Sistemas Operativos
Introducción al bash

Febrero, 2015

GUIÓN SOBRE BOURNE SHELL (bash) - Primera parte

Primeros pasos

1) Hacer *login* con el usuario y contraseña asignados.

Vamos a trabajar con el *shell* más habitualmente usado en las distribuciones actuales de Linux: el Bourne *shell* (*bash*), para familiarizarnos con el uso de sus comandos y, de paso, con la estructura y configuración de un sistema operativo Linux.

Cuando un servidor Linux opera sin interfaz gráfico de usuario (o, incluso, sin pantalla), el intérprete de línea de comandos es la única forma en que el usuario puede interaccionar con el sistema operativo y gestionarlo. Todo administrador de sistemas debe, por tanto, estar bien familiarizado con el uso del *shell*, y ser capaz de usarlo con proficiencia y seguridad.

Cuando un servidor Linux tiene corriendo un interfaz gráfico de usuario (es decir, presenta un escritorio), se accede al intérprete de comandos del shell ejecutando una aplicación gráfica (existe un buen número de aplicaciones terminales*: xterm, xterminal, gnome-terminal, konsole*, ..., siendo una preferencia de la distribución) que emula el comportamiento de un terminal de caracteres (es decir, terminal de sólo texto).

2) Observamos el prompt, y la información que proporciona. Generalmente es algo como lo que sigue, chequéalo en tu caso:

practicas@debian:~$, tenemos:

- *practicas*: nombre de usuario
- *debian*: nombre del servidor
- *~*: Directorio actual de trabajo. En Linux, por convenio, el caracter '~' es una abreviatura que representa la ruta completa hasta el directorio raíz del usuario (generalmente /home/nombre_de_usuario)
- *$*: Careta de usuario no priviliegiado (no root).
  La careta del usuario root suele ser *#*.

IMPORTANTE: Linux es "case-sensitive" (es decir, diferencia entre mayúsculas y minúsculas). Por tanto, a lo largo del desarrollo de la práctica es imprescindible respetar estrictamente el uso de mayúsculas o minúsculas en los nombres de comandos o ficheros que se especifiquen en el guion.

3) Consultamos nuestro nombre de usuario, ejecutando **whoami**

- Esta consulta (que aquí no es más que una trivialidad) suele ser muy útil cuando un administrador de sistemas (root de la máquina) está adoptando la identidad de diversos usuarios no privilegiados (usando *su* -) para administrar sus cuentas. Si la configuración del prompt se ha modificado (para acortarla) y ya no muestra el username actual, el comando *whoami* permite al administrador asegurarse de cuál es la identidad de usuario a la que está suplantando en un momento dado.

4) Vamos a enterarnos de cuál es la ruta completa a nuestro directorio home.
Para ello ejecutamos **pwd**

- ***pwd*** = iniciales de "print working directory"
- La salida del comando es la representación de la ruta al directorio actual (similar a la obtenida en un Windows), con una concatenación de directorios describiendo el camino desde el directorio raíz del sistema (/) hasta el actual. Un ejemplo sería */home/practicas* Nótese que la orientación de la barra es exactamente la inversa a la de Windows.

5) Limpiamos el contenido de la ventana, ejecutando **clear** ó **reset**.

Documentacion en línea: uso de man, apropos, whatis y whereis

Un clásico del sistema operativo Unix, es el acceso a una documentación online sobre sus comandos y APIs. Esta documentación se accede a través del comando ***man*.**

6) Como ejemplo, podemos leer la documentación del comando ***ls*** para ver qué significa el flag *-d* que le hemos añadido. Ejecuta:

man ls
man man

Observar que *man* implementa internamente un mecanismo de paginación y búsquedas. Comprobar lo siguiente:

- La pulsación de la barra espaciadora mueve una página hacia adelante.
- La pulsación de ***b*** mueve una página hacia atrás.
- Se pueden hacer búsquedas en el texto con la barra **/** (la de la tecla 7). Así, escribir **/lista** busca en el texto, desde la página en que estemos en adelante, todas las apariciones de la cadena *lista*, y nos subraya las que aparecen en la página actualmente visible.
- Cada pulsación de ***n*** tras la búsqueda nos muestra la siguiente coincidencia encontrada de la cadena especificada (comprobarlo desplazándose, pulsando **n**, a las 6 primeras coincidencias).
- La pulsación de ***N*** hace lo mismo, pero desplazándose por las coincidencias a la inversa (es decir, de final hacia inicio). Comprobarlo volviendo hacia atrás en las coincidencias, pulsando 6 veces **N**.
- Pulsando **h** se accede a una página de ayuda del propio ***man***, con el listado completo de los muchos comandos de teclado disponibles.
- Se sale del programa (y de la página de ayuda) pulsando **q**. Comprobarlo.

El comando ***man*** se complementa con otros tres (***apropos***, ***whatis*** y ***whereis***), que permiten localizar nombres de comandos o funciones, y encontrar su ubicación en el sistema:

- ***whatis***: Busca entre los nombres de las páginas de manual uno que se parezca al que se ha pasado como argumento.
- ***apropos***: Busca entre las descripciones (contenido) de las páginas de manual la cadena especificada como argumento.
- ***whereis***: ubica el ejecutable y página *man* del comando pasado como argumento.

