Introducción a Sistemas Operativos: Empezando

Clips 11 a 13
Francisco J Ballesteros

1. Obteniendo ayuda

La mayoría de los sistemas UNIX incluyen el manual en línea. Esto es, disponemos de comandos para acceder al manual. Saber utilizar el manual en UNIX es equivalente a saber utilizar Google en internet.

El manual se divide en secciones. Cada sección tiene una serie de páginas, cada una dedicada al elemento que documenta:

- La sección 1 está dedicada a comandos. Esto quiere decir que cada página de manual de la sección 1 documenta un comando (o varios).
- La sección 2 está dedicada a llamadas al sistema. Cada página de la sección 2 documenta una o varias llamadas al sistema.
- La sección 3 está dedicada a funciones de la librería de C que podemos utilizar para programar en dicho lenguaje. De nuevo, cada página documenta una o varias funciones.
- La sección 8 documenta programas dedicados a la administración del sistema (por ejemplo, formateo o inicialización de discos, etc.).

Dependiendo del UNIX que utilizamos, el resto de secciones suelen variar. Para aprender a utilizar el manual podemos utilizar el comando *man* y pedirle la página de manual de *man* en la sección 1:

```
unix$ man 1 man
```

O simplemente

```
unix$ man man
```

Eso produciría un resultado similar al que sigue.

```
unix$ man man
man(1)                                  man(1)

NAME
       man  -  format  and  display the on-line
       manual pages

SYNOPSIS
       man [-acdfFhkKtwW] [--path] [-m  system]
       [-S section_list] [section] name ...

DESCRIPTION
       man formats  and  displays  the  on-line
       manual  pages.   If you specify section,
       man only looks in that  section  of  the
       manual.   name  is  normally the name of
...
```

En la mayoría de los casos, el texto de la página de manual no cabe en tu pantalla y tendrás que pulsar el espacio en el teclado para avanzar. Pulsando la tecla "q" (quit) puedes abandonar la página y pulsando "b" (backward) puedes retroceder. Tu manual documenta como navegar por el texto utilizando el teclado.

Si no indicamos qué sección del manual nos interesa y existen páginas en varias secciones con el nombre que hemos indicado, *man* nos mostrará una de ellas (o todas ellas, dependiendo del UNIX que utilicemos).

La primera línea de una página de manual describe el nombre de la página (*man*) y la sección en que se encuentra (1 en nuestro caso). Habitualmente escribimos "ls(1)" para referirnos a la página de manual *ls* en la sección 1. Normalmente, sigue una sección "NAME" que describe el nombre del comando (o función es una página de la sección 2 o 3) y una descripción en una sola línea del mismo.

La sección siguiente (*synopsis*) describe una guía rápida de uso (que es útil sólo si sabes utilizar ya el comando, o función, y quieres recordar algún argumento). Y a continuación puedes encontrar la descripción detallada del comando o función.

Si sigues avanzando, verás cerca del final otra sección denominada "*see also*" (ver también), que menciona otras páginas de manual relacionadas con la que estás leyendo. Esta sección es muy útil para descubrir otros comandos o llamadas relacionados con lo que estás haciendo.

La página de manual que acabamos de ver mencionará dos comandos que te resultarán muy útiles: *whatis* y *apropos*. El primero puedes utilizarlo para averiguar qué es un comando o función. Por ejemplo:

```
unix$ whatis man
man(1) - display manual pages
man.conf(5) - configuration file for man 1
man(7) - legacy formatting language for manual pages
```

Y puedes ver qué hace el comando *man*. Como hay varias páginas para *man*, *whatis* ha enumerado las que conoce.

El comando *apropos* lo puedes utilizar para buscar comandos y funciones casi del mismo modo que utilizas un buscador en internet. Por ejemplo, para ver cómo compilar nuestro programa en C...

```
unix$ apropos compiler
c++(1) - GNU project C and C++ compiler
cc(1) - GNU project C and C++ compiler
gencat(1) - NLS catalog compiler
rpcgen(1) - RPC protocol compiler
zic(8) - time zone compiler
```

La segunda página tiene buen aspecto, y podemos ahora ejecutar...

```
unix$ max cc
...
```

para ver la página de manual del compilador.

