Capítulo 4: Comandos GNU/Linux

Introducción

En el capítulo 3 presentamos el uso de los comandos `pwd`, `cd`, y `ls`. Este último, en particular, posee numerosas opciones que nos permiten modificar la visualización de los datos que devuelve, o ver más detalles sobre los mismos. En este capítulo hablaremos más sobre esta utilidad y continuaremos nuestro aprendizaje agregando más herramientas que podamos emplear desde la línea de comandos de Linux.

El comando ls

Este comando nos permite distinguir los siguientes tipos de contenidos en un directorio. También vale la pena mencionar que, si el argumento representa un archivo, enlace simbólico, dispositivo, etc., veremos datos sobre el mismo en la salida de `ls`.

- En *blanco* (en el caso de que el color de fondo de la terminal sea negro u otro color oscuro, que es lo más común), los archivos de texto o binarios no ejecutables.
- En *verde*, los archivos de texto (scripts) y otros binarios ejecutables.
- En *celeste*, los enlaces simbólicos.
- En *fondo negro con letras rojas*, los enlaces simbólicos *rotos*. Nos referimos a aquellos que apunten a un recurso del sistema que no existe.
- En *amarillo*, los dispositivos de bloques.
- En *violeta*, los archivos de imágenes o archivos temporales.
- En *rojo*, los archivos comprimidos.

En la imagen siguiente podemos distinguir algunos de los casos que nombramos arriba. En el primero, vemos que el archivo en cuestión se muestra en texto negro debido a que el color de fondo de la terminal es un tono claro.

```
alumno@stretch:~$ ls
adm.txt                ejemplos   for.tar        pro
alumnosconectados.txt  for-1.txt  link_archivo1  pru
archivo2               for-2.txt  link_archivo2  rai
archivo3               for-3.txt  mbox           red
clase1operador.txt     for-4.txt  nohup.out      res
datostutor.txt         for-5.txt  op.txt         RMD
alumno@stretch:~$
```

En este caso, **adm.txt** y **datostutor.txt** son ejemplos de archivos. Por otro lado, **archivo3** (contrario a lo que podría sugerir su nombre) se trata de un directorio, **link_archivo2** es un enlace simbólico activo, **link_archivo1** uno roto, y **archivo2** es un archivo binario ejecutable.

Mostrar archivos ocultos

En Linux podemos ver que tanto los archivos como los directorios pueden tener varios puntos en su nombre, pero cuando el punto está adelante los transforma en *ocultos*. Por ejemplo, **.hola** no se verá con un `ls` normal. Sin embargo, al utilizar el modificador `-a`, podremos ver todos los archivos ocultos que se encuentran en ese directorio.

Generalmente los archivos ocultos son los que contienen información acerca de configuraciones. Al no ser visibles fácilmente es menos probable que sean borrados de forma accidental.

Para ilustrar, veamos la diferencia que existe entre hacer `ls` y `ls -a` en el directorio personal de un usuario.

```
alumno@stretch:~$ ls
adm.txt                 ejemplos    for.tar         proc
alumnosconectados.txt   for-1.txt   link_archivo1   prue
archivo2                for-2.txt   link_archivo2   raid
archivo3                for-3.txt   mbox            rede
clase1operador.txt      for-4.txt   nohup.out       resu
datostutor.txt          for-5.txt   op.txt          RMD
alumno@stretch:~$ ls -a
.                       .config          link_archivo2
..                      datostutor.txt   mbox
adm.txt                 ejemplos         .nano
alumnosconectados.txt   for-1.txt        nohup.out
archivo2                for-2.txt        op.txt
archivo3                for-3.txt        procesos.txt
.bash_aliases           for-4.txt        .profile
.bash_history           for-5.txt        pruebas
.bash_logout            for.tar          raid.txt
.bashrc                 .lesshst         redes1.txt
clase1operador.txt      link_archivo1    resultado
alumno@stretch:~$
```

Información detallada sobre archivos y directorios

Al listar con el modificador `-l` obtenemos mayor información de los archivos y directorios. La salida del comando en este caso está dispuesta en columnas para que nos resulte más sencillo reconocerla. Veamos cómo queda cada columna:

1. El primer caracter que aparece en la primera columna indica si lo que estamos viendo es un archivo (`-`), un directorio (`d`), un enlace simbólico (`l`), o un dispositivo de bloques (`b`), por nombrar algunos ejemplos. A continuación, vemos los permisos que poseen sobre el mismo **a)** el dueño del archivo, **b)** el grupo dueño, y **c)** el resto de los usuarios del sistema.