7) Para ver la diferencia entre ellos, buscaremos qué comandos ofrece el sistema operativo con el nombre ***time***. Ejecutar:

- **apropos time**
  Ver que la salida incluye TODO lo que contenga la cadena *time*.
- **whatis time**
  Observar que la salida se reduce ahora a cuatro entradas (un comando de sistema, una system call POSIX para programas en C, y una descripción genérica de los temporizadores de Linux.
- **whereis time**
  Usando la ruta de búsqueda por defecto para intentar localizarlo, devuelve la ubicación del fichero ejecutable con el comando de sistema ***time*** (que está tanto en */usr/bin/time* como en */usr/bin/X11/time*

La documentación de man está dividida en varios *libros* distintos, por lo que puede ser necesario, al invocar ***man***, usar el identificador numérico del libro deseado.

8) Ejecuar **whatis time**. Observar que se pueden obtener, para el nombre *time*, tres juegos distintos de documentación man:

- Libro 1 = ***time*** como comando del sistema.
- Libro 2 = Descripción de la función ***time*** de la API C POSIX (coincide con el libro *lposix*).
- Libro 7 = Descripción de los temporizadores del sistema.

9) Comprobar la diferencia entre estos juegos de documentación, ejecutando:

- **man time** (es equivalente a **man 1 time**).
- **man 2 time**
- **man 7 time**

Historial y autoterminación de comandos

Bourne *shell* guarda un registro histórico de los comandos ejecutados, que no sólo puede consultarse, sino que también puede emplearse desde el *shell* para ahorrarnos el teclear comandos ya usados recientemente. Este listado contiene no sólo los comandos usados en la sesión de *shell* actual, sino también los usados en sesiones previas, almacenados en una lista FIFO que, por defecto, tiene como profundidad los últimos 500 comandos ejecutados.

El intérprete de comandos ejecuta tanto comandos externos como comandos internos (*built-in commands*). Los comandos externos son programas compilados, cuyo ejecutable binario se encuentra en una ruta incluida en la variable $PATH ó el cual se ha expresado con la ruta completa, bien absoluta o relativa.

10) Ejecutar el comando **history**. La salida es el listado con el contenido actual del histórico de comandos.

11) Podemos reutilizar cualquiera de los comandos del histórico, navegando hasta él usando las teclas de cursor arriba y cursor abajo. Como ejemplo, navegar hacia arriba el histórico hasta que se muestre el comando **pwd**, y ejecutarlo pulsando Enter.

12) Podemos también invocar directamente cualquier comando de la lista, introduciendo en el *prompt* el signo **!** seguido del número (en el listado) del comando que se quiere ejecutar, o las primeras letras del comando (al menos las suficientes para que el *shell* pueda encontrar sin ambigüedad la línea). Como ejemplo, buscar en la salida del comando **history** el número correspondiente a la ejecución del comando ***clear*** (llamémoslo *número-de-clear*), y ejecutar **!numero-de-clear**.

Bash ofrece también el uso del tabulador como acelerador de teclado para autocompletar nombres de comandos o sus argumentos (nombres de ficheros, rutas, etc), siempre que la terminación no sea ambigua (es decir, si sólo hay una forma posible de completar el comando o argumento).

13) Como ejemplo, teclear la palabra **hist**, y pulsar TAB (el tabulador). El shell completará automáticamente la palabra **history**. Pulsar Enter para ejecutar el comando.

Una útil característica incorporada a esta funcionalidad de autoterminación es que, cuando existe ambigüedad, una segunda pulsación de TAB ofrece un listado de posibles terminaciones.

14) Como ejemplo, vamos a usar el comando **ls** para listar el contenido del directorio */etc/samba*, siguiendo estos pasos:

- Teclear **ls** y, SIN DEJAR NINGUN ESPACIO TRAS LA S, pulsar una primera vez TAB (no ocurrirá nada, porque la terminación es ambigua), y luego una segunda vez (aparecerá la lista de posibles terminaciones, que en este caso son todos los comandos que empiezan por ***ls***).
- Como el comando que queremos es, justamente, **ls**, proseguimos escribiendo tras la **s** un espacio y el carácter **/**, tras el que volvemos a pulsar dos veces TAB; el resultado es el listado de todos los subdirectorios que cuelgan directamente del raíz.
- Queremos el directorio ***/etc***, así que tras la barra escribimos una **e**, y volvemos a pulsar TAB. Ya no hay ambigüedad en la terminación, que se autocompleta hasta **/etc**.
- Escribimos ahora **sa** tras la barra, y pulsamos (dos veces) TAB. Se nos ofrecerán dos posibles terminaciones (*samba* y *sane.d*).
- Escribimos una **m** adicional, y volvemos a pulsar TAB. Ya no hay ambigüedad, por lo que el comando se autocompleta a **ls /etc/samba**, que es justamente el que queremos ejecutar. Pulsamos Enter, y obtenemos el resultado.

Cambio del directorio de trabajo

Como en todos los sistemas operativos, el directorio de trabajo (indicado en el *prompt*) es donde (si no se especifica una ruta completa) se buscarán los ficheros de usuario (datos y/o aplicaciones) y se escribirán los ficheros generados por las aplicaciones.

El sistema de ficheros Linux está constituido por un espacio de nombres:

- con estructura jerárquica en árbol
- cuyo directorio raíz es `/`
- símbolos especiales:
  - directorio actual `.`,
  - su padre `..`,
  - el *home* del usuario `~`
- rutas absolutas (desde el directorio raíz): `/home/practicas/.bashrc`, `/etc/passwd`
- rutas relativas (desde el directorio actual (pwd)): `../practicas/./../practicas/`
  (observa que `./fichero` y `fichero` son nombres de rutas equivalentes)

Para cambiar el directorio actual de trabajo se usa el comando ***cd***

15) Vamos a usar el comando **cd** (*change directory*) para mover el directorio actual a */bin*, que es un directorio de sistema que contiene comandos del sistema operativo. Para ello, ejecutamos **cd /bin (**observar cómo cambia el prompt). Para ver los contenidos del directorio, ejecutamos **ls**; en pantalla aparecerá un listado (en varias columnas) con todos los ficheros contenidos en este directorio.

