Sistemas Informáticos

1º de DAM/DAW

Tema 7

Comandos de Linux

IES San Vicente

Curso 2020/2021

Índice de contenido

1. Comandos básicos de Linux

1.1 Sintáxis de un comando

Hay que recordar primero que Linux diferencia entre mayúsculas y minúsculas y hay que tener especial cuidado con ello, ya que *cd Home* no es lo mismo que *cd home.*

La sintaxis general de las órdenes del shell es:

$ **orden [-opciones] argumento1 argumento2 ….Estilo predeterminado**

Las opciones pueden ser cortas, con un guión, "*ls -a*", o largas con 2 guiones "*ls --all*". Muchas veces las opciones largas tienen su equivalente corta. Las opciones cortas a veces se pueden agrupar, ya que sólo se constituyen por una letra, por ejemplo: "*ls -l -a*" = "*ls -la*".

1.2 Ayuda sobre comandos

man *comando*

Ofrece una página de ayuda sobre el comando, si este tiene el manual instalado en el sistema. Para navegar por la ayuda se utilizan las flechas y av-pag, re-pag, y para salir se utiliza la tecla q.

info *comando*

Algunos comandos tienen un manual info que ofrece una descripción mucho más detallada que *man.*

comando --help

En muchos comandos, la opción --help ofrece una ayuda rápida sobre el uso del comando, bastante más breve que la página del manual. A veces (pero no siempre), la opción -h es una abreviatura de lo mismo.

help comando

El comando help ofrece una ayuda similar a la opción --help (más breve que man) en los comandos que lo soportan.

1.3 Algunos comandos básicos

cal

Muestra un calendario del mes actual, con la opción "**-y**", muestra el año completo. Como argumentos se le pueden pasar el año que queremos que nos muestre, o un numero de mes y el año, para que nos muestre sólo un mes de ese año (**cal 3 2012**).

cat *fich1 fich2 …*

Visualiza el contenido de los ficheros. Si le pasamos más de un fichero, visualiza el

contenido de todos ellos concatenándolos en el orden dispuesto. Algunas opciones:

- **-n** = numera todas las lineas de salida.
- **-b** = numera todas las lineas de salida menos las que están en blanco.
- **-s** = si encuentra varias lineas en blanco, las agrupa en una sola.

cd *directorio*

Entra en el directorio que especifiquemos. También se puede utilizar el tabulador para autocompletar el nombre del directorio en este caso. Podemos especificar una ruta relativa (desde el directorio donde nos encontramos → cd musica/mp3), o una ruta absoluta (desde el directorio raiz "/" → cd /usr/bin), "../" representa el directorio anterior o padre, y "./" representa el directorio actual.

clear

Limpia la pantalla y sólo se queda la primera linea para introducir comandos. La combinación de teclas **Ctrl+l** hace la misma función.

cp

Copia uno, o varios archivos a un directorio de destino, dependiendo como se utilice:

- **cp *archivo_origen archivo_destino*** -> Crea una copia idéntica a archivo_origen con el nombre de archivo_destino. Si se especifica un directorio en el destino, entonces crea una copia con el mismo nombre del archivo original dentro de ese directorio.
- **cp *arch1 arch2 arch3 directorio_destino*** -> Copia varios archivos dentro de un directorio destino.

Algunas opciones generales:

- **-r** puede copiar directorios con todo lo que contienen
- **-i** : pide confirmación antes de sobreescribir un archivo que ya existe.
- **-b** : crea copias de seguridad de los archivos que se sobreescriban.
- **--help** : ayuda sobre todas las opciones.

date

muestra la fecha del sistema y nos permite cambiarla si le pasamos el siguiente argumento: **date *MMDDhhmm[AAAA][.ss]***

- **MM**: numero del mes. 2 dígitos.
- **DD**: numero del día. 2 dígitos.
- **hh**: hora (00-23). 2 dígitos.
- **Mm**: minutos. 2 dígitos
- **AAAA**: año. 4 dígitos o 2 últimos dígitos (opcional)

- ss: segundos. 2 dígitos con un punto delante (opcional)

Ejemplo: 12 de marzo de 2012 a las 15:16 → **date 031215162012**

Mostrar fecha con un formato concreto

- date +"formato"
- **formato** sería la cadena de texto que formará la fecha. Hay una serie de comodines que se pueden consultar en el man, por ejemplo %d se sustituye por el día del mes, %m por el número de mes del año y %Y por el año completo.
- Ejemplo **date +"Hoy es: %d del %m de %Y"** → **Hoy es 13 del 03 de 2012**

Mostrar otra fecha que no sea la actual:

- Con la opción -d o –date seguida de una cadena en un formato muy libre que represente una fecha o un periodo de tiempo anterior o posterior al actual. Para más información consultar **info date**.
- Podemos usar entre otros muchos el formato típico de bases de datos **AAAA-MM-DD hh:mm**, o cadenas como **'1 day ago'**, **'1 month ago'**, **'5 months ago 5 days ago'**, **'this saturday'**, **'tomorrow'**, **'yesterday'**, **'next week'**, **'1 week'**, etc.
  - **date -d '2013-04-06 18:19' +'Dia: %d del %m de %Y a las %H:%M'**

Comando diff

Compara ficheros o el contenido de directorios.

Formato: diff [OPCIONES] FICHERO1 [FICHERO2]

Opciones básicas:

-b ---------- Ignora los espacios en blanco.
-text ---------- Compara el texto línea por línea.
-w ---------- Descarta espacio en blanco cuando compara líneas.
-r ---------- Compara directorios de forma recursiva.
-q ---------- Informa sólo de si los ficheros difieren.
-y ---------- Muestra la salida a dos columnas.

du *archivo*

Este comando muestra la información sobre el tamaño de un archivo. Por defecto el tamaño lo muestra en KB, aunque con la opción **-h** por ejemplo, cambia el formato por uno más amigable (MB, GB, … dependiendo del tamaño). La opción **--time**, por ejemplo, muestra la fecha de la última modificación sobre el archivo.

Si el parámetro es un directorio, recorrerá recursivamente su contenido mostrando la información referente a cada archivo que contiene. Para limitar cuantos subdirectorios o niveles desciende cuando analiza un directorio está la opción **-d N** (siendo N los niveles que descenderá. Por ejemplo, el valor 1 evitará que se interne en ningún subdirectorio).

echo *expresion*

Muestra la expresión por pantalla, es decir, repite lo mismo que le pasamos como argumento. Si la expresión es una variable (las veremos en el futuro), como $HOME,

escribiría su valor (en el caso del usuario guest, escribiría /home/guest). La expresión puede ir entre comillas.

exit

Cierra la sesión actual en la consola. Si es una sesión remota o una sesión local a la que hemos accedido desde otro usuario, vuelve a la sesión anterior. Si no, cierra la consola.

find *directorio*

Permite buscar archivos en el árbol de directorios, desde el directorio especificado con diferentes criterios o filtros:

- **-name "nombre"** → archivos/directorios que tengan el nombre especificado. Se pueden usar caracteres comodín como *, ? y [] para darle flexibilidad.
- **-type tipo**→ Tipo de archivo. 'f' (fichero normal), 'd' (directorio), 's' (enlace simbólico), etc.
- **-size tamaño** → Buscar archivos que tengan menos o más que el tamaño especificado, además es recomendable poner la unidad al final (**c → bytes, k → kilobytes, M → megabytes, G → gigabytes**), ya que por defecto el tamaño se mide en bloques (unidades de 512 bytes, o medio kilobyte).
  - **-size +200M** → Archivos de más de 200 megas.
- **-mtime días** → Buscar archivos que fueron modificados hace más de x días o menos, dependiendo el signo utilizado delante del número.
  - **-mtime -2** → Archivos modificados en los últimos 2 días (2 días o menos)
- **-exec comando {}** → Ejecuta un comando en cada uno de los archivos encontrados con find, '{}' se sustituye por el archivo automáticamente. Por ejemplo, si queremos borrar todos los archivos con extensión tgz de más de 20 megas creados hace más de 7 días, buscando desde el directorio actual:
  - **find . -name '*.tgz' -size +20M -mtime +7 -exec rm -f {}**

history

Muestra el historial de comandos que se han ejecutado en la sesión actual (y pasadas). Los comandos ejecutados en sesiones anteriores se guardan en el archivo $HOME/.bash_history. Los comandos de la sesión actual se añaden al archivo al cerrar la sesión.

La opción **-c** borra el historial de comandos, mientras que la opción **-w** actualiza el archivo del historial sin necesidad de cerrar la sesión.

hostname

Indica el nombre del equipo donde ejecutamos el comando.

hwclock

Igual que **date** muestra y cambia la fecha del sistema operativo (pero si reiniciamos ese cambio desaparece). Si queremos ver o establecer la fecha y hora del reloj interno

hardware (cambio permanente), usamos este comando. Ejecutándolo sin más nos muestra la fecha hardware.

- **hwclock --systohc** → Establece la misma fecha de nuestro SO (establecida con date por ejemplo) en el reloj hardware.
- **hwclock --hctosys** → Al revés. Establece la misma fecha del reloj hardware en nuestro SO.

ln

Crea un enlace entre 2 archivos. En Linux hay 2 tipos de enlaces, duro y blando:

***Enlace duro**:* Es el que se crea por defecto si no se utiliza ninguna opción. Los enlaces duros representan un nombre alternativo al fichero, es decir, se puede acceder al fichero desde cualquiera de sus nombres, y el fichero no será borrado del sistema hasta que no se borren todos sus nombres.

Enlace blando (o simbólico): Enlace simbólico a un archivo. Es equivalente a un acceso directo, y si se borra el archivo original desaparece del sistema y el enlace se queda "huérfano". Este tipo de enlaces se pueden identificar porque al ejecutar "ls -l", aparece una "l" como primera letra de la línea y porque aparece la flecha y la ruta donde apunta después del nombre del archivo. Si el archivo enlazado desaparece, el enlace aparecerá normalmente con fondo rojo y parpadeando.

ln ***archivo enlace*** -> crea un enlace al archivo o directorio.

ln ***arch1 arch2 arch3 ... directorio_dest***-> Crea enlaces a los archivos dentro del directorio destino y con los mismos nombres que los archivos (sólo para enlaces duros).

- "**-s**": crea un enlace blando o simbólico.
- "**-i**": pide confirmación para sobrescribir si ya existe un fichero o enlace con el mismo nombre que el enlace que se va a crear.

Sólo el usuario **root** puede hacer enlaces duros a directorios, y además debe utilizar la opción "**-d**" para ello.

ls

Lista los archivos y directorios de la carpeta en la que nos encontramos. Las opciones "**-a**" y "**-l**", que muestran los archivos ocultos e información detallada sobre los archivos respectivamente, son las más utilizadas, pero hay otras tantas que se pueden consultar con man o con la opción "--help". En UNIX/Linux los archivos ocultos son los que llevan un punto delante del nombre.

mkdir *directorio*

Crea un directorio nuevo con el nombre que le pasemos. No se permiten espacios en blanco a no ser que utilicemos la barra "\" delante de cada espacio.

- **-p** : si creamos un directorio dentro de otro/s que no existe/n, con esta opción los crea también. Sin ella no podríamos crearlo. Ejemplo: **mkdir -p dir1/dir2** (si dir1 no existía, lo crea también).

mv

Parecido a **cp**, pero en lugar de funcionar como copiar-pegar, funciona como cortar-pegar, es decir, borra los archivos de origen una vez realizada la copia. No se requiere en este caso la opción **-r** para mover un directorio.

mv *archivo_orig archivo_dest*

mv *arch1 arch2 arch3 directorio_destino*

Funciona también como comando para renombrar un archivo o directorio.

passwd

Nos permite cambiar la contraseña del usuario actual. Se puede cambiar la contraseña de cualquier usuario del sistema pasándole como argumento el nombre del usuario al que se le quiera cambiar la contraseña (**passwd usuario**), eso sí, si no lo ejecutamos como root nos pedirá primero la contraseña actual de dicho usuario. Siempre tendremos que escribir la contraseña nueva 2 veces para confirmar el cambio, y si nos equivocamos, tendremos que repetir el proceso.

pwd

Nos indica la ruta completa al directorio donde nos encontramos actualmente.

reboot

Reinicia el sistema (solo root). Con **shutdown** apagaríamos el sistema.

rm *archivo*

Borra un archivo existente. Puede borrar más de un archivo o directorio a la vez si se le pasan como argumentos (o se utilizan caracteres comodín como *). Algunas opciones: (--help muestra todas)

- **-r** : Borra directorios con todo lo que contengan dentro
- **-f** : Borra sin pedir confirmación e ignora mensajes de error si hay archivos que no existen.
- **-i** = Pide confirmación cada vez que va a borrar un archivo.

stat *archivo*

Muestra información relativa al archivo.

```
stat /etc/passwd
Fichero: «/etc/passwd»
Tamaño: 2194          Bloques: 8          Bloque E/S: 4096   fichero regular
Dispositivo: 804h/2052d Nodo-i: 133639      Enlaces: 1
Acceso: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Acceso: 2016-02-13 17:12:00.400792899 +0100
Modificación: 2015-12-28 22:24:03.747699873 +0100
Cambio: 2015-12-28 22:24:03.751699962 +0100
Creación: -
```

La opción **-c** permite establecer, al estilo del comando date, qué campos se van a mostrar y el formato de la salida. Se puede consultar como mostrar cada campo en la página

del manual del comando. Por ejemplo, la orden '**stat -c "%x" archivo'** nos mostraría la fecha de última modificación.

su

Nos permite hacernos pasar por otro usuario durante una sesión pero no actualiza la sesión como si hiciéramos login desde 0. Si queremos que el login sea completo, hay que utilizar la opción "**-**", "**-l**" o "**--login**" (son equivalentes). Después de las opciones, como primer argumento utilizaremos el nombre del usuario al que queremos cambiar (por defecto si no se escribe nada el usuario es *root*).

time *comando*

muestra el tiempo que ha tardado un comando o programa en ejecutarse.

touch *archivo*

Crea un archivo vacío con el nombre que le demos. Puede crear más de un archivo a la vez. Si lo aplicamos a un archivo que ya existe le actualiza la fecha de última modificación.

tty

Nos informa del terminal en el que estamos trabajando en este momento.

uname

Muestra el nombre del sistema, por defecto sólo muestra el nombre del sistema operativo. Con la opción "-a" se muestra toda la información, y con "--help" o usando el *man* se puede consultar lo que muestran el resto de opciones.

who

Muestra las sesiones de usuario que hay iniciadas actualmente en la máquina (puede haber más de una por usuario, ya sea sesión en modo texto o gráfica "**tty**", o una (pseudo)terminal abierta desde sesión gráfica "**pts**"). Con la opción "**-q**" te dice al final el número total de sesiones.

