Tema 2 - Concpetos básicos de Linux y Comandos básicos sobre ficheros y directorios.

ÍNDICE

`Directorios especiales en Linux`

`Directorios importantes del Sistema`

`Wildcards`

`Comandos básicos de Linux`

- `cd` : Cambiar de directorios
- `mkdir` : Creación de directorios
- `rmdir` : Borrado de directorios
- `touch` : Creación y actualización de ficheros
- `cat` : Mostrar el contenido de un fichero
- `less` : Mostrar el contenido de un fichero, permitiendo interacción dinámica
- `tail` : Mostrar las últimas 10 líneas de un fichero
- `head` : Mostrar las primeras líneas 10 de un fichero (cabecera)
- `echo` : Imprimir en la consola y en ficheros
- `ln` : Enlaces simbólicos
- `cp` : Copiar ficheros (y directorios)
- `mv` : Mover/Renombrar ficheros y directorios
- `rm` : Eliminar ficheros (y directorios)

5 Comandos Avanzados sobre Ficheros

- `truncate` : Como vaciar un fichero
- `grep` : Búsqueda de textos en ficheros y a la salida de comandos
- `find` : Buscando ficheros dentro de un directorio (y subdirectorios)
- `awk` : Una herramienta avanzada para el tratamiento de textos
- `sed` : Edición de textos desde el terminal
- `tr` : Traductor (reemplazo o eliminación de caracteres).
- `cut` : Cortar textos
- `wc` : Word Count - Calculadora en el terminal

6 Comandos para Empaquetar y Comprimir ficheros y directorios

`Empaquetar (.tar)`

`Comprimir (.tgz, .tar.gz, .bz2)`

`Desempaquetar y descomprimir con el comando .tar`

`Uso de comados de compresión y descompresión de ficheros`

`Comprimir`

- `zip`
- `gzip`
- `bzip2`

`Empaquetar`

- `tar`

`Descomprimir`

- `unzip`
- `gunzip` (equivalente a: gzip -d)
- `bunzip2` (equivalente a bzip -d)
- `tar` (para descomprimir)

7 Redirecciones

`Redireccionamiento de salida`

`Redireccionamiento de entrada`

`Uso de los streams stdin, stdout stderror`

8 Tuberías (pipes)

9 Gestión de Paquetes

`Gestor de Paquetes`

`Repositorios y Paquetes`

`apt-get vs aptitude`

`Como instalar aptitude`

10 Otros comandos de utilidad: La fecha, la hora y el calendario

`date`

`cal`

`timedatectl`

EMPEZAMOS.....

1 Instalación y Conceptos básicos

Debemos tener instalado Ubuntu 22.04 en una máquina virtual. El software de virtualización que vamos a usar es VirtualBox.

Realizaremos el proceso de instalación en clase utilizando las guías de instalación proporcionadas en "Tema2 Instalación":

- **"Intalación de virtualBox"**
- **"Instalación de Ubuntu 22.04"**

Cuando la instalación termine, accederemos al sistema mediante nuestras credenciales de acceso (**usuario y contraseña**) y si las credenciales son correctas, el sistema nos proporcionará una ***shell***.

Shell o interprete de comandos

Una ***shell*** es un *intérprete de comandos*, es decir, es un programa que permite a los usuarios interactuar con el Sistema Operativo, procesando las órdenes que se le indican.

La shell puede interpretar comandos internos (corresponden en realidad a órdenes interpretadas por la propio shell) o comandos externos (correspondientes a ficheros ejecutables externos al shell). Además, las shells ofrecen otros elementos para mejorar su funcionalidad, tales como variables, funciones o estructuras de control. El conjunto de comandos internos y elementos disponibles, así como su sintaxis, dependerá del shell concreto empleado.

Bourne shell (sh)

En los S.O.'s basados en unix **existen múltiples implementaciones de shell** (en Windows, el equivalente serían los programas "command.com" o "cmd.exe"). **Entre las shells de Unix destacaremos "sh"** que fue usado desde las primeras versiones y de la que han partido diferentes versiones que podriamos identificar como de la misma familia.

sh (Bourne Shell): recibe ese nombre por su desarrollador, Stephen Bourne, de los Laboratorios Bell de AT&T. A raíz de él han surgido múltiples shells, tales como zsh (Z shell), ash (almquist shell), bash (Bourne again shell), dash (Debian almquist shell) o ksh (Korn shell).

Bourne Shell **ha llegado a convertirse en un estándar de facto** de tal modo que todos los sistemas Unix tienen, al menos, una implementación del Bourne Shell (o un shell compatible con él), ubicada en `/bin/sh`.

En el caso concreto de **los S.O.'s Linux, no existe ninguna implementación del Bourne Shell, manteniéndose la entrada `/bin/sh` (así como su manual man sh) como un enlace simbólico a una implementación de shell compatible**.

bash shell

bash: fue desarrollado para ser un superconjunto de la funcionalidad del Bourne Shell (en la que incluye funcionalidades de ksh y csh), **siendo el intérprete de comandos asignado por defecto a los usuarios en las distribuciones de Linux**, por lo que es el shell empleado en la mayoría de las consolas de comandos de

Linux. Se caracteriza por una gran funcionalidad adicional a la del Bourne Shell. Como ficheros personales de los usuarios emplea `$HOME/.bashrc` y `$HOME/.profile` (o `$HOME/.bash_profile`).

Prompt

La shell **puede utilizarse desde la línea de comandos**, basada en el ***prompt*** como la indicación del shell para anunciar que espera una orden del usuario, **o puede emplearse para la interpretación de shell-scripts**. Un shell-script es un fichero de texto que contiene un conjunto de comandos y órdenes interpretables por el shell.

Un prompt puede tener un aspecto como este:

```
rpalacios@localhost ~ $
```

- **rpalacios**: es el nombre del usuario
- **localhost**: es el nombre de la máquina.
- **~** : hace referencia al directorio de trabajo del usuario (también llamado "home"), es decir, el directorio en el que nos encontramos por defecto.
- **$**: es el final del prompt. A partir de aquí escribiremos nuestros comandos. Para el usuario **root**, el símbolo es **#** en lugar de **$**.

Si queremos terminar nuestra sesión ejecutamos:

```
exit
```

Una forma rápida de acceder al Terminal desde Ubuntu Desktop es mediante la combinación de teclas:

```
Ctrl + Alt + T
```

2 Variables de entorno en Linux

Veamos un comando, un tanto peculiar:

```
echo $PATH

/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local
/games:/snap/bin
```

Mmm, probemos de nuevo:

```
echo $SHELL

/bin/bash
```

El comando **echo** nos permite ver el contenido de las `variables de entorno`.

Que son las variables de entorno

Una **variable** es un objeto que almacena un valor que se puede cambiar según las condiciones o la información del programa o entorno donde se ejecuta.

Las variables de entorno son valores dinámicos que afectan los programas o procesos que se ejecutan en un servidor. Existen en todos los sistemas operativos y su tipo puede variar. Las variables de entorno se pueden crear, editar, guardar y eliminar.

En Linux, **las variables de entorno permiten al sistema pasar datos a los programas iniciados en shells (intérpretes de comando) o sub-shells.**

`printenv` comando linux que muestra el valor de todas las variables de entorno (en el ejemplo siguiente no se muestran todas).

```
egg@UTADLRSO:~$ printenv
SHELL=/bin/bash
SESSION_MANAGER=local/UTADLRSO:@/tmp/.ICE-unix/1218,unix/UTADLRSO:/tmp/.ICE-
unix/1218
QT_ACCESSIBILITY=1
COLORTERM=truecolor
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
SSH_AGENT_LAUNCHER=gnome-keyring
XDG_MENU_PREFIX=gnome-
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
GNOME_SHELL_SESSION_MODE=ubuntu
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
XMODIFIERS=@im=ibus
DESKTOP_SESSION=ubuntu
GTK_MODULES=gail:atk-bridge
```

```
PWD=/home/egg
LOGNAME=egg
XDG_SESSION_DESKTOP=ubuntu
XDG_SESSION_TYPE=wayland
SYSTEMD_EXEC_PID=1451
XAUTHORITY=/run/user/1000/.mutter-Xwaylandauth.MLN6B2
HOME=/home/egg
USERNAME=egg
IM_CONFIG_PHASE=1
LANG=es_ES.UTF-8
```

Cada línea contiene el nombre de la variable de entorno seguido de = y el valor. Por ejemplo:

```
HOME=/home/egg
```

Comandos para ver el valor de una variable de entorno

Para ver el valor de una variable de entorno podemor usar `printenv` pasando el nombre de esa variable como argumento a dicho comando o usar el comando `echo` visto anteriormente. EL resultado es el mismo

```
printenv HOME
echo $HOME

egg@UTADLRSO:~$ printenv HOME
/home/egg
egg@UTADLRSO:~$ echo $HOME
/home/egg
egg@UTADLRSO:~$
```

Cómo crear una variable de entorno

La sintaxis básica de este comando se vería así:

```
export VAR="value"
```

- export: el comando utilizado para crear la variable.
- VAR: el nombre de la variable.
- = indica que la siguiente sección es el valor.
- «value»: el valor real

Cómo revertir el valor de una variable

Usaremos el comando `unset`. La sintaxis del comando es la siguiente:

```
unset VAR
```

Las partes del comando son:

- unset: el comando en sí
- VAR: la variable cuyo valor queremos revertir

Variable PATH

La variable PATH es una variable de entorno que contiene el listado de rutas o "paths" donde Linux busca los ejecutables de los comandos o programas a ejecutar.

```
echo $PATH

Retornará algo como esto:

/usr/local/bin:/usr/bin:/bin:/usr/games/bin:/usr/local/sbin:/usr/sbin:/sbin:/usr/lib/qt/bin
```

Cada ruta de directorio aparece separada por dos puntos ":".

En caso de que queramos agregar directorios a nuestra variable PATH podemos hacerlo de dos formas:

- La primera los hace válidos por el tiempo que dura tu sesión
- La segunda los hace permanentes.

Para el primer caso basta escribir en el terminal:

```
PATH="$PATH:<ruta nueva1>:<ruta nueva2>:...<ruta nuevaN>"
```

Supongamos que tenemos un usuario quiere agregar dos carpetas a su PATH, ambas contenidas en ~/ (directorio home): la carpeta "Scripts" y la carpeta "Compilados". Tendría que escribir en el terminal:

```
PATH="$PATH:/home/nombreusuario/Scripts:/home/nombreusuario/Compilados”
```

De esa manera **ambos directorios se agregan a su variable PATH. Sin embargo, estos cambios no son permanentes y la próxima vez que el usuario acceda a su cuenta todas las modificaciones se habrán perdido**.

Para que los cambios **sean permanentes es necesario editar un par de archivos: ~/.profile (o ~/.bash_profile) y ~/.bashrc**. Son los ficheros de configuración de bashy los veremos más adelante.

Algunas de las variables de entorno más relevantes

SHELL: Hace referencia al tipo de Shell (descritos anteriormente) que se está utilizando en la sesión y que interpretará los comandos introducidos

```
SHELL=/bin/bash
```

PWD: Hace referencia al directorio actual de trabajo.

```
PWD=/home/egg
```

LOGNAME:Indica el login del usuario de la sesión

```
LOGNAME=egg
```

HOME: Indica el directorio base del usuario de la sesión.

```
HOME=/home/egg
```

USERNAME: Usuario de la sesión

```
USERNAME= egg
```

3 La terminal de comandos

Algunos trucos:

- TAB Completion --> Escribir en el terminal "ls /et" y pulsar la tecla TAB (tabulador).
- Historial de comandos --> En el terminal, pulsar las teclas UP y DOWN, e intentar recuperar un comando anterior del historial de comandos (incluso se puede modificar).
- CTRL + R --> Búsqueda de comandos anteriores en el historial.
- CTRL + L --> Borra la pantalla (equivale al comando `clear`).

