

4. El Intérprete de Comandos o *shell*

4.1 Definición

Los *shells* constituyen el interfaz entre los usuarios del sistema y el sistema operativo. Son por tanto programas del sistema que tienen la capacidad de recibir comandos introducidos por los usuarios del sistema y encargarse posteriormente de que el sistema operativo lleve a cabo las acciones requeridas. Generalmente, cada una de las órdenes del usuario será ejecutada por un programa del sistema que a su vez hará uso de llamadas al sistema, tal y como se contempló en los capítulos 1 y 3.

El *shell* o intérprete comandos de Linux, recibe una cadena de caracteres que el usuario teclea y procede a su análisis. Dicha cadena contendrá generalmente uno – o varios – comandos y opcionalmente, una serie de parámetros. El *shell* extrae de ella el nombre del comando y busca un programa del sistema de igual nombre (generalmente dentro del directorio `/bin`) para crear un proceso que se encargue de la petición. Es decir, el *shell* no ejecuta el comando sino que inicia su ejecución, al crear un nuevo proceso que se encargue de ello. El *shell* pasa los posibles parámetros recibidos, y espera a que el proceso finalice. Una vez el *shell* reciba la respuesta generada la mostrará por pantalla (es decir, se la transmitirá al usuario).

Puede ocurrir que la cadena de caracteres no contenga ningún comando. En este caso, el *shell* no encontrará el programa correspondiente y devolverá al usuario un mensaje de error.

4.2 Introducción a los *Shells*.

Siempre que un usuario se conecta a un sistema Linux, se encontrará situado en un directorio determinado y se estará ejecutando un *shell* o intérprete de comandos. Es decir, el sistema mostrará el *prompt* del sistema (`#` en el caso del superusuario o usuarios de su grupo, y `$` para el resto de usuarios) indicando que el sistema – y más en concreto el intérprete de comandos – está preparado para recibir una orden o comando.

Los sistemas operativos pueden proporcionar diversos *shells*. Del mismo modo, la mayoría de distribuciones de Linux también vienen acompañadas de diversos *shells*. Varios de ellos, como es el caso de Bourne, C, T y Korn, son comunes a todas ellas y pueden considerarse estándares,

4.2.1 EL *SHELL* BOURNE

Este *shell* es el más antiguo y probablemente por ello, el que menos utilidades ofrece. Por ejemplo, no permite la recuperación y edición de órdenes ya emitidas (lo que en MS-DOS se conoce como *doskey*). No obstante, se trata del *shell* en el que está basado el *bash* (Bourne Again *Shell*), que es el predeterminado de Linux, y que incluye un mayor número de utilidades.

4.2.2 EL *SHELL* C

El *shell* C, conocido como *csh*, fue desarrollado por Bill Joy en la Universidad de California, en Berkeley, con el propósito de construir un *shell* que fuera más adecuado para labores de programación. Tiene características diferentes respecto al anterior, incluso en cuanto a la sintaxis – la manera de introducir las órdenes. De hecho, su sintaxis es muy parecida al lenguaje de programación C, lo que no ocurre con prácticamente ninguno de los demás *shells*, que por tanto no pueden ejecutar secuencias de sentencias escritas para este *shell*. El programa ejecutable correspondiente a este *shell* se encuentra en el archivo `/bin/csh`.

4.2.3 EL *SHELL* KORN

Se trata de un *shell* “híbrido” que trata de reunir las características del *shell* C, pero con la sintaxis del *shell* Bourne. De hecho, algunas distribuciones como Slackware 96 no incluyen una copia del *shell* Korn.

4.2.4 EL *SHELL* BASH

Es el predeterminado de Linux y se encuentra en `/bin/bash`. Proporciona optimizaciones como por ejemplo, un historial de ordenes y la capacidad de edición y/o terminación de las mismas. Está incluido en todas las distribuciones de Linux, y en algunas de ellas se instala por defecto otro.

4.2.5 LA VARIABLE *SHELL*

Para saber qué *shell* se está utilizando en un momento dado, puede accederse al valor de la variable de entorno¹⁶ *SHELL*, mediante el comando:

```
$ echo $SHELL
```

¹⁶ Ver glosario.

La orden `echo` visualiza en la pantalla del terminal lo que haya escrito tras la palabra `echo`. La variable `SHELL` contiene el nombre del *shell* actual; `$SHELL` es el valor de dicha variable.

