Introducción a GNU/Linux

PID_00270626

Gerard Farràs Ballabriga

Tiempo mínimo de dedicación recomendado: 4 horas

Universitat Oberta de Catalunya

Gerard Farràs Ballabriga

Ingeniero técnico en Informática de sistemas por la Universidad Autónoma de Barcelona (UAB). Ingeniero en Informática y máster en Sociedad de la información y el conocimiento por la Universitat Oberta de Catalunya (UOC). Actualmente trabaja como profesor en una escuela de secundaria y formación profesional. Anteriormente ha desarrollado su actividad profesional en el área de sistemas de información de un centro tecnológico y también como profesional autónomo (*freelance*) trabajando como administrador de sistemas y desarrollador web.

El encargo y la creación de este recurso de aprendizaje UOC han sido coordinados por el profesor: Julià Minguillón Alfonso (2020)

Primera edición: febrero 2020

Avda. Tibidabo, 39-43, 08035 Barcelona
Realización editorial: FUOC

Índice

1. Introducción

El objetivo de este documento consiste en ofrecer una introducción al sistema operativo GNU/Linux para especialistas en tratamiento de datos.

GNU/Linux es un sistema avanzado, con muchas herramientas disponibles y posibilidades de automatización, estable, además de gratuito y con licencia de software libre.

Por estos motivos, se recomienda a los estudiantes el aprendizaje de este sistema operativo. Ciertamente, sería posible hacer la mayoría de operaciones con otros sistemas, pero consideramos que este es el más adecuado. Por otro lado, el aprendizaje sobre el funcionamiento de un sistema operativo se puede realizar desde múltiples perspectivas (por ejemplo, poniendo el foco en el funcionamiento interno del sistema, en su rendimiento o en su seguridad). No obstante, el objetivo de este documento es ofrecer unas primeras pinceladas iniciales sobre qué es GNU/Linux y el enfoque en los comandos necesarios para el tratamiento de datos masivos, así que dejaremos de lado el resto de perspectivas.

El objetivo pedagógico de este documento consiste en ofrecer una introducción a los sistemas GNU/Linux para usuarios no experimentados y en ofrecer a los estudiantes una guía para que puedan emplear este tipo de sistemas para las tareas relacionadas con la ciencia de datos.

Nuestra recomendación para conseguir este objetivo sería disponer de un sistema GNU/Linux (la manera más sencilla es empleando una máquina virtual) y leer, en paralelo, este documento con objeto de ir probando todos los comandos. Los usuarios más experimentados emplearán otros métodos para el aprendizaje. Son los usuarios más neófitos los que necesitarán una guía inicial para empezar. En todo caso, no se trata de memorizar comandos: a medida que se usen, se recordarán por repetición. Así, pues, se recomienda instalar el sistema, leer este documento y hacer algunas primeras pruebas. El orden de lectura tampoco es excesivamente importante: se pueden leer y probar apartados concretos, no es necesario que la lectura sea lineal. A medida que el aprendizaje avance, esta guía perderá interés.

Usuarios experimentados

Los usuarios experimentados emplearán otros métodos para el aprendizaje, por ejemplo, utilizando los manuales internos de la máquina, búsquedas avanzadas en internet y más «ensayo-error»

2. Breve historia de Linux

Linux es un sistema operativo de software libre basado en el *kernel* desarrollado por Linus Torvalds y liberado por primera vez el 17 de septiembre de 1991. Linux acostumbra a ir empaquetado en distribuciones que contienen, además del *kernel*, software de sistema, librerías y utilidades diversas, muchas de ellas del proyecto GNU. Este GNU[1] fue iniciado por Richard Stallman con el objetivo de generar software libre con la licencia pública general (GPL).

(1)Acrónimo recursivo que juega con las palabras y significa *GNU's Not Unix*.

Kernel

Con *kernel* nos referimos al corazón del sistema operativo.

El 25 de agosto de 1991, el estudiante finlandés Linus Torvalds escribía este famoso mensaje en la lista de noticias comp.os.minix, en el que anunciaba el embrión de lo que sería Linux:

Minix

Minix es otro sistema operativo de entorno académico.

Es posible leer el hilo entero en «What would you like to see most in minix?».

```
Hello everybody out there using Minix -

I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu)
 for 386(486) AT clones. This has been brewing since April, and is starting to get ready.
 I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat
 (same physical layout of the file-system (due to practical reasons) among other things).

I've currently ported bash(1.08) and gcc(1.40), and things seem to work. This implies that
I'll get something practical within a few months, and I'd like to know what features most
people would want. Any suggestions are welcome, but I won't promise I'll implement them :-)

Linus (torv...@kruuna.helsinki.fi)

PS. Yes - it's free of any minix code, and it has a multi-threaded fs. It is NOT portable
(uses 386 task switching etc), and it probably never will support anything other than
AT-harddisks, as that's all I have :-(.
```

Desde que el desarrollo del *kernel* empezó a emplear un software que se denomina *Git* (en 2005) y que ha permitido, entre otras cosas, ver mejor las contribuciones al código fuente de este *kernel*, se puede afirmar que un total de 15.637 desarrolladores han contribuido al sistema. En 1991 este *kernel* contenía un pequeño número de ficheros escritos en lenguaje de programación C. La versión 4.15 del año 2018 contenía más de 23 millones de líneas de código.

Git

Git es un software para el control de versiones, también diseñado por Linus Torvalds.

Linux es un ejemplo prominente de **colaboración abierta en el desarrollo de software**, puesto que, durante años, se ha generado un producto altamente complejo gracias a la intervención de centenares de personas distribuidas en todo el planeta con un modelo abierto y más o menos descentralizado.

Lectura recomendada

J. Corbet; G. Kroah-Hartman (2017). «Linux Kernel Development Report» [en línea]. Fundación Linux. Disponible en: go.pardot.com/l/6342/2017-10-24/3xr3f2/6342/188781/Publication_LinuxKernelReport_2017.pdf.

Respecto a la licencia pública general GNU GPL,[2] es la licencia para software libre más utilizada, que permite hacer copias, distribuirlas (tanto en versiones comerciales como las que no lo son) y modificar el código fuente, siempre que cualquier modificación se continúe distribuyendo con la misma licencia. Lo que no permite la licencia GPL es la distribución de software ejecutable sin disponer del código fuente correspondiente. Richard Stallman, de la Free Software Foundation, fue el primer autor del proyecto GNU, que otorgaba a los destinatarios de un programa de ordenador los derechos correspondientes al software libre. Se trata de una licencia *copyleft*, lo que significa que los trabajos derivados se tienen que distribuir con los mismos términos.

(2)GNU GPL, GNU *General Public License*.

Free Software Foundation

La Free Software Foundation (FSF) es una organización sin ánimo de lucro global que tiene como objetivo la promoción del software libre (en el sentido de otorgar libertad al usuario).

Figura 1. Fotografía de Linus Torvalds recibiendo un premio en 2018

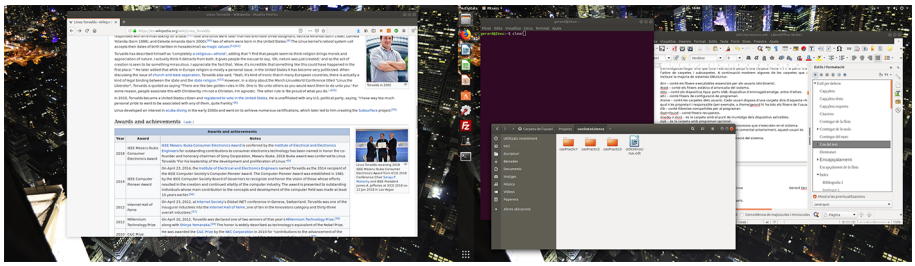
Fuente: imagen extraída de la Wiquipedia (en.wikipedia.org/wiki/Linus_Torvalds).

3. Entorno gráfico contra línea de comandos

La mayoría de sistemas operativos actuales se utilizan con entornos de ventanas (por medio de interfaces gráficas de usuario),[3] aunque siguen manteniendo la posibilidad de ser utilizados por medio de terminales. El uso de un terminal es ciertamente más complicado y menos intuitivo para los usuarios no experimentados, pero otorga más flexibilidad. Además, hay tareas que solamente se pueden realizar empleando comandos de terminal. La gran ventaja del uso de terminales consiste en la posibilidad de automatización de tareas por medio de *scripts* (guiones de secuencias). Otra ventaja consiste en el hecho de que hay multitud de herramientas que funcionan solamente por medio de terminal, y no con software basado en ventanas.

[3] En inglés GUI, *Graphical User Interface*.

Figura 2. Captura de pantalla del escritorio de una máquina con la distribución Ubuntu en entorno gráfico



Este escritorio concreto contiene dos pantallas diferentes.

Consejo

Aconsejamos trabajar en una instalación similar con entorno gráfico, pero será necesario aprender a utilizar los comandos de terminal.

Así, pues, el alumnado deberá familiarizarse con el uso de terminales en sistemas operativos GNU/Linux. Una vez instalado un sistema operativo GNU/Linux en modo gráfico, hay que buscar solamente la aplicación de «Terminal» en la barra de aplicaciones. El resto del manual que escribimos a continuación se basará en los detalles de este entorno.

Cuando abrimos el terminal, observaremos que aparece una cadena de texto similar a la siguiente:

```
usuario@nombreMaquina:$
```

Figura 3. Captura de pantalla de un terminal sobre un fondo de pantalla con entorno gráfico

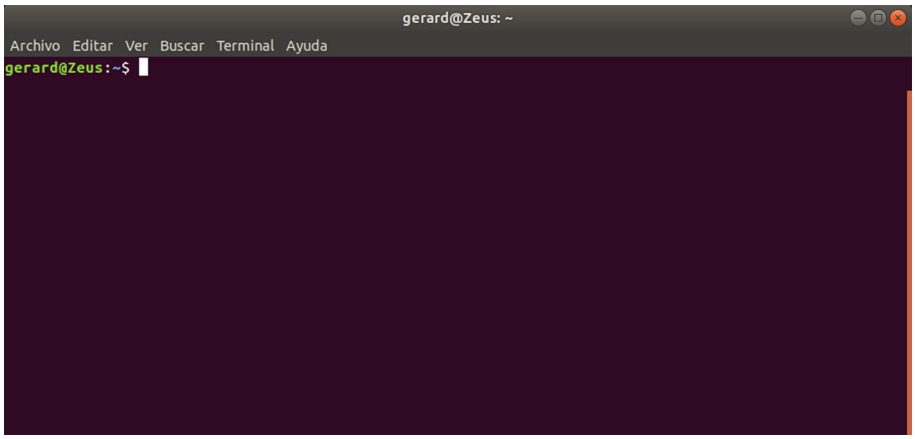


Figura 3

En este caso, podemos observar que el usuario es Gerard y el nombre de la máquina (*hostname*) es Zeus.

Esto es el ***prompt***, que permanece a la espera de que el usuario introduzca comandos y los ejecute con la tecla de «retorno». Este *prompt* contiene el nombre de usuario, una arroba y el nombre de la máquina (técnicamente se denomina *hostname*). El software que permanece a la espera e interpreta lo que escribe el usuario se denomina ***shell*** y, en el marco de esta asignatura, emplearemos el software **Bash**, aunque hay otros.

Los terminales están diseñados para ejecutar los comandos uno detrás del otro, sin la idea «de ir hacia atrás». Cuando el terminal esté todo ocupado por letras y queramos vaciarlo, se puede emplear el comando siguiente para limpiarlo:

```
usuario@nombreMaquina:$ clear
```

Este comando se puede emplear tantas veces como haga falta y sin que tenga ninguna afectación.

Bash contiene algunos trucos que permiten a los usuarios ganar velocidad en el teclado y no tener que escribir siempre todos los comandos. Comentamos algunos a continuación:

1) Con la flecha superior del teclado, se reescribirán los comandos que se han ejecutado anteriormente.

