Guía básica de la linea de comandos

Fuentes:

- http://culturacion.com/sistema-de-ficheros-en-linux/
- https://hipertextual.com/archivo/2014/04/comandos-basicos-terminal/

Atajos

Puesto que teclear comandos suele ser una tarea costosa hay una serie de atajos que hacen que esa tarea sea mucho más eficiente

- Las teclas de cursor (arriba y abajo) nos permiten consultar los comandos anteriores.
- El tabulador suguiere posbles continuaciones al comando que estamos escribiendo:

  ```
  $ ls /ho<Tab>
  ```

  se autocompleta a

  ```
  $ ls /home
  ```

- Si nos acordamos de un comando anterior, podemos intentar buscarlo con pulsando `ctrl+R` y luego pulsamos unos caracteres que estén en el comando. Por ejemplo, si hemos ejecutado el comando

  ```
  $ pdflatex -sycntex=1 main.tex
  ```

  Podemos recuperarlo pulsando `ctrl+R` y luego las teclas s, y, n, y c.

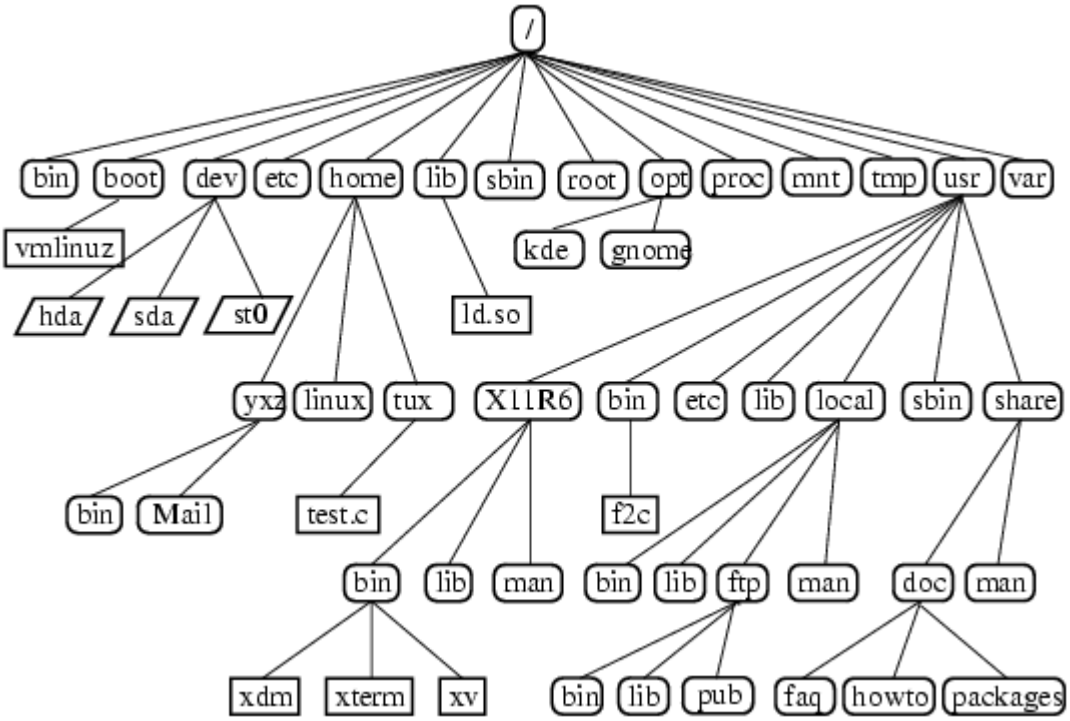
- Wildchars:
  - El carácter `*` se puede sustituir por cualquier sequencia de caracateres:

    ```
    $ ls -al *py
    ```

    muestra los ficheros que acaban en py.
  - El carácter ? Se puede sustituir por 1 y solo 1 carácter.

El sistema de ficheros

El sistema de ficheros de los sistemas Unix, y ello incluye tanto a Linux como MacOS) se estructura en forma de árbol.



