

Recopilatorio de comandos para sistemas LINUX/UNIX

Martín Gil Blanco

Índice

- Sistema de ficheros de Linux
- Metacaracteres
- Expresiones regulares, grep y egrep
- Entorno
- sort
- ls
- wc
- dd
- df
- du
- alias
- which y type
- command y builing
- history
- redirecciones
- tar
- atime, mtime, ctime, touch
- Gestion de usuarios
- Permisos
- ACLs
- Repositorios
- ip e ifconfig
- wget y curl
- ss y netstat
- ssh y scp
- systemd, systemctl y /etc/init.d
- Niveles de arranque en Linux
- sed
- find
- cut
- awk
- ps, psgrep, ptree, pidof, kill, killall
- mount y umount
- crontab y at
- declare, typeset y readonly
- set ,unset y env

- [source, export y env](#)
 - [Comillas en variables de bash](#)
 - [Uso de {variable}](#)
 - [Ejecución de scripts](#)
 - [subshells](#)
 - [arrays en bash](#)
 - [Prácticas sobre arrays](#)
-

Sistema de ficheros de Linux

- **/etc/**: archivos de configuración.
 - **/bin/**: programas básicos.
 - **/lib/**: bibliotecas básicas.
 - **/sbin/**: programas de sistema.
 - **/usr/**: aplicaciones. Este directorio está subdividido en bin, sbin, lib.
 - **/tmp/**: archivos temporales. Generalmente se vacía este directorio durante el arranque.
 - **/root/**: archivos personales del administrador (root).
 - **/home/**: archivos personales de los usuarios.
 - **/dev/**: archivos de dispositivo.
 - **/mnt/**: punto de montaje temporal.
 - **/media/**: puntos de montaje para dispositivos removibles (CD-ROM, llaves USB, etc.).
 - **/opt/**: aplicaciones adicionales provistas por terceros.
 - **/run/**: datos volátiles en tiempo de ejecución que no persisten entre reinicios.
 - **/srv/**: datos utilizados por los servidores en este sistema.
 - **/var/**: datos variables administrados por demonios. Esto incluye archivos de registro, colas, cachés, etc.
 - **/boot/**: núcleo Linux y otros archivos necesarios para las primeras etapas del proceso de arranque.
 - **/proc/** y **/sys/** son específicos del núcleo Linux (y no son parte del FHS). El núcleo los utiliza para exportar datos a espacio de usuario.
-

Metacaracteres

Son caracteres con significado especial que representan clases de caracteres o repeticiones.

- Algunos metacaracteres comunes incluyen:
 - **.**: coincide con cualquier carácter excepto nueva línea.
 - *****: coincide con cero o más repeticiones del elemento anterior.

- `+`: coincide con una o más repeticiones del elemento anterior.
- `?`: coincide con cero o una repetición del elemento anterior.
- `[]`: coincide con cualquier carácter dentro del conjunto especificado.
- `^`: coincide con el inicio de una línea.
- `$`: coincide con el final de una línea.

1. Asterisco (*)

- `ls *.txt`: Lista todos los archivos con extensión ".txt" en el directorio actual.
- `cp file* directory/`: Copia todos los archivos cuyo nombre comience con "file" al directorio especificado.

2. Signo de interrogación (?)

- `ls file?.txt`: Lista archivos como "file1.txt", "fileA.txt", pero no "file10.txt".
- `rm file?.txt`: Elimina archivos como "file1.txt", "fileA.txt", pero no "file10.txt".

3. Corchetes ([])

- `ls [aeiou]*`: Lista archivos cuyos nombres comiencen con una vocal, es decir, puede ser solo un caracter igual a una de las vocales indicadas.
- `rm [0-9]*`: Elimina archivos cuyos nombres comiencen con un dígito.

4. Barra vertical (|)

- `ls -l | grep filename`: Muestra detalles de archivos que contienen "filename".
- `cat file.txt | sed 's/old/new/g'`: Reemplaza todas las instancias de "old" con "new" en el contenido del archivo.

5. Punto y coma (;)

- `mkdir folder1 ; cp file.txt folder1/`: Crea un directorio y copia un archivo en una sola línea.
- `rm *.txt ; rm *.csv`: Elimina todos los archivos con extensión ".txt" y ".csv".

6. Ampersand (&)

- `./script.sh &`: Ejecuta un script en segundo plano.
- `make &`: Compila un programa en segundo plano mientras se realizan otras tareas en el terminal.

Expresiones regulares, grep y egrep

Las expresiones regulares (regex o regexp) son patrones utilizados para encontrar una determinada secuencia de caracteres dentro de una cadena de texto. Son ampliamente utilizadas en la búsqueda y manipulación de cadenas de texto en diversos contextos de programación y procesamiento de datos. A continuación, se presenta una guía básica de los elementos fundamentales de las expresiones regulares:

1. Caracteres Literales

- `.`: Representa cualquier caracter excepto un salto de línea.
- `[]`: Define un conjunto de caracteres permitidos en esa posición.

2. Conjuntos de Caracteres

- **[rfc]+**: 1 o más coincidencias de 'r', 'f' o 'c'.
- **[a-z]+**: 1 o más letras minúsculas.
- **[A-Z]+**: 1 o más letras mayúsculas.
- **[a-Z]+**: 1 o más letras, tanto minúsculas como mayúsculas.
- **[^rfc]+**: 1 o más caracteres que no sean 'r', 'f' o 'c'.
- **[^a-z]?**: 0 o 1 caracter que no sea una letra minúscula.
- **[^A-Z]?**: 0 o 1 caracter que no sea una letra mayúscula.

3. Anclas de Línea

- **^**: Indica el inicio de una línea.
- **\$**: Indica el final de una línea.

4. Operadores de Cantidad

- *****: 0 o más coincidencias del patrón anterior.

5. Grupos

- **()**: Permite agrupar expresiones regulares.

6. Caracteres de Escape

- ****: Escapa un metacarácter para que sea interpretado literalmente.

7. Repeticiones

- **{n}**: Exactamente n coincidencias del patrón anterior.
- **{n,}**: Como mínimo n coincidencias del patrón anterior.
- **{n,m}**: Entre n y m coincidencias del patrón anterior.

Ejemplos

- **[rfc]***: Cero o más ocurrencias de 'r', 'f' o 'c'.
- **(rfc)**: Agrupa los caracteres 'rfc'.
- **(r.c)**: Agrupa un conjunto de tres caracteres donde el primero es 'r', el tercero es 'c' y el segundo puede ser cualquier caracter.
- **{2}**: Exactamente 2 caracteres minúsculos.
- **[a-z]{2,}**: Como mínimo 2 caracteres minúsculos.
- **[a-z]{2,4}**: Entre 2 y 4 caracteres minúsculos.

Clases de Caracteres POSIX

Las clases de caracteres POSIX son atajos para definir conjuntos de caracteres comunes:

- **[[:lower:]]**: [a-z].
- **[[:upper:]]**: [A-Z].

- **[[:alpha:]]**: [A-Za-z] o [[:lower:]] + [[:upper:]].
- **[[:digit:]]**: [0-9].
- **[[:xdigit:]]**: [0-9A-Fa-f].
- **[[:alnum:]]**: [0-9A-Za-z] o [[:alpha:]] + [[:digit:]].
- **[[:blank:]]**: Caracteres de espacio y tabulado.
- **[[:cntrl:]]**: Caracteres de control.
- **[[:punct:]]**: Caracteres de puntuación, equivalente a los símbolos de puntuación comunes.
- **[[:graph:]]**: [[:alnum:]] + [[:punct:]].
- **[[:print:]]**: [[:alnum:]] + [[:punct:]] + espacio.
- **[[:space:]]**: Caracteres de espacio en blanco, como tabuladores, saltos de línea, etc.

Uso de **grep** y **egrep**

grep es una herramienta de línea de comandos que busca patrones en archivos o en la salida de otros comandos. **egrep** es una versión extendida de **grep** que admite una sintaxis de expresiones regulares más amplia.

```
grep [opciones] patrón [archivo...]
```

- **Opciones Comunes:**
 - **-i**: Ignora mayúsculas y minúsculas.
 - **-v**: Invierte la búsqueda para mostrar líneas que no coincidan.
 - **-n**: Muestra el número de línea junto con la coincidencia.
 - **-E**: Esta opción indica a **grep** que interprete el patrón de búsqueda como una expresión regular extendida (ERE), lo que significa que puedes utilizar una sintaxis más avanzada de expresiones regulares.
 - **-o**: Esta opción le indica a **grep** que solo muestre las partes de las líneas que coinciden con el patrón de búsqueda, en lugar de toda la línea. Esto es útil cuando solo estás interesado en ver qué partes de las líneas coinciden con tu patrón.

Ejemplos de **grep**

- Buscar una palabra en un archivo:

```
grep "patrón" archivo.txt
```

- Buscar una palabra en varios archivos:

```
grep "patrón" archivo1.txt archivo2.txt
```

Uso de **egrep**

egrep es una versión más potente de **grep** que admite una sintaxis extendida de expresiones regulares sin necesidad de escapar ciertos caracteres.

```
egrep [opciones] patrón [archivo...]
```

Ejemplos de **egrep**

- Buscar múltiples palabras en un archivo:

```
egrep "patrón1|patrón2" archivo.txt
```

- Buscar una palabra ignorando mayúsculas y minúsculas:

```
egrep -i "patrón" archivo.txt
```

Entorno

Variables de Entorno en Linux

Listado de variables de entorno interesantes:

- **SHELL**: El nombre de la shell por defecto del usuario.
- **HOME**: El nombre de la ruta de tu directorio personal.
- **LANG / LANGUAGE**: Define el conjunto de caracteres y el orden de cotejo de su idioma.
- **PATH**: Una lista separada por dos puntos de los directorios que se buscan cuando introducimos el nombre de un programa ejecutable.
- **PWD**: El directorio de trabajo actual.
- **_**: Una referencia al programa /usr/bin/printenv (prueba a ejecutar el comando `$_`).
- **USER**: Tu nombre de usuario.

Configuración de Archivos de Entorno en Linux

Ficheros con aplicación a todo el sistema (todos los usuarios del sistema)

- **/etc/enviroment**: Fichero específico para la definición de variables de entorno. No puede contener scripts. Se ejecuta al iniciar el sistema.
- **/etc/profile**: Permite definir variables de entorno y scripts, aunque no es apropiado modificar este fichero directamente (debe crearse un nuevo fichero en /etc/profile.d). Se ejecuta en shells con login.

- **/etc/profile.d:** Contiene scripts que se ejecutan en shells con login.
- **/etc/bash.bashrc:** Permite definir variables de entorno y scripts que estarán disponibles para programas iniciados desde la shell bash. Las variables que se definan en este fichero no van a estar disponibles para programas iniciados desde la interfaz gráfica. No se ejecuta en shells con login.

Ficheros con aplicación a un usuario específico

- **~/.profile:** Permite definir variables de entorno y scripts. Se ejecutan al iniciar la sesión de Escritorio o en una shell con login. Las variables afectan a todos los programas ejecutados desde el escritorio gráfico o desde la shell.
- **~/.bash_profile, ~/.bash_login:** Permiten definir variables de entorno y scripts. Se ejecutan cuando se utiliza una shell con login. Las variables definidas solo afectaran a los programas ejecutados desde bash.
- **~/.bashrc:** Permite definir variables de entorno y scripts. Se ejecuta cuando se abre la shell sin necesidad de hacer login. Las variables definidas solo afectaran a los programas ejecutados desde bash.

sort

El comando sort ordena líneas de texto en un archivo o entrada estándar según un criterio específico, como orden alfabético (predeterminado) o numérico. Parámetros:

1. -k: indica la columna por la que se realiza la ordenación.

```
martin@debian12:/tmp/prueba$ echo -e "1 2 3 4 \n4 3 2 1\n11 22 33 44" | sort -k1
1 2 3 4
11 22 33 44
4 3 2 1
```

2. -n: Indica que se debe realizar una ordenación numérica en lugar de una ordenación alfabética.

```
martin@debian12:/tmp/prueba$ echo -e "1 2 3 4 \n4 3 2 1\n11 22 33 44" | sort -k1n
1 2 3 4
4 3 2 1
11 22 33 44
```

3. -t: Establece el delimitador.

```
martin@debian12:/tmp/prueba$ echo -e "1,2,3,4 \n4,3,2,1\n11,22,33,44" | sort -t',' -k1n
1,2,3,4
4,3,2,1
11,22,33,44
```

4. -f: Ignora diferencia entre mayúsculas y minúsculas.

```
martin@debian12:/tmp/prueba$ echo -e "1 2 3 A\n4 3 2 a" | sort -k4 -f
1 2 3 A
4 3 2 a

martin@debian12:/tmp/prueba$ echo -e "1 2 3 a\n4 3 2 A" | sort -k4 -f
1 2 3 a
4 3 2 A
```

5. -r: Realiza la ordenación inversa.

```
martin@debian12:/tmp/prueba$ echo -e "a b c\nA B C" | sort -k1
a b c
A B C

martin@debian12:/tmp/prueba$ echo -e "a b c\nA B C" | sort -k1 -r
A B C
a b c
```

6. -u: Ignora duplicados. Lo que significa que da el mismo resultado que utilizar el comando `uniq`.

```
martin@debian12:/tmp/prueba$ echo -e "a b c\nA B C\na b c" | sort -u
a b c
A B C
```

Otros

```
martin@debian12:/tmp/prueba$ echo -e "1 2 3 A\n4 3 2 a\n1 2 3 4" | sort -k4
1 2 3 4
4 3 2 a
1 2 3 A

martin@debian12:/tmp/prueba$ echo -e "1 2 3 A\n4 3 2 a\n1 2 3 4" | sort -k4n
1 2 3 A
4 3 2 a
1 2 3 4
```

ls

El comando `ls` lista archivos y directorios en un directorio especificado.


```
martin@debian12:~$ ls -l
total 36
drwxr-xr-x 2 martin martin 4096 mar  4 09:58 d1
drwxr-xr-x 2 martin martin 4096 feb  2 18:10 Descargas
drwxr-xr-x 2 martin martin 4096 feb  2 18:10 Documentos
drwxr-xr-x 2 martin martin 4096 feb  2 18:10 Escritorio
-rw-r--r-- 1 martin martin   0 abr 12 16:29 fichero.txt
drwxr-xr-x 2 martin martin 4096 feb  2 18:10 Imágenes
drwxr-xr-x 2 martin martin 4096 feb  2 18:10 Música
drwxr-xr-x 2 martin martin 4096 feb  2 18:10 Plantillas
drwxr-xr-x 2 martin martin 4096 feb  2 18:10 Público
drwxr-xr-x 2 martin martin 4096 feb  2 18:10 Vídeos
```

Nota: Se refiere el campo de fecha a la hora en la que se hizo la última modificación de contenido del fichero o directorio (creación o eliminación de contenido en su interior).

Parámetros bien conocidos:

- -l: Información extendida.
- -h: Tamaño del archivo en formato humano.
- -i: Número de inodo.
- -a: Archivos ocultos.
- -r: Inverso.
- --sort: Para ordenar.
- --size: En función del tamaño.
- --format: Para dar un formato a la salida.
- --time: Para mostrar por atime, mtime, o ctime.
- -1: Muestra en vertical la salida.

1. -t: Lista los ficheros o directorios más nuevos al principio.

```
martin@debian12:~$ ls -l -t
martin@debian12:~$ ls -l --sort=time
```

```
martin@debian12:~$ ls -l -t
total 36
-rw-r--r-- 1 martin martin   0 abr 12 16:29 fichero.txt
drwxr-xr-x 2 martin martin 4096 mar  4 09:58 d1
drwxr-xr-x 2 martin martin 4096 feb  2 18:10 Descargas
drwxr-xr-x 2 martin martin 4096 feb  2 18:10 Documentos
drwxr-xr-x 2 martin martin 4096 feb  2 18:10 Escritorio
drwxr-xr-x 2 martin martin 4096 feb  2 18:10 Imágenes
drwxr-xr-x 2 martin martin 4096 feb  2 18:10 Música
drwxr-xr-x 2 martin martin 4096 feb  2 18:10 Plantillas
drwxr-xr-x 2 martin martin 4096 feb  2 18:10 Público
drwxr-xr-x 2 martin martin 4096 feb  2 18:10 Vídeos
```

2. -S: Lista por tamaño de mayor a menor.

```
martin@debian12:~$ ls -l -S
martin@debian12:~$ ls -l --sort=size
```

```
martin@debian12:~$ ls -l -S
total 36
drwxr-xr-x 2 martin martin 4096 mar  4 09:58 d1
drwxr-xr-x 2 martin martin 4096 feb  2 18:10 Descargas
drwxr-xr-x 2 martin martin 4096 feb  2 18:10 Documentos
drwxr-xr-x 2 martin martin 4096 feb  2 18:10 Escritorio
drwxr-xr-x 2 martin martin 4096 feb  2 18:10 Imágenes
drwxr-xr-x 2 martin martin 4096 feb  2 18:10 Música
drwxr-xr-x 2 martin martin 4096 feb  2 18:10 Plantillas
drwxr-xr-x 2 martin martin 4096 feb  2 18:10 Público
drwxr-xr-x 2 martin martin 4096 feb  2 18:10 Vídeos
-rw-r--r-- 1 martin martin   0 abr 12 16:29 fichero.txt
```

3. -m: Listado de elementos separados por comas.

```
martin@debian12:~$ ls -m
martin@debian12:~$ ls -l --format=commas
```

```
martin@debian12:~$ ls -m
d1, Descargas, Documentos, Escritorio, fichero.txt, Imágenes, Música, Plantillas,
Público, Vídeos
```

4. Vertical, Horizontal y una columna: El listado se realiza por línea y no por columna.

```
martin@debian12:~$ ls -x
martin@debian12:~$ ls -l --format=horizontal
```

```
martin@debian12:~$ ls
d1          Documentos fichero.txt  Música     Público
Descargas  Escritorio  Imágenes   Plantillas Vídeos

martin@debian12:~$ ls --format=vertical
d1          Documentos fichero.txt  Música     Público
Descargas  Escritorio  Imágenes     Plantillas Vídeos

martin@debian12:~$ ls -x
```

```
d1      Descargas Documentos Escritorio fichero.txt Imágenes Música
Plantillas
Público Vídeos

martin@debian12:~$ ls --format=horizontal
d1      Descargas Documentos Escritorio fichero.txt Imágenes Música
Plantillas
Público Vídeos

martin@debian12:~$ ls --format=single-column
d1
Descargas
Documentos
Escritorio
fichero.txt
Imágenes
Música
Plantillas
Público
Vídeos
```

WC

El comando **wc** en sistemas Unix/Linux es una herramienta muy útil para contar palabras, líneas y caracteres en archivos de texto.