2) La tecla «tabulador» escribe de manera automática el nombre de los programas o ficheros que estén disponibles. A continuación, el usuario escribe las letras «ho» y pulsa el tabulador: aparecen todos los comandos que empiezan por «ho». Si escribe una letra más, la «s», y pulsa de nuevo el tabulador, se escribe solamente el comando que empieza por «hos», en este caso *host*:

```
usuario@nombreMaquina:$ ho            (el usuario pulsa el tabulador)
hoichess hoixiangqi host hostid hostname hostnamectl
usuario@nombreMaquina:$ host
```

Con el comando `history` se pueden ver los últimos comandos ejecutados:

```
usuario@nombreMaquina:$ history
 1987 gzip -d docs.tar.gz
```

```
 1988 tar -xvf docs.tar
 1989 exit
 1990 hostclear
 1991 clear
 1992 clear
 1993 clear
 1994 history
 1995 history | more
```

Y se pueden ejecutar de nuevo empleando un identificador del comando (el número que lo acompaña), precedido de un signo de exclamación. Por ejemplo:

```
usuario@nombreMaquina:$ !1991
```

De manera similar, se pueden ejecutar comandos que ya se han ejecutado anteriormente de manera directa empleando el carácter «!» con las primeras letras del comando que se desee ejecutar. Por ejemplo:

```
usuario@nombreMaquina:$ !wge
```

Finalmente, con la combinación de teclas «control» + «r», aparece un buscador de comandos en el que se pueden introducir letras concretas de comandos también anteriores:

```
(reverse-i-search)'c': clear
```

4. Usuarios del sistema

El sistema operativo puede integrar varios usuarios. Durante el proceso de instalación, el asistente solicitará el nombre de usuario y la contraseña para, al menos, un usuario, además de la contraseña para el usuario administrador (que, en entornos GNU/Linux se denomina *root*). Así, pues, en un sistema operativo estándar, se dispondrá de, como mínimo, dos usuarios, un usuario *root* y otro usuario normal.

La información de todos los usuarios se aloja en el fichero «/etc/password», que contiene los campos siguientes, donde el carácter «:» actúa como delimitador. En concreto, la información que almacena cada línea es la siguiente:

- el nombre de usuario,
- la contraseña cifrada (en las distribuciones actuales esta información está en el fichero «/etc/shadow», así que ya no reside aquí),
- un número identificador de usuario,
- el identificador del grupo al cual pertenece el usuario,
- el nombre completo,
- la carpeta con los ficheros del usuario (se denomina técnicamente la *home*), y
- el software que actuará de intérprete de los comandos (en el caso de esta asignatura «/bin/bash»).

Por ejemplo:

```
usuario@nombreMaquina:$ cat /etc/passwd | grep usuario
usuario:x:1000:1000:usuario,,,:/home/usuario:/bin/bash
```

Con el comando `id` podemos conocer el identificador de usuario, y también los grupos a los que pertenece:

```
usuario@nombreMaquina:$ id
uid=1000(usuario) gid=1000(usuario) grupos=1000(usuario),4(adm),24(cdrom),27(sudo),30(dip),
46(plugdev),113(lpadmin),128(sambashare)
```

El usuario administrador que dispone de todos los permisos del sistema se denomina *root* y tiene el identificador «0». Es posible «suplantar» al usuario administrador con los comandos `sudo` y `su`. A continuación mostramos un par de ejemplos:

```
usuario@nombreMaquina:$ sudo apt-get install xboard #Comando que accederá como root
(con autenticación previa e instalará por medio de los repositorios de software el paquete
xboard). Este comando solamente ejecutará una instrucción como superusuario.
```

Recomendación

Se recomienda a todos los usuarios que siempre empleen un usuario concreto (por ejemplo, «usuario») y que no se acostumbren a trabajar habitualmente con el usuario *root*. Sin

embargo, suele pasar a menudo que un usuario concreto tiene que ejecutar comandos para los que no dispone de permisos.

En el supuesto de que requiramos ejecutar comandos diversos como administrador, podemos convertirnos en este empleando el comando:

```
usuario@nombreMaquina:$ sudo su
```

Una vez finalizadas las tareas de administración, podemos volver al usuario normal estándar con el comando siguiente:

```
usuario@nombreMaquina:$ exit
```

5. El sistema de ficheros

El sistema operativo GNU/Linux dispone de un sistema de ficheros para el almacenamiento de la información y los datos estructurado por medio de carpetas y ficheros.

A diferencia de los sistemas basados en Microsoft Windows, GNU/Linux no utiliza unidades tipo «C:», «D:» o «E:» para referirse a espacios de almacenamiento, sino que toda la estructura cuelga de una carpeta inicial (/). De ahí cuelga todo el árbol de carpetas y subcarpetas. A continuación mostramos algunas de las carpetas que acostumbran a incluir la mayoría de sistemas GNU/Linux:

Carpeta	Contenido
/bin	Contiene los ficheros ejecutables esenciales para los usuarios (los binarios).
/boot	Contiene los ficheros estáticos de arrancada del sistema.
/dev	Contiene los dispositivos tipo puertos USB, dispositivos de almacenamiento, entre otros.
/etc	Contiene ficheros de configuración de software.
/home	Contiene las carpetas de los usuarios. Cada usuario dispone de una carpeta dentro de esta «home», de la cual es propietario y responsable (por ejemplo, en «/home/usuario» están todos los ficheros del usuario «usuario»).
/lib	Contiene librerías compartidas para el software.
/lost+found	Contiene ficheros recuperados.
/media o /mnt	Carpeta con el punto de montaje de los dispositivos extraíbles.
/opt	Carpeta con software opcional.
/proc	Contiene ficheros del *kernel* e información de los procesos que se ejecutan en el sistema.
/root	«home» del usuario root. Tal como hemos comentado anteriormente, este usuario es el administrador del sistema.
/sbin	Contiene ficheros ejecutables para la administración del sistema.
/tmp	Contiene ficheros temporales.
/usr	Contiene ficheros ejecutables (binarios) de los usuarios.
/var	Contiene ficheros de datos variables.

A continuación veremos algunos primeros comandos de terminal para navegar por medio de esta estructura de carpetas.

`pwd`[4] es la herramienta que muestra en qué carpeta reside el usuario en ese momento. Se trata de una herramienta de cariz solamente informativo, que mostrará al usuario en qué carpeta está y no ejecuta ninguna otra acción. Por ejemplo:

(4)Proviene del acrónimo *Print Working Directory*.

```
usuario@nombreMaquina:$ pwd
/home/usuario
```

`ls` es una herramienta para listar los ficheros que están en un directorio. Las opciones más utilizadas son las siguientes:

- `a` mostrará todos los ficheros (los ficheros que se inician con un punto . están ocultos y no se ven con un `ls` normal).
- `l` proporciona detalles con un formato de lista largo.
- `t` ordena por fecha de modificación, primero el más nuevo.
- `S` ordena las entradas por medida, de mayor a menor.
- `r` invierte el orden.

Toda la información del comando `ls` se puede consultar en la página del manual de la herramienta, accesible con el comando:

```
usuario@nombreMaquina:$ man ls
```

También es posible ver un resumen de los parámetros disponibles empleando el comando:

```
usuario@nombreMaquina:$ ls --help
```

Algunos ejemplos sencillos:

```
usuario@nombreMaquina:# ls
CCA0.aux CCA0.synctex.gz CCA0.tex CCA1.tex

usuario@nombreMaquina:# ls -l
total 16
-rw-r--r-- 1 root root 454 de ma 5 21:01 CCA0.aux
-rw-r--r-- 1 root root 3803 de ma 5 21:01 CCA0.synctex.gz
-rw-r--r-- 1 root root 3497 de ma 5 21:01 CCA0.tex
-rw-r--r-- 1 root root 3497 de ma 10 19:22 CCA1.tex
```

`cd` es la herramienta para cambiar de la carpeta actual. Si el usuario está en «/home/usuario/» y, ejecutando `ls` observamos que hay una carpeta que se denomina «PACS», será posible acceder a ella usando el comando `cd PACS`.

Para regresar a la carpeta anterior, habrá que ejecutar `cd ..`

Es posible acceder directamente a la *home* del usuario usando el comando `cd` sin emplear ningún parámetro. Por ejemplo:

```
usuario@nombreMaquina:/Projects$ pwd
```

```
/home/usuario/Projects
usuario@nombreMaquina:/Projects$ cd
usuario@nombreMaquina:$ pwd
/home/usuario
```

`mkdir`[5] es una herramienta para crear carpetas. Con el parámetro `-p` es posible crear más de una carpeta a la vez. Mostramos un par de ejemplos:

(5)Del inglés *make directory*.

```
usuario@nombreMaquina:$ mkdir datos
usuario@nombreMaquina:$ mkdir -p PACS/PAC1/
```

`cp` es un comando que permite copiar ficheros. Los argumentos que hay que indicarle a este comando son, en primer lugar, el fichero origen (qué se desea copiar) y, en segundo lugar, dónde se desea realizar la copia. Por ejemplo:

```
usuario@nombreMaquina:$ cp /etc/passwd /home/usuario/PACS/ #Copiará el fichero passwd ubicado
dentro de la carpeta etc en la carpeta /home/usuario/PACS.
```

`mv` es un comando que sirve para mover o renombrar ficheros. Algunos ejemplos:

```
usuario@nombreMaquina:$ mv enunciadoPAC1.doc respuestasPAC1.doc #Este comando
cambiará el nombre del fichero.
usuario@nombreMaquina:$ mv respuestasPAC1.doc PAC1 #Este otro comando movería el fichero
a la carpeta PAC1 (que ya existe).
```

`rm`[6] es un comando para suprimir ficheros. Algunos ejemplos:

(6)Proviene de *remove*.

```
usuario@nombreMaquina:$ rm respuestasPAC1.doc #Este comando suprimirá el fichero indicado.
usuario@nombreMaquina:$ rm -R PACS/ #Esta carpeta suprimirá de manera recursiva
todos los ficheros que hay dentro de la carpeta PACS.
```

`rmdir` es un comando para suprimir directorios vacíos.

```
usuario@nombreMaquina:$ rmdir PACS #Este comando suprimirá la carpeta PACS.
```

En el supuesto de que la carpeta no esté vacía, habrá que suprimir previamente todos los ficheros:

```
usuario@nombreMaquina:$ rm -R PACS
```

`touch` permite cambiar la fecha de un fichero o crear un nuevo fichero en blanco vacío. Este comando cambiará la fecha de acceso al fichero a la actual. Si el fichero al que se refiere no existe, se creará un fichero en blanco.

```
usuario@nombreMaquina:$ touch fichero.txt
usuario@nombreMaquina:$ ls -l fichero.txt
-rwxr-x--x 1 usuario usuario 0 de se 29 18:14 fichero.txt
```

`df` muestra el uso de espacio de disco utilizado en cada partición del sistema. El parámetro `-h` mostrará los datos en megabytes o gigabytes (mostrará una versión más legible para los humanos).

```
usuario@nombreMaquina:$ df -h
S. ficheros Medida En uso Libre %Uso Montado en
udev 7,8G 0 7,8G 0% /dev
tmpfs 1,6G 2,0M 1,6G 1% /run
/dev/sdb1 459G 125G 311G 29% /
```

```
tmpfs 7,9G 81M 7,8G 2% /dev/shm
tmpfs 5,0M 4,0K 5,0M 1% /run/lock
tmpfs 7,9G 0 7,9G 0% /sys/fs/cgroup
tmpfs 1,6G 16K 1,6G 1% /run/user/127
tmpfs 1,6G 40K 1,6G 1% /run/user/1000
```

`ln` es una herramienta para crear enlaces entre ficheros (se trata del concepto de accesos directos de Windows). Acostumbraremos a realizar *soft links*. Por ejemplo:

```
usuario@nombreMaquina:$ ln -s /etc/passwd contraseñas.txt #Este comando creará en la carpeta
actual un fichero contraseñas.txt que será un acceso directo a /etc/passwd.
usuario@nombreMaquina:$ ls -l
lrwxrwxrwx 1 usuario usuario 11 de ju 22 11:13 contraseñas.txt -> /etc/passwd
```

El primer carácter de la línea anterior es una «l», que indica que se trata de un enlace (*link*).

`chown` es la herramienta que hay que utilizar para cambiar el propietario de un fichero. Cada fichero pertenece a un propietario y también a un grupo propietario. Examinemos el ejemplo siguiente:

```
usuario@nombreMaquina:$ touch hola.txt #Con este comando crearemos un fichero hola.txt.
usuario@nombreMaquina:$ ls -l hola.txt #Y con este otro comando veremos quién es el propietario.
-rw-r--r-- 1 usuario usuario 0 de ju 22 11:15 hola.txt
```

El primer «usuario» se refiere al usuario propietario del fichero. El segundo se refiere al grupo propietario. En los ficheros «/etc/passwd» se pueden ver el listado de usuarios existentes en un sistema. En el fichero «/etc/group» está el listado de los grupos. En este caso concreto, hay un grupo que se denomina igual que el usuario, aunque no siempre es así.

