

Uso básico de la shell de Unix/Linux

Miguel Ortuño
Escuela de Ingeniería de Fuenlabrada
Universidad Rey Juan Carlos

Abril de 2024



© 2024 Miguel Angel Ortuño Pérez.
Algunos derechos reservados. Este documento se distribuye bajo la
licencia *Atribución-CompartirIgual 4.0 Internacional* de Creative
Commons, disponible en
<https://creativecommons.org/licenses/by-sa/4.0/deed.es>

- 1 Anexo: Uso básico de la shell de Unix/Linux
 - Introducción a la shell
 - Conceptos básicos sobre la shell
 - Órdenes básicas de la shell de Unix

Uso básico de la shell de Unix/Linux

Hasta ahora has manejado ordenadores usando interfaces gráficos de usuario, con ratón (o similar), ventanas, menús, botones, etc. Aquí aprenderas a manejar lo más elemental de la *shell* de Unix/Linux.

- Unix es una familia de sistemas operativos, a la que pertenece Linux.
- La *shell* es un programa que nos permite manejar nuestro sistema usando solo teclado y pantalla en modo texto, sin gráficos ni ratón.

Es una forma de trabajar más antigua y un poco más complicada que los interfaces gráficos, pero con con ventajas importantes.

Terminal

A la combinación de teclado y pantalla sin gráficos se le llama *terminal* o también *consola*. Desde el terminal, manejamos la shell. Con un terminal se puede trabajar de dos formas:

- En *local*, esto es, usar el ordenador que tenemos delante de nosotros.
- En *remoto*. Usando un cliente del protocolo ssh, como p.e. SmarTTY o Terminal, podemos trabajar en una máquina en la otra punta del mundo, exactamente igual que si la tuviéramos a un metro.

Sesión

En nuestro caso, una *sesión* es un intercambio de información entre el usuario y el ordenador.

- Empieza cuando el usuario (o su cliente, p.e. SmarTTY) introduce sus credenciales (nombre de usuario y contraseña) en el sistema.
- Concluye cuando el usuario lo decide o cuando algún error lo fuerza.
- En una sesión en modo texto, el usuario escribe órdenes en el terminal y el ordenador devuelve los resultados de las órdenes. A las órdenes también se les llama *comandos*.

Ficheros y directorios

Fichero y *directorio* son los nombres tradicionales en Unix para lo que normalmente conoces como *documento* y *carpeta*. Podemos considerarlos sinónimos, usa los que prefieras.

- Hay que tener cuidado con la palabra *fichero*, porque en Unix, los directorios son un caso particular de fichero. En otras palabras: cuando decimos *fichero* podemos referirnos a un fichero *ordinario* (un documento) o a un directorio.

Nombres de fichero (y directorio)

Como sabes, cada fichero tiene un nombre y tal vez una extensión. La extensión es el sufijo del nombre, a partir del último punto.

- Por ejemplo en el fichero llamado `hola mundo.pas` la extensión *pas* indica que se trata de un fichero en código fuente de Pascal.

Esto es igual que en Windows. Pero en los nombres de ficheros en Unix/Linux hay dos diferencias importantes respecto a Windows:

- 1 Uso de espacios.
- 2 Uso de mayúsculas.

- ❶ No es recomendable que un nombre incluya espacios.
 - En Windows es frecuente usar nombres con espacios, como `primer ejemplo.docx`
 - En la shell esto serían dos ficheros: por un lado `primer` y por otro `ejemplo.docx`
 - Hay varias soluciones para este problema, aquí recomendamos usar la barra baja (`_`) en vez del espacio.
`primer_ejemplo.docx`
- ❷ Mayúsculas y minúsculas son letras distintas. Si un enunciado te pide por ejemplo un fichero llamado `holamundo.pas`, no puedes llamarlo `Holamundo.pas`, es un nombre distinto.

Directorios

- Directorio *home*

Cuando un usuario tiene cuenta en una máquina, puede escribir en diversos sitios, pero se reserva para él un directorio donde guardar su trabajo. En español se puede llamar *carpeta personal*, *directorio hogar*, etc. Pero posiblemente lo más habitual es llamarlo *home*, en inglés, a secas. Se representa por la virgulilla (~).

Virgulilla en el teclado:

- Windows y Linux: AltGr 4 (y un espacio)
- macOS: opt ñ

Cuidado: si copias y pegas una virgulilla desde un pdf, estarás pegando una virgulilla diferente, más pequeña, que no funciona.

- Directorio actual:
En una sesión, el usuario *está* en cierto directorio: el directorio actual. Siempre que el usuario escriba una orden sobre un directorio, mientras no indique lo contrario, se supone que se refiere al directorio actual.
- Subdirectorio:
Directorio que está dentro de otro directorio.

Argumento (de una orden)

Cuando escribimos órdenes de shell, podemos añadirles parámetros adicionales a los que se llama *argumentos*.

- Ejemplo:

```
cd ..
```

Es la orden `cd` con el argumento `..`.

- Para indicar cual es el comportamiento de una orden cuando no especificamos argumentos, decimos *por omisión la orden hace ...*

Opción

Una orden puede incluir *opciones*. Las opciones modifican el comportamiento de las órdenes de la shell. Se escriben como un guión seguido de una o más letras.

- Ejemplo:

```
rm -r probando
```

Esto ejecuta la orden de shell `rm`, con la opción `r`. El argumento es *probando*.

