Resumen de comandos usuales de Unix y uso básico del shell `bash`

Indice

---

1. Generalidades sobre introducción de comandos en el shell

- **Introducción de un comando en línea de comandos**.

  En la línea de comandos se escribe un nombre de comando, seguido de uno o más modificadores (en general una letra precedida de un guión), y de uno o más nombres de ficheros dependiendo del comando. Si se especifica más de un modificador u opción se consigue su efecto combinado.

  ```
  $ comando -a -b -c ... fich1 fich2 fich3 ...
  ```

  Este conjunto (nombre + modificadores/opciones + nombres fichero) se considera un sólo comando.

- **Varios comandos en una línea.**

  - En secuencia. Separados por punto y coma. Se ejecuta un comando detrás de otro en primer plano o *foreground*.

    ```
    $ comando1 ; comando2 ; comando3
    ```

  - En segundo plano. Separados, o el último finalizado, por el carácter *ampersand* &. Cada comando terminado por este carácter se ejecuta en segundo plano o *background* (es decir, no se espera a su terminación para continuar).

    ```
    $ comando1 & comando2 & comando3 &
    ```

- Mezcla de comandos en primer y segundo plano. Los que acaban en punto y coma obligan a los que les siguen a esperar a su terminación; los que acaban en & se lanzan en segundo plano y se siguen ejecutando los siguientes.

  ```
  $ comando1 ; comando2 & comando3 ; comando4 & comando5 ...
  ```

- Agrupación de comandos. Se pueden agrupar comandos entre paréntesis para que un punto y coma o un & afecten a varios de ellos a la vez. En el siguiente ejemplo una secuencia de comandos se lanza como conjunto en background.

  ```
  $ ( comando1 ; comando2 ; comando3 ) &
  ```

- **Redirección de la salida por defecto.**

  Los resultado de un comando en pantalla pueden redireccionarse a un fichero de dos formas distintas:

  - Borrando el fichero si ya existía, creándolo nuevo si no:

    ```
    $ comando > fichero
    ```

  - Añadiendo la información al fichero si ya existía, creándolo nuevo si no:

    ```
    $ comando >> fichero
    ```

- **Tuberías o *pipelines***

  Los resultado de un comando en pantalla pueden redireccionarse a la entrada de otro comando separándolos con el carácter de *tubería* o *pipe*: |. Sólo el último comando de una tubería muestra sus resultados en pantalla (que también podrían redicreccionarse a un fichero).

  ```
  $ comando1 | comando2 | ... | comandoN
  ```

---

2. Manual

- `$ man [seccion] comando/palabra clave`
  Muestra la página de manual del comando o palabra clave. Si se incluye un número de sección sólo busca en esa. Si no, muestra la página de la primera sección en que se encuentre

---

3. Directorios

- `$ pwd`
  Muestra la ruta al directorio seleccionado por defecto
- `$ cd directorio`
  Cambia el directorio seleccionado por defecto
- `$ mkdir directorio`

Crea directorio. Falla si no existe alguno de los antecesores de la ruta
`-p` Crea todos los antecesores de la ruta que no existan

- `$ rmdir directorio`
  Borra directorio. Falla si no esta vacio
  `-p` Borra todos los antecesores en la ruta que estén vacios

