Unidad 5

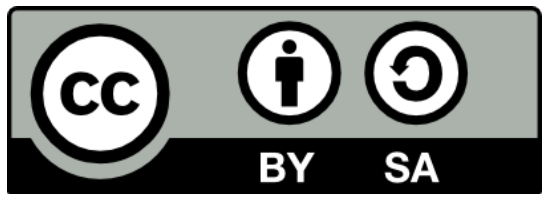
Gestión remota GNU/Linux

Implantación y administración remota y centralizada
de Sistemas Operativos

Manuel Morán Vaquero

mmv@edu.xunta.es

http://www.immv.es



Contenidos

Licencia de uso y renuncia de responsabilidad

Este documento tiene licencia Creative Commons BY–SA

http://creativecommons.org/licenses/by-sa/3.0/es/

- Usted es libre de
  - **copiar, distribuir y comunicar públicamente la obra**
  - **hacer obras derivadas**
- Bajo las condiciones siguientes
  - **Reconocimiento:** Debe reconocer los créditos de la obra de la manera especificada por el autor o el licenciador (pero no de una manera que sugiera que tiene su apoyo o apoyan el uso que hace de su obra)
  - **Compartir bajo la misma licencia** Si transforma o modifica esta obra para crear una obra derivada, sólo puede distribuir la obra resultante bajo la misma licencia

Disclaimer (Renuncia de responsabilidad)

Este manual es una guía de instalación. Se realizan procedimientos que pueden conllevar, entre otros, a pérdidas de datos, agujeros informáticos, etc. . . El autor no será responsable de ningún daño o pérdida de datos que se produzca. ÚSELA A SU PROPIO RIESGO.

¿Qué problema queremos solucionar?

- Tenemos un sistema de autoinstalación de equipos GNU/Linux y sabemos que podemos administrar los clientes de forma sencilla y remota conectándonos a ellos vía ssh
- No obstante, en la mayoría de los casos desearemos administrar no uno sino **varios** clientes, por ejemplo en los siguientes casos
  - Instalar/desinstalar un paquete en todos los clientes
  - Añadir un parche (sobreescribir un archivo, borrar un archivo, buscar-reemplazar sobre un archivo, etc...) en todos los clientes
  - Actualizar un paquete en todos los clientes
  - Reiniciar / apagar un conjunto de clientes
- Aunque hagamos un script que ejecute esto en todos los clientes, nos encontraremos con un problema: los clientes pedirán autentificación uno a uno, por lo que el proceso será manual y por tanto no conseguiremos nuestro objetivo de hacer una administración remota en masa productiva
- Veamos esto con un ejemplo (no es necesario hacerlo, el objetivo es simplemente entender el problema al que vamos a dar solución)

Primera aproximación a la administración remota GNU/Linux

- Queremos instalar la aplicación `mc` en todos los equipos con IP comprendida entre el rango 172.30.0.150 y 172.30.0.154
- Tenemos las siguientes posibilidades

Posibilidad 1

- Loggearse remotamente en cada una de las máquinas con el comando

```
$ ssh root@172.30.0.150
```

- Y posteriormente ejecutar en la máquina cliente el comando

```
# yum -y install mc
```

- Saldríamos de la sesión en dicha máquina cliente y repetiríamos el proceso para cada máquina
- Esto tiene un claro inconveniente y es que es un proceso lento y tedioso

Primera aproximación a la administración remota GNU/Linux

Posibilidad 2

- No obstante el comando ssh nos permite pasarle un tercer parámetro que es el comando a ejecutarse en la máquina remota. De tal manera que no obtenemos una sesión interactiva en el equipo cliente sino simplemente el resultado de la ejecución del comando que se pasa como parámetro. Al finalizar dicha ejecución, nuestra shell sigue siendo la de nuestro equipo desde el que estamos administrando nuestros clientes

```
$ ssh root@172.30.0.150 "yum -y install mc"
```

- Tendrá como efecto la instalación inmediata del paquete mc, aunque con el inconveniente de que debemos repetir el comando anterior para cada PC de la subred esto lo solucionamos como se expone en el siguiente punto

Primera aproximación a la administración remota GNU/Linux

Posibilidad 3

- Podemos programar un bucle FOR que recorra los equipos cliente que deseemos

```
$ for I in 150 151 152 153 154
> do
> ssh root@172.30.0.$I "yum -y install mc"
> done
```

- De este modo el comando de instalación se ejecutará en todos los PCs del rango especificado

¡¡PERO OJO!!