Con el comando siguiente estableceremos el cambio de propietario y de grupo propietario del fichero «hola.txt». Para traspasar un fichero para que sea propiedad de *root*, habrá que emplear la herramienta `sudo`, que permite ejecutar comandos como si fuéramos *root*.

```
usuario@nombreMaquina:$ sudo chown root:root hola.txt
usuario@nombreMaquina:$ ls -l hola.txt
-rw-r--r-- 1 root root 0 de ju 22 11:15 hola.txt
```

`chmod` es la herramienta empleada para cambiar los permisos de ficheros. En entornos GNU/Linux, hay tres permisos y tres tipos de usuario:

Permisos	Usuarios
«r» de lectura	«u», el propietario del fichero
«w» de escritura	«g», el grupo propietario
«x» de ejecución	«o» para el resto

Para un fichero, «ejecutar» significa poder ejecutar el programa que hay dentro. Para un directorio, el permiso de ejecución es necesario para hacer cualquier cosa dentro de la carpeta.

```
usuario@nombreMaquina:$ ls -l script.sh
-rw-r--r-- 1 usuario usuario 0 de ju 22 11:15 script.sh
```

Con este comando observamos los permisos de este hola.txt. El usuario *root* puede hacer `rw-` (puede leer y escribir, pero no ejecutar, el grupo propietario puede solamente leer `r--` y, finalmente, el resto de usuarios, pueden solamente leer `r--`). El primer guion sirve para conocer de qué tipo de fichero se trata (las carpetas contienen una «d» y los enlaces una «l», tal como hemos visto previamente).

Para otorgar permiso de ejecución, podemos realizar lo siguiente:

```
usuario@nombreMaquina:$ chmod u+x script.sh
```

Y ahora:

```
usuario@nombreMaquina:$ ls -l script.sh
-rwxr--r-- 1 usuario usuario 0 de ju 22 11:15 script.sh
```

El parámetro `u` del comando `chmod` se refería al usuario. De manera similar, podríamos haber empleado `g` para el grupo y `o` para los otros.

Hay otra manera de emplear el comando `chmod`. Consideramos que los permisos consisten en tres números (observamos que hay tres permisos para tres objetos –usuario, grupo y el resto). Consideramos ahora estos permisos en binario (un «0» si no se le otorga y un «1» si se le otorga). De binario, traspasamos a decimal.

Por ejemplo:

- 7 se escribe en binario 111.
- 5 se escribe en binario 101.
- 1 se escribe en binario 001.

Es posible establecer permisos empleando esta codificación:

```
usuario@nombreMaquina:$ chmod 751 fichero.txt
usuario@nombreMaquina:$ ls -l fichero.txt
-rwxr-x--x 1 usuario usuario 0 de se 16 06:44 fichero.txt
```

En este caso, el usuario «usuario» puede realizar todas las operaciones (leer, escribir y ejecutar), que se leería como 111 y en binario corresponde al número 7, el grupo «usuario» solamente leer y ejecutar (`r-w`), que en binario se leería 101 y corresponde al número 5 y, finalmente, el resto de usuarios solamente ejecutar (`--x`), que en binario se leería 001 y corresponde al número 1.

6. Búsqueda de ficheros

Para buscar ficheros en el sistema, mencionaremos un par de comandos. El primero de ellos es el comando `find`, que, indicando como parámetro una ruta, realizará la búsqueda de ficheros que contengan ciertos patrones. El esquema del comando es tal como sigue a continuación:

```
find opciones ruta_de_investigación expresión
```

En el ejemplo que mostramos a continuación, se realizará una investigación del fichero «testfile.txt» a partir de la ruta actual y de todas las subcarpetas presentes en la carpeta:

```
usuario@nombreMaquina:$ find . -name testfile.txt
```

En este otro ejemplo, es lo mismo pero parte de la carpeta «/var/www/» y busca solamente ficheros que finalicen con la extensión «.html»:

```
usuario@nombreMaquina:$ find /var/www/ -name "*.html"
```

En el ejemplo que mostraremos a continuación hay tres parámetros nuevos. El parámetro `user` indica el usuario propietario del fichero. El parámetro `mtime` indica el número de días que hace que el fichero fue modificado y el parámetro `iname` es el mismo que `name` pero haciendo que el texto no sea sensible entre minúsculas y mayúsculas:

```
usuario@nombreMaquina:$ find /home -user exampleuser -mtime 7 -iname ".db"
```

Por lo tanto, este comando buscaría ficheros modificados durante los últimos siete días, propiedad del usuario exampleuser y que acabaran con la extensión «.db», ya sea en minúsculas o mayúsculas.

Con el parámetro `-delete` sería posible suprimir los ficheros que se hubieran encontrado:

```
usuario@nombreMaquina:$ find . -name "*.bak" -delete
```

El comando `locate` es más rápido buscando ficheros que el comando anterior, puesto que, en realidad, no los busca, sino que mira en una base de datos. No obstante, existen dos desventajas en el uso de este comando:

- No es tan flexible como `find` (que permite buscar por nombre, fechas de modificación, etc.).
- Parte de una base de datos que hay que construir y actualizar a menudo.

De todas formas, el uso del comando es muy sencillo y es muy rápido.

El comando para la creación de la base de datos se tiene que realizar como *root*:

```
usuario@nombreMaquina:$ sudo updatedb
```

Hay que ejecutar este comando a menudo. Idealmente, se pone en el Cron para que se ejecute de manera automática.

Con la base de datos construida, ya es posible buscar ficheros:

```
usuario@nombreMaquina:$ locate mkpasswd
/usr/bin/grub-mkpasswd-pbkdf2
/usr/bin/mkpasswd
/usr/share/man/man1/grub-mkpasswd-pbkdf2.1.gz
/usr/share/man/man1/mkpasswd.1.gz
```

7. Manipulación de ficheros

En este apartado trataremos el tema de la manipulación de ficheros por medio de las herramientas de un terminal con GNU/Linux.

Primero empezaremos indicando en qué consisten los **descriptores de ficheros**: se trata de enteros positivos que representan ficheros abiertos.

En el supuesto de que el sistema disponga de cincuenta ficheros abiertos, habrán cincuenta descriptores que apunten a estos. Dado que en los sistemas operativos basados en Unix todo es un fichero, los procesos incorporan **descriptores «especiales»**:

Descriptor	Tarea	Operador
stdin	Un descriptor de fichero que representa la entrada de datos de los procesos y al que se le asigna el entero 0.	<
stdout	Un descriptor de fichero que apunta dónde se publican los resultados de las ejecuciones y al que se le asigna el entero 1.	>
stderr	Un descriptor que apunta en el fichero dónde se incorporan los errores y al que se le asigna el 2.	2>

Operadores

Con los operadores es posible sobrescribir los descriptores.

En el ejemplo siguiente, el resultado del comando `ls` se redirige a un fichero externo:

```
usuario@nombreMaquina:$ ls -l > ficheros.txt
```

Hay que tener en cuenta que, en el supuesto de que «ficheros.txt» ya existiera, se sobrescribiría y se perdería la información que contuviera. En el supuesto de que se desee agregar información al ya existente, habría que emplear el operador >>.

Por ejemplo, el comando siguiente agregaría a «ficheros.txt» el resultado del comando `pwd`:

```
usuario@nombreMaquina:$ pwd >> ficheros.txt
```

La entrada estándar también se puede modificar empleando el carácter <. Por ejemplo:

```
usuario@nombreMaquina:$ sort < listado.txt
```

La entrada del comando `sort` proviene del fichero «listado.txt».

Hay otro operador de interés para comentar en este apartado. Son los ***pipes***, que enlazan el resultado de un comando (`stdout`) con la entrada de otro (`stdin`). Estos *pipes* se representan con el carácter «|». En el ejemplo siguiente, el primer comando mostrará todos los procesos existentes en el sistema. El resultado de este comando se enlazará con un *grep* que filtrará las líneas que contienen la palabra «firefox»:

```
usuario@nombreMaquina:$ ps -a | grep firefox
```

Una vez visto el tema de los descriptores de los ficheros y redirecciones, veamos algunos comandos de terminal útiles para la gestión de ficheros.

`cat` es un comando que permite mostrar el contenido de un fichero en un terminal. Por ejemplo:

```
usuario@nombreMaquina:$ cat /etc/passwd
```

`head` es un comando que permite mostrar solamente las primeras líneas de un fichero. El comando siguiente, por ejemplo, mostrará solamente las primeras siete líneas del fichero «/etc/passwd»:

```
usuario@nombreMaquina:$ head -7 /etc/passwd
```

`tail` es un comando similar al anterior, pero que muestra solamente las últimas líneas. Para ver las siete últimas líneas del mismo fichero anterior, es posible ejecutar:

```
usuario@nombreMaquina:$ tail -7 /etc/passwd
```

Combinando los comandos `head` y `tail`, y uniéndolos empleando *pipes*, podemos realizar tareas como las de visualizar solamente una línea en concreto. Por ejemplo:

```
usuario@nombreMaquina:$ head -12 /etc/passwd | tail -1
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
```

El primer comando `head -12 /etc/passwd` muestra las 12 primeras líneas del fichero indicado, aunque la salida no es en `sdtout`, sino que se redirige a la entrada del comando `tail`, que mostrará solamente la última línea.

`cut` es un comando que permite extraer campos de un fichero de texto delimitado por caracteres concretos. Algunos ejemplos:

El fichero «/etc/password» contiene varios campos separados por el carácter «:». La opción `-d` del comando `cut` especifica este delimitador. La opción `-f` indica qué campos queremos mostrar. El comando siguiente mostrará los campos 6 y 7 de este fichero:

```
usuario@nombreMaquina:$ cut -d":" -f6,7 /etc/passwd
```

```
/root:/bin/bash
/usr/sbin:/usr/sbin/nologin
/home/usuario:/bin/bash
/home/sandra:/bin/bash
```

Este otro ejemplo mostrará solamente del carácter 1 al carácter 9 del fichero «/etc/hosts».

```
usuario@nombreMaquina:$ cut -c 1-9 /etc/hosts
127.0.0.1
127.0.1.1
# The fol
::1 i
fe00::0 i
ff00::0 i
ff02::1 i
ff02::2 i
```

`tr` es un comando que sirve para sustituir, suprimir o restringir algunos caracteres concretos. Veamos algunos ejemplos. En el primero, el comando se prepara para realizar sustituciones:

```
usuario@nombreMaquina:$ tr abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ
grado de datos
GRADO DE DATOS
```

O este otro, que suprime la letra 'a':

```
usuario@nombreMaquina:$ echo "grado de datos" | tr -d 'a'
grudo de dtos
```

`wc` es un comando que recuenta el número de líneas, de palabras o de bytes que contiene un fichero determinado.

```
usuario@nombreMaquina:$ wc /etc/passwd
 49 85 2787 /etc/passwd
```

Así, pues, vemos que este fichero contiene 49 líneas, 85 palabras y ocupa 2.787 bytes. Hay parámetros que permiten ver algunos campos concretos. Por ejemplo, este, que muestra solamente el número de líneas:

```
usuario@nombreMaquina:$ wc -l /etc/passwd
49 /etc/passwd
```

`tee` es un comando que copia información de la entrada estándar en el `stdout`, pero también en un fichero externo. Así, pues, duplica la información de salida. El comando siguiente muestra en formato lista los ficheros dentro de la carpeta «/etc/» que empiezan por *host*. Traspasa este listado por medio de un *pipe* al comando `wc`, que recontará el número de líneas y, el resultado de este comando lo traspasa a `tee`, que almacenará este resultado en el fichero «número.txt», pero que también lo mostrará por pantalla.

```
usuario@nombreMaquina:$ ls -l /etc/host* | wc -l | tee número.txt
5
usuario@nombreMaquina:$ more número.txt
5
```

`paste` es un comando que sirve para mezclar línea por línea el contenido de varios ficheros con un resultado de salida. Veamos un ejemplo:

```
usuario@nombreMaquina:$ cat n.txt
1
2
3
4

usuario@nombreMaquina:$ cat a.txt
Gerard
Sandra
Joel
Julià

usuario@nombreMaquina:$ paste n.txt a.txt
1   Gerard
2   Sandra
3   Joel
4   Julià
```

`sort` es un comando que sirve para ordenar el contenido de varios ficheros. Se pueden ordenar los ficheros de manera ascendente o descendente. Veamos un par de ejemplos.