2. Esta columna nos dice si este objeto posee enlaces que lo estén apuntando.

3. Usuario dueño del objeto.

4. Grupo dueño del objeto.

5. Tamaño en bytes del objeto. Para mostrar este dato en una unidad más *amigable* (KB, MB, etc.) podemos agregar la opción `-h`.

6. Fecha de última modificación del objeto.

Por ejemplo, el comando

```
ls -lh adm.txt
```

nos indica lo siguiente:

```
alumno@stretch:~$ ls -lh adm.txt
-rw-r--r-- 1 alumno finanzas 3,9K abr  2 19:03 adm.txt
alumno@stretch:~$
```

1. Se trata de un archivo propiamente dicho, ya que el primer caracter de la secuencia `-rw-r--r--` es `-`.

2. El archivo tiene **un (1)** enlace que apunta hacia el mismo.

3. El dueño del archivo es el usuario **alumno**.

4. El grupo dueño es **finanzas**.

5. El tamaño del archivo es **3,9 KB**.

6. La última modificación fue realizada el día **2 de abril del corriente año a las 19:03**.

Si dentro de un directorio existen subdirectorios y nos interesa ver el contenido de todos simultáneamente, se puede utilizar la opción `-R` para mostrar de forma recursiva tal información. Dicho de otra forma, `ls -lR pruebas` nos devolverá un listado detallado de tal directorio y de todos los subdirectorios que se encuentren dentro del mismo. En este caso solamente existe un subdirectorio, llamado **resultado**:

```
alumno@stretch:~$ ls -lR pruebas
pruebas:
total 620
-rw-r--r-- 1 alumno alumno   5864 ene 23 22:00 archivoprue
-rw-r--r-- 1 alumno alumno    493 ene 27 09:19 archivo.tar
-rw-r--r-- 1 alumno alumno 508929 ene 27 09:17 archivo.tar
-rw-r--r-- 1 alumno alumno  76472 ene 27 09:21 archivo.tar
-rw-r--r-- 1 alumno alumno   1998 ene  8 22:37 clasessh.tx
-rw-r--r-- 1 alumno alumno     32 ene 23 22:31 ejemplo.txt
-rw-r--r-- 1 alumno alumno    101 ene 23 21:16 file1.txt
-rw-r--r-- 1 alumno alumno    101 ene 23 21:16 file2.txt
-rw-r--r-- 1 alumno alumno     11 ene  8 22:32 holamundo.t
-rw-r--r-- 1 alumno alumno   1719 ene 23 22:51 martinfierr
-rw-r--r-- 1 alumno alumno    472 ene 23 22:28 prueba.txt
drwxr-xr-x 2 alumno alumno   4096 nov 22 21:18 resultado

pruebas/resultado:
total 8
-rw-r--r-- 1 alumno alumno 93 nov 22 21:03 adm.txt
-rw-r--r-- 1 alumno alumno 80 nov 22 20:57 op.txt
alumno@stretch:~$
```

El comando du (disk usage)

El comando `du` nos dice qué espacio ocupan uno o varios archivos en el disco y alternativamente cuál es el total del directorio (con la opción `-c`). Al igual que sucede con `ls`, podemos usar la opción `-h` para que nos muestre la unidad de medida que está usando.