16) Para volver rápidamente al directorio home del usuario basta con ejecutar **cd** (observar cómo el *prompt* cambia a `~`).

17) Cambiando de directorio y empleando rutas absolutas y relativas, juega con otras opciones del comando **ls**

```
cd
ls -l
cd ..
ls -al
cd /
ls -hl
ls -al etc
cd ~
ls -ltr
cd ~/./..
ls -lSr
cd ~/../../
```

Concatenación de comandos

En Bourne shell (bash) existen diferentes formas de concatenar o combinar comandos.

La más simple implica la ejecución secuencial de dos comandos usando **;** (punto y coma) como separador al invocarlos; el segundo comando comienza a ejecutarse una vez el primero ha terminado por completo su ejecución.

18) Como ejemplo, ejecuta las siguientes secuencias (el comando **sleep** se bloquea y despierta tantos segundos después como le indiquemos):

cd /etc; ls -al
date; sleep 10; date
date; sleep 2; echo "Han pasado 2 segundos" ; sleep 3; date

Esta invocaciones concatena en una sola llamada la ejecución, de una secuencia de comandos. Observa el resultado, fíjate en las fechas impresas y el tiempo transcurrido.

19) Las secuencias de comando se pueden agrupar con paréntesis **(...)**. En este caso los comandos se ejecutan en un subshell. Prueba por ejemplo:

(cd /etc; pwd;); pwd

Observa el resultado, ¿cuál es el *working directory* al final de la secuencia?

Uso de caracteres comodín

El *shell* implementa el uso de metacaracteres para poder referenciar múltiples nombres de fichero con una sola cadena de caracteres. Estos metacaracteres se conocen como *glob style wilcards*. Desde la línea de comandos del *shell*, los dos más útiles son:

***** - Sustituye a cualquier número de caracteres.
? - Sustituye a sólo un caracter

20) Como ejemplo vamos a usar ***ls*** para ver, en el directorio */etc*, qué ficheros y subdirectorios comienzan con la letra "s". Ejecutar **ls /etc/s* |more** .Observar que el comando ***ls*** está listando también los contenidos de los subdirectorios obtenidos; para que sólo nos liste el nombre del subdirectorio, y no sus contenidos, añadimos un *flag* al comando ***ls***, y ejecutamos **ls -d /etc/s* |more**

21) Ahora, veremos el uso de ***?*** con dos nombres de ficheros que difieran sólo en un carácter.
Ejecutar **ls -d /etc/ap?**
Observar cómo el listado de salida contiene tanto el subdirectorio *apm* como el *apt* del directorio */etc.*

Redirección a fichero de entrada y salida estándar

El *shell* permite redireccionar a fichero la salida de un programa (tanto la estándar como las notificaciones de error), o redireccionar la entrada estándar de un programa (para que, por ejemplo, lea datos desde un fichero en lugar de desde teclado).

Recuerda que por defecto, existen los siguientes *streams* de entrada/salida por cada proceso:

stdin = entrada estándar = el teclado

stdout = salida estándar = el terminal (pantalla)

stderr = salida estándar de error = el terminal (pantalla),

el mecanismo de redirección, permite cambiar estos flujos por defecto hacia/desde ficheros.

Se usan los siguientes operadores seguidos del nombre de un fichero para realizar la redirección:

< Redirección de *stdin*

> Redirección de stdout sobreescribiendo el contenido del fichero

>> Redirección de stdout añadiendo el contenido al fichero

2> Redirección de stderr sobreescribiendo el contenido del fichero

>& Redirección de stdout y stderr conjuntamente sobreescribiendo el contenido del fichero

22) Vamos a redireccionar a un fichero la salida del listado de contenidos de */etc*, para poder verlo con tranquilidad. Ejecuta:

- **cd** (para ir al home del usuario)
- **ls /etc > listado.txt**
  Este comando redirecciona la salida de ***ls*** al fichero *listado.txt*.
- **ls** , para comprobar que se ha creado el fichero *listado.txt*
- **more listado.txt**
  Este comando presenta paginado el contenido del fichero, pero ahora podemos también paginar hacia atrás pulsando ***b***. (*NOTA: en este último comando se puede evitar tener que teclear el nombre completo del fichero. Probar a escribir **more lis**, y luego pulsar TAB; el nombre se completará automáticamente*).

23) Para probar la redirección de la entrada estándar, usaremos el comando ***cat***. Este comando tiene comportamientos diferentes según sus argumentos. Realiza las pruebas que se proponen, experimentando con el significado de las redirecciones.

Sin argumentos: lee de la entrada estándar y escribe en la salida estándar (teclado → terminal) . Prueba a teclear **cat** sin argumento y teclea líneas a continuación.
Nota: Usa ^D para cortar la entrada estándar de teclado; Por defecto ^D=EOF.
Nota: También se puede usar ^C para interrumpir la ejecución de cualquier comando

Con un argumento (o varios), **cat** lee un fichero (o varios) y lo vuelca en la salida estándar:

```
echo "Adiós mundo cruel" > cruel.txt
cat cruel.txt
```

Prueba ahora:

```
echo "Hola mundo"
echo "Hola mundo" > mundo.txt
cat mundo.txt
```

cat < mundo.txt
cat cruel.txt
cat cruel.txt ≫ mundo.txt
cat mundo.txt
cat mundo.txt cruel.txt
cat mundo.txt cruel.txt > /dev/null

¿Qué hemos hecho en esta última línea?