---

4. Ficheros

- `$ ls [directorio]`
  Lista de ficheros del directorio (por defecto el actual)
  `-l` Con detalles
  `-a` Incluye ficheros ocultos (Los que comienzan con el carácter punto ".")
- `$ cat fich1 [fich2] ...`
  Muestra el contenido de los ficheros
- `$ more fich1 [fich2] ...`
  Muestra el contenido página a página (avanzar con la tecla enter una línea, con el espaciador una página, salir con la letra *q*).
  Un uso muy típico del comando *more* es como terminación de una tubería, para mostrar los resultados de otro comando página a página. Por ejemplo: `ps -a -x -l | more`
- `$ cp fichOrigen fichDestino`
  Copia el fichero origen con el nuevo nombre destino sobreescribiendolo si existía
- `$ mv fichOrigen fichDestino`
  Mueve el fichero. Cambia el nombre y/o la localización
- `$ rm fich`
  Borra la entrada del directorio
  `-i` Pide confirmación
  `-r` Recursivo en los subdirectorios
- `$ file fich`
  Indica de que tipo es el fichero (Ejecutable, texto, ...)
- `$ ln fichOrigen fichLinkNuevo`
  Crea un enlace (*link*) al fichero origen con el nombre de fichero nuevo
  `-s` Crea un enlace simbólico duplicando el *i-nodo*
- `$ cmp fich1 fich2`
  Compara los ficheros e indica el primer *byte* en que difieren
- `$ diff fich1 fich2`
  Muestra las lineas en que difieren los dos ficheros
- `$ sort [fich]`
  Ordena el fichero, o la entrada por defecto. Ver opciones en la página de manual El nombre de fichero es opcional. Si no hay fichero busca en la entrada por defecto. Por eso se puede utilizar como parte de una tubería (filtro).
- `$ cut`
  Filtra la entrada por defecto eliminando caracteres o campos de datos según las opciones. Ver página de manual.

---

5. Control de acceso: Permisos

- `$ chown usuario fich`
  Cambia el propietario de un fichero propio

- `$ chgrp grupo fich`
  Cambia el grupo al que pertenece un fichero propio
- `$ chmod permisos fich`
  Cambia los permisos de acceso del fichero añadiendo o quitando, o según se especifica en la cadena de permisos.
  `-u[+/-r][+/-w][+/-]x` Añade o quita el permiso especificado para el usuario
  `-g[+/-r][+/-w][+/-]x` Añade o quita el permiso especificado para el grupo
  `-o[+/-r][+/-w][+/-]x` Añade o quita el permiso especificado para el resto del mundo
- `umask dígitos`
  Fija las protecciones de acceso que los nuevos ficheros tendrán por defecto

---

6. Identificación y procesos

- `$ id`
  Muestra el nombre y número de usuario y grupo/s al que pertenece
- `$ whoami`
  Muestra el nombre de usuario
- `$ who`
  Muestra los nombres de todos los usuarios conectados al sistema, sus terminales y la fecha/hora de acceso
- `$ ps`
  Muestra la lista de procesos con sus PID y nombre
  `-a` Muestra también procesos de otros usuarios
  `-x` Muestra también procesos sin terminal de control (*daemons*)
  `-u usuario` Muestra los procesos de un usuario específico.
  `-f` Muestra los procesos en forma de árbol indicando de forma gráfica quíen es hijo de quien.
  `-l` Con todos los detalles (estado, UID, PPID, valor de nice, uso de memoria y CPU, ...)

  Un ejemplo de la salida del comando `ps -l` puede ser:

```
F S UID  PID   PPID  C PRI NI ADDR SZ    WCHAN TTY    TIME     CMD
0 S 1000 30592 30591 0 75  0  -    663   wait4 pts/60 00:00:00 bash
0 S 1000 30601 30592 0 75  0  -    34134 -     pts/60 00:30:28 netscape-bin
0 S 1000 30605 30592 0 75  0  -    506   wait4 pts/60 00:00:00 vi
0 R 1000 30608 30605 0 76  0  -    568   -     pts/60 00:00:00 ps
```

  Algunos datos de interes son:
  - S: Status (S - Sleep, R - Running, T - Stopped)
  - UID: Identificación del usuario que ha lanzado el proceso
  - PID: Identificación del proceso
  - PPID: Identificación del padre del proceso
  - PRI: Prioridad dinámica asignada en cada momento por el kernel
  - NI: Valor de *nice*, que indica si el usuario ha solicitado una menor prioridad global para el proceso con el comando `nice`
  - SZ: Tamaño que ocupa en memoria el proceso
  - TTY: Nombre del terminal asociado al proceso
  - TIME: Tiempo acumulado de CPU, invertido por el proceso hasta el momento
  - CMD: Nombre del comando o programa que está ejecutando el proceso

