Introducción a Linux

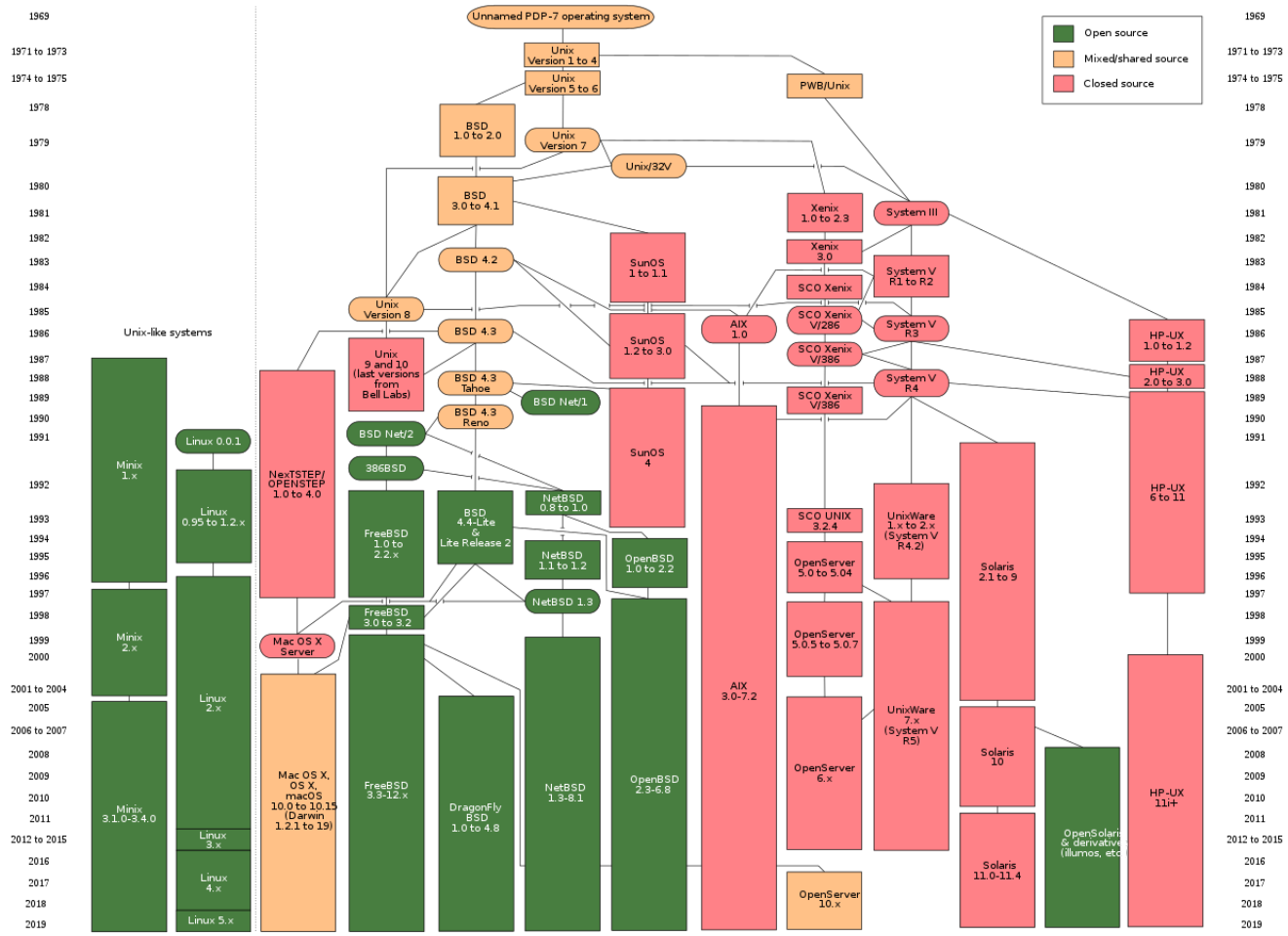
- Introducción a Linux
- 1. Introducción a Linux / Unix
    - 1.1 Unix
    - 1.2 Linux
    - 1.3 Distribuciones
    - 1.4 El escritorio
    - 1.5 La terminal de Linux/Unix
    - 1.6 Ayuda
    - 1.7 Autocompletado e historia
- 2. El sistema de ficheros
    - 2.1 Sistemas jerárquicos
    - 2.2 Rutas absolutas, relativas y directorio de trabajo
    - 2.3 Directorio $HOME
    - 2.4 Moviendo, renombrando y copiando ficheros
    - 2.5 Nombres de directorios y archivos
    - 2.6 WildCards
    - 2.6 Obteniendo información sobre archivos y directorios
    - 2.7 Permisos
    - 2.8 Obteniendo información sobre el sistema de archivos
    - 2.9 Compresion y descompresión de ficheros
    - 2.10 Enlaces duros y blandos
- 3. Procesos
    - 3.1 *ps*
    - 3.2 *kill*
    - 3.3 *free*
    - 3.4 *top*
- 4. Flujo de información
    - 4.1 Flujos de información estándar
    - 4.2 Redirección del flujo estándar
    - 4.3 Pipes
- 5. Procesamiento de ficheros de texto
    - 5.1 Editores de texto
    - 5.2 Imprimiendo el contenido de ficheros
- 6. Administrador
- 6.1 sudo
- 6.2 su
- 7. Variables de entorno
- 7.1 ¿Donde están los ejecutables? $PATH
- 7.2 HOME
- 8. Administración de software
    - 8.1 Repositorios
    - 8.2 Herramientas de manejo de paquetes
        - 8.2.1 APT

- 8.2 dpkg
- 8.3 Estándar de jerarquía del sistema de archivos

1. Introducción a Linux / Unix

1.1 Unix

Unix es una familia de sistemas operativos. La primera versión de Linux fue desarrollada a partir de 1969. Unix se caracteriza por ser portable y multitarea.



Hoy en día los sistemas operativos Unix son ampliamente utilizados en multitud de dispositivos que abarcan desde los supercomputadores más capaces hasta los teléfonos móviles más populares, pasando por los ordenadores que utilizamos diariamente en nuestros escritorios. La filosofía de los sistemas Unix se caracteriza por:

- un sistema de ficheros jerárquico,
- una gran colección de pequeños programas que pueden trabajar en serie,
- el uso de ficheros de texto para almacenar los datos,
- tratar los dispositivos como ficheros.

> Linux y MacOS X son ejemplos de sistemas Unix.

Unix es un sistema operativo portable, multitarea y multiusuario desarrollado a partir de 1969.

1.2 Linux

Linux es una familia de sistemas operativos de tipo Unix que utilizan el kernel Linux. Linux puede instalarse en prácticamente cualquier ordenador personal además en en teléfonos móviles y supercomputadores.

El nombre proviene del programador original, un estudiante llamado Linus Torvals, que en 1991 completando las herramientas GNU desarrolladas por el proyecto GNU de la Fundación del Software Libre, creó la primera versión de este sistema operativo. El papel fundamental jugado por estas herramientas libres del proyecto GNU hace que este sistema operativo sea denominado también como GNU/Linux, pero en este texto utilizaremos la denominación más sencilla y corta.

El desarrollo de Linux es uno de los ejemplos más claros de desarrollo de software libre por una comunidad dispersa de programadores. Cualquiera puede usar el sistema operativo, estudiarlo y modificarlo. Estos derechos están protegidos por la licencia GPL (GNU General Public License).

1.3 Distribuciones

Linux, como cualquier otro sistema operativo, se compone de un gran número de piezas, que, en este caso, son desarrolladas de forma independiente por miles de programadores y proyectos. Normalmente estas piezas son integradas por un distribuidor y Linux es suministrado como una distribución Linux. Las distribuciones Linux incluyen todo el software necesario para instalar un servidor o un escritorio. Algunas de las aplicaciones comúnmente incluidas incluyen: el navegador web Firefox y las aplicaciones de oficina LibreOffice.

Existen cientos de distribuciones Linux. Estas distribuciones están adaptadas para usuarios o tareas específicas. Algunas de estas distribuciones están desarrolladas o apoyadas por empresas como Fedora (Red Hat) y Ubuntu (Canonical) mientras que otras son mantenidas por la propia comunidad de usuarios como Debian.

1.4 El escritorio

> En nuestro caso, nosotros vamos a centrarnos en el uso de Linux desde la línea de comandos (CLI). Aun así, es bueno conocer los diferentes entornos de escritorio, por si queremos hacer de Linux nuestro sistema operativo *de cabecera*.

Todos las distribuciones basadas en entornos graficos (GUI) suelen tener varios entornos de escritorio para elegir. Los entornos de escritorio suelen diferir por:

- El estilo y apariencia del entorno
- La forma en la que los diferentes elementos se disponen en la pantalla
- La forma en la que el usuario navega por el escritorio

En el caso de Ubuntu el entorno de escritorio por defecto se denomina Unity. Se caracteriza por tener dos barras, la denominada Menu Bar y el Launcher. El Menu Bar incorpora por una lado los menus de las aplicaciones que están activas y, por otro, un area de indicadores que nos ofrecen informacion actualizada del sistema en todo momento. El Launcher es la barra vertical que facilita el acceso a las aplicaciones mas usadas y a su estado, además de a los discos montados y a la papelera. Además tenemos el selector de escritorios virtuales. En el launcher encontramos varias aplicaciones especiales:

- El menu

- El selector de escritorios virtuales
- La papelera

Los escritorios virtuales sirven para ampliar la zona de trabajo. Por defecto hay 4 escritorios virtuales que amplían nuestro monitor por cuatro.