No olvides que con un sólo carácter `>`, la redirección de salida sobrescribe el fichero destino, caso de que exista. Si lo que se desea es añadir la nueva salida al final del fichero, sin perder su contenido previo, es necesario usar un doble ángulo `>>`.

24) Vamos a comprobar la diferencia entre ambas redirecciones. Para ello, seguir la secuencia de pasos descrita a continuación:

- Ejecutar **ls /var > salida.txt**
- Comprobar el resultado haciendo **more salida.txt**
- Ejecutar ahora **ls /boot > salida.txt**
- Comprobar, haciendo **more salida.txt**, que el contenido previo del fichero ha sido sobreescrito.
- Ejecutar ahora **echo "==== Sobreescritura ====" >> salida.txt**, seguido por **ls /var >> salida.txt**
- Comprobar el resultado haciendo **more salida.txt** .Se puede observar cómo la salida de estos últimos dos comandos se ha ido añadiendo a la cola del fichero.

Finalmente, es frecuente que, a la hora de procesar trabajos en lotes, se quiera salvar a fichero no sólo la salida estándar de los comandos o aplicaciones, sino también las notificaciones de error producidas durante la ejecución (lo que se denomina *salida de error* o *stderr*). Esta redirección se indica usando `2>`. El siguiente ejemplo muestra la diferencia entre redireccionar la salida estándar o la salida de error *stderr*.

25) Ejecutar **ls /kk** .El sistema operativo nos indica con un error que el directorio no existe. Ahora, seguir estos pasos:

- Ejecutar **ls /kk > error.txt** .Vemos que vuelve a aparecer el mensaje de error.
- Hacer **more error.txt**. Como vemos, el fichero está vacío (es decir, la redirección de la salida estándar no ha copiado el mensaje de error al fichero).
- Ahora, ejecutar **ls /kk 2> error.txt** .Observar cómo ahora no aparecer en pantalla el mensaje de error.
- Haciendo **more error.txt** comprobamos que, esta vez, el mensaje de error ha ido a parar al fichero *error.txt*.

Pipes ó tuberías

Otra manera en que *bash* permite combinar comandos es mediante el uso de tuberías ó *pipes*. El concepto de *pipe* está muy relacionado con el de redirección.

Mediante la combinación de comandos con pipe, la salida estándar (*stdout*) del primer comando sirve como entrada al siguiente. El operador *pipe* es el carácter |.

En este caso los comandos se ejecutan simultáneamente (no en secuencia), y el software subyacente al *shell* se encarga internamente de redireccionar la salida estándar (*stdout*) del primer programa a la entrada estándar (*stdin*) del segundo programa.

26.I) Como ejemplo, usaremos un pipe para paginar la salida del comando ***ls*** de un directorio de forma recursiva y no perdernos la parte inicial del listado que será largo:

```
ls -Rl /etc/ |more
```

observa cómo ahora el sistema nos muestra una página de listado, y espera la pulsación de una tecla antes de mostrar la siguiente (se pueden usar los mismos comandos de teclado que en el comando *man*).

He aquí algunos ejemplos de pipes, consulta la documentación de los diferentes comandos (echo, wc, less, grep, sort, cut)

```
cat /etc/password | wc            # cuenta caracteres palabras y líneas
cat /etc/group | less             # pagina el contenido del fichero
cat /etc/password | grep home     # filtra aquellas lineas que contienen la cadena "home"
cat /etc/password | sort          # ordena alfabéticamente las líneas
echo "adiós mundo cruel" | wc | cut -b 1-7   # ¿qué hacen estos comandos unidos por |?
```

Observa que la acción de las 4 primeras líneas se podría haber conseguido sin usar pipes:

```
wc /etc/passwd
less /etc/group
grep home /etc/passwd
sort /etc/password
```

comandos como **cat, wc**, **less**, **grep**, **sort** leen de la entrada estándar cuando no se les suministra un nombre de fichero como argumento.

26.II) El comando **grep** es de especial utilidad ya que permite filtrar aquellas líneas de la entrada estándar que cumplan con una expresión regular.
Si se le proporciona un segundo argumento que es un nombre de un fichero, leerá de ese fichero en lugar de la *stdin*.
Analiza los siguientes ejemplos usando el fichero `/proc/cpuinfo` que contiene información sobre las CPUs de la máquina:

```
grep model /proc/cpuinfo                  #Filtra las líneas que contengan "model"
grep '^cpu' /proc/cpuinfo                 #Filtra las líneas que empiezan por cpu
cat /proc/cpuinfo | grep '^cpu' | grep core  # Filtra las líneas que empiezan por cpu
                                           # y contienen "core"
grep processor /proc/cpuinfo | wc -l      # ¿Cuántas CPUs tenemos?
```

Comandos para manejar el sistema de ficheros

Los comandos básicos para manejar el sistema de ficheros en Linux son:

- ***mkdir, rmdir***: Creación y destrucción de directorios.
- ***cp***: Copia de ficheros.
- ***mv***: Mueve (o renombra ficheros).
- ***rm***: Borra ficheros
- ***ln:*** Crea enlaces (*softlinks* ó *hardlink*)
- ***touch***: Actualiza la hora de actualización de un fichero o lo crea si no existe
- ***file:*** Informa del tipo de fichero
- ***stat:*** Informa de los atributos del fichero
- ***pwd:*** Imprime el directorio actual de trabajo

27) Ejecutar **cd** para ir al directorio home del usuario, y crear dos directorios, *dir1* y *dir2* y dos ficheros usando los comandos siguientes:

cd
mkdir dir1
mkdir dir2
touch vacio.txt
echo "Hola mundo" > mundo.txt
mv mundo.txt cruel.txt

28) Copiar los dos ficheros de texto (`vacio.txt` *y* `mundo.txt`) que hay en el home del usuario al directorio `dir1`, comprobando el resultado con ls:

cp *.txt dir1
ls -l dir1.