- `$ kill [señal] pid`
  Manda una señal al proceso especificado. La señal por defecto es SIGTERM, que indica terminación, pero puede ser enmascarada por el proceso que la recibe. SIGKILL es la señal no enmascarable de terminación; SIGSTOP detiene un proceso, SIGCONT indica que un proceso detenido puede continuar. (La señal se puede especificar como nombre o con un código numérico).
  `-l` Muestra la lista de nombres y números de las señales
- `$ nice -n valorNice comando`
  Lanza un comando con un valor de *nice* diferente del de por defecto. El valor de *nice* esta entre 0 y 19 para usuarios normales, entre -19 y 19 para el superusuario. Valores de nice positivos indican una bajada general de la prioridad de ejecución del comando, valores negativos un incremento de la misma.
- `$ sleep segundos`
  El comando realiza una espera en estado dormido o *sleep* durante el número de segundos especificado. Se suele utilizar en secuencias de comandos o en scripts.
- `$ nohup comando`
  Se desconecta el segundo comando del terminal, mandando por defecto los resultados en pantalla al fichero *nohup.out*. Este comando se utiliza casi siempre con & para lanzar el proceso en background e independiente del terminal, de forma que si se termina el shell el proceso continue de forma autónoma.
- `$ top`
  Es un programa que muestra interactivamente el estado del sistema y la utilización de CPU y memoria de los procesos más activos. Los datos se renuevan cada pocos segundos. Para salir pulsar la letra `'q'`.

---

7. Mensajes y comunicación con otros usuarios

- `$ echo mensaje`
  Muestra el mensaje en la pantalla.
- `$ mesg [y|n]`
  Habilita o Deshabilita la recepción de avisos en el terminal
- `$ write usuario [tty]`
  Escribe mensajes en el terminal del usuario si está conectado
- `$ wall [mensaje]`
  Envía el mensaje a todos los terminales activos
- `$mail, $mailx, $elm, $pine, ...`
  Diversos gestores de correo, desde simples comandos a complejas herramientas

---

8. Control de inicio y terminación de sesión

- `$ passwd [usuario]`
  Cambiar la contraseña de entrada. El superusuario puede especificar a que usuario desea cambiar la contraseña
- `$ exit`
  Finalizar la sesión

---

9. Utilidades estándar diversas

- `$ wc [fich]`
  Cuenta el número de lineas, palabras y caracteres del fichero El nombre de fichero es opcional. Si no hay fichero busca en la entrada por defecto. Por eso se puede utilizar como parte de una tubería (filtro).
- `$ date`
  Muestra la hora del sistema (Permite al superusuario fijarla)
- `$ cal [año]`
  Calendario
- `$ grep cadena [fichero]`
  Busca las lineas del fichero donde aparece la cadena. Tiene bastantes más opciones. El nombre de fichero es opcional. Si no hay fichero busca en la entrada por defecto. Por eso se puede utilizar como parte de una tubería (filtro).
- `$ find directorio -name nombreFich -print`
  Busca en un subárbol el nombre de fichero y muestra en pantalla todas las ocurrencias. El comando `find` tiene muchas opciones y permite ejecutar acciones sobre los nombres de fichero encontrados

---

10. Imprimir

- `$ lpr fich`
  Imprimir el fichero en la impresora por defecto (Mandarlo a la cola de impresión)
  `-P impresora` Utilizar otra impresora del sistema
- `$ lpq`
  Ver el estado de la cola de impresión por defecto
  `-P impresora` Ver otra cola de impresión del sistema
- `$ lprm job-id`
  Borrar de la cola de impresión el trabajo identificado por `job-id`

---

11. Variables de entorno

Las variables de entorno son variables de tipo cadena que almacena el shell. Cuando se inicia el shell ya hay algunas con valores especiales (ver más adelante).