1.5 La terminal de Linux/Unix

La Shell (o terminal) es un interprete de comandos. Es simplemente un modo alternativo de controlar un ordenador basado en una interfaz de texto. La terminal nos permite ejecutar software escribiendo el nombre del programa que queremos ejecutar en la terminal. Podemos pedirle al ordenador que ejecute un programa mediante el ratón ciclando en distintos lugares del escritorio o podemos escribir una orden para conseguir el mismo objetivo. Por ejemplo, para pedirle al ordenador que nos de una lista de los archivos presentes en un directorio podemos abrir un navegador de archivos o podemos escribir en la terminal:

```
$ ls folder_name
file_1.txt
file_2.txt
```

Ninguna de las dos formas de comunicarse con el ordenador es mejor que la otra aunque en ciertas ocasiones puede resultar más conveniente utilizar una u otra Las ventajas de la línea de comandos son:

- **Necesidad**. Existe mucho software que está sólo disponible en la terminal. Esto es especialmente cierto en el área de la bioinformática.
- **Flexibilidad**. Los programas gráficos suelen ser muy adecuados para realizar la tarea para la que han sido creados, pero son difíciles de adaptar para otras tareas. Los programas diseñados para ser usados en la línea de comandos suelen ser muy versátiles.
- **Reproducibilidad**. Documentar y repetir el proceso seguido para realizar un análisis con un programa gráfico es muy costoso puesto que es difícil describir la secuencia de clicks y doble clicks que hemos realizado. Por el contrario, los procesos realizados mediante la línea de comandos son muy fáciles de documentar puesto que tan sólo debemos guardar el texto que hemos introducido en la pantalla.
- **Fiabilidad**. Los programas básicos de Unix fueron creados en los años 70 y han sido probados por innumerables usuarios por lo que se han convertido en piezas de código extraordinariamente confiables.
- **Recursos**. Las interfaces gráficas suelen consumir muchos recursos mientras que los programas que funcionan en línea de comandos suelen ser extraordinariamente livianos y rápidos. Este poco uso de recursos facilita, por ejemplo, que se utilice a través de la red.

El problema de la terminal es que para poder utilizarla debemos saber previamente qué queremos hacer y cómo. Es habitual descubrir como funciona un programa con una interfaz gráfica sin tener que leer un manual, esto no sucede en la terminal.

Para usar la línea de comandos hay que abrir una terminal. Se abrirá una terminal con un mensaje similar a:

```
usuario $
```

Este pequeño mensaje se denomina prompt y el cursor parpadeante que aparece junto al él indica que el ordenador está esperando una orden. El mensaje exacto que aparece en el prompt puede variar ligeramente, pero en Ubuntu suele ser similar a:

```
usuario@ordenador:~/documentos$
```

En el prompt de Ubuntu se nos muestra el nombre del usuario, el nombre del ordenador y el directorio en el que nos encontramos actualmente, es decir, el directorio de trabajo actual.

Cuando el prompt se muestra podemos ejecutar cualquier cosa, por ejemplo le podemos pedir que liste los ficheros mediante el comando ls (LiSt):

```
usuario $ ls
lista_libros.txt
rectas_cocina/
```

ls, como cualquier otro comando, es en realidad un programa que el ordenador ejecuta. Cuando escribimos la orden (y pulsamos enter) el programa se ejecuta. Mientras el programa está ejecutándose el prompt desaparece y no podemos ejecutar ningún otro comando. Pasado el tiempo el programa termina su ejecución y el prompt vuelve a aparecer. En el caso del comando ls el tiempo de ejecución es tan pequeño que suele ser imperceptible.

Los programas suelen tener unas entradas y unas salidas. Dependiendo del caso estas pueden ser ficheros o caracteres introducidos o impresos en la pantalla. Por ejemplo, el resultado de ls es simplemente una lista impresa de ficheros y directorios en la interfaz de comandos.

Normalmente el comportamiento de los programas puede ser modificado pasándoles parámetros. Por ejemplo, podríamos pedirle al programa ls que nos imprima una lista de ficheros más detallada escribiendo:

```
$ ls -l
```

1.6 Ayuda

Cada comando tiene unos parámetros y opciones distintos. La forma estándar de pedirles que nos enseñen cuales son estos parámetros suele ser utilizar las opciones '–help', '-h' o '-help', aunque esto puede variar en comandos no estándar.

```
$ ls --help
Modo de empleo: ls [OPCIÓN]... [FICHERO]...
List information about the FILEs (the current directory by default).
Sort entries alphabetically if none of -cftuvSUX nor --sort.
```

Otro modo de acceder a una documentación más detallada es acceder al manual del programa utilizando el comando `man` (MANual):

```
$ man ls
(para terminar pulsar "q")
```

man es un programa interactivo, cuando ejecutamos el comando el programa se abre y el prompt desaparece. *man* es en realidad un visor de ficheros de texto por lo que cuando lo ejecutamos la pantalla se rellena con la ayuda del programa que hemos solicitado. Podemos ir hacia abajo o hacia arriba y podemos buscar en el contenido de la ayuda. El prompt y la posibilidad de ejecutar otro programa no volverán a aparecer hasta que no cerremos el programa interactivo. En el caso de man para cerrar el programa hay que pulsar la tecla “q”.

1.7 Autocompletado e historia

El intérprete de comandos dispone de algunas utilidades para facilitarnos su uso. Una de las más utilizadas es el completado automático. Podemos evitarnos escribir una gran parte de los comandos haciendo uso de la tecla tabulador. Si empezamos a escribir un comando y pulsamos la tecla tabulador el sistema completará el comando por nosotros. Para probarlo creemos los ficheros datos_1.txt, datos_2.txt y tesis.txt::

```
~$ touch datos_1.txt
~$ touch datos_2.txt
~$ touch experimento.txt
```

Si ahora empezamos a escribir cp e y pulsamos el tabulador dos veces, el intérprete de comandos completará el comando automáticamente::

```
~$ cp e
~$ cp experimento.txt
```

Si el intérprete encuentra varias alternativas completará el comando hasta el punto en el que no haya ambigüedad. Si deseamos que imprima una lista de todas las alternativas disponibles para continuar con el comando deberemos pulsar el tabulador dos veces.

```
~$ cp d
$ cp datos_
datos_1.txt  datos_2.txt
~$ cp datos_
```

Otra de las funcionalidades que más nos pueden ayudar es la historia. El intérprete recuerda todos los comandos que hemos introducido anteriormente. Si queremos podemos obtener una lista de todo lo que

hemos ejecutado utilizando el comando history. Pero lo más socorrido es simplemente utilizar los cursores arriba y abajo para revisar los comandos anteriores. Otra forma de acceder a la historia es utilizar la combinación de teclas control y r. De este modo podemos buscar comandos antiguos sencillamente.

2. El sistema de ficheros

El sistema de archivos controla como se almacenan los archivos en el ordenador. Sus dos tareas principales son guardar y leer archivos previamente guardados.

2.1 Sistemas jerárquicos

Los sistemas de archivos suelen tener directorios en los que organizar los archivos y estos directorios suelen estar organizados jerárquicamente. La jerarquía implica que un directorio puede contener subdirectorios. El directorio más alto en la jerarquía del que cuelgan todos los demás se denomina raíz (root). En los sistemas Unix el directorio raíz se representa con una barra "/" y sólo existe una jerarquía, es decir, sólo existe un directorio raíz, incluso aunque haya distintos discos duros en el ordenador.

Dentro del directorio raíz podemos encontrar diversos subdirectorios, por ejemplo en Linux existe el directorio home. home es por tanto un subdirectorio del directorio raíz. Esta relación se representa como:

```
/home
```

home es el directorio dónde se encuentran los directorios de los usuarios en un sistema Linux. Imaginemos que tiene los subdirectorios alicia y juan. Se representaría como:

```
/home/alicia
/home/juan
```

Existe un estándar, denominado Filesystem Hierarchy Standard que define la estructura de directorios de los sistemas Unix. Los sistemas Unix suelen seguir este estándar, aunque a veces lo violan en algunos aspectos. Por ejemplo en MacOS X el directorio donde se encuentran los directorios de los usuarios se denomina *Users* y no *home*

En el directorio raíz hay diversos directorios que, en la mayoría de los casos, sólo deberían interesarnos si estamos administrando el ordenador. Los usuarios normalmente sólo escriben dentro de un directorio de su propiedad localizado dentro de /home y denominado como su nombre de usuario.

Los usuarios también pueden escribir en /tmp aunque normalmente son los procesos lanzados por estos lo que hacen esta escritura. Es importante revisar el espacio libre en la partición en la que se encuentra /tmp para que no se colapse el sistema. Recuerda que /tmp es borrado habitualmente por el sistema. Normalmente con cada nuevo arranque.