- **Opciones comunes:**

- **-l**: Muestra solo el recuento de líneas.
- **-w**: Muestra solo el recuento de palabras.
- **-c**: Muestra solo el recuento de bytes (caracteres).
- **-m**: Muestra el recuento de caracteres de ancho.
- **-L**: Muestra la longitud de la línea más larga.

```
si@si-VirtualBox:/tmp$ wc -lwc prueba.txt
 2   9 118 prueba.txt
```

dd

El comando **dd** se utiliza principalmente para copiar y convertir archivos de datos, con opciones muy flexibles para manejar bloques de datos.

- **Uso básico:** El formato básico del comando **dd** es:

```
dd if=archivo_de_entrada of=archivo_de_salida
```

Esto copia el contenido del archivo de entrada (**if** significa "input file") al archivo de salida (**of** significa "output file"). Por ejemplo:

```
dd if=/dev/sda of=copia_de_seguridad.img
```

Esto copiaría todo el contenido del disco `/dev/sda` en el archivo `copia_de_seguridad.img`.

- **Parámetros adicionales:** **dd** puede aceptar una serie de parámetros para personalizar su comportamiento, como el tamaño del bloque, el desplazamiento, etc. Por ejemplo:

```
dd if=archivo_de_entrada of=archivo_de_salida bs=tamaño_del_bloque  
count=número_de_bloques
```

Aquí, **bs** es el tamaño del bloque (block size) y **count** es el número de bloques. Esto puede ser útil cuando se quiere ajustar el rendimiento de la copia o la conversión.

df

El comando **df** en sistemas Unix/Linux se utiliza para mostrar el espacio disponible y utilizado en los sistemas de archivos montados. Proporciona información sobre el uso del disco a nivel de sistema de archivos. Aquí te muestro cómo se usa y algunas de sus opciones más comunes:

Opciones comunes:

- **-h** (human-readable): Esta opción muestra los tamaños en un formato legible para humanos (por ejemplo, KB, MB, GB).
- **-T** (print-type): Muestra el tipo de sistema de archivos.
- **-i** (inodes): Muestra información sobre el número de inodos utilizados y disponibles en lugar de la información sobre el espacio de disco.

```
si@si-VirtualBox:/tmp$ df
Filesystem      1K-blocks    Used Available Use% Mounted on
tmpfs           400104      1468   398636    1% /run
/dev/sda3       50770432 16547340  31611688  35% /
tmpfs           2000504        0   2000504    0% /dev/shm
tmpfs           5120         4     5116    1% /run/lock
/dev/sda2       524252     6220   518032    2% /boot/efi
tmpfs           400100        80   400020    1% /run/user/128
tmpfs           400100        68   400032    1% /run/user/1000
```

```
si@si-VirtualBox:/tmp$ df -h
Filesystem      Size  Used Avail Use% Mounted on
tmpfs           391M  1,5M  390M   1% /run
/dev/sda3       49G   16G   31G  35% /
tmpfs           2,0G    0  2,0G   0% /dev/shm
```

```
tmpfs          5,0M  4,0K  5,0M   1% /run/lock
/dev/sda2      512M  6,1M  506M   2% /boot/efi
tmpfs          391M   80K  391M   1% /run/user/128
tmpfs          391M   68K  391M   1% /run/user/1000

si@si-VirtualBox:/tmp$ df -Th
Filesystem      Type      Size  Used Avail Use% Mounted on
tmpfs           tmpfs     391M  1,5M  390M   1% /run
/dev/sda3       ext4      49G   16G   31G  35% /
tmpfs           tmpfs     2,0G    0  2,0G   0% /dev/shm
tmpfs           tmpfs     5,0M  4,0K  5,0M   1% /run/lock
/dev/sda2       vfat      512M  6,1M  506M   2% /boot/efi
tmpfs           tmpfs     391M   80K  391M   1% /run/user/128
tmpfs           tmpfs     391M   68K  391M   1% /run/user/1000

si@si-VirtualBox:/tmp$ df -Tih
Filesystem      Type      Inodes  IUsed  IFree  IUse% Mounted on
tmpfs           tmpfs     489K    999   488K    1% /run
/dev/sda3       ext4     3,1M   244K   2,9M    8% /
tmpfs           tmpfs     489K     1   489K    1% /dev/shm
tmpfs           tmpfs     489K     5   489K    1% /run/lock
/dev/sda2       vfat         0     0     0    - /boot/efi
tmpfs           tmpfs     98K    82   98K    1% /run/user/128
tmpfs           tmpfs     98K    74   98K    1% /run/user/1000
```

du

El comando **du**, que significa "disk usage" (uso de disco), se utiliza para estimar el espacio utilizado por archivos y directorios en el sistema de archivos.

- **Uso básico:** La forma más básica de utilizar **du** es simplemente invocarlo con el nombre de un directorio:

```
du directorio
```

Esto mostrará el espacio utilizado por ese directorio y sus subdirectorios. Por ejemplo:

```
du /home/usuario
```

Esto mostrará el espacio utilizado por el directorio **/home/usuario** y sus subdirectorios.

- **Opciones adicionales:** **du** puede tomar varias opciones para personalizar su salida. Por ejemplo, **-h** (human-readable) muestra los tamaños de una manera más legible para los humanos, mientras que **-a** muestra el tamaño de cada archivo en un directorio:

```
si@si-VirtualBox:/tmp$ du -h prueba/
64K prueba/

si@si-VirtualBox:/tmp$ du -ah prueba/
4,0K prueba/fichero1.txt
4,0K prueba/fichero9.txt
4,0K prueba/fichero13.txt
4,0K prueba/fichero6.txt
4,0K prueba/fichero4.txt
4,0K prueba/fichero10.txt
4,0K prueba/fichero2.txt
4,0K prueba/fichero7.txt
4,0K prueba/fichero8.txt
4,0K prueba/fichero11.txt
4,0K prueba/fichero15.txt
4,0K prueba/fichero12.txt
4,0K prueba/fichero3.txt
4,0K prueba/fichero14.txt
4,0K prueba/fichero5.txt
64K prueba/
```

alias

Considerando que en ~/.bashrc existe el `alias listar='ls -ltr'`, se ejecuta el comando `alias listar='ls -lahi --color'`. Si por consola lanzamos en ese momento el comando obtenemos:

```
martin@debian12:~$ alias listar='ls -lahi --color'
martin@debian12:~$ listar
total 156K
392450 drwx----- 16 martin martin 4,0K abr 12 16:29 .
392449 drwxr-xr-x  3 root  root  4,0K feb 17 18:08 ..
392746 -rw-----  1 martin martin 1,5K mar  5 16:11 .bash_history
392451 -rw-r--r--  1 martin martin  220 feb  2 17:57 .bash_logout
392453 -rw-r--r--  1 martin martin 3,5K feb  2 17:57 .bashrc
392470 drwxr-xr-x  8 martin martin 4,0K feb  2 18:11 .cache
392460 drwx-----  7 martin martin 4,0K feb  2 18:10 .config
```

Si en ese momento ejecutamos el comando `source ~/.bashrc && listar` se ha recargado el .bashrc y por lo tanto el alias es `listar='ls -ltr'`.

```
martin@debian12:~$ alias
total 36
drwxr-xr-x 2 martin martin 4096 feb  2 18:10 Vídeos
drwxr-xr-x 2 martin martin 4096 feb  2 18:10 Público
drwxr-xr-x 2 martin martin 4096 feb  2 18:10 Plantillas
drwxr-xr-x 2 martin martin 4096 feb  2 18:10 Música
drwxr-xr-x 2 martin martin 4096 feb  2 18:10 Imágenes
```

```
drwxr-xr-x 2 martin martin 4096 feb  2 18:10 Escritorio
drwxr-xr-x 2 martin martin 4096 feb  2 18:10 Documentos
drwxr-xr-x 2 martin martin 4096 feb  2 18:10 Descargas
drwxr-xr-x 2 martin martin 4096 mar  4 09:58 d1
-rw-r--r-- 1 martin martin   0 abr 12 16:29 fichero.txt
```

```
martin@debian12:~$ alias listar='ls -lhai'
martin@debian12:~$ alias directorioTrabajo='pwd'
martin@debian12:~$ alias
alias directorioTrabajo='pwd'
alias listar='ls -lhai'
martin@debian12:~$ unalias directorioTrabajo
martin@debian12:~$ alias
alias listar='ls -lhai'
martin@debian12:~$ unalias -a
martin@debian12:~$ alias
```

which y type

Ambos comandos, **which** y **type**, son útiles para identificar la ubicación de un comando específico y para determinar cómo se interpretará un comando en particular.

1. **which**:

- **which** es un comando que se utiliza para encontrar la ubicación de un comando ejecutable en el sistema de archivos del usuario. Devuelve la ruta completa del ejecutable de ese comando si está presente en alguna de las rutas de búsqueda del sistema.

Por ejemplo:

```
si@si-VirtualBox:/tmp$ which ls
/usr/bin/ls
```

2. **type**:

- **type** es un comando que no solo muestra la ubicación de un comando ejecutable, sino también cómo será interpretado por el shell. Puede ser un comando interno del shell, un comando externo (ubicación del archivo ejecutable) o una función shell definida por el usuario.

Por ejemplo:

```
si@si-VirtualBox:/tmp$ type ls
ls is aliased to `ls --color=auto'
si@si-VirtualBox:/tmp$ type -a ls
ls is aliased to `ls --color=auto'
```

```
ls is /usr/bin/ls
ls is /bin/ls
```

command y builtin

El orden de ejecución en Linux es:

- **alias:**
 - Localización: \$HOME/.bashrc, /etc/bashrc
 - Descripción: Crea un alias para un comando.
- **keywords:**
 - Palabras clave: function, if, for...
 - Descripción: Palabras reservadas del lenguaje de scripting de Bash.
- **functions:**
 - Funciones: nombre_funcion() {...}
 - Descripción: Define funciones personalizadas en Bash.
- **builtin:**
 - Descripción: Comandos internos de Bash que siempre están cargados en memoria.
- **file:**
 - Descripción: Scripts y programas ejecutables (según PATH).

command:

- Descripción: Busca comandos builtin y files para su ejecución.

builtin:

- Descripción: Busca comandos internos (builtin) para su ejecución.

```
martin@debian12:~$ type -a ls
ls es un alias de `ls --color=auto'
ls is /usr/bin/ls
ls is /bin/ls
```

```
martin@debian12:~$ command ls
d1      Documentos  fichero.txt  Música  Público
Descargas  Escritorio  Imágenes    Plantillas  Vídeos
```

```
martin@debian12:~$ type -a cd
cd es una orden interna del shell
martin@debian12:~$ command cd
martin@debian12:~$ builtin cd
```


history

El comando history en Linux muestra el historial de comandos ejecutados en la sesión actual del shell.

- La opción -c limpia el historial de comandos, borrando todos los comandos almacenados.
 - La opción -a guarda los comandos introducidos desde el último borrado en el archivo de historial, usualmente ~/.bash_history.
 - HISTFILE: Especifica la ubicación del archivo donde se guarda el historial de comandos.
 - HISTSIZE: Determina el número máximo de comandos almacenados en el historial.
 - !!: Atajo para repetir el último comando ejecutado.
 - !15: Ejecuta el comando número 15 del historial.
 - !ls: Ejecuta el último comando que comienza con "ls" en el historial.
-

redirecciones

```
martin@debian12:~$ ls -l > fichero.txt
martin@debian12:~$ ls -l >> fichero.txt
martin@debian12:~$ ls -li > fichero.txt 2>&1
martin@debian12:~$ ls -li &> fichero.txt
martin@debian12:~$ cat < fichero.txt
```

```
martin@debian12:~$ cat > fichero.txt << VAI
> hola
> Que
> Tal?
> VAI
martin@debian12:~$ cat fichero.txt
hola
Que
Tal?
```

```
martin@debian12:~$ cat /etc/passwd | tail -2 | tee /tmp/pass.tmp
martin:x:1000:1000:martin,,,:/home/martin:/bin/bash
vboxadd:x:999:1::/var/run/vboxadd:/bin/false

martin@debian12:~$ cat /tmp/pass.tmp
martin:x:1000:1000:martin,,,:/home/martin:/bin/bash
vboxadd:x:999:1::/var/run/vboxadd:/bin/false

martin@debian12:~$ sudo cat /etc/shadow | tail -2 | tee -a /tmp/pass.tmp
martin:$y$j9T$D1YstIGHwPXktsEmolZg./$I7fKcY0m9yE2LYgGBEn8yolExy5PLvBTI1Zf5keudM3:19770:0:99999:7:::
vboxadd:!:19755:::::::

martin@debian12:~$ cat /tmp/pass.tmp
martin:x:1000:1000:martin,,,:/home/martin:/bin/bash
```

```
vboxadd:x:999:1::/var/run/vboxadd:/bin/false
martin:$y$j9T$D1YstIGHwPXktsEmolZg./$I7fKcY0m9yE2LYgGBEn8yo1Exy5PLvBTI1Zf5keudM3:1
9770:0:99999:7:::
vboxadd:!:19755:::~:
```

tar

El comando **tar** se utiliza en sistemas Linux y Unix para crear, listar, extraer y manipular archivos y directorios empaquetados en un solo archivo.

Parámetros

- **c**: Crea un nuevo archivo tar.
- **x**: Extrae los archivos de un archivo tar.
- **t**: Lista el contenido de un archivo tar.
- **v**: Muestra información detallada (verbose) sobre las operaciones realizadas.
- **f**: Especifica el nombre del archivo tar que se va a crear, extraer o manipular.
- **j**: Utiliza el formato bzip2 para comprimir o descomprimir el archivo tar.
- **z**: Utiliza el formato gzip para comprimir o descomprimir el archivo tar.

Proceso de compresión con **tar**

```
martin@debian12:/tmp/prueba$ tar cvfz fichero.tar.gz fichero*.txt
fichero1.txt
fichero2.txt
fichero3.txt
fichero4.txt
fichero5.txt

martin@debian12:/tmp/prueba$ tar cvfj fichero.tar.bz2 fichero*.txt
fichero1.txt
fichero2.txt
fichero3.txt
fichero4.txt
fichero5.txt

martin@debian12:/tmp/prueba$ ls -lh
total 28K
-rw-r--r-- 1 martin martin  5 abr 13 08:52 fichero1.txt
-rw-r--r-- 1 martin martin  5 abr 13 08:52 fichero2.txt
-rw-r--r-- 1 martin martin  5 abr 13 08:52 fichero3.txt
-rw-r--r-- 1 martin martin  5 abr 13 08:52 fichero4.txt
-rw-r--r-- 1 martin martin  5 abr 13 08:52 fichero5.txt
-rw-r--r-- 1 martin martin 201 abr 13 08:53 fichero.tar.bz2
-rw-r--r-- 1 martin martin 179 abr 13 08:53 fichero.tar.gz
```

Proceso de listado de contenido con **tar**

```
martin@debian12:/tmp/prueba$ tar tf fichero.tar.gz
fichero1.txt
fichero2.txt
fichero3.txt
fichero4.txt
fichero5.txt

martin@debian12:/tmp/prueba$ tar tvf fichero.tar.gz
-rw-r--r-- martin/martin      5 2024-04-13 08:52 fichero1.txt
-rw-r--r-- martin/martin      5 2024-04-13 08:52 fichero2.txt
-rw-r--r-- martin/martin      5 2024-04-13 08:52 fichero3.txt
-rw-r--r-- martin/martin      5 2024-04-13 08:52 fichero4.txt
-rw-r--r-- martin/martin      5 2024-04-13 08:52 fichero5.txt

martin@debian12:/tmp/prueba$ tar tvfz fichero.tar.gz
-rw-r--r-- martin/martin      5 2024-04-13 08:52 fichero1.txt
-rw-r--r-- martin/martin      5 2024-04-13 08:52 fichero2.txt
-rw-r--r-- martin/martin      5 2024-04-13 08:52 fichero3.txt
-rw-r--r-- martin/martin      5 2024-04-13 08:52 fichero4.txt
-rw-r--r-- martin/martin      5 2024-04-13 08:52 fichero5.txt

martin@debian12:/tmp/prueba$ tar tvfj fichero.tar.gz
bzip2: (stdin) is not a bzip2 file.
tar: Child returned status 2
tar: Error is not recoverable: exiting now
```

Proceso de descompresión con `tar`

```
martin@debian12:/tmp/prueba$ ls
fichero.tar.bz2  fichero.tar.gz

martin@debian12:/tmp/prueba$ tar xvfz fichero.tar.gz
fichero1.txt
fichero2.txt
fichero3.txt
fichero4.txt
fichero5.txt

martin@debian12:/tmp/prueba$ ls
fichero1.txt  fichero3.txt  fichero5.txt  fichero.tar.gz
fichero2.txt  fichero4.txt  fichero.tar.bz2

martin@debian12:/tmp/prueba$ mkdir /tmp/temporal && tar xvfj fichero.tar.bz2 -C
/tmp/temporal
fichero1.txt
fichero2.txt
fichero3.txt
fichero4.txt
fichero5.txt
```

```
martin@debian12:/tmp/prueba$ ls /tmp/temporal/
fichero1.txt fichero2.txt fichero3.txt fichero4.txt fichero5.txt
```

Importante lo siguiente en **tar** si se quiere usar el **-** con los parametros. Siempre que usemos **-** tenemos que poner el orden de los parametros **cvzf**, es decir, siempre la **f** que indica el fichero resultante al final.

```
martin@debian12:/tmp/prueba$ tar -cvzf fichero.tar.gz fichero*.txt
fichero1.txt
fichero2.txt
fichero3.txt
fichero4.txt
fichero5.txt

martin@debian12:/tmp/prueba$ ls
fichero1.txt fichero3.txt fichero5.txt
fichero2.txt fichero4.txt fichero.tar.gz
```

atime, mtime, ctime, touch

- **atime (Access Time)**: Indica la última vez que se accedió al contenido del archivo.
- **mtime (Modified Time)**: Representa la última vez que el contenido del archivo fue modificado.
- **ctime (Change Time)**: Refleja el tiempo en que se cambiaron los metadatos del archivo, es decir, a nivel de permisos, propietario. Podemos verlo como cambios a nivel de inodo.