29) Mover los dos ficheros de texto del home a `dir2`, **,** comprobar el resultado usando **ls:**

mv *.txt dir2
ls -l dir2
ls -l dir1
ls -l .

30) Borrar los ficheros de texto de *dir2*, uno a uno:

rm dir2/error.txt
rm dir2/salida.txt

Observar que por defecto NO hay aviso pidiendo confirmación del borrado (*NOTA: Teniendo en cuenta esto último, recordar siempre que en Linux un **rm *** o un **rm *.*** es MUY PELIGROSO*).

31) Borrar `dir2`. Observar que **rm dir2** falla, porque es un directorio; es necesario hacer **rm -r dir2** (que borra recursivamente) para conseguirlo.

Vamos a hacer más seguro el comando de borrado usando el comando ***alias***, que permite sustituir un comando por una cadena de texto.

32) Ejecutar

alias rm="rm -i"

Hay varias maneras de comprobar este alias:

alias
alias |grep rm
type rm

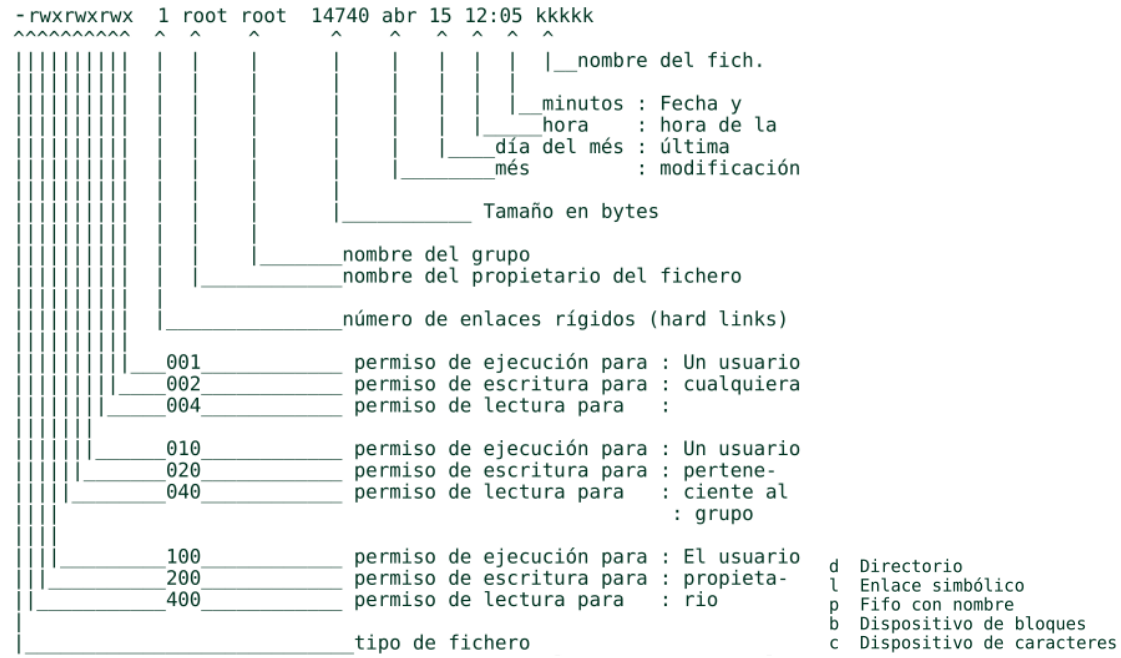
33) Ejecutar ahora

rm -r dir1

Vemos que ahora, al incluir el flag *-i*, cada borrado individual requiere una confirmación explícita.

Permisos del sistema de ficheros

Como todo sistema operativo multiusuario, Linux debe proporcionar un mecanismo de control de acceso a los ficheros, de tal forma que mis ficheros sólo sean visibles para aquellos usuarios a los que explícitamente se lo haya permitido.
Al listar un directorio con **ls -l** obtenemos una entrada extendida por cada fichero que nos informa de los permisos del fichero y otros atributos, según se describe en la siguiente figura:



Fuente: "Curso de Linux para novatos", A. Castro Snurmacher, 2008.

34) Ejecutar el comando **id**, que proporciona como salida nuestro identificador de usuario (numérico y nombre de usuario) y el identificador de grupo (numérico y nombre de grupo).
¿Cuál es el grupo principal?

35) Mira el contenido de los ficheros `/etc/passwd` y `/etc/group`, que es de donde se toma la información de usuario y grupo respectivamente. Interpreta su contenido.

36) Sitúate en tu directorio *home* y ejecuta la siguiente secuencia de comandos para crear los siguientes elementos, y lista los ficheros creados con todos sus detalles:

cd
touch a.file
echo hello > a.file2
mkdir a.dir
ln -sf a.file a.slink
ln a.file a.hlink
ls -l | grep 'a\.'

37) Interpreta detalladamente la cadena de permisos de cada uno de los ficheros `a.*` que hemos creado en el apartado interior (de la forma `-rwxr-xr-x`).
Utiliza **stat a.file** para ver los detalles completos de un fichero.