- **who am i** → Nos muestra la sesión desde la que ejecutamos el comando

El comando **finger** se comporta de forma similar.

whoami

Nos muestra quienes somos en el sistema (nuestro nombre de usuario simplemente).

2. Rutas absolutas y relativas

El sistema de archivos en Linux está organizado en un directorio raíz "/", que contiene a su vez otros directorios, que a su vez contienen otros directorios y archivos, y así sucesivamente. De esta manera se construye un árbol de directorios.

Siempre que queramos acceder/crear/modificar/eliminar un archivo o directorio que no se encuentre en el directorio actual, tendremos que indicar la ruta que nos lleva a él.

Si queremos crear un archivo llamado "**arch2**" en "**dir2**/", tendremos que indicar al comando touch lo siguiente: ***touch dir2/arch2***.

Rutas absolutas: son las que se indican desde el directorio raíz siempre (comienzan por "/"). Un ejemplo sería: "/**home**/**usuario**/**dir1**/**dir2**".

Rutas relativas: son las que se indican desde el directorio que nos encontramos en este momento. Un ejemplo sería "**dir1**/**dir2**", no empiezan por "/".

- ../ : Representa al directorio anterior. Si en el ejemplo del comando tree de antes estamos dentro de dir1 y queremos crear un directorio "dir2-1" en dir2, podemos hacerlo con "***mkdir ../dir2/dir2-1***", sin tener porqué volver al directorio anterior para ello (precisamente para ir al directorio anterior se utiliza "***cd ..***")

- ./ : representa al directorio actual. Es igual hacer ***"touch fichero"*** que ***"touch ./fichero"***, y el comando ***"*****cd ./**" nos mantendrá en el directorio actual.

tree

Con este comando se muestra el arbol de directorios y ficheros que hay desde el directorio actual (esto es, una rama del árbol principal que parte desde el directorio raíz). Hay que instalarlo previamente, ya que en la mayoría de distribuciones no viene instalado por defecto.

```
~/prueba $ tree

.
|-- dir1
|    |-- dir1-1
|    |-- dir1-2
|
|-- dir2
|
|-- dir3
     |-- arch1
```

3. Permisos de archivos

En Linux, todo archivo pertenece a un usuario y a un grupo que se pueden cambiar mediante unos sencillos comandos. Esto hace que por cada archivo se puedan otorgar permisos a 3 entidades diferentes: **usuario, grupo y resto de usuarios**. Obviamente los permisos que se le otorguen al grupo afectarán a los usuarios que pertenezcan a dicho grupo (menos al propietario).

- **Permisos de usuario**: Afectan al propietario del archivo
- **Permisos de grupo**: Afectan a los usuarios que estén dentro del grupo al que pertenece el archivo.
- **Permisos otros usuarios**: Resto de usuarios.

El usuario **root** tendrá prioridad sobre cualquier propietario de archivo, y no le afectarán los permisos establecidos, excepto en los archivos de su propiedad.

Hay 3 tipos de permisos en un archivo que se pueden tener o no habilitados dependiendo del tipo de usuario al que pertenezcas en los 3 grupos anteriores:

- **r** (Permiso de lectura): Permite al usuario visualizar el contenido de un fichero. En un directorio se podrá visualizar la lista de ficheros que contiene.
- **w** (Permiso de escritura): Permite al usuario modificar/eliminar un fichero. En un directorio permite crear y borrar ficheros (esto último sólo si el fichero a borrar también tiene permisos de escritura).
- **x** (Permiso de ejecución): Permite ejecutar un fichero ejecutable (binario o un script). En un directorio este permiso significa que se puede entrar en él y ejecutar comandos dentro del mismo.

Ejecutando el comando “ls -l” podremos ver los permisos de cada fichero:

```
$ ls -l
total 4
drwxr-xr-x   2 arturo users 48 ene 26 12:16 dir1
lrwxrwxrwx   1 arturo users  5 ene 26 12:16 enlace1 -> fich1
-rw-r--r--   1 arturo users 28 ene 26 12:16 fich1
```

- 1ª columna. Muestra los atributos de cada fichero (tipo y permisos)
- 2ª columna. Muestra el número de enlaces que tiene un fichero o directorio.
- 3ª columna. Muestra el propietario del fichero.
- 4ª columna. Indica a qué grupo pertenece el fichero.
- 5ª columna. Es el tamaño en bytes del fichero o directorio.
- 6ª columna. Es la fecha, y en algunos caso la hora, de la última modificación.

- 7ª columna. Es el nombre del fichero o directorio. En el caso de un enlace blando, también indica el fichero que enlaza.

Los atributos de cada fichero están representados mediante una cadena de 10 caracteres, y tiene la siguiente estructura:

Tipo	Usuario/Propietario			Grupo propietario			Otros		
-	r	w	x	r	w	x	r	w	x

El tipo de fichero puede ser:

- **-** : archivo normal y corriente.
- **d**: directorio
- **l**: enlace simbólico (o blando) a otro archivo
- **b**: archivo de bloques (discos duros, disqueteras, etc...)
- **c**: archivo de caracteres (terminales, puertos usb, serie, teclado, ...)

En los permisos, cuando aparece un guión "-", significa que no tiene habilitado ese permiso.

3.1 Comandos para cambiar permisos y propietarios

chown

Cambia el propietario de uno o varios archivo (sólo lo puede hacer el usuario root).

chown [opciones] propietario archivo1 archivo2 ...

Algunas opciones:

- **-R**: si alguno de los ficheros afectados es un directorio, cambia también los propietarios de los archivos que contenga dentro.
- **-v**: muestra los cambios que realiza con detalle.
- **-L**: Si algún fichero afectado es un enlace simbólico, lo atraviesa y cambia el propietario del fichero que se enlaza.

chgrp

Parecido al anterior pero sólo cambia el grupo al que pertenece el archivo.

chgrp [opciones] grupo archivo1 archivo2 ...

Con el comando chown podemos cambiar también el grupo de los archivos simplemente añadiendo ":" y el nombre del nuevo grupo al usuario. ***chown usuario:grupo archivo***

- -R: si alguno de los ficheros afectados es un directorio, cambia también los propietarios de los archivos que contenga dentro.

chmod

Cambia los permisos asociados a un archivo. Una manera de hacerlo es asociando una máscara de 3 bits (del 0 al 7) a cada entidad (propietario, grupo, otros), de la siguiente manera:

Propietario	Grupo	Otros
111	111	111
7	7	7

Los 3 bits se corresponden con lectura, escritura y ejecución. Al comando **chmod** hay que indicarle del 0 al 7 los permisos que se les concederán a cada entidad. Cuando un bit es 0, no se tiene el permiso, y cuando es 1 si se tiene.

Ejemplo: ***chmod 754 archivo***.

Propietario: 7 (111 -> rwx, lectura, escritura y ejecución), **Grupo**: 5 (101 -> r-x, lectura y ejecución), **Otros**: 4 (100 -> r--, sólo lectura).