2.2 Rutas absolutas, relativas y directorio de trabajo

Para referirnos a un archivo o a un directorio debemos indicar su ruta (path. Un ejemplo de ruta podría ser:

```
/home/alicia/documentos/tesis.md
```

Este tipo de rutas en las que se especifican todos los subdirectorios empezando desde el directorio raíz se denominan rutas absolutas.

Para no tener que escribir la ruta absoluta completa cada vez que queremos referirnos a un archivo o a un directorio se crearon los conceptos de directorio de trabajo y de ruta relativa.

El directorio de trabajo es una propiedad del terminal (del shell) en la que estamos trabajando. Siempre que estemos trabajando en una terminal tendremos asignado un directorio de trabajo. Por ejemplo, si nuestro usuario es alicia sería normal que al abrir un terminal nuestro directorio de trabajo fuese:

```
/home/alicia
```

El directorio de trabajo se utiliza para escribir rutas a archivos relativas al mismo. De este modo nos ahorramos escribir bastante. Imaginemos que Alicia tiene en su directorio un documento llamado peliculas.txt. La ruta absoluta sería.

```
/home/alicia/peliculas.txt
```

Mientras su directorio de trabajo sea `/home/alicia` la ruta relativa sería simplemente:

```
peliculas.txt
```

Es decir, podemos escribir rutas relativas al directorio de trabajo, rutas que en vez de partir del directorio raíz parten desde el directorio de trabajo. Las rutas relativas se diferencian de las absolutas en los sistemas Unix porque las absolutas empiezan por “y” las relativas no.

Es común referirse al directorio de trabajo de una terminal como a un lugar en el que nos encontramos mientras estamos trabajando en la terminal. Siempre que estemos en una terminal estaremos dentro de un directorio de trabajo.

Por ejemplo, cuando abrimos un nuevo terminal el directorio de trabajo se sitúa en /home/nombre_de_usuario. Si ejecutamos el comando ls, el programa asumirá que queremos listar los archivos presentes en ese directorio y no en otro cualquiera. Existe un comando que nos informa sobre el directorio de trabajo actual, pwd (*Print Working Directory*):

```
$ pwd
/home/alicia
```

Si deseamos podemos modificar el directorio de trabajo "moviéndonos" a otro directorio. Para lograrlo hay que utilizar el comando *cd* (Change Directory):

```
$ cd documentos
$ pwd
/home/alicia/documentos
```

A partir de ese momento los comandos asumirán que si no se les indica lo contrario el directorio desde el que deben trabajar es /home/alicia/documentos.

cd además tiene algunos parámetros especiales:

```
cd      Ir al directorio $HOME del usuario.
cd -    Ir al directorio de trabajo previo
```

2.3 Directorio $HOME

El directorio $HOME en los sistemas Unix, que son sistemas multiusuario, es el directorio en el que el usuario debe mantener sus ficheros y directorios. Fuera de este directorio el usuario tendrá unos permisos restringidos puesto que sus acciones podrían afectar a otros usuarios.

En Linux los directorios $HOME de los usuarios son subdirectorios del directorio /home.

El directorio $HOME de un usuario es además el directorio de trabajo por defecto, es decir, el directorio de trabajo que se establece cuando se abre una terminal.

2.4 Moviendo, renombrando y copiando ficheros

En primer lugar vamos a crear un fichero de prueba:

```
~$ touch data.txt
~$ ls
data.txt
```

El comando *touch*, en este caso, ha creado un fichero vacío.

Los ficheros se copian con el comando *cp* (CoPy):

```
~$ cp data.txt data.bak.txt
~$ ls
data.bak.txt  data.txt
```

Se mueven y renombran con el *mv* (MoVe):

```
~$ mv data.txt experimento_1.txt
~$ ls
data.bak.txt  experimento_1.txt
```

Para crear un nuevo directorio podemos utilizar la orden *mkdir* (MaKeDIRectory):

```
~$ mkdir exp_1
~$ ls
data.bak.txt  exp_1  experimento_1.txt
```

mv también sirve para mover ficheros entre directorios:

```
~$ mv experimento_1.txt exp_1/
~$ ls
data.bak.txt  exp_1
~$ ls exp_1/
experimento_1.txt
```

Los ficheros se eliminan con la orden *rm* (ReMove):

```
~$ rm data.bak.txt
~$ ls
exp_1
```

En la línea de comandos de los sistemas Unix cuando se borra un fichero se borra definitivamente, no hay papelera. Una vez ejecutado el *rm* no podremos recuperar el archivo.

Los comandos *cp* y *rm* no funcionarán bien con los directorios a no ser que modifiquemos el comportamiento que muestran por defecto:

```
~$ rm exp_1/
rm: cannot remove exp_1/ Is a directory
~$ cp exp_1/ exp_1_bak/
cp: omitting directory exp_1/
```

Esto sucede porque para copiar o borrar un directorio hay que copiar o borrar todos sus contenidos recursivamente y esto podría alterar muchos datos con un sólo comando. Por esta razón se exige que estos dos comandos incluyan un modificador que les indique que sí deben funcionar recursivamente cuando tratan con directorios:

```

```

```
~$ cp -r exp_1/ exp_1_bak/
~$ ls
exp_1  exp_1_bak
~$ rm -r exp_1_bak/
~$ ls
exp_1
```

2.5 Nombres de directorios y archivos

En Unix los archivos pueden tener prácticamente cualquier nombre. Existe la convención de acabar los nombres con un punto y una pequeña extensión que indica el tipo de archivo. *Pero esto es sólo una convención, en realidad podríamos no utilizar este tipo de nomenclatura*.

Si deseamos utilizar nombres de archivos que no vayan a causar extraños comportamientos en el futuro lo mejor sería seguir unas cuantas reglas al nombrar un archivo:

- Añadir una extensión para recordarnos el tipo de archivo, por ejemplo .txt para los archivos de texto.
- No utilizar en los nombres:
    - espacios,
    - caracteres no alfanuméricos,
    - ni caracteres no ingleses como letras acentuadas o eñes.

Por supuesto, podríamos crear un archivo denominado "$ñ 1.txt" para referirnos a un archivo de sonido, pero esto conllevaría una sería de problemas que aunque son solventables nos dificultarán el trabajo.

Además es importante recordar que en Unix las mayúsculas y las minúsculas no son lo mismo. Los ficheros "documento.txt", "Documento.txt" y "DOCUMENTO.TXT" son tres ficheros distintos.

Otra convención utilizada en los sistema Unix es la de ocultar los archivos cuyos nombres comienzan por punto ".". Por ejemplo el archivo ".oculto" no aparecerá normalmente cuando pedimos el listado de un directorio. Esto se utiliza normalmente para guardar archivos de configuración que no suelen ser utilizados directamente por los usuarios. Para listar todos los archivos (All), ya sean éstos ocultos o no se puede ejecutar:

```
$ ls -a
.               .fontconfig          .HyperTree           .pki
..               fsm.jpg              .ICEauthority        .recently-used
```

Esta convención de ocultar los ficheros cuyo nombre comienza por un punto se mantiene también en el navegador gráfico de ficheros. En este caso podemos pedir que se muestren estos archivos en el menú Ver -> Mostrar los archivos ocultos.

Para acelerar el acceso a ciertos directorios existen algunos nombres especiales que son bastante útiles:

```
* ".." indica el directorio padre del directorio actual
* "." indica el directorio actual
* "~" representa la $HOME del usuario
```

2.6 WildCards

En muchas ocasiones resulta útil tratar los ficheros de un modo conjunto. Por ejemplo, imaginemos que queremos mover todos los ficheros de texto a un directorio y la imágenes a otro. Creemos una pequeña demostración:

```
~$ touch exp_1a.txt
~$ touch exp_1b.txt
~$ touch exp_1b.jpg
~$ touch exp_1a.jpg
~$ ls
exp_1  exp_1a.jpg  exp_1a.txt  exp_1b.jpg  exp_1b.txt
```

Podemos referirnos a todos los archivos que acaban en txt utilizando un asterisco:

```
~$ mv *txt exp_1
~$ ls
exp_1  exp_1a.jpg  exp_1b.jpg
```

El asterisco sustituye a cualquier texto, por lo que al escribir *txt incluimos a cualquier fichero que tenga un nombre cualquiera, pero que termine con las letras txt. Podríamos por ejemplo referirnos a los ficheros del experimento 1a:

```
~$ ls *1a*
exp_1a.jpg
```

Esta herramienta es muy potente y útil, pero tenemos que tener cuidado con ella, sobre todo cuando la combinamos con rm. Por ejemplo la orden:

```
$ rm -r *
```

Borraría todos los ficheros y directorios que se encuentren bajo el directorio de trabajo actual, si lo hacemos perderemos todos los ficheros y directorios que cuelgan del actual directorio de trabajo, puede que esto sea lo que queramos, pero hemos de andar con cuidado.

