1 Introducción

Este resumen no pretende ser una referencia completa del intérprete de comandos ("shell") de los sistemas operativos similares a UNIX, pero puede utilizarse para obtener un conocimiento mínimo que permite entrar en el sistema, crear archivos, y compilar programas. Para una referencia más completa se recomienda acudir al manual "en línea", o bien a las numerosas obras dedicadas al tema, como por ejemplo el libro de Kernighan y Pike, "El Entorno de Programación UNIX".

En lo sucesivo una cadena de caracteres que se introduce desde teclado y posee un determinado significado se escribirá <-explicación del significado->, para indicar que se trata de una **cualquier** valor admisible, no uno en particular. "QNX" y "Ubuntu" hacen referencia concretamente a las versiones disponibles en la Plataforma de Enseñanza Virtual (QNX 6.2.3 y Ubuntu 12.04), aunque lo que se afirma de ellas es en muchos casos generalizable a otras.

1.1 El intérprete de commandos o "shell"

El programa que acepta las órdenes del usuario es el "shell", que es en realidad un proceso como los demás. Hay diferentes aplicaciones que pueden cumplir esta función, como **sh**, **csh**, **ksh**, o **bash**. En QNX se utiliza por defecto **ksh** (aunque se invoque como "sh"), y en Linux-Ubuntu **bash**. Para lo que aquí se explica el comportamiento de todas las versiones es muy similar. En todos ellas es posible:

- Introducir y ejecutar comandos.
- Fijar un fichero de inicialización, que el "shell" lee al arrancar y contiene una lista de comandos para modificar su comportamiento de una manera sistemática.
- Crear y ejecutar archivos de comandos, como si fuesen programas. Son los "script".

El "shell" preprocesa lo que se teclea. Así el **asterisco** se expande para abarcar todos los ficheros del directorio de trabajo coincidentes con el patrón tecleado: <comando> *.c quiere decir que <comando> recibe los argumentos a.c, programa.c y principal.c, por ejemplo. Si no se desea que el asterisco se interprete hay que ponerlo entre comilla simple: Si se escribe '*.c' el argumento es, exactamente, *.c. El mismo resultado se obtiene precediendo el asterisco de la barra '\', o también con dobles comillas; en este último caso, sin embargo, el shell sí interpreta otros caracteres especiales, como '$' y '\'.

El **carácter $** hace que el shell busque una variable de entorno que coincida con ése nombre. Las variables de entorno se parecen a las constantes simbólicas de C; una de ellas, PATH, es la ruta de búsqueda de ejecutables. Otra variable de entorno importante es HOME, que contiene el nombre del directorio raíz del usuario.

Otros caracteres especiales del shell permiten:

- **&:** Arrancar un programa en "background". El "shell" no espera a que el programa concluya, y nos pide otro comando. El programa se ejecuta de

manera concurrente con el “shell”, y se desvincula del terminal, de manera que aún podremos teclear órdenes y ver sus resultados.

- **Redirección:** Es posible alterar desde el la conexión por defecto de la entrada y salida “standard” y la salida de error de los programas que ejecutemos. Así ‘>’ redirige la salida a un fichero, ‘<‘ hace lo mismo con la entrada, 2> redirige la salida de error. Si lo que se desea es añadir datos a un fichero existente se utiliza ‘>>’. Para interconectar programas (salida con entrada “standard”) se utiliza ‘|’ (barra partida). “cat fich.txt | grep abc” muestra a la salida sólo las líneas de “fich.txt” que contienen “abc”.

1.2 Comandos básicos

Los comandos pueden ser ejecutados por el “shell” directamente (los “built-in”, o incorporados), o, más frecuentemente, programas ejecutables que se encuentran en directorios tales como **/bin** o **/usr/bin**. Por lo general las opciones se indican en la línea de comandos precedidas por el signo ‘-’. Muchos comandos proporcionan una ayuda resumida al utilizarlos con la opción “–help” (con dos guiones), por ejemplo “gcc –help”. Para una ayuda más general pueden utilizarse los comandos **man** (Ubuntu) o **use** (QNX).

ls: Lista ficheros. “ls <directorio>” muestra el contenido de un directorio. “ls *.c” lista los ficheros .c. Opciones: “-l” da un listado completo, incluyendo permisos y otros detalles. “-t” ordena por tiempo de modificación. “-r” ordena al revés.

cat: Muestra un fichero por pantalla (lo hace en cualquier caso, pero mejor asegurarse que sea de texto, o de lo contrario los caracteres no imprimibles pueden cambiar la configuración de la consola). Es similar al “type” de MS-DOS.

cd: Cambiar el directorio de trabajo: “cd /usr/home/dir1” cambia al directorio “/usr/home/dir1”. Podemos volver al directorio raíz con “cd $HOME”, y al directorio “padre” del actual con “cd ..”.

diff: Compara ficheros y muestra las líneas en las que hay diferencias. El comando “diff fich1.c fich2.c” muestra las diferencias entre los ficheros “fich1.c” y “fich2.c”. Si escribimos “diff –r dir1 dir2” **diff** compara de manera recurrente todos los ficheros de los directorios “dir1” y “dir2”, incluyendo subdirectorios. Normalmente sólo tiene sentido usar este comando con ficheros de texto.