Observa que:

- El fichero `a.file` tiene permisos por defecto para un fichero no ejecutable; estos permisos sólo permiten que lo escriba (cambie o borre) el usuario, pero el fichero es legible por todo el mundo.
- Análogamente, el directorio `a.dir` sólo puede ser escrito (es decir, crear o borrar ficheros en él) por el usuario, pero todo el mundo puede acceder a él, y todo el mundo puede leer sus contenidos.
- Hay un número asociado a cada fichero que aparece antes del nombre de usuario al ejecutar el **ls** anterior, este número es *el número de nombres del fichero* ¿qué significa esto?
- El primer carácter de la cadena de permisos es `-` `d` ó `l`, que indica si es un fichero regular, un directorio o un *softlink*.

38) Ejecuta los siguientes cambios de permisos comprobando cuál es su efecto:

chmod ugo-wrx a.file
ls -l a.file
cat a.file
echo hola > a.file

chmod ug+rw a.file
ls -l a.file
cat a.file
echo hola > a.file

chmod -x a.dir
ls -l |grep a.dir
cd a.dir

39) Podemos cambiar los permisos usando ***chmod*** para , por ejemplo, crear un directorio privado del usuario, al que sólo él (y root) puedan acceder.
Crear un directorio haciendo **mkdir privado**. Ver sus permisos con **ls -l | grep privado.**

Restringir totalmente los permisos al grupo y a *others* haciendo **chmod 700 privado**, donde 700 es el patrón binario de permisos deseado (control total usuario, ningún permiso para los demás). Comprobar el resultado con **ls -l**. ¿Cuál sería el equivalente *chmod* con formato no numérico de permisos?

Ficheros ocultos

Si el nombre de un fichero comienza por '.', el sistema lo mantiene oculto en los listados, por defecto. Este tipo de ficheros habitualmente son ficheros de configuración del sistema o de aplicaciones.

40) Aprovechamos que estamos en el home del usuario y hacemos **ls -al** ; el flag *-a* muestra los ficheros ocultos.

41) Si hacemos **more .bash-history**, vemos que éste es el fichero donde el *shell* guarda el histórico de comandos. Ejecutamos **history -c** para eliminar el histórico, y **history -w** para escribir la historia actual al fichero; si ahora repetimos **more .bash-history**, vemos que el fichero sólo contiene el último comando.

42) Haciendo **more .bashrc** podemos ver que éste es un fichero de inicialización usado por *bash* al ser invocado. Hacemos ***/alias*** dentro de la paginación mostrada por el more para buscar la primera coincidencia de *alias*. Podemos ver, en las líneas que nos aparecen, cómo es posible ahí definir los aliases que queramos tener ya definidos siempre que abramos un bash (y el candidato más obvio a ello es el alias ***rm="rm -i"*** que hemos usado antes).

Soft y hard links

Un *softlink* es una característica ofrecida por el sistema de ficheros mediante la cual es posible crear una entidad, que se comporta como un fichero, pero que es en realidad tan sólo un puntero a otro fichero (y, por tanto, no sólo no duplica el espacio usado, sino que siempre mantiene su estado actualizado y coherente con el fichero original).

Un *softlink* es útil, por ejemplo, para crear aliases (de nombres o de rutas) a un directorio.

Por otro lado, en los sistemas de ficheros Unix, un fichero puede tener varios nombres, cada uno de estos nombres es un *hardlink* al fichero. El nombre de un fichero aun siendo una propiedad del fichero, siempre se almacena en un directorio. No se permiten *hardlinks* a directorios ya que los nombres permitidos para un directorio son sólo su nombre particular y los nombres especiales `.` (directorio actual) y `..` (directorio padre). Esto se hace para evitar bucles en el árbol de directorios. No obstante un fichero regular puede tener un número arbitrario de nombres ubicados en cualquier carpeta.

43) Vamos a crear con un *softlink* un nombre alternativo con el que acceder al directorio `privado` creado anteriormente. Para ello ejecutamos

ln -s privado privadolink

Fíjate que el orden de los argumentos es el mismo que el comando copiar fichero *cp* (*ln origen destino*)

44) Si ejecutamos **ls** vemos que *otro nombre* aparece simplemente como otro fichero más. En cambio, si hacemos **ls -l** ,obtenemos información más detallada que explícitamente nos informa de que *privadolink* no es más que un *softlink* a *privado*. Si ejecutamos **ls privadolink** vemos que se ha comportado como si fuera un directorio normal.

45) Ahora movemos *salida.txt* a *privadolink* con el comando **mv salida.txt privadolink** ,y volvemos a ejecutar **ls privado**. Como vemos, el resultado refleja el cambio en el contenido de *privado*.

46) Anteriormente se creó el *hardlink* `a.hlink (paso 36),` ¿cuántos nombres tiene este fichero? Múevelo a privado con **mv a.hlink privado** ¿cuántos nombres tiene ahora? y si borramos **a.file**, ¿se ha borrado el fichero? ¿cuántos nombres tiene ahora?

Variables de entorno

La forma más corriente de crear una variable de entorno es ejecutar a través del operador `=` (¡ojo! sin espacios ni antes ni después del símbolo "="):

variable=valor

export variable=valor

en el segundo caso la variable estará accesible por cualquier *shell* (ó proceso) creado desde la sesión en marcha.