Practiquemos un poco...

Páginas del manual de comandos (man)

Como en Linux hay muchos comandos, y es difícil recordar todas las opciones y argumentos de cada uno, existen unas páginas de ayuda, llamadas **manual** del sistema:

```
man ls

# Para salir de una página del manual, pulsamos la tecla 'q' (quit).
```

Dentro de una página del manual se pueden hacer búsquedas:

- / --> busca una cadena
- n --> va a la siguiente aparición de la cadena

También se puede buscar una palabra en todas las páginas del manual:

```
man -k inode

# Busca en todas las páginas del manual la palabra "inode".
```

Podemos buscar más información sobre todos los comandos que hemos usado hasta ahora:

```


man ls

man echo

man history

man pwd

man file

man which

```

```
# :-)
man man
```

4 Conceptos básicos sobre el Sistema de Ficheros

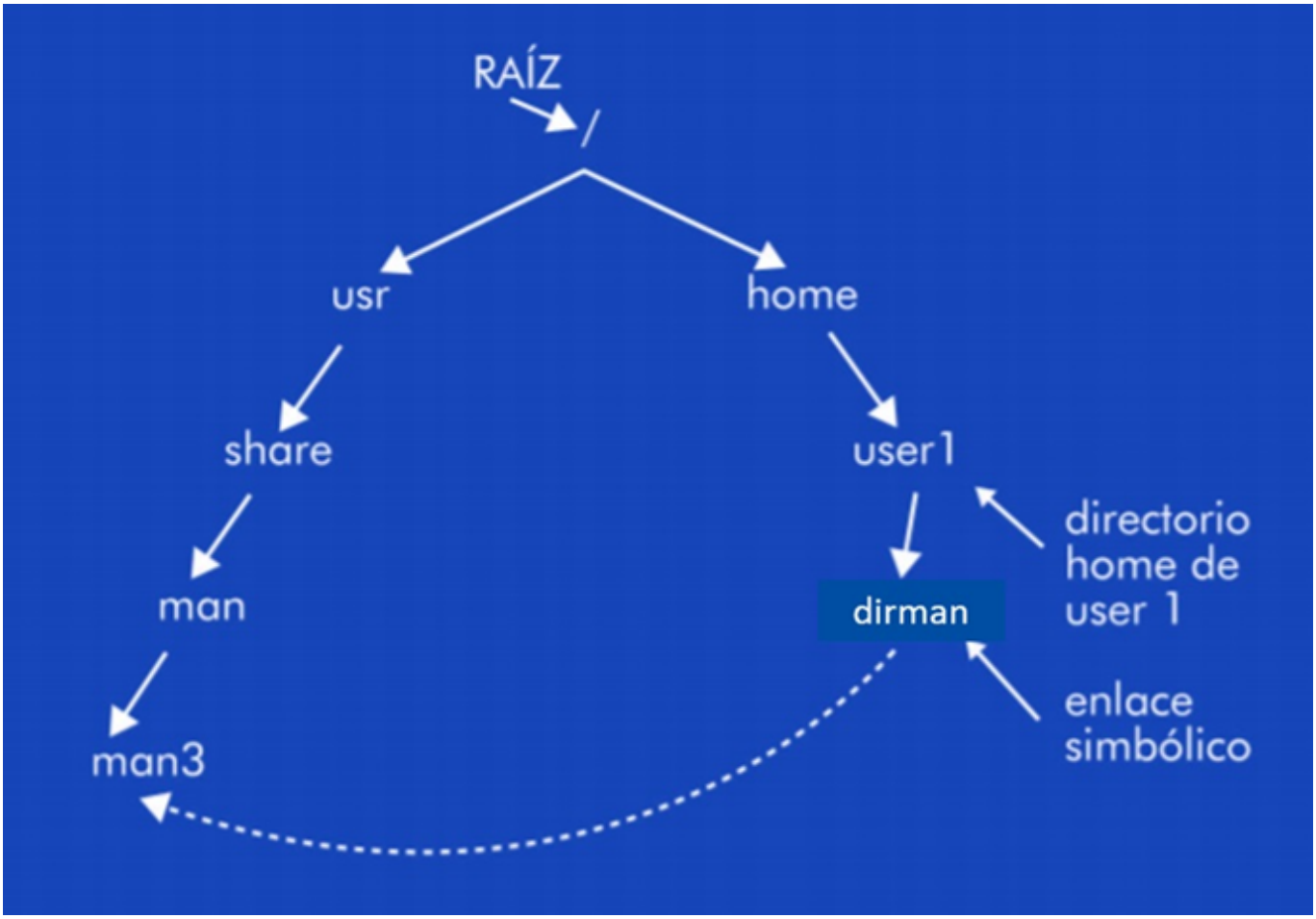
Sistema de Ficheros

Una característica muy importante de todos los sistemas operativos basados en UNIX es que **todos los ficheros y directorios así como dispositivos del sistema se pueden tratar como si fueran ficheros.** Igualmente, cuando queramos acceder al contenido de un CD, disquete o cualquier otro dispositivo de almacenamiento, deberemos montarlo en un directorio ya existente en el sistema y navegaremos por él como si se tratara de una carpeta más.

En los SO basado en UNIX, y por tanto en Linux, todo el sistema de ficheros **parte de una misma raíz, a la cual nos referiremos con el carácter "/ "**.

Es el origen de todo el sistema de ficheros y sólo existe una. Para organizar los ficheros adecuadamente, el sistema proporciona lo que llamaremos directorios (o carpetas), dentro de las cuales podemos poner archivos y más directorios. De este modo conseguimos una organización jerárquica en modo de árbol como la que vemos en la siguiente figura:

Arbol de Directorios



Cuando entramos en el sistema, el **login** nos sitúa en nuestro directorio home, **este directorio se referencia con el carácter "~".**

En todos los directorios **existe una entrada "." y otra ".."**

El punto "." es la referencia al directorio actual, mientras que **los dos puntos seguidos hacen referencia al directorio inmediatamente superior** (en el árbol de jerarquías). Cuando estamos situados en la raíz del

sistema de ficheros, la entrada ".." no existirá porque nos encontramos en el nivel superior.

Atributos de los ficheros

Antes de estudiar en profundidad los detalles del Sistema de Ficheros de Linux, aprenderemos algunos conceptos. Para ello neceistamos manejar algunos de los comandos más básicos.

Un comando nos permite ejecutar una orden a través del terminal de nuestro sistema operativo.

Comando `ls`

Nos permite ver (listar) el contenido de un directorio

```
egg@UTADLRSO:~$ ls
compartida2  Documentos  Imágenes  Plantillas  Público  vboxshared
Descargas    Escritorio  Música    prueba_dir  snap     Vídeos



# Y para ver ficheros ocultos:
ls -a

egg@UTADLRSO:~$ ls -a
.              Escritorio  .sudo_as_admin_successful
..             Imágenes    .vboxclient-clipboard.pid
.bash_history  .lesshst    .vboxclient-display-svga-x11.pid
.bash_logout   .local      .vboxclient-draganddrop.pid
.bashrc        Música      .vboxclient-seamless.pid
.cache         Plantillas  .vboxclient-vmsvga-session-tty2.pid
compartida2    .profile    vboxshared
.config        prueba_dir  Vídeos
Descargas      Público
Documentos     snap
```

Veamos otro ejemplo :

```
ls -l /home/rpalacios

total 40
drwxr-xr-x 2 rpalacios rpalacios 4096 sep  8 19:48 Descargas
-rw-rw-r-- 1 rpalacios rpalacios    0 sep 21 18:55 doc_rpalacios
drwxr-xr-x 2 rpalacios rpalacios 4096 sep  8 19:48 Documentos
drwxr-xr-x 2 rpalacios rpalacios 4096 sep  8 19:48 Escritorio
drwxr-xr-x 2 rpalacios rpalacios 4096 sep  8 19:48 Imágenes
-rw-rw-r-- 1 rpalacios rpalacios    0 sep 21 18:56 lab_SSOO
lrwxrwxrwx 1 rpalacios rpalacios   14 sep 21 18:55 manual.txt -> muydificil.txt
drwxrwxr-x 2 rpalacios rpalacios 4096 sep 21 18:54 MisDocumentos
```

```
drwxr-xr-x 2 rpalacios rpalacios 4096 sep  8 19:48 Música
-rw-rw-r-- 1 rpalacios rpalacios    0 sep 21 18:53 paraleer.txt
drwxr-xr-x 2 rpalacios rpalacios 4096 sep  8 19:48 Plantillas
-rw-rw-r-- 1 rpalacios rpalacios    0 sep 21 18:52 prueba
drwxr-xr-x 2 rpalacios rpalacios 4096 sep  8 19:48 Público
drwxr-xr-x 3 rpalacios rpalacios 4096 sep  9 18:35 snap
drwxr-xr-x 2 rpalacios rpalacios 4096 sep  8 19:48 Vídeos


__________ __ ________ _________ ____ ____________ ______________

    |      |    |     |       |       |              |-> Nombre
    |      |    |     |       |       |
    |      |    |     |       |       -> Fecha y hora de modificación
    |      |    |     |       |
    |      |    |     |       -> Tamaño en bytes
    |      |    |     |
    |      |    |     -> Nombre del grupo con permisos sobre el fichero
    |      |    |
    |      |    -> Nombre del Usuario propietario del fichero
    |      |
    |      -> Número de enlaces duros al archivo/directorio
    |
    -> Permisos del fichero:
       - Tipo de fichero (regular, directorio, enlace, etc).
       - Permisos del propietario ( r w x )
       - Permisos del grupo ( r - x )
       - Permisos para el resto de usuarios ( r - - )


Nota: La segunda columna es el número de enlaces duros al archivo. Para un
directorio, el número de enlaces duros es el número de subdirectorios inmediatos
que tiene más su directorio padre y él mismo.
```

En general, los comandos pueden tener:

- Argumentos
- Opciones

Y en función de eso generan una salida.

En general, las opciones pueden escribirse de dos formas: una forma larga, y una forma abreviada. Por ejemplo, estos dos comandos hacen lo mismo:

```
ls -a              # Forma abreviada
ls --all           # Froma larga
```

Las siguientes opciones nos listan todos los detalles de los ficheros (long list) o de forma recursiva los ficheros que existen en los directorios

```
ls -l

ls -R
```

Usando la notación abreviada, podemos agrupar varias opciones de forma cómoda:

```
ls -laR

ls -la
```

Comando `history`

Comando muy útil que nos muestra el historial de comandos

```
history
```

Para recuperar un comando del Historial de comandos --> En el terminal, pulsar las teclas UP y DOWN

Comando `pwd`

Nos permite saber en qué directorio estamos ahora mismo

```
pwd

/home/egg
```

Nota: Como se observa, las rutas en Linux se van componiendo usando la barra '/' (*slash*). En sistemas Windows, las rutas se expresan usando la barra invertida '\' (*backslash*). Por eso en Windows las rutas son del tipo 'C:\Usuarios\Jacinto Flores'.

Ejercicio: Jugar con *ls* para ver qué hay en mi directorio y en mis directorios ascendentes.