- Existe un solo directorio en lo más alto de la estructura a partir del cual nace todo el resto. A este directorio se le denomina (por la analogía con el árbol) directorio raíz. En Unix se representa con el símbolo /.
- Todo directorio tiene un solo directorio padre. Al directorio padre se le denomina siempre `..` (dos puntos seguidos). De esta manera con la orden `cd ..` (que pronto veremos) iremos siempre al directorio padre del actual.
- Del mismo modo que existe la entrada `..` , existe una entrada llamada `.` (un solo punto). Representa al propio directorio. De esta manera, la orden cd . no tendría el menor efecto puesto que nos llevaría al propio directorio donde ya nos encontramos. Esta entrada es útil en muchas ocasiones como veremos más adelante.

Rutas

Absolutas: indica la ruta total del fichero desde la raíz del sistema de ficheros, empieza por `/`:
`/home/luis/programs/pr1.py`

Relativas: indica la ruta desde el directorio actual de trabajo:

`programs/pr.py` (en el directorio actual hay un subdirectorio llamado programs)

`../data/data1.csv` (Hay un directorio hermano llamado data)

Comandos útiles

Para practicar los comando nos descargamos un fichero de pruebas:

`$ wget` http://dana.estad.ucm.es/gead17/inf-mat_17-18.tgz

Lo descomprimimos:

```
$ tar xvf inf-mat_17-18.tgz
```

cd

Cd (de *change directory* o cambiar directorio), es como su nombre lo indica el comando que necesitarás para acceder a una ruta distinta de la que te encuentras. Por ejemplo, si estas en el directorio /home y deseas acceder a /home/ejercicios, seria:

```
$ cd /home/ejercicios
```

Si estás en /home/ejercicios y deseas subir un nivel (es decir ir al directorio /home), ejecutas:

```
$ cd ..
```

ls

Ls (de listar), permite listar el contenido de un directorio o fichero. La sintaxis es:

```
$ ls /home/directorio
```

El comando ls tiene varias opciones que permiten organizar la salida, lo que resulta particularmente útil cuando es muy grande. Por ejemplo, puedes usar *-a* para mostrar los archivos ocultos y *-l* para mostrar los usuarios, permisos y la fecha de los archivos. Así como para todos los comandos Linux, estas opciones pueden combinarse, terminando en algo como:

```
$ ls -la /home/directorio
```

cat

Cat (de concatenar), es una maravillosa utilidad que nos permite visualizar el contenido de un archivo de texto sin la necesidad de un editor. Para utilizarlo solo debemos mencionarlo junto al archivo que deseamos visualizar:

```
$ cat prueba.txt
```

touch

Touch crea un archivo vacío, si el archivo existe actualiza la hora de modificación. Para crear el archivo prueba1.txt en /home, seria:

```
$ touch /home/prueba1.txt
```

mkdir

Mkdir (de *make directory* o crear directorio), crea un directorio nuevo tomando en cuenta la ubicación actual. Por ejemplo, si estas en /home y deseas crear el directorio ejercicios, sería:

```
$ mkdir /home/ejercicios
```

Mkdir tiene una opción bastante útil que permite crear un árbol de directorios completo que no existe. Para eso usamos la opción *-p*:

```
$ mkdir -p /home/ejercicios/prueba/uno/dos/tres
```

cp

Cp (de *copy* o copiar), copia un archivo o directorio origen a un archivo o directorio destino. Por ejemplo, para copiar el archivo prueba.txt ubicado en /home a un directorio de respaldo, podemos usar:

```
$ cp /home/prueba.txt /home/respaldo/prueba.txt
```

En la sintaxis siempre se especifica primero el origen y luego el destino. Si indicamos un nombre de destino diferente, cp copiará el archivo o directorio con el nuevo nombre.