Hay diferentes maneras de listar las variables de entorno:

set

printenv

env

47) Leemos el valor de una variable con el operador `${...}`, los paréntesis son opcionales si no hay ambigüedad. Por ejemplo,

agua=H20

echo ${agua}

echo $agua.2015

echo $agua2015

Una variable especialmente importante es PATH, que contiene la ruta de búsqueda de ejecutables.

48) He aquí algunas formas distintas de consultar el valor de la variable de entorno PATH en nuestro shell:

echo $PATH

printenv |grep PATH

El valor de la variable PATH es muy importante porque un fichero ejecutable sólo será encontrado al invocarlo si:

- Está contenido en uno de los directorios incluidos en el PATH, o bien,
- si la invocación contiene explícitamente la ruta completa al fichero ejecutable, bien absoluta o bien relativa al directorio actual.

Como ejemplo de esto último, vamos a crear un ejecutable ficticio en el directorio *home* del usuario, y veremos las distintas formas de conseguir que sea localizado por el sistema al invocarlo.

49) Nos aseguramos, ejecutand **cd**, de estar en el directorio home del usuario, y creamos un fichero vacío haciendo **touch vacio.exe**. Comprobamos sus permisos haciendo **ls -l vacio.exe**

50) Vamos a darle permisos de ejecución y lectura para todo el mundo, así como de escritura al usuario. Hacemos **chmod 755 vacio.exe**, seguido por **ls -l vacio.exe** para comprobar el cambio.

51) Intentamos ejecutar el comando, invocando **vacio.exe**. Obtendremos un error de comando no encontrado.

52) Lo invocamos ahora dando una ruta relativa al directorio actual, invocando **./vacio.exe**. Ahora sí es encontrado (aunque, estando vacío, obviamente no hace nada).

53) Vamos a invocarlo con una ruta absoluta.
Ejecuta **pwd** para ver la ruta del directorio actual, y escribir manualmente este resultado antes del comando, en la forma **/salida/de/pwd/vacio.exe**. De nuevo el comando es encontrado (*NOTA: Se puede expandir la variable de entorno PWD para evitar escribir a mano la ruta; comprobarlo ejecutando **echo $PWD/vacio.exe***).

54) Ahora vamos a modificar la variable de entorno PATH para añadir a la ruta el directorio de trabajo actual (ruta relativa = "."). Para ello, ejecutar:

```
export PATH=$PATH:.  #Añadir . al path
echo $PATH
vacio.exe
```

- Comprueba que, al ejecutar sin ninguna ruta, el comando es encontrado puesto que el directorio actual . está incluido en la variable PATH. ¿También hubiera valido la redefinición **PATH=$PATH:~** ?
- Meter la ruta "." en la variable de entorno PATH no suele ser buena idea, ¿imaginas porqué?

55) Imprime el valor de otras variables predefinidas interesantes:

```
echo $PWD
echo $HOME
echo $SHELL
echo $RANDOM
```

Uso de las comillas

En *Bourne shell* encontramos tres tipos de comillas:

- `'` comillas simples (apóstrofe) sustituye literalmente lo entrecomillado sin expansión alguna
- `"` comillas dobles sustituye lo entrecomillado expandiendo las variables encerradas
- `` ` `` comillas simples invertidas (acento grave) sustituye lo entrecomillado por la salida estándar de su ejecución, expandiendo variables. Una forma alternativa a `` `...` `` es `$(...)`

El carácter de escape `\` se puede emplear para incluir la comilla de ese tipo entre el texto entrecomillado, por ejemplo `"\""`.

56) Ejecuta los siguientes ejemplos, comprendiendo la expansión de las comillas:

```
export A=H20
echo 'Variable $A'
echo "Variable $A"
echo "Variable \"\$A\" ${A}"
echo `echo "echo"`
#Observa el fichero destino de la redirección
echo `date '+%D %T'`
cal > `whoami`_`date +%Y`
ls -tr | grep `whoami`

#Este ejemplo crea un fichero y  lo ejecuta
echo "echo `date '+%D %T'`" > fich_echo1
more fich_echo1
chmod u+x fich_echo1
./fich_echo1

#Este ejemplo crea otro fichero y lo ejecuta
echo "echo \`date '+%D %T'\`" > fich_echo2
more fich_echo2
chmod u+x fich_echo2
./fich_echo2
```

Control de trabajos en el shell

A partir de ahora a los comandos o grupos de comandos lanzados desde el shell les denominaremos trabajos o tareas (*jobs*).
Hasta el momento los comandos lanzados en nuestro *shell* de Linux han constituido tareas en *foreground* (primer plano), es decir, el *shell* no toma el control del terminal hasta que la tarea finaliza.

Otro modo de ejecución de una tarea es en *background* (segundo plano). En este modo la tarea pierde el control del terminal (se convierte en una tarea independiente) por lo que el *shell* continúa aceptando comandos independientemente de la tarea lanzada en segundo plano.
También una tarea puede estar suspendida (*stopped*), que es un estado en el que pierde el control del terminal y deja de ejecutarse en la CPU. Una tarea suspendida, puede reanudarse y volver a planificarse en la CPU. En este caso si toma el control del terminal, la tarea suspendida se reanuda en *foreground*, por el contrario si no toma el control se reanuda en modo *background*.

Estos son los comandos y acciones que permiten conmutar de modo una tarea o trabajo:

- `&` Un ampersand al final de un comando lanza el comando en background (si no se pone nada, como los ejemplos vistos hasta ahora, por defecto es en foreground)
- `^Z` Al pulsar simultáneamente control + z, el trabajo en foreground actual pasa a suspendido
- **bg** Es un comando interno que hace que el último trabajo suspendido pase a background
- **fg** Es un comando interno que hace que el último trabajo suspendido pase a foreground, el shell pierde pues el control del terminal
- **jobs** Lista los trabajos actuales de este *shell*

Nota: bg y fg pueden tener como argumento un número de trabajo de la lista de trabajos. En este caso la acción *background* o *foreground* se aplica a dicho trabajo. La sintaxis es **fg job_number** en este caso.