Comando `cd`

Comando utilizado para cambiar de directorio

Importante para el uso de `cd`:

- El directorio raíz `/` (*root directory*)
- ~ referencia al directorio home (~ caracter virgulilla)
- . referencia al directorio actual
- .. referencia al directorio padre

Asi como tener en cuenta las **Rutas absolutas y Rutas relativas**

```
# Absoluta
egg@UTADLRSO:~$ cd /etc

# Vamos a comprobarlo:
egg@UTADLRSO:/etc$ pwd
/etc

# Relativa
eegg@UTADLRSO:/etc$ cd ../home/egg

# Comprobamos
egg@UTADLRSO:~$ pwd
/home/egg
```

Algunas formas de usar del comando cd:

Para volver al directorio home (el inicial) hay dos opciones

```
cd
cd ~
```

Nota: Para acceder a la virgulilla en Linux, pulsar Alt + ñ o Alt + 4 (esto último tambien sirve para Windows)

Ir al directorio raiz

```
cd /
```

Volver al directorio anterior

```
cd -
```

Subir un nivel en la jerarquía de directorios

```
cd ../
cd ..
```

Subir dos niveles en la jerarquía de directorios mediante cd ../ ../

```
egg@UTADLRSO:~$  pwd
/home/egg
egg@UTADLRSO:~$  cd ../../
egg@UTADLRSO:~/$  pwd
/
egg@UTADLRSO:~$
```

Tipos de Ficheros

Es un sistema sensible a mayúsculas y minúsculas (*case-sensitive*).

¡Cuidado con los espacios y las tildes en los nombres de los ficheros y directorios!

- Comillas dobles y simples: las comillas simples, se utilizan para obtener un valor literal; en cambio las comillas dobles se utilizan para obtener un valor interpretado.
- Escape de caracteres \

Ya hemos visto que todos los directorios y ficheros en Linux son ficheros, pero entonces, ¿todos los ficheros son iguales? La respuesta es que no. Hay diferentes tipos de ficheros.

En Linux existen básicamente 5 tipos de archivos:

- **Archivos ordinarios**: Contienen la información con la que trabaja cada usuario.
- **Enlaces físicos o duros (hard links)**: No son específicamente un tipo de archivo sino un segundo nombre que se le da a un archivo.
- **Enlaces simbólicos**: También se utilizan para asignar un segundo nombre a un archivo. La diferencia con los enlaces duros es que los simbólicos solamente hacen referencia al nombre del archivo original, mientras que los duros hacen referencia al inodo en el que están situados los datos del archivo original. (Nota: i-nodo índice de una estructura de datos que contiene los metadatos de un fichero, tales como permisos de acceso, tamaño, ubicación, fecha y hora de modificación, etc). De esta manera, si tenemos un enlace simbólico y borramos el archivo original perderemos los datos, mientras que si tenemos un enlace duro los datos no se borrarán hasta que se hayan borrado todos y cada uno de los enlaces duros que existen hacia esos datos en el sistema de ficheros.
- **Directorios**. Son archivos que contienen referencias a otros archivos o directorios.
- **Archivos especiales**. Suelen representar dispositivos físicos, como unidades de almacenamiento, impresoras, terminales, etc. En Linux, todo dispositivo físico que se conecte al equipo está asociado a un archivo. Linux trata los archivos especiales como archivos ordinarios

Caracter	Tipo
-	Archivo ordinario
d	Directorio
l	Enlace simbólico
c	Archivo especial (fichero de dispositivo de caracteres)
b	Archivo especial (fichero de dispositivo de bloques)

Algunos ejemplos reales de los tipos de ficheros serían (primer carácter indica el tipo de archivo):

```
- rw-rw-r--   1     cresino     cresino   4     may 12   19:35   datos.txt
lrwxrwxrwx    1     root        root      23    may  9   19:52   vtrgb ->
/etc/alternatives/vtrgb
drwxr-xr-x    2     root        root      4096  abr 19   12:05   xml
crw-rw----    1     root        tty  7    70    may 13 09:50     vcsu6
```

Estos son algunos comandos que nos ayudaran a distinguir los diferentes tipos de archivos:

- **file**: muestra el tipo de fichero en cuanto a su categoría en el sistema de archivos (es decir, tipo de archivo y formato)
- **type** indica que tipo de "comando" es el que estamos utilizando: alias, programa incluidos en la shell...
- **which**: indica donde se encuentra el archivo binario de un ejecutable o prgrama. En el modo estándar, which responde con el primer archivo que encuentra. Si se desea, la opción -a muestra todos los archivos que cumplen con el criterio de búsqueda.

```
file /etc
file /etc/hosts
file /bin/cp

type /etc
type ls
type cp
type type

which cp
```

Estructura del Sistema de Ficheros

Como ya hemos comentado, en Linux, todo es un fichero. Los directorios son ficheros, los ficheros son ficheros, y los dispositivos son ficheros.

Los sistemas de ficheros de Linux se organizan en una estructura jerárquica, de tipo árbol. El nivel más alto del sistema de ficheros es / o directorio raíz.

Todos los demás ficheros y directorios están bajo el directorio raíz.

Ejemplo: /home/cresino/datos.txt

muestra la ruta o path absoluto del fichero datos.txt

Este fichero se encuentra en el directorio cresino que a su vez está bajo el directorio home, que por su parte está bajo el directorio raíz (/). Como se indica la ruta desde el directorio raiz(/) por esto es una ruta absoluta.

Para visualizar la estructura de árbol de un sistema de archivos Linux, podemos ejecutar el comando tree.

Comando `tree`

Permite visaulizar la estructura de árbol del sistema de ficheros de Linux. No viene instalado conla distribución.

```
tree
“No se ha encontrado la orden «tree», pero se puede instalar con: sudo apt install
tree”


sudo apt-get install tree
[sudo] contraseña para usuario:
```

Una vez que se introduce la constraseña correcta del usuario (con permisos) se comienza a descargar e instalar el programa “tree”.

```
tree
locales-launch: Data of es_ES locale not found, generating, please wait...
locales-launch: Data of es_ES locale not found, generating, please wait...

.

├── datos.txt

├── Descargas

├── Documentos

├── dump-gestion_pedidos-202303061206.sql

├── dump-gestion_pedidos-202303061407.sql

├── Escritorio

├── Imágenes

├── laboratorio

│   ├── PRUEBA

│   ├── trafico2.dat

│   └── trafico.dat

├── Música

├── Plantillas

├── Público
```

```
├── Scripts
│   └── script1_gp.sql
├── snap
│   ├── firefox
│   ├── snapd-desktop-integration
│   └── tree
│       ├── 18
│       ├── common
│       └── current -> 18
├── vboxshared
└── Vídeos
```

Directorios especiales en Linux

Un directorio en linux es un fichero especial, que puede contener otros ficheros y sub-directorios.

El comando file nos indicará si un fichero es un directorio

```
file /etc
```

Ya lo hemos visto al ver el comando `ls`, lo volvemos a recordar:

- El directorio raíz `/` (*root directory*)
- El directorio de trabajo del usuario con el que nos hemos registrado "~" Rutas relativas:
- "." referencia al directorio actual en el que me encuentro
- ".." referencia al directorio padre al actual o directorio inmediatamente superior al directorio en el que me encuentro. Se denomina directorio padre.

Directorios importantes del Sistema

En la mayoría de distribuciones basadas en GNU/Linux se siguen unas recomendaciones para la distribución de directorios que son propios del sistema operativo, encontrando los siguientes directorios principales:

Directorio	Descripción
/	Es el directorio raíz, todos los demás directorios cuelgan de este directrio. Suele ser el directorio donde se monta el directorio principal.
/root/	Directorio home para el root del sistema.

Directorio	Descripción
/home/	Contiene los directorios o carpetas de los usuarios.
/boot/	Archivos estáticos necesarios para el arranque del sistema.
/bin/	Aquí están localizados los ficheros binarios ejecutables del sistema y comandos básicos para todos los usuarios del sistema.
/sbin/	Aquí se guardan solo los ejecutables del sistema que debe ejecutar únicamente el usuario root.
/etc/	Archivos de configuración de las aplicaciones y servicios instalados en el sistema.
/lib/	Librerías esenciales para el núcleo del sistema y módulos del mismo.
/dev/	Archivos de dispositivos. Este directorio contiene todos los dispositivos de almacenamento detectados, en forma de archivos, conectados desde que el sistema está arrancado, es decir, cualquier disco duro conectado, partición, memoria USB, o CDROM conectado al sistema y que el sistema pueda entender como un volumen lógico de almacenamiento
/mnt/	Puntos de montaje para almacenamiento.
/media/	Puntos de montaje temporal para dispositivos o medios extraibles.
/proc/	Se monta desde la memoria RAM. Es un directorio dinámico (virtual) especial que mantiene información sobre el estado del sistema, incluyendo el hw, las variables y los procesos actualmente en ejecución.
/opt/	Directorio donde instalar aplicaciones opcionales.
/tmp/	Archivos temporales. Según la distribución utilizada (o la configuración que utilicemos) se borran al arrancar el sistema o cada cierto período de tiempo. Es un directorio cargado en memoria.
/usr/	Estructura jerárquica, utilizada para almacenar todo el software instalado en el sistema al que pueden acceder la mayoría de los usuarios. Software instalado mediante el Gestor de Paquetes de la distibución. Dentro tiene directorios como "bin", "lib", etc. en los que se guardan ficheros y librerías que no son del sistema.
/var/	Directorio para ficheros variables como los trabajos de impresión, ficheros de log, etc. Es muy recomendable conservar y no eliminar.
/sys	Archivos del sistema y de dispositivios HW. Muy parecido a /proc. Sólo existe en memoria.
/srv	Puede contener ficheros y directorios de los servicios ofrecidos pore el sistema: ftp, http, etc..
/lost+foud	"perdido y encontrado". Se guardan ficheros recuperados debajo de /.

.

.

Wildcards

Los ***wildcards*** (o comodines) nos permiten seleccionar nombres de ficheros basado en patrones de caracteres.

Wildcard	Significado
*	Selecciona cualquier conjunto de caracteres.
?	Selecciona un carácter cualquiera.
[caracteres]	Selecciona cualquier carácter que esté incluido en un conjunto de caracteres específicos. Pueden expresarse como una *clase de caracteres POSIX*
[!caracteres]	Selecciona cualquier carácter que **NO** esté incluido en un conjunto de caracteres específicos. También pueden expresarse como una *clase de caracteres POSIX*

- **Nota: POSIX** (Portable Operating System Interface) define un conjunto de estándares especificados por el IEEE para varias interfaces de herramientas, comandos y API para garantizar la compatibilidad entre sistemas operativos.

.

Estas **Clases de caracteres POSIX** son las siguientes:

Clase	Significado
[:alnum:]	Selecciona caracteres alfanuméricos.
[:alpha:]	Selecciona caracteres alfabéticos.
[:digit:]	Selecciona caracteres numéricos.
[:upper:]	Selecciona caracteres alfabéticos en *MAYÚSCULAS*.
[:lower:]	Selecciona caracteres alfabéticos en *minúsculas*.
[:xdigit:]	Selecciona dígitos hexadecimales [0-9A-Fa-f].
[:space:]	Selecciona espacio, tabulador, tabulador vertical, retorno de carro, salto de línea.
[:blank:]	Selecciona espacio y tabulador.
[:print:]	Selecciona caracteres imprimibles.
[:punct:]	Selecciona símbolos de puntuación: ! ' # S % & ' ( ) * + , - . / : ; < = > ? @ [/] ^ _ {
[:word:]	Selecciona cadenas de caracteres alfanuméricas, incluyendo los guiones bajos (*underscores*)
[:ascii:]	Selecciona cualquier caracter ASCII (rango 0-127).
[:cntrl:]	Selecciona caracteres de control, es decir, cualquier carácter que no pertenenzca a ninguna de las clases de caracteres POSIX que aparecen en esta tabla.
[:graph:]	Selecciona cualquier caracter imprimible que no pertenezca a la clase [:space:].

.

Veamos algunos ejemplos para seleccionar nombres usando patrones de búsqueda y *wildcards*:

Patrón	Significado
*	Selecciona todos los nombres de fichero.
g*	Selecciona los que empiezan con el carácter 'g'.
b*.txt	Selecciona los que empiezan con el carácter 'b' y terminan con la cadena ".txt".
Data???	Selecciona los que empiezan con la cadena "Data" y tienen exactamente 3 caracteres más.
[abc]*	Selecciona los que empiezan con el carácter '**a**', o con el '**b**' o con el '**c**', seguidos de cualquier otra cadena.
[[:upper:]]*	Selecciona los que empiezan con una MAYÚSCULA seguida de cualquier otra cadena.
BACKUP.[[:digit:]] [[:digit:]]	Selecciona los que empiezan con la cadena "BACKUP." seguida por exactamente dos números.
*[![:lower:]]	Selecciona cualquier fichero que no termine en una letra en minúscula.