Si nos interesa conocer el espacio que ocupan **todos** los archivos de un directorio, podemos utilizar el *comodín* *.

```
alumno@stretch:~$ du -sch pruebas
632K    pruebas
632K    total
alumno@stretch:~$ du -sh pruebas
632K    pruebas
alumno@stretch:~$ du -sch pruebas/*
8,0K    pruebas/archivoprueba.txt
4,0K    pruebas/archivo.tar.bz2
500K    pruebas/archivo.tar.gz
76K     pruebas/archivo.tar.xz
4,0K    pruebas/clasessh.txt
4,0K    pruebas/ejemplo.txt
4,0K    pruebas/file1.txt
4,0K    pruebas/file2.txt
4,0K    pruebas/holamundo.txt
4,0K    pruebas/martinfierro-cap1.txt
4,0K    pruebas/prueba.txt
12K     pruebas/resultado
628K    total
alumno@stretch:~$
```

El comando df (disk free)

Este comando nos permite ver cuál es el espacio libre que nos queda disponible por *file system* (particiones o volúmenes lógicos) y cuánto es el usado. Si a esto le agregamos el modificador `-h` lo podremos observar con unidades.

```
alumno@stretch:~$ df -h
S.ficheros                          Tamaño Usados  Disp Uso% Montado en
udev                                  235M      0  235M   0% /dev
tmpfs                                  50M   2,1M   48M   5% /run
/dev/mapper/debiancla--vg-root         52G   6,1G   44G  13% /
tmpfs                                 247M    12K  247M   1% /dev/shm
tmpfs                                 5,0M      0  5,0M   0% /run/lock
tmpfs                                 247M      0  247M   0% /sys/fs/cgroup
/dev/mapper/vg--curso-redes           1,1G   2,5M  1,1G   1% /curso/redes
/dev/mapper/vg--curso-administrador   921M   652M  208M  76% /curso/administrador
/dev/mapper/vg--curso-operador        926M   2,5M  879M   1% /curso/operador
/dev/mapper/debiancla--vg-home        5,1G   773M  4,1G  16% /home
/dev/mapper/debiancla--vg-tmp         248M   2,1M  229M   1% /tmp
/dev/mapper/debiancla--vg-var          51G   579M   48G   2% /var
/dev/sda1                             236M    52M  172M  24% /boot
tmpfs                                  50M      0   50M   0% /run/user/1000
alumno@stretch:~$
```

La imagen anterior nos muestra todos los sistemas de archivos disponibles en el sistema de prueba. Cada columna indica lo siguiente:

1. Sistema de archivos.
2. Tamaño total.
3. Espacio usado.
4. Espacio disponible.
5. Porcentaje de uso, redondeado al entero más cercano.
6. Punto o directorio de montaje. Se trata del lugar en el árbol de directorios con el que se encuentra asociado cada sistema de archivos.

El comando history

Este comando nos permite observar cuáles son los comandos que hemos usado. Para darnos esta información, lee lo que dice el archivo oculto **.bash_history** que se encuentra dentro del directorio **home** de los usuarios. Sin opciones, el comando `history` nos devuelve la lista de los últimos comandos ejecutados por el usuario en cuestión precedidos por un número que nos indica el orden respectivo.

La utilidad de esta herramienta reside en que podemos fácilmente volver a ejecutar un comando dado anteponiendo el signo **!** al número del comando que deseamos correr nuevamente.

Por ejemplo, para ejecutar el comando `bunzip2 archivo-bzip2.bz2` que vemos en la línea 384, podemos hacer

!384

```
alumno@stretch:~$ history
  382  ls -lh
  383  gunzip archivo-gzip.gz
  384  bunzip2 archivo-bzip2.bz2
  385  unxz archivo-xz.xz
  386  ls -l
  387  clear
  388  rm *
  389  ls -l
  390  echo "Hoy es la última clase del mó
  391  echo "En total, fueron 8 clases" >>
  392  gzip op.txt
  393  ls -l
  394  file op.txt.gz
  395  zcat op.txt.gz
  396  gunzip op.txt.gz
  397  clear
  398  ls
  399  bzip2 op.txt
  400  bzcat op.txt
```

El archivo **.bash_history** es muy importante porque contiene todos los comandos que el usuario fue ejecutando durante sus sesiones. Permanece oculto para que los usuarios no lo borren accidentalmente.