- **Crear una variable nueva**

  No hace falta definir las variables. Se crean cuando se les da valor por primera vez. Por ejemplo:

  ```
  $ variable="Una cadena cualquiera"
  ```

  Las variables así creadas no las heredan los procesos hijos que se creen desde ese shell. Para que estos las hereden (puedan acceder a ellas), hay que *exportarlas*.

  ```
  $ export variable
  ```

Se puede cambiar el valor y exportar una variable en una sóla línea de comandos:

```
$ export variable="Una cadena cualquiera"
```

- **Usar el valor de una variable**

  El valor de una variable se puede utilizar en cualquier línea de comando en cualquier parte. Para substituir la variable por su valor se precede el nombre de la variable con el signo $. Por ejemplo, para ver el contenido de una variable:

  `echo $variable` es equivalente a: `echo Una cadena cualquiera`

  Otros ejemplos:

  ```
  $ export fichero="cosa1.txt"
  $ ls -l $fichero, es equivalente a $ ls -l cosa1.txt
  $ export fichero="stdio.h"
  $ ls -l /usr/include/$fichero, es equivalente a $ ls -l /usr/include/stdio.h
  $ export directorio="/usr/include"
  $ ls -l $directorio/$fichero, es equivalente a $ ls -l /usr/include/stdio.h
  ```

- **Variables predefinidas**

  Existen una serie de variables predefinidas por el shell que contienen valores útiles. Por ejemplo:
  - `$PWD`, el nombre del directorio actual
  - `$HOME`, el nombre del directorio raíz del usuario
  - `$PATH`, una lista de directorios separados por : donde el shell busca los comandos

  El valor de cualquiera de ellas se puede ver en cualquier momento con el comando `echo`. Por ejemplo: `$ echo $HOME`

---

12. Scripts de comandos

Un script de comandos es un fichero de texto cuyas líneas son comandos ejecutables, tal y como se escribirían en la línea de comandos del shell. Los scripts pueden ejecutarse como si fueran un nuevo comando. Las líneas que empiezan con el carácter # son comentarios y son ignoradas por el shell al ejecutar el script.

- **Crear un script de comandos**

  Escribir los comandos en un fichero de texto. Grabar. Cambiar los permisos del fichero para que tenga permiso de ejecución.

- **Ejecutar un script**

  Escribir en la línea de comandos el nombre del fichero de script, como si fuera un nuevo comando. El shell busca el fichero en los directorios que están incluídos en la variable PATH. Si el directorio actual '.' no está en el PATH, para ejecutar el script hay que indicar donde se

encuentra, por ejemplo: `./miScript`

- **Parámetros de un script**

  Al ejecutar un script se le pueden pasar argumentos. Cada cadena (separada por espacios) que se le indique detrás del nombre del fichero en la línea de comandos es un argumento (si se desea pasar una cadena que contiene espacios como un sólo argumento hay que encerrarla entre comillas).

  Dentro del script, las variables $1, $2, $3, ... se substituyen por la cadena correspondiente a cada parámetro.
  La variable $# se substituye por el número de argumentos pasado.

  Por ejemplo, se puede crear un fichero denominado `escribeArgumentos` con el siguiente contenido:

```
#
# Script de prueba
#
echo "Iniciando script de prueba"
echo Núm de argumentos: $#
echo Argumento1: $1
echo Argumento2: $1
echo Argumento3: $1
echo Argumento4: $1
```

  Cuando se ejecuta de la siguiente forma:

```
$ ./escribeArgumentos Hola 200 "Una sóla cadena"
```

```
Se obtiene en pantalla:

Núm argumentos: 3
Argumento1: Hola
Argumento2: Hola
Argumento3: Una sóla cadena
Argumento4:
```