Comandos básicos de Linux

`cd` : Cambiar de directorios

Cuando usamos el comando cd le podemos pasar como parámetro la ruta a dónde queremos movernos en el árbol de directorios, y **dicha ruta se le puede indicar de forma absoluta o de forma relativa.**

De **forma relativa significa que partiremos del directorio donde estamos** en el momento de ejecutar el comando y de forma **absoluta siempre partiremos de la raíz "/".**

Por ejemplo: si estamos en el directorio /usr/bin/ y queremos ir al /etc/, deberíamos introducir el siguiente comando:

```
#Forma relativa:
cd ../../etc
los dos primeros puntos indican /usr/ y los siguientes la raíz "/" del sistema, a
partir de la cual ya podemos acceder a /etc/

#Forma absoluta:
cd /etc

#Para saber en qué directorio estamos, podemos utilizar el comando
pwd
```

```
cd /home/rpalacios
pwd
man cd
```

`mkdir` : Creación de directorios

```
cd ~
mkdir projects
ls -la projects

# ¿Que pasa si hacemos:?
mkdir projects/games/super_pang

# La opción -p crea los padres (ancestros) si no existen
mkdir -p projects/games/super_pang
mkdir -p projects/games/counter_strike
ls -laR projects

# Practiquemos:

cd projects/games/
pwd
cd ../
pwd
cd ~
pwd
cd projects/games
pwd
cd ../..

man mkdir
```

`rmdir` : Borrado de directorios

```
cd ~

# Probemos el siguiente comando
rmdir projects/games

# rmdir SOLO FUNCIONA SI EL DIRECTORIO ESTÁ VACIO
rmdir projects/games/counter_strike
rmdir projects/games

# Para borrar directorios que no estén vacios se utiliza `rm`
mkdir projects/games/counter_strike
rm -rf projects


man rmdir
```

`touch` : Creación y actualización de ficheros

```
cd ~
touch projects/description.txt
touch projects/Description.txt


man touch
```

`cat` : Mostrar el contenido de un fichero

```
cat /etc/profile


man cat
```

`less` : Mostrar el contenido de un fichero, permitiendo interacción dinámica

Less muestra el fichero en modo interactivo de forma que se puede interactuar pero no editar ni modificar.

```
less /etc/profile

- ESPACIO (ir al final)
- ENTER (avanza una linea)
- b o Ctrl-b Scroll backward (hacia atras) el tamaño de una ventana
- d o Ctrl-D Scroll forward (hacia adelante) el tamaño de la ventana
- G (ir al final)
- r o Ctrl-R refresca la pantalla y descarta lo que tuiera en el buffer.  Útil si
el fichero puede estar cambiando mientras se visualiza
-  /palabra (destacar plalabra)
- n (repite la última búsqueda)
- q (salir)

man less
```

`tail` : Mostrar las últimas 10 líneas de un fichero

- -n muestra las últimas n líneas del fichero
- -f muestra las últimas líneas del fichero en tiempo real (para visualiazción de cambios en el fichero)

```
tail /etc/rsyslog.conf
tail -n 5 /etc/rsyslog.conf
tail -f /var/log/syslog


man tail
```

head : Mostrar las primeras líneas 10 de un fichero (cabecera)

```
head /etc/rsyslog.conf
head -n 5 /etc/rsyslog.conf


man head
```

echo : Imprimir en la consola y en ficheros

```
cd

# Imprimir un texto en la salida estándar (típicamente la consola):
# opción -e: habilita la interpretación de los saltos backslah '\'

echo "aprendemos Linux!"
echo "saltos de línea \n\n\n"
echo -e "saltos de línea \n\n\n Bien!"

# Imprimir un texto en un fichero (sobreescribe el fichero,
# y lo crea si no existía):
echo "nuevo contenido" > mi_fichero.txt

# Por defecto, echo incluye un carácter de final de línea, que en Linux es '\n'.
# Se puede omitir la inserción de un carácter '\n' al final de la líne (-n)
echo -n "El prompt aparecerá justo a la derecha de este texto"


man echo
```

ln : Enlaces simbólicos

Son parecidos a los accesos directos de Windows (los que solemos tener en nuestro escritorio).

Un link simbólico es un “enlace o puente” a un archivo o directorio perteneciente al sistema; una referencia que podemos poner en cualquier sitio y que actúa como un acceso directo a cualquier otro.

Este mecanismo nos permite acceder a carpetas o archivos de forma más rápida y cómoda, sin tener que desplazarnos por la jerarquía de directorios o evitar la necesidad de tener archivos duplicados cuando queremos utilizarlos desde varios directorios. **Si borrásemos el fichero destino, el enlace no apuntaría a ninguna parte.**

```
# Veamos si tenemos algun enlace simbólico:
ls -l /boot

ls -l /lib
```

Veamos cómo crear enlaces (sintaxis):

ln -s <destino> <enlace_simbolico>

- Siendo <destino> el nombre del fichero existente al que se crea el enlace
- Siendo <enlace_simbolico> nombre del enlace simbólico

```
cd ~

touch document.txt
mkdir softlinks
cd softlinks

#Crea un enlace simbolico al fichero que se le pasa como primer argumento.
ln -s ../document.txt

# Crea un enlace simbólico(arg2) al fichero que se pasa en como arg1
ln -s /home/rpalacios/document.txt ./my_document.txt


# NOTA: no se pueden usar rutas relativas en el segundo argumento. Como norma
general y para no cometer errores se aconseja ejecutar el comando ln -s siempre en
el directorio en el que se quiere crear el enlace.

ls -l
man ln

# AVISO: No se pueden crear enlaces simbólicos desde el S.O. invitado en las
carpetas compartidas creadas con VirtualBox.
# La opción está deshabilitada por defecto por motivos de seguridad
```

cp : Copiar ficheros (y directorios)

```
# Copia simple de ficheros (orgien -> destino):
cd ~
touch flower.txt

cp flower.txt flower_ORIG.txt
cp /etc/rsyslog.conf ~/rsyslog_FOR_REVIEW.conf
cp /etc/rsyslog.conf ~
ls -l
cat rsyslog_FOR_REVIEW.conf

# ¡CUIDADO! cp sobreescribe el fichero si ya existía:
cp /etc/fstab ~/rsyslog_FOR_REVIEW.conf
ls -l
cat rsyslog_FOR_REVIEW.conf

# Podemos usar la opción -i para pedir confirmación al usuario:
```

```
cp -i /etc/fstab ~/rsyslog_FOR_REVIEW.conf

# Copia un fichero a un directorio destino. El fichero en el destino mantiene el
nombre del fichero original:
mkdir copias
cp /etc/rsyslog.conf ./copias/

# También podemos especificar un nombre para el fichero destino:
cp /etc/rsyslog.conf ./copias/rsyslog_COPY.conf

ls -l

# Copia un directorio (y todo su contenido) en otro directorio:
cp -R copias  copias_2

ls -laR copias_2

# Podemos copiar muchos ficheros cuyos nombres sigan un patrón (usando wildcards):
touch presupuesto_1.txt
touch presupuesto_2.txt
touch presupuesto_3.txt

mkdir documentos_de_texto

cp *.txt documentos_de_texto


man cp
```

`mv` : Mover/Renombrar ficheros y directorios

```
# Como Renombrar ficheros:
cd
touch plato1.txt

# Renombro un fichero
mv plato1.txt plato2.txt

# Como Mover ficheros o directorios a otro directorio:
touch tipos_de_cubiertos.txt
mkdir recetas
mkdir cosas_cocina

ls -l

#Muevo un fichero a un directorio
mv tipos_de_cubiertos.txt cosas_cocina

#Muevo un directorio a otro directorio
mv recetas cosas_cocina
```

```
ls -l
ls -l cosas_cocina

# Mover varios ficheros/directorios a la vez:
touch tipos_de_sartenes.txt
touch tipos_de_cazuelas.txt
touch tipos_de_vajilla.txt

ls -l

mv tipos_de_sartenes.txt tipos_de_cazuelas.txt tipos_de_vajilla.txt cosas_cocina

ls -l
ls -l cosas


man mv
```

`rm` : Eliminar ficheros (y directorios)

```


# Borrar ficheros:
touch document_01.txt
touch document_02.txt
touch document_03.txt
touch document_04.txt

ls -l

rm document_01.txt

ls -l

rm document_0?.txt

ls -l

# Borrar directorios:
mkdir rpalacios

# Para directorios vacios:
rmdir rpalacios

mkdir rpalacios
touch rpalacios/datos.txt

ls -l rpalacios

# Para directorios que tienen ficheros debemos usar -r:
rm -r rpalacios
```

```
ls -l rpalacios
ls -l

# ¡CUIDADO! Cuando borramos algo con rm, se borra para siempre.


man rm
```

5 Comandos Avanzados sobre Ficheros

`truncate`: Permite variar el tamaño de un fichero

Se usa fundamentalmente para vaciar el contenido de un fichero

```
# Con la opción -s establecemos el tamaño del fichero
history > mi_fichero.txt
ls -l

# Con tamaño 0 vaciamos el fichero
truncate -s 0 mi_fichero.txt
ls -l

# Y hay más formas de vaciar un fichero....

#Redirigir "nada" a un fichero
> mi_fichero.txt

# echo sin ningún parametro y sin nueva linea al final (-n)
echo -n > mi_fichero.txt


#true  no  hace  nada  excepto  devolver  un valor de salida de 0, significando
"éxito".
true > mi_fichero.txt


#/dev/null o «null device», es un archivo especial que descarta toda la
información que se escribe en o se redirige hacia él. A su vez, proporciona ningún
dato a cualquier proceso que intente leer de él, devolviendo simplemente un EOF o
fin de archivo.
cp /dev/null mi_fichero.txt


man truncate
```

`grep` : Búsqueda de textos en ficheros y a la salida de comandos

```
# Buscar una expresión en un fichero
grep "elenagg" /etc/group
grep "elenagg" /etc/passwd
grep $(whoami) /etc/passwd
```

sintaxis grep

```
grep -r busca recursiva
grep -n muestra numeros de línea
grep -i case insensitive (mayusculas o minusculas)
grep -c cuenta las veces que aparece una palabra dentro de un archivo
grep -v excluir una expresión de busqueda
grep -e -e busqueda de varias expresiones
grep "^cadena" buscar expresiones que empiecen por cadena
grep "cadena$" buscar expresiones que acaben en cadena


# Buscar una expresión en múltiples ficheros, de forma recursiva (-r)
cd /etc
grep -r "elenagg" *


# Buscar una expresión en un fichero mostrando los números de línea (-n)
grep -rn "elenagg" *
cat /etc/group | grep -n "elenagg"


# Buscar una expresión a la salida de otro comando
date --help
date --help | less

date --help | grep "%" (muestra los modificadores  de date %xxx )
```

```
# Nota previa:
# Comando ps: muestra procesos en ejecución. Opciones:
# Opcion a: muestra los procesos para todos los usuarios (todos los terminales)
# Opción u: muestra información adicional
# Opción x: muestra información de procesos sin terminal (demonios)

# ps (muestra los procesos de usuario)
# ps u (informacion de usuario extendida)
# ps aux (a: todos los usuarios, x: procesos sin terminal -demonios-, u: extendida
)

# Buscar sin considerar las MASYÚSCULAS/minúsculas (-i)
ps aux
ps aux | grep -i "Cron"

cat /etc/group | grep -ni "elenagg" | less


# Excluir una expresión en una búsqueda (-v)
# Ejecutaremos primero
ps aux | grep -i "cron"

# Y ahora:
ps aux | grep -i "cron" | grep -v "grep"
```

```
#Con la opción -i ignora MAY/Mins, es decir, ignora la cadena a pesar de escribir
"Grep"
ps aux | grep -i "cron" | grep -vi "Grep"

# Excluir varias palabras de una búsqueda ( -v: excluye, -e: expresion)
cat /etc/passwd

grep -v -e "elenagg" -e "nologin" /etc/passwd

#Otro formato, con las expresiones separadas por el símbolo "|"
grep -vE "elenagg|nologin" /etc/passwd

history | grep "ls" | grep -v -e "-l" -e "-a" -e "~"

# Ver que pasa si usamos este formato:
history | grep "ls" | grep -vE "-l|-a|~"

# Hay que 'escapar' el primer signo -
history | grep "ls" | grep -vE "\-l|-a|~"

#---

cat /etc/passwd | grep "elenagg"

# Buscar líneas que empiecen con un patrón (^)
cat /etc/passwd | grep "^elenagg"

# Buscar líneas que terminen con un patrón ($)
cat /etc/passwd | grep "elenagg$"


man grep
```