Cada usuario puede decidir utilizar un *shell* predeterminado distinto. El *shell* por defecto se especifica en el momento de dar de alta a cada usuario y queda almacenado en el fichero `/etc/passwd`. De cualquier manera, es posible variar el *shell* por defecto de un usuario siempre y cuando se goce de los privilegios para ejecutar la orden `usermod`, y en tiempo de ejecución cada usuario puede cambiar de *shell* cuando quiera y cuantas veces quiera.

4.3 El entorno de la sesión.

Linux configura el entorno de trabajo para un usuario, durante el proceso de *login*, es decir, desde que introduce su nombre de usuario y su contraseña hasta que se muestra el *prompt* del sistema. Este entorno incluye configuraciones y datos que controlan su sesión de trabajo, y puede modificarse libremente en cualquier momento. Básicamente tiene dos componentes: el entorno del terminal y el entorno del *shell*.

4.3.1 EL ENTORNO DEL TERMINAL

Se trata de información relativa al terminal del usuario. Si el usuario de Linux se conecta desde un terminal de la propia máquina en la que está instalado el sistema operativo, el monitor y el teclado del PC componen dicho terminal.

La gestión del mismo la lleva a cabo el controlador de dispositivos del sistema operativo. De hecho, es el controlador de dispositivos el que en primera instancia recibe los caracteres que se introducen por el teclado y posteriormente los envía al *shell*, que los examina y trata. Una vez que el *shell* obtiene los datos o información para proporcionar al usuario se los envía al controlador de dispositivos, que será el que finalmente los muestre por pantalla.

Para que estas operaciones sean posibles, y puesto que en Linux todos los dispositivos se manipulan como si fuesen ficheros, es necesaria la existencia de programas que realicen las transformaciones necesarias a los flujos de caracteres, para posibilitar el trabajo con cada dispositivo. Estos programas precisan de parámetros que identifiquen el dispositivo concreto con el que se está trabajando, así como de caracteres especiales o caracteres de control, que hacen las veces de marcadores de final de archivo y final de línea para el *shell* y señales de control que se

pueden enviar a los programas en ejecución, como por ejemplo, una señal de interrupción.

4.3.2 EL ENTORNO DEL *SHELL*.

Se trata de información relativa tanto al *shell* como a los programas que sobre el mismo se ejecutan. Así, es posible configurar aspectos como el nombre del *shell* a utilizar, el directorio de trabajo para cada programa, etc.

Estos parámetros propios del *shell* se almacenan en ciertas variables denominadas “variables del *shell*”, muchas de las cuales se definen durante el proceso de *login* de un usuario al sistema o establecimiento de sesión. Aunque algunas de estas variables son de sólo lectura, es posible variar los valores de otras muchas, simplemente haciendo uso de la sintaxis: `VARIABLE=valor`, tecleada directamente al lado del *prompt*.

Las variables del *shell* hacen en ocasiones referencia a parámetros que influyen en la ejecución de los programas, como puede ser el tipo de terminal (variable `TERM`), o el *shell* en concreto que se está utilizando (variable `SHELL`). A estas variables se les denomina variables del entorno de ejecución. Probablemente la más importante es la variable `PATH` en la que se almacenan las rutas o directorios en los que el *shell* debe buscar los programas correspondientes a las órdenes o comandos que se introduzcan desde la consola. Entre los directorios almacenados en ella siempre deberán estar `/bin`, `/sbin` y `/usr/bin`, que son los directorios en los que se guardan por defecto los programas correspondientes a las órdenes de Linux.

Para ver el contenido de las variables especiales del *shell* puede usarse la orden `env`. Las variables más habituales del *shell* son las siguientes:

Nombre	Contenido
HOME	Nombre completo de la ruta del directorio de usuario.
SHELL	Nombre del <i>shell</i> actual.
MAIL	Nombre completo de la ruta del buzón de correo.
LOGNAME	Nombre de entrada del usuario al sistema.
PATH	Directorios que el <i>shell</i> revisa en busca de órdenes.
TZ	Zona horaria.

SECONDS	Número de segundos desde que se invocó el <i>shell</i> .
PS1	<i>Prompt</i> del sistema.
TERM	Tipo de terminal que se está utilizando.

Tabla 5: Variables más comunes del *shell*.

Independientemente de las variaciones que se establezcan mediante la sintaxis reseñada en la página anterior, es posible personalizar el *shell* cambiando los valores de las variables del *shell*, editando el fichero `.profile` si se utilizan el *shell* Bourne o el *bash*. Si se utiliza el *shell* C habrá que editar el fichero `.login`.

4.4 El análisis de orden del *shell*.

En Linux, cuando se introduce una orden desde la consola, ésta es tratada por el *shell*, que en primero la divide en sus partes, y luego, identifica la orden, los indicadores y los parámetros, para poder invocar al programa correspondiente.