2.6 Obteniendo información sobre archivos y directorios

ls es un comando capaz de mostrarnos información extra sobre los archivos y directorios que lista. Por ejemplo podemos pedirle, usando la opción -l (Long), que nos muestre quién es el dueño del archivo y cuanto ocupa y qué permisos tiene además de otras cosas:

```
~$ ls
exp_1
~$ ls -l
total 4
drwxr-xr-x 2 usuario usuario 4096 Oct 13 09:48 exp_1
```

La información sobre la cantidad de disco ocupada la da por defecto en bytes, si la queremos en un formato más inteligible podemos utilizar la opción -h (Human):

```
~$ ls -lh
total 4.0K
drwxr-xr-x 2 usuario usuario 4.0K Oct 13 09:48 exp_1
```

Podemos consultar el tipo de un archivo mediante el comando file.

```
~$ file imagen.png
imagen.png: PNG image data, 1920 x 1080, 8-bit/color RGB, non-interlaced
```

En principio, el tipo de un archivo no está determinado por la extensión, la extensión es sólo parte del nombre, aunque hay software que viola o complementa este principio. El tipo de archivo está determinado por su magic number. El magic number está compuesto por una corta serie de bytes que indican el tipo de archivo.

2.7 Permisos

Unix desde su origen ha sido un sistema multiusuario. Para conseguir que cada usuario pueda trabajar en sus archivos, pero que no pueda interferir accidental o deliberadamente con los archivos de otros usuarios se estableció desde el principio un sistema de permisos. Por defecto un usuario tiene permiso para leer y modificar sus propios archivos y directorios, pero no los de los demás. En los sistemas Unix los ficheros pertenecen a un usuario concreto y existen unos permisos diferenciados para este usuario y para el resto. Además el usuario pertenece a un grupo de trabajo. Por ejemplo, imaginemos que la usuaria alicia puede pertenecer al grupo de trabajo “diagnostico”. Si alicia crea un fichero este tendrá unos permisos diferentes para alicia, para el resto de miembros de su grupo y para el resto de usuarios del ordenador. Podemos ver los permisos asociados a los ficheros utilizando el comando ls con la opción -l (Long):

```
~$ ls -l
total 7324
-rw-r--r-- 1 alicia diagnostico    1059 Oct 20 12:42
busqueda_leukemia_100.txt
-rw-r--r-- 1 alicia diagnostico       0 Oct 13 10:53 datos_1.txt
drwxr-xr-x 2 alicia diagnostico    4096 Oct 13 10:29 experimento
```

En este caso, los ficheros listados pertenecen Alicia y al grupo diagnostico. Los permisos asignados al usuario, a los miembros del grupo y al resto de usuarios están resumidos en la primeras letras de cada línea:

```
drwxr-x---
```

La primera letra indica el tipo de fichero listado: (d) directorio, (-) fichero u otro tipo especial. Las siguientes nueve letras muestran, en grupos de tres, los permisos para el usuario, para el grupo y para el resto de usuarios del ordenador. Cada grupo de tres letras indica los permisos de lectura (Read), escritura (Write) y ejecución (eXecute). En el caso anterior el usuario tiene permiso de lectura, escritura y ejecución (rwx), el grupo tiene permiso de lectura y ejecución (r-x), es decir no puede modificar el fichero o el directorio, y el resto de usuarios no tienen ningún permiso (—).

En los ficheros normales el permiso de lectura indica si el fichero puede ser leído, el de escritura si puede ser modificado y el de ejecución si puede ser ejecutado. En el caso de los directorios el de escritura indica si podemos añadir o borrar ficheros del directorio y el de ejecución si podemos listar los contenidos del directorio.

Estos permisos pueden ser modificados con la orden chmod. En chmod cada grupo de usuarios se representa por una letra:

- u: usuario dueño del fichero
- g: grupo de usuarios del dueño del fichero
- o: todos los otros usuarios
- a: todos los tipos de usuario (dueño, grupo y otros)
-

Los tipos de permisos también están abreviados por letras:

- r: lectura
- w: escritura
- x: ejecución

Con estas abreviaturas podemos modificar los permisos existentes.

Hacer un fichero ejecutable:

```
$ chmod u+x
```

O:

```
$ chmod a+x
```

También podemos mediante chmod indicar los permisos para un tipo de usuario determinado.

```
$ chmod a=rwx
```

Un modo algo menos intuitivo, pero más útil de utilizar chmod es mediante los números octales que representan los permisos.

- lectura: 4
- escritura: 2
- ejecución: 1

Para modificar los permisos de este modo debemos indicar el número octal que queremos que represente los permisos del fichero. La primera cifra representará al dueño, la segunda al grupo y la tercera al resto de usuarios. Por ejemplo si queremos que único permiso para el dueño y su grupo sea la lectura y que no haya ningún permiso para el resto de usuarios:

```
$ chmod 110 fichero.txt
```

También podemos combinar permisos sumando los números anteriores. Por ejemplo, permiso para leer y escribir para el dueño y ningún permiso para el resto.

```
$ chmod 300 fichero.txt
```

Permisos de lectura, escritura y ejecución para el dueño y su grupo y ninguno para el resto.

```
$ chmod 770 fichero.txt
```

Las restricciones para los permisos no afectan al usuario root, al administrador del sistema. root también puede modificar quien el dueño y el grupo al que pertenecen los ficheros mediante los comando chown y chgrp.

```
$ chown alicia fichero.txt
$ chown diagnostico fichero.txt
```

2.8 Obteniendo información sobre el sistema de archivos

El sistema de archivos puede abarcar una o más particiones. Una partición es una región de un disco o de cualquier otro medio de almacenamiento. Las instalaciones de Windows tienen normalmente una partición por disco, pero en Linux esto no es tan habitual. Cada partición tiene un sistema de archivos propio, pero en Unix estos sistemas deben estar montados en algún lugar dentro de la jerarquía que cuelga de la raíz. En Windows cada partición tiene por defecto una jerarquía independiente.

Podemos pedir información sobre el espacio ocupado por las distintas particiones que tenemos actualmente montadas usando el comando df (Disk Free).

```
$ df -h
S.ficheros     Tamaño Usados  Disp Uso% Montado en
udev             7,8G      0  7,8G   0% /dev
tmpfs            1,6G   9,8M  1,6G   1% /run
/dev/nvme0n1p2    25G   8,1G   16G  35% /
tmpfs            7,8G   5,3M  7,8G   1% /dev/shm
tmpfs            5,0M   4,0K  5,0M   1% /run/lock
tmpfs            7,8G      0  7,8G   0% /sys/fs/cgroup
/dev/nvme0n1p4   206G    18G  178G   9% /home
/dev/nvme0n1p1   511M   3,6M  508M   1% /boot/efi
/dev/sda1        2,7T   117G  2,5T   5% /home/jose/magnet
tmpfs            1,6G    64K  1,6G   1% /run/user/1000
```

Algunos de los sistemas de archivos montados puede que no se correspondan con particiones en un disco físico sino con espacios de la memoria RAM que son utilizados como sistemas de archivos especiales.

El comando du (disk usage) informa sobre el espacio que ocupa un árbol de directorios. Este comando tiene equivalentes gráficos como Baobab o xdiskusage. Podemos pedir a du que nos muestre cuanto espacio ocupan los directorios bajo el directorio analysis:

```
$ du -h analyses/
36K analyses/alicia/cache
204K    analyses/alicia/differential_snps/differential
252K    analyses/alicia/differential_snps/non_differentia
919M    analyses/
```

Si sólo queremos obtener el resultado para el directorio que le hemos dado y no para sus subdirectorios podemos utilizar el parámetro -s:

```
$ du -sh analyses/
919M    analyses/
```

Si queremos información sobre todos los archivos y no sólo los directorios podemos usar -a:

```
$ du -ha analyses/
32K analyses/alicia/cache/min_called_rate_samples_cache.pickle
36K analyses/alicia/cache
8,0K    analyses/alicia/look_for_matching_accessions.py
```

2.9 Compresion y descompresión de ficheros

Existen distintos formatos de compresión de ficheros como: gzip, bzip, zip o rar. Los formatos más utilizados en Unix son gzip y bzip.

Comprimir un fichero con gzip o bzip:

```
$ gzip informacion_snps.txt
$ ls
informacion_snps.txt.gz

$ bzip2 accs.txt
$ ls
accs.txt.bz2
```

bzip2 comprime más que gzip, pero es más lento. gzip también dispone de varios niveles de compresión, cuanto más comprime más lenta suele ser la compresión.