`find` : Buscando ficheros dentro de un directorio (y subdirectorios)

Sintaxis básica

find directorio_de_busqueda opciones termino_a_buscar

- **directorio_de_busqueda**: es el punto de origen desde donde deseas iniciar la búsqueda. Puede ser el directorio raíz "/", el directorio actual ".", el directorio de trabajo "~" o cualquier otro diectorio.
- **opciones**: es el filtro a utilizar para buscar el archivo. Podría ser el nombre, tipo, fecha de creación o de modificación del archivo, etc.
- **termino_a_buscar**: especificará el término de búsqueda relevante

.

Buscar un fichero por su nombre, opción: -name

```
cd

# Buscar un nombre de **fichero** por su nombre (-name)
find . -name "doc1.txt"
find . -name "doc*.txt"


# Buscar un nombre de fichero sin diferenciar entre MAYÚSCULAS/minúsculas (-iname)
find . -iname "DOC*.txt"

# Buscar todos los nombres de fichero excepto los que se indiquen (-not)
find . -not -name "doc*.txt"

# Buscar y borrar (¡CUIDADO!)
find . -name "mi_fichero.txt" -delete
```

Buscar un fichero por el tipo, opción: -type

```
# Buscar por tipo de fichero (-type [f (fichero normal) d(directorio) l(enlace
simbólico)])
# Muestra todos los directorios del sistema
find / -type d

# Muestra los directorios del home del usuario
find ~ -type d

# Se pueden combinar opciones
# Busca ficheros que cumplen un patron y excluye directorios y enlaces
find ~ -type f -name "doc*"

# Busca directorios que cumplen un patrón excluyendo ficheros y enlaces
find ~ -type d -name "doc*"
```

Buscar por tamaño (-size)

```
# El tamaño debe ir acompñado de la "magnitud" -> (c:caracter/byte, k:kilobytes,
M:megabytes, G:gigabytes, b:bloques de 512 bytes)

# Buscar por tamaño (exacto) (-size)
# Usando bloques de 512 y redondeando
find ~ -size 10M

## Otras búsquedas por tamaño
find ~ -size +5G
find ~ -size -1k
```

Buscar por propietario o grupo (-user y -group)

```
ls -l

# Buscar por propietario o grupo (-user y -group)
find /tmp -user elenagg
find /tmp -group elenagg
```

Buscar por permisos

```
# Búsqueda por permisos (exacta).
find ~ -perm 644

# Búsqueda por permisos (como mínimo 644).
find ~ -perm -644
```

Otras opciones de búsqueda

```
# Algunas otras opciones útiles
## Buscar ficheros y directorios vacíos
find ~ -empty

## Buscar ejecutables
find / -executable

## Buscar legibles
find / -readable


man find
```

awk : Una herramienta avanzada para el tratamiento de textos

awk es en realidad un lenguaje de programación interpretado. *awk* lee una línea de la entrada y la va procesando (procesa línea a línea).

Todas las instruciones que hacemos con *awk* se van aplicando secuencialmente a los datos de entrada, es decir, AWK asume que va a tener que procesar un flujo de datos (entrada estandar, fichero de texto, tubería) y que este flujo está medianamente estructurado en registros (lineas) y campos (columnas). Por tanto, lee cada una de sus lineas como si fuese un registro, separa ese registro en campos, y hace lo que se le ordene con esos campos para finalmente producir un flujo de salida.

Es una herramienta muy potente, pero con cierta complejidad. Aquí sólo se muestran unos pocos usos comunes.

Su utilización estándar es la de filtrar ficheros o salida de comandos de Linux, tratando las líneas para, por ejemplo, mostrar una determinada información sobre las mismas

Se puede considerar que AWK tiene 2 partes principales: **el patrón y la acción**. Para ver como funciona lo veremos con un ejemplo.

Sintaxis básica

awk [opcion] [patron] {accion} nombre_fichero

```
# Mostrar el contenido de un fichero
cd
echo "1) John       Maths         6.54" > notes.txt
echo "2) Paul       Physics       8.23" >> notes.txt
echo "3) Michael    Biology       7.98" >> notes.txt
echo "4) Steve      Physics       5.10" >> notes.txt
echo "5) Jack       History       4.68" >> notes.txt
echo "6) Richard    Programming   9.05" >> notes.txt
echo "7) John       Programming   8.42 - this note should be reviewed" >>
notes.txt

ls -l
cat notes.txt


# En este caso sólo hay una acción "print" de todas las líneas de un fichero
awk '{print}' notes.txt

#En este caso se imprimen las líneas del fichero que cumplen el patrón:
awk '/Phy/ {print}' notes.txt
```

Una de las peculiaridades qde AWK es que al leer cada linea da valores a una serie de variables predefinidas:

- NR es el número de la linea que ha leido
- NF es el número de campos en la linea que ha leido
- $0 contiene toda la linea leida
- $1, $2, ... $NF representan cada uno de los campos leidos

Por ejemplo la orden:

```
# muestra en pantalla las lineas del fichero numeradas.
awk '{print NR,$0 }' notes.txt
```

NOTA: El separador de campos es el espacio o el tabulñador.

Veamos más ejemplos:

```
# Imprimir columnas (o campos)
# \t representa un carácter de tabulador, y se suele usat para igualar los límites
de la línea de salida.
# Los números precedidios del caracter "$" representan las posiciones de las
columnas de la línea de entrada.
# $1 es para el primer campo, $2 es para el segundo campo, etc
# E $0 es para toda la línea

awk '{print $1}' notes.txt
awk '{print $2"\t"$3}' notes.txt
awk '{print $2"\t\t"$3}' notes.txt

# Imprimir las filas que cumplan un patrón
awk '/Phy/ {print}' notes.txt

# Imprimir las columnas que cumplan un patrón
awk '/Phy/ {print $1}' notes.txt
awk '/Phy/ {print $2"\t\t"$3}' notes.txt
```

NOTA: Aparte de permitirnos definir los patrones que queramos, awk introduce dos patrones especiales **BEGIN** y **END**. El primero permite definir acciones que se ejecutarán antes de empezar a procesar el fichero de entrada, el segundo acciones que se ejecutaran al final del proceso.

Ejemplos:

```
# Contar e imprimir el número de filas que cumplen un patrón
awk 'BEGIN {print "Alumnos con asignatura de Physics"} /Phy/{++counter} END {print
"Valor = ", counter}' notes.txt


# Acepta una sere de funciones de manejo de Cadenas. Veamos  ocmo imprimir las
filas que superan los 35 caracteres:
awk 'length($0) > 35' notes.txt



# Se puede especificar otro carácter como separador de campos (opción -F).
Recuerda que el separador por defeco es el espacio o el TAB.
echo "10;20;30;40;50;60;70" > scale.csv
cat scale.csv | awk -F ";" '{print $1}'
cat scale.csv | awk -F ";" '{print $1"\t"$2"\t"$7}'



# Es muy útil para tratar la salida de otros comandos
# Para imprimir tamaño de ficheros:
```

```
ls -la | awk '{print $5 "\t" $9}'


# Imprimir los ficheros cuyo tamaño es mayor de 100 en un fichero
ls -l|awk '$5>100{print $0}' > grandes.txt


man awk
info awk
```

`sed` : Edición de textos desde el terminal

`sed` es un editor de flujos (**s**tream **ed**itor) utilizado para transformación de cadenas. Al igual que *awk* trata cada línea de su flujo de entrada. Por eso se suele usar para editar ficheros o leer la salida de algún comando, sustituyendo un texto (o patrón) por otro.

Sintaxis básica

- sed 's/cadena_original/cadena_nueva/' fichero_de_entrada
- sed 's/cadena_original/cadena_nueva/' < fichero_de_entrada
- sed 's/cadena_original/cadena_nueva/' < fichero_de_entrada > fichero_salida
- resultado_comando | sed 's/cadena_original/cadena_nueva/'

```
# Sustitución de un patrón por otro patrón
echo "Es un poema de Antonio" | sed 's/Antonio/Federico/'
echo "Es un poema de Antonio" | sed 's/nton/urel/'

# ¡Atención! Por defecto, sed sólo modifica la primera ocurrencia del
# patrón:
echo "one two three, one two three one two three" > example.txt
echo "four three two one four three two one" >> example.txt
echo "one hundred" >> example.txt
cat example.txt

cat example.txt | sed 's/one/ONE/'

# Para reemplazar TODAS las ocurrencias, se usa la opción de reemplazo global:
cat example.txt | sed 's/one/ONE/g'

# Para reemplazar a partir de una ocurrencia combinamos la posición y la letra g.
En el ejemplo se sustituyen la ocurrencia 2, 3, 4, etc..
cat example.txt | sed 's/one/ONE/2g'

# Se puede especificar otro delimitador diferente a '/'
echo 'PATH=$PATH:/usr/local/bin' > wrong_path.txt
cat wrong_path.txt

# Si la cadena a reemplazar contiene '/', tenemos que escapar cada  slash'/'
mediante un backslash '\' para que sed no las confunda con los separdores que
necesita
#Vamos a cambiar en wrong_path.txt  /usr/local/bin por /opt/bin:
```

```
cat wrong_path.txt | sed 's/\/usr\/local\/bin/\/opt\/bin/'

# Esto se puede evitar cambiando el símbolo delimitador usado por "sed" (que es el
"/"). En los siguientes ejemplos se ha sustituido por "_" ":" "\" o un "+":

cat wrong_path.txt | sed 's_/usr/local/bin_/opt/bin_'
cat wrong_path.txt | sed 's:/usr/local/bin:/opt/bin:'
cat wrong_path.txt | sed 's|/usr/local/bin|/opt/bin|'
cat wrong_path.txt | sed 's+/usr/local/bin+/opt/bin+'

# sed se usa mucho para modificar ficheros ya existentes
## Generando un nuevo fichero con los cambios:
sed 's/\/usr\/local\/bin/\/opt\/bin/' < wrong_path.txt > right_path.txt
cat wrong_path.txt
cat right_path.txt

## Sobreescribiendo el fichero original, pero con los nuevos cambios: (-i)
cat wrong_path.txt
sed -i 's/\/usr\/local\/bin/\/opt\/bin/' wrong_path.txt
cat wrong_path.txt

# Se puede usar sed para modificar varias expresiones simultáneamente (opción -e):
cat example.txt | sed -e 's/one/ONE/' -e 's/two/TwO/'
cat example.txt | sed -e 's/one/ONE/g' -e 's/two/TwO/'
cat example.txt | sed -e 's/one/ONE/' -e 's/two/TwO/g'
cat example.txt | sed -e 's/one/ONE/g' -e 's/two/TwO/g'

# También se pueden modificar sólo las líneas que además contengan otro
# patrón:
cat example.txt | sed '/hundred/s/one/ONE/'

# Se pueden buscar patrones para reemplazar, independientemente de las
# MAYÚSCULAS/minúsculas:
cat example.txt | sed 's/oNe/ONE/I'

# Por último, se pueden agrupar varias opciones a la vez:
cat example.txt | sed -e 's/oNe/ONE/Ig'

# Como sustituir todos los espacios en blanco por una ','
echo "esto es un texto de prueba" > fichero.txt
sed 's/\s/,/g' fichero.txt

# Y si hay varios espacion en blanco.....
# \s: "espacio en blanco"
# \+: más de un espacio (es necesario "escapar" el caracter +)
echo "esto     es    un     texto     de     prueba" > fichero2.txt
sed 's/\s\+/,/g' fichero2.txt

man sed
info sed
```

`tr` : "Translate" o Traductor (reemplazo o eliminación de caracteres).