No obstante, en una orden, el usuario ha podido hacer uso de caracteres comodines con diversos significados, y que por ejemplo pueden hacer referencia a varios archivos, que hay que localizar. En Linux, los comodines son los mostrados en la tabla siguiente:

Comodín	Significado
*	Cualquier conjunto de caracteres.
?	Un solo carácter
[]	Un solo carácter dentro de una serie. [0-3] puede ser 0, 1, 2 o 3.

Tabla 6: Comodines más frecuentes.

4.4.1 CONEXIÓN CON PIPES O TUBERÍAS.

Los *pipes*, tuberías o conducciones permiten la “conexión” o comunicación de dos o más procesos (y por tanto, de dos o más comandos). Cuando se especifica un *pipe* entre dos comandos o procesos, se está indicando que la salida del primero de ellos debe tomarse como entrada para el segundo. Del mismo modo, la salida del 2º puede ser considerada la entrada del 3º, y así sucesivamente. Este mecanismo evita al usuario tener que definir un fichero para almacenar la salida de un programa, y pasar después al otro programa este fichero como parámetro de entrada. No obstante, la

implementación interna de un *pipe* en un sistema *Unix-like* – algo transparente al usuario –, es la de un fichero temporal, de tamaño limitado. A nivel de intérprete de comandos, se especificará con el símbolo “|”; es decir: `$ <comando1> | <comando2> ...`

Así, por ejemplo, el comando `$ps al | grep 2001` provocaría la visualización en pantalla de los procesos lanzados por el usuario 2001 (verdaderamente se mostrarían en pantalla todos los procesos en los que aparezca “2001” entre su información. Otros ejemplos de comandos unidos mediante pipes, los encontramos en el cuaderno de laboratorio.

4.4.2 REDIRECCIÓN DE ENTRADAS Y SALIDAS.

Habitualmente, muchos programas hacen uso de dispositivos de E/S como teclados y pantallas. Linux considera el teclado como el dispositivo de entrada estándar y tiene asociado el archivo `stdin`, y la pantalla como la salida estándar, siendo `stdout` el archivo asociado a la misma. No obstante, es posible hacer que la entrada y salida estándar de un programa no se correspondan con estos dispositivos sino con otros.

Para redireccionar momentáneamente la entrada y/o la salida de un comando, se puede hacer uso de los operadores de redireccionamiento, que son: el símbolo “<” para la entrada, y el símbolo “>” para la salida. Con ellos, es posible hacer que la entrada de un comando sea un archivo determinado, o bien, que la salida de un comando se almacene en un fichero en vez de mostrarse por pantalla.

Así, la utilización por ejemplo del comando `$ ls -l > fichero1` provocará la creación de un archivo de nombre *fichero1* en el que se depositará el resultado de la ejecución de este comando. Si ya existía un fichero del mismo nombre, su contenido se perderá. También se puede utilizar la cadena “>>” para redireccionar la salida, y en este caso si el fichero destino ya existe, el resultado de la ejecución de este comando se concatena con su contenido, al final del mismo.

4.5 Escritura de programas con el shell

Al igual que en MS-DOS se puelden escribir programas de comandos, como por ejemplo el contenido en `AUTOEXEC.BAT`, en Linux también se pueden escribir programas con el *shell*. Simplemente es necesario conocer la sintaxis del manejo de las variables y de las estructuras de control.

Como ocurre con la escritura de programas fuente, es deseable la inserción de comentarios en los programas del *shell*. Esto es posible mediante el símbolo “#”; las líneas que comienzan por este símbolo se omitirán.

Otro elemento de gran utilidad es la orden *ECHO*, que permite mostrar mensajes por pantalla. De hecho, *ECHO* muestra por pantalla sus argumentos, es decir, lo escrito después del comando.

4.5.1 USO DE LAS VARIABLES EN LOS PROGRAMAS DE *SHELL*

Utilizar una variable en un programa de *shell* consiste en asignarle un valor para acceder posteriormente al mismo. Existen varias maneras distintas de asignar valores a las variables:

- Asignación directa: Su sintaxis es `<nombre_de_variable> = <valor>`. Por ejemplo, `nombre = "Gregorio Sánchez"` asigna a la variable `nombre` el valor “Gregorio Sánchez”.
- Utilización de la orden `read`: esta orden asigna a una variable la cadena que se introduzca por teclado. Así, “`read nombre`” guarda en la variable `$nombre` lo que se introduzca por el teclado después de interpretarse la orden `read`.
- Parámetros de línea de órdenes: Cuando el *shell* interpreta una orden, asigna nombres de variables a cada elemento (palabra) de la línea de órdenes. Los nombres de dichas variables son “`$i`”, siendo “`i`” un número entero entre “0” y “N”, que representa el orden en que fueron tecleadas las palabras, de manera que `$0` representará al nombre del comando en sí, `$1` al primer argumento y así sucesivamente.
- Utilización de la salida de una orden: Para asignar, por ejemplo, el nombre del directorio actual en la variable `dir`, se pondrá `dir = `pwd``. Es importante resaltar que los caracteres entre los que está la orden Linux son acentos graves.
- Implementación de la versión ANSI estándar de la función `main`, de la manera `void main (int argc; char * argv[])`, que permite capturar parámetros introducidos en la línea de comandos y pasárselos después al programa invocado.

4.5.2 PROGRAMACIÓN CON ESTRUCTURAS DE CONTROL

Al igual que ocurre con los lenguajes de programación tradicionales, para realizar programas, *scripts* o guiones del *shell*, se dispone de estructuras alternativas y repetitivas.

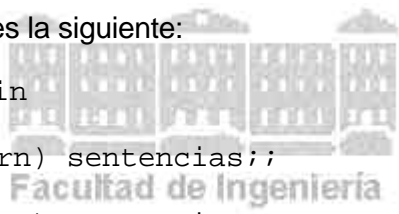
Estructuras alternativas:

Trabajar con estructuras alternativas implica que cierta rama de la estructura se recorrerá (se ejecutará) exclusivamente si se verifica cierta condición. En los *scripts* del *shell*, la formulación de condiciones se lleva a cabo mediante la orden `test`.

La estructura alternativa “`if ... then ... else ... fi`” permite seleccionar entre dos posibilidades de acción en función de que se cumpla o no cierta condición. Además, dado que la rama *else* es opcional, por lo que esta estructura puede ser considerada como una alternativa de una o de dos ramas.

La estructura `case` permite escoger entre “`n`” posibilidades de acción, según el valor de una variable. Su sintaxis es la siguiente:

```
case word in
    pattern) sentencias;;
    pattern) sentencias;;
    ...
esac
```



Estructuras iterativas:

Hay dos estructuras iterativas o bucles: *for* y *while*. Con la estructura *bucle for* se puede especificar un grupo de archivos o de valores para ser utilizados por un conjunto de comandos. Así, para copiar todos los archivos `.txt` a otro directorio se puede escribir el siguiente conjunto de órdenes:

```
for i in *.txt
do
    cp $i otro_dir/$i
done
```


El *shell* interpreta la expresión `for i in *.txt` y hace que la variable “i” tome el nombre de cualquier archivo `.txt` del directorio actual, para después utilizar el valor de la variable, `$i`, entre las palabras clave `do` y `done`.

El *bucle while* examina el estado de salida de una orden, de manera que las órdenes especificadas entre las palabras clave `do` y `done` se ejecutarán mientras se cumpla la condición.

En el siguiente ejemplo se comprueba si se ha recibido correo electrónico. Para ello, se compara el buzón (`$MAIL`) con el valor anterior del buzón mediante el comando `diff`, que permite realizar comparaciones entre archivos, y obtener información sobre posibles diferencias entre ellos.

```
cp $MAIL omail

# comenzamos con una copia del fichero

while diff omail $MAIL > /dev/null
do
    # aquí está el bucle
    cp $MAIL omail
    sleep 30
done

echo "Tienes un e-mail!"
```



4.6 “Exportación” de variables a un nuevo shell.

Cuando un usuario se conecta al sistema, el *shell* comienza su ejecución asignando valores a ciertas variables especiales para definir su propio entorno, y siempre que se emite una orden se inicia un nuevo proceso, o una nueva instancia del *shell*, que heredará muchas de las características (no todas) del *shell* existente. Se ejecutará en el directorio actual y además hará uso de muchas de las variables ya definidas.

En Linux, las variables son accesibles únicamente para el proceso que las crea y/o modifica, y para todos los procesos que éste cree. Así, todas las variables establecidas por el *shell* de entrada al sistema, estarán disponibles para todos los procesos que se cree, para la ejecución de todos y cada uno de los comandos que él introduzca.

En cambio, una variable establecida dentro de un *shell* creado para la ejecución de un comando – un *shell* hijo – conserva su valor únicamente dentro de ese *shell*, de manera que un valor desaparece o se vuelve a restaurar a su valor original al salir de él. Para evitar esto, se puede usar la orden `export nombre_variable`, que traslada el valor de la variable nombrada a los *shells* “posteriores”.