El comando mkdir (make directory)

Con este comando creamos nuevos directorios vacíos. En este ejemplo veremos cómo hacerlo al crear un nuevo directorio llamado **nuevodir**:

```
mkdir nuevodir
```

Si deseamos crear una estructura completa de directorios (un subdirectorio dentro de otro, y este último dentro de otro, y así sucesivamente) deberemos utilizar la opción `-p` de la siguiente manera:

```
mkdir -p dir1/dir2/dir3
```

El comando anterior creará **dir1** dentro del directorio actual. Si *miramos* dentro de **dir1** encontraremos a **dir2**, y dentro de este último a **dir3**:

```
alumno@stretch:~$ mkdir -p dir1/dir2/dir3
alumno@stretch:~$ ls -lR dir1
dir1:
total 4
drwxr-xr-x 3 alumno alumno 4096 abr  2 19:39 dir2

dir1/dir2:
total 4
drwxr-xr-x 2 alumno alumno 4096 abr  2 19:39 dir3

dir1/dir2/dir3:
total 0
alumno@stretch:~$
```

El comando rmdir (remove directory)

El comando `rmdir` permite borrar directorios, si los mismos están vacíos. También podemos utilizar la opción `-p` (de forma similar a como lo hicimos con `mkdir`) para eliminar una estructura de directorios que cumplan con la misma condición.

Si los directorios contienen archivos, tendremos que utilizar el comando `rm` para borrarlos previamente como veremos más adelante.

El comando cp (copy)

La herramienta `cp` se usa para copiar archivos. A continuación del comando propiamente dicho, se debe escribir el archivo que se desee copiar y el destino donde se habrá de guardar la copia.

Una opción interesante de este comando es `-a`, la cual hace una copia exacta de los directorios y subdirectorios. Esto también incluye los permisos o links que pudiera haber en el directorio de origen.

Por ejemplo:

```
cp pruebas/holamundo.txt .
```

copiará el archivo **holamundo.txt**, ubicado dentro del directorio **pruebas**, en el directorio actual (representado por **.**) con el mismo nombre.

Por otro lado,

```
cp pruebas/ejemplo.txt copiaejemplo.txt
```

hará una copia del archivo **ejemplo.txt** que se halla dentro de **pruebas**, en el directorio actual, pero con el nombre **copiaejemplo.txt**.

Veamos el resultado de los dos comandos anteriores:

```
alumno@stretch:~$ cp pruebas/holamundo.txt .
alumno@stretch:~$ cp pruebas/ejemplo.txt copiaejemplo.txt
alumno@stretch:~$ ls -l holamundo.txt copiaejemplo.txt
-rw-r--r-- 1 alumno alumno 32 abr  2 19:45 copiaejemplo.txt
-rw-r--r-- 1 alumno alumno 11 abr  2 19:45 holamundo.txt
alumno@stretch:~$
```

Es importante tener en cuenta que, si el archivo de destino existe, se sobrescribirá mediante el proceso de copia.

El comando mv (move)

Esta herramienta se utiliza tanto para *mover* archivos de un lugar a otro como también para *renombrarlos*. Es importante tener en cuenta que si el archivo de destino existe, se sobreescribirá mediante este proceso.

Para ilustrar, cambiemos el nombre de **copiaejemplo.txt** a **nuevacopiaejemplo.txt**. Luego de ejecutar el comando

```
mv copiaejemplo.txt nuevacopiaejemplo.txt
```

el primer archivo no existe más.

A continuación, movamos **holamundo.txt** desde el directorio actual a **archivo3**.

```
mv holamundo.txt archivo3
```

En la imagen siguiente vemos el resultado de los comandos anteriores:

```
alumno@stretch:~$ mv copiaejemplo.txt nuevacopiaejemplo.txt
alumno@stretch:~$ ls -l copiaejemplo.txt
ls: no se puede acceder a 'copiaejemplo.txt': No existe el fichero o el directorio
alumno@stretch:~$ ls -l nuevacopiaejemplo.txt
-rw-r--r-- 1 alumno alumno 32 abr  2 19:45 nuevacopiaejemplo.txt
alumno@stretch:~$ mv holamundo.txt archivo3
alumno@stretch:~$ ls -l holamundo.txt
ls: no se puede acceder a 'holamundo.txt': No existe el fichero o el directorio
alumno@stretch:~$ ls -l archivo3/holamundo.txt
-rw-r--r-- 1 alumno alumno 11 abr  2 19:45 archivo3/holamundo.txt
alumno@stretch:~$
```

Vemos que **holamundo.txt** ya no se encuentra en el directorio actual pero sí dentro del directorio **archivo3**.