57) Es muy frecuente usar el modo background para los procesos con interfaz gráfico. Prueba lo siguiente:

```
xclock &
xclock -bg yellow -update 1 &
xcalc &
xterm &
jobs
# en este punto cierra algunas de las ventanas abiertas (reloj, calculadora ...)
# y ejecuta de nuevo jobs
jobs
```

58) Juega ahora con los conceptos de primer y segundo plano. Observa detenidamente qué ocurre tanto en la aplicación como en el shell:

```
xclock -update 1   #lanzar reloj con secundero en primer plano
^Z        #observa que al suspenderse el secundero del reloj se detiene
jobs
fg        # pasar el reloj a primer plano
^Z        # suspenderlo
jobs
bg        # pasarlo a segundo plano,
          # observa que el shell recupera el control del terminal
jobs
```

```
fg 1      # lo ponemos en primer plano (job nº 1 si solo está el reloj, usa otro número si hay
          más procesos en la lista)
^C        # cortar (aniquilar) el proceso en foreground
```

59) Experimenta con las siguientes secuencias de comando, observa que el operador background & constituye un separador de comandos tal como lo es el punto y coma ;

```
sleep 1 ; xclock & echo "Tras un segundo se lanzó el reloj"
(sleep 5 ; xclock ) &
sleep 10 & date ; sleep 5; date
(sleep 5 ; date ) & (sleep 1; date)
```

Monitorización y operaciones de procesos

60) El comando **ps** nos lista los procesos existentes en el sistema. Dada su importancia, este comando admite un gran número de opciones, que nos permiten ver todos los atributos de los procesos. En UNIX, cada proceso tiene asignado un identificador único, su PID. También se almacena el PID de su padre, que se abrevia con PPID.
No se deben confundir los procesos con los *jobs*. Los *jobs* son trabajos asociados a un terminal (*shell*), desde donde se lanzaron, pudiendo estar en primer o segundo plano. Por otro lado un *job* puede estar constituido por un grupo de procesos (por ejemplo conectados por un pipe).
Ejecuta las siguientes combinaciones, y analiza qué ocurre. Consulta el man de ps para ver qué significa cada una de las opciones

```
ps
ps uax                #Listando procesos todos los usuarios
ps -edfl              # Listado extendido
ps -edfl |less
ps -edfl |grep root   # Filtrando los procesos del usuario root
```

61) La variable de entorno especial $$ contiene el PID del propio shell. Imprime dicho PID y localiza el proceso en el listado ps:

```
echo $$  # Este es el PID de nuestro proceso shell
ps -edlf  | grep $$
```

62) Puedes ver todos los procesos en formato árbol genealógico con el comando **pstree**. Prueba la siguiente secuencia de acciones:

```
pstree
xclock &
pstree -p  $$  # Esto muestra sólo la parte del arbol que cuelga de un PID ¿quién cuelga de
               # nuestro shell en este momento?
```

63) Otro comando para monitorizar los procesos es **top**, con el podemos ver qué procesos están consumiendo más recursos en un momento dado. Una versión mejorada es **htop**. Prueba estos comandos.

64) El directorio /proc es un directorio "virtual" a través del cual el sistema operativo nos ofrece información de los procesos accesible de una manera textual. Observa que los directorios con nombre numeral representan los procesos. El número es el PID del proceso (por ejemplo el proceso con PID=123, tiene asociado /proc/123). Dentro de dicho directorio existen ficheros virtuales con información del proceso. Curiosea el contenido de dicho directorio:

```
ls -l /proc
ls -l /proc/$$    #Este directorio está asociado al proceso del shell
```

65) Lanza un **xclock** en *background*. Localiza su PID filtrando la salida de **ps -edf**. Localiza su entrada asociada en `/proc`.

66) En ocasiones es necesario aniquilar o eliminar un proceso del sistema. Para ello disponemos del comando **kill**. El comando lleva como argumento el PID del proceso a eliminar.
Veamos un ejemplo.

```
xeyes &                  # Lanzar xeyes en segundo plano
ps -edfl  | grep xeyes   # Localiza el proceso, mira su PID
                         # (si hay varios con el mismo nombre hay que elegir uno)
kill <pid>    # Sustituye <pid> por el PID del proceso xeyes
```

67) La acción kill, es algo más que aniquilar un proceso. En realidad lo que estamos haciendo es enviar una señal al proceso, consecuencia de la cual el proceso muere.
Ejecuta **kill -l** con ello listamos todas las señales disponibles que están numeradas.
Por defecto estamos enviando la señal SIGTERM (15) que provoca la terminación ordenada de un proceso. Otra señal de interés es la señal SIGKILL (9), que provoca una aniquilación forzada.
Observa que un usuario puede enviar señales solo a sus procesos:

```
ps -edfl |grep root     # Localiza un proceso del usuario root
kill -9 <pid>           # Sustituye <pid> por el PID de uno de los procesos de root
kill -KILL <pid>        # Esta sintaxis es equivalente a la acción anterior (SIGKILL=9)
```

Como ves no podemos forzar la muerte de un proceso que no pertenece a nuestro usuario.

Nota: El tema de las señales se abordará con más detalle más adelante.