Nota: El inodo en Linux es una estructura de datos que almacena información importante sobre archivos y directorios en un sistema de archivos. Cada archivo y directorio en un sistema de archivos Linux está asociado con un inodo único. El inodo contiene metadatos sobre el archivo o directorio, como permisos, tamaño, propietario, tipo de archivo, fechas de acceso, modificación y cambio, y punteros a bloques de datos que contienen el contenido real del archivo o la lista de nombres de archivos en un directorio. Los inodos son cruciales para la gestión y organización de los archivos en el sistema de archivos, permitiendo al sistema operativo acceder eficientemente a la información y los datos de los archivos.

```
martin@debian12:/tmp/temporal$ stat fichero1.txt
Fichero: fichero1.txt
Tamaño: 5          Bloques: 8          Bloque E/S: 4096   fichero regular
Device: 8,1        Inode: 1700631      Links: 1
Acceso: (0644/-rw-r--r--) Uid: ( 1000/ martin)  Gid: ( 1000/ martin)
Acceso: 2024-04-13 09:23:48.618999425 +0200
Modificación: 2024-04-13 08:52:06.000000000 +0200
Cambio: 2024-04-13 09:23:48.618999425 +0200
Creación: 2024-04-13 09:23:48.618999425 +0200

martin@debian12:/tmp/temporal$ cat fichero1.txt
Hola
martin@debian12:/tmp/temporal$ stat fichero1.txt
Fichero: fichero1.txt
Tamaño: 5          Bloques: 8          Bloque E/S: 4096   fichero regular
```

```

Device: 8,1      Inode: 1700631      Links: 1
Acceso: (0644/-rw-r--r--) Uid: ( 1000/ martin)  Gid: ( 1000/ martin)
      Acceso: 2024-04-13 10:12:22.169098495 +0200
Modificación: 2024-04-13 08:52:06.000000000 +0200
      Cambio: 2024-04-13 09:23:48.618999425 +0200
      Creación: 2024-04-13 09:23:48.618999425 +0200

martin@debian12:/tmp/temporal$ echo "Adios" >> fichero1.txt
martin@debian12:/tmp/temporal$ stat fichero1.txt
  Fichero: fichero1.txt
    Tamaño: 11          Bloques: 8          Bloque E/S: 4096   fichero regular
Device: 8,1      Inode: 1700631      Links: 1
Acceso: (0644/-rw-r--r--) Uid: ( 1000/ martin)  Gid: ( 1000/ martin)
      Acceso: 2024-04-13 10:12:22.169098495 +0200
Modificación: 2024-04-13 10:12:36.010174997 +0200
      Cambio: 2024-04-13 10:12:36.010174997 +0200
      Creación: 2024-04-13 09:23:48.618999425 +0200

martin@debian12:/tmp/temporal$ chmod 777 fichero1.txt
martin@debian12:/tmp/temporal$ stat fichero1.txt
  Fichero: fichero1.txt
    Tamaño: 11          Bloques: 8          Bloque E/S: 4096   fichero regular
Device: 8,1      Inode: 1700631      Links: 1
Acceso: (0777/-rwxrwxrwx) Uid: ( 1000/ martin)  Gid: ( 1000/ martin)
      Acceso: 2024-04-13 10:12:22.169098495 +0200
Modificación: 2024-04-13 10:12:36.010174997 +0200
      Cambio: 2024-04-13 10:12:59.494427849 +0200
      Creación: 2024-04-13 09:23:48.618999425 +0200

```

Para ver por separado con stat cada parametro tenemos las opciones:

- **x:** atime
- **y:** mtime
- **z:** ctime

```

martin@debian12:/tmp/temporal$ stat fichero1.txt
  Fichero: fichero1.txt
    Tamaño: 11          Bloques: 8          Bloque E/S: 4096   fichero regular
Device: 8,1      Inode: 1700631      Links: 1
Acceso: (0777/-rwxrwxrwx) Uid: ( 1000/ martin)  Gid: ( 1000/ martin)
      Acceso: 2024-04-13 10:12:22.169098495 +0200
Modificación: 2024-04-13 10:12:36.010174997 +0200
      Cambio: 2024-04-13 10:12:59.494427849 +0200
      Creación: 2024-04-13 09:23:48.618999425 +0200

martin@debian12:/tmp/temporal$ stat -c '%x' fichero1.txt
2024-04-13 10:12:22.169098495 +0200

martin@debian12:/tmp/temporal$ stat -c '%y' fichero1.txt
2024-04-13 10:12:36.010174997 +0200

```

```
martin@debian12:/tmp/temporal$ stat -c '%z' fichero1.txt
2024-04-13 10:12:59.494427849 +0200

martin@debian12:/tmp/temporal$ stat -c '%z, %n' fichero1.txt
2024-04-13 10:12:59.494427849 +0200, fichero1.txt
```

También se puede ordenar por tiempo con ls. `ls --time=ctime`

```
martin@debian12:/tmp/temporal$ ls --time=atime
martin@debian12:/tmp/temporal$ ls -u

martin@debian12:/tmp/temporal$ ls --time=mtime
martin@debian12:/tmp/temporal$ ls -t

martin@debian12:/tmp/temporal$ ls --time=ctime
martin@debian12:/tmp/temporal$ ls -c
```

`touch` en Linux es un comando que se utiliza para crear archivos vacíos o actualizar las marcas de tiempo de archivos existentes. Se usa principalmente para crear archivos nuevos o actualizar las fechas de acceso y modificación de archivos existentes sin cambiar su contenido.

```
martin@debian12:/tmp/temporal$ touch fichero.txt
martin@debian12:/tmp/temporal$ ls -l fichero.txt
-rw-r--r-- 1 martin martin 0 abr 13 11:01 fichero.txt
martin@debian12:/tmp/temporal$ stat fichero.txt
Fichero: fichero.txt
Tamaño: 0          Bloques: 0          Bloque E/S: 4096   fichero regular
vacío
Device: 8,1      Inode: 1700631      Links: 1
Acceso: (0644/-rw-r--r--) Uid: ( 1000/  martin)  Gid: ( 1000/  martin)
Acceso: 2024-04-13 11:01:20.991071163 +0200
Modificación: 2024-04-13 11:01:20.991071163 +0200
Cambio: 2024-04-13 11:01:20.991071163 +0200
Creación: 2024-04-13 11:01:20.987073163 +0200

martin@debian12:/tmp/temporal$ touch --date='2022-03-29 17:53:03' fichero.txt
martin@debian12:/tmp/temporal$ ls -l fichero.txt
-rw-r--r-- 1 martin martin 0 mar 29  2022 fichero.txt
martin@debian12:/tmp/temporal$ stat fichero.txt
Fichero: fichero.txt
Tamaño: 0          Bloques: 0          Bloque E/S: 4096   fichero regular
vacío
Device: 8,1      Inode: 1700631      Links: 1
Acceso: (0644/-rw-r--r--) Uid: ( 1000/  martin)  Gid: ( 1000/  martin)
Acceso: 2022-03-29 17:53:03.000000000 +0200
Modificación: 2022-03-29 17:53:03.000000000 +0200
Cambio: 2024-04-13 11:01:36.831147160 +0200
Creación: 2024-04-13 11:01:20.987073163 +0200
```

```
martin@debian12:/tmp/temporal$ touch -a --date='2023-04-29 17:53:03' fichero.txt
martin@debian12:/tmp/temporal$ stat fichero.txt
  Fichero: fichero.txt
Tamaño: 0          Bloques: 0          Bloque E/S: 4096   fichero regular
vacío
Device: 8,1      Inode: 1700631      Links: 1
Acceso: (0644/-rw-r--r--) Uid: ( 1000/ martin)  Gid: ( 1000/ martin)
  Acceso: 2023-04-29 17:53:03.000000000 +0200
Modificación: 2022-03-29 17:53:03.000000000 +0200
  Cambio: 2024-04-13 11:02:06.188461159 +0200
  Creación: 2024-04-13 11:01:20.987073163 +0200

martin@debian12:/tmp/temporal$ touch -m --date='2023-04-29 17:53:03' fichero.txt
martin@debian12:/tmp/temporal$ stat fichero.txt
  Fichero: fichero.txt
Tamaño: 0          Bloques: 0          Bloque E/S: 4096   fichero regular
vacío
Device: 8,1      Inode: 1700631      Links: 1
Acceso: (0644/-rw-r--r--) Uid: ( 1000/ martin)  Gid: ( 1000/ martin)
  Acceso: 2023-04-29 17:53:03.000000000 +0200
Modificación: 2023-04-29 17:53:03.000000000 +0200
  Cambio: 2024-04-13 11:02:33.170963160 +0200
  Creación: 2024-04-13 11:01:20.987073163 +0200
```

Gestion de usuarios

Archivos de configuración

Los archivos de configuración importantes en sistemas Linux referentes a la gestión de usuarios y grupos son:

- **/etc/passwd**: Almacena información sobre usuarios del sistema, como nombres de usuario, IDs de usuario, directorios de inicio y shell predeterminada.

```
martin:x:1000:1000:martin,23,666777888,981234567:/home/martin:/bin/bash``
```

1. **martin**: Usuario.
2. **x**: Usado antiguamente para colocar la clave. En la actualidad no se coloca la clave en este campo (se pone siempre una x) y la clave se coloca en /etc/shadow.
3. **1000**: UID.
4. **1000**: GID.
5. **martin,23,666777888,981234567**: Datos del usuario.
6. **/home/martin**: Directorio de trabajo.
7. **/bin/bash**: Shell.

- **/etc/shadow**: Contiene contraseñas encriptadas de los usuarios y datos de seguridad relacionados, como políticas de contraseñas y fechas de caducidad.

```
martin:$6$QeYgJjn3$WUebH8Ku2cb3Z0UVpqA3c.zrCzH38hJrM7m3cE8HQitQBr6FF/4Ussq/gnGzBom
MBmCxgjyQamLE8V3GHe.rn1:18792:0:99999:7:::
```

- **/etc/group**: Guarda información sobre los grupos del sistema, incluyendo nombres de grupo y listas de usuarios asociados a cada grupo.

```
dam:x:1001:martin,lucas
```

1. **dam**: Nombre del grupo.
 2. **x**: Contraseña del grupo (generalmente se coloca una "x" para indicar que está encriptada en **/etc/gshadow**).
 3. **1001**: GID (identificador de grupo).
 4. **martin,lucas**: Lista de usuarios secundarios separados por comas.
- **/etc/gshadow**: Similar a **/etc/shadow**, pero para grupos, almacena contraseñas encriptadas para grupos y datos de seguridad relacionados.

```
dam:RsdRTGHtdrs:moncho:lipido,rivelora
```

1. **dam**: Nombre del grupo.
2. **RsdRTGHtdrs**: Contraseña del grupo, así un usuario podrá ingresar al grupo si conoce la contraseña.
3. **moncho**: Administrador del grupo.
4. **lipido,rivelora**: Usuarios que no se quiere que se conozca su pertenencia al grupo.

su y sudo

```
martin@debian12:/etc/sudoers.d$ sudo su -
martin@debian12:/etc/sudoers.d$ sudo su
martin@debian12:/etc/sudoers.d$ sudo -i
martin@debian12:/etc/sudoers.d$ su juan
martin@debian12:/etc/sudoers.d$ su - juan
martin@debian12:/etc/sudoers.d$ su - juan -c "pwd"
martin@debian12:/etc/sudoers.d$ su -l juan
martin@debian12:/etc/sudoers.d$ su --c "pwd"
```

1. **sudo su -**: Inicia un shell interactivo como el usuario root, cargando el entorno de inicio de sesión del usuario root.
2. **sudo su**: Inicia un shell interactivo como el usuario root, heredando el entorno del usuario actual.
3. **sudo -i**: Inicia un nuevo shell interactivo como el usuario root, cargando completamente el entorno de inicio de sesión del usuario root.
4. **su juan**: Cambia al usuario "juan" iniciando un nuevo shell pero sin cargar su entorno de inicio de sesión.

5. `su - juan`: Cambia al usuario "juan" iniciando un nuevo shell y cargando su entorno de inicio de sesión.
6. `su - juan -c "pwd"`: Cambia al usuario "juan", ejecuta el comando "pwd" y luego regresa al usuario actual.
7. `su -l juan`: Cambia al usuario "juan" iniciando un nuevo shell y cargando su entorno de inicio de sesión (equivalente a `su - juan`).
8. `su --c "pwd"`: Intenta ejecutar el comando "pwd" en el shell actual, pero probablemente generaría un error ya que la opción "--c" no es válida para el comando `su`.

visudo y sudoers

`visudo` es el editor que permite modificar el archivo `sudoers`.

`sudoers` es el fichero donde se especifican las reglas para poder establecer quienes pueden ejecutar que comandos en nombre de sudo y en que hosts.

```
# Alias
Cmd_Alias RECARGAR_SSH = systemctl restart sshd.service
Host_Alias PC_ALEX = 192.168.1.22

# User privilege specification
root    ALL=(ALL:ALL) ALL

# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL

martin  192.168.1.14,192.168.1.15=(juan:dam) NOPASSWD: /bin/pwd

%asir 192.168.1.14,192.168.1.15=(juan:dam) /bin/pwd

martin  PC_ALEX=(alex) NOPASSWD: RECARGAR_SSH
```

Al final estas entradas especifican para el usuario martin (analogo para el grupo asir) lo siguiente:

- **martin**: El usuario al que se aplica la regla. En este caso, el usuario "martin" puede ejecutar el comando especificado.
- **192.168.1.14,192.168.1.15**: Las direcciones IP de los Host de la red donde el usuario "martin" puede ejecutar el comando.
- **(juan:dam)**: El usuario y el grupo sobre los que martin puede ejecutar comandos en su nombre.
- ****NOPASSWD:****: Indica que se puede ejecutar el comando sin que se pregunte la contraseña.
- **/bin/pwd**: El comando permitido.

En el archivo `sudoers`, cuando especificas una entrada como esta:

```
martin 192.168.1.14,192.168.1.15=(juan:dam) /bin/pwd
```

Significa que el usuario "martin" puede ejecutar el comando `/bin/pwd` en el host **192.168.1.14** en nombre de juan o del grupo dam. Esto no significa que el usuario "martin" pueda ejecutar el comando `pwd` en su propio sistema local desde cualquier host.

id, groups, passwd

id: Permite ver uid, gid y grupos secundarios a los que pertenece el usuario.

groups: Permite ver unicamente los grupos secundarios del usuario.

```
martin@debian12:~$ id
uid=1000(martin) gid=1000(martin)
grupos=1000(martin),24(cdrom),25(floppy),27(sudo),29(audio),30(dip),44(video),46(p
lugdev),100(users),106(netdev),113(bluetooth),116(lpadmin),119(scanner)
martin@debian12:~$ groups
martin cdrom floppy sudo audio dip video plugdev users netdev bluetooth lpadmin
scanner

martin@debian12:~$ sudo su -
root@debian12:~# id
uid=0(root) gid=0(root) grupos=0(root)
root@debian12:~# groups
root
```

passwd: Permite modificar la contraseña. Los parametros destacables son:

- l: Bloquea el acceso al sistema al usuario (usermod -L).
- u: Desbloquea el acceso al sistema del usuario (usermod -U).

```
martin@debian12:~$ sudo passwd -l martin && sudo tail /etc/shadow | grep martin
passwd: contraseña cambiada.
martin:!!$y$j9T$D1YstIGHwPXktsEmolZg./$I7fKcY0m9yE2LYgGBEn8yo1Exy5PLvBTI1Zf5keudM3:
19770:0:99999:7:::
martin@debian12:~$ sudo passwd -u martin && sudo tail /etc/shadow | grep martin
passwd: contraseña cambiada.
martin:$y$j9T$D1YstIGHwPXktsEmolZg./$I7fKcY0m9yE2LYgGBEn8yo1Exy5PLvBTI1Zf5keudM3:1
9770:0:99999:7:::
```

useradd, usermod, userdel, groupadd, groupdel

useradd: Permite añadir nuevos usuarios al sistema.

```
useradd -m -d /home/juan -p "$(mkpasswd 'abc123..')" -g sistemas -G dam -s
/bin/bash juan
```

Nota: Si queremos que un usuario tenga un grupo principal con el mismo nombre, no hay que indicarlo con la opción -g, es automático

```
si@si-VirtualBox:~/Desktop/scripts/ejercicios/ej2$ sudo useradd -m -d /home/alumno  
-p $(mkpasswd 'abc123.') -s "/bin/bash" alumno
```

```
si@si-VirtualBox:~/Desktop/scripts/ejercicios/ej2$ tail -1 /etc/passwd  
geoclue:x:124:131:./var/lib/geoclue:/usr/sbin/nologin  
pulse:x:125:132:PulseAudio daemon,,:/run/pulse:/usr/sbin/nologin  
gnome-initial-setup:x:126:65534:./run/gnome-initial-setup:/bin/false  
hplip:x:127:7:HPLIP system user,,:/run/hplip:/bin/false  
gdm:x:128:134:Gnome Display Manager:/var/lib/gdm3:/bin/false  
si:x:1000:1000:si,,:/home/si:/bin/bash  
vboxadd:x:999:1:./var/run/vboxadd:/bin/false  
sshd:x:129:65534:./run/sshd:/usr/sbin/nologin  
mysql:x:130:137:MySQL Server,,:/nonexistent:/bin/false  
alumno:x:1001:1001:./home/alumno:/bin/bash
```

```
si@si-VirtualBox:~/Desktop/scripts/ejercicios/ej2$ tail -1 /etc/group  
pulse:x:132:  
pulse-access:x:133:  
gdm:x:134:  
lxd:x:135:si  
si:x:1000:  
sambashare:x:136:si  
vboxsf:x:999:  
vboxdrmpic:x:998:  
mysql:x:137:  
alumno:x:1001:
```

usermod: Permite modificar las propiedades de un usuario existente en el sistema.

```
usermod -d /home/juan_nuevo -s /bin/zsh juan
```

userdel: Permite eliminar un usuario del sistema.

```
userdel -r juan
```

- *Nota Este comando eliminaría el usuario "juan" del sistema, junto con su directorio de inicio (-r), así como cualquier archivo o directorio relacionado con el usuario.*

groupadd: Permite crear un nuevo grupo en el sistema.

```
groupadd dam
```

groupdel: Permite eliminar un grupo del sistema.

```
groupdel dam
```

chfn, chsh

chfn: Permite editar los datos personales del usuario. **chsh:** Permite editar la shell del usuario.

Permisos

A continuación se van a explicar los permisos que pueden existir en un fichero o directorio y como editarlos. Supongamos que tienes un archivo llamado "documento.txt".

1. Permisos ugo para fichero:

- **r (read):** Permite leer el contenido del archivo.
- **w (write):** Permite modificar el contenido del archivo.
- **x (execute):** Permite ejecutar el archivo como un programa o script. El propietario o los usuarios autorizados pueden ejecutar el archivo si es ejecutable.

Permisos ugo para fichero:

- **r (read):** Permite ver el listado de archivos que contiene el directorio.
- **w (write):** Permite modificar el contenido del directorio.
- **x (execute):** Permite acceder al contenido del directorio.

2. Tipos de permisos:

- **r:** Lectura
- **w:** Escritura
- **x:** Ejecución

Ejemplo 1:

El archivo "documento.txt" tiene permisos de lectura y escritura para el propietario, permisos de lectura para el grupo y permisos de lectura para otros usuarios.

Representación con números:

- Propietario: Lectura y escritura (4 + 2 = 6)
- Grupo: Lectura (4)
- Otros: Lectura (4)

Por lo tanto, el comando para establecer estos permisos sería:

```
chmod 644 documento.txt
```

Representación con letras:

- Propietario: Lectura y escritura (rw)
- Grupo: Lectura (r)
- Otros: Lectura (r)

El comando sería el mismo:

```
chmod u=rw,g=r,o=r documento.txt
```

Ejemplo 2:

El archivo "documento.txt" tiene permisos de lectura, escritura y ejecución para el propietario, permisos de lectura y ejecución para el grupo, y permisos de ejecución para otros usuarios.