El comando `tr` es un "traductor": sirve para **reemplazar o eliminar** un conjunto de caracteres por otro conjunto de caracteres.

Sintaxis básica

tr [opciones]... CONJUNTO1 [CONJUNTO2]

Donde CONJUNTO1 Y CONJUNTO2 son ambos secuencias de caracteres definidas explícitamente o conjuntos predefinidos por este comando.

Parámetros:

- "-c": Usa el **complemento** del CONJUNTO1. Esto significa que define al CONJUNTO1 como todos los caracteres que no se encuentran en la definición dada por el Usuario. Este parámetro es útil para indicar caracteres que no queremos que se vean afectados.
- "-d": Borra los caracteres definidos en CONJUNTO1.
- "-s": (squeeze-repeats): Elimina la secuencia continua de caracteres repetidos, definidos en el CONJUNTO1.

Conjuntos de caracteres POSIX (algunos de los más comunes):

- [:alnum:] : Las letras y Dígitos.
- [:alpha:] : Letras.
- [:digit:] : Dígitos.
- [:graph:] : Caracteres imprimibles, sin incluir el Espacio en Blanco.
- [:print:] : Caracteres imprimibles, incluyendo el Espacio en Blanco.
- [:lower:] : Letras minúsculas.
- [:upper:] : Letras mayúsculas.
- [:punct:] : Signos de puntuación.
- [:space:] : Los Espacios en Blanco (horizonatales y verticales).

```
echo "Cambiemos espacios por tabuladores" | tr ' ' '\t'

# Como se puede ver, se pueden usar clases de caracteres POSIX
echo "mi nombre es Ricardo" | tr [:lower:] [:upper:]

# Y es útil, porque aunque se puede especificar un conjunto completo de
caracteres, no siempre es elegante
echo "mi nombre es Ricardo" | tr abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ

#Incluso codificar mensajes secretos
echo "mi nombre es elena" | tr abcdefghijklmnopqrstuvwxyz
ZYWVUTSRQPONMLKJIHGFEDCBA > secreto1.txt
cat secreto1.txt
cat secreto1.txt | tr ZYWVUTSRQPONMLKJIHGFEDCBA abcdefghijklmnopqrstuvwxyz

# Se pueden usar intervalos. En este ejemplo, además, se ejecuta tr en modo
interactivo. Usar ^C para salir.
tr a-z A-Z
```

```
mi nombre es Ricardo


# Se puede utilizar tr de fomra encadenada
echo "mi nombre es Ricardo, profesor de la asignatura" |tr ' ' z |tr a-z b-z |  tr
b-z a-z | tr y ' '

# ¿Y si hay muchos espacios entre palabras?
echo "Cambiemos  espacios     por tabuladores" | tr [:space:] '\t'

# Podemos ignorar las repeticiones de los caracteres que son iguales y van
seguidos y dejar uno solo (-s)
echo 'Este es un simple ejemplo!!!.... Algo práctico tal vez???' | tr -s '!¡.?¿'

# Podemos sustituir las repeticiones de los caracteres que son iguales y van
seguidos (-s)
# El flag -s es útil para "comprimir" los caracteres repetidos seguidos
echo "Cambiemos  espacios         por tabuladores" | tr -s [:space:] '\t'
echo "ssss BBB ss BBBBB sssss" | tr -s 's' 'a'

# Más ejemplos
echo "Comprimimos        los      espacios    :)" | tr -s [:space:]
echo "Comprimimos        los      espacios    :)" | tr -s [:space:] ' '

# En lugar del comodín [:space:] también puedo usar el carácter espacio
directamente
echo "Comprimimos        los      espacios    :)" | tr -s ' '

# También permite suprimir o borrar ciertos caracteres (-d)
echo "Borrremos todas las errrres" | tr -d 'r'

# Eiminemos números
echo "Mi DNI es 555424242A" | tr -d [:digit:]

# De nuevo, se pueden usar intervalos
echo "Mi DNI es 555424242A" | tr -d 0-9

# Se puede usar el complemento de un conjunto (-c), por ejmeplo para borrar
# Imprimir solamente los números (borro todo lo que no son digitos)
# La opción -c indica que se toma el Complement de digit (todo lo que no sean
dígitos)
echo "Mi DNI es 555424242A" | tr -cd [:digit:]

# Borro lo que no son números ni espacios
echo "Mi DNI es 555424242A" | tr -cd [:digit:][:space:]

# Para eliminar todos los caracteres repetidos se hace uso de -s con el
complemento (-c) del Conjunto "":
echo 'Esteeee es ooootro simple ejemplo!!!.... ' | tr -c -s ""

# El complemento (-c)  se puede utilizar para eliminar todos los caracteres no
imprimibles de un fichero (ej: saltos de línea \n, tabuladores: \t, etc)
# Utilizamos primero el comando `echo`con la opción -e para interpretar los saltos
de línea y los tabuladores:
```

```
echo -e "Hola,\n\n\t\t mundo"
echo -e "Hola,\n\n\t\t mundo" | tr -cd [:print:]

# Otra forma enviando antes la información a un fichero
echo -e "Hola,\n\n\t\t mundo" > non_printable.txt
cat non_printable.txt

# Paso el fichero como entrada del cmando tr
tr -cd [:print:] < non_printable.txt

# Podría guardarse el fichero modificado en un nuevo fichero
tr -cd [:print:] < non_printable.txt > printable.txt


man tr
```

`cut` : Cortar textos

El comando `cut` permite seleccionar y cortar campos de cada línea de un fichero, o de la entrada estándar, para extraer las partes seleccionadas. Para diferenciar un campo de otro, usa delimitadores. El delimitador por defecto es el tabulador.

Sintaxs básica

cut OPCIONES... [ARCHIVO]...

- "-c": Esta opción selecciona los **caracteres** de cada línea en base al archivo de entrada
- "-d": Permite indicar el **delimitador** entre los campos
- "-f": Esta opción permite seleccionar las **columnas** separadas por un delimitador en particular. El delimitador por defecto es el tabulador.

```
cd
echo "Lee y aprende, aprende y haz, haz y evoluciona." > meme.txt
echo "Learn and code, code and share, share and evolve." >> meme.txt

# Selecciona unos caracteres a partir de su posición en la línea (-c)
cut -c 1-3 meme.txt
cut -c 7-14 meme.txt

# -f: Selecciona campos, -d: separados por un delimitador

# Seleccionar campos (separados por ',')  o selecciona un rango de campos
(separados por '-'),  separados por un delimitador (-d) (-f)
cut -d "," -f 1,2 meme.txt
cut -d "," -f 1,3 meme.txt
cut -d "," -f 1-3 meme.txt

cut -d " " -f 1,2 meme.txt
cut -d " " -f 1,3 meme.txt
cut -d " " -f 1-3 meme.txt
```

```
# ¡Cuidado con los límites de los campos! (nuestro fichero solo tiene 3 campos
separados por ',')
cut -d "," -f 1,4 meme.txt
cut -d "," -f 1,5 meme.txt

# Ejemplo: Como ver la memoria RAM (total), comando free.
free
free | grep Mem
free | grep Mem | tr -s ' ' ','
free | grep Mem | tr -s ' ' ',' | cut -d "," -f 2

# Otra forma
free | grep Mem | tr -s [:space:] | sed 's/ /,/g' | cut -d , -f 2


# Otra forma: usando sed con una Expresión Regular (entre dos slash)
# \s --> space
# \+ --> más de un espacio. Es necesario "escapar" (con backslash) el caracter "+"
free | grep Mem | sed 's/\s\+/,/g'
free | grep Mem | sed 's/\s\+/,/g' | cut -d , -f2

#Usando tr
free | grep Mem |  tr -s ' ' ',' | cut -d , -f2

# Para especificar bytes en lugar de caracteres (útil con ficheros binarios),
# se puede usar la opción para especificar el número de bytes
 (-b).


man cut
info cut
```

`wc` : Word Count - Calculadora en el terminal

`wc`es una utilidad que permite contar el número de líneas, el número de palabras y el número de bytes de un fichero, o de las líneas que reciba por su entrada.

```
# Número de líneas, número de palabras, y número de bytes
wc /etc/bash.bashrc

# Para contar el número de líneas
wc -l /etc/bash.bashrc

# Para contar el número de bytes
wc -c /etc/bash.bashrc

# Para contar el número de caracteres
wc -m /etc/bash.bashrc

# Para contar el número de palabras
wc -w /etc/bash.bashrc
```

```
# Tamaño de la línea más larga
wc -L /etc/bash.bashrc

# Se puede usar en combinación con otros comandos
ls -l /etc | wc -l


man wc
```

6 Comandos para Empaquetar y Comprimir ficheros y directorios

- **Empaquetar**: Es agrupar en un solo fichero varios ficheros y/o directorios.
- **Comprimir**: Es reducir el tamaño de un fichero a través del uso de un algoritmo de compresión.

Empaquetar (.tar)

Con el comando **tar**, (tape archive) se pueden empaquetar y desempaquetar archivos. Para que haya compresión es necesario utilizarlo en combinación con programas tales como gzip o bzip. Este comando se puede utilizar con múltiples opciones, aunque hay algunas que quizá son importantes recordar.

Opción	Significado
-c	Crear un nuevo archivo .tar
-v	(verbose) Muestra una descripción detallada del progreso de la compresión
-f	Nombre del archivo
-z	Compresión gzip
-j	Compresión bzip2
-C	Extrae archivos en un directorio diferente
-x	Extraer el archivo
-r	Actualizar o agregar un archivo o directorio en un archivo .tar existente

Se pueden utilizar el comando .tar tanto para un archivo como para directorios.