Otra forma de hacerlo es de la siguiente manera:

chmod (ugo)(+-=)(rwx) archivo

- **u** (modifica los permisos del propietario), **g** (grupo), **o** (otros).
- **+** (añade los permisos que se especifiquen), **-** (los elimina), **=** (añade los permisos, pero eliminando antes los que ya tuviera asignados).
- **r** (lectura), **w** (escritura), **x** (ejecución).

Ejemplo: ***chmod go+wx***. Añade los permisos de escritura y ejecución al grupo y a otros usuarios. Se pueden separar por comas las asignación de permisos: ***chmod g+wx,o+x***

- **-R**: si alguno de los ficheros afectados es un directorio, cambia también los propietarios de los archivos que contenga dentro.

umask

Especifica que permisos son los que se darán o no, cuando se crea un nuevo archivo. Con el comando ***umask*** (sin opciones) podemos ver los permisos que se dan actualmente.

La máscara de bits de umask funciona de manera contraria a chmod, un 1 indica que no se dará permiso, y un 0 indica que sí. **umask 0111 → chmod 666**

4. Redireccionamiento: > y >>

Estas expresiones, al colocarlas detrás de un comando, dirigen el texto que saldría por pantalla al ejecutar dicho comando a un fichero, en lugar de mostrarlo por pantalla (salida estandar).La diferencia entre ambas es que "**>**" borra el contenido que tuviera el fichero previamente, y lo escribe desde 0, y "**>>**" añade los nuevos contenidos al final del fichero sin borrar lo que ya había (concatena). Si el fichero no existía **se crea uno nuev**o con ese nombre.

echo "Hola que tal" > fich1 → Almacena la expresión en el archivo "fich1" en lugar de mostrarla por pantalla.

cat > fich2 → Almacena lo que vayamos escribiendo a continuación en el archivo "fich2". Lo que había previamente en ese fichero se borra. Con **Ctrl+d** terminamos de escribir.

cat fich1 >> fich2 → En lugar de mostrar el contenido de fich1 por pantalla, lo almacena al final de "fich2", sin borrar lo que ya contenía este último.

cat fich1 fich2 > fich3 → Concatena los archivos "fich1" y "fich2" en "fich3", es decir, copia los contenidos de fich1 y seguidamente los de fich2 dentro de fich3.

ls -l > fich1 → Almacena el resultado de ejecutar "**ls -l**" en "fich1", en lugar de mostrarlo por pantalla.

5. Alias

El comando **alias** nos permite definir atajos a comandos largos, es decir, podemos resumir un largo comando con varias opciones y argumentos que usemos con frecuencia en una sola palabra. Si tecleamos **alias** sin opciones, nos mostrará los que hay activos en el sistema. Un ejemplo típico de alias es el siguiente (suele estar definido en el sistema):

alias ll='ls -alF' → Cuando ejecutamos **ll**, en realidad estamos ejecutando **ls -alF**

Podríamos definirnos nosotros alias para cuando queramos subir más de un directorio hacia arriba por ejemplo:

alias cd..='cd ..'

alias cd..2='cd ../..'

alias cd..3='cd ../../..'

Así cuando escribamos **cd..3** en el terminal, subiremos 3 directorios en el árbol.

A los comandos definidos con un alias le podemos seguir concatenando opciones y argumentos igualmente, que se sumarán a los que hayamos definido en el alias. En resumen un alias es como una variable que al ejecutarlo se sustituye tal cual por lo que hayamos definido.

alias tgzc='tar cvzf'

tgzc archivo.tgz dir1 dir2 dir3

Si queremos que un alias sea permanente debemos añadirlo al final del archivo **.profile** en el home del usuario, de esta forma se ejecutará cada vez que iniciemos una sesión en la consola. Si queremos que sea válido para todos los usuarios del sistema en lugar de sólo uno, debemos añadirlo a /**etc**/**profile** ,o mejor, crear un archivo dentro de /**etc**/**profile.d**/ (con extensión .sh, aunque es opcional), y añadirlo a ese archivo.

El comando **unalias** seguido de un alias establecido sirve para borrarlo. Mientras que el comando **type** seguido de un nombre de comando, alias, etc. Informará al usuario de si la palabra que ha escrito a continuación es un comando, un alias, o no existe.

6. Pipes (tuberías)

A veces nos puede interesar que la información generada por un proceso pueda servir para que otro proceso trabaje con ella. Hasta ahora hemos visto que podemos redireccionar la salida de un proceso a un archivo (con > y >>). Bien, pues hay una forma de hacer que la salida de un proceso se convierta a su vez en la entrada de otro proceso, es decir, en lugar de redireccionar a un archivo, redireccionamos hacia otro proceso que recibirá la información como si fueran parámetros.

comando1 | comando2 | comando3 | .....

Con el símbolo "|" podemos enlazar todos los procesos que queramos. Se ejecutarán uno detrás de otro (no a la vez), y la salida que produzca uno, se utilizará como entrada del siguiente.

echo "dir1 dir2 dir3 dir4" | ls -l → El mensaje que envía el comando echo lo recoge "ls -l" como si fueran parámetros añadidos, hubiera sido equivalente a hacer "ls -l dir1 dir2 dir3 dir4".

Se pueden concatenar varios comandos a través de pipes:

cat usuarios.txt | grep 101 | sort

6.1 Comandos para procesar la salida

more

Cuando el texto no cabe por pantalla, nos muestra la parte que si cabe y podemos ir presionando 'enter' para avanzar y leer el texto.

less

Similar a 'more', pero más potente, permite movernos por el texto con las flechas, av/re-pag, etc... Como si de una página man se tratara (salimos con 'q').

tail -n

Muestra las últimas líneas escribiendo un número entero a continuación. También se puede aplicar directamente a un archivo → **tail -n 5 archivo**. Con el formato **tail -n +5** mostraría a partir de la quinta línea hasta el final.

sort

Ordena alfabéticamente las líneas del texto que se le pasa. Si las líneas están separadas por campos, se puede ordenar por otro campo que no sea el primero:

- **-n** → Ordena numéricamente en lugar de alfabéticamente.
- **-t ':' -u -k 3 -n /etc/passwd** → Usando como separador (**-t**) el carácter **':'**, sin mostrar líneas con valores repetidos (**-u**), ordena las líneas por el tercer campo (**-k 3**) y numéricamente (**-n**).

wc

Muestra la información sobre el número de líneas, palabras y bytes (en este orden) que contiene un archivo o un texto pasado a través de una pipe.