El comando también cuenta con la opción *-r* que copia no sólo el directorio especificado sino todos sus directorios internos de forma recursiva. Suponiendo que deseamos hacer una copia del directorio /home/ejercicios que a su vez tiene las carpetas ejercicio1 y ejercicio2 en su interior, en lugar de ejecutar un comando para cada carpeta, ejecutamos:

```
$ cp -r /home/ejercicios /home/respaldos/
```

Si el tamaño de los datos a copiar es grande, es mejor usar el comando `rsync`.

```
$ rsync -va /home/ejercicios /home/respaldos/ejercicios
```

mv

Mv (de *move* o mover), mueve un archivo a una ruta específica, y a diferencia de *cp*, lo elimina del origen finalizada la operación. Por ejemplo:

```
$ mv /home/prueba.txt /home/respaldos/prueba2.txt
```

Al igual que *cp*, en la sintaxis se especifica primero el origen y luego el destino. Si indicamos un nombre de destino diferente, mv moverá el archivo o directorio con el nuevo nombre.

rm

Rm (de *remove* o remover), es el comando necesario para borrar un archivo o directorio. Para borrar el archivo prueba.txt ubicado en /home, ejecutamos:

```
$ rm /home/prueba.txt
```

Este comando también presenta varias opciones. La opción *-r* borra todos los archivos y directorios de forma recursiva. Por otra parte, *-f* borra todo sin pedir confirmación. Estas opciones pueden combinarse causando un borrado recursivo y sin confirmación del directorio que se especifique. Para realizar esto en el directorio respaldos ubicado en el /home, usamos:

```
$ rm -fr /home/respaldos
```

Este comando es muy peligroso, por lo tanto es importante que nos documentemos bien acerca de los efectos de estas opciones en nuestro sistema para así evitar consecuencias nefastas.

pwd

Pwd (de *print working directory* o imprimir directorio de trabajo), es un conveniente comando que imprime nuestra ruta o ubicación al momento de ejecutarlo, así evitamos perdernos si estamos trabajando con múltiples directorios y carpetas. Su sintaxis seria:

```
$ pwd
```

du

Se usa para saber cuanto ocupan los ficheros o directorios

```
$ du -sh /var
```

Muestra lo que ocupa el directorio var (-s no muestra los subdirectorios y -h indica el tamaño para ‘humanos’)

clear

Clear (de limpiar), es un sencillo comando que limpiara nuestra terminal por completo dejándola como recién abierta. Para ello ejecutamos:

```
$ clear
```

Como *bonus* les recomiendo utilizar *man* que muestra una documentación completa de todos los comandos. Para *clear*, por ejemplo:

```
$ man clear
```

chmod

Chmod (del inglés *change mode*) es un comando que permite cambiar los permisos de acceso de un directorio o archivo. Su sintaxis es:

```
$ chmod [opciones] <modo> <archivo>
```

Donde *opciones* nos permite entre otras cosas, cambiar los permisos recursivamente para un directorio con *-R*, *modo* son los permisos de lectura, escritura y ejecución representados en notación octal que previamente explicamos y *archivo* es el nombre del directorio o archivo que queremos modificar.

Por ejemplo, para asignar permisos de lectura, escritura y ejecución para el dueño, el grupo y remover los permisos para el resto de los usuarios al archivo prueba.txt, sería:

```
$ chmod 770 prueba.txt
```

chown

Chown (del inglés *change owner*) nos permite cambiar el propietario de un archivo o directorio. Su sintaxis es:

```
$ chown [opciones] <nuevo-propietario> <archivo>
```

Donde *opciones* son las opciones del comando, como *-R* para cambiar recursivamente el propietario de un directorio y todo su contenido, *nuevo-propietario* será el nuevo propietario y *archivo* es el nombre del directorio o archivo que queremos modificar.

Por ejemplo, para cambiarle el propietario del directorio /home/ejercicios y todo su contenido y asignarlo al usuario *pedro*, hacemos:

```
$ chown -R pedro /home/ejercicios
```

useradd

Useradd (de agregar usuario) se utiliza para crear nuevos usuarios en tu sistema Linux. Su sintaxis es:

```
$ useradd [opciones] <nombre-usuario>
```

Donde *opciones* nos permite asignar un grupo al usuario con *-g*, asignar el directorio /home con *-d*, crearlo con *-m* si no existía previamente y *-s* para asignarle un intérprete de comandos o *shell*, entre otras.