Al crear un archivo con este programa (tar), en primer lugar, hay que escribir las opciones, seguidas del nombre del archivo que se quiere crear (extension .tar)y finalmente los ficheros y/o carpetas que este fichero debe contener.

```
cd
echo "Mi primer fichero" > file_1.txt
echo "Mi segundo fichero" > file_2.txt
echo "Mi tercer fichero" > file_3.txt

# Archiva o empaqueta varios ficheros/directorios en un fichero .tar
tar cvf mis_ficheros.tar file*.txt

ls -l
```

Comprimir (.tgz, .tar.gz, .bz2)

Si lo que queremos además es comprimir archivos tenemos que utilizar **tar** con un método de compresión (gzip o bzip)

gzip: Comprime varios ficheros/directorios en un fichero con extensión .tgz o .tar.gz La compresión GZIP es la compresión de archivos estándar en sistemas Unix/Linux y es un proceso de dos pasos: primero se usa el

programa tar para unir todos los archivos a comprimir en uno solo, sobre el que luego se usa el compresor gzip. Esta secuencia es tan común que **el comando tar incluye una opción para comprimir directamente el archivo al finalizar. La opción es -z con el comando tar.**

```
tar cvzf mis_ficheros_comprimidos.tgz file*.txt

# Donde:
-z Comprime el archivo usando gzip
-c Crea un archivo
-v Verbose, escribe en pantalla información sobre el proceso de compresión
-f Nombre del archivo

ls -l

# También se puede comprimir un directorio en un fichero .tgz o .tar.gz
mkdir mi_directorio
cp file*.txt mi_directorio

tar cvzf mi_directorio_comprimido1.tgz mi_directorio

ls -l
```

bzip: Comprime varios ficheros en un fichero con extensión .bz2. El algoritmo .bz2 proporciona más compresión en comparación con gzip. Sin embargo, esta alternativa tomará mas tiempo para comprimir y descomprimir. **Para usar el comando tar con la compresión bzip, debemos usar la opción -j.**

```
tar cvjf mi_directorio_comprimido2.bz2 mi_directorio

# Donde:
-j Comprime el archivo usando bzip
-c Crea un archivo
-v Verbose, escribe en pantalla información sobre el proceso de compresión
-f Nombre del archivo

ls -l
```

Desempaquetar y descomprimir con el comando .tar

Para desempaquetar debemos usar la opción -x del comando tar. Si además queremos descomprimir debemos usar la opción -z si el fichero estaba comprimido con gzip o -j si estaba comprimido con bzip.

```
# Extraer el contenido de un fichero .tar
tar xvf mis_ficheros.tar

# Extraer el contenido de un fichero .tgz o .tar.gz (GZIP)
tar xvzf mis_ficheros_comprimidos.tgz
```

```
# Extraer el contenido de un fichero .bz2 (BZIP2)
tar xvjf mi_directorio_comprimido2.bz2
```

Mostrar el contenido de un ficero .tar

```
# Listar el contenido de un fichero .tar
tar tvf mis_ficheros.tar

man tar
info tar
```

Comados de compresión y descompresión de ficheros

Antes de empezar vamos a preparar algunos ficheros:

```
# Preparamos unos ficheros de prueba
cd
mkdir -p "$HOME/docus"
cd "$HOME/docus"
echo "Este es el documento 1" > doc1.txt
echo "Este es el documento 2" > doc2.txt
echo "Este es el documento 3" > doc3.txt
echo "Este es el documento 4" > doc4.txt
echo "Este es el documento 5" > doc5.txt

cat doc1.txt doc2.txt doc3.txt doc4.txt doc5.txt > big_doc.txt
cp big_doc.txt  big_doc_to_gzip.txt
cp big_doc.txt  big_doc_to_bzip2.txt

pwd
ls -l
```

Comprimir

`zip`

Algunas veces es necesario que el archivo sea descomprimido en otros sistemas operativos, en ese caso es útil usar el formato zip que tiene mayor compatibilidad (el más usado en Windows)

Sintaxis: zip <archivo_comprimdo> <archivo(s)_a_comprimir>

Par comprmir un directorio usar la opción -r.

Sintaxis: zip -r <dir_comprimido.zip> <directorio_a_comrpimir>

```
# Sintaxis: zip <archivo comprimdo> <archivo(s) a comprimir>

zip docus.zip doc[[:digit:]].txt
ls -l
```

gzip

Es uno de los métodos más utilizados para comprimir en Linux, pero puede utilizarse también en sistemas Windows y macOS. Solo comprime archivos y no directorios.

Sintaxis: gzip [opciones] <archivo(s)_a_comprimir>

gzip remplaza el archivo original por el comprimido añadiendo la extensión .gz, salvo que utilicemos la opción -k

Si se le pasan varios archivos comprime cada uno de ellos.

```
# Sintaxis: gzip [opciones] <archivo(s) a comprimir>
# La opción -k mantiene el fichero original
# gzip doc[[:digit:]].txt --> si se le pasan varios ficheros comprime cada uno de
los archivos

gzip big_doc_to_gzip.txt

# Vemos que se ha generado el archivo: big_doc_to_gzip.txt.gz
ls -l
```

bzip2

Permite comprimir archivos en Linux sin pérdidas con una gran calidad. Los archivos empaquetados con bzip2 obtienen la extensión .bz2. Solo comprime archivos y no directorios.

Sintaxis: bzip2 [opciones] <archivo(s)_a_comprimir> gzip remplaza el archivo original por el comprimido añadiendo la extensión .gz, salvo que utilicemos la opción -k

Si se le pasan varios archivos comprime cada uno de ellos.

```
# Sintaxis: bzip2 [opciones] <archivo(s) a comprimir>
# La opción -k mantiene los archivos despues del proceso de compresión
# bzip2 doc[[:digit:]].txt --> si se le pasan varios ficheros comprime cada uno de
los archivos

bzip2 -k doc[[:digit:]].txt
# Vemos que se ha generado un fichero .bz2 por cada archivo y que se han mantenido
los originales
ls -l
```

```
bzip2 big_doc_to_bzip2.txt
# Vemos que se ha generado el archivo big_doc_to_bzip2.txt.bz2
ls -l
```

Descomprimir

```
# Usaremos los ficheros de prueba empleados al comprimir.
cd "$HOME/docus"
```

`unzip`

```
unzip docus.zip
ls -l
```

`gunzip` **(equivalente a: gzip -d)**

```
# gunzip es equivalente a gzip -d
gzip -d big_doc_to_gzip.txt.gz
gunzip big_doc_to_gzip.txt.gz
ls -l
```

`bunzip2` **(equivalente a bzip2 -d)**

```
# bunzip2 es equivalente a bzip2 -d
bzip2 -d big_doc_to_bzip2.txt.bz2
bunzip2 big_doc_to_bzip2.txt.bz2
ls -l
```

`tar` **(para descomprimir)**

```
# El comando tar para descomprimir también se ha visto antes, pero se deja aquí
una referencia rápida.

# tar usando gunzip
tar zxvf docus.tgz
tar zxvf docus.tar.gz
```

```
ls -l

# tar usando bzip2
tar jxvf docus.bz2
ls -l

man unzip
man gunzip
man bunzip2
man tar
```

Listar el contenido de un fichero comprimido, se usan estos comandos.

```
unzip -l docus.zip
gzip -l doc1.gz
tar ztvf docus.tgz
tar ztvf docus.tar.gz
tar jtvf docus.tar.bz2
```

7 Redirecciones

En Linux existe la capacidad de redireccionar la visualización de la pantalla y por tanto, redireccionar la salida de un comando, de una aplicación…. hacia:

- Fichero.
- Impresora.
- Cualquier otro periférico.

Los comandos en Linux pueden generan dos tipos de mensajes:

- Los asociados a la salida o resultados del comando
- Los mesnsajes asociados a los errores que puede generar la ejecución de un comando.

Esto nos permite diferenciar esos tipos de mensajes y por ejemplo poder enviar los mensajes de error de un programa hacia un fichero y posteriormente tratarlos de manera automatizada.

También nos permite sustituir la introducción de datos vía teclado por los datos existentes en un fichero.

Las *shell* en Linux, manejan tres flujos de E/S (*I/O streams*):

- `0 - stdin` - **Entrada estándar**: Proporciona la entrada a los comandos. Tiene el descriptor de fichero **0**.
- `1 - stdout` - **Salida estándar**: Muestra la salida de los comandos. Tiene el descriptor de fichero **1**.
- `2 - stderr` - **Salida de error estándar**: Muestra la salida de los errores de los comandos. Tiene el descriptor de fichero **2**.

Redireccionamiento de salida

Se utiliza el carácter '>' para redireccionar la salida ($comando > fichero) Redirecciona la salida de un descriptor de fichero hacia otro fichero. Crea el fichero de salida si no existe. Si ya existe, los contenidos del fichero de salida son sobreescritos, generalmente sin avisar.

Ejemplo:

```
ls -l > listado.txt
```

a) Si no existe el fichero listado.txt, se creará.

b) Si existe listado.txt, se sobreescribirá, incluso si el comando introducido fuera incorrecto.

El Shell de Linux, en primer lugar crea el fichero y posteriormente introduce el resultado del comando en listado.txt, sea el que sea.

Si lo que se quiere es añadir información/datos a un fichero existente se utilizará el carácter '>>'. Crea el fichero de salida si no existe. Si ya existe, los contenidos se anexan al fichero de salida.

Ejemplo:

```
ls -l >> listado.txt
```

De esta forma se añadirán nuevos datos a listado.txt sin sobreescribirlo.

Redireccionamiento de entrada

Se utiliza el carácter '<'

Los comandos que esperan datos o parámetros pueden también recibirlos desde un fichero usando el redirector '<'.

```
$wc < fichero.txt
$tr ' ' '\t' <prueba.txt
```

Ejemplo:

```
 $wc < fichero.txt
```

Pasamos fichero.txt al comando wc que contará y mostrará las líneas del fichero

También podemos redireccionar la entrada y la salida encadenando las redirecciones. Ejemplo:

```
$rev < fichero_ejemplo.txt >> salida.dat
```

NOTA: El comando "rev" invierte la posición de cada uno de los caracteres del contenido del fichero.

En el ejemplo anterior el contenido de "fichero_ejemplo.txt" se va a invertir y se va a añadir al fichero salida.dat (no se sobrescribe, sino que se añade al final).

Anteriormente se han visto varios ejemplos, similares al siguiente:

```
echo "Esto es una prueba" > text1
tr ' ' '\t' <text1
cat text1
cat < text1
```

Algunas *shells*, como Bash, incluyen una forma de redirección de entrada llamada `here-document`, y que es habitualmente usada en scripts.

Los *here-documents* también se usan con el operador `<<`.

A continuación se muestran algunos ejemplos de *here-documents*:

```
# Usamos un here-document para simular un fichero como entrada para el comando
'sort'. El here-document está delimitado por el identificador END.

# 'sort' es una utilidad que toma el archivo(s) que figuran en su lista de
argumentos y ordena sus líneas según unos parametros.

sort -k2 <<END
1 física
2 programación
3 estadística
END
# el documento se acaba al introducir END. sort -k2 ordena alfabeticamente por la
segunda columna


# Es común crear el contenido de un fichero así:
cat <<TORT >tortilla.txt
patatas
huevo
CEBOLLA
sal
aceite

y amor :)
TORT

cat tortilla.txt
```

Uso de los streams stdin, stdout stderror

Ya hemos comentado que los canales estándar en Linux, por defecto, son:

- stdin ó 0. Está conectado al teclado, es realmente la entrada estándar de nuestro ordenador.
- stdout ó 1. Está conectado a la pantalla. Es la salida estándar.
- stderr o 2. Cuando se produce un error, este se envía a este archivo, que también está conectado con la pantalla. Es la salida estándar de errores. Si se desea, se puede redireccionar el canal de error estándar hacia otro fichero.

En el siguiente ejemplo:

```
$ls t* y*
```

Si en el directorio donde estamos no hay ningún fichero/directorio que comience por ‘t’ o por ‘y’ este comando dará error y dicho error se muesta por pantalla.

Sin embargo, en el siguiente ejemplo:

```
$ls t* y* 2> error.txt
```

Si en el directorio donde estamos no hay ningún fichero/directorio que comience por 't' o por 'y' este comando dará error y dicho error se redireccionará al fichero error.txt")

NOTA: observa que no puede haber espacio entre el 2 (id. de stderr y el símbolo de la redirección)

Es decir, no se muestran por pantalla los errores, sino que los reencaminamos al fichero error.txt

Veamos otro ejemplo:

```
 $ls t* n* > resultado.log
```

En este caso los errores se muestran por pantalla mientras que la salida correcta se va al fichero resultado.log. Aparecerá en la pantalla:

```
ls: no se puede acceder a 't*': No existe el archivo o el directorio
```

Si ahora mostramos el contenido del fichero resultado.log aparecerán los ficheros que tenemos en el directorio que empiezan por n.

```
$cat resultado.log
nombre_archivo1.txt
nombre_archivo2.txt
nombre_archivo3.txt
```

Otro ejemplo:

```
$ ls z* y* 2> errores.log
```

Este ejemplo mostraría la salida correcta (ficheros que cumplan patrón) por pantalla y los errores se enviarían errores.log.