El comando siguiente ordenará el contenido del fichero especificado a partir del 5.º campo, donde los diferentes campos son definidos por el carácter «:».

```
usuario@nombreMaquina:$ sort -k 5 -t ":" /etc/passwd
```

Como comentábamos, también es posible realizar el orden inverso:

```
usuario@nombreMaquina:$ paste n.txt a.txt | sort -r
4   Julià
3   Joel
2   Sandra
1   Gerard
```

`uniq` es un comando que puede servir para reportar o filtrar las líneas repetidas en un fichero. Hay que prestar atención, porque el comando examina solamente las líneas adyacentes (líneas que están una bajo la otra):

```
usuario@nombreMaquina:$ more a.txt
Gerard
Gerard
Sandra
Joel
Julià

usuario@nombreMaquina:$ uniq a.txt
Gerard
Sandra
Joel
Julià

usuario@nombreMaquina:$ uniq -c a.txt
 2 Gerard
 1 Sandra
 1 Joel
 1 Julià
```

`file` es un comando que sirve para obtener información sobre el tipo de fichero. Mostramos algunos ejemplos:

```
usuario@nombreMaquina:$ file /etc/passwd
/etc/passwd: ASCII text

usuario@nombreMaquina:$ file generaWeb.sh
generaWeb.sh: Bourne-Again shell script, ASCII text executable

usuario@nombreMaquina:$ file /usr/share/tuxpaint/templates/rocks.jpg
/usr/share/tuxpaint/templates/rocks.jpg: JPEG image data, JFIF standard 1.01, resolution (DPI),
density 72x72, segment length 16, baseline, precision 8, 1296x864, frames 3
```

Con el parámetro `-i` el comando `file` mostrará información sobre la codificación del fichero.

```
usuario@nombreMaquina:$ file -i /etc/passwd
/etc/passwd: text/plain; charset=os-ascii
```

`iconv` es un comando que permite convertir la codificación de un fichero de texto. Por ejemplo, en el caso de que se deseara convertir un fichero de texto escrito en codificación de UTF-8 y traspasarla a codificación ASCII:

```
nombreMaquina:$ iconv -f ISO-8859-1 -t UTF-8//TRANSLIT input.file -u out.file
```

Con el parámetro `-l` es posible ver todo un listado de codificaciones de caracteres (truncamos la salida para no ocupar demasiado espacio):

```
usuario@nombreMaquina:$ iconv -l
La siguiente lista contiene todos los conjuntos de caracteres conocidos. Esto no quiere decir
necesariamente que todas las combinaciones de estos nombres se puedan usar como parámetros
FROM y TO en la línea de órdenes. Un determinado conjunto de caracteres puede aparecer con
varios nombres (aliases).

 437, 500, 500V1, 850, 851, 852, 855, 856, 857, 858, 860, 861, 862, 863, 864, 865, 866, 866NAV,
 869, 874, 904, 1026, 1046, 1047, 8859_1, 8859_2, 8859_3, 8859_4, 8859_5, 8859_6, 8859_7,
 8859_8, 8859_9, 10646-1:1993, 10646-1:1993/UCS4, ANSI_X3.4-1968, ANSI_X3.4-1986, ANSI_X3.4,
 ANSI_X3.110-1983, ANSI_X3.110, ARABIC, ARABIC7, ARMSCII-8, ASCII, ASMO-708, ASMO_449, BALTIC,
 BIG-5, BIG-FIVE, BIG5-HKSCS, BIG5, BIG5HKSCS, BIGFIVE, BRF, BS_4730, CA, CN-BIG5, CN-GB, CN,
 CP-AR, CP-GR, CP-HU, CP037, CP038, CP273,
```

`grep` es un comando que procesa texto línea por línea buscando patrones (expresiones regulares) y mostrando solamente las líneas que cumplen dichos patrones. Por ejemplo:

```
usuario@nombreMaquina:$ grep localh /etc/hosts
127.0.0.1   localhost
::1 ip6-localhost ip6-loopback
```

`jq` es un comando de terminal para procesar ficheros en formato JSON. Probablemente habrá que instalar este software aparte, puesto que no es habitual encontrarlo en una instalación estándar. Más adelante, en este mismo documento, se muestra cómo llevar a cabo esta tarea. Un programa con Jq contiene un filtro que, a partir de una entrada, produce una salida.

Expresiones regulares

El aprendizaje de expresiones regulares se escapa del alcance de este documento, aunque se recomienda al alumnado su aprendizaje.

Imaginamos que disponemos del ejemplo siguiente:

```
usuario@nombreMaquina:$ more f.json
["Archivo Fotogr\uno00e0fico de Barcelona",["Archivo Fotogr\uno00e0fico de Barcelona"],
```

```
["El Archivo Fotogr\uno00e0fico de Barcelona (AFB) \uno00e9s el archivo dedicado a la
recogida, conservaci\uno00f3, organizaci\uno00f3 y difusi\uno00f3 de los fondos
fotogr\uno00e0ficos de car\uno00e0cter hist\uno00f2rico generados por el Ayuntamiento de
Barcelona a consecuencia de su actividad, as\uno00ed como de aquellos fondos y colecciones
fotogr\uno00e0metes, de car\uno00e0cter no municipal, de inter\uno00e8s para la
hist\uno00f2ría de la ciudad y para la hist\uno00f2ría de la fotografía."],
["https://es.wikipedia.org/wiki/Archivo_FotogrC3%A1fico_de_Barcelona"]]>
```

Con el comando `jq` podemos obtener campos concretos de este fichero en formato JSON. Por ejemplo:

```
usuario@nombreMaquina:$ jq .[2] f.json
[
El Archivo Fotográfico de Barcelona (AFB) es el archivo dedicado a la recogida, conservación,
organización y difusión de los fondos fotográficos de carácter histórico generados por el
Ayuntamiento de Barcelona a consecuencia de su actividad, así como de aquellos fondos y
colecciones fotográficas, de carácter no municipal, de interés para la historia de la ciudad
y para la historia de la fotografía.
]
usuario@nombreMaquina:$ jq .[3] f.json
[
 "https://es.wikipedia.org/wiki/Archivo_Fotogr%C3%A1fico_de_Barcelona"
]
```

`xmlstarlet` es un conjunto de comandos de terminal que se pueden usar para transformar, seleccionar, validar o editar ficheros XML.[7] Se trata de herramientas que pueden hacer algo similar al comando `grep`, pero en ficheros con formato XML. Por medio del comando siguiente, se mostrarían los nombres de estos *xpaths* del fichero «museos.xml».

[7] Del inglés *Extensible Markup Language*.

```
usuario@nombreMaquina:$ xmlstarlet sel -t -v "/xml/datosMuseos/Equipamiento" museos.xml
```

Aquí se ha ofrecido una descripción muy breve de los últimos tres comandos (`grep`, `jq` y `xmlstarlet`), pero profundizar más está fuera del alcance del cometido de este documento de introducción a GNU/Linux. Se han mencionado solamente como herramientas disponibles, aunque el aprendizaje para su uso es complicado y se deja para otros espacios de documentación.

8. Compresión de ficheros

En sistemas informáticos, es habitual comprimir un conjunto de ficheros, ya sea para almacenarlos en copias de seguridad, traspasarlos por red u otras muchas acciones. En entornos GNU/Linux, es habitual que varios ficheros se empaqueten en uno solo primero y, después, se compriman.

Windows

En sistemas Windows, normalmente se hace solo una operación: se genera un solo fichero «.zip» a partir de ficheros diversos y ya está.

El comando empleado para empaquetar o desempaquetar ficheros será la utilidad `tar`. Para comprimir y descomprimir existen diversos comandos, aunque los más habituales son los siguientes: `gzip`, `bzip2`, `zip` y `unzip`.

A continuación mostramos algunos ejemplos. En el primer comando, se empaqueta todo el contenido de la carpeta «Documentos» en un fichero «docs.tar». El parámetro `-cf` hay que leerlo como «crear fichero». Después, se comprime en formato zip:

```
usuario@nombreMaquina:$ tar -cf docs.tar Documentos/
usuario@nombreMaquina:$ ls docs.tar
docs.tar
usuario@nombreMaquina:$ gzip docs.tar
usuario@nombreMaquina:$ ls *.tar.gz
docs.tar.gz
```

Ahora hacemos el proceso contrario: a partir de un fichero «.tar.gz» lo descomprimimos. El parámetro `-d` en `gzip` es para descomprimir. El parámetro `-xvf` sirve para extraer un fichero.

```
usuario@nombreMaquina:$ gzip -d docs.tar.gz
usuario@nombreMaquina:$ tar -xvf docs.tar
Documentos/
```

Existe la posibilidad de efectuar ambas operaciones de una manera muy elegante empleando un *pipe*:

```
usuario@nombreMaquina:$ gzip -dc docs.tar.gz | tar -xv
```

9. Variables de entorno

En los sistemas GNU/Linux hay **variables de entorno** para almacenar datos y opciones de configuración que dependen del usuario.

Para conocer el listado de variables de entorno disponibles, podemos ejecutar los comandos `env` o `printenv`. En el ejemplo siguiente, hemos truncado la información para facilitar la lectura.

```
usuario@nombreMaquina:$ env
LANG=es_ES.UTF-8
DISPLAY=:1
GNOME_SHELL_SESSION_MODE=ubuntu
GTK2_MODULES=overlay-scrollbar
COLORTERM=truecolor
USERNAME=usuario
XDG_VTNR=2
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
MANDATORY_PATH=/usr/share/gconf/ubuntu.mandatory.path
XDG_SESSION_ID=4
USER=usuario
DESKTOP_SESSION=ubuntu
QT4_IM_MODULE=xim
TEXTDOMAINDIR=/usr/share/locale/
GNOME_TERMINAL_SCREEN=/org/gnome/Terminal/screen/7e2d77d9_3137_4a30_8957_2f5409826d1e
DEFAULTS_PATH=/usr/share/gconf/ubuntu.default.path
QT_QPA_PLATFORMTHEME=appmenu-qt5
PWD=/home/usuario
HOME=/home/usuario
TEXTDOMAIN=im-config
SSH_AGENT_PID=3019
QT_ACCESSIBILITY=1
XDG_SESSION_TYPE=x11
XDG_DATA_DIRS=/usr/share/ubuntu:/usr/local/share:/usr/share:/var/lib/snapd/desktop
XDG_SESSION_DESKTOP=ubuntu
GJS_DEBUG_OUTPUT=stderr
GTK_MODULES=gail:atk-bridge
WINDOWPATH=2
TERM=xterm-256color
SHELL=/bin/bash
VTE_VERSION=5202
QT_IM_MODULE=ibus
XMODIFIERS=@im=ibus
IM_CONFIG_PHASE=2
XDG_CURRENT_DESKTOP=ubuntu:GNOME
GPG_AGENT_INFO=/run/user/1000/gnupg/S.gpg-agent:0:1
GNOME_TERMINAL_SERVICE=:1.104
XDG_SEAT=seat0
SHLVL=1
GDMSESSION=ubuntu
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
LOGNAME= usuari
PATH=/home/usuario/bin:/home/usuario/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/
bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
GJS_DEBUG_TOPICS=JS ERROR;JS LOG
SESSION_MANAGER=local/Zeus:@/tmp/.ICE-unix/2911,unix/Zeus:/tmp/.ICE-unix/2911
LESSOPEN=| /usr/bin/lesspipe %s
GTK_IM_MODULE=ibus
_=/usr/bin/env
```

Entre otras, las cuatro variables de entorno que consideramos de más interés son las siguientes:

Variables de entorno	Descripción
HOSTNAME	Contiene el nombre del ordenador.
HOME	Contiene el directorio del usuario.
PATH	Contiene los directorios que se recorrerán para la búsqueda de los comandos.
EDITOR	Contiene el nombre del editor de texto por defecto.