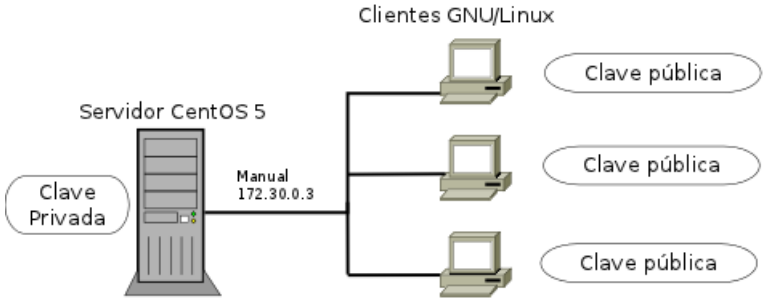
- Esto sigue siendo muy improductivo dado que tendremos que teclear la password de root cada vez que haya que loggearse en un cliente
- **Lo resolveremos haciendo uso de la autentificación publickey integrada en ssh**

Preparación del entorno de trabajo

- **IMPORTANTE: usaremos el servidor de la unidad anterior (UD4)**
- Instalaremos 3 nuevos clientes CentOS 5 (no usaremos el VDI patrón). Esto es muy sencillo gracias al sistema kickstart que hemos configurado en la unidad anterior
- A dichos clientes les asignaremos las IPs siguientes (vía static-lease en el DHCP o accediendo manualmente a su configuración una vez instalados)
  - 172.30.0.251 – 172.30.0.252 – 172.30.0.253
  - Puerta de enlace: 172.30.0.3, DNS: 8.8.8.8
- Antes de continuar es conveniente comprobar que los clientes pueden acceder a Internet con esta configuración IP

¿Qué queremos obtener?

- Deseamos que desde el servidor se puedan administrar los clientes sin que estos pidan password cuando el usuario root del servidor se conecta a los clientes
- Esto lo conseguiremos distribuyendo un par de claves pública/privada, quedando la privada en el servidor y la pública en los clientes



Creación del par de claves

- Abrimos una sesión de root en el servidor y tecleamos el siguiente comando
  ```
  # ssh-keygen -t dsa
  ```
- Nos pedirá algunos parámetros y los dejamos como están por defecto (pulsar ENTER)
- Lo que habremos obtenido es:
  - Un archivo `/root/.ssh/id_dsa` que es la clave privada y que no debe salir del servidor
  - Un archivo `/root/.ssh/id_dsa.pub` que es la clave pública y que copiaremos a los clientes en el siguiente paso
- Copiamos la clave pública a los clientes ejecutando en el servidor el comando
  ```
  # ssh-copy-id -i /root/.ssh/id_dsa.pub 172.30.0.251
  ```
- Repetir el comando anterior para los otros equpos (252 y 253)
- Y ahora probamos a loggearnos en los clientes remotos desde el servidor
  ```
  ssh # 172.30.0.251
  ```
  ```
  # ssh 172.30.0.252
  ```
  ```
  # ssh 172.30.0.253
  ```
- Si no pide password, ¡lo hemos conseguido!. Sal de la sesión de los equipos cliente con CTRL+D o `exit`
- A partir de ahora puedo ejecutar comandos remotos de forma inmediata y sencilla en los clientes. Prueba por ejemplo desde el servidor
  ```
  # ssh 172.30.0.251 "uptime"
  ```
- ¡¡OJO!! Observa que en este caso aún sigues con la shell en tu máquina cliente: la ejecución del anterior comando no ha producido que se haya abierto una sesión interactiva