Podemos descomprimir cualquier fichero utilizando la línea de comandos:

```
$ gunzip informacion_snps.txt.gz
$ ls
informacion_snps.txt
$ bunzip2 accs.txt.bz2
$ ls
accs.txt
```

Muchos estamos acostumbrados al formato zip. Un fichero zip no se corresponde en realidad con un sólo fichero comprimido sino con varios. Un fichero zip hace dos cosas: unir varios ficheros en uno y comprimir el resultado. Los comandos que hemos visto (gzip y bzip2) son capaces de comprimir un sólo archivo, pero no pueden unir varios archivos en uno. tar es el comando capaz de unir varios archivos en uno.

```
$ ls
seq1.fasta  seq2.fasta
$ tar -cvf secuencias.tar seq*
seq1.fasta
seq2.fasta
$ ls
secuencias.tar  seq1.fasta  seq2.fasta
```

tar también es capaz de desempaquetar los archivos que habíamos unido.

```
$ ls
secuencias.tar
$ rm seq1.fasta seq2.fasta
$ tar -xvf secuencias.tar
seq1.fasta
```

```
seq2.fasta
$ ls
secuencias.tar  seq1.fasta  seq2.fasta
```

El problema es que utilizando el comando tar tal y como lo hemos hecho hemos conseguido unir y separar archivos, pero no hemos comprimido el fichero unido. Para hacerlo podríamos utilizar los comandos gzip o bzip2, pero este no es el modo habitual de hacerlo. Dado que casi siempre que unamos archivos en un archivo tar también querremos comprimir el resultado el comando tar tiene también la capacidad de comprimir y descomprimir utilizando los algoritmos gzip y bzip2. Unir y comprimir con gzip varios archivos:

```
$ tar -cvzf secuencias.tar.gz seq*
seq1.fasta
seq2.fasta
$ ls
secuencias.tar.gz  seq1.fasta  seq2.fasta
```

Descomprimir un archivo tar.gz:

```
$ tar -xvzf secuencias.tar.gz
seq1.fasta
seq2.fasta
$ ls
secuencias.tar.gz  seq1.fasta  seq2.fasta
```

También podemos descomprimir el contenido de un fichero de texto y enviar el resultado a la terminal con el comando zcat.

```
$ zcat fichero.txt.gz
```

Con bzip2.

```
$ tar -cvjf secuencias.tar.bz seq*
seq1.fasta
seq2.fasta
$ ls
secuencias.tar.bz  seq1.fasta  seq2.fasta

$ tar -xvjf secuencias.tar.bz
seq1.fasta
seq2.fasta
```

2.10 Enlaces duros y blandos

Podemos pensar en el nombre de un fichero como en una etiqueta que apunta a una posición concreta en el disco duro, en realidad es un puntero a un inodo.

Podmeos pensar en un enlace duro como en un nombre adicional para un archivo. Si tenemos un archivo en el disco y creamos un enlace duro tendremos dos nombres para ese único archivo.

```
$ ls
archivo1.txt
$ ln archivo1.txt nombre2.txt
$ ls
archivo1.txt nombre2.txt
```

Las dos referencias, nombres, al archivo serán indistinguibles. Si borramos un nombre quedará el otro. Si modificamos un archivo se modifica independientemente del nombre por el cual estemos accediendo a él. No es muy común utilizar enlaces duro salvo en aplicaciones muy concretas, por ejemplo en versiones de copias de seguridad.

Un enlace blando, más comumente conocido como un enlace simbólico, es una referencia al nombre de un archivo, no al archivo en sí.

```
$ ls
archivo1.txt
$ ln -s archivo1.txt nombre3.txt
$ ls -l
-rw-rw-r-- 1 jose jose  0 sep 27 15:16 archivo1.txt
lrwxrwxrwx 1 jose jose 12 sep 27 15:16 nombre3.txt -> archivo1.txt
```

Si eliminamos el archivo original el enlace quedará roto.

```
$ rm archivo1.txt
$ cat nombre3.txt
cat: nombre3.txt: No existe el archivo o el directorio
```

El comportamiento de ambos tipos de enlaces cambia si sobreescribimos el fichero.

```
$ echo "hola" > hola.txt
$ cat hola.txt
hola
$ ln hola.txt hola2.txt
$ ln -s hola.txt hola3.txt
$ ls -l
-rw-rw-r-- 2 jose jose 5 sep 27 15:23 hola2.txt
lrwxrwxrwx 1 jose jose 8 sep 27 15:25 hola3.txt -> hola.txt
-rw-rw-r-- 2 jose jose 5 sep 27 15:23 hola.txt
$ echo "adios" > adios.txt
```

```
$ mv adios.txt hola.txt
$ cat hola.txt
adios
$ cat hola2.txt
hola
```

Los enlaces blandos funcionan incluso entre distintos sistemas de archivos o particiones, los duros no.

3. Procesos

Un proceso es una instancia de un programa en ejecución. Programas y procesos son entidades distintas. En un sistema operativo multitarea, múltiples instancias de un programa pueden ejecutarse simultáneamente. Cada instancia es un proceso separado.

Prácticamente todo lo que se está ejecutando en el sistema en cualquier momento es un proceso, incluyendo la shell, el ambiente gráfico, que puede tener múltiples procesos, etc.

Linux, como ya hemos visto, es un sistema operativo multitarea y multiusuario. Esto quiere decir que múltiples procesos pueden operar simultáneamente sin interferir unos con otros. Por ejemplo, si cinco usuarios desde equipos diferentes, ejecutan el mismo programa al mismo tiempo, habría cinco instancias del mismo programa, es decir, cinco procesos distintos.

Cada proceso que se inicia es identificado con un número de identificación único conocido como Process ID (PID), que es siempre un número natural.

Haciendo análisis muchas veces ejecutaremos programas, crearemos procesos, que duren un tiempo considerable. Es interesante que durante el tiempo que dure el proceso podamos consultar su estado. Los entornos UNIX tienen una serie de herramientas para poder conocer el estado de los procesos y del sistema en general.

3.1 *ps*

El comando ps (process status) nos informa sobre el estado de los procesos. Dependiendo de los parámetros que le demos nos mostrara un tipo de información u otra y unos procesos u otros.

```
$ ps
```

Si queremos obtener la lista completa de procesos podemos usar las opciones -ef:

```
$ ps –ef
```

En este caso la segunda columna nos indicará el PID o identificador único del proceso.

3.2 *kill*

El comando kill, a pesar de su nombre, no sólo sirve para matar o terminar procesos sino también para enviar señales a los procesos. La señal por defecto (cuando no se indica ninguna es terminar o matar el proceso), y la sintaxis es kill PID, siendo PID el número de ID del proceso. Pero hay otras señales que podemos enviar. Así, por ejemplo, es posible enviar una señal de STOP al proceso y se detendrá su ejecución, después cuando se quiera reanudar su ejecución podemos enviar la señal CONTinuar y el proceso continuara desde donde se quedo detenido. Con kill -l podemos acceder a una lista de todas las señales que podemos mandar a un proceso:

```
$ kill -l
```

El modo más convencional de matar un proceso es intentar primero que muera ordenadamente con un -15 (Termination signal) y sino lo conseguimos matarlo con un -9 (Kill signal):

```
$ kill -15 4719
$ kill -9 4719
```

3.3 *free*

free nos muestra información sobre el uso y disponibilidad de la memoria. Es aconsejable usar la opción -h ya que así generará la información en una forma más fácil de leer para los seres humanos.

```
$ free -h
```

3.4 *top*

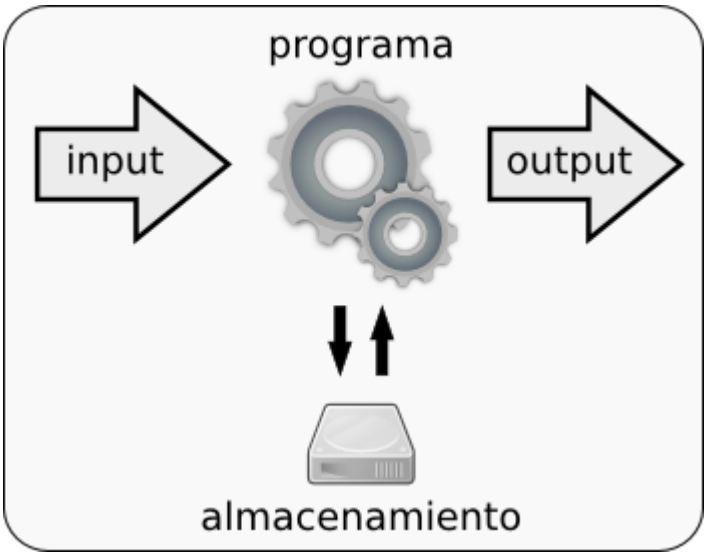
Una herramienta muy usada y muy útil para el monitoreo en tiempo real del estado de los procesos y de otras variantes del sistema es el programa llamado top, se ejecuta desde la línea de comandos, es interactivo y por defecto se actualiza cada 3 segundos.

Estando dentro de la aplicación, presionando ‘h’ se accede a una ayuda de los posibles comandos que permiten configurar top, por ejemplo, al presionar ‘s’ pregunta por el tiempo en segundos de actualización, etc.

Una alternativa más moderna al comando top, que normalmente no es instalada por defecto, es htop.

4. Flujo de información

Los programas en línea de comandos, además de poder leer y escribir en ficheros, pueden tomar argumentos como entrada, realizar una tarea e imprimir una salida en la pantalla.