Representación con números:

- Propietario: Lectura, escritura y ejecución ($4 + 2 + 1 = 7$)
- Grupo: Lectura y ejecución ($4 + 1 = 5$)
- Otros: Ejecución (1)

El comando sería:

```
chmod 755 documento.txt
```

Representación con letras:

- Propietario: Lectura, escritura y ejecución (rwx)
- Grupo: Lectura y ejecución (r-x)
- Otros: Ejecución (--x)

El comando equivalente sería:

```
chmod u=rwx,g=rx,o=x documento.txt
```

Ejemplo 3:

```
chmod u-r,g+rw,o=w documento.txt
```

Ejemplo 4:

```
chmod u=rwx,g=rw,o=w documento.txt
```

Ejemplo 5:

```
chmod u+wx,g-rw,a=w documento.txt
```

chattr y lsattr

A mayores existen en Linux a editar con los comandos **chattr** y listar con **lsattr** otros permisos. **chattr** es un comando en sistemas Unix y Linux que se utiliza para cambiar los **atributos** de un archivo en el sistema de archivos. Estos atributos pueden controlar varios aspectos del archivo, como su capacidad de modificación, eliminación o incluso si puede ser movido o renombrado. Uno de los atributos más comunes es el atributo de solo lectura.

- **a** (append only) permite solo añadir datos, útil para registros.
- **i** (immutable) hace el archivo inmutable, impidiendo modificaciones, borrados o renombrados, común en archivos críticos del sistema.

Como ejemplos tenemos:

```
si@si-VirtualBox:/tmp/prueba$ ls -l fichero1.txt
-rw-rw-r-- 1 si si 9 abr 13 22:42 fichero1.txt

si@si-VirtualBox:/tmp/prueba$ lsattr fichero1.txt
-----e----- fichero1.txt

si@si-VirtualBox:/tmp/prueba$ sudo chattr +a fichero1.txt
[sudo] password for si:

si@si-VirtualBox:/tmp/prueba$ lsattr fichero1.txt
-----a-----e----- fichero1.txt

si@si-VirtualBox:/tmp/prueba$ echo "Otra linea" > fichero1.txt
-bash: fichero1.txt: Operation not permitted

si@si-VirtualBox:/tmp/prueba$ echo "Otra linea" >> fichero1.txt

si@si-VirtualBox:/tmp/prueba$ cat fichero1.txt
Fichero1
Otra linea

si@si-VirtualBox:/tmp/prueba$ sudo chattr +i fichero1.txt

si@si-VirtualBox:/tmp/prueba$ echo "Otra nueva linea" >> fichero1.txt
-bash: fichero1.txt: Operation not permitted

si@si-VirtualBox:/tmp/prueba$ sudo echo "Otra nueva linea" >> fichero1.txt
-bash: fichero1.txt: Operation not permitted

si@si-VirtualBox:/tmp/prueba$ lsattr fichero1.txt
-----ia-----e----- fichero1.txt
```

```
si@si-VirtualBox:/tmp/prueba$ sudo chatter -ai fichero1.txt

si@si-VirtualBox:/tmp/prueba$ lsattr fichero1.txt
-----e----- fichero1.txt
```

ACLs

En caso de conflicto entre el usuario propietario/otros (uo de ugo) y las ACLs, prevalecen los permisos uo de ugo.

- **Atributos** → **uo de ugo** → **ACLs**:

En caso de conflicto entre el grupo propietario (g de ugo)/otros grupos distintos del propietario/otros usuarios distintos del propietario y las ACLs, prevalecen las ACLs.

- **Atributos** → **mascara/ACLs** → **g de ugo**:

Asignar ACLs a usuario

```
setfacl -R -m u:mateo:rwX prueba/
```

```
root@si-VirtualBox:/mnt# setfacl -R -m u:mateo:rwX prueba/
```

```
root@si-VirtualBox:/mnt# getfacl -R prueba/
```

```
# file: prueba/
```

```
# owner: root
```

```
# group: root
```

```
user::rwX
```

```
user:mateo:rwX
```

```
group::r--
```

```
mask::rwX
```

```
other::---
```

```
# file: prueba//fichero1.txt
```

```
# owner: root
```

```
# group: root
```

```
user::rw-
```

```
user:mateo:rwX
```

```
group::r--
```

```
mask::rwX
```

```
other::r--
```

Asignar ACLs a grupo

```
setfacl -R -m g:dam:r-x prueba/
```

```
root@si-VirtualBox:/mnt# setfacl -R -m g:dam:r-x prueba/
```

```
root@si-VirtualBox:/mnt# getfacl -R prueba/
```

```
# file: prueba/
```

```
# owner: root
```

```
# group: root
```

```
user::rwx
```

```
group::r--
```

```
group:dam:r-x
```

```
mask::r-x
```

```
other::---
```

```
# file: prueba//fichero1.txt
```

```
# owner: root
```

```
# group: root
```

```
user::rw-
```

```
group::r--
```

```
group:dam:r-x
```

```
mask::r-x
```

```
other::r--
```

Asignación de ACP por defecto

```
setfacl -R -d -m g:dam:w prueba/
```

```
root@si-VirtualBox:/mnt# setfacl -R -d -m g:dam:w prueba/
```

```
root@si-VirtualBox:/mnt# getfacl -R prueba/
```

```
# file: prueba/
```

```
# owner: root
```

```
# group: root
```

```
user::rwx
```

```
group::r--
```

```
other::---
```

```
default:user::rwx
```

```
default:group::r--
```

```
default:group:dam:-w-
```

```
default:mask::rw-
```

```
default:other::---
```

```
# file: prueba//fichero1.txt
```

```
# owner: root
```

```
# group: root
```

```
user::rw-
```



```
group::r--
other::r--
```

Máscara en ACLs

```
setfacl -m m::rwx prueba/
```

```
root@si-VirtualBox:/mnt# setfacl -m m::rwx prueba/

root@si-VirtualBox:/mnt# getfacl -R prueba/
# file: prueba/
# owner: root
# group: root
user::rwx
group::r--
mask::rwx
other::---

# file: prueba//fichero1.txt
# owner: root
# group: root
user::rw-
group::r--
other::r--
```

Otros en ACLs

```
setfacl -R -m u::x prueba/
setfacl -R -m g::r prueba/
setfacl -R -m o::w prueba/
```

```
root@si-VirtualBox:/mnt# setfacl -R -m u::x prueba/
root@si-VirtualBox:/mnt# setfacl -R -m g::r prueba/
root@si-VirtualBox:/mnt# setfacl -R -m o::w prueba/
root@si-VirtualBox:/mnt# getfacl -R prueba/
# file: prueba/
# owner: root
# group: root
user::--x
group::r--
mask::r--
other::-w-

# file: prueba//fichero1.txt
```

```
# owner: root
# group: root
user::--x
group::r--
other::-w-
```

Eliminar ACLs

```
setfacl -R -b -k datosEmpresa/
```

```
root@si-VirtualBox:/mnt# setfacl -R -b -k prueba/

root@si-VirtualBox:/mnt# getfacl -R prueba/
# file: prueba/
# owner: root
# group: root
user::rwx
group::r--
other:---

# file: prueba//fichero1.txt
# owner: root
# group: root
user::rw-
group::r--
other::r--
```

*Notas: Es importante tener en cuenta que las ACL añaden tal cual los permisos que se especifican, es decir **g:dam:r-x** asigna solo permisos a dam de lectura y ejecución, igual que **g:dam:rx**. En caso de que existiera el permiso de escritura, este habría desaparecido.*

*Notas 2: Cuando añadimos ACLs a un fichero o directorio aparece un signo **+** al final de los permisos UGO.*

```
-rw-rwx---+ 1 root root 8 abr 13 20:29 fichero1.txt
```

*Notas 3: Es incorrecto hacer **setfacl -m u:rw prueba/** o **setfacl -m g:r-w prueba/** pero si es correcto hacer **setfacl -m o:rw prueba/**.*

*Nota 4: Se pueden juntar los parametros de una ACL para realzar algo como lo siguiente **setfacl -Rbk prueba/**.*

Repositorios

Por supuesto:

apt:

- **-get**: Obtiene los paquetes de los repositorios pero no los instala. Actualmente ya no es necesario.
- **-install**: Instala uno o varios paquetes.
- **-remove**: Elimina uno o varios paquetes.
- **-purge**: Elimina uno o varios paquetes y también sus archivos de configuración.
- **-update**: Actualiza la lista de paquetes disponibles en los repositorios.
- **-upgrade**: Actualiza todos los paquetes instalados a las versiones más recientes disponibles.

dpkg:

- **-i**: Instala un paquete.
- **-r**: Elimina un paquete pero conserva sus archivos de configuración.
- **-P**: Elimina un paquete y también sus archivos de configuración.
- **-l**: Lista todos los paquetes instalados en el sistema.
- **-L**: Lista todos los archivos instalados por un paquete específico.

```

root@debian12:~/Descargas# ls
debian-refcard_12.0_all.deb

root@debian12:~/Descargas# dpkg -i debian-refcard_12.0_all.deb
Seleccionando el paquete debian-refcard previamente no seleccionado.
(Leyendo la base de datos ... 260028 ficheros o directorios instalados
actualmente.)
Preparando para desempaquetar debian-refcard_12.0_all.deb ...
Desempaquetando debian-refcard (12.0) ...
Configurando debian-refcard (12.0) ...

root@debian12:~/Descargas# dpkg -l debian-refcard
Deseado=desconocido(U)/Instalar/eliminar/Purgar/retener(H)
| Estado=No/Inst/ficheros-Conf/desempaquetado/medio-conf/medio-inst(H)/espera-
disparo(W)/pendiente-disparo
|/ Err?=(ninguno)/requiere-Reinst (Estado,Err: mayúsc.=malo)
||/ Nombre          Versión          Arquitectura Descripción
+++=====
=====
ii  debian-refcard 12.0          all          printable reference card for the
Debian system

root@debian12:~/Descargas# dpkg -L debian-refcard
/.
/usr
/usr/share
/usr/share/doc
/usr/share/doc/debian-refcard
/usr/share/doc/debian-refcard/changelog.gz
/usr/share/doc/debian-refcard/copyright
/usr/share/doc/debian-refcard/index.html
/usr/share/doc/debian-refcard/openlogo-nd-25.png
/usr/share/doc/debian-refcard/refcard-bg-a4.pdf.gz
/usr/share/doc/debian-refcard/refcard-ca-a4.pdf.gz

```

```
root@debian12:~/Descargas# dpkg -P debian-refcard
(Leyendo la base de datos ... 260072 ficheros o directorios instalados
actualmente.)
Desinstalando debian-refcard (12.0) ...

root@debian12:~/Descargas# dpkg -l debian-refcard
dpkg-query: no se ha encontrado ningún paquete que corresponda con debian-refcard.
```

ip e ifconfig

Aunque **ip** como comando ha terminado por desbancar al popular **ifconfig** para las configuraciones a nivel de red en linux, se presenta a continuación el conjunto de comandos **ifconfig** con su equivalencia a comando **ip**.

Puedo ayudarte con eso. Aquí tienes los comandos organizados en un archivo Markdown:

ifconfig

```
$ ifconfig          # Listar interfaces activas. Normalmente como usuario sin
permisos de root (sudo) da error.
$ ifconfig -a       # Listar interfaces estén o no activas
$ /sbin/ifconfig    # Listar interfaces activas
$ /sbin/ifconfig -a # Listar interfaces estén o no activas
$ ifconfig eth0     # Listar la configuración de la interfaz eth0
$ ifconfig eth0 up   # Activar interfaz eth0
$ ifconfig eth0 down # Deshabilitar interfaz eth0
$ ifconfig eth0 192.168.100.100          # Configuración de red para la
interfaz eth0: IP=192.168.100.100, MS=255.255.255.0
$ ifconfig eth0 192.168.100.100/24      # Configuración de red para la
interfaz eth0: IP=192.168.100.100, MS=255.255.255.0
$ ifconfig eth0 192.168.100.100 netmask 255.255.255.0 # Equivale al comando
anterior.
$ ifconfig eth0:0 192.168.100.101/24     # Generar el alias eth0:0 para la
interfaz eth0 con otra configuración de red: IP=192.168.100.101, MS=255.255.255.0.
$ ifconfig eth0:0 192.168.100.101 netmask 255.255.255.0 # Equivale al comando
anterior.
$ ifconfig eth0:web 192.168.100.102 netmask 255.255.255.0 # Nuevo alias eth0:web
para la interfaz eth0.
```

ip

ip address

```
$ ip addr          # Listar interfaces activas
$ ip -c addr       # Listar interfaces activas y aporta ayuda en color
$ ip addr show     # Equivale al comando anterior.
```

```
$ ip addr show eth0 # Listar la configuración de la interfaz eth0
$ ip link           # Muestra únicamente información a nivel de MAC.
$ ip link show eth0 # Muestra únicamente información a nivel de MAC para una
tarjeta concreta.
$ ip link set eth0 up    # Activar interfaz eth0
$ ip link set eth0 down  # Deshabilitar interfaz eth0
$ ip address add 192.168.100.100 dev eth0 # Configuración de red para la
interfaz eth0: IP=192.168.100.100, MS=255.255.255.255. NOTA: Si eth0 con la IP
192.168.100.100 ya había sido configurada con ifconfig, la nueva configuración
realizada mediante ip no se vería en la ejecución de ifconfig.
$ ip address add 192.168.100.100/24 broadcast 192.168.100.255 dev eth0 #
Configuración de red para la interfaz eth0: IP=192.168.100.100, MS=255.255.255.0
$ ip address add 192.168.100.104/24 dev eth0 label eth0:4 # Generar el alias
eth0:4 para la interfaz eth0 con otra configuración de red: IP=192.168.100.104,
MS=255.255.255.0.
$ ip address del 192.168.100.101/24 dev eth0 # Eliminar esa configuración IP en
la interfaz eth0. En este caso esa configuración corresponde con el alias eth0:0,
por lo que elimina ese alias de la configuración de red.
$ ip address del 192.168.100.102/24 dev eth0:web # Eliminar el alias eth0:web
```

ip route

```
$ route           # Listar tabla de enrutamiento.
$ ip route        # Listar tabla de enrutamiento
$ ip route show   # Equivale al comando anterior.
$ ip route list   # Equivale al comando anterior.
$ route -n        # Listar tabla de enrutamiento sin resolución DNS

$ route add default gw 192.168.100.1 # Configurar puerta de enlace
(gateway).
$ ip route add default via 192.168.100.1 # Equivale al comando anterior.

$ route del default gw 192.168.100.1 # Eliminar puerta de enlace (gateway).
$ ip route del default via 192.168.100.1 # Equivale al comando anterior.

$ route add -net 192.168.200.0 netmask 255.255.255.0 dev eth0 # Añadir regla de
enrutamiento para la red 192.168.200.0 en la interfaz eth0
$ ip route add 192.168.200.0/24 dev eth0 # Equivale al comando anterior.

$ route del -net 192.168.200.0 netmask 255.255.255.0 dev eth0 # Eliminar regla de
enrutamiento para la red 192.168.200.0 en la interfaz eth0
$ ip route delete 192.168.200.0/24 dev eth0 # Equivale al comando anterior.

$ route add -net 192.168.100.0 netmask 255.255.255.0 gw 192.168.100.1 dev eth0 #
Añadir regla de enrutamiento para la red 192.168.100.0 en la interfaz eth0 con la
puerta de enlace 192.168.100.1
$ ip route add 192.168.100.0/24 via 192.168.100.1 dev eth0 onlink # Equivale al
comando anterior.
```

- Nota: Es conveniente saber que tanto *ip a*, *ip addr* como *ip address* son equivalentes.

wget y curl

wget: Es una herramienta de línea de comandos que permite la descarga de archivos desde servidores remotos a través de HTTP, HTTPS y FTP. Es útil para descargar archivos de Internet de forma sencilla y automatizada.

- **-O**: Especifica el nombre del archivo de salida.
- **-q**: Ejecución silenciosa, sin mensajes de progreso.
- **-P**: Especifica el directorio de destino para guardar el archivo.
- **-c**: Continuar descargas interrumpidas.
- **-r**: Descarga recursiva, sigue enlaces dentro de la página.

```
si@si-VirtualBox:/tmp/pr$ wget
https://d.winrar.es/d/97z1713015469/hrYvljKNPEqbS2FjdvSpsQ/rarlinux-x64-700.tar.gz
--2024-04-13 15:36:35--
https://d.winrar.es/d/97z1713015469/hrYvljKNPEqbS2FjdvSpsQ/rarlinux-x64-700.tar.gz
Resolving d.winrar.es (d.winrar.es)... 82.98.170.158
Connecting to d.winrar.es (d.winrar.es)|82.98.170.158|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 730268 (713K) [application/octet-stream]
Saving to: 'rarlinux-x64-700.tar.gz'

rarlinux-x64-700.tar. 100%[=====>] 713,15K   251KB/s   in 2,8s

2024-04-13 15:36:38 (251 KB/s) - 'rarlinux-x64-700.tar.gz' saved [730268/730268]

si@si-VirtualBox:/tmp/pr$ ls
rarlinux-x64-700.tar.gz
```

```
si@si-VirtualBox:/tmp/pr$ wget -q
https://d.winrar.es/d/97z1713015469/hrYvljKNPEqbS2FjdvSpsQ/rarlinux-x64-700.tar.gz

si@si-VirtualBox:/tmp/pr$ ls
rarlinux-x64-700.tar.gz
```

```
si@si-VirtualBox:/tmp/prueba$ wget -qO /tmp/rar.tar.gz
https://d.winrar.es/d/97z1713015469/hrYvljKNPEqbS2FjdvSpsQ/rarlinux-x64-700.tar.gz
si@si-VirtualBox:/tmp/prueba$ ls /tmp/
rar.tar.gz
```

curl: Es una herramienta de línea de comandos para transferir datos desde o hacia un servidor, utilizando uno de los protocolos compatibles, como HTTP, HTTPS, FTP, etc. Es muy versátil y admite una amplia gama de funciones y protocolos.

- **-o**: Especifica el nombre del archivo de salida.