Veamos algunos ejemplos más:

```
# Enviamos los datos de salida a un fichero y los errores a otro fichero:
ls -laR /  1>fichero_datos_salida 2>fichero_errores

# LOs errores los podemos ignorar: redirigiendo la salida de error (stderr) a
/dev/null
```

```
ls -laR /  2>/dev/null

# Si queremos ignorar la salida del comando: redirige la salida estándar (stdout)
a /dev/null
ls -laR /  1>/dev/null

# Si queremos ignorar la salida y los errores: primero redirigir la salida
estándar (stdout)
# a /dev/null, y después redirigir la salida de error (stderr) hacia donde
# apunte la salida estándar (que se había apuntado a /dev/null)
ls -laR / 1>/dev/null 2>&1

# Lo más común y úitl para guardar la salida de un comando (o programa) y los
errores es utilizar:
ls -laR / 1>salida_y_errores.txt 2>&1
```

¿Qué es **/dev/null?** Es un fichero especial en Linux. Es donde se envía la información para que sea descartada. Representa "la nada".

Sigamos con más ejemplos usando los ficheros que hemos preparado antes:

```
#mkdir redir
cd ~/redir
echo "Hola" > xprueba.txt

ls x* z*
ls x* z* 1>stdout.txt 2>stderr.txt
cat stdout.txt
cat stderr.txt
```

Si se omite el descriptor de fichero, se toma la salida estándar por defecto stdout. La salida estandar por defecto "stdout" tiene como descriptor de fichero: 1.

```
ls x* z* >stdout.txt 2>stderr.txt

ls w* y*
ls w* y* >>stdout.txt 2>>stderr.txt

cat stdout.txt
cat stderr.txt
```

Se puede redirigir la salida estándar y la salida de error estándar al mismo destino, y para ello se emplean los operadores `&>` y `&>>`.

```
ls x* z* &>stdout_and_stderr.txt
ls w* y* &>>stdout_and_stderr.txt
```

El orden en el que se redireccionan las salidas **es importante**. Por ejemplo:

```
[ls 2>&1 >salida.txt]
ls x* z* 2>&1 >salida.txt
cat salida.txt
#en el fichero no se incluye la salida de error
```

no es lo mismo que:

```
[ls >salida.txt 2>&1]
ls x* z* >salida.txt 2>&1
cat salida.txt
#en este caso salida.txt si está afectado por la redirección
```

- En el primer caso, *stderr* es redireccionada al sitio actual de *stdout* y luego *stdout* es redireccionada al fichero *salida.txt*. Pero esta segunda redirección afecta sólo a *stdout*, no a *stderr*.
- En el segundo caso, *stderr* es redireccionada al sitio actual de *stdout* (que es *salida.txt*), por ello el error aparece en el fichero

Observar que en el primer comando que la salida estándar fue redireccionada después de que el error estándar, por lo tanto la salida del error estándar todavía va a la ventana de la terminal.

```
# Por defecto, las salidas stdout y stderr de un comando, están apuntando por
defecto al terminal/pantalla
#
#                   +----------+   (stdout)
#                   |          |----> 1 (terminal, pantalla)
#            0 >----|    ls    |
#         (stdin)   |          |----> 2 (terminal, pantalla)
#                   +----------+   (stderr)
#
```

```
# Redirige stdout y stderr hacia el fichero salida.txt
ls x* z* &>salida.txt
cat salida.txt
#en la pantalla NO se presenta la salida de stderr
```

```
# Redirige stdout al fichero salida.txt, y después redirige stderr hacia donde
apunte stdout (que es salida.txt)
ls x* z* >salida.txt 2>&1
cat salida.txt
#en la pantalla NO se presenta la salida de stderr


# Redirige stderr a donde apunte stdout (terminal, pantalla). Después redirige
stdout hacia el fichero salida.txt. ¡Pero stderr se había quedado apuntando al
terminal (pantalla)!
ls x* z* 2>&1 >salida.txt    # stderr no va hacia salida.txt
cat salida.txt
#en la pantalla SÍ se presenta la salida de stderr


# Ahora podemos ejecutar los siguientes comandos para ver la diferencia
find / -user elenagg | grep -vi denegado
find / -user elenagg 2>&1 | grep -vi denegado
```

8 Tuberías (pipes)

Ya se han usado tuberías anteriormente. Básicamente sirven para redirigir la salida de un comando hacia la entrada de otro comando. Se pueden así encadenar comandos usando tuberías. El operador tubería es `|` (AltGr + 1).

```
ls y* x* z* u* q*

ls y* x* z* u* q*  2>&1 | sort -r
# comando sort: ordena. Con opción -r invierte la ordenacion
```

Para ampliar...

Operador "-"

Cada comando de la cadena puede tener sus opciones o argumentos. Algunos comandos usan el operador `-` (guión) en lugar de un nombre de fichero, cuando la entrada del comando debe provenir de un stdin (en lugar de hacerlo de un fichero).

```
# Preparamos el fichero para el ejemplo
echo  "Texto 1" > text1
echo  "Texto 2" > text2
echo  "Texto 3" > text3
cat text1 text2 text3  > tuberias.txt
cat tuberias.txt

tar cvf tuberias.tar tuberias.txt
ls -l
bzip2 tuberias.tar
ls -l

# Ahora podemos ver el operador - en acción:
 bunzip2 -c tuberias.tar.bz2 | tar -xvf -
```

En el caso de **usar tuberías para encadenar varios comandos**, y que algún comando necesite redireccionar su entrada (con el operador **<**), se pondrá esa redirección de la entrada en primer lugar dentro de la cadena de comandos.

Esto quiere decir, que será el primer comando el que tome como entrada un fichero, por ejemplo. Así, el resto de comandos que estén encadenados harán operaciones y tratamientos sobre el fichero leído por el primer comando.

9 Gestión de Paquetes

Gestor de Paquetes

Un Gestor de Paquetes administra “paquetes”, siendo estos un conjunto de archivos que permiten la instalación y ejecución de un programa. Dicho de otro modo, son los paquetes que contienen programas adicionales al core de Linux. Por ejemplo, en el paquete Chrome estarían todos los archivos necesarios para instalar y ejecutar la aplicación Google Chrome. Y el administrador se encargaría de buscar, instalar, desinstalar, actualizar un paquete. La distribución Ubuntu (Debian) tiene dos gestores principales:

- APT & APT-GET
- APTITUDE

En RedHat se usa YUM.

Repositorios y Paquetes

Un repositorio es un almacén de contenido en la nube que se encarga de almacenar paquetes y programas para que el usuario pueda descargar e instalar en su distribución GNU/Linux.

En el fichero /etc/apt/sources.list se encuentra la lista de repositorios y podríamos añadir nuevos repos no oficiales.

Nota: En RedHat las paquetes se guardan con extensión .rpm y en Debian/Ubuntu con extensión .deb

Haciendo:

```
$cat /etc/apt/sources.list
```

Podemos ver la lista de repositorios que usará tu Sistema Operativo.

http://es.archive.ubuntu.com/ubuntu/dists/ (aquí puedes ver un ejemplo de repo)

Tenemos dos gestores disponibles:

- APT-GET (pre-instalado en Ubuntu)
- APTITUDE

- Se tiene que instalar mediante:

```
$sudo apt-get install aptitude
```

- Versión mejorada de apt-get
- Al eliminar un paquete, elimina las dependencias (otros programas requeridos para ejecutar)
- Antes de hacer cualquier cosa, siempre pregunta

NOTA: mira el "usage" (cómo usarlo), ejecutando "sudo apt-get" en la consola.

```
    $sudo apt-get
```

Debemos utilizar "sudo" para tener privilegios de superusuario necesarios para acceder y/o modificar ficheros del sistema.

apt-get vs aptitude

APT	***APTITUDE***
SEARCH	SEARCH
LIST	UPDATE
	UPGRADE
	INSTALL
	REMOVE
	PURGE

APT-GET
UPDATE
UPGRADE
INSTALL
REMOVE
PURGE

- search: busca el paquete en los repositorios (apt search packageName)
- list: lista los paquetes instalados (apt list –installed)
- update: actualiza la lista de los repositorios (sudo apt-get update)
- upgrade: actualiza los paquetes que tengan actualizaciones (sudo apt-get upgrade)
- install: instala un paquete (sudo apt-get install packageName)
- remove: desinstala un paquete (sudo apt-get remove packageName)

- purge: desinstala un paquete y elimina sus ficheros de configuración (sudo apt-get purge packageName)

Como instalar aptitude

```
$sudo apt-get install aptitude
$sudo aptitude update
$sudo aptitude upgrade
```

Buscamos cualquier aplicación, por ejemplo “tilda” es una consola personalizada. Si solo vamos a leer no es necesario usar “sudo”:

```
$aptitude search tilda
```

La instalamos (aquí ya necesario sudo):

```
$sudo aptitude install tilda
```

con F1 se muestra/oculta la terminal.

Si ahora queremos eliminar sus ficheros de configuración:

```
$sudo aptitude remove tilda
```

O bien:

```
$sudo aptitude purge tilda
```

10 Otros comandos de utilidad: La fecha, la hora y el calendario

date

```
# Mostrar la fecha y hora


#Muestra la fecha y la hora en formato CET
date

# Nuestra la fecha y la hora en formato UTC
date -u

# Muestra el año
date +"%Y"

# Muestra el mes
date +"%m"

# Muestra el día, mes y año en el formato indicado
#El carácter %Y se reemplazará con el año, %m con el mes y %d con el día del mes.
date +"%d-%m-%Y"
date +"%d-%m-%Y %H:%M:%S"
date +"Año: %Y, Mes: %m, Día: %d"

# Establecer la fecha y la hora (necesita permisos de root y no tener la
sincronización NTP activada)
sudo date --set 2020-10-02
sudo date --set 19:00:07
sudo date --set="20190701 14:30"
```

NTP (Network Time Protocol) es un protocolo que se basa en el envio de señales de sincronizacion horaria através de la red IP. Es el método más común para sincronizar el reloj del software de un sistema GNU/Linux con los servidores horarios de Internet.

cal

```
# Mostrar el calendario
cal    (mes actual)
cal -3 (muestra el mes anterior, mes actual, mes posterior)
cal -m 12 (calendario de diciembre o mes 12 del año actual)
cal -m 3 2019 (calendario del mes 3 del año 2019)
cal -y (calendario anual)
cal -y 2021  (calendario del 2021)
ncal   (formato vertical)
ncal -e (fecha de semana santa, -o semana santa ortodoxa)
ncal -w -3 (muestra el numero de la semana del mes anterior, el actual y el
siguiente)
```

timedatectl

```
# Gestión de la fecha y hora

#El comando timedatectl muestra la hora actual del hardware (RTC) y la hora del reloj del sistema
timedatectl

# timedatectl: establecer la fecha y la hora (sólo se puede hacer si no está establecida sincronización automática)
timedatectl set-time 22:53:48
timedatectl set-time "2019-10-02 19:00:07"
timedatectl set-time "2019-10-02"

# timedatectl: listar las zonas horarias
timedatectl list-timezones
timedatectl list-timezones | grep -i madrid
timedatectl list-timezones | grep -i europe

# timedatectl: establecer la zona horaria
timedatectl set-timezone Europe/Madrid

# Des/Activar la sincronización de la fecha y hora por NTP
timedatectl set-ntp no
timedatectl set-ntp yes
```

Otros comandos relacionados con el tiempo

```
#Comienzo, Estado, ficheros, PID, del servidor NTP
systemctl status systemd-timesyncd.service

#Informacion sobre el servidor NTP
timedatectl timesync-status

man date
man cal
man timedatectl
```