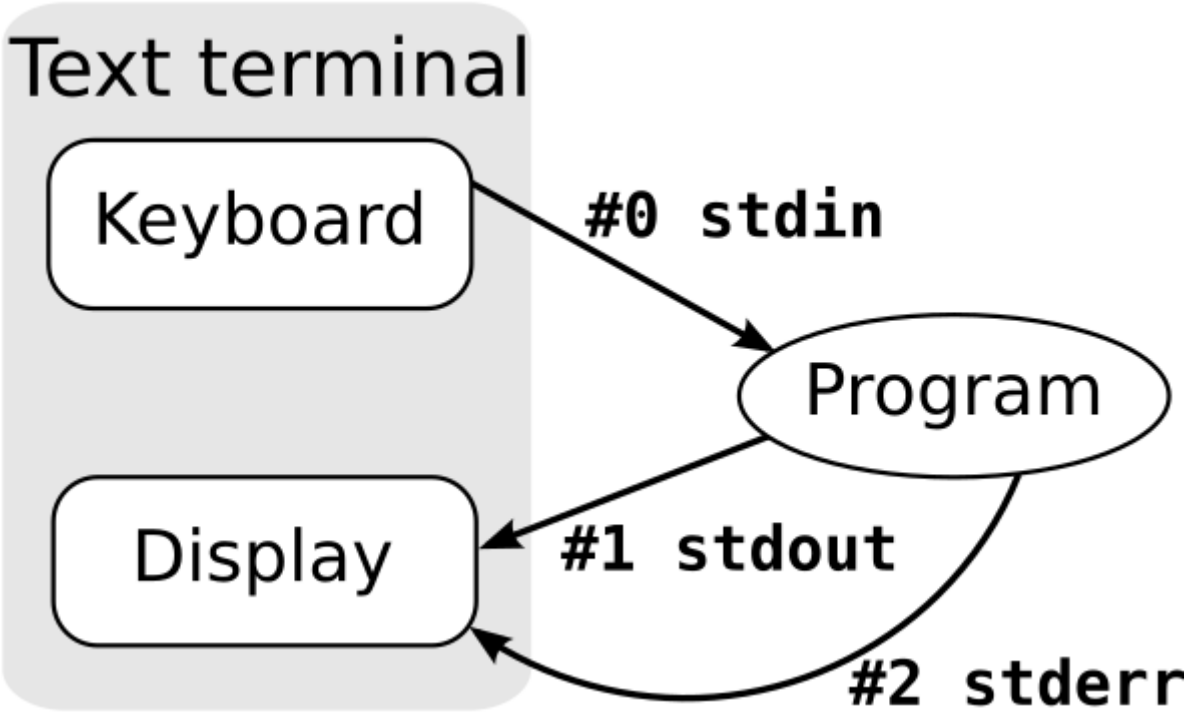


Por ejemplo, al ejecutar el comando ls se imprime un listado en la pantalla. Esto que acaba siendo impreso en la pantalla constituye lo que se denomina un flujo de información.

4.1 Flujos de información estándar

En todos los sistemas Unix existen tres flujos de información:

- standard input (stdin): datos que se envian al programa
- standard output (stdout): datos que devuelve el programa
- standard error (stderr): flujo usado por los programas para enviar un informe de errores



Los flujos permiten, por ejemplo, conectar unos programas con otros de manera que el stdout de un programa se le pase a otro por stdin, o guardar el stdout y el stderr en ficheros diferentes.

Los flujos estándar se representan por un número:

- stdin: 0

- stdout: 1
- stderr: 2

4.2 Redirección del flujo estándar

El flujo estándar puede ser redirigido para que no se imprima en pantalla, sino que quede almacenado en un fichero. Para redirigir el flujo de información desde la pantalla a un fichero no necesitamos más que utilizar el símbolo > seguido del nombre de un fichero.

```
~$ ls > listado.txt
```

Si lo hacemos se creará un nuevo fichero y en él se escribirá lo que estaba destinado a la pantalla.

```
~$ ls
listado.txt
```

En el caso de que el fichero de salida exista previamente será eliminado y vuelto a crear.

Podríamos añadir contenidos al fichero antiguo sin borrarlo utilizando dos veces el símbolo mayor que >>. Por ejemplo, añadir el listado de archivos y directorios del directorio anterior en el árbol de directorios.

```
~$ ls .. >> listado.txt
```

En los ejemplos anteriores, técnicamente lo que estamos haciendo redirigir el standard output. Sin emabargo, si ejecutamos el siguiente comando:

```
~$ ls -7 > listado.txt
```

Observaremos que el archivo listado.txt esta vacio y el programa habrá mostrado en pantalla un mensaje de error ya que la opción -7 no existe en ls. En este caso el programa envía el mensaje de error al flujo stderr, que no está siendo redirigido. Si quisiéramos redirigir también el stderr deberíamos indicarlo así:

```
~$ ls -7 2> error.txt
```

Este división de los flujos estándard nos permite guardanos por separado el resultado normal de un programa y los avisos de error

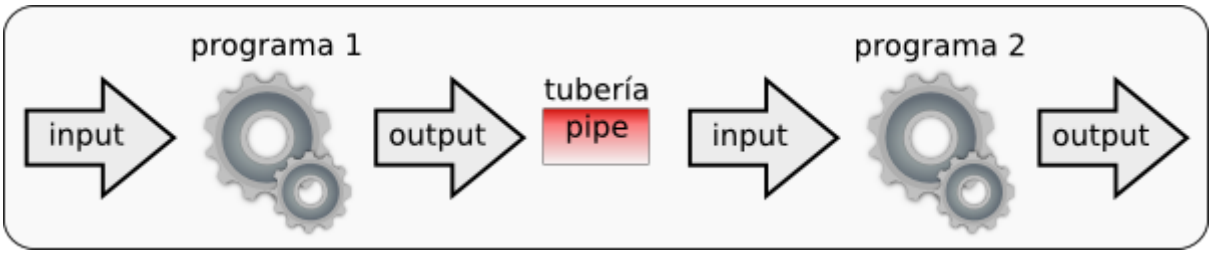
```
~$ ls > listado.txt 2> error.txt
```

Existe tambien la posibilidad de redirigir el stderr al stdout:

```
~$ ls > listado.txt 2>&1
```

4.3 Pipes

En Unix además de redirigir los flujos de información hacia un fichero podemos unir el flujo de salida de un programa con el de entrada de otro utilizando un pipe (tubería). El símbolo para el pipe es |.



Hagamos un ejemplo con los comandos wc (Word Count) y cat. wc sirve para contar líneas, palabras y caracteres. Veamos cuántos archivos y directorios hay en el directorio actual:

```
~$ ls > listado.txt
~$ wc listado.txt
 7  7 80 listado.txt
```

Podríamos hacer lo mismo en un solo paso utilizando un pipe:

```
~$ ls | wc
    7       7      80
```

En este caso la salida del comando ls (i.e., el standard output) ha sido redirigida a la entrada del comando wc (i.e., el standard input).

Esta técnica es una de las grandes fortalezas de los sistemas Unix ya que permite enlazar comandos sencillos para realizar tareas complejas y vamos a utilizarla ampliamente durante el curso.

Esta técnica posibilita, entre otras cosas, el procesamiento de los ficheros de texto de una forma potente y sencilla.

5. Procesamiento de ficheros de texto

5.1 Editores de texto

En Linux existen editores de texto que funcionan con entorno gráfico, como el gedit, KWrite o Leafpad, y editores que funcionan en la terminal, como el nano, pico o vim

vim es un editor de texto potente y versátil pero dominar su manejo requiere bastante tiempo.

nano es una versión mejorada de pico. Se trata de un editor simple y versátil pero mucho más fácil de usar que, por ejemplo, vim y está instalado por defecto en muchas distribuciones de Linux.

Vamos a ver cómo usar nano para crear o modificar archivos de texto, así como alguna de sus funcionalidades.

Podemos abrir nano directamente y se abrirá con un archivo vacio:

```
~$ nano
```

o bien indicarle la dirección de un archivo. En el caso que el archivo no exista lo creará.

```
~$ nano nombre_archivo
```

En la parte superior indica la versión y el nombre del archivo y en la parte superior se muestran algunas de las opciones del editor. Si deseamos ver una lista más completa podemos acceder a la ayuda mediante *Ctrl + g*:

- Crtl + x : Cerrar el fichero / Salir de nano
- Crtl + o : Guardar
- Alt + u : Deshacer
- Alt + e : Rehacer

Nos podemos mover por el documento con los cursores, y los botones de inicio, fin y avance y retroceso de página.

- Alt + a : Seleccionar texto desde la posición actual del cursor
- Alt + 6 : Copiar selección o línea actual
- Crtl + k : Cortar selección o línea actual
- Crtl + u : Pegar
- Alt + w : Buscar una cadena de texto o expresión regular
- Crtl + \ : Buscar y reemplazar una cadena de texto o expresión regular
- Crtl + t : Invocar el corrector ortográfico (requiere tener instalado spell)
- Alt + d : Contar el número de palabras, líneas y caracteres

5.2 Imprimiendo el contenido de ficheros

A veces, si los archivos son muy grandes incluso los editores en línea pueden tener problemas para abrirlos. Otra forma de acceder a los contenidos del fichero sería imprimir el fichero en la terminal utilizando el comando cat.