- **-s**: Ejecución silenciosa, sin mensajes de progreso.
- **-O**: Descargar y guardar usando el nombre del archivo remoto.
- **-C**: Continuar descargas interrumpidas.
- **-L**: Seguir redirecciones.

```
si@si-VirtualBox:/tmp/pr$ curl -O
https://d.winrar.es/d/97z1713015469/hrYvljKNPEqbS2FjdvSpsQ/rarlinux-x64-700.tar.gz
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
100  169  100  169    0     0  1264      0 --:--:-- --:--:-- --:--:--  1261
si@si-VirtualBox:/tmp/pr$ curl -sO
https://d.winrar.es/d/97z1713015469/hrYvljKNPEqbS2FjdvSpsQ/rarlinux-x64-700.tar.gz
si@si-VirtualBox:/tmp/pr$ ls
rarlinux-x64-700.tar.gz
```

```
si@si-VirtualBox:/tmp/prueba$ curl -so /tmp/rar.tar.gz
https://d.winrar.es/d/97z1713015469/hrYvljKNPEqbS2FjdvSpsQ/rarlinux-x64-700.tar.gz
si@si-VirtualBox:/tmp/prueba$ ls /tmp/
rar.tar.gz
```

ss y netstat

Tanto **ss** como **netstat** son comandos utilizados en sistemas Unix y Linux para mostrar información sobre conexiones de red, enrutamiento y estadísticas de red. Opciones comunes para ambas herramientas:

- **-l**: Muestra sólo las conexiones que están escuchando (es decir, los sockets en estado de escucha).
- **-a**: Muestra todas las conexiones, tanto las que están escuchando como las establecidas.
- **-t**: Muestra sólo las conexiones TCP.
- **-u**: Muestra sólo las conexiones UDP.
- **-n**: Muestra las direcciones IP y los números de puerto en formato numérico (sin resolución de nombres).

ssh y scp

SSH (Secure Shell) es un protocolo de red seguro que permite la comunicación segura y el acceso remoto a sistemas informáticos a través de una conexión cifrada. Se utiliza comúnmente para administrar sistemas remotos de forma segura mediante una interfaz de línea de comandos.

```
ssh usuario@192.168.1.100
```

```
ssh -p 2222 si@localhost
```

```
PS C:\Users\gilbl> ssh -p 1010 si@localhost
PS C:\Users\gilbl> ssh -p 1010 -l si localhost
si@localhost's password:
Welcome to Ubuntu 22.04.4 LTS (GNU/Linux 6.5.0-27-generic x86_64)

si@si-VirtualBox:~$
```

También puede ejecutar un único comando sin necesidad de tener una conexión abierta hasta que se decida finalizarla.

```
PS C:\Users\gilbl> ssh -p 1010 si@localhost ls
si@localhost's password:
Desktop
Documents
Downloads
Music
Pictures
Public
snap
Templates
Videos
PS C:\Users\gilbl>

PS C:\Users\gilbl> ssh -p 1010 si@localhost "ls && ls -l"
si@localhost's password:
Desktop
Documents
Downloads
Music
Pictures
Public
snap
Templates
Videos
total 36
drwxr-xr-x 2 si si 4096 abr 14 17:38 Desktop
drwxr-xr-x 2 si si 4096 ene  3 17:18 Documents
drwxr-xr-x 2 si si 4096 mar 11 22:32 Downloads
drwxr-xr-x 2 si si 4096 ene  3 17:18 Music
drwxr-xr-x 2 si si 4096 ene  3 17:18 Pictures
drwxr-xr-x 2 si si 4096 ene  3 17:18 Public
drwx----- 6 si si 4096 abr  9 13:02 snap
drwxr-xr-x 2 si si 4096 ene  3 17:18 Templates
drwxr-xr-x 2 si si 4096 ene  3 17:18 Videos
```

También se podría ejecutar un script.


```
PS C:\Users\gilbl> ssh -p 1010 si@localhost bash
/home/si/Desktop/scripts/apagar.sh
```

SCP (Secure Copy) es una herramienta de línea de comandos basada en SSH que se utiliza para transferir archivos de forma segura entre sistemas locales y remotos. Utiliza SSH para cifrar la transferencia de datos, proporcionando una forma segura de copiar archivos de un lugar a otro a través de una red.

Se puede transferir a mi equipo.

```
scp usuario@192.168.1.100:/home/usuario/Desktop/bd.sql .
```

```
scp -P 2222 -r si@localhost:~/Desktop .
```

O transferir a otro equipo diferente.

```
scp archivo.txt si@localhost:~/Desktop
```

```
si@si-VirtualBox:~/Desktop$ scp si@192.168.2.19:~/Desktop/bd.sql .
si@192.168.2.19's password:
bd.sql                                100% 5232      2.0MB/s   00:00
si@si-VirtualBox:~/Desktop$ ls
bd.sql

si@si-VirtualBox:~/Desktop$ scp bd.sql si@192.168.2.19:~/Desktop
si@192.168.2.19's password:
bd.sql                                100% 5232      1.6MB/s   00:00
```

En caso de no indicar ruta a continuación de la ip, se adjuntarán los archivos en la ruta \$HOME del usuario.

```
si@si-VirtualBox:~/Desktop$ mkdir prueba && for i in $(seq 1 100); do touch
prueba/fichero${i}.txt; done

si@si-VirtualBox:~/Desktop$ scp -r prueba/ si@192.168.2.19:
si@192.168.2.19's password:
fichero1.txt                          100%    0      0.0KB/s   00:00
fichero71.txt                         100%    0      0.0KB/s   00:00
fichero9.txt                          100%    0      0.0KB/s   00:00
fichero17.txt                         100%    0      0.0KB/s   00:00
...
```

```
si@si-VirtualBox:~$ ls
Desktop    Downloads  Pictures   Public    Templates
Documents  Music      prueba    snap      Videos
```

También hay que mencionar que podemos tener escenarios donde desde una máquina **A** hagamos una transferencia de archivos desde **B** hacia **C**.

```
si@si-VirtualBox:~/Desktop$ scp -r usuarioB@192.168.2.19:~/Desktop/prueba
usuarioC@192.168.2.20:~/Desktop
```

El archivo **known_hosts** en el directorio **/root/.ssh/** contiene las claves públicas de los hosts a los que te has conectado anteriormente mediante SSH desde la cuenta de usuario **root** en ese sistema. Cada línea en este archivo representa la información de un host específico, incluyendo su dirección IP, nombre de host y su clave pública.

Aquí hay un ejemplo simplificado de cómo se ve el contenido del archivo **known_hosts**:

```
192.168.2.19 ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQACXyOePwBsMn5t...
192.168.2.20 ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQACXyOePwBsMn5t...
```

systemd, systemctl y /etc/init.d

1. Utilizando **systemctl**:

```
systemctl status ssh.service
```

2. Utilizando **service**:

```
service ssh status
```

3. Utilizando **/etc/init.d**:

```
/etc/init.d/ssh status
```

Niveles de arranque en Linux

Los runlevels (niveles de ejecución) son una característica del sistema **init** de Unix y Linux que determinan qué servicios y procesos se ejecutan en un momento dado. En sistemas modernos basados en **systemd**, que es el

caso de muchas distribuciones recientes, los runlevels tradicionales han sido reemplazados por "targets". Sin embargo, la idea general sigue siendo la misma: diferentes niveles de ejecución representan diferentes configuraciones del sistema.

A continuación se adjunta una descripción general de los runlevels tradicionales en sistemas que utilizan System V init:

- **Runlevel 0:** Apagado del sistema.
- **Runlevel 1:** Modo de rescate o de un solo usuario. Solo se ejecuta un conjunto mínimo de servicios y no se inicia el entorno gráfico.
- **Runlevel 2:** Multiusuario sin red. Similar al nivel de ejecución 3, pero sin servicios de red.
- **Runlevel 3:** Multiusuario con red. El sistema arranca en modo multiusuario y se inician todos los servicios necesarios para permitir conexiones de red.
- **Runlevel 4:** Reservado para un uso personalizado. Por lo general, no se utiliza en las distribuciones estándar de Linux.
- **Runlevel 5:** Multiusuario con interfaz gráfica (GUI). Similar al nivel de ejecución 3, pero también inicia el entorno gráfico.
- **Runlevel 6:** Reinicio del sistema.

sysVinit

En sysVinit se invocan niveles de ejecución en base al script `rc` que puede estar situado en `/etc/rc.d` o `/etc/init.d` pasandole por parametro el nivel de ejecución.

El script `rc` ejecuta los ficheros que existen en `/etc/rcN.d` siendo `N` el nivel de ejecución. Estos ficheros serán enlaces a `/etc/init.d` que comenzarán por `S` para iniciar el servicio o `K` para terminar con el servicio.

```
root@debian12:/# ls -ld /etc/rc*
drwxr-xr-x 2 root root 4096 abr 13 14:05 /etc/rc0.d
drwxr-xr-x 2 root root 4096 abr 13 14:05 /etc/rc1.d
drwxr-xr-x 2 root root 4096 abr 13 14:05 /etc/rc2.d
drwxr-xr-x 2 root root 4096 abr 13 14:05 /etc/rc3.d
drwxr-xr-x 2 root root 4096 abr 13 14:05 /etc/rc4.d
drwxr-xr-x 2 root root 4096 abr 13 14:05 /etc/rc5.d
drwxr-xr-x 2 root root 4096 abr 13 14:05 /etc/rc6.d
drwxr-xr-x 2 root root 4096 feb  2 17:52 /etc/rcS.d

root@debian12:/# ls -l /etc/rc*

/etc/rc0.d:
total 0
lrwxrwxrwx 1 root root 20 feb  2 17:52 K01alsa-utils -> ../init.d/alsa-utils
lrwxrwxrwx 1 root root 19 feb  2 17:52 K01bluetooth -> ../init.d/bluetooth
lrwxrwxrwx 1 root root 20 feb  2 17:52 K01cryptdisks -> ../init.d/cryptdisks
lrwxrwxrwx 1 root root 26 feb  2 17:52 K01cryptdisks-early ->
../init.d/cryptdisks-early
lrwxrwxrwx 1 root root 22 feb  2 17:52 K01cups-browsed -> ../init.d/cups-browsed
lrwxrwxrwx 1 root root 15 feb  2 17:52 K01exim4 -> ../init.d/exim4
lrwxrwxrwx 1 root root 20 feb  2 17:52 K01hwclock.sh -> ../init.d/hwclock.sh
lrwxrwxrwx 1 root root 17 feb  2 17:52 K01lightdm -> ../init.d/lightdm
```

```

lrwxrwxrwx 1 root root 20 feb 2 17:52 K01live-tools -> ../init.d/live-tools
lrwxrwxrwx 1 root root 15 feb 2 17:52 K01mdadm -> ../init.d/mdadm
lrwxrwxrwx 1 root root 24 feb 2 17:52 K01mdadm-waitidle -> ../init.d/mdadm-waitidle
lrwxrwxrwx 1 root root 20 feb 2 17:52 K01networking -> ../init.d/networking
lrwxrwxrwx 1 root root 18 feb 2 17:52 K01plymouth -> ../init.d/plymouth
lrwxrwxrwx 1 root root 37 feb 2 17:52 K01pulseaudio-enable-autospawn ->
../init.d/pulseaudio-enable-autospawn
lrwxrwxrwx 1 root root 15 feb 2 17:52 K01saned -> ../init.d/saned
lrwxrwxrwx 1 root root 23 feb 2 17:52 K01smartmontools -> ../init.d/smartmontools
lrwxrwxrwx 1 root root 27 feb 2 17:52 K01speech-dispatcher -> ../init.d/speech-dispatcher
lrwxrwxrwx 1 root root 14 feb 2 17:52 K01udev -> ../init.d/udev
lrwxrwxrwx 1 root root 15 feb 2 17:52 K01uuid -> ../init.d/uuid

/etc/rc1.d:
total 0
lrwxrwxrwx 1 root root 20 feb 2 17:52 K01alsa-utils -> ../init.d/alsa-utils
lrwxrwxrwx 1 root root 19 feb 2 17:52 K01bluetooth -> ../init.d/bluetooth
lrwxrwxrwx 1 root root 14 feb 2 17:52 K01cups -> ../init.d/cups
lrwxrwxrwx 1 root root 22 feb 2 17:52 K01cups-browsed -> ../init.d/cups-browsed
lrwxrwxrwx 1 root root 15 feb 2 17:52 K01exim4 -> ../init.d/exim4
lrwxrwxrwx 1 root root 17 feb 2 17:52 K01lightdm -> ../init.d/lightdm
lrwxrwxrwx 1 root root 15 feb 2 17:52 K01mdadm -> ../init.d/mdadm
lrwxrwxrwx 1 root root 37 feb 2 17:52 K01pulseaudio-enable-autospawn ->
../init.d/pulseaudio-enable-autospawn
lrwxrwxrwx 1 root root 15 feb 2 17:52 K01saned -> ../init.d/saned
lrwxrwxrwx 1 root root 23 feb 2 17:52 K01smartmontools -> ../init.d/smartmontools
lrwxrwxrwx 1 root root 27 feb 2 17:52 K01speech-dispatcher -> ../init.d/speech-dispatcher
lrwxrwxrwx 1 root root 15 feb 2 17:52 K01uuid -> ../init.d/uuid

/etc/rc2.d:
total 0
lrwxrwxrwx 1 root root 27 feb 2 17:52 K01speech-dispatcher -> ../init.d/speech-dispatcher
lrwxrwxrwx 1 root root 17 feb 2 17:52 S01anacron -> ../init.d/anacron
lrwxrwxrwx 1 root root 19 feb 2 17:52 S01bluetooth -> ../init.d/bluetooth
lrwxrwxrwx 1 root root 26 feb 2 17:52 S01console-setup.sh -> ../init.d/console-setup.sh
lrwxrwxrwx 1 root root 14 feb 2 17:52 S01cron -> ../init.d/cron
lrwxrwxrwx 1 root root 14 feb 2 17:52 S01cups -> ../init.d/cups
lrwxrwxrwx 1 root root 22 feb 2 17:52 S01cups-browsed -> ../init.d/cups-browsed
lrwxrwxrwx 1 root root 14 feb 2 17:52 S01dbus -> ../init.d/dbus
lrwxrwxrwx 1 root root 15 feb 2 17:52 S01exim4 -> ../init.d/exim4
lrwxrwxrwx 1 root root 17 feb 2 17:52 S01lightdm -> ../init.d/lightdm
lrwxrwxrwx 1 root root 15 feb 2 17:52 S01mdadm -> ../init.d/mdadm
lrwxrwxrwx 1 root root 18 feb 2 17:52 S01plymouth -> ../init.d/plymouth
lrwxrwxrwx 1 root root 37 feb 2 17:52 S01pulseaudio-enable-autospawn ->
../init.d/pulseaudio-enable-autospawn
lrwxrwxrwx 1 root root 15 feb 2 17:52 S01rsync -> ../init.d/rsync
lrwxrwxrwx 1 root root 15 feb 2 17:52 S01saned -> ../init.d/saned
lrwxrwxrwx 1 root root 23 feb 2 17:52 S01smartmontools -> ../init.d/smartmontools
lrwxrwxrwx 1 root root 13 abr 12 19:14 S01ssh -> ../init.d/ssh

```

```
lrwxrwxrwx 1 root root 14 feb  2 17:52 S01sudo -> ../init.d/sudo
lrwxrwxrwx 1 root root 17 feb  2 17:52 S01sysstat -> ../init.d/sysstat
lrwxrwxrwx 1 root root 15 feb  2 17:52 S01uidd -> ../init.d/uidd

/etc/rc3.d:
...

/etc/rc4.d:
...

/etc/rc5.d:
...

/etc/rc6.d:
...
```

systemd

- Se ejecuta un único programa **systemd** que utilizará ficheros de configuración para cada elemento a gestionar llamados **unidades**, que pueden ser de diversos tipos: automount, device, mount, path, **service**, snapshot, socket y target.
- Las **unidades** se agrupan en **target**, donde también podemos definir el orden de ejecución y las dependencias con otros target o services. Son los equivalentes a los runlevels en SysVinit (hay target compatibles con estos). A continuación se adjunta una equivalencia aproximada entre los runlevels en SysVinit y los targets en systemd:
- **Runlevel 0**: Apagado del sistema.
 - **Systemd Target**: **poweroff.target**
- **Runlevel 1**: Modo de rescate o de un solo usuario.
 - **Systemd Target**: **rescue.target**
- **Runlevel 2**: Multiusuario sin red.
 - **Systemd Target**: **multi-user.target**
- **Runlevel 3**: Multiusuario con red.
 - **Systemd Target**: **multi-user.target**
- **Runlevel 4**: Reservado para un uso personalizado.
 - **Systemd Target**: No tiene un equivalente directo, se puede personalizar según las necesidades.
- **Runlevel 5**: Multiusuario con interfaz gráfica (GUI).
 - **Systemd Target**: **graphical.target**
- **Runlevel 6**: Reinicio del sistema.

◦ **Systemd Target:** `reboot.target`

- Es importante tener en cuenta que, aunque hay una cierta correspondencia entre los runlevels de SysVinit y los targets de systemd, systemd es más flexible y puede tener una configuración diferente en diferentes distribuciones y sistemas. Además, systemd introduce conceptos adicionales como "targets especiales" (`default.target`, `emergency.target`, etc.) que no tienen un equivalente directo en SysVinit.
- Cada unidad se define en un fichero con el nombre de dicha unidad y en la extensión se indica el tipo de unidad, por ejemplo `ssh.service` que se encuentra en `/etc/systemd/system`.

```
root@debian12:/# cat /etc/systemd/system/sshd.service
[Unit]
Description=OpenBSD Secure Shell server
Documentation=man:sshd(8) man:sshd_config(5)
After=network.target auditd.service
ConditionPathExists=!/etc/ssh/sshd_not_to_be_run

[Service]
EnvironmentFile=-/etc/default/ssh
ExecStartPre=/usr/sbin/sshd -t
ExecStart=/usr/sbin/sshd -D $SSHD_OPTS
ExecReload=/usr/sbin/sshd -t
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process
Restart=on-failure
RestartPreventExitStatus=255
Type=notify
RuntimeDirectory=sshd
RuntimeDirectoryMode=0755

[Install]
WantedBy=multi-user.target --> Cuando se cargue esta unidad ()
Alias=sshd.service
```

Aquí podemos ver la correspondencia entre los diferentes `target` actuales y el `runlevel`.

```
root@debian12:/# ls -ld /usr/lib/systemd/system/
drwxr-xr-x 29 root root 20480 abr 13 14:05 /usr/lib/systemd/system/
root@debian12:/# ls -ld /usr/lib/systemd/system/runlevel*
lrwxrwxrwx 1 root root 15 feb 2 17:48 /usr/lib/systemd/system/runlevel0.target
-> poweroff.target
lrwxrwxrwx 1 root root 13 feb 2 17:48 /usr/lib/systemd/system/runlevel1.target
-> rescue.target
drwxr-xr-x 2 root root 4096 nov 10 01:25
/usr/lib/systemd/system/runlevel1.target.wants
lrwxrwxrwx 1 root root 17 feb 2 17:48 /usr/lib/systemd/system/runlevel2.target
-> multi-user.target
drwxr-xr-x 2 root root 4096 nov 10 01:25
/usr/lib/systemd/system/runlevel2.target.wants
```

```
lrwxrwxrwx 1 root root 17 feb 2 17:48 /usr/lib/systemd/system/runlevel3.target
-> multi-user.target
drwxr-xr-x 2 root root 4096 nov 10 01:25
/usr/lib/systemd/system/runlevel3.target.wants
lrwxrwxrwx 1 root root 17 feb 2 17:48 /usr/lib/systemd/system/runlevel4.target
-> multi-user.target
drwxr-xr-x 2 root root 4096 nov 10 01:25
/usr/lib/systemd/system/runlevel4.target.wants
lrwxrwxrwx 1 root root 16 feb 2 17:48 /usr/lib/systemd/system/runlevel5.target
-> graphical.target
drwxr-xr-x 2 root root 4096 nov 10 01:25
/usr/lib/systemd/system/runlevel5.target.wants
lrwxrwxrwx 1 root root 13 feb 2 17:48 /usr/lib/systemd/system/runlevel6.target
-> reboot.target
``
```

sed

El comando **sed** en Linux es un editor de flujo de texto que permite realizar cambios en archivos de texto desde la línea de comandos. Por ejemplo, para reemplazar todas las instancias de "hola" por "adiós" en un archivo llamado **archivo.txt**, usarías el siguiente comando:

```
sed 's/hola/adiós/g' archivo.txt
```

1. **-e**: Este parámetro permite especificar múltiples comandos de **sed** en una sola línea de comando. Por ejemplo, **sed -e 'comando1' -e 'comando2' archivo** ejecutaría ambos **comando1** y **comando2** en el archivo.
2. **-i**: Modificará el archivo de entrada directamente. Por ejemplo, **sed -i 's/antiguo/nuevo/g' archivo** cambiaría todas las ocurrencias de "antiguo" por "nuevo" en el archivo **archivo**, modificando el archivo en su lugar.
3. **-n**: Este parámetro suprime la salida automática de **sed**. Por defecto, **sed** imprime todas las líneas después de aplicar los comandos. Con **-n**, solo imprime las líneas que se le indiquen explícitamente. Por ejemplo, **sed -n '5p' archivo** imprimiría solo la quinta línea del archivo.
4. **-r**: Permite el uso de expresiones regulares.

