Más todavía del bash

1. Alias

Un alias es un atajo a un comando y a sus posibles parámetros. Se define con el comando **alias**. Utilizado sin argumentos, lista los alias disponibles.

```
$ alias
alias ..='cd ..'
alias ...='cd ../..'
alias cd..='cd ..'
alias dir='ls -l'
alias l='ls -alF'
alias la='ls -la'
alias ll='ls -l'
alias ls='ls $LS_OPTIONS'
alias ls-l='ls -l'
alias md='mkdir -p'
alias o='less'
alias rd='rmdir'
...
```

Puede crear sus propios alias.

```
$ alias deltree='rm -rf'
```

2. Agrupación de comandos

El encadenamiento de comandos es posible con «;». También es posible agrupar los comandos.
Cuando ejecuta los comandos siguientes:

```
$ uname -a ; pwd ; ls -l >resultado.txt &
```

Se ejecuta únicamente el último comando en segundo plano y sólo se redirecciona su resultado en el archivo resultado.txt. Una solución sería:

```
$ uname -a >resultado.txt & ; pwd >>resultado.txt & ; ls -l >>resultado.txt &
[1] 18232
[2] 18238
[3] 18135
```

Es una solución compleja y no funcionará siempre. Además, incluso si se inician los comandos de manera secuencial, se ejecutan todos en paralelo. El primero que finalice será el primero en escribir en el archivo. La solución consiste en utilizar paréntesis.

```
$ (uname -a ; pwd ; ls -l) > resultado.txt &
[1] 18239
$
[1]   Done              ( uname -a; pwd; ls -l ) > resultado.txt
```

En cualquier caso, se inician todos los comandos colocados entre paréntesis con un subshell, que luego ejecutará los comandos precisados de manera secuencial tal como esté indicado. Así, la redirección concierne al conjunto de los comandos y nada impide lanzar este subshell en segundo plano. De hecho, se distingue claramente un solo PID 18239 durante la ejecución de los comandos.

Otra posibilidad consiste en utilizar llaves {...}. En este caso, no se ejecutará ningún subshell, y si se ejecuta un comando interno (cd u otro), concierne al shell activo. Se debe colocar la llave de cierre justo después de un ;.

```
$ { uname -a; pwd; ls -l; } > resultado.txt
```

Se puede comprobar la diferencia fácilmente entre las dos sintaxis con exit. El primer ejemplo parece no hacer nada, manteniéndose en el shell hijo. El segundo sale de su shell.

```
$ (exit)
```

```
$ { exit; }
```

Cuidado con los paréntesis, en particular en programación. Como se lanza la agrupación dentro de otro proceso, las posibles variables modificadas dentro de la agrupación no serán visibles una vez terminada la ejecución.

3. Relación y ejecución condicional

Además del encadenamiento clásico, se puede relacionar y ejecutar los comandos de manera condicional. La condición de ejecución de un comando es el éxito o no del anterior. Una vez ejecutado, cada comando devuelve un código de retorno, en general 0 si todo ha salido bien, 1 en caso de error. El shell puede recuperar este valor con la variable $?. El valor suele estar indicado en la ayuda del comando utilizado.

```
$ ls
...
$ echo $?
0
```

Los caracteres **&&** y || permiten efectuar una ejecución condicional.

```
comando1 && comando2
comando1 || comando2
```

Se ejecutará el comando colocado después de **&&** únicamente si el comando anterior ha devuelto 0 (éxito). Sólo se ejecutará el comando colocado después de **||** si el comando anterior ha devuelto algo diferente a 0.

```
$ grep "ratón" lista && echo "Ratón encontrado" || echo "Ratón no encontrado"
ratón  óptico 30     15
Ratón encontrado
$ grep "memoria" lista && echo "Memoria encontrada" || echo "Memoria
imposible de encontrar"
Memoria imposible de encontrar
```