> Recuerda que las herramientas que estamos viendo sólo sirven para trabjar con ficheros de texto, no binarios.

```
~$ cat microarray_adenoma_hk69.csv
```

Si el archivo es extremadamente largo es posible que la terminal se bloquee durante bastante tiempo

> **Nota: recuerda que con Crtl + c los programas suelen terminar inmediatamente y se vuelve a mostrar el prompt.**

cat es además capaz de concatenar textos uno detrás de otro en el orden en que se los pasamos y mostrarlos en pantalla

```
~$ cat file1 file2 file3
```

Algunas opciones interesantes de cat son:

- -A : muestra también los caracteres de control, básicamente los tabuladores (como ^I) y los retornos de carro ($)
- -n : numera todas las líneas

Para hacernos una idea del contenido del fichero sin bloquear la terminal podemos imprimir tan solo las primeras líneas utilizando el comando head:

```
~$ head microarray_adenoma_hk69.csv
"!Exptid=10029"
"!Experiment Name=Adenoma (HK69)"
"!Organism=Homo sapiens"
"!Category=Adenoma"
"!Subcategory=Liver"
"!Experimenter=Xin Chen"
"!Contact email=chenx@pharmacy.ucsf.edu"
"!Contact Address1=Dept. of Biopharmaceutical Sciences"
"!Contact Address2=513 Parnassus Ave. S-816"
"!Contact Address3=Box 0446"
```

Existe otro comando equivalente pero que nos permite imprimir el final de los archivos (*tail*):

```
~$ tail microarray_adenoma_hk69.csv
24183       "EMPTY" "EMPTY" 19      27      32      0
```

Tanto a head como a tail podemos pedirles que impriman el número de líneas que nosotros deseemos mediante la opción -n:

```
~$ head -n 2 microarray_adenoma_hk69.csv
"!Exptid=10029"
"!Experiment Name=Adenoma (HK69)"
```

Otro comportamiento de tail que resulta útil es que puede mostrar todas las líneas excepto las k primeras líneas. Para ello hay que usar la opción -n y el número de líneas que queremos omitir precedido por un +. Por ejemplo, para omitir la cabecera del archivo de micro_adenoma que ocupa las primeras 20 líneas podemos hacer:

```
~$ tail -n +20 microarray_adenoma_hk69.csv
```

6. Administrador

Los sistemas Unix entre los que se encuentra Linux, son sistemas multiusuario, en los que muchos usuarios pueden usar el sistema concurrentemente. No todos los usuarios tienen los mismos permisos en el sistema, existe un usuario especial llamado root que es el administrador o superusuario.

El usuario root a diferencia del resto de los usuarios, no tiene su carpeta personal en `/home` sino que la tiene en una carpeta separada: `/root`.

Por defecto un usuario no puede modificar nada excepto su carpeta personal. Por ejemplo un usuario no puede instalar ninguna aplicacion fuera de sus carpeta personal. Para poder hacerlo el usuario tiene dos opciones:

1. Adquirir temporalmente privilegios de root
2. Convertirse en root.

6.1 sudo

En ubuntu por defecto el usuario root está deshabilitado y la unica forma de conseguir permisos de superusuario es mediente el comando `sudo`. Escribiendo sudo en la consola delante de cualquier comando, estaremos ejecutando el comando con permisos de superusuario:

```
user@virtual:~$ cat /etc/shadow
cat: /etc/shadow: Permiso denegado
user@virtual:~$ sudo cat /etc/shadow
root::17074:0:99999:7:::
daemon:*:17001:0:99999:7:::
bin:*:17001:0:99999:7:::
sys:*:17001:0:99999:7:::
sync:*:17001:0:99999:7:::
```

No todos los usuarios tienen permiso para usar sudo, solo aquellos usuarios que son administradores. Para convertirse en administrador y poder usar sudo, un usuario solo tiene que ser añadido al grupo sudo.

Además con sudo se puede hacer login como root. Con la opcion `sudo -s`:

```
user@virtual:~$ sudo -s
[sudo] password for user:
root@virtual:~#
```

6.2 su

Si no estamos en el grupo de sudo o si sudo no está disponible en el sistema, podemos usar el comando su para poder cambiar de usuario en la misma sesión.

```
user@virtual:~$ su
root@virtual:/home/user# id
uid=0(root) gid=0(root) grupos=0(root)
```

En ubuntu no podremos convertirnos en root ya que el usuario está deshabilitado. Si queremos habilitarlo, lo unico que tenemos que hacer es darle un password nuevo:

```
~$ sudo passwd
[sudo] password for user:
Introduzca la nueva contraseña de UNIX:
Vuelva a escribir la nueva contraseña de UNIX:
passwd: contraseña actualizada correctamente
```

su ademas de cambiar al usuario root, tambien permite cambiar a cualquier otro usuario, siempre que sepamos el password. Un truco muy util para “convertirmos” en otro usuario sin saber su password es combinar sudo y su:

```
user@virtual:~$ sudo su  user2
user2@virtual:/home/user$ id
uid=1001(user2) gid=1001(user2) grupos=1001(user2)
user2@virtual:/home/user$
```

7. Variables de entorno

Las variables de entorno son una lista de ajustes que guardan varios estados de una sesión. Cuando se inicia una sesión ya sea en el entrono gráfico o en una terminal, se leen las variables de entorno. Para poder acceder al contenido de una variable podemos utilizar el commando `echo` y el nombre de la variable precedido de un $:

```
~$ echo $LANG
  es_ES.UTF-8
```

Este comando nos muestra el idioma en el que se mostraran los mensajes en la shell.

Si queremos ver las variables de entorno que estamos usando, tenemos el comando `env`. Nos mostrará todas las variables de entorno que hay cargadas en nuestra sesión.

Podemos configurar las variables de entorno modificando los archivos de configuración de la shell. En nuestro caso la shell por defecto que usamos es bash, por lo tanto el fichero de configuración que debemos usar es: `$HOME/.bashrc`.

```
~$ nano $HOME/.bashrc
```

Una de las variables de entorno que se puede configurar es la variable PATH.

7.1 ¿Donde están los ejecutables? $PATH

Cuando entramos en la shell y ejecutamos un programa la shell ha de saber dónde encontrar el ejecutable correspondiente a ese comando. Este ejecutable es un fichero con el mismo nombre que hemos escrito en la shell y que tiene el permiso de ejecución activado. Por ejemplo el ejecutable del comando cp es:

```
~$ ls -l /bin/cp
-rwxr-xr-x 1 root root 109648 2010-09-21 20:32 /bin/cp
```

En este caso el fichero /bin/cp es un ejecutable que pertenece al usuario root, pero que todo el mundo puede leer y ejecutar y que se encuentra en el directorio /bin. Dado que el comando cp es simplemente un fichero ejecutable en bin para utilizarlo deberíamos escribir su ruta completa:

```
~$ /bin/cp fichero1.txt fichero2.bak
```

Aunque podríamos ejecutar el comando de ese modo nunca lo hemos hecho. De alguna forma la shell ha sabido encontrar el comando cp en el directorio /bin a pesar de que no se lo hemos dicho.

Cuando intentamos ejecutar un comando lo que la shell hace es buscar el fichero ejecutable correspondiente en una lista de directorios que se encuentra almacenada en una variable de entorno denominada `PATH`. Podemos imprimir esta lista utilizado el comando echo:

```
~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
```

En el ejemplo mostrado al escribir un comando en la shell, ésta lo buscará primero en el directorio /usr/local/sbin, después en /usr/local/bin y así sucesivamente hasta que lo encuentre en alguno de los directorios del $PATH. Si el comando no se encuentra la shell devolverá un error::

```
~$ hola
  No se ha encontrado la orden «hola», quizás quiso decir:
  La orden «cola» del paquete «git-cola» (universe)
  hola: orden no encontrada
```

Cuando intentemos ejecutar programas creados por nosotros mismos o descargados desde algún lugar deberemos tener este detalle en cuenta. Por ejemplo, si descargamos un ejecutable llamado hola_mundo y después intentamos ejecutarlo, la shell no lo encontrará:

```
~$ ls
 hola_mundo
~$ hola_mundo
 No se ha encontrado la orden «hola», quizás quiso decir:
 La orden «cola» del paquete «git-cola» (universe)
 hola_mundo: orden no encontrada
```

La shell no lo encuentra porque el directorio de trabajo (“.”) no está incluido en el PATH. Podríamos ejecutar el comando si le indicamos a la shell la ruta en la que se encuentra el comando:

```
~$ /home/usuario/hola_mundo
 Hola mundo!
```

También podríamos utilizar una ruta relativa:

```
~$ ./hola_mundo
 Hola mundo!
```

O podemos añadir el directorio donde se encuentra el ejecutable en el PATH. Si por ejemplo instalamos un programa en nuestra home en el directorio programa, podemos añadir este directorio al path y ejecutar el programa simplemente ejecutandolo con su nombre.

```
programa: /home/user/dibujacirculos/bin/draw_circle
añadimos al path:/home/user/dibujacirculos/bin
nuevo PATH: PATH=/home/user/dibujacirculos/bin:$PATH
```

Para que la shell cargue la variable de entorno, podemos utilizar el comando `source` que actualizará las variables de entorno con los valores que hayamos puesto en el fichero que carguemos, en este caso .bashrc.

```
~$ source $HOME/.bashrc
```

Se puede dar el caso que tengamos el mismo programa instalado por dos metodos distintos. Ambos programs tienen un ejecutable que se llama igual. Como podemos saber cual de los ejecutables se está ejecutando? En linux tenemos el comando `which` que nos revolvera la ruta completa. La shell va recorriendo los directorios del path en orden hasta que en alguno encuentra el ejecutable. `which` devuelve el path completo del ejecutable que la shell usa.

```
~$ which cp
 /bin/cp
```

7.2 HOME

Otra variable de entorno que se usa mucho es la variable HOME. Hace referencia a la carpeta personal de usuario. Para el usuario user la carpeta es: `/home/user`

```
~$ echo $HOME
/home/user
```

Podemos cambiar el valor de una variable de entorno. Los cambios que hagamos solo se veran en la terminal en la que estemos. Si queremos que la variable la "hereden" las aplicaciones que lanzamos desde esa shell tenemos que hacer la variable disponible, para ello podemo user el comando `export` :

```
export PATH=/binarios:$PATH
```

Por ello en el fichero .bashrc las variables las tenemos que definir con export delante.