Ejemplos de uso

Cambia los parametros " y , por y edita el fichero file.tmp.

```
si@si-VirtualBox:/tmp/prueba$ sed -i -e 's#"# #g' -e 's#,# #g' file.tmp
user11  p11  /bin/bash  /tmp
user2   p2   /bin/false /home/user2
user2   p2   /bin/false /home/user2
```

- No muestra cambios por pantalla por `-n`.

```
si@si-VirtualBox:/tmp/prueba$ sed -n -e 's#"# #g' -e 's#,# #g' file.tmp
si@si-VirtualBox:/tmp/prueba$
```

- Si no se usa `g` solo se elimina la primera ocurrencia por linea.

```
si@si-VirtualBox:/tmp/prueba$ sed -e 's#user2#usuario#g' file.tmp
"user11","p11","/bin/bash","/tmp"
"usuario","p2","/bin/false","/home/usuario"
"usuario","p2","/bin/false","/home/usuario"

si@si-VirtualBox:/tmp/prueba$ sed -e 's#user2#usuario#' file.tmp
"user11","p11","/bin/bash","/tmp"
"usuario","p2","/bin/false","/home/user2"
"usuario","p2","/bin/false","/home/user2"
```

- La opción `p` hace que se muestren las lineas donde se han realizado sustituciones.

```
si@si-VirtualBox:/tmp/prueba$ sed -e 's/user2/usuario/gp' file.tmp
"user11","p11","/bin/bash","/tmp"
"usuario","p2","/bin/false","/home/usuario"
"usuario","p2","/bin/false","/home/usuario"
"usuario","p2","/bin/false","/home/usuario"
"usuario","p2","/bin/false","/home/usuario"

si@si-VirtualBox:/tmp/prueba$ sed -n -e 's/user2/usuario/gp' file.tmp
"usuario","p2","/bin/false","/home/usuario"
"usuario","p2","/bin/false","/home/usuario"
```

- Elimina las lineas con sed y `d`.

```
si@si-VirtualBox:/tmp/prueba$ for i in $(seq 1 80); do $(touch prueba.txt && echo
"${i}" >> prueba.txt); done;
si@si-VirtualBox:/tmp/prueba$ head prueba.txt
1
2
3
4
5
6
7
8
9
10
```



```
si@si-VirtualBox:/tmp/prueba$ sed -i '2,7d' prueba.txt
si@si-VirtualBox:/tmp/prueba$ head prueba.txt
1
8
9
10
11
12
13
14
15
16
```

```
si@si-VirtualBox:/tmp/prueba$ sed -i '1d' prueba.txt
si@si-VirtualBox:/tmp/prueba$ head prueba.txt
8
9
10
11
12
13
14
15
16
17
```

```
si@si-VirtualBox:/tmp/prueba$ sed -i 1'd' prueba.txt
si@si-VirtualBox:/tmp/prueba$ head prueba.txt
9
10
11
12
13
14
15
16
17
18
```

```
si@si-VirtualBox:/tmp/prueba$ sed -i 1,5'd' prueba.txt
si@si-VirtualBox:/tmp/prueba$ head prueba.txt
14
15
16
17
18
```

```
19
20
21
22
23
```

- Rangos para sustituir.

```
si@si-VirtualBox:/tmp/prueba$ sed -i 's/false/bash/g' file.tmp
si@si-VirtualBox:/tmp/prueba$ cat file.tmp
"user11","p11", "/bin/bash", "/tmp"
"user2","p2", "/bin/bash", "/home/user2"
"user2","p2", "/bin/bash", "/home/user2"

si@si-VirtualBox:/tmp/prueba$ sed 2's/bash/false/g' file.tmp
"user11","p11", "/bin/bash", "/tmp"
"user2","p2", "/bin/false", "/home/user2"
"user2","p2", "/bin/bash", "/home/user2"

si@si-VirtualBox:/tmp/prueba$ sed '2s/bash/false/g' file.tmp
"user11","p11", "/bin/bash", "/tmp"
"user2","p2", "/bin/false", "/home/user2"
"user2","p2", "/bin/bash", "/home/user2"

si@si-VirtualBox:/tmp/prueba$ sed 1,3's/bash/false/g' file.tmp
"user11","p11", "/bin/false", "/tmp"
"user2","p2", "/bin/false", "/home/user2"
"user2","p2", "/bin/false", "/home/user2"

si@si-VirtualBox:/tmp/prueba$ sed '1,3s/bash/false/g' file.tmp
"user11","p11", "/bin/false", "/tmp"
"user2","p2", "/bin/false", "/home/user2"
"user2","p2", "/bin/false", "/home/user2"
```

- Guardar la modificación en otro fichero con **w**

```
si@si-VirtualBox:/tmp/prueba$ sed -e "s/user/usuario/gw fileModificado.tmp"
file.tmp
"usuario11","p11", "/bin/bash", "/tmp"
"usuario2","p2", "/bin/bash", "/home/usuario2"
"usuario2","p2", "/bin/bash", "/home/usuario2"

si@si-VirtualBox:/tmp/prueba$ cat fileModificado.tmp
"usuario11","p11", "/bin/bash", "/tmp"
"usuario2","p2", "/bin/bash", "/home/usuario2"
"usuario2","p2", "/bin/bash", "/home/usuario2"
```

find

Permite buscar archivos y directorios en el sistema de archivos basándose en diversos criterios como el nombre del archivo, el tipo, la fecha de modificación, etc.

Además, es muy común emplear `find` con `-exec` y `xargs`. Ambos son formas de ejecutar comandos en archivos encontrados por `find`, pero difieren en su funcionamiento y flexibilidad.

- **-exec con find:** Con `-exec`, `find` ejecuta el comando especificado una vez por cada archivo que encuentra.

```
find . -name "*.txt" -exec cp {} /ruta/de/destino \;
```

- **xargs con find:** `xargs` es un comando que toma la entrada estándar y la convierte en argumentos para otro comando.

```
find . -name "*.txt" | xargs -I PATTERN cp -t PATTERN /ruta/de/destino
```

Si se necesita ejecutar un comando simple en cada archivo encontrado por `find`, `-exec` es la opción más directa. Sin embargo, si se quiere realizar manipulaciones adicionales en la lista de archivos o si la lista de archivos es muy larga, `xargs` puede ser más apropiado.

```
si@si-VirtualBox:/tmp/prueba$ ls
fileModificado.tmp file.tmp file.tmp2 prueba.txt

si@si-VirtualBox:/tmp/prueba$ find . -name "file*" -exec ls -l {} \;
-rw-rw-r-- 1 si si 127 abr 14 16:12 ./fileModificado.tmp
-rw-rw-r-- 1 si si 114 abr 14 15:51 ./file.tmp2
-rw-rw-r-- 1 si si 112 abr 14 16:02 ./file.tmp

si@si-VirtualBox:/tmp/prueba$ find . -name "file*" | xargs -I X ls -l X;
-rw-rw-r-- 1 si si 127 abr 14 16:12 ./fileModificado.tmp
-rw-rw-r-- 1 si si 114 abr 14 15:51 ./file.tmp2
-rw-rw-r-- 1 si si 112 abr 14 16:02 ./file.tmp
```

Parametros de find

1. -name:

- Utilizado para buscar archivos por su nombre exacto (distingue entre mayúsculas y minúsculas).
- Ejemplo: Buscar todos los archivos con extensión ".txt".

```
find /ruta -name "*.txt"
```

2. -iname:

- Similar a `-name`, pero no distingue entre mayúsculas y minúsculas.
- Ejemplo: Buscar archivos con cualquier extensión de texto, ignorando mayúsculas y minúsculas.

```
find /ruta -iname "*.txt"
```

3. -perm:

- Utilizado para buscar archivos por permisos.
- Ejemplo: Buscar archivos con permisos de lectura, escritura y ejecución para el propietario. El `/700` encontraría archivos con al menos todos los permisos para root y cualquier otro permiso para go.

```
find /ruta -perm 700  
find /ruta -perm /700
```

4. -maxdepth:

- Especifica la profundidad máxima de búsqueda en el árbol de directorios.
- Ejemplo: Buscar archivos en el directorio actual y un nivel hacia abajo.

```
find . -maxdepth 1 -type f
```

5. -type:

- Filtra archivos por tipo (fichero regular, directorio, enlace simbólico, etc.).
- Ejemplo: Buscar todos los directorios en el sistema.

```
find / -type d
```

6. -user:

- Utilizado para buscar archivos pertenecientes a un usuario específico.
- Ejemplo: Buscar todos los archivos pertenecientes al usuario "usuario1".

```
find /ruta -user usuario1
```

Para combinar condiciones con `find`, puedes utilizar `-o` (OR) y `-a` (AND):

- **-o (OR):** Encuentra archivos que cumplan al menos una de las condiciones especificadas.

```
find / -name "*.txt" -o -name "*.jpg"
```

- **-a (AND):** Encuentra archivos que cumplan todas las condiciones especificadas.

```
find /ruta -name "*.txt" -a -user usuario1
```

Estos son solo ejemplos básicos de cómo usar cada parámetro y cómo combinar condiciones con **-o** y **-a** en **find**. Las posibilidades son bastante amplias y puedes construir búsquedas más complejas según las necesidades como por ejemplo:

```
find $HOME -type f -iname "*.png" -mtime +3 -mtime -5 -perm 644 -size +2M -user  
www-data -and -not -user root -and -group www-  
data -a ! -group root -exec ls -lah {} \; 2>/dev/null
```

cut

cut:

El comando **cut** se utiliza para cortar secciones específicas de cada línea de un archivo de texto. Su sintaxis básica es:

```
cut [opciones] archivo
```

- **-d:**

La opción **-d** se utiliza para especificar el delimitador que **cut** debe utilizar para separar los campos en cada línea del archivo de entrada. Por defecto, **cut** utiliza el tabulador como delimitador. La sintaxis es la siguiente:

```
cut -d <delimitador> archivo
```

- **<delimitador>**: Especifica el carácter que actúa como delimitador de campos.

Por ejemplo, si tienes un archivo de texto donde los campos están separados por comas (,), puedes usar la opción **-d** para extraer los campos correctamente:

```
cut -d ',' -f1,3 archivo.csv
```

Esto extraería el primer y tercer campo de cada línea del archivo `archivo.csv`, considerando que los campos están separados por comas.

- **-c:**

La opción `-c` se utiliza para seleccionar caracteres específicos de cada línea en lugar de campos delimitados. La sintaxis es:

```
cut -c <lista_de_caracteres> archivo
```

- `<lista_de_caracteres>`: Especifica los caracteres que deseas seleccionar de cada línea. Puedes especificar un rango de caracteres utilizando el formato `inicio-fin`, o simplemente enumerar los caracteres que deseas seleccionar.

Por ejemplo, si deseas extraer los primeros tres caracteres de cada línea de un archivo, puedes hacerlo así:

```
cut -c 1-3 archivo.txt
```

Esto devolverá los primeros tres caracteres de cada línea en el archivo `archivo.txt`.

También puedes especificar caracteres individuales:

```
cut -c 1,3,5 archivo.txt
```

Esto devolverá el primer, tercer y quinto carácter de cada línea en el archivo `archivo.txt`.

Además, si necesitas excluir ciertos caracteres, puedes usar el signo de menos `-`:

```
cut -c -3 archivo.txt
```

Esto devolverá todos los caracteres de cada línea hasta el tercero, excluyéndolo.

Nota: Con `cut` si escribimos `-f3,1` no obtenemos la tercera columna y luego la primera. Obtenemos siempre primera y luego tercera. Para este comportamiento necesitaremos el `awk`

```
si@si-VirtualBox:~$ head /etc/passwd | cut -d ':' -f1,2
root:x
daemon:x
bin:x
sys:x
sync:x
games:x
man:x
```

```
lp:x
mail:x
news:x

si@si-VirtualBox:~$ head /etc/passwd | cut -d ':' -f2,1
root:x
daemon:x
bin:x
sys:x
sync:x
games:x
man:x
lp:x
mail:x
news:x
```

awk

awk es un lenguaje de programación de patrones y acciones que se utiliza para el procesamiento de texto. Aunque es más poderoso y versátil que **cut**, también puede ser más complejo de usar. Su sintaxis básica es:

```
awk '{patrón}' archivo
```

- **{patrón}**: Especifica el patrón que **awk** buscará en cada línea del archivo y qué acciones tomará cuando encuentre una línea que coincida con el patrón.
- **archivo**: Es el archivo que se va a procesar.

Por ejemplo, para imprimir el primer campo de cada línea de un archivo, puedes usar:

```
awk -F'addr:' '{print $2 " y " $1}' archivo.txt
```

El comando de **awk** anterior realiza:

- **awk**: Es el comando **awk** que invoca el intérprete de **awk** para procesar el texto.
- **-F'addr: '**: La opción **-F** especifica el delimitador de campo utilizado por **awk**. En este caso, se establece como **'addr: '**, lo que significa que **awk** dividirá cada línea de entrada en campos cada vez que encuentre la cadena **'addr: '**.
- **' {print \$2 \$1}'**: Esta es la acción que **awk** tomará en cada línea de entrada. En este caso, **\$2** hace referencia al segundo campo y **\$1** al primer campo después de dividir la línea según el delimitador especificado. La acción **print** imprime los campos especificados. Al imprimir **\$2** antes de **\$1** y no separarlos con una coma ni espacio, se concatenarán los campos sin ningún espacio adicional entre ellos.
- **awk** es muy interesante para poder mostrar la última columna si no sabemos en qué posición está con el valor **\$NF**.

```
si@si-VirtualBox:~$ cat /etc/passwd | awk -F ':' '{print $NF}'  
/bin/bash  
/usr/sbin/nologin  
/usr/sbin/nologin  
/usr/sbin/nologin  
/bin/sync
```

Por ejemplo, si tenemos una línea de entrada como esta:

```
addr:192.168.1.1
```

El comando `awk` separará esta línea en dos campos: "addr" y "192.168.1.1". Luego, al imprimir `$2 $1`, producirá la salida:

```
192.168.1.1addr:
```

Lo que significa que `$2` se colocará antes que `$1`, y no habrá espacio ni otro carácter entre ellos.

Este comando puede ser útil para cambiar el orden o el formato de los campos en líneas de texto que siguen un patrón específico, como en este caso, donde se manipula una dirección IP precedida por la etiqueta "addr:".

`ps`, `psgrep`, `ptree`, `pidof`, `kill`, `killall`

`ps` muestra los procesos asociados con la terminal desde la que se ejecuta el comando. Muestra el PID (identificador de proceso), la terminal asociada, tiempo desde que se anzó el proceso y el comando que lo desencadena.

```
si@si-VirtualBox:~$ ps  
  PID TTY          TIME CMD  
 2708 pts/1    00:00:00 bash  
 3716 pts/1    00:03:34 yes  
 3734 pts/1    00:00:00 ps
```

1. **Sintaxis Estándar:** Esta es una convención comúnmente utilizada en sistemas que siguen las especificaciones de POSIX (Portable Operating System Interface), como la mayoría de las distribuciones de Linux.