El comando rm (remove)

Este comando se utiliza para borrar archivos. Debemos tener en cuenta que desde la línea de comandos no tenemos papelera de reciclaje, y tampoco hay un *undelete*, así que **cuando borramos nunca más podemos recuperar el archivo original**. Para remover un directorio junto con todo su contenido empleamos la opción `-r`, **con sumo cuidado** por la misma razón señalada antes.

Debido a que no podemos recuperar archivos borrados con `rm`, es una buena idea utilizar este comando **siempre** con la opción `-i`, lo que nos pedirá confirmación antes de efectuar la operación. Dicho de otra forma:

```
rm -ri midirectorio
```

es una opción más adecuada que

```
rm -r midirectorio
```

para evitar el borrado accidental de los contenidos de **midirectorio**.

El comando ln

En el capítulo 3 hablamos brevemente de los enlaces simbólicos o *soft links*. En esa oportunidad, por simplicidad dijimos que podíamos considerarlos como si fueran los meros *accesos directos* que conocemos en otros sistemas operativos.

En realidad, podemos considerar a los enlaces simbólicos como una referencia a un archivo determinado, con la siguiente ventaja en Linux. Supongamos que para funcionar, un programa **X** ha sido compilado para utilizar un cierto archivo llamado **Y**. Si este último está sujeto a cambios, en algún momento podemos llegar a encontrarnos con varias versiones del mismo presentes en nuestro sistema, como por ejemplo:

- **Y-version1**
- **Y-version2**

y así sucesivamente.

¿Cómo nos aseguramos de que nuestro programa **X** siempre utilice la versión más actualizada de **Y** (**Y-version2** en este ejemplo)? La respuesta es crear un enlace simbólico a **Y-version2** llamado **Y**. De esta forma, cuando **X** *pida* a **Y**, lo que en realidad obtendrá es **Y-version2**. Cuando haya un nuevo cambio de **Y**, borramos el enlace simbólico anterior, y lo volvemos a crear con la nueva referencia. Esto hace posible que ante los cambios de **Y**, el programa **X** no tenga que ser compilado nuevamente para apuntar a la nueva versión.

En nuestro ejemplo, esto sería:

```
ln -s Y-version2 Y
```

Como podemos observar, luego de `ln -s` (el comando y la opción para crear el enlace simbólico) debe ir el archivo objetivo, seguido del nombre del enlace simbólico. Vale la pena aclarar que se deben incluir las rutas completas (o *absolutas*) si esta operación no se está realizando en el directorio donde ambos -archivo objetivo y enlace- estarán colocados.

En realidad, existen dos tipos de enlaces:

- Los *hard links* realizan un enlace desde un *ínodo* determinado hasta los bloques que componen el archivo. Para crear un link de este tipo omitimos el modificador `-s` que utilizamos al crear un enlace simbólico. Las limitaciones de los *hard links* consisten en que los mismos no pueden apuntar a directorios ni tampoco a objetos que se encuentren fuera del mismo sistema de archivos.
- Los *soft links* representan un enlace desde un *ínodo* hasta otro *ínodo*.

---

*Un **ínodo**, **nodo-i**, **nodo índice** (o **i-node** en inglés) es una estructura de datos propia de los sistemas de archivos tradicionalmente empleados en los sistemas operativos tipo UNIX como es el caso de Linux. Un ínodo contiene las características (permisos, fechas, ubicación, pero NO el nombre) de un archivo regular, directorio, o cualquier otro objeto que pueda contener el sistema de archivos.*

El comando touch

El comando touch nos permite crear un nuevo archivo vacío o actualizar la fecha de modificación de uno existente. Uno se puede preguntar, ¿cuál sería el propósito de crear archivos vacíos? La respuesta es simple. Los archivos vacíos en Linux permiten facilitar la práctica de tareas de manipulación de archivos evitando la demora de tener que ingresar algún tipo de contenido en los mismos.