-l → Sólo líneas. **-c** → Sólo caracteres. **-w** → Sólo palabras

xargs

Algunos comandos no son capaces de procesar la entrada que le llega a través de una tubería, como por ejemplo el comando **rm**. Para solucionar esto, podemos apoyarnos en el comando xargs que se pone delante de lo que queremos ejecutar y concatena la entrada de la tubería al final para que la procese.

find . -name 'ar*' | xargs rm -f

tr

Sustituye unos caracteres por otros. Por ejemplo, si usamos **ls -l | tr " " "-"** sustituirá los espacios por guiones "-". Si queremos que sustituya varios caracteres seguidos por uno solo, sería con la opción **-s** al principio (**tr -s " " "-"**)

Se pueden sustituir más de 2 caracteres a la vez:

- **tr 'abc' '123'** → sustituciones (a → 1, b → 2, c → 3)
- **tr 'abcd' '12'** → Cuando hay menos caracteres en la sustitución, se toma el último como referencia para los que no tienen equivalencia (a → 1, b → 2, c → 2, d → 2)

Se pueden usar rangos al igual que en las expresiones regulares que veremos más adelante:

- **tr 'a-z' 'A-Z'** → Transforma las minúsculas a mayúculas
- **tr [:lower:] [:upper:]** → Equivale a lo anterior

cut

Su uso habitual es el de buscar y seleccionar columnas en un archivo de texto estructurado. En lugar de un archivo podemos usar la salida de un comando pasada por una tubería. Lo primero es saber cual es el separador que delimita las columnas. Si es el tabulador, el comando cut no necesita ningún parámetro que se lo indique (separador por defecto). En otro caso, el delimitador se le indica con la opción **-d** "**delimitador**".

Después debemos indicar las columnas que queremos que muestre. Esto se hace con la opción -f seguida del número de columna que queremos mostrar. Si queremos mostrar más columnas se deben separar por comas, o indicar un rango, ejemplo 3-6.

- **echo "12,32,45,65,78" | cut -d "," -f 2,4**
- **cut -d "-" -f 1-3,5 archivo.txt**

sed

Sed tiene más de una utilidad. Una de ellas es borrar líneas de un fichero (o de la salida de un comando pasada por una tubería).

- Indicando el número de línea seguida de la letra 'd', la borra → **sed '3d' fichero**
- **-n** y el número de línea seguido de **'p'**, imprime sólo esa línea → **sed -n '3p' fichero**.
- Podemos indicar un rango de líneas separadas por coma. El segundo carácter puede ser '$' que indica hasta el final → **sed '4,$p' fichero**
- Concatenar instrucciones con la opción **-e** delante de cada → **sed -n -e '3p' -e '9p'** (Imprime las líneas 3 y 9)
- Indicar que nos imprima sólo las líneas pares, impares, cada 3, etc. Simplemente indicando **INICIO~SALTO**. Por ejemplo, mostrar las líneas impares es empezar por la primera mostrando cada 2 líneas → **sed -n '1~2p'**
- Sustituir una cadena por otra con el siguiente formato (al final van las opciones):
  - **sed 's/cadena/sustitucion/g' fichero**
  - **Opciones**:
    - **g:** Si queremos que sustituya en cada línea todas las veces que aparezca la expresión . Por defecto, sólo sustituye la primera ocurrencia.
    - **I (i mayúscula)**: No diferencia entre mayúsculas y minúsculas.
    - **Número**: Reemplaza la coincidencia numero N.
  - **Ejemplo**: Tenemos un fichero cuyas columnas están separadas, bien por coma, o por punto y coma, y necesitamos un separador común para poder separarlas con el comando **cut**. Vamos a sustituir los punto y coma por comas y mostrar la segunda y cuarta columnas
    - **cat fichero.txt | sed -r 's/;/,/g' | cut -d "," -f 2,4**

Expresiones regulares: Tanto para borrar e imprimir líneas, como para sustitución de cadenas se pueden emplear expresiones regulares. Esto se explicará más adelante.

grep

Se utiliza para buscar las líneas que contengan un patrón (expresión regular) en un texto. **grep 'expresion'** es por tanto equivalente a **sed -n '/expresion/p'**. La expresión en grep, a diferencia de sed, no va encerrada entre barras '/'. Por lo demás es muy similar.

Se puede usar el comando **egrep**, o **grep -E**, para usar expresiones regulares extendidas. Entre otras cosas permiten el uso de caracteres como **{,},(,),|,+,?, sin la barra invertida** delante para escaparlos. Con el comando **sed**, esto se consigue con la opción **-r**.

- **-i** → No es sensible a minúsculas/mayúsculas.
- **-w** → Obliga a que las coincidencias sean una palabra entera, y no parte de una palabra. Ejemplo: **grep -w -i 'casa'** haría coincidir la palabra casa, y no casanova.

Ejemplo: buscar líneas con un dni (usamos **egrep** para expresiones regulares **extendidas**).

- **cat fichero.txt | egrep -i -w '[0-9]{8}[a-z]'**

7. Expresiones regulares

En la mayoría de los lenguajes identificaremos las expresiones regulares porque van encerradas entre las barras '/' → /**expresión**/

La expresión regular más básica es una cadena de texto simple y llana. El texto a comparar deberá coincidir exactamente con el texto:

/**casa**/ → "Vivo en una **casa** grande".

7.1 Opciones y operaciones con expresiones regulares (comando sed)

Después de la expresión regular pueden venir varias opciones. Por ejemplo la opción **d**, que además de para borrar números de líneas en concreto, también sirve para borrar las líneas que tengan alguna coincidencia con la expresión regular (**-r** sirve para poder usar expresiones regulares extendidas → más posibilidades):

sed -r '/expresión/d' → Mostraría todas las líneas excepto las que tengan coincidencia con la expresión, que las elimina.

La opción **p** se puede decir que es contraria a la anterior, ya que lo que hace es imprimir las líneas que coinciden con la expresión. Se suele usar con la opción **-n** del comando, ya que si no lo imprime todo, y no sólo lo que coincide:

sed -n -r '/expresión/p' → Mostraría las líneas que tengan alguna coincidencia con la expresión. De forma parecida funciona el comando **grep -E 'expresión'** o **egrep 'expresión'**.

El comando sed también permite la sustitución de unas cadenas por otras. La coincidencia de la cadena a sustituir también puede ser una expresión regular. Para indicar sustitución, se debe usar la opción **s** **antes** de la expresión, indicando después la cadena por la que se sustituye.

sed -r 's/casa/ mansión/g' → Sustituye la palabra casa por la palabra mansión. La opción **g** al final indica que se sustituya todas las veces que aparezca (por defecto sólo se sustituye la primera). Otras opciones que podemos usar (e incluso combinar son):

- **g** → Reemplaza todas las ocurrencias en la línea.
- **NÚMERO** → Reemplaza sólo la ocurrencia (**número indicado**). El resto no.
- **p** → Imprime las líneas en las que se haya sustituido algo (recomendable usar **-n**)
- **I (i mayúscula)** → No diferencia entre mayúsculas y minúsculas
- **echo "Casa, casa, CASA" | sed 's/casa/mansion/gI'** → "mansion, mansion, mansion"

7.2 Comodines y caracteres especiales

En las expresiones regulares tenemos una serie de caracteres especiales o comodines

que tienen un significado, y algunos deben colocarse en una posición determinada.

^ → Se debe poner como primer carácter en la expresión regular, ya que significa '**inicio de cadena**', es decir, indica que la cadena debe empezar por lo indicado en la expresión regular (aquí no tiene sentido la opción **g** ya que sólo habrá un inicio de cadena siempre).

$ → Se debe poner como último carácter. Indica que la cadena termina con la expresión regular (tampoco la opción **g** tendría sentido).

- **sed -r 's/^hola/Adiós/i'** → Sólo se sustituye hola por adiós si es la primera palabra en el texto.
- **sed -r 's/^error$/x_x/i'** → Sólo las líneas cuyo único texto sea '**error**' se sustituyen por '**x_x**'.

. (punto) → Implica cualquier carácter. Incluido el de salto de línea.