Así, para crear el usuario *andrea* cuyo grupo principal sera editores, ejecutamos:

```
$ useradd -g editores -d /home/andrea -m -s /bin/bash andrea
```

usermod

Usermod (de modificar usuario) modifica algunos parámetros de un usuario existente, como el nombre, su directorio /home y los grupos a los que pertenece, entre otros. Su sintaxis es:

```
$ usermod [opciones] <nombre-usuario>
```

Donde *opciones* cambia el directorio home con *-d*, mueve todo el contenido del directorio anterior con *-m* y cambia el nombre de usuario con *-l*, entre otras. Para cambiar el nombre al usuario andrea por violeta, sería:

```
$ usermod -l violeta andrea
```

deluser

Deluser (del inglés *delete user*) es un sencillo comando para borrar usuarios. Tiene la opción *-r* que adicionalmente borra su directorio /home. Para borrar el usuario violeta con su /home, ejecutamos:

```
$ deluser -r violeta
```

passwd

Passwd (del inglés *password*) es una utilidad que se usa para cambiar o generar la contraseña de un usuario existente. Al invocarlo, pedirá la contraseña actual (si existe) y luego que la contraseña nueva sea introducida dos veces para verificar que fue escrita correctamente. Por ejemplo para asignar una contraseña al usuario violeta, sería:

```
$ passwd violeta
```

whoami

Whoami (del inglés *Who Am I* o Quien Soy Yo en español) muestra el identificador del usuario actual. Para ejecutarlo solo basta con invocarlo:

```
$ whoami
```

uptime

Uptime muestra el tiempo que el ordenador ha pasado encendido sin ser reiniciado, así como el *load average* o carga promedio del sistema que es el número de trabajos que se han realizado en los últimos 1, 5 y 15 minutos. Para ver su salida, solo escribimos en la terminal:

```
$ uptime
```

uname

Uname es un programa de sistemas operativos de tipo Unix que imprime detalles de la máquina y del sistema operativo que se está ejecutando. Su salida es diferente dependiendo de las opciones, por ejemplo, uname solo muestra el nombre del sistema operativo pero cuando le pasamos la opción *-r* muestra la versión del kernel y con *-a* de *all*, su salida es mucho mas completa. Se ejecuta de la siguiente forma:

```
$ uname -a
```

En mi caso, su salida es:

```
$ Linux adamantium 3.14.4-1-ARCH #1 SMP PREEMPT Tue May 13 16:41:39 CEST 2014
x86_64 GNU/Linux
```

ps

El comando ps muestra los procesos que actualmente se están ejecutando en el sistema. Proporciona información útil como el número de proceso, el usuario y el usuario que está ejectando el proces. La forma más habitual de usarlo es

```
$ ps -aux
```

Si hay muchos procesos podemos paginarlos con en comando less

```
$ ps -aux | less
```

kill

Kill es un comando utilizado para enviar mensajes sencillos a los procesos en segundo plano ejecutándose en el sistema. Por defecto el mensaje que se envía es la señal de terminación. Su sintaxis más sencilla es:

```
$ kill [-s] <pid>
```

Donde *-s* es la señal a enviar, de no ser especificada ninguna se manda la señal por defecto y *pid* es el identificador del proceso. Otra de sus opciones es *-9* que fuerza la terminación de un proceso.

Por ejemplo, para terminar un proceso cuyo id es 3477, ejecutamos:

```
$ kill 3477
```

Recuerden utilizar *man* para obtener una documentación completa de cada comando que quieran probar. ¿Qué comandos te gustaría aprender a usar?

file

File determina el tipo de un archivo y te imprime en pantalla el resultado. No hace falta que el archivo tenga una extensión para que File determine su tipo, pues la aplicación ejecuta una serie de pruebas sobre el mismo para tratar de clasificarlo.

Ejecutarlo sería tan sencillo como:

```
$ file un_archivo_de_texto.txt
```

ln

Ln es una aplicación que permite crear enlaces a los archivos, tanto físicos (*hard links*) como simbólicos (*soft links*). En pocas palabras, un enlace simbólico es como un *acceso directo* en Windows o un *alias* en OSX mientras que un enlace físico es un nombre diferente para la misma información en disco.