8. Administración de software

En los sistemas Linux las distribuciones se encargan de preparar una gran cantidad de software para ser instalado. Por ejemplo, cuando queremos instalar un programa de edición fotográfica como el gimp no es necesario ni recomendable ir a la página web de los creadores del software para descargar el software. La distribución que estemos utilizando ya se ha encargado de buscar el software y empaquetarlo para que el software pueda ser instalado de un modo estandarizado. Por lo tanto si queremos instalar el gimp o

cualquier otro programa lo que debemos hacer es utilizar las herramientas que la distribución ha preparado para administrar el software que tenemos instalado en nuestro ordenador. Por ejemplo, en Ubuntu, abriríamos la aplicación llamada Software de Ubuntu y en ella buscaríamos el gimp. Con un solo click el programa quedaría instalado e integrado en nuestro sistema.

Además todo el software instalado por este método se actualiza cuando los desarrolladores del software corrigen fallos o problemas de seguridad sin que nosotros debamos preocuparnos por ello. La propia distribución se encarga de recopilar esas mejoras y de aplicarlas a todo el software que tengamos instalado.

El modo de operación es bastante distinto al usual en Windows. En Windows seríamos nosotros los que compraríamos el software y lo instalaríamos a partir de un CD o de un archivo descargado desde la web. Esta es, dicho sea de paso, una de las fuentes de virus y troyanos más habitual. Esto está cambiando y ya empezamos a tener un canal de distribucion controlado por la propia microsoft.

Mientras que las distribuciones Linux facilitan la instalación de un software certificado y libre de virus en Windows debemos ser nosotros quienes nos encarguemos de verificar que el software está libre de virus. Seríamos nosotros también los encargados de estar al tanto de instalar las actualizaciones o los fallos de seguridad a no ser que cada software implemente esta funcionalidad.

Las distribuciones distribuyen el software en forma de paquetes. Cada distribucion tiene su propio formato de paquetes. En debian y ubuntu se utiliza el formato .deb. Las distribuciones organizan el software de forma que se optimiza lo mejor que se pueda los recursos de un ordenador. Por ejemplo si dos aplicaciones utilizan una libreria, en vez de poner la libreria por duplicado en cada una de las aplicaciones se crea un nuevo paquete solo con la libreria y se crea una dependencia desde las aplicaciones que la utilizan. De las dependencias se encargan los sistemas de paquetes que veremos más adelante.

Por desgracia, la cantidad de software incluido en las distribuciones no es infinito y, aunque el catálogo es amplio y más que suficiente para la gran mayoría de las aplicaciones que un usuario doméstico o de la administración pueda necesitar, hay programas, especialmente los hechos para campos específicos que no están incluidos en las distribuciones.

8.1 Repositorios

Los repositorios son los bancos de datos que alojan los catalogos de aplicaciones de las distribuciones. Todas las distribuciones tienen sus propios repositorios, se encargan de mantenerlos actualizados, de hacer tests de seguridad y de buscar errores en las aplcaciones.

Ademas de los repositorios de de las distribuciones, tambien tenemos repositorios de otras fuentes, los más conocidos son los PPA(Personal Package Archives). Son repositorios que los usuarios ponen a disposicion de otros usuarios para que puedan instalarse aplicaciones que las distribuciones no tienen, como por ejemplo el Java de oracle.

En ubuntu y debian el fichero donde encontramos la configuracion de los repositorios es `/etc/apt/sources.list`. Cada una de las lineas del fichero nos da acceso a los paquetes de un repositorio. En cada repositorio tenemos un listado con las aplicaciones que dispone, la version de cada paquete y una direcciones para poder descargar e instalarla.

En el directorio `/etc/apt/sources.list.d` podemos tener tambien ficheros con más lineas de repositorios.

8.2 Herramientas de manejo de paquetes

Existen aplicaciones graficas para poder actualizar o instalar nuevos paquetes. En ubuntu tenemos el Software de Ubuntu que nos permite instalar aplicaciones de una forma fácil y comoda.

Tenemos que tener en cuenta que si usamos Software de Ubuntu solo podremos acceder a unos cuantos paquetes disponibles en la distribucion. El porque es algo que podemos preguntar a Ubuntu.

Por debajo esta aplicacion utiliza Apt (del inglés Advanced Package Tool) que es un set de herramientas del nucleo de debian y que Ubuntu a importado. APT es uno de los sistemas más avanzados en cuento a la gestion de software.

8.2.1 APT

Apt es un set de herramientas que nos ayudan a que la instalacion, borrado o actualizacion de programas sea experiencia no traumatica. Apt esta compuesto de muchos programas, en el curso nos vamos a centrar en aquellos que consideramos más utiles: Para borrar, instalar, actualizar:

- apt-get
  - update
  - upgrade
  - dist-upgrade
  - install
  - remove
  - clean
  - autoremove

Para hacer busquedas de paquetes:

- apt-cache
  - search
  - show

Para hacer busquedas en los ficheros de los paquetes:

- apt-file
  - update
  - search

Añadir repositorios PPA

- add-apt-repository

8.2 dpkg

Es otra herramienta para interactuar con paquetes .deb. Con las diferentes opciones de `dpkg` se pueden instalar y borrar programas pero al no tener en cuenta las dependencias, lo hace más tedioso de usar. Aunque tiene varias opciones que las hacen muy util:

- `dpkg -l` : Lista todas las aplicaciones del sistema, las que se han instalado como las que se han borrado

- `dpkg -L paquete` : Lista la ruta de todos los ficheros que se han intalado al instalar el paquete.

8.3 Estándar de jerarquía del sistema de archivos

Antes de instalar software no administrado por nuestra distribución conviene que tengamos una idea de como se organiza el sistema de archivos en Linux para que entendamos dónde se instala el software. Linux sigue una norma estándar llamada Estándar de jerarquía del sistema de archivos que define los directorios del sistema y la localización de los distintos tipos de archivos.

En Unix todos los archivos y directorios aparecen bajo el directorio raíz (/). Es habitual que dentro del directorio raíz existen tres jerarquías en las que se distribuyen los archivos y directorios de los programas:

- `/`,**jerarquía primaria**. De ella cuelgan el resto de jerarquías. Los archivos incluidos directamente en esta jerarquía son sólo los esenciales para el sistema.
- `/usr/`, **jerarquía secundaria**. Contiene la mayoría de aplicaciones del sistema. En las distribuciones Linux es la jerarquía en las que los programas de uso común son instalados, como por ejemplo el LibreOffice o el entorno de usuario Gnome.
- `/usr/local/`, **jerarquía terciaria**. Contiene la mayoría de las aplicaciones que instalamos sin la mediación de la distribución.

Dentro de cada una de las jerarquías hay varios directorios en los que se distribuyen los archivos de las aplicaciones dependiendo del tipo de archivo. Por ejemplo, los ejecutables se encuentran en los directorios *bin* y las librerías en *lib*. En mi ordenador el ejecutable cp, que es esencial para el sistema se encuentra en `/bin/cp`, el editor de textos gedit que ha sido instalado por la distribución y no es esencial para el sistema se encuentra en `/usr/bin/gedit` y el alineador de secuencias bwa (software usado en biliogía) que yo he instalado manualmente sin utilizar el gestor de paquetes se encuentra en `/usr/local/bin/bwa`.

A diferencia de otros sistemas operativos en los sistemas Linux las aplicaciones normalmente no están contenidas en un sólo directorio. Los ejecutables están en `bin`, las librerías de los que dependen en `lib`, etc. Cuando se instalan programas estáticos, que incluyen todas sus librerías, suelen instalarse en `/opt/`. *opt/* sigue un modelo similar al Archivos de programa de Windows. No es muy común que los programas se instalen de esta forma en Linux, pero algunos programas como el IDE java eclipse sí suelen instalarse en `/opt`.

> **Bibliografía** Este material es un resumen del encontrado en este enlace. Consultado el 04/09/2021.