Es posible ver el valor de estas variables de entorno empleando el comando `echo` y agregando el carácter «$» al nombre de la variable. Por ejemplo:

```
usuario@nombreMaquina:$ echo $HOME
/home/usuario

usuario@nombreMaquina:$ echo $PATH
/home/usuario/bin:/home/usuario/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/
sbin:/bin:/usr/games:/usr/local/games:/snap/bin
```

A menudo puede ser necesario cambiar o agregar información a estas variables de entorno. Por ejemplo, en el caso de que deseáramos agregar una nueva carpeta a la variable de entorno **PATH**. Para realizar esta operación habrá que emplear el comando `export`:

```
usuario@nombreMaquina:$ export PATH=$PATH:/home/usuario/nuevacarpeta
```

Este comando configura la variable de entorno **PATH** con la información que ya contenía y también agrega esta nueva carpeta «/home/usuario/nuevacarpeta».

De manera similar, podemos crear nuevas variables:

```
usuario@nombreMaquina:$ export VARIABLE="hola !"
usuario@nombreMaquina:$ echo $VARIABLE
hola !
```

En el supuesto de que deseáramos que estas nuevas variables fuesen persistentes (que no se pierdan cuando se reinicie el ordenador), hará falta que las almacenamos en ficheros. En concreto, podemos agregar los comandos al final del fichero «$HOME/.bashrc»:

```
usuario@nombreMaquina:$ cd $HOME
usuario@nombreMaquina:$ tail -2 .bashrc
export VARNOVA=" adios !"
usuario@nombreMaquina:$ echo $VARNUEVA
adios !
```

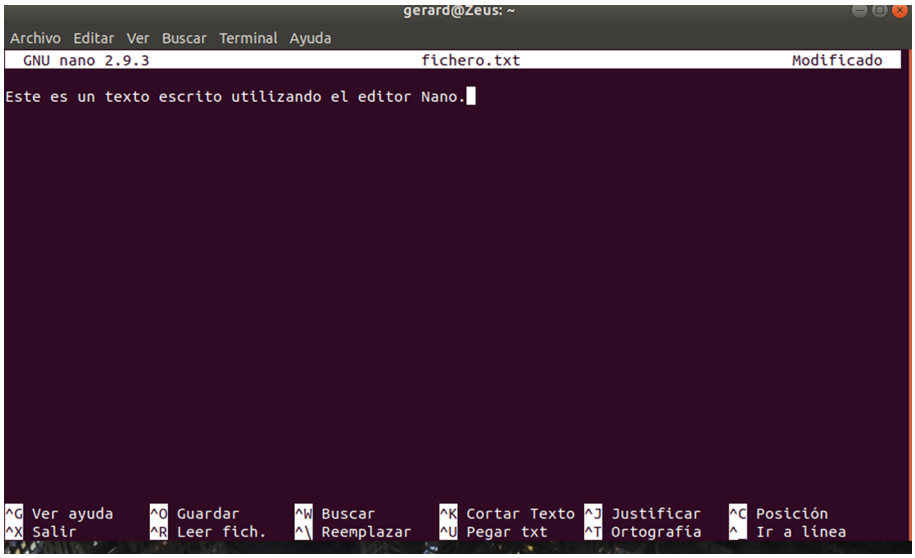
10. Editores de texto

Es importante saber crear ficheros de texto por medio del terminal. Existen varios editores. Uno de los más antiguos y que se mantiene en la mayoría de terminales es el **Vi**, aunque se desaconseja en el marco de esta asignatura. Recomendamos, por su simplicidad, la utilidad **Nano**. Así, pues, para la edición de un fichero de texto hará falta solamente ejecutar el comando:

```
usuario@nombreMaquina:$ nano fichero.txt
```

Una vez dentro del editor de texto, se observará que los comandos de control se realizan de manera conjunta con las teclas «control» + una tecla concreta. Por ejemplo, para guardar el fichero se hace con «control + o» y, para salir, «control + x». En este caso concreto, la distinción entre minúsculas y mayúsculas no es importante. Excepto para estas combinaciones de teclas, para el resto se escribe normalmente, tal como se haría con otros editores de texto.

Figura 4. Captura de pantalla en la que se observa la interfaz del editor de textos Nano



11. Procesos con GNU/Linux

Cuando el usuario ejecuta un software se crea un **proceso**.

Los sistemas operativos actuales son capaces de ejecutar múltiples procesos a la vez (con técnicas diversas de planificación de la CPU). Pero aquí la idea principal es que cada programa es un proceso que se identifica con un número (el PID –identificador del proceso). Ejemplificaremos este apartado con comandos.

El comando `sleep` ejecuta un proceso que no hace nada durante los segundos indicados, en este caso 10 segundos. Observamos que mientras se ejecuta este proceso, el usuario no puede hacer ninguna otra operación hasta que no finalice.

```
usuario@nombreMaquina:$ sleep 10
```

Se trata de una ejecución en primer plano. Es posible ejecutar el mismo proceso pero «de fondo» (lo que técnicamente se denomina en *background*):

```
usuario@nombreMaquina:$ sleep 10 &
[1] 9244
```

Cuando se ejecuta un proceso con esta modalidad, se mostrará el identificador del proceso que se le habrá asignado, en este último caso, el 9244. Es posible ver los procesos que están en marcha en una máquina con los comandos `top` y `ps`:

```
usuario@nombreMaquina:$ ps
 PID TTY TIME CMD
 6306 pts/0 00:00:00 bash
 7671 pts/0 00:00:00 ps
usuario@nombreMaquina:$ sleep 10 &
[1] 7672
usuario@nombreMaquina:$ ps
 PID TTY TIME CMD
 6306 pts/0 00:00:00 bash
 7672 pts/0 00:00:00 sleep
 7675 pts/0 00:00:00 ps
```

El comando `ps` puede mostrar información diversa en función de los parámetros que se le asignen. Por ejemplo:

Parámetro	Información
UID	Identificador de usuario al que pertenece el proceso (o sea, la persona que lo está ejecutando).
PID	Identificador del proceso.

Parámetro	Información
PPID	Identificador del proceso padre (en el caso de que otro proceso lo haya empezado).
C	Grado de utilización de la CPU por parte del proceso.
STIME	Fecha de inicio del proceso.
TTY	Tipo de terminal asociado al proceso.
TIME	Tiempo de CPU empleada por el proceso.
CMD	Comando que se ha empleado para iniciar el proceso.

El comando `ps` cuenta con varios parámetros disponibles. Entre otros:

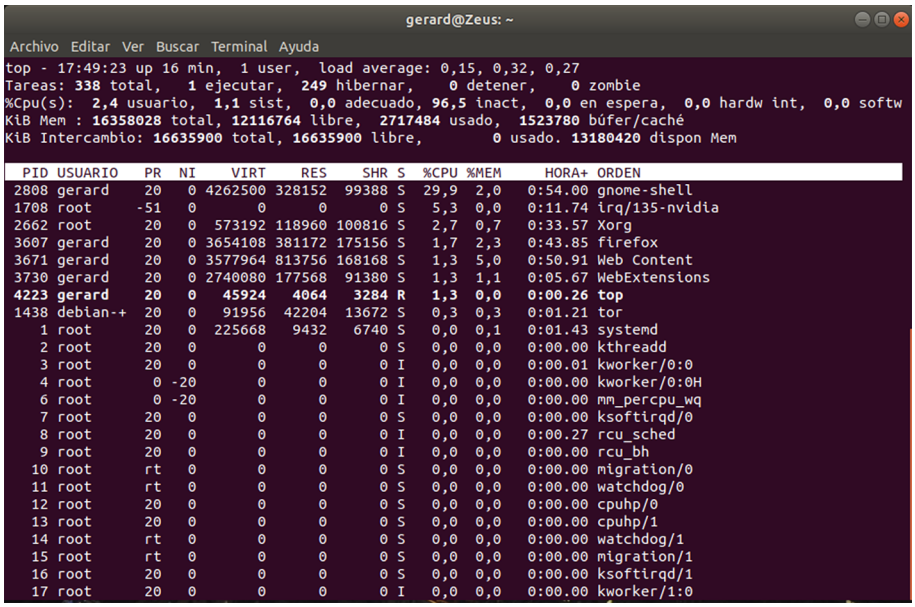
- **a:** muestra información de todos los usuarios.
- **x**: muestra información de los procesos sin los terminales.
- **u**: muestra información adicional.

El comando `top` es una herramienta de administración del sistema útil para conocer la actividad del procesador, saber qué procesos están en marcha y cuántos recursos ocupan. Para ejecutarlo hace falta simplemente escribir en un terminal:

```
usuario@nombreMaquina:$ top
```

En la pantalla de la figura 5 se observa el listado de procesos actuales, qué tanto por ciento de memoria RAM ocupa cada uno, el grado de utilización de la CPU, el comando en concreto que ha generado el proceso y la prioridad con la que se ejecuta, entre otros. Para salir del comando hay que pulsar la tecla «q».

Figura 5. Captura de pantalla en la que se observa el resultado de la ejecución del comando `top`



También es posible filtrar qué procesos observar, por ejemplo, los de un usuario en concreto:

```
usuario@nombreMaquina:$ top -u usuario
```

E, incluso, generar un fichero de texto plano con la información que genera este comando:

```
usuario@nombreMaquina:$ top -n 1 -b > salida-top.txt
```

A los procesos se les puede enviar señales diversas, por ejemplo, señales para que se paren. Cuando se ejecuta un proceso en primer plano, con la combinación de las teclas «control + c» se envía una señal para que el proceso pare. Cuando el proceso se ejecuta en segundo plano (en *background*), en cambio, habrá que emplear el comando `kill`. Veamos unos ejemplos:

```
usuario@nombreMaquina:$ sleep 10 &
[1] 7734
```

Este 7734 se refiere al proceso de identificador del proceso (el PID) y, de hecho, lo podemos ver ejecutando un *process status*:

```
usuario@nombreMaquina:$ ps
 PID TTY TIME CMD
 6306 pts/0 00:00:00 bash
 7734 pts/0 00:00:00 sleep
 7735 pts/0 00:00:00 ps
```

A continuación, enviaremos una señal con el comando `kill` al proceso en concreto para pedirle que se pare:

```
usuario@nombreMaquina:$ kill -TERM 7734
usuario@nombreMaquina:$ ps
 PID TTY TIME CMD
 6306 pts/0 00:00:00 bash
 7736 pts/0 00:00:00 ps
[1]+ Terminado sleep 10
```

A veces hay procesos que, por el motivo que sea, no se paran. Hay otras señales menos amigables. En concreto, esta de `KILL`:

```
usuari@nommaquina:$ kill -KILL 7734
```

Hay diversas señales que podemos enviar a los procesos, aunque las más empleadas habitualmente son las dos que hemos comentado.

Es posible ver el listado de señales disponibles usando el comando siguiente:

```
usuario@nombreMaquina:$ kill -l
 1) SIGHUP     2) SIGINT     3) SIGQUIT    4) SIGILL     5) SIGTRAP
 6) SIGABRT    7) SIGBUS     8) SIGFPE     9) SIGKILL   10) SIGUSR1
11) SIGSEGV   12) SIGUSR2   13) SIGPIPE   14) SIGALRM   15) SIGTERM
16) SIGSTKFLT 17) SIGCHLD   18) SIGCONT   19) SIGSTOP   20) SIGTSTP
21) SIGTTIN   22) SIGTTOU   23) SIGURG    24) SIGXCPU   25) SIGXFSZ
26) SIGVTALRM 27) SIGPROF   28) SIGWINCH  29) SIGIO     30) SIGPWR
31) SIGSYS    34) SIGRTMIN  35) SIGRTMIN+1   36) SIGRTMIN+2   37) SIGRTMIN+3
38) SIGRTMIN+4   39) SIGRTMIN+5   40) SIGRTMIN+6   41) SIGRTMIN+7   42) SIGRTMIN+8
43) SIGRTMIN+9   44) SIGRTMIN+10  45) SIGRTMIN+11  46) SIGRTMIN+12  47) SIGRTMIN+13
```

```
48) SIGRTMIN+14  49) SIGRTMIN+15  50) SIGRTMAX-14  51) SIGRTMAX-13  52) SIGRTMAX-12
53) SIGRTMAX-11  54) SIGRTMAX-10  55) SIGRTMAX-9   56) SIGRTMAX-8   57) SIGRTMAX-7
58) SIGRTMAX-6   59) SIGRTMAX-5   60) SIGRTMAX-4   61) SIGRTMAX-3   62) SIGRTMAX-2
63) SIGRTMAX-1   64) SIGRTMAX
```

El comando `killall` permite matar los procesos mediante su nombre, sin tener que conocer su identificador:

```
usuario@nombreMaquina:$ sleep 10 &
[1] 7790
usuario@nombreMaquina:$ killall sleep
[1]+ Terminado sleep 10
```

También está el comando `nohup`, que permite ejecutar otro comando instruyendo al sistema para que lo continúe ejecutando incluso cuando se desconecte la sesión. Este «nohup» proviene de «no hangup», que se refiere a una señal de sistema que mata los procesos cuando el usuario se desconecta. Este comando permite la ejecución en segundo plano de comandos que pueden requerir mucho tiempo. Por ejemplo:

```
usuario@nombreMaquina:$ nohup ping -i 10 www.uoc.edu
```

Podemos emplear el comando `time` para determinar cuánto tiempo requiere un comando para su ejecución. Puede ser útil para testar el rendimiento de un guion de secuencias o de un comando concreto. Imaginad que disponéis de dos *scripts* diferentes y deseáis saber cuál de los dos es más rápido. Esto se podría saber con el comando `time`. El resultado de este comando puede variar en función de la *shell* utilizada. No obstante, con Bash, aparecen los resultados siguientes:

- `real`: es el tiempo desde que ha empezado el proceso hasta que ha finalizado.
- `user`: es el tiempo que el proceso ha estado en espacio de modo de usuario.
- `system`: es el tiempo que el proceso ha estado en el espacio del *kernel*.