Cada vez que veas un comando en este curso, puedes utilizar el manual para ver cómo se utiliza. Lo mismo sucede con las llamadas que hamos en C. Si algunos trozos de las páginas de manual resultan difíciles de entender no hay que preocuparse. Puedes ignorar esas partes y buscar la información que te interesa. Una vez completes este curso no deberías tener problema en entender el manual.

Miremos ahora la página de un comando que ya hemos utilizado, *uname(1)*:

```
unix$ man uname
NAME
     uname - print operating system name

SYNOPSIS
     uname [-amnprsv]

DESCRIPTION
     The uname utility writes symbols representing one or more system
     characteristics to the standard output.

     The options are as follows:

     -a      Behave as though all of the options -mnrsv were specified.
...
```

El epígrafe *synopsis* muestra cómo utilizar el comando de forma rápida. Los argumentos (y opciones) que aparecen entre corchetes son opcionales. Así pues, podemos ejectuar

```
uname
```

o bien

```
uname -a
```

o quizá

```
uname -m
```

etc. El epígrafe *description* muestra habitualmente el significado de cada una de las opciones y podemos utilizarlo para ver qué uso tenía cada opción o para buscar una opción que produzca el efecto que queremos. Cuando buscamos un comando para hacer algo, resulta útil buscar un comando que haga algo similar o que tenga algo que ver y mirar si hay alguna opción que quizá consiga lo que andamos buscando.

Si miramos ahora la página de *who(1)* podemos aprender algo más:

```
unix$ man who
NAME
     who - display who is logged in

SYNOPSIS
     who [-HmqTu] [file]
     who am i
...
```

Esta vez la synopsis muestra varias líneas. Normalmente eso quiere decir que podemos utilizar el comando de cualquiera de esas formas, pero indica que no podemos combinar ambas en un sólo uso.

Por ejemplo, la segunda línea es precisamente

```
who am i
```

lo que indica que en este caso no se espera que podamos utilizar opciones. No obstante, la primera línea indica que podemos utilizar `who` con la opción `-H`, con la opción `-m`, etc.

Las opciones pueden indicarse por separado como en

```
ls -l -a
```

o pueden indicarse en un sólo argumento, como en

```
ls -al
```

o

```
ls -la
```

El efecto suele ser el mismo.

2. Utilizando ficheros

Antes de continuar y utilizar UNIX para escribir nuestros programas, resulta útil aprender algunos comandos y ver cómo utilizar el shell. Como hemos visto, podemos escribir líneas terminadas en un *enter* (llamado también fin de línea) para escribir comandos. Siempre que las escribamos en un terminal que ejecute un intérprete de comandos. Por ejemplo, para ver la fecha:

```
unix$ date
Wed May  4 17:32:32 CEST 2016
```

Como ya dijimos, "`unix$`" es el prompt del shell y nosotros hemos escrito "`date`" y pulsado *intro* a continuación. El shell ha ejecutado *date*, y dicho comando ha escrito la fecha y hora.

Para listar ficheros puedes utilizar el comando *ls*.

```
unix$ ls
bin    lib    tmp
```

O listar no sólo el nombre, sino también el tipo de fichero, permisos, dueño, fecha en que se modificaron y el nombre. A esto se le llama un listado largo:

```
unix$ ls -l
total 0
drwxr-xr-x  2 nemo  wheel  136 May  3 18:21 bin
drwxr-xr-x  2 nemo  wheel   68 May  3 16:31 lib
drwxr-xr-x  2 nemo  wheel  170 May  3 17:13 tmp
```

