Redirecciones Y Pipes

Cuando utilizamos la consola a menudo la salida de un comando la tenemos que aprovechar en otro, preferiríamos que la salida se nos guardase directamente en un fichero, o simplemente deseamos utilizar cierta información de la salida de dicho comando. En GNU/Linux, hay dos mecanismos que nos facilitan enormemente esta tarea, y que con la costumbre llegan a ser casi imprescindibles: las redirecciones, y los *pipes* (o tuberías).

Redirecciones

En GNU/Linux, al final todo es tratado como si fuera un fichero y como tal, tenemos *descriptores de fichero* para aquellos puntos donde queramos acceder. Hay unos descriptores de fichero por defecto:

- 0: Entrada estándar (normalmente el teclado).
- 1: Salida estándar (normalmente la consola).
- 2: Salida de error.

Para redirigir las salidas utilizaremos el **descriptor de fichero** seguido del símbolo '>' (o < si redirigimos la entrada hacia un comando, pero eso todavía no lo explicaremos). Veamos unos ejemplos:

```
$ ls -l >fichero
```

Guarda la salida de *ls -l* en *fichero*. Si no existe lo crea, y si existe lo sobreescribe. `$ ls -l >>fichero`
Añade la salida del comando a *fichero*. Si no existe lo crea, y si existe, lo **añade al final**.

```
$ ls -l 2>fichero
```

Si hay algún error, lo guarda en *fichero* (podría salir un error si no tuviéramos permiso de lectura en el directorio).

Es importante ver que **si no se especifica el descriptor de fichero se asume que se redirige la salida estándar**. En el caso del operador **<** se redirige la entrada estándar, es decir, el contenido del fichero que especificáramos,

se pararía como parámetro al comando.

Si quisiéramos redirigir todas las salidas a la vez había un mismo fichero, podríamos utilizar **>&**. Además, con el carácter **&** podemos redirigir salidas de un tipo hacia otras, por ejemplo, si quisiéramos redirigir la salida de error hacia la salida estándar podríamos indicarlo con: **2>&1**. Es importante tener en cuenta que el orden de las redirecciones es significativo: se ejecutarán de izquierda a derecha.

Tuberías o *pipes*

Este mecanismo nos permite pasar la salida de un comando a otro. Para ello se usa la sintaxis: **<comando1> | <comando2>**. Con esto, la salida de *comando1* será la entrada de *comando2*. Vamos a ver unos ejemplos:

```
$ rpm -qa | grep <nombre_paquete>
```

El primero de los dos comandos nos haría una lista de todos los paquetes instalados. Imaginemos que sólo queremos saber si tenemos instalado uno en concreto. Con el segundo comando limitamos la salida a los paquetes que en el nombre que contengan el *patrón* que especificamos en *<nombre_paquete>*. Por ejemplo, para saber si tenemos instalado algún paquete llamado **glibc** haríamos:

```
$ rpm -qa | grep "glibc"
```

grep es un *parseador* de expresiones regulares, es decir, le damos un patrón y un fichero (o introducimos lo que sea por consola, o lo pasamos con un *pipe*) y de ese texto nos devuelve sólo loq eu coincide con el patrón. Este es el funcionamiento básico de **grep** pero es muchísimo más potente. Para más información: man grep.

Otro ejemplo útil seria, por ejemplo, cuando queremos saber el PID de un proceso. En vez de mostrarlos todos y tener que buscarlo podríamos hacer:

```
$ ps -e | grep <nombre_proceso>
```

y así nos mostraría sólo las líneas que contuvieran *<nombre_proceso>* (es decir, limitaríamos la salida al proceso que queremos).

Conclusión

Realmente estos mecanismos son muy potentes y la gran mayoría de veces nos facilitan mucho la tarea. Al principio pueden aparecer extraños o innecesarios, pero cuando empiezan a utilizarse, cada vez se usan más a menudo y terminan siendo muy útiles para ahorrar tiempo.

Tienen infinidad de usos y hay muchas ocasiones en las que nos serán útiles. No solamente sirven para limitar la salida a lo que nosotros queramos; podemos darle muchísimos usos distintos más. Sólo es cuestión de encontrarse en una situación en la que queramos ahorrarnos el tener que escribir unas líneas de más.

Tutorial elaborado por Nacx para ADSLAyuda.com