A continuación mostramos un ejemplo sencillo:

```
usuari@nomMaquina:$ time ping -c 1 www.google.es
PING www.google.es(ocsp.pki.goog (2a00:1450:4003:80b::2003)) 56 data bytes
64 bytes from ocsp.pki.goog (2a00:1450:4003:80b::2003): icmp_seq=1 ttl=53 time=43.7 ms
--- www.google.es ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 43.749/43.749/43.749/0.000 ms
real    0m0,051s
user    0m0,005s
sys     0m0,000s
```

El último comando que mencionamos en este apartado es `nice`, que permite cambiar la prioridad con la que se ejecutará un proceso. Con el número `-20` se indica máxima prioridad (hecho que significa que la CPU priorizará la ejecución de este proceso en vez de otros procesos existentes), hasta el rango del número 19 (que indica prioridad mínima). Por ejemplo:

```
usuario@nombreMaquina:$ nice -n 13 nano h.txt
```

12. Ejecución de procesos automatizados con Cron

Cron proviene del término griego *chronos*, que significa «tiempo». En sistemas operativos de tipo Unix, Cron es un programa que ejecuta procesos o guiones de secuencia en segundo plano (esto significa, de manera automática, sin intervención del usuario). Este proceso está instalado en todos los sistemas operativos GNU/Linux y se configura por medio del fichero «crontab», ubicado habitualmente en «/etc/crontab».

Este fichero contiene unas cuantas líneas y cada línea especifica una tarea que se va a ejecutar. Los campos concretos son los siguientes:

```
# m h dom mon dow user command
```

donde:

- `m` corresponde al minuto en el que se ejecutará el *script*, con un valor entre 0 y 59.
- `h` es la hora exacta, con un valor entre 0 y 23.
- `dom` hace referencia al día del mes (*day of month*).
- `dow` hace referencia al día de la semana (*day of week*). Se indica con un valor numérico (donde el 0 será el domingo, 1 lunes y así sucesivamente) o con las tres primeras letras del nombre en inglés: mon, tue, wed, thu, fri, sat, sun.
- `user` se refiere al usuario que ejecutará el proceso o el *script*.
- `command` se refiere al comando que se desea ejecutar. Se aconseja indicar aquí la ruta completa. Por ejemplo: «/home/usuario/scripts/descarga.sh».

No es obligatorio indicar todos los campos. Algunos de estos se pueden sustituir con un asterisco. Algunos ejemplos:

```
00 22 * * * usuario /home/usuario/scripts/descarga.sh
```

El usuario «usuario» ejecutará todos los días a las 22:00 h este *script* «descarga.sh» ubicado en la carpeta señalada (concretamente «/home/usuario/scripts/»).

O este otro ejemplo:

```
30 3 * * sun root apt-get -y update
```

Donde todos los domingos, a las 3:30 h de la noche, el usuario *root* ejecutará este comando `apt-get -y update`.

13. Red, conectividad y clientes web

En este apartado indicamos algunos comandos de interés para la red y la conectividad a internet. Hay quien puede considerar que algunos de estos comandos pueden estar fuera del alcance de esta asignatura, pero se agregan aquí con el único objetivo de que el usuario disponga de los conocimientos adecuados para poder realizar test sencillos sobre la conectividad en la red y en internet. Después de estos comandos más técnicos, se mencionan dos comandos mucho más útiles para descargar ficheros vía web, por ejemplo.

Los comandos que hemos decidido incorporar a esta sección son los siguientes:

`ip address` es el comando que mostrará información sobre las direcciones de red asignadas a las diferentes interfaces del ordenador.

```
usuario@nombreMaquina:$ ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
 link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
 inet 127.0.0.1/8 scope host lo
 valid_lft forever preferred_lft forever
 inet6 ::1/128 scope host
 valid_lft forever preferred_lft forever
2: enp0s31f6: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel state DOWN group
default qlen 1000
 link/ether b0:6e:bf:61:9c:18 brd ff:ff:ff:ff:ff:ff
3: wlp4s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default
qlen 1000
 link/ether 1c:87:2c:b8:3d:b5 brd ff:ff:ff:ff:ff:ff
 inet 192.168.1.106/24 brd 192.168.1.255 scope global noprefixroute wlp4s0
 valid_lft forever preferred_lft forever
 inet6 2a02:2e02:9e3c:8700:7c1c:f871:6d61:345a/64 scope global temporary dynamic
 valid_lft 880sec preferred_lft 280sec
 inet6 2a02:2e02:9e3c:8700:1d2e:ea31:a1:c293/64 scope global dynamic mngtmpaddr noprefixroute
 valid_lft 880sec preferred_lft 280sec
 inet6 fe80::dcfb:a25a:71c5:d60f/64 scope link noprefixroute
 valid_lft forever preferred_lft forever
4: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
 link/ether 02:42:88:28:f6:09 brd ff:ff:ff:ff:ff:ff
 inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
 valid_lft forever preferred_lft forever
```

En este caso concreto, se observan cuatro interfaces de red (la primera, la de *loopback*, que siempre dispone de la dirección 127.0.0.1, la segunda, que en este caso no dispone de dirección de red, la tercera, que es un enlace vía wifi y que dispone de la dirección 192.168.1.106 y, finalmente, una cuarta virtual).

Con el comando `ip route` es posible conocer cuáles son las máquinas que actúan de puerta de enlace de nuestra red.

```
usuario@nombreMaquina:$ ip route
default via 192.168.1.1 dev wlp4s0 proto dhcp metric 600
169.254.0.0/16 dev docker0 scope link metric 1000 linkdown
172.17.0.0/16 dev docker0 proto kernel scope link src 172.17.0.1 linkdown
192.168.1.0/24 dev wlp4s0 proto kernel scope link src 192.168.1.106 metric 600
En este caso concreto, la máquina que actúa de puerta de enlace dispone de la dirección de red
```

```
192.168.1.1.
```

`ping` es el mejor comando para testar la conectividad entre varios nodos de una red informática, tanto si se trata de una red de alcance local (LAN) como de una de alcance global (WAN). `ping` utiliza paquetes del tipo ICMP[8] para comunicarse con otros dispositivos. En concreto, se trata de paquetes ICMP Echo Reply. Veamos a continuación un par de ejemplos.

(8)Del inglés *Internet Control Message Protocol.*

```
usuario@nombreMaquina:$ ping www.google.es
PING www.google.es(www.google.es (2a00:1450:4003:801::2003)) 56 data bytes
64 bytes from www.google.es (2a00:1450:4003:801::2003): icmp_seq=1 ttl=53 time=44.7 ms
64 bytes from www.google.es (2a00:1450:4003:801::2003): icmp_seq=2 ttl=53 time=44.6 ms
64 bytes from www.google.es (2a00:1450:4003:801::2003): icmp_seq=3 ttl=53 time=45.5 ms
64 bytes from www.google.es (2a00:1450:4003:801::2003): icmp_seq=4 ttl=53 time=44.3 ms
^C

--- www.google.es ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 44.377/44.846/45.567/0.464 ms
```

Observamos que este *ping* no se detiene. Para hacerlo, habrá que enviar al proceso una señal **TERM** para que lo haga empleando la combinación de teclas «control + c».

Con el parámetro `-c` podemos indicar el número exacto de paquetes ICMP que deseamos enviar:

```
usuario@nombreMaquina:$ ping -c 1 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=1.72 ms
--- 192.168.1.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.726/1.726/1.726/0.000 ms
```

Con el comando `traceroute` el usuario puede saber por cuántas (y por cuáles) máquinas viajan sus peticiones. Por ejemplo, desde el ordenador donde escribimos este texto hasta el servidor web de la UOC (www.uoc.edu) hay ocho máquinas:

```
usuario@nombreMaquina:$ traceroute www.uoc.edu
traceroute to www.uoc.edu (213.73.40.242), 30 hops max, 60 byte packets
 1 Livebox (192.168.1.1) 1.739 ms 2.945 ms 4.075 ms
 2 192.0.0.1 (192.0.0.1) 21.776 ms 22.141 ms 23.234 ms
 3 10.34.35.134 (10.34.35.134) 26.595 ms 26.761 ms 27.691 ms
 4 10.34.35.206 (10.34.35.206) 38.789 ms 39.234 ms 39.435 ms
 5 anella01.01.catnix.net (193.242.98.38) 40.333 ms 41.607 ms 42.500 ms
 6 in3-anella.cesca.cat (84.88.18.42) 45.243 ms 34.008 ms 35.061 ms
 7 * * *
 8 www.uoc.edu (213.73.40.242) 27.121 ms 25.073 ms 25.665 ms
```

`wget` es una utilidad de terminal que permite descargar ficheros de servidores web remotos, empleando protocolos como HTTP, HTTPS o también FTP. `wget` también puede ser útil para descargar ficheros de manera automática dentro de los *scripts*. Con el comando siguiente descargaremos un fichero concreto del servidor web especificado:

```
usuario@nombreMaquina:$ wget http://tldp.org/Linux-HOWTO-text.tar.gz
--2019-10-01 06:42:38-- http://tldp.org/Linux-HOWTO-text.tar.gz
Se está resolviendo tldp.org (tldp.org)… 152.19.134.41
```

```
Se está conectando a tldp.org (tldp.org)|152.19.134.41|:80… conectado.
HTTP: se ha enviado la petición, se está esperando una respuesta… 200 OK
Medida: 9048579 (8,6M) [application/x-gzip]
Se está guardando en: «Linux-HOWTO-text.tar.gz»
Linux-HOWTO-text.ta 100%[==================>] 8,63M 2,56MB/s in 3,4s
2019-10-01 06:42:42 (2,56 MB/s) - se ha guardado «Linux-HOWTO-text.tar.gz» [9048579/9048579]
```

A veces sucede que deseamos descargar ficheros de servidores web con HTTPS y el cliente desconoce el certificado digital. Por este motivo, se recomienda agregar el parámetro `--no-check-certificate`.

```
usuario@nombreMaquina:$ wget https://tldp.org/Linux-HOWTO-text.tar.gz
--2019-10-01 06:43:22-- https://tldp.org/Linux-HOWTO-text.tar.gz
Se está resolviendo tldp.org (tldp.org)… 152.19.134.41
Se está conectando a tldp.org (tldp.org)|152.19.134.41|:443… conectado.
ERROR: ningún nombre común alternativo del certificado concuerda con el nombre del servidor
comando «tldp.org».Para conectar a tldp.org de manera insegura, usad «--no-check-certificate».
```

En cambio:

```
usuario@nombreMaquina:$ wget --no-check-certificate https://tldp.org/Linux-HOWTO-text.tar.gz
--2019-10-01 06:44:01-- https://tldp.org/Linux-HOWTO-text.tar.gz
Se está resolviendo tldp.org (tldp.org)… 152.19.134.41
Se está conectando a tldp.org (tldp.org)|152.19.134.41|:443… conectado.
AVISO: ningún nombre común alternativo del certificado concuerda con el nombre del servidor
pedido «tldp.org».
HTTP: se ha enviado la petición, se está esperando una respuesta… 200 OK
Medida: 9048579 (8,6M) [application/x-gzip]
Se está guardando en: «Linux-HOWTO-text.tar.gz»
Linux-HOWTO-text.ta 100%[==================>] 8,63M 4,05MB/s in 2,1s
2019-10-01 06:44:03 (4,05 MB/s) - se ha desat «Linux-HOWTO-text.tar.gz» [9048579/9048579]
```

`curl` es un comando similar al anterior, con ligeras diferencias, que emplearemos de manera indistinta con el anterior. Por ejemplo:

```
usuario@nombreMaquina:$ curl -o Linux-HOWTO-text.tar.gz http://tldp.org/Linux-HOWTO-text.tar.gz
% Total % Received % Xferd Average Speed Time Time Time Current Dload Upload Total Spent Left
Speed 100 8836k 100 8836k 0 0 3397k 0 0:00:02 0:00:02 --:--:-- 3397k
```

donde el parámetro `-o` indica el nombre del fichero de salida.