echo: Muestra en pantalla el argumento. Así “echo mensaje” imprime “mensaje”. Resulta más útil para ver el valor de variables de entorno, como en “echo $PATH” que muestra el valor actual de la variable de entorno “PATH”.

grep: Localiza cadenas de caracteres en un conjunto de ficheros. “grep printf *.c” nos dice en qué ficheros .c del directorio actual se llama a **printf**.

history: Se trata de un comando integrado que muestra una lista de los comandos ejecutados anteriormente acompañados de un número de referencia. Para volver a ejecutar uno de ellos basta teclear “r <n>” en QNX o “!<n>” en Ubuntu, dónde <n> es el número de referencia que proporciona **history**.

kill: Envía una señal a un proceso, identificado por su "pid". Tecleando "kill <pid>" envía una señal SIGTERM (terminación por defecto); este es el procedimiento habitual para eliminar un proceso. Tecleando "kill -9 <pid>" se envía una señal SIGKILL (terminación incondicional); sólo debe utilizarse como última opción. Tecleando "kill -s <nombre de señal> <pid>" puede utilizarse cualquier señal, identificada por su nombre. También existen comandos que permiten enviar señales utilizando el nombre del proceso, pero no están estandarizados; en QNX existe **slay**, y en Ubuntu **pkill**.

mkdir: Crea un nuevo directorio.

man (Ubuntu)**:** Permite acceder a los manuales. Uso: "man <nombre de comando o llamada>". Por ejemplo: "man printf", "man man", "man ls". No hace falta decir que se trata de un comando útil. En Ubuntu está disponible para todos los comandos y la mayoría de las funciones de biblioteca.

ps: Permite conocer qué procesos están activos. Su comportamiento es algo distinto en Ubuntu y en QNX. Sin opciones, **ps** muestra en Ubuntu sólo procesos relacionados con la consola en la que se ejecuta, mientras que en QNX muestra además otros procesos. Con la opción "-A" proporciona en ambos un listado de todos los procesos. Con "-l" la salida contiene datos adicionales. También puede controlarse la salida con la opción "-o": "ps –o args,pid,threads,tid" muestra en QNX el comando que dio lugar al proceso, su "pid", el número de hilos y el identificador de cada hilo. Para conseguir el mismo efecto en Ubuntu hay que teclear "ps –A –T args,pid,thcount,tid". El comando **ps** es interesante para conocer el "pid" de un proceso, y también para saber si alguno se ha quedado "colgado" y no acaba.

pwd: Nos indica en qué directorio nos encontramos.

rm: Borra un fichero. Es similar al "erase" de MS-DOS. La opción "–r" permite borrado recurrente de un directorio y de sus subdirectorios. El comando **rm** debe utilizarse con cuidado, porque no utiliza ninguna carpeta de "papelera"; simplemente destruye los archivos.

rmdir: Borra un directorio (si está vacio).

tar: Permite crear un fichero que contiene otros ficheros, e incluso árboles de directorios, de manera que se pueden manipular como una unidad. "tar cvf mitar.tar a b c" incluye en el fichero "mitar.tar" los ficheros "a", "b" y "c"; si alguno es un directorio, incluye todo su contenido. El comando "tar xvf mitar.tar" extrae todo el contenido del fichero "mitar.tar" expandiéndolo a su estado original. Para saber qué contiene un fichero .tar basta hacer "tar tvf mitar.tar" y muestra todos los archivos individuales que contiene. El comando **tar** no incluye por defecto ningún tipo de compresión; para comprimir archivos puede utilizarse **gzip**.

use (QNX): Cumple una función parecida a la de **man** en Ubuntu, aunque el resultado suele estar más resumido. Por ejemplo: "use ps".

who: Dice qué usuarios están conectados a la máquina.

1.3 Compilación

Tanto en QNX como en Ubuntu El compilador se llama ‘gcc’, y normalmente basta con compilar del siguiente modo:

gcc –o programa prog1.c prog2.c prog3.o

Así se creará un ejecutable llamado “programa”, con los fuentes “prog1.c” y “prog2.c” y el objeto reubicable “prog3.o”. Estos módulos se compilan si es necesario y se enlazan junto con las bibliotecas predefinidas. A veces es necesario añadir bibliotecas de manera explícita. Por ejemplo, si se utilizan funciones matemáticas (coseno, seno...) hay que añadir la biblioteca “m” con la opción “-lm”:

gcc –o math math.c –lm

Cuando se utilizan llamadas POSIX en QNX no hay que añadir ninguna biblioteca, pero sí en Ubuntu. Se trata de “rt” (de “real time”). Cuando además se utilizan los hilos POSIX es necesario añadir “pthread” (de “POSIX threads”):

gcc –o prog_posix prog_posix.c –lrt

gcc –o prog_posix_hilos prog_posix_hilos.c –lrt -lpthread

Si es preciso compilar los módulos separadamente, habrá que hacer:

gcc -c prog1.c

gcc -c prog2.c

gcc –o programa prog1.o prog2.o

En este caso sólo será preciso compilar a objeto (opción “-c”) los módulos que han cambiado; el resto puede utilizarse directamente en el último paso, que consiste únicamente en la edición de enlaces (“linking”).