsculas todas las letras mayúsculas del fichero `fuente` *y lo graba en el fichero* `resultado`.

Ejercicios:

1. Escriba una línea de órdenes que muestre los nombres de los usuarios del sistema de forma ordenada y en mayúsculas.