14. Creación de guiones de secuencias (*scripts*)

Es posible elaborar sistemas que permitan la ejecución de los comandos anteriores en bloque, de manera continuada, con la ejecución de uno detrás del otro y sin que el usuario tenga que intervenir. Estos programas son los ***scripts***, los **guiones de secuencias**. Se trata de ficheros de texto que contienen las instrucciones. Elaborar estos *scripts* puede ser útil para la ejecución de tareas que se realizan de manera repetitiva o que pueden ser útiles para la automatización de tareas diversas (por ejemplo, la creación de copias de seguridad nocturnas, la descarga de grandes ficheros de internet, entre otras, todo de manera automática).

Para la elaboración de *scripts*, habrá que emplear un editor de texto del tipo que hemos mencionado antes y escribir algo similar a lo siguiente:

```
#!/bin/bash
echo "Este es mi primer script. Mostrará por pantalla el fichero de passwd"
cat /etc/passwd
```

La primera línea indica cuál tiene que ser el programa que interprete las instrucciones que seguirán a continuación (en el caso que nos ocupa será «/bin/bash»). Antes de poder ejecutar este *script*, habrá que otorgarle el permiso de ejecución:

```
usuario@nombreMaquina:$ chmod +x script.sh
```

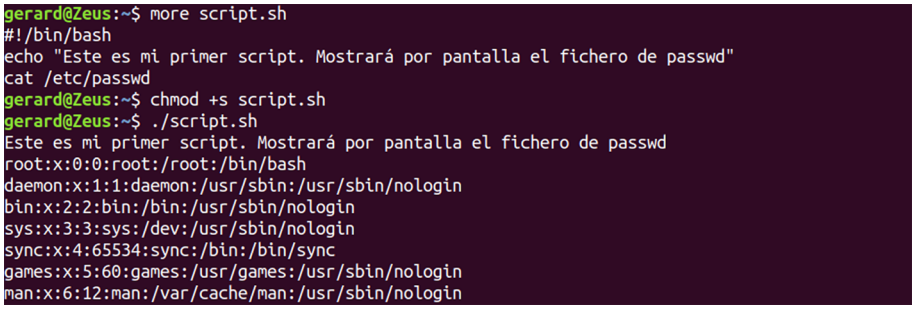
Ahora, ya será posible ejecutarlo con:

```
usuario@nombreMaquina:$ ./script.sh
```

El `./` inicial indica dónde está el fichero en concreto. En este caso, en la misma carpeta donde está el usuario. También sería posible ejecutarlo indicando la ruta completa de ubicación del fichero. Por ejemplo:

```
/home/usuario/script.sh
```

Figura 6. Captura de pantalla en la que se observa el proceso de ejecución de este `script.sh`



De manera similar, se pueden ejecutar guiones escritos con otros lenguajes de programación. Por ejemplo, podríamos ejecutar *scripts* con Python. La primera línea del *script* contiene el intérprete que será el encargado de ejecutar los comandos que vendrán a continuación. Veamos el caso siguiente:

```
usuario@nombreMaquina:$ more scriptenpython.sh
#!/usr/bin/env python3
print('¡Hola a todo el mundo desde Python!')

usuario@nombreMaquina:$ chmod +x scriptenpython.sh

usuario@nombreMaquina:$ ./scriptenpython.sh
¡Hola a todo el mundo desde Python!
```

15. Distribuciones de GNU/Linux

15.1. Introducción

Una **distribución de GNU/Linux** consiste en un sistema operativo generado por medio de un conjunto de software que se basa, principalmente, en el *kernel* de Linux y, normalmente, con un sistema de gestión de paquetes de software.

Habitualmente, los usuarios de GNU/Linux descargan una ISO con su distribución preferida, primero lo graban en un soporte extraíble tipo memoria flash o CDROM, y después la instalan en su hardware. Hay distribuciones para todo tipo de hardware: para dispositivos embebidos o empotrados, ordenadores personales o supercomputadores. También hay distribuciones diseñadas con recopilaciones de software específico para ciertas tareas (por ejemplo, distribuciones con aplicaciones educativas, científicas, de seguridad informática, entre otras).

Una distribución típica de GNU/Linux contiene, tal como hemos comentado, un *kernel*, herramientas y librerías GNU, software, documentación y un entorno gráfico (aunque es algo opcional). La mayoría del software que se incluye es libre y está disponible por medio del código fuente o compilado con ficheros binarios ejecutables. Algunas distribuciones incluyen de manera opcional software propietario, sin disponer de código fuente, normalmente para algunos controladores de dispositivos (lo que técnicamente se denomina *drivers*).

Normalmente, los responsables de la distribución se encargan de adaptar el software, ponerlo en paquetes y distribuirlo en línea mediante lo que se llama **repositorios** (servidores con software) distribuidos en todo el mundo.

Actualmente hay centenares de distribuciones diferentes. Algunas de ellas están mantenidas con **soporte comercial**, como, por ejemplo, Fedora (Red Hat), openSUSE (SUSE) o Ubuntu (Canonical Ltd.) y otras están completamente mantenidas por la **comunidad**, como, por ejemplo Slackware o Debian.

15.2. La distribución escogida: Ubuntu

Ubuntu es una distribución de GNU/Linux basada en otra más antigua que se denomina Debian. Se trata de una distribución gratuita y de software libre con diferentes versiones: una para entornos de escritorio (*Desktop*), otra para entornos de servidor (*Server*) y otra para dispositivos IoT (*Internet of Things*)

Figura 7. Logotipo de Ubuntu

y robots (*Core*). Se libera una versión de Ubuntu cada seis meses, con una versión con soporte para largo plazo cada dos años. En el momento de escribir estos apuntes (verano de 2019), la última versión es la 19.04 (llamada «Disco Dingo») y la versión para largo plazo es la 18.04 (llamada «Bionic Beaver»).

Ubuntu está desarrollada por toda una comunidad de usuarios y la empresa Canonical, con un modelo de gobernanza basado en la **meritocracia**. El nombre Ubuntu proviene del término filosófico africano que Canonical tradujo como «la humanidad hacia los otros» o «yo soy quien soy en función de lo que todas las personas somos».

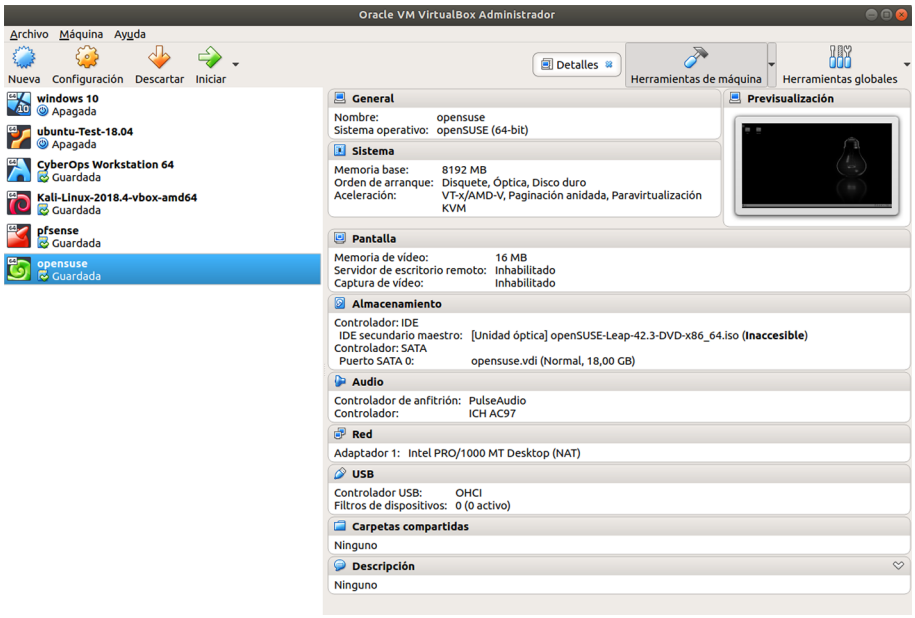
15.3. Instalación de Ubuntu

Hay diferentes maneras para disponer de una distribución Ubuntu accesible. Una de ellas consiste en descargar la imagen de instalación del sitio web oficial, grabar este fichero en un dispositivo externo tipo USB, CD-ROM o DVD e iniciar un ordenador que no disponga de sistema operativo por medio de estos medios extraíbles. El sistema iniciará un proceso de instalación en el ordenador y luego lo tendremos disponible.

Otra opción, que se aconseja para los estudiantes menos experimentados, consiste en llevar a cabo la misma operación pero en un entorno de máquinas virtuales. Una máquina virtual es un software que permite la ejecución simulada de diferentes sistemas operativos sobre otro sistema operativo ya existente. La ventaja es que no se necesita disponer de una máquina completa para la instalación del sistema y se puede mantener, por ejemplo, un sistema base con Microsoft Windows y también ejecutar un sistema Ubuntu por medio de este software de máquinas virtuales.

Hay diferentes softwares para la instalación de máquinas virtuales, por ejemplo, VirtualBox o VMware Workstation Player.

Figura 8. Captura de pantalla de la ejecución del software VirtualBox



En esta instalación hay varias máquinas virtuales (un Windows 10 y una distribución Linux llamada Kali Linux, con software especializado en seguridad informática).

Otra ventaja de este sistema de virtualización consiste en la posibilidad de poder descargar e instalar sistemas operativos ya preparados. El tipo de fichero **OVA**, por ejemplo, es un estándar abierto que consiste en un formato de virtualización que empaqueta sistemas operativos virtuales. Así, es posible descargar un fichero «.OVA» con Ubuntu, incorporarlo al software de máquinas virtuales y, rápidamente, disponer de la distribución accesible.

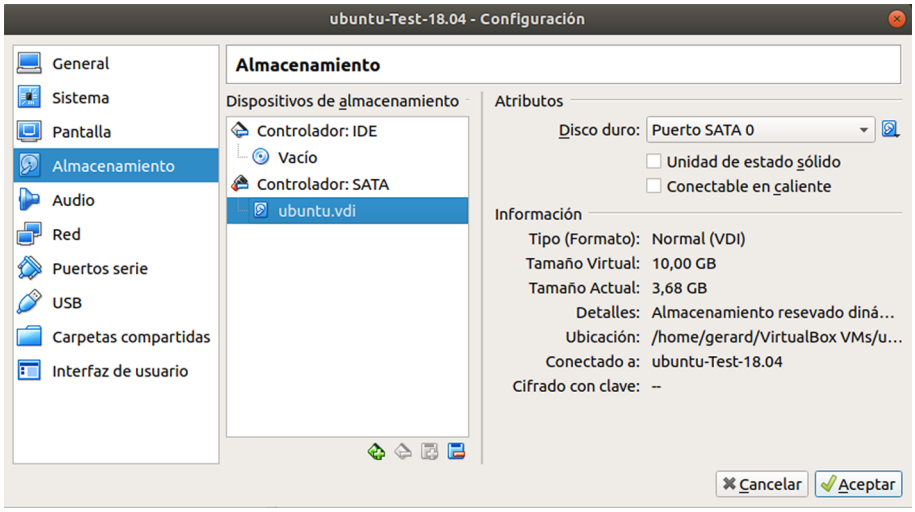
Para llevar a cabo esta operación con el software de VirtualBox, hace falta solamente ir a «Archivo» --> «Importar servicio virtualizado».

Otra posibilidad consiste en descargar una copia de un disco duro virtual (se trata de ficheros con las extensiones VDI o VMDK). Entonces, una vez descargado, hay que crear una máquina nueva en Ubuntu y configurarla para que utilice el disco duro cargado.