Para crear un enlace físico ejecutamos:

```
$ ln archivo_origen nombre_enlace
```

Y para crear un enlace simbólico:

```
$ ln -s archivo_origen nombre_enlace
```

cmp

Cmp compara el contenido de dos archivos y devuelve 0 si los archivos son idénticos ó 1 si los archivos tienen diferencias. En caso de error devuelve -1.

Para ejecutarlo basta con:

```
$ cmp -s archivo1 archivo2
```

Cmp también puede mostrar algo de información sobre las diferencias pero para un reporte más detallado tenemos el siguiente comando.

diff

Diff, al igual que cmp, compara el contenido de dos archivos pero en lugar de devolver un valor imprime en pantalla un resumen detallado línea a línea de las diferencias. Ejecutarlo es tan simple como:

```
$ diff archivo1.txt archivo2.txt
```

Diff también puede usarse con directorios. En este caso comparará los nombres de los archivos correspondientes en cada directorio por orden alfabético e imprimirá en pantalla los archivos que estén en un directorio pero no estén en el otro.

wc

Wc imprime en pantalla la cantidad de saltos de línea, palabras y bytes totales que contenga un archivo. Para usarlo con un archivo cualquiera ejecutamos:

```
$ wc archivo.txt
```

sort

Sort imprime en pantalla las líneas de un archivo ordenadas alfabéticamente. Para ejecutarlo basta con:

```
$ sort archivo.txt
```

tail

Tail muestra en pantalla las últimas líneas de un archivo.

```
$ tail archivo.txt
```

Por defecto siempre muestra 10 pero podemos indicarle un número diferente de líneas a visualizar usando el parámetro -n:

```
$ tail -50 archivo.txt
```

head

Head es el comando opuesto a tail, muestra las primeras líneas de un archivo.

```
$ head archivo.txt
```

Al igual que tail, muestra por defecto las 10 primeras líneas pero podemos indicarle un número diferente usando el parámetro -n:

```
$ head -10 archivo.txt
```

more

More es un filtro que permite paginar el contenido de un archivo para que se vea a razón de una pantalla a la vez. Era muy usado en las viejas terminales cuya resolución era de 80×25 para visualizar archivos muy grandes. Para utilizarlo simplemente ejecutamos:

```
$ more archivo.txt
```

More permite navegar a través del contenido del archivo usando las flechas direccionales *arriba* y *abajo*, *Espacio* o la tecla *Enter*. Para salir de more usamos la tecla *Q*

less

Aunque su nombre es lo opuesto de **more** es realmente una versión mejorada de éste último. Less es otro filtro que permite paginar el contenido de un archivo pero que además de permitir la navegación hacia adelante y hacia atrás, está optimizado para trabajar con archivos muy grandes.

Ejecutarlo es tan simple como escribir:

```
$ less archivo.txt
```

Less permite navegar a través del contenido del archivo usando las flechas direccionales *arriba* y *abajo*, *Espacio* o la tecla *Enter*. Para salir de less también usamos la tecla *Q*.

Recuerda que para obtener más información sobre los parámetros y la sintaxis de los comandos puedes usar la aplicación **man** desde la terminal. Por ejemplo:

```
$ man less
```

grep

El comando grep filtra el contenido de un fichero. Se suele usar en pipeline con otros comandos para focalizar la busqueda. Por ejemplo el comando

```
$ ps -aux | grep ^luis
```

Muestra los procesos que ha arrancado el usuario `luis`.

find

Este comando busca ficheros que cumplen ciertos requisitos. El comando

```
$ find /var -name “*ps”
```

busca ficheros acabados en `ps` dento del directorio `/var`.

Pipelines (tuberías)

La entrada estándar (que se suele identificar con lo que se introduce por teclado) y la salida estándar de los comando se suelen poder conectar. Por ejemplo, si queremos ver cuáles son los ficheros/directorios que más ocupan en un sitio

```
$ du -s /var/* | sort -n
```

Si queremos mostrar los detalles de los ficheros acabados en py

```
$ find . -name “*py” -print0 | xargs -0 ls -l {}
```