En este caso, hemos utilizado "`-l`" como argumento de *ls*. Este argumento produce un listado largo. En realidad, "`-l`" es una opción para *ls*. Si miras la página de manual *ls(1)* verás que la sinopsis muestra argumentos que comienzan por un "`-`" y muchas veces son un sólo carácter. Cada uno de estos caracteres se pueden utilizar como un interruptor o *flag* para modificar el comportamiento del comando. En nuestro caso, hemos activado el flag "`l`" utilizando la opción "`-l`". Sencillamente, los primeros argumentos que utilizamos al ejecutar un comando pueden ser opciones (que empiezan por un "`-`" y siguen con los caracteres de las opciones). Por ejemplo, la opción "`k`" de *ls* utiliza tamaños en Kbytes y la opción "`s`" muestra el tamaño para cada fichero. Sabiendo esto...

```
unix$ ls -ks /bin/ls
16 /bin/ls
```

vemos que el fichero `/bin/ls` contiene 16 Kbytes. Como verás, puedes decir a *ls* qué ficheros quieres listar escribiendo su nombre como argumentos.

Podríamos haber ejecutado:

```
unix$ ls -k -s /bin/ls
16 /bin/ls
unix$ ls -s -k /bin/ls
16 /bin/ls
unix$ ls -sk /bin/ls
16 /bin/ls
```

Pero fíjate en esto...

```
unix$ ls /bin/ls -sk
ls: -sk: No such file or directory
/bin/ls
```

Esta vez, hemos escrito el argumento "`/bin/ls`" antes de "`-sk`". Dado que dicho argumento no empieza por "`-`", *ls* entiende que se refiere al fichero o directorio que queremos listar. ¡Y entiende que no hay mas opciones! Cuando intenta listar el fichero "`-sk`", ve que dicho fichero no existe y escribe un mensaje de error para informarnos de ello. Las opciones debe estar antes del resto de argumentos.

Hay dos opciones más que resultan útiles con `ls`. La primera es `-a`, que hace que `ls` muestre también los ficheros cuyo nombre comienza por un "`.`", cosa que normalmente no hace `ls`. Estos ficheros suelen utilizarse para guardar configuración para diversos programas y normalmente no se desea listarlos, pero pueden estar en cualquier directorio y no los veremos salvo que utilicemos el flag `-a`.

La segunda opción es `-d`, que hace que `ls` liste la información de un directorio en sí mismo y no la de los ficheros que contiene, cuando pidamos a `ls` que liste un directorio.

Si quieres escribir más de un comando en una línea, puedes separarlos por un carácter "`;`", como en:

```
unix$ date ; ls
Wed May  4 17:36:55 CEST 2016
bin    lib    tmp
```

En otras ocasiones queremos escribir líneas muy largas y podemos utilizar un "`\`" justo antes de pulsar *intro* para *escapar* el fin de línea. El carácter "`\`" (*backslash*) se utiliza en el shell para quitarle el significado especial a caracters tales como *intro*, que habitualmente tienen significado para el shell (en este caso, ejecutar el comando terminado por el fin de línea). Por ejemplo:

```
unix$ date ;\
> date ;\
> date
Wed May  4 17:40:19 CEST 2016
Wed May  4 17:40:19 CEST 2016
Wed May  4 17:40:19 CEST 2016
```

Hemos pulsando *intro* tras cada "`\`" y tras el último "`date`". Como verás, el shell ha leído las tres líneas antes de ejecutarlas. Tras cada línea el prompt ha cambiado a "`>`" para indicarnos que el shell está leyendo una nueva línea como continuación del comando. Habría dado igual si ejecutamos:

```
unix$ date ; date ; date
...
```

Otro comando realmente útil, a pesar de lo poco que hace, es *echo*. Este comando hace eco. Se limita a escribir sus argumentos separados por espacios en blanco y terminados por un salto de línea. Por ejemplo:

```
unix echo$ uno y otro
uno y otro
```

¿Qué sucede si hacemos eco de un carácter especial como el "`;`"? Probemos...

```
unix$ echo uno ; otro
uno
-sh: otro: command not found
```

El shell ha visto el "`;`" y ha ejecutado dos comandos. Uno era *echo* y el otro comando era "`otro`", que no existe. Como no existe, el shell nos ha informado del error. Pero podemos ejecutar esto otro...

```
unix$ echo uno \; otro
uno ; otro
```

Como puedes ver, el *backslash* le quita el significado especial al "`;`". Así pues, el shell ejecuta un único comando (*echo*) con tres argumentos. Y *echo* se limita a hacer eco de ellos.