Para crear un nuevo archivo vacío llamado **ejemplotouch.txt** en el directorio actual:

```
touch ejemplotouch.txt
```

Por otro lado, observemos que cuando utilizamos el comando `touch` contra el archivo existente **datostutor.txt**, se actualiza la fecha de última modificación de **4 de enero 21:06** a **2 de abril 20:03**, tal como se observa en la salida de

```
ls -l datostutor.txt
```

```
alumno@stretch:~$ touch ejemplotouch.txt
alumno@stretch:~$ ls -l ejemplotouch.txt
-rw-r--r-- 1 alumno alumno 0 abr  2 20:03 ejemplotouch.txt
alumno@stretch:~$ ls -l datostutor.txt
-rw-r--r-- 1 alumno alumno 155 ene  4 21:06 datostutor.txt
alumno@stretch:~$ touch datostutor.txt
alumno@stretch:~$ ls -l datostutor.txt
-rw-r--r-- 1 alumno alumno 155 abr  2 20:03 datostutor.txt
alumno@stretch:~$
```

Apagar y reiniciar el sistema

Aunque en teoría Linux puede funcionar durante semanas, meses, e incluso años sin necesidad aparente de reiniciar o o apagar el equipo, pueden surgir situaciones en las que es necesario hacerlo. Esto puede deberse a la necesidad de reemplazar un dispositivo de hardware interno, al mantenimiento eléctrico de la red desde donde recibe energía el equipo, o a un simulacro para verificar que el mismo puede reaccionar correctamente ante un evento inesperado.

Ante tales casos, es preciso conocer las opciones de las que disponemos a fin de proceder correctamente y con cautela. Dentro de lo posible, debemos evitar a toda costa el apagar cualquier equipo (no solamente aquellos que corren Linux) a la fuerza o desconectándolos repentinamente del suministro de energía eléctrica. Para casos críticos (servidores principalmente), los equipos deben estar conectados a una UPSs o a un generador independiente en paralelo.

Veamos cuáles son las opciones de las que disponemos para apagar el equipo. La primera alternativa (`init 0`) funciona por compatibilidad hacia atrás con el *runlevel* 0.

```
init 0
poweroff
halt
shutdown -h [CUANDO] [MENSAJE OPCIONAL]
```

El comando `shutdown` suele ser preferido por el hecho de que permite enviar un mensaje a las terminales conectadas e impedir el login de nuevos usuarios. Por ejemplo:

```
shutdown -r 14:00 'El sistema se reiniciará a las 14 hs'
```

Si deseamos cancelar un apagado agendado de esta manera, bastará utilizar la opción `-c` como se muestra a continuación:

```
shutdown -c
```

Para reiniciar el equipo los comandos disponibles son similares:

```
init 6
reboot
shutdown -r [CUANDO] [MENSAJE OPCIONAL]
```

Bajo **SysVinit**, la utilización de `shutdown` traía aparejada la ventaja adicional de que los procesos que estuvieran corriendo se detenían correctamente, y los sistemas de archivos se desmontaban prolijamente (a diferencia de las otras opciones que provocaban un apagado abrupto del equipo). Por otra parte, en **systemd** todos los comandos anteriores son sinónimos, y constituyen enlaces simbólicos al propio sistema de inicio o a **systemctl**, como podemos apreciar en la imagen siguiente:

```
alumno@stretch:~$ ls -l /sbin/init
lrwxrwxrwx 1 root root 20 jul  5  2017 /sbin/init -> /lib/systemd/systemd
alumno@stretch:~$ ls -l /sbin/poweroff
lrwxrwxrwx 1 root root 14 jul  5  2017 /sbin/poweroff -> /bin/systemctl
alumno@stretch:~$ ls -l /sbin/reboot
lrwxrwxrwx 1 root root 14 jul  5  2017 /sbin/reboot -> /bin/systemctl
alumno@stretch:~$ ls -l /sbin/shutdown
lrwxrwxrwx 1 root root 14 jul  5  2017 /sbin/shutdown -> /bin/systemctl
alumno@stretch:~$
```

Para apagar o reiniciar el sistema de manera correcta (deteniendo los servicios y desmontando sistemas de archivos), podemos también recurrir a

`systemctl poweroff`

o

`systemctl reboot`

respectivamente.