`ps -e` muestra los procesos de todo el sistema en cualquier terminal. La información que obtenemos por pantalla es la misma que con `ps` pero para todas las terminales.


```
si@si-VirtualBox:~$ ps -e
  PID TTY          TIME CMD
    1 ?           00:00:02 systemd
    2 ?           00:00:00 kthreadd
    3 ?           00:00:00 rcu_gp
    4 ?           00:00:00 rcu_par_gp
    5 ?           00:00:00 slub_flushwq
    6 ?           00:00:00 netns
    8 ?           00:00:00 kworker/0:0H-events_highpri
    9 ?           00:00:01 kworker/0:1-events
```

ps -p muestra el proceso según su pid.

```
si@si-VirtualBox:~$ ps -p `pidof bash`
  PID TTY          TIME CMD
 1457 pts/1      00:00:00 bash
```

ps -ef muestra todos los procesos del sistema con mayor nivel de detalle que **ps -e**.

```
si@si-VirtualBox:~$ ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1        0  0  16:07 ?           00:00:02 /sbin/init splash
root           2        0  0  16:07 ?           00:00:00 [kthreadd]
root           3        2  0  16:07 ?           00:00:00 [rcu_gp]
root           4        2  0  16:07 ?           00:00:00 [rcu_par_gp]
root           5        2  0  16:07 ?           00:00:00 [slub_flushwq]
root           6        2  0  16:07 ?           00:00:00 [netns]
root           8        2  0  16:07 ?           00:00:00 [kworker/0:0H-events_highpri]
root           9        2  0  16:07 ?           00:00:01 [kworker/0:1-events]
root          11        2  0  16:07 ?           00:00:00 [mm_percpu_wq]
root          12        2  0  16:07 ?           00:00:00 [rcu_tasks_kthread]
```

ps -efH muestra todos los procesos del sistema con mayor nivel de detalle que **ps -e** y en forma de arbol.

```
si@si-VirtualBox:~$ ps -efH
UID          PID    PPID  C STIME TTY          TIME CMD
root           2        0  0  16:07 ?           00:00:00 [kthreadd]
root           3        2  0  16:07 ?           00:00:00  [rcu_gp]
root           4        2  0  16:07 ?           00:00:00  [rcu_par_gp]
root           5        2  0  16:07 ?           00:00:00  [slub_flushwq]
root           6        2  0  16:07 ?           00:00:00  [netns]
root           8        2  0  16:07 ?           00:00:00  [kworker/0:0H-
events_highpri]
root           9        2  0  16:07 ?           00:00:01  [kworker/0:1-events]
```

```
root          11          2  0 16:07 ?           00:00:00 [mm_percpu_wq]
root          12          2  0 16:07 ?           00:00:00 [rcu_tasks_kthread]
```

ps -o muestra la información de los procesos que se especifique por el usuario.

```
si@si-VirtualBox:~$ ps -eo cmd,pid,ppid,time,user,%cpu, --sort -%mem | head
CMD                                PID    PPID    TIME USER    %CPU
/usr/sbin/mysqld                   761      1 00:00:18 mysql    0.7
/usr/bin/gnome-shell               3849    2516 00:00:07 si        0.4
/usr/bin/python3 /usr/bin/u        5628    2516 00:00:04 si        0.2
/snap/snap-store/959/usr/bi       4142    2516 00:00:02 si        0.1
/usr/libexec/fwupd/fwupd           4537      1 00:00:01 root      0.0
/usr/libexec/gsd-xsettings         4345    2516 00:00:00 si        0.0
/usr/libexec/evolution-data        4179    3821 00:00:00 si        0.0
/usr/bin/Xwayland :0 -rootl        4261    3849 00:00:00 si        0.0
/usr/libexec/packagekitd           3189      1 00:00:06 root      0.2
```

2. **Sintaxis BSD:** Esta es una convención utilizada principalmente en sistemas derivados de BSD (Berkeley Software Distribution), como FreeBSD, OpenBSD y macOS. En la sintaxis BSD, las opciones del comando **ps** se especifican sin guiones y pueden estar combinadas.

ps aux o **ps -aux** muestra una lista detallada de todos los procesos en el sistema.

```
si@si-VirtualBox:~$ ps aux
USER          PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1   0.2   0.3 168064 13252 ?        Ss   16:07   0:02 /sbin/init
splash
root           2   0.0   0.0     0     0 ?        S    16:07   0:00 [kthreadd]
root           3   0.0   0.0     0     0 ?        I<   16:07   0:00 [rcu_gp]
root           4   0.0   0.0     0     0 ?        I<   16:07   0:00 [rcu_par_gp]
root           5   0.0   0.0     0     0 ?        I<   16:07   0:00 [slub_flushwq]
root           6   0.0   0.0     0     0 ?        I<   16:07   0:00 [netns]
root           8   0.0   0.0     0     0 ?        I<   16:07   0:00 [kworker/0:0H-events_highpri]
root           9   0.1   0.0     0     0 ?        I    16:07   0:01 [kworker/0:1-events]
root          11   0.0   0.0     0     0 ?        I<   16:07   0:00 [mm_percpu_wq]
root          12   0.0   0.0     0     0 ?        I    16:07   0:00 [rcu_tasks_kthread]
```

pstree muestra información de los procesos en forma de árbol

```
si@si-VirtualBox:~$ pstree -u si | head
gdm-wayland-ses-+-gnome-session-b--2*[{gnome-session-b}]
    `--2*[{gdm-wayland-ses}]

gnome-keyring-d---3*[{gnome-keyring-d}]
```

```
sshd---bash--head
      | -pstree
      ` -yes

systemd--(sd-pam)
```

pidof es más simple y directo, útil para encontrar PIDs basados en nombres de procesos específicos, mientras que **pgrep** es más versátil y puede realizar búsquedas más avanzadas con una variedad de criterios. La elección entre ellos depende de las necesidades específicas de búsqueda que tengas en un momento dado.

pgrep muestra los IDs de proceso (PID) de los procesos sshd que están siendo ejecutados por los usuarios root y si.

```
si@si-VirtualBox:~$ pgrep -u root,si sshd
735
1331
2707
```

pidof muestra los IDs de proceso (PID) asociados con un programa específico.

```
si@si-VirtualBox:~$ pidof yes
3716
```

Señales

En sistemas operativos basados en Unix, incluyendo Linux, las señales son mecanismos de comunicación entre procesos y entre el kernel y los procesos. Estas señales se utilizan para notificar a un proceso de eventos importantes, solicitar la terminación de un proceso, manejar errores, y para una variedad de otras funciones.

```
si@si-VirtualBox:~$ kill -l
1) SIGHUP      2) SIGINT      3) SIGQUIT     4) SIGILL      5) SIGTRAP
6) SIGABRT     7) SIGBUS      8) SIGFPE      9) SIGKILL     10) SIGUSR1
11) SIGSEGV    12) SIGUSR2    13) SIGPIPE    14) SIGALRM     15) SIGTERM
16) SIGSTKFLT  17) SIGCHLD    18) SIGCONT    19) SIGSTOP     20) SIGTSTP
21) SIGTTIN    22) SIGTTOU    23) SIGURG     24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM  27) SIGPROF    28) SIGWINCH   29) SIGIO        30) SIGPWR
31) SIGSYS     34) SIGRTMIN   35) SIGRTMIN+1 36) SIGRTMIN+2 37) SIGRTMIN+3
38) SIGRTMIN+4 39) SIGRTMIN+5 40) SIGRTMIN+6 41) SIGRTMIN+7 42) SIGRTMIN+8
43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6 59) SIGRTMAX-5 60) SIGRTMAX-4 61) SIGRTMAX-3 62) SIGRTMAX-2
63) SIGRTMAX-1 64) SIGRTMAX
```

Estas señales son parte del mecanismo de gestión de señales en Linux y se utilizan para diversos propósitos, desde el control de procesos hasta la notificación de eventos importantes.

El comando `kill` en Linux se utiliza para enviar señales a procesos en ejecución, mientras que `killall` se usa para enviar señales a todos los procesos que coincidan con un nombre determinado. Aquí tienes ejemplos de cómo usar estos comandos con las señales mencionadas:

1. SIGINT (2):

- Es la señal de interrupción. Se envía a un proceso cuando el usuario presiona `Ctrl+C` en la terminal. Por lo general, esto indica al proceso que debe **detener su ejecución de forma controlada**.

```
kill -2 1234
kill -INT $(pidof yes)
kill -s INT $(pidof yes)
```

2. SIGKILL (9):

- Es una señal especial que **no puede ser manejada ni ignorada por los procesos**. Se utiliza para terminar un proceso de forma inmediata y forzosa. No permite al proceso realizar ninguna acción de limpieza.

```
kill -9 5678
kill -KILL $(pidof yes)
kill -s KILL $(pidof yes)
```

3. SIGCONT (18):

- Indica al sistema operativo que continúe la ejecución de un proceso que previamente ha sido detenido (por ejemplo, con SIGSTOP).

```
kill -18 7890
kill -CONT $(pidof yes)
kill -s CONT $(pidof yes)
```

4. SIGSTOP (19):

- Detiene la ejecución de un proceso. Similar a SIGKILL pero permite reanudar la ejecución posteriormente. No puede ser manejada ni ignorada por los procesos

```
kill -19 2345
kill -STOP $(pidof yes)
kill -s STOP $(pidof yes)
```

5. SIGTSTP (20):

- Se envía a un proceso cuando el usuario presiona **Ctrl+Z** en la terminal. Detiene la ejecución del proceso, **permite que el proceso sea reanudado más tarde**.

```
kill -20 3456
kill -TSTP $(pidof yes)
kill -s TSTP $(pidof yes)
```

```
si@si-VirtualBox:~$ ps
  PID TTY          TIME CMD
 2708 pts/1        00:00:00 bash
 13649 pts/1        00:00:00 ps
si@si-VirtualBox:~$ yes >/dev/null &
[1] 13650
si@si-VirtualBox:~$ ps
  PID TTY          TIME CMD
 2708 pts/1        00:00:00 bash
 13650 pts/1        00:00:00 yes
 13651 pts/1        00:00:00 ps
si@si-VirtualBox:~$ killall yes
[1]+  Terminated                  yes > /dev/null
si@si-VirtualBox:~$ ps
  PID TTY          TIME CMD
 2708 pts/1        00:00:00 bash
 13653 pts/1        00:00:00 ps
si@si-VirtualBox:~$ yes >/dev/null &
[1] 13654
si@si-VirtualBox:~$ killall -s INT yes
[1]+  Interrupt                      yes > /dev/null
si@si-VirtualBox:~$ ps
  PID TTY          TIME CMD
 2708 pts/1        00:00:00 bash
 13656 pts/1        00:00:00 ps
```

mount y umount

El comando **mount** y **umount** son fundamentales en sistemas Unix/Linux para montar y desmontar sistemas de archivos respectivamente. Aquí te explico cómo funcionan y cómo se utilizan:

mount:

El comando **mount** se utiliza para montar (conectar) un sistema de archivos en una ubicación específica dentro del árbol de directorios del sistema. Esto permite que el contenido del sistema de archivos sea accesible en esa ubicación. Aquí hay una descripción básica de cómo se utiliza:

```
mount [opciones] dispositivo punto_de_montaje
```

- **dispositivo** es la partición o dispositivo que contiene el sistema de archivos que deseas montar.
- **punto_de_montaje** es el directorio en el que deseas montar el sistema de archivos.

Por ejemplo, si tienes una partición **/dev/sdb1** que quieres montar en **/mnt/nueva_particion**, puedes hacerlo así:

```
mount /dev/sdb1 /mnt/nueva_particion
```

Además de montar dispositivos, **mount** también puede montar sistemas de archivos de red, como NFS (Network File System) o Samba.

umount:

El comando **umount** se utiliza para desmontar (desconectar) un sistema de archivos previamente montado de una ubicación específica. Aquí está cómo se utiliza:

```
umount [opciones] punto_de_montaje
```

- **punto_de_montaje** es el directorio donde se montó el sistema de archivos y que deseas desmontar.

Por ejemplo, si deseas desmontar la partición que acabamos de montar en **/mnt/nueva_particion**, puedes hacerlo así:

```
umount /mnt/nueva_particion
```

3. Opciones comunes:

- **-o** (options): Permite especificar opciones adicionales para el montaje o desmontaje, como opciones de montaje específicas del sistema de archivos.
- **-t** (type): Permite especificar el tipo de sistema de archivos que se está montando o desmontando.

4. Ejemplos:

- Montar una unidad USB en **/mnt/usb**:

```
mount /dev/sdb1 /mnt/usb
```

- Desmontar la unidad USB de **/mnt/usb**:

```
umount /mnt/usb
```

- Desmontar la carpeta compartida de red NFS:

```
umount /mnt/nfs
```

crontab y at

Crontab

Crontab es un archivo de configuración en sistemas operativos basados en Unix, como Linux, situado en la ruta `/etc/crontab` y que se utiliza para programar tareas periódicas para que se ejecuten automáticamente en momentos específicos. La estructura básica de un archivo crontab consta de siete campos y es la siguiente:

```
30 2 * * * (usuario) /ruta/al/script/backup.sh
```

- **Minuto:** El minuto en que se ejecutará la tarea, especificado como un número de 0 a 59.
- **Hora:** La hora en que se ejecutará la tarea, especificada como un número de 0 a 23 (formato de 24 horas).
- **DíaDelMes:** El día del mes en que se ejecutará la tarea, especificado como un número de 1 a 31.
- **Mes:** El mes en que se ejecutará la tarea, especificado como un número de 1 a 12 o como los nombres abreviados en inglés (jan, feb, mar, etc.).
- **DíaDeLaSemana:** El día de la semana en que se ejecutará la tarea, especificado como un número de 0 a 7 donde 0 y 7 representan domingo, o como los nombres abreviados en inglés (sun, mon, tue, etc.).
- **Usuario:** El usuario que se pretende que ejecute el comando o ejecutable.
- **Comando:** El comando que se ejecutará.

En la ruta `/etc/crontab` tenemos la siguiente información.

```
# Example of job definition:
# .----- minute (0 - 59)
# | .----- hour (0 - 23)
# | | .----- day of month (1 - 31)
# | | | .----- month (1 - 12) OR jan,feb,mar,apr ...
# | | | | .---- day of week (0 - 6) (Sunday=0 or 7) OR
sun,mon,tue,wed,thu,fri,sat
# | | | | |
# * * * * * user-name command to be executed

17 * * * * root    cd / && run-parts --report /etc/cron.hourly

25 6 * * * root    test -x /usr/sbin/anacron || ( cd / && run-parts --report
/etc/cron.daily )
```

```
47 6    * * 7    root    test -x /usr/sbin/anacron || ( cd / && run-parts --report
/etc/cron.weekly )

52 6    1 * *    root    test -x /usr/sbin/anacron || ( cd / && run-parts --report
/etc/cron.monthly )

50 13 17 04 4 si touch /home/si/contab.txt
```

Accedemos al archivo donde introducimos las reglas `/var/spool/cron/crontabs/usuario` gracias al comando `crontab -e` y podemos listar sus reglas con `crontab -l`.

```
# m h dom mon dow  command
30 17 10 5 0 touch /tmp/hola.txt
```

También se pueden utilizar las notaciones:

- `@yearly` / `@annually`: Es equivalente a poner en el crontab:

```
0 0 1 1 * root /bin/bash /tmp/ejecutable.sh
```

- `@monthly`: Es equivalente a poner en el crontab:

```
0 0 1 * * root /bin/bash /tmp/ejecutable.sh
```

- `@weekly`: Es equivalente a poner en el crontab:

```
0 0 * * 0 root /bin/bash /tmp/ejecutable.sh
```

Nota: Fíjate que significa todos los domingos y podría ser un 7 en el último carácter.

- `@daily` / `@midnight`: Es equivalente a poner en el crontab:

```
0 0 * * * root /bin/bash /tmp/ejecutable.sh
```

- `@hourly`: Es equivalente a poner en el crontab:

```
0 * * * * root /bin/bash /tmp/ejecutable.sh
```


at

El comando **at** en Linux es una herramienta que te permite ejecutar comandos o scripts en un momento específico en el futuro.

1. **Programación de tareas:** Puedes usar el comando **at** seguido de la hora en la que deseas que se ejecute el comando.
2. **Ejecución de comandos:** Después de especificar el tiempo, puedes escribir los comandos que deseas ejecutar en ese momento. Puedes escribir múltiples comandos, uno por línea.
3. **Envío de comandos:** Una vez que hayas escrito los comandos que deseas ejecutar, presionas **Ctrl+D** para finalizar la entrada. En este punto, el sistema registrará estos comandos para ejecutarlos en el tiempo especificado.
4. **Confirmación y salida:** Una vez que hayas enviado los comandos, el sistema te mostrará un mensaje que confirma la programación de las tareas.
5. **Ejecución de tareas:** Cuando llegue el momento especificado, el sistema ejecutará los comandos que programaste con **at**.

Aquí tienes un ejemplo de cómo programar un comando para ejecutarse a una hora específica:

```
si@si-VirtualBox:~$ at now + 2 minute
warning: commands will be executed using /bin/sh
at Wed Apr 17 14:22:00 2024
at> touch /home/si/at.txt
at> <EOT>
job 8 at Wed Apr 17 14:22:00 2024

si@si-VirtualBox:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  snap  Templates  Videos

si@si-VirtualBox:~$ ls
at.txt  Desktop  Documents  Downloads  Music  Pictures  Public  snap  Templates  Videos
```

Luego, puedes escribir los comandos que deseas ejecutar y presionar **Ctrl+D** para enviarlos.

Por otra parte, el parámetro **-f** de **at** se utiliza para especificar un archivo que contiene los comandos que deseas ejecutar en lugar de escribir los comandos directamente en la línea de comandos. Esto puede ser útil cuando tienes una serie de comandos complejos o largos que deseas ejecutar en un momento específico.

La sintaxis básica de **at** con el parámetro **-f** es la siguiente:

```
at [hora] -f archivo.sh
```

- **[hora]** es el momento en el que deseas que se ejecuten los comandos.

- **archivo** es el nombre del archivo que contiene los comandos que deseas ejecutar.

```
at 10:00 PM -f script.txt
```

declare, typeset y readonly

declare e **typeset** son sinónimos. Permiten declarar variables.

```
si@si-VirtualBox:~$ declare var1='AAA'
si@si-VirtualBox:~$ var2='BBB'
si@si-VirtualBox:~$ declare | grep var[12]
var1=AAA
var2=BBB

si@si-VirtualBox:~$ var1='CCC'
si@si-VirtualBox:~$ declare -p | grep var[12]
declare -- var1="CCC"
declare -- var2="BBB"
```

- **declare -r** no permite volver a redeclarar una variable. En este caso tanto **declare -r** como **readonly** para declarar variables de solo lectura funcionan del mismo modo.

```
si@si-VirtualBox:~$ declare -r var3='CCC'
si@si-VirtualBox:~$ declare -p | grep var3
declare -- _="var3=CCC"
declare -r var3="CCC"
si@si-VirtualBox:~$ var3='NO PUEDO DECLARAR'
bash: var3: readonly variable
```

```
si@si-VirtualBox:~$ readonly var3='No puedo declarar de nuevo'
bash: var3: readonly variable
si@si-VirtualBox:~$ readonly var4='No puedo declarar de nuevo'
si@si-VirtualBox:~$ declare -p | grep var4
declare -- _="var4=No puedo declarar de nuevo"
declare -r var4="No puedo declarar de nuevo"
```

- **declare -p** permite ver todas las variables del sistema junto con las nuevas variables creadas en la shell.

set ,unset y env

El comando **set** en Linux, sin opciones, muestra los nombres y valores de cada variable de la shell en un formato que puede ser reutilizado como entrada. Con la opción "-o" o "+o", permite establecer o quitar respectivamente atributos de la shell.

```
$ set -o # Muestra todas las opciones activadas (on) y desactivadas (off) en la shell
$ set +o # Muestra todas las opciones activadas (-o) y desactivadas (+o) en la shell, con el formato de las órdenes utilizadas para lograr ese resultado
$ set -o noclobber # Con esta opción, la shell bash no puede sobrescribir un archivo existente con los operadores de redirección (>, >&, <>). Añade la opción noclobber a la variable SHELLOPTS, siendo esta variable una lista de elementos separados por dos puntos, de opciones activas de la shell.
$ set +o noclobber # Desactiva el comando anterior, es decir, se activa la posibilidad de sobrescribir archivos con los operadores de redirección
$ set -C # Equivale a set -o noclobber
$ set +C # Equivale a set +o noclobber
```

unset elimina variables de la shell. No puede eliminar variables de solo lectura (definidas con "readonly" o "declare -r").

```
si@si-VirtualBox:~$ declare papelera='AAA'
si@si-VirtualBox:~$ echo $papelera
AAA
si@si-VirtualBox:~$ unset papelera
si@si-VirtualBox:~$ echo $papelera
```

El comando **env** ejecuta un programa con un entorno modificado según los parámetros con los que se ejecute. Esto significa que ejecuta un programa definiendo qué variables de entorno reconoce. Sin opciones o nombre de programa, el comando muestra el entorno resultante (las variables globales del entorno), similar al comando **printenv**.