Ficheros VDI

En www.osboxes.org/ubuntu podéis descargar ficheros VDI para máquinas Ubuntu con el sistema operativo ya instalado.

Figura 9. Captura de la pantalla del software VirtualBox donde es posible configurar los discos duros de las máquinas virtuales



15.4. Instalación de software

Al menos hay dos vías para la instalación de software con sistemas GNU/Linux. La primera de estas consiste en descargar el código del programa, compilarlo y obtener un fichero ejecutable. Se trata de un método tradicional pero para usuarios más avanzados. Recordemos que el **proceso de compilación** consiste en convertir un código escrito por humanos en un código ejecutable por una máquina. Una ventaja de este método es que permite controlar todo el proceso y configurar el software tal como desea el usuario, pero hay que estar mucho más atento a todos los detalles.

El segundo sistema, mucho más sencillo para usuarios no tan experimentados y el recomendado en el marco de esta asignatura, utiliza los **repositorios de paquetes de software** mantenidos por la misma distribución. Para la gestión de estos paquetes, Ubuntu emplea un sistema desarrollado por Debian que se denomina *APT*.[9] Se trata de un sistema capaz de gestionar software en nuestro ordenador. Por ejemplo, descargando paquetes en servidores externos (repositorios) e instalándolos en el sistema.

[9]Del inglés *Advanced Packaging Tool*.

15.4.1. Instalación de software empleando código fuente

A continuación mostraremos un ejemplo de cómo instalar software descargando el código fuente de la red. Se trata de un sistema avanzado, puesto que, aunque permite conocer todos los detalles de la instalación, requiere dependencias de otros programas.

En este caso concreto, descargaremos y compilaremos el programa **Grep**. La primera instrucción descarga el fichero de la web (con la opción `-q` muestra menos información por pantalla). La segunda instrucción desempaqueta y descomprime el fichero. Y la tercera instrucción permite acceder al interior de la carpeta descomprimida.

```
usuario@nombreMaquina:$ wget -q https://ftp.gnu.org/gnu/grep/grep-3.3.tar.xz
usuario@nombreMaquina:$ tar -xvf grep-3.3.tar.xz
usuario@nombreMaquina:$ cd grep-3.3/
```

A continuación, se ejecuta un *script* (habitualmente se denomina *configure*) que se asegura de disponer de todas las dependencias de software en nuestro ordenador y que genera un fichero «Makefile» que contiene las instrucciones de compilación para nuestra máquina en concreto. De nuevo, añadimos la opción `-q` para obtener una salida con menos líneas.

```
usuario@nombreMaquina:/grep-3.3$ ./configure -q
config.status: creating po/POTFILES
config.status: creating po/Makefile
```

Con el «Makefile» construido, ya es posible compilar el software con el comando `make`. Una vez finalizado el proceso, el programa reside dentro de la carpeta «src» y se puede ejecutar el binario. Recordemos que, como que esta carpeta no reside en la variable de entorno **PATH**, hay que indicar la ubicación exacta del fichero empleando el «./» (que indica la carpeta actual):

```
usuario@nombreMaquina:/grep-3.3$ make
usuario@nombreMaquina:/grep-3.3$ cd src
usuario@nombreMaquina:/grep-3.3/src$ ./grep
Usage: grep [OPTION]... PATTERNS [FILE]...
Usad «grep --help» para obtener más información.
```

En caso de que se pruebe este mismo proceso en otro ordenador, puede suceder que no se disponga de todo el software requerido para la compilación del programa (compilador, *make*, librerías que requiere el programa, etc.). Por este motivo se aconseja a los estudiantes emplear este método solamente en el caso de que el software que deseamos instalar no esté en los repositorios de software mantenidos por la distribución.

15.4.2. Instalación de software empleando repositorios de Ubuntu

Recordemos que este es el método recomendado para la instalación de software, puesto que, aparte de ser mucho más sencillo, los administradores de la distribución se habrán asegurado de que todo funcione adecuadamente.

El listado de repositorios está definido en el fichero «/etc/apt/sources.list» y en la carpeta «/etc/apt/sources.list.d», aunque no es necesario que manipulemos directamente estos ficheros.

```
usuario@nombreMaquina:/etc/apt$ cat /etc/apt/sources.list | grep -v ^# | sed '/^$/de
deb http://es.archive.ubuntu.com/ubuntu/ bionic main restricted
deb http://es.archive.ubuntu.com/ubuntu/ bionic-updates main restricted
deb http://es.archive.ubuntu.com/ubuntu/ bionic universe
deb http://es.archive.ubuntu.com/ubuntu/ bionic-updates universe
deb http://es.archive.ubuntu.com/ubuntu/ bionic multiverse
deb http://es.archive.ubuntu.com/ubuntu/ bionic-updates multiverse
deb http://es.archive.ubuntu.com/ubuntu/ bionic-backports main restricted universe multiverse
deb http://security.ubuntu.com/ubuntu bionic-security main restricted
deb http://security.ubuntu.com/ubuntu bionic-security universe
deb http://security.ubuntu.com/ubuntu bionic-security multiverse
```

En el comando anterior, el primer `cat` muestra el contenido del fichero «/etc/apt/sources.list», que, empleando un *pipe*, traspasa el resultado al comando `grep`. Este comando excluye las líneas que empiecen por el carácter almohadilla (el parámetro `-v` es para excluir, el carácter «^» significa inicio de línea, seguido por una «#»). Finalmente, el último `sed`, suprime las líneas que están en blanco.

Veamos a continuación algunos comandos de `apt`:

```
usuario@nombreMaquina:$ apt-cache search nombredeprograma #Este comando buscará
en los repositorios software que contenga la palabra «nombredeprograma». Por ejemplo:

usuario@nombreMaquina:/etc/apt$ apt-cache search data science
algotutor - program for observing the intermediate steps of algorithm
beneath-a-steel-sky - classic 2D point and click science fiction adventure game
bibutils - interconvert various bibliographic data formats
cernlib - CERNLIB data analysis suite - general use metapackage
cernlib-base - CERNLIB data analysis suite - common files
cernlib-base-dev - CERNLIB data analysis suite - dependencies checking script
cernlib-core - CERNLIB data analysis suite - main libraries and programs
cernlib-core-dev - CERNLIB data analysis suite - core development files
cernlib-extras - CERNLIB data analysis suite - extra programs
cernlib-montecarlo - CERNLIB Monte Carlo libraries
cwltool - Common Workflow Language reference implementation
daligner - local alignment discovery between long nucleotide sequencing reads
datalad - data files crawler and data distribution
dict-wn - electronic lexical database of English language for dict
```

También es posible obtener información de un software en concreto:

```
usuario@nombreMaquina:$ apt-cache show jq
Package: jq
Architecture: amd64
Version: 1.5+dfsg-2
Multe-Arch: foreign
Priority: optional
Section: universe/utils
Origin: Ubuntu
Maintainer: Ubuntu Developers <ubuntu-devel-discuss@lists.ubuntu.com>
Original-Maintainer: ChangZhuo Chen <czchen@debian.org>
Bugs: https://bugs.launchpad.net/ubuntu/+filebug
Installed-Size: 88
Depends: libjq1 (= 1.5+dfsg-2), libc6 (>= 2.4)
Filename: pool/universe/j/jq/jq_1.5+dfsg-2_amd64.deb
Size: 45646
MD5sum: 9a9c411594822e8c69d82d2d7aa97766
SHA1: d03195debo de7219can7d3a9fbf712a8c92d0c2b7801
SHA256: 7da6b3fe4de02f50169923134b1e6c4cbf4ee693afa963d02d1eb8a0e3adb472
Homepage: https://github.com/stedolan/jq
Description-en: lightweight and flexible command-line JSON processor
 jq is like sed for JSON data - you can use it to slice
 and filter and map and transform structured data with
 the same ease that sed, awk, grep and friends let you
 play with text.
 .
 It is written in portable C, and it has minimal runtime
 dependencies.
 .
 jq can mangle the data format that you have into the
 one that you want with very little effort, and the
 program to do so is often shorter and simpler than
 you'd expect.
Description-md5: fd8d7d97b13012ce68c52772c1ce56aa
```

```
usuario@nombreMaquina:$ apt-get install nombredeprograma #Este comando descargará el programa
del repositorio y lo instalará en nuestro sistema. En caso de que este software tenga
```

```
dependencias, también las instalará. También podría suceder que el software ya estuviera instalado.
Así, pues, no realizaría ninguna tarea. Por ejemplo:

usuario@nombreMaquina:$ sudo apt-get install jq
[sudo] contraseña para usuario:
Se está leyendo la lista de paquetes… Hecho
Se está construyendo el árbol de dependencias
Se está leyendo la información del estado… Hecho
jq ya está en la versión más reciente (1.5+dfsg-2).
Los paquetes siguientes se han instalado automáticamente y ya no serán necesarios:
 liblog4cxx10v5 libqt5script5 libqt5scripttools5 libzzip-0-13
Emplead «sudo apt autoremove» para suprimirlos.
0 actualizados, 0 nuevos a instalar, 0 a suprimir y 0 no actualizados.
```

```
usuario@nombreMaquina:$ apt-get remove nombredelprograma #Este comando suprimirá de nuestro
ordenador el programa mencionado. Por ejemplo:

usuario@nombreMaquina:/etc/apt$ sudo apt-get remove jq
[sudo] contraseña para usuario:
Se está leyendo la lista de paquetes… Hecho
Se está construyendo el árbol de dependencias
Se está leyendo la información del estado… Hecho
Los paquetes siguientes se han instalado automáticamente y ya no serán necesarios:
 libjq1 liblog4cxx10v5 libonig4 libqt5script5 libqt5scripttools5 libzzip-0-13
Emplead «sudo apt autoremove» para suprimirlos.
Se SUPRIMIRÁN los paquetes siguientes:
 jq
0 actualizados, 0 nuevos a instalar, 1 a suprimir y 0 no actualizados.
Después de esta operación se liberarán 90,1 kB de espacio en disco.
¿Queréis continuar? [S/n] S
(Se está leyendo la base de datos… hay 291242 ficheros y directorios instalados actualmente.)
Se está desinstalando jq (1.5+dfsg-2)…
Se están procesando los activadores para man-db (2.8.3-2ubuntu0.1)…
```

```
usuario@nombreMaquina:$ apt-get update #Con este parámetro el programa apt-get actualizará
los índices con los repositorios que dispone.
```

Conviene mencionar que algunos de estos comandos realizan tareas de administración en el sistema y realizan cambios importantes. Por tanto, habrá que ejecutarlos empleando permisos de administración. Por ejemplo, con el comando `sudo`.

Bibliografía

Garrels, M. (2008). «Introduction to Linux. A Hands on Guide» [en línea]. [Fecha de consulta: 1 de octubre de 2019]. tldp.org/ldp/intro-linux/intro-linux.pdf

GNU nano (2016). «nano Command Manual» [en línea]. [Fecha de consulta: 1 de octubre de 2019]. www.nano-editor.org/dist/v2.1/nano.html

Oracle Corporation (2019). «Oracle® VM VirtualBox®. User Manual» [en línea]. [Fecha de consulta: 1 de octubre de 2019]. www.virtualbox.org/manual/

Peek, J.; Strang, J.; Todino, G. (2014). *Learning the Unix Operating System. A concise Guide for the New User*. California: O'Reilly Media.

UBUNTU Documentation (2013). «CompilingEasyHowTo» [en línea]. Ubuntu documentation. [Fecha de consulta: 1 de octubre de 2019]. help.ubuntu.com/community/CompilingEasyHowTo

UBUNTU Documentation (2016). «CronHowto» [en línea]. Ubuntu documentation. [Fecha de consulta: 1 de octubre de 2019]. help.ubuntu.com/community/cronhowto

UBUNTU Documentation (2016). «Getting Started with Ubuntu 16.04» [en línea]. [Fecha de consulta: 1 de octubre de 2019]. files.ubuntu-manual.org/manuals/getting-started-with-ubuntu/16.04/en_US/screen/Getting%20Started%20with%20Ubuntu%2016.04.pdf

Zonker (2010). «Writing A Simple Bash Script» [en línea]. Linux.com. [Fecha de consulta: 1 de octubre de 2019]. www.linux.com/tutorials/writing-simple-bash-script/.