- **echo "El pato pijo era un poco pavo" | sed 's/p..o/loco/g'** → "El loco loco era un loco loco"

[lista de caracteres] → Implica un carácter (**uno sólo**) a elegir de entre la lista que pondremos entre los corchetes. No se separan (los espacios cuentan también como carácter.

- Se puede indicar un rango de caracteres, por ejemplo, de la **a,** a la **z** (minúsculas)→ **a-z**
  - **'/[0-9abc]aaa/' → 4aaa baaa caaa ~~5caaa~~**
- Si al principio de la lista usamos el carácter **^**, implica cualquier carácter **menos** los de la lista (al revés).
  - **'/[^0-9abc]aaa/' → ~~4aaa~~ jaaa**

\b → Implica el principio y final de una palabra. Si queremos buscar una coincidencia con una palabra entera y no con un trozo de palabra, basta con poner esto al principio y al final de la misma. Sin embargo, podemos ponerlo sólo al principio o al final si así lo queremos.

- **s/\bno\b/sí/gi** → Sólo sustituye la palabra "**no**" por "**sí**". **No** sustituiría, por ejemplo, **nota** por **síta**.
- Para indicar el principio y final podemos usar en su lugar **\<** y **\>**. Así **s/\<no\>/sí/gi** es equivalente a lo anterior

\w → Implica un carácter alfanumérico (números, letras y subrayado '_'). Es equivalente a utilizar **[a-zA-Z0-9_]**.

\W → Lo contrario, un carácter no alfanumérico.

\n → Carácter de salto de línea.

***** → Implica que el carácter anterior puede aparecer 0 o más veces.

- **'/a[0-9]*/' → a a6 a908345**

+ (Sin la opción -r en sed sería \+)→ El carácter anterior puede aparecer 1 o más veces.

- **'/a[0-9]+/' → ~~a~~ a6 a908345**

? (Sin la opción -r en sed sería \?) → El carácter anterior puede aparecer o no (0 o 1 vez).

- **'/a[0-9]?bc/' → abc a6bc ~~a908bc~~**

{N} (Sin la opción -r en sed sería \{N\}) → (N es un número) El carácter anterior debe aparecer un número N exacto de veces

- **'/a[0-9]{3}bc/' → ~~a6bc~~ a456bc**

{N,M} (Sin la opción -r en sed sería \{N,M\}) → (N y M son números) El carácter anterior debe aparecer entre N y M veces. Si **no especificamos** número límite se entiende como de N a infinito (o a partir de N todas las veces que se quiera).

- **'/a[0-9]{2,4}bc/' → ~~a67867bc~~ a45bc a123bc**
- **'/a[0-9]{3,}bc/' → a67867bc ~~a45bc~~ a123bc**

(expresión) (Sin la opción -r en sed sería \(expresión\)) → Los paréntesis se usan para englobar un trozo de la expresión regular y así poder aplicarle comodines como si de un carácter se tratara.

- **'/(a[0-9])+bc/' → a4a5bc ~~a1a23bc~~ a0bc**

(expr1|expr2) (Sin la opción -r en sed sería \(expr1\|expr2\)) → El carácter '|' dentro de paréntesis es un operador 'o'. Es decir, el trozo de la expresión da a elegir entre coincidir con lo que hay antes de ese carácter **o** con lo que hay después.

- **'/(ab|cd)+[0-9]/' → ~~acbd3~~ ababcdab3 ~~abc3~~**

Algunos caracteres especiales (**punto, asterísco**, etc...) deben ir "escapados", precedidos de la barra '\' si queremos representarlos como **caracteres normales** y no como caracteres especiales que definen la expresión regular. Debemos usar la opción **-r** en **sed** o la opción **-E** en **grep** (expresiones regulares extendidas) o funcionaría al revés.

Caracteres normales → '\/\(\)\{\}\+*\.\[\]'

Usando los corchetes, también disponemos de listas especiales de caracteres, en lugar de tener que ponerlas nosotros mismos.

[[:alnum:]]	Caracteres alfanuméricos [a-zA-Z0-9]
[[:alpha:]]	Caracteres alfabéticos [a-zA-Z]
[[:blank:]]	Caracteres de espaciado (espacio o tabulador)
[[:digit:]]	Números [0-9]
[[:lower:]]	Letras en minúscula [a-z]
[[:punct:]]	Caracteres de puntuación

[[:upper:]]	Letras en mayúscula [A-Z]
[[:xdigit:]]	Dígitos hexadecimales [0-9a-fA-F]

Más información básica y avanzada sobre expresiones regulares en general (inglés):

http://www.regular-expressions.info/

7.3 Comando awk

Es un comando más complejo y potente que puede hacer lo mismo y más que cut, grep y sed juntos. En este caso, sólo vamos a tratar con su capacidad de mostrar columnas de forma desordenada (cut -f 3,1 no muestra primero la 3ª y luego la 1ª).

El separador se indica con la opción -F "separador" (igual que -d en cut pero puede ser más de un carácter además). Luego indicamos cómo queremos que formatee el texto de esta forma:

Ejemplo: **awk -F ';' '{print $2" va antes que "$1}'**

El comando al contrario que cut, **SÍ** que elimina el separador. Si lo queremos volver a poner (o cualquier texto por en medio), debemos indicarlo como texto normal. El texto normal se encierra entre comillas dobles ", también podemos separar por comas, pero en este caso, el comando además introducirá un espacio de separación automáticamente.

awk -F ';' '{print $2" va antes que "$1}' <==> awk -F ';' '{print $2,"va antes que", $1}'

En resumen **$NÚMERO** hace referencia a la columna a imprimir, mientras que el texto que queramos introducir entre medias, lo debemos poner encerrado entre **comillas**.

El comando **awk** también admite operaciones matemáticas con las columnas:

awk -F ';' '{print "Precio: "$1", Cantidad: "$2". Total: "$1*$2}'

'{print toupper($1), tolower($2), $3}' → Funciones predefinidas para transformar a mayúsculas y minúsculas.

8. Visualización de memoria disponible

free

Visualiza la cantidad de memoria RAM y Swap total, ocupada y disponible en el sistema. Con la opciónes “-b”, “-k”, “-m” o “g”, visualiza estas cantidades en bytes, KB, MB o Gbytes. Si no se le indica nada nos dará la información en KB.

Un ejemplo de la salida que produce este comando (**free -m**) es el siguiente:

```
              total usado libre compartido búfer/caché disponible
Memoria:      16036  2943  9913         39        3179      12728
Swap:         14303     0 14303
```

La columna “**usado**” se refiere a la memoria utilizada por aplicaciones, archivos, etc abiertos o en ejecución en este mismo instante. Memoria **libre** se refiere a la memoria RAM que el sistema no está usando, contando la memoria ocupada por **buffers** y **caché** del sistema operativo, es decir, zona de memoria que se suele usar para acelerar la carga de aplicaciones que han sido abiertas anteriormente pero al cerrarlas el sistema no ha liberado la memoria (o no del todo)

La memoria **disponible** es aquella que realmente está libre para ejecutar más procesos si así lo requiere el sistema operativo. Es decir, si la memoria se llena, el sistema podría liberar gran parte de lo que tiene en buffer/cache para cargar nuevos procesos. En resumen, esta es la memoria que realmente tenemos para abrir nuevas aplicaciones, etc.