- **env bash**: Este comando ejecuta el shell de bash con el entorno actual, es decir, utiliza las variables de entorno existentes tal como están en ese momento.
- **env variable=JEJEJE bash**: Aquí se ejecuta el shell de bash con una variable de entorno llamada **variable** establecida en el valor "JEJEJE". Esto significa que al abrir el shell, la variable **variable** tendrá el valor "JEJEJE".
- **env -i bash**: Este comando ejecuta el shell de bash con un entorno vacío. La opción **-i** indica "ignorar el entorno existente", por lo que no se pasan variables de entorno al nuevo shell bash, dejándolo con un entorno limpio.

set -> todas las variables env -> variables de un entorno

```
nuevo@si-VirtualBox:~$ env | wc -l
32
```

```
nuevo@si-VirtualBox:~$ set | wc -l
2136
```

```
si@si-VirtualBox:~$ env | wc -l
54
```

```
si@si-VirtualBox:~$ set | wc -l
2161
```

source, export y env

`source` y el carácter punto `.` son sinónimos. Leen y ejecutan los comandos existentes en un archivo dado en el entorno actual de la shell. El archivo no necesita ser ejecutable y se busca primero en las rutas del PATH y luego en la ruta actual (`pwd`) de ejecución del comando.

```
source $HOME/.bashrc # Recarga el archivo $HOME/.bashrc en la shell actual
```

El comando `export` permite exportar variables al entorno actual del shell, de manera que una vez exportadas, son válidas tanto en el entorno actual del shell como en cualquier subshell.

```
export HHH='hes' # Declara una variable llamada HHH con el valor 'hes' y además
exporta la variable para que pueda ser reconocida en el entorno actual de la
shell.
```

Comillas en variables de bash

Lo que estás viendo es un ejemplo de cómo se comportan las variables en bash (o cualquier shell de Unix) con diferentes tipos de comillas y la expansión de comandos.

1. Aquí estás asignando el valor "ls" a la variable "a".

```
si@si-VirtualBox:~$ a=ls
```

2. Al imprimir el valor de la variable "a" con `$a`, bash expande la variable y ejecuta el comando "ls", mostrando el contenido del directorio actual.

```
si@si-VirtualBox:~$ echo $a
ls
```

3. Al imprimir '\$a' entre comillas simples, bash toma el texto literalmente y no expande la variable "a", por lo que muestra "\$a".

```
si@si-VirtualBox:~$ echo '$a'
$a
```

4. Al imprimir "\$a" entre comillas dobles, bash expande la variable "a" y ejecuta el comando "ls", mostrando el contenido del directorio actual.

```
si@si-VirtualBox:~$ echo "$a"
ls
```

5. Aquí estás intentando ejecutar el comando "ls" usando la expansión de comandos, pero la forma en que está escrito no es correcta. Esto ejecuta el resultado de "ls" como un comando independiente, lo que significa que si hay archivos o directorios en el directorio actual, intentará ejecutarlos como comandos y puede generar errores si no son ejecutables.

```
si@si-VirtualBox:~$ echo `ls`
Desktop Documents Downloads env-exemplo1.sh Music Pictures Public script.sh snap
Templates Videos
```

Uso de {variable}

Definición de variables.

```
si@si-VirtualBox:~$ a=1
si@si-VirtualBox:~$ b=2
si@si-VirtualBox:~$ c=3
```

1. `echo $a-$b-$c`: Aquí, bash expande las variables y las une con el guión (-) como se indica en la cadena de formato. El resultado es "1-2-3".

```
si@si-VirtualBox:~$ echo $a-$b-$c
1-2-3
```

2. `echo ${a}_${b}_${c}`: Al utilizar las llaves {} para delimitar las variables, bash entiende claramente que la variable es "a", "b" y "c", mientras que el guión bajo (_) se utiliza como un literal y se concatena entre ellas. El resultado es "1_2_3".

```
si@si-VirtualBox:~$ echo ${a}_${b}_${c}
1_2_3
```

3. `echo $a_$b_$c`: En este caso, bash interpreta la expresión `$a_` como una variable llamada "a*" y expande su valor, que es "3" según lo definido anteriormente, y las demás variables y guiones bajos son ignorados. Entonces, solo imprime el valor de "a*" que es "3". Es importante destacar que el guión bajo no se considera un separador de nombres de variables en bash, por lo que "a_" se interpreta como una variable diferente a "a".

```
si@si-VirtualBox:~$ echo $a_$b_$c
3
```

Nota: Siempre es preferible hacer uso de `{}` para invocar a variables.

Ejecución de scripts

Shebang: También conocido como hashbang o sha-bang, es una convención en sistemas operativos tipo Unix que se utiliza en scripts para indicar qué intérprete de comandos debe ser utilizado para ejecutar el script. El shebang consiste en los caracteres `"#!"` seguidos de la ruta al intérprete. Por ejemplo:

1. En scripts bash:

```
#!/bin/bash
```

2. En scripts python:

```
#!/usr/bin/env python3
#!/usr/bin/env python
```

1. `chmod -x env.sh && bash env.sh`

Si ejecutamos `bash env.sh`, no es necesario tener permisos de ejecución en el script y estamos **ejecutando el script en una subshell**, por lo que al finalizar el script se elimina la subshell.

2. `chmod +x env.sh && ./env.sh`

Si el shebang es `#!/bin/bash` y lo ejecutamos mediante `./env.sh`, siempre y cuando el script tenga permisos de ejecución, estamos **ejecutando el script en una subshell**, por lo que al finalizar el script se elimina la subshell. Es análogo a la ejecución mediante el comando `bash`.

3. `chmod -x env.sh && . ./env.sh`

Si ejecutamos mediante `./env.sh` o `source ./env.sh`, no es necesario tener permisos de ejecución y estamos **ejecutando el script en la shell actual**.

Es fundamental comprender de que forma se ejecutan nuestros scripts para poder comprender si van a modificar aspectos de nuestro entorno o no. En el siguiente ejemplo podemos apreciar como el nivel de shell que es diferente en función de como lanzamos nuestro script.

Script **env-ejemplo1.sh**

```
#!/bin/bash
env | sort | grep -v '^_' | tee env1.txt
```

Si ejecutamos con las opciones 1 o 2, es decir, con **bash** o con **./** se ejecutará el script en una subshell por lo que no afectará a mi entorno.

Podemos ver que al lanzarlo con **bash** (o con **./**) estamos en el nivel de shell 1.

```
si@si-VirtualBox:~$ bash env-ejemplo1.sh
COLORTERM=truecolor
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
DESKTOP_SESSION=ubuntu
DISPLAY=:0
GDMSESSION=ubuntu
...

si@si-VirtualBox:~$ cat env1.txt | grep SHLVL
SHLVL=1
```

En este caso, ejecutamos directamente el comando y podemos ver que estamos en el nivel de shell 0, es decir, el mismo nivel de shell donde lanzamos comandos. Esto quiere decir que las modificaciones de este comando si podrían afectar a mi entorno a diferencia del caso anterior.

```
si@si-VirtualBox:~$ env | sort | grep -v '^_' | tee env2.txt
COLORTERM=truecolor
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
DESKTOP_SESSION=ubuntu
DISPLAY=:0
GDMSESSION=ubuntu
...

si@si-VirtualBox:~$ cat env2.txt | grep SHLVL
SHLVL=0
```

```
si@si-VirtualBox:~$ source env-ejemplo1.sh
COLORTERM=truecolor
```

```
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
DESKTOP_SESSION=ubuntu
DISPLAY=:0
GDMSESSION=ubuntu
...

si@si-VirtualBox:~$ cat env1.txt | grep SHLVL
SHLVL=0
```

Este es el motivo por el cual cuando queremos modificar nuestro entorno se hace uso de ficheros como `.bashrc` y este se lanza con `source`. El objetivo es hacer una modificación de nuestro entorno.

Cargamos `.bashrc` con `.` por lo tanto al ser lo mismo que `source` se convierten en variables de entorno las variables definidas dentro a las que se le aplica un `export`.

```
si@si-VirtualBox:~$ cat .profile | grep ".bashrc"
# include .bashrc if it exists
if [ -f "$HOME/.bashrc" ]; then
. "$HOME/.bashrc"
```

subshells

Si colocamos una secuencia de órdenes entre paréntesis, forzamos a que estos comandos se ejecuten en una subshell.

```
$ (readonly AAA='aes'; echo $AAA; unset AAA; echo $AAA); AAA='bes'; echo $AAA;
unset AAA;
echo $AAA # En la subshell definida con paréntesis y en la shell, las variables
AAA son variables distintas.
```

arrays en bash

declarar arrays

Para declarar un array podemos hacerlo de varias maneras.

- `declare -a`: Utilizado para declarar una variable como un array en Bash. Esto significa que la variable podrá contener múltiples valores, accesibles a través de índices numéricos. Esta opción es útil para asegurar que una variable se comporte como un array, incluso si no contiene valores al principio.

```
si@si-VirtualBox:~$ vectorUno[0]=nuevo

si@si-VirtualBox:~$ declare -p | tail -3
declare -- snap_bin_path="/snap/bin"
```



```
declare -- snap_xdg_path="/var/lib/snapd/desktop"
declare -a vectorUno=([0]="nuevo")

si@si-VirtualBox:~$ vectorUno[1]=nuevo2
si@si-VirtualBox:~$ vectorUno[3]=3

si@si-VirtualBox:~$ declare -p | tail -3
declare -- snap_bin_path="/snap/bin"
declare -- snap_xdg_path="/var/lib/snapd/desktop"
declare -a vectorUno=([0]="nuevo" [1]="nuevo2" [3]="3")
```

```
si@si-VirtualBox:~$ vectorDos=(uno dos tres)
si@si-VirtualBox:~$ declare -p vectorDos
declare -a vectorDos=([0]="uno" [1]="dos" [2]="tres")
```

```
si@si-VirtualBox:~$ vectorTres=([0]=Uno [1]=Dos [4]=Cuatro)
si@si-VirtualBox:~$ declare -p vectorTres
declare -a vectorTres=([0]="Uno" [1]="Dos" [4]="Cuatro")
```

```
si@si-VirtualBox:~$ declare -a vectorCuatro[0]=Hola
si@si-VirtualBox:~$ echo ${vectorCuatro[0]}
Hola
```

- `declare -p`: Esta opción muestra información sobre una o más variables, incluyendo su tipo y valor. Es útil para depurar y entender el estado de las variables en un script de Bash.

```
si@si-VirtualBox:~$ declare -p vectorUno
declare -a vectorUno=([0]="nuevo" [1]="nuevo2" [3]="3")
```

- `local` es una palabra clave que se utiliza dentro de funciones para declarar variables locales. Esto significa que las variables definidas como locales solo son visibles y accesibles dentro de la función donde se declaran, y no afectan a variables con el mismo nombre fuera de la función. Por ejemplo:

```
mi_funcion() {
    local variable_local="Este es un valor local"
    echo "Dentro de la función: $variable_local"
}

mi_funcion
echo "Fuera de la función: $variable_local" # Esto mostrará un valor vacío o
error, ya que $variable_local no está definido fuera de la función
```

- `local -a` es una variante de `local` que se utiliza para declarar variables locales como arrays. Esto asegura que la variable declarada sea tratada como un array dentro de la función. Por ejemplo:

```
mi_funcion() {
    local -a array_local
    array_local=("elemento1" "elemento2" "elemento3")
    echo "Dentro de la función: ${array_local[@]}"
}

mi_funcion
echo "Fuera de la función: ${array_local[@]}" # Esto mostrará un valor vacío o
error, ya que $array_local no está definido fuera de la función
```

Referenciar Arrays

- `${nombre[índice]}` donde nombre es el nombre del array e índice es la posición entera que contiene su valor correspondiente.
- `${nombre[@]}` donde nombre es el nombre del array y el índice @ representa todos los elementos del vector. Devuelve una cadena con los elementos separados por espacio.
- `${nombre[*]}` donde nombre es el nombre del array y el índice * representa todos los elementos del vector. Devuelve una cadena con los elementos separados por espacio.
- `"${nombre[@]}"` donde nombre es el nombre del array y el índice @ representa todos los elementos del vector. Devuelve una cadena con los elementos separados por espacio.
- `"${nombre[*]}"` donde nombre es el nombre del array y el índice * representa todos los elementos del vector. Devuelve una cadena con los elementos separados por el primer caracter de la variable separador de campos IFS.
- `${nombre}` equivale a `${nombre[0]}`

```
$ nombre=(primero segundo tercero)
$ declare -p nombre
declare -a nombre=([0]="primero" [1]="segundo" [2]="tercero")

$ echo ${nombre[*]}
primero segundo tercero
$ echo "${nombre[*]}"
primero segundo tercero

$ set | grep ^IFS
IFS=$' \t\n'
$ IFS=$'Z\t\n'
$ set | grep IFS
IFS=$'Z\t\n'

$ echo ${nombre[*]}
primero segundo tercero
$ echo "${nombre[*]}"
primeroZsegundoZtercero
```

Es decir, a "\${nombre[*]}" Le afecta el primer caracter de la variable separador de campos IFS.

Eliminar arrays

- `unset nombre`: Elimina el array declarado como `nombre` siempre que no esté declarado en modo lectura.
- `unset nombre[índice]`: Elimina el índice del array `nombre` en la posición [índice] siempre que no esté declarado en modo lectura.

Recorrer arrays

- Recorre los índices del array declarado como `nombre` y muestra cada valor en pantalla. Siempre utiliza comillas dobles para evitar errores de separadores como espacios que puedan existir en los valores de los índices.

```
for i in "${nombre[*]}"
do
echo $i
done
```

- Recorre los índices del array declarado como `nombre` y muestra cada valor en pantalla. Siempre utiliza comillas dobles para evitar errores de separadores como espacios que puedan existir en los valores de los índices.

```
for i in "${nombre[@]}"
do
echo $i
done
```

Longitud de los arrays e índices

```
echo ${#nombre[@]}
```

- Muestra el número de elementos existentes en el array declarado como `nombre`, es decir, proporciona el número de posiciones ocupadas en el array.

```
echo ${!nombre[@]}
```

- Muestra los índices de los elementos que no son NULOS en el array declarado como `nombre`.

```
echo ${#nombre[índice]}
```

- Muestra el número de caracteres de longitud del índice `índice` para el array declarado como `nombre`.

```
si@si-VirtualBox:~$ vector3=(uno dos tres)
si@si-VirtualBox:~$ echo ${vector3[*]}
uno dos tres
si@si-VirtualBox:~$ echo ${vector3[@]}
uno dos tres
si@si-VirtualBox:~$ echo ${vector3[0]}
uno
si@si-VirtualBox:~$ echo ${vector3}
uno
si@si-VirtualBox:~$ echo ${vector3[1]}
dos
si@si-VirtualBox:~$ echo ${vector3[2]}
tres
si@si-VirtualBox:~$ echo ${vector3[3]}

si@si-VirtualBox:~$ echo ${#vector3[@]}
3
si@si-VirtualBox:~$ echo ${!vector3[@]}
0 1 2
si@si-VirtualBox:~$ echo ${#vector3[0]}
3
si@si-VirtualBox:~$ echo ${#vector3[1]}
3
si@si-VirtualBox:~$ echo ${#vector3[2]}
4
```

Arrays como parámetros de funciones

En la llamada a la función utilizamos como parámetro la notación "nombre[@]", y dentro del cuerpo de la función utilizamos la referencia indirecta `${!número}`, donde `número` es el número del parámetro en la llamada a la función: 1, 2, 3...

```
f_nombre() { # Definición de la función llamada f_nombre
    parametros=("${!1}") # Referencia indirecta del parámetro $1 dentro de la
    definición del array parametros
    echo "${parametros[@]}" # Muestra todos los valores contenidos en los índices
    del array llamado parametros
} # Fin del cuerpo de la función f_nombre

nombres=(Anxo Brais) # Se declara la variable array nombres con los valores Anxo y
Brais en los índices 0 y 1 respectivamente.
f_nombre "nombres[@]" # Llamada a la ejecución de la función llamada f_nombre
donde el primer parámetro $1 toma el valor "nombres[@]", lo cual es equivalente a
los valores de todos los índices del array llamado nombres
```

Prácticas sobre arrays

Declarar y eliminar arrays

Muestra el valor de la variable llamada "curso" y cómo está declarada en caso de existir.

```
declare -p curso
```

Crea automáticamente el array llamado "curso" donde en el índice 0 toma el valor '2015-2016'.

```
curso[0]='2015-2016'
```

Genera un nuevo array llamado "curso" donde los índices 0 y 1 toman los valores '2017-2018' y '2018-2019' respectivamente.

```
curso=( '2017-2018' '2018-2019' )
```

Aunque no existe el índice 2 en el array llamado "curso", la sintaxis del comando genera un nuevo array con solo un índice: el índice 2 que toma el valor '2019-2020'.

```
curso=( [2]='2019-2020' )
```

Genera un nuevo array llamado "curso" donde solo se hacen referencia en la declaración a los índices 0 y 5, por lo que el resto de los elementos del array irán guardando sus valores a partir del índice 5. Así, los índices 0, 5, 6 y 7 toman los valores '1111-1112', '1112-1113', '1113-1114' y '1114-1115' respectivamente.

```
curso=( [0]='1111-1112' [5]='1112-1113' '1113-1114' '1114-1115' )
```

Muestra el valor de la variable array llamada "curso" y cómo está declarada.

```
declare -p curso
```

Declara la variable array llamada "curso" en modo solo lectura.

```
declare -r curso
```

No puede eliminar la variable "curso" ya que está declarada en modo solo lectura.

```
unset curso
unset curso[@]
unset curso[*]
```

No puede eliminar el elemento de la posición 1 de la variable "curso" ya que está declarada en modo solo lectura.

```
unset curso[1]
```

Nota: No es lo mismo pedir por teclado unos datos para el array que pedir los datos para un índice concreto del array.

1. Pedir datos para un array

Lee desde la entrada estándar (teclado) una lista de palabras como índices en un array llamado "curso5", comenzando desde 0.

Los campos separadores de palabras, que serán recogidos en los índices del array, están determinados por el valor de la variable separador de campos \$IFS.

```
read -p 'Escriba valores del array en una lista: ' -a curso5
```

Muestra el mensaje 'Escriba valores del array en una lista: ' y recoge desde la entrada estándar (teclado) la lista de palabras como índices en un array llamado curso5, empezando por 0. Los campos separadores de palabras, que serán recogidos en los índices del array, están determinados por el valor de la variable separador de campos \$IFS.

```
1 2 3
```

Como el valor de IFS='\$' '\r\n' y el número de palabras de la lista insertada es 3, se sustituyen los valores de los índices 0, 1 y 2 por estos nuevos: 1, 2 y 3 respectivamente.

2. Pedir datos para una posición del array

Recoge por teclado el valor del índice 1 del array llamado "curso5".

```
read -p 'Escriba valor para el índice 1: ' curso5[1]
```

Asigna el valor '4 5 6' al índice 1, ya que en la asignación al índice no se tiene en cuenta la variable IFS.

4 5 6