Observa que:

- No es lo mismo la opción que el argumento.
- No es lo mismo una letra minúscula que una mayúscula.
- En la opción (u opciones), no puede haber espacios entre el guión y la(s) letra(s)

```
rm - r probando # ¡Esto está mal!
```

Prompt

El *prompt* es la línea de texto que vemos en el terminal cuando la shell está preparada para que escribamos una orden. P.e.

```
jperez@f-l-vm01:~$
```

Es importante que sepamos interpretar el prompt porque aporta mucha información útil. En este ejemplo vemos.

- Nuestro nombre de usuario (jperez)
- El nombre de host (f-l-vm01)
- El directorio actual (virgulilla, es decir, *home*)

Observa que:

- La arroba separa el nombre de usuario del nombre de host.
- Los dos puntos separan el nombre de host del directorio actual.
- El dólar indica el fin del prompt, y que podemos escribir a continuación.

Path

Path significa *trayecto*. Es un texto que de forma compacta especifica dónde está un fichero.

Ejemplo:

- `holamundo.pas`

Esto es un nombre sin *path*. No especifica dónde está

- `~/fpi/practica01/holamundo.pas`

Esto es un nombre con *path completo*. Significa que en mi directorio *home*, hay un directorio llamado *fpi*, dentro, un subdirectorio llamado *practica01*, y dentro, un fichero llamado *holamundo.pas*.

Observa que los nombres de directorio están separados por el carácter barra (/), igual que en las direcciones de internet.

En Windows para este propósito se emplea la barra invertida (\)

Órdenes básicas

Las órdenes básicas que necesitarás aquí son:

Con directorios:

- Ver su contenido (`ls`)
- Ver su estructura (`tree`)
- Entrar en un directorio (`cd`)
- Salir de un directorio (`cd ..`)
- Crear un directorio (`mkdir`)
- Borrar un directorio (`rm -r`)

Con ficheros:

- Borrar un fichero (`rm`)
- Si es un fichero de texto, editarlo (`nano`)
- Si es un programa en pascal, compilarlo (`fpc`)

Abreviatura de *list*. Sirve para ver el contenido de un directorio.

- `ls`
Muestra un listado de los ficheros y subdirectorios del directorio actual.
- `ls -l`
Listado *largo*. No solo vemos el nombre de los ficheros, también su fecha de creación, tamaño y algunos otros *atributos*.

cd

Abreviatura de *change directory*. Sirve para cambiar el directorio actual, esto es, para *entrar* en un directorio o *salir* de él.

- cd ejemplo

Si en el directorio actual hay un subdirectorio llamado *ejemplo*, entraremos en él.

- cd

La orden *cd* sin indicar ningún argumento, nos lleva al *home*, esto es, equivale a `cd ~`.

- cd ..

Estos dos puntos (en horizontal y sin espacios por medio) representan al directorio padre de cada directorio. Esta orden hace que el directorio actual pase a ser el directorio padre. En otras palabras, salimos del directorio en el que estamos.

mkdir

Abreviatura de *make directory*. Sirve para crear directorios.

- `mkdir fpi`

Crea un directorio llamado *fpi* en el directorio actual. Si por ejemplo mi directorio actual es el *home*, esta orden creará `~/fpi`.

- `mkdir ~/fpi`

En este caso indico el nombre del fichero con su path completo. Por tanto, se creará exactamente ahí, sin importar cual sea mi directorio actual.

rm

Abreviatura de *remove*. Sirve para borrar uno o más ficheros.

- `rm ejemplo`

Borra un fichero llamado *ejemplo* del directorio actual. Si es un directorio no lo borrará.

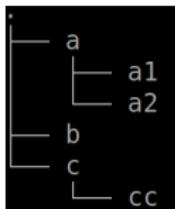
- `rm -r ejemplo`

La opción `-r` significa *recursive*. Con esta opción la orden `rm` borra ficheros y también directorios, recursivamente. Esto es, recorriendo y borrando sucesivamente todos los subdirectorios que haya.

tree

Sirve para ver la estructura en forma de árbol de todos los ficheros, directorios y subdirectorios contenidos dentro de mi directorio actual.

Si `tree` nos devuelve por ejemplo esta salida:



Significa que

- En el directorio actual tenemos tres elementos: a, b y c. Observaremos que a y c son subdirectorios, pero en el caso de b no podemos distinguir si se trata de un subdirectorio vacío o un fichero.
- Dentro de a, están los ficheros o directorios a1, y a2.
- Dentro de c, el fichero o directorio cc.

Visualización de salidas largas

En ocasiones, un comando puede generar una salida con muchas líneas, de forma que no quepan en tu terminal. Para ver la salida completa, tienes varias soluciones:

- 1 Usar la barra de desplazamiento vertical de tu terminal (si está disponible).
- 2 Añadir al comando la barra vertical y el comando `less`. P.e.

```
ls | less  
tree | less
```

- De esta forma, podrás usar las flechas arriba y abajo del teclado para desplazarte por la salida.
- Para escribir la barra vertical, pulsa simultáneamente las teclas *alt gr* y *1*.

exit

exit

Finaliza la shell actual, por tanto cierra la sesión.

- Si no teníamos ningún programa funcionando y cerramos la ventana del terminal, el efecto es el mismo. Pero es una buena costumbre cerrarlo todo ordenadamente.
- Si había un programa funcionando, por ejemplo un editor de texto abierto, y cerramos la ventana *por las malas*, sin usar `exit`, podremos tener problemas en la siguiente sesión.

En este vídeo puedes ver una sesión básica

<https://youtu.be/70BUma0M4ao>