Puedes utilizar *echo* para ver qué argumentos reciben los comandos, como has comprobado en este ejemplo. Existen otras formas de pedir al shell que tome texto literalmente como una sóla palabra, sin que tenga significado especial ninguno de sus caracteres. La más simple es encerrar en comillas simples dicho texto. Por ejemplo:

```
unix$ echo 'uno ; otro'
uno ; otro
```

hace que el shell ejecute un sólo comando, *echo*. Esta vez, el comando recibe un único argumento (y no tres como antes). El argumento contiene el texto que hay entre comillas simples.

La opción "`-n`" de *echo* hace que *echo* no escriba el salto de línea tras escribir los argumentos. Por ejemplo, ejecutando

```
unix$ echo -n hola
holaunix$
```

la salida de *echo* ("`hola`") aparece justo antes del prompt del shell para el siguiente comando. Simplemente nadie ha escrito el salto de línea y el shell se ha limitado a escribir el prompt. Otro ejemplo:

```
unix$ echo a ; echo b
a
b
unix$ echo -n a ; echo  b
ab
unix$
```

Fíjate como el penúltimo *echo* ha escrito su argumento pero no ha saltado a la siguiente línea.

Podemos crear un fichero utilizando el comando *touch*. Por ejemplo:

```
unix$ ls
bin    lib    tmp
unix$ touch fich
unix$ ls
bin    fich    lib    tmp
```

Hemos utilizado el *argumento* "`fich`" para el comando *touch*. Como no existe dicho fichero, el comando lo crea vacío. Una vez creado, el comando *ls* lo ha listado.

Otra forma útil de crear fichero pequeños es utilizar *echo* y pedirle al shell que lo engañe para que escriba su salida en un fichero. Por ejemplo:

```
unix$ echo hola >fich
```

Ejecuta *echo* pero envía su salida al fichero `fich`. El comando no sabe dónde está escribiendo. El "`>`" le indica al shell que queremos que le pida a UNIX que la salida de *echo* se escriba en el fichero indicado.

Podemos ver el contenido del fichero utilizando el comando *cat*, que escribe el contenido de los ficheros que le indicamos como argumento:

```
unix$ cat fich
hola
```

Es posible copiar un fichero en otro utilizando *cp*:

```
unix$ cp fich otro
cp fich otro
unix$ ls
bin    fich    lib    otro    tmp
```

Y ahora podemos borrar el fichero antiguo utilizando *rm*:

```
unix$ rm fich
unix$ ls
bin    lib    otro    tmp
```

La mayoría de los comandos capaces de aceptar un nombre de fichero como argumento son capaces de aceptar varios. Por ejemplo:

```
unix$ touch a b c
unix$ ls
a    b    bin    c    lib    otro    tmp
unix$ rm a b c
unix$ ls
bin    lib    otro    tmp
```

Además, hay comandos como *ls* que utilizan un valor por omisión cuando no reciben un nombre de fichero como argumento. Por ejemplo, *ls* lista el directorio en el que estás si no indicas qué ficheros quieres listar, como has visto antes.

Es importante escribir los nombres de fichero exactamente. Casi todos los sistemas UNIX son sensibles a la capitalización (son *case sensitive*). Por ejemplo:

```
unix$ touch Uno
unix$ rm uno
rm: uno: No such file or directory
```

El comando *rm* no encuentra el fichero uno. Pero...

```
unix$ rm Uno
```

funciona perfectamente.

Otro comando útil es *mv*. Se utiliza para mover (y renombrar) ficheros. Por ejemplo, podemos utilizar

```
unix$ mv fich otro
```

en lugar de

```
unix$ cp fich otro ; rm fich
```

Antes de seguir utilizando *cp*, *mv* y otros comandos relacionados con fichero necesitamos ver qué son los directorios y como se utilizan.