La segunda fila se refiere la memoria **swap** o de intercambio, partición del disco que se utiliza para guardar los datos menos utilizados de la RAM cuando esta se llena o está casi llena. El tamaño de la memoria swap está definido por el tamaño que le hayamos dado a la partición en la instalación del sistema.

df

Muestra el tamaño de las particiones montadas en el sistema (las que no estén montadas no las mostrará). También indica el tamaño utilizado y disponible. Con la opción “**-h**”, muestra los tamaños de forma más amigable, es decir, en KB, MB, o GB en función de como mejor se entienda (también están las opciones -k y -m, por ejemplo, para forzar kilobytes o megabytes).

9. Gestión de procesos

9.1 Visualizar los procesos

Una forma simple de visualizar los procesos y su relación de parentesco con otros procesos es mediante el comando **pstree**, que muestra el arbol de procesos actual. Con la opción -p nos mostrará además el PID de cada proceso.

```
pstree -p

init(1)─┬─agetty(8018)
        ├─agetty(8019)
        ├─agetty(8020)
        ├─agetty(8021)
        ├─agetty(8022)
        ├─agetty(8023)
        ├─amarokapp(9569)─┬─ruby(9582)
        │                 ├─ruby(9583)
        │                 ├─{amarokapp}(9574)
        │                 ├─{amarokapp}(9575)
        │                 ├─{amarokapp}(9576)
        │                 ├─{amarokapp}(9577)
        │                 ├─{amarokapp}(9578)
        │                 └─{amarokapp}(9725)
        ├─cron(7886)
        ├─cupsd(7543)
        ├─cvsd(7608)
         ........
```

Con el comando **ps** obtendremos información más detallada de los procesos, lo más común es utilizarlo con las opciones a, u y x, es decir “ps aux” para que nos muestre todos los procesos.

```
ps aux

USER     PID %CPU %MEM    VSZ   RSS  TTY   STAT  START   TIME  COMMAND
root       1   0.0  0.0   1556   544    ?   Ss    10:56   0:00  init [3]
root       2   0.0  0.0      0     0    ?   SN    10:56   0:00  [ksoftirqd/0]
root       3   0.0  0.0      0     0    ?   S<    10:56   0:00  [events/0]
root       4   0.0  0.0      0     0    ?   S<    10:56   0:00  [khelper]
root       5   0.0  0.0      0     0    ?   S<    10:56   0:00  [kthread]
root      55   0.0  0.0      0     0    ?   S<    10:56   0:00  [kblockd/0]
root      56   0.0  0.0      0     0    ?   S<    10:56   0:00  [kacpid]
root     157   0.0  0.0      0     0    ?   S<    10:56   0:00  [ata/0]
........
```

Las columnas de ps significan:

- **USER**: dueño del proceso
- **PID**: PID del proceso
- **%CPU**: Cantidad de CPU que está consumiendo actualmente
- **%MEM**: Cantidad de RAM que está consumiendo actualmente
- **VSZ**: Memoria total que ocuparía el proceso si se cargase entero en memoria RAM

- **RSS**: Memoria que realmente está ocupando el proceso
- **TTY**: Terminal desde la que se ejecutó el proceso (? para ninguna)
- **STAT**: Estado del proceso. Primera columna: (R → preparado, S → Dormido o Suspendido, D → Dormido y no se puede despertar, T → Parado y Z → Zombie)
- **START**: Hora a la que se inició el proceso
- **TIME**: Tiempo que el proceso ha utilizado la CPU
- **COMMAND**: Comando que inició el proceso

La memoria ocupada se indica en KB, a no ser que aparezca el sufijo m (MB).

También existe el comando **top**, que no es más que una versión interactiva de ps, en la cual la información sobre los procesos se va actualizando periodicamente. Se pueden utilizar comandos interactivos, como u seguido del nombre de un usuario para ver solamente los procesos de dicho usuario.

top

```
top -13:43:27 up  2:46,  1 user,  load average: 0.45, 0.39, 0.31
Tasks: 104 total,   4 running,  99 sleeping,   0 stopped,   1 zombie
Cpu(s):  8.7% us,  0.0% sy,  0.0% ni, 90.0% id,  0.0% wa,  0.3% hi,  1.0% si
Mem:   1034808k total,  1006960k used,    27848k free,    70704k buffers
Swap:   506036k total,      116k used,   505920k free,   479840k cached
  PID  USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 7234 root      15   0  110m   74m  5452 R  4.7  7.4   5:10.48  X
 8643 arturo    15   0  203m   73m  32m  S  1.7  7.3   2:08.98  firefox-bin
 9569 arturo    15   0  130m   46m  26m  S  1.7  4.6   0:23.86  amarokapp
 8709 arturo    15   0  35016  18m  13m  R  0.3  1.8   0:01.47  konsole
    1 root      15   0   1556   544  472  S  0.0  0.1   0:00.52  init
```

Para más información sobre **ps** y **top** consultar los respectivos man ps y man top.

Existe una versión más completa y amigable que top llamada **htop**.

9.2 Señales

Las señales sirven para que los procesos se comuniquen entre sí, y el kernel(núcleo del sistema operativo) a su vez, se comunique con los procesos. Los procesos tendrán una manera de actuar ante diferentes señales, y habrá otras que ignoren. Las señales se identifican mediante SIGNOMBRE, donde NOMBRE es el nombre de la señal. Las señales más importantes se identifican además mediante un número.

Algunas señales comunes son:

- **SIGHUP (1)**: El proceso vuelve a leer sus archivos de configuración
- **SIGINT (2)**: Interrumpir el programa (señal que se envía al presionar Ctrlc en el terminal)

- **SIGKILL (9)**: Mata un proceso, esta señal no puede ser ignorada por ningún proceso.
- **SIGTERM (15)**: A diferencia de SIGKILL, esta señal indica al proceso que termine, pero con normalidad, haciendo primero lo que tuviese que hacer. No lo mata en el acto.
- **SIGCHLD (17)**: Señal que envía un hijo a su padre cuando acaba.

Los comandos kill y killall se utilizan para enviar señales a los procesos. La forma de utilizarlos es:

kill -SEÑAL PID

killall -SEÑAL NombreProceso

La señal puede ser el nombre o el número. kill sólo mata un proceso, mientras que killall mata todos los procesos que tengan el nombre especificado. Ejemplo:

kill SIGKILL 2345

kill -9 2345

killall -9 firefox-bin (Mata todos los procesos que tengan de nombre firefox-bin)

9.3 Prioridad de un proceso

Cuando queremos que un comando se ejecute con mayor o menor prioridad, lo que indica que cuando el sistema esté saturado hay que darle mayor preferencia a unos procesos que a otros, que pueden quedarse esperando sin problemas, se utiliza el comando nice, o renice si el proceso ya se está ejecutando y queremos cambiarle la prioridad.

nice -n prioridad comando

renice prioridad PID

prioridad será un número entre -20 (máxima prioridad) y +20 (mínima prioridad). Un usuario sólo puede establecer una prioridad máxima de 0. Solamente root puede establecer prioridades negativas.

10. Ejecución en primer y segundo plano

Cuando ejecutamos un programa desde la línea de comandos (terminal o consola), este puede ejecutarse en primer o segundo plano. La diferencia consiste en que un proceso ejecutándose en primer plano bloquea el terminal o la consola en la que se ejecuta hasta que acaba, es decir, la consola no responderá a nuestras órdenes hasta que termine con el programa que está ejecutando en primer plano, o lo cerremos nosotros.

Cuando ejecutamos un programa en segundo plano, este deja libre el terminal para que se puedan seguir ejecutando otras órdenes, y al usuario se le informa del PID del proceso creado, y del número de tarea dentro de bash (ahora veremos para que sirve saber el número de tarea).

```
sleep 3 &
[1] 27154   → Número de tarea y PID del proceso sleep creado
[1]+  Done   sleep 3 → Si presionamos enter cuando ha acabado
nos informa.
```

El PID del proceso nos sirve como ya vimos, para mandarle algún tipo de señal con kill (también se puede utilizar el número de tarea → **kill %1**). El número de tarea nos sirve para poder pasar tareas de primer a segundo plano y viceversa (sólo 1 en primer plano como máximo):

- **fg %N** → lleva una tarea a primer plano (N → número de tarea).
- **bg %N** → lleva una tarea a segundo plano.

Ejemplos:

- Tarea en segundo plano (por ejemplo, tarea 1): **fg %1** la traería a primer plano.
- Tarea en primer plano (tarea 2): Presionamos **ctrl+z** para dormir el proceso. A continuación si ejecutamos **fg %2**, continuaría en primer plano, y con **bg %2** pasaría a continuar en segundo plano.

Ctrl+c → Envía la señal SIGTERM a la tarea ejecutándose en primer plano y normalmente esto la cierra.

Hay que tener en cuenta que aunque un proceso esté ejecutándose en segundo plano, seguirá escribiendo sus mensajes por pantalla, lo cual puede resultar molesto si tenemos varias tareas a la vez en segundo plano.

El comando **jobs** te muestra las tareas que hay activas en el terminal (en segundo plano o dentenidas).

11. Demonios/Servicios

Un demonio (daemon), o servicio, es un proceso que se carga normalmente al inicio del sistema, se ejecuta en segundo plano todo el tiempo, y se encarga de realizar una tarea determinada (gestión de la red, gestión de dispositivos de almacenamiento, bluetooth, impresión, conexión de dispositivos, sonido, etc...).

Para controlar estos servicios, hay métodos obsoletos, que por compatibilidad con scripts y programas más antiguos aún se mantiene como pueden ser:

/etc/init.d/demonio orden

service demonio orden

Actualmente, prácticamente todas las distribuciones han migrado del sistema SysVinit a SystemD para gestionar procesos y servicios, por lo que se recomienda el comando **sysctl** para controlar dichos servicios:

systemctl orden demonio

Órdenes:

- **start**: inicia un demonio (lo carga en memoria)
- **reload**: lee los ficheros de configuración y carga la nueva configuración (por ejemplo, si cambiamos la configuración del demonio que realiza tareas de firewall o cortafuegos).
- **stop**: detiene un demonio.
- **restart**: lo para y lo vuelve a arrancar
- **status**: informa sobre el estado actual del demonio.

Algunos demonios importantes:

- **udev** → Gestiona los dispositivos conectados a nuestra máquina. Se encarga de añadirlos al directorio /**dev**.
- **cron** → Es un demonio al que se le pueden programar tareas para realizar en fechas y horas concretas (como una copia de seguridad por ejemplo...)
- **cups** → Controla la comunicación con las impresoras conectadas al sistema directa o remotamente. Es el servidor de impresión.
- **dbus** → Provee un sistema para que las aplicaciones se comuniquen entre ellas fácilmente.
- **iptables** → iptables es un firewall (cortafuegos) incorporado el núcleo de Linux, se ejecuta como demonio.
- **samba** → Controla la configuración y gestiona la red local (compartición de recursos).

12. Variables de entorno o globales

Las variables de entorno o globales son variables a las que normalmente se les da valor cuando se inicia el sistema. Estas variables se suelen escribir en mayúsculas y conservan su valor para todos terminales abiertos, scripts, etc. Lo cual permite que varias aplicaciones accedan a ellas para conocer valores de configuración del sistema, como por ejemplo, el idioma (con el comando 'locale' se muestran todas las variables globales de idioma).

- **set** → Comando que nos permite ver todos los valores de las variables de entorno actuales, muestra también las variables locales que se hubieran definido en el shell actual y las funciones globales. Mejor visualizarlo ejecutando **set | less** por ejemplo
- **env** → Muestra sólo las variables globales definidas actualmente.
- **export VARIABLE** → Con 'export' podemos convertir una variable en variable de entorno, pero sólo dentro de la sesión del terminal donde estemos.
- **export VARIABLE=valor** → Si no tenía valor previamente, podemos darselo directamente y exportarla en un sólo paso.

La variable de entorno PATH indica al sistema donde puede buscar los comandos a ejecutar sin necesidad de indicarle la ruta completa, es decir, desde cualquier directorio. Nos indicará cada una de las rutas separadas por 2 puntos:

echo $PATH

/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games

Si queremos añadir otra ruta a la variable PATH para que busque scripts o programas que hayamos hecho nosotros hay que añadirla precedida de los 2 puntos para separar de la anterior, claro.

PATH="$PATH:$HOME/bin"

Las variables de entorno del sistema para exportar suelen indicarse dentro del fichero **/etc/profile** (sólo modificable por root), o dentro de un archivo con extensión **.sh** creado en **/etc/profile.d/** (opción recomendada) y podrán ser accedidas desde cualquier sitio por cualquier usuario. Las variables de entorno de cada usuario, se suelen definir en **$HOME/.bash_profile** o **$HOME/.bashrc**

Algunas variables de entorno comunes son:

- $**PS1**: contiene la forma del símbolo del sistema
- $**HOME**: la ruta absoluta hasta el directorio inicial del usuario actual
- $**PATH**: las rutas a los programas que hay establecidas para el usuario actual. Cuando se ejecuta un comando, se buscará en estos directorios.
- **$EDITOR**: Editor de texto predeterminado, por ejemplo "**/usr/bin/nano**".

13. Scripts de inicio de sesión

Cuando el usuario inicia sesión en el sistema, se ejecutan una serie de scripts para, entre otras cosas, cargar las variables de entorno de la sesión, pero se pueden añadir las instrucciones que interesen, como podría ser, borrar un directorio, o crear una copia de seguridad de algo, etc....

Primero se ejecuta el script /**etc**/**profile**, que incluye la ejecución de todo lo que se encuentra dentro de /**etc**/**profile.d**/. Estos scripts son comunes a todos los usuarios. Posteriormente se carga la configuración personal del usuario que ha iniciado sesión, cuyos scripts se encuentran en el directorio $HOME con estos nombres:

- **.bash_profile** → Se ejecuta en cada inicio de sesión, y sólo en ese momento Se podría usar para añadir directorios al PATH y exportarlo.
- **.bashrc** → Se ejecuta cada vez que se inicia una sesión de bash, es decir, cuando se abre un terminal, o se inicia sesión. Es preferible utilizar este.

Existen otros ficheros de **bash** útiles, además de los de inicio de sesión:

- **.bash_history** → Historial de comandos
- **.bash_logout** → Se ejecuta al cerrar la sesión bash (por ejemplo, para limpiar el historial)