Primeros comandos en masa

- Ahora que tenemos la posibilidad de ejecutar comandos remotos sin abrir una sesión interactiva y sin teclear password, veamos las posibilidades que esto nos abre de cara a la administración de equipos en masa
- Volvamos a probar ahora el bucle FOR de la anterior posibilidad 3, en este caso instalaremos Thunderbird en todos nuestros clientes

Bucle FOR que instala Thunderbird en los clientes

```
# for I in 251 252 253
> do
> ssh root@172.30.0.$I "yum -y install thunderbird"
> done
```

- Entra como usuario en un equipo cliente y comprueba que, efectivamente, se ha instalado Thunderbird (Menú Inicio - Internet)
- No obstante este FOR genera 3 comandos ssh de forma secuencial y por tanto hay que esperar a que cada uno de los equipos cliente hayan instalado el Thunderbird para pasar a ejecutar el comando en el siguiente equipo

Primeros comandos en masa

- El procedimiento anterior ralentiza la operación de forma innecesaria. Bastará con lanzar el ssh en segundo plano añadiendo simplemente un & al final del comando. Probemos esto desinstalando Thunderbird y comprobaremos que la operación se ejecuta de forma concurrente en todos los clientes de tal manera que evoluciona muy rápido.

Bucle FOR que desinstala Thunderbird en los clientes

```
# for I in 251 252 253
> do
> ssh root@172.30.0.$I "yum -y remove thunderbird < /dev/null" &
> done
```

- Como usuario desde un equipo cliente comprueba que, efectivamente, se ha desinstalado Thunderbird
- **La grandes posibilidades de administración remota de los sistemas UNIX, vienen dadas en gran medida por poseer unas herramientas potentísimas de scripting que además vienen instaladas por defecto y están listas para su uso en cualquier equipo. ¡Vamos a aprovecharlas!**

Ejecución en un gran número de clientes

- ¡¡OJO!! Aunque en nuestros ejemplos ejecutaremos los comandos únicamente en 3 equipos a modo de ejemplo, es muy sencillo extender esto a un grupo de equipos. En caso de querer ejecutar los comandos anteriores en un rango de 100 equipos (p.ej del 51 al 150) bastaría con usar una expansión del comando `seq` del siguiente modo:

Ejecución en 100 clientes

```
# for I in $(seq 51 150)
> do
> ssh root@172.30.0.$I "yum -y remove thunderbird < /dev/null" &
> done
```

- Del mismo modo,para ejecutarlo en 100 clientes de 10 subredes:

Ejecución en 100 clientes

```
# for SUBRED in $(seq 0 9)
> do
> for EQUIPO in $(seq 51 150)
> do
> ssh root@172.30.$SUBRED.$EQUIPO "yum -y remove thunderbird < /dev/null" &
> done
> done
```

Uniendo comandos: los scripts

Recomendado: consultar el Anexo

Esta documentación consta de un anexo rápido para refrescar la memoria en cuanto a los detalles más útiles de bash scripting en la administración de equipos. Puedes echarle un vistazo ahora antes de empezar con la creación del primer script (sobre todo la creación y lanzamiento de scripts)

- Programemos un primer script que se conecte a nuestros clientes y nos muestre el espacio de disco ocupado en los clientes

`ocupado-clientes.sh`

```
#!/bin/bash

for I in 251 252 253
do
echo Espacio ocupado en cliente $I
ssh root@172.30.0.$I "df /"
done
```

- Bastará con arrancar `./ocupado-clientes.sh` para obtener dicha información

Lanzando scripts en máquina local

- Para evitar tener que editar el script cada vez que queramos ejecutar un comando distinto, podemos hacer un script que ejecute un comando determinado en todos los clientes. Este script será muy útil y nos permitirá ejecutar en masa cualquier comando que deseemos de forma inmediata.

exec-todos.sh

```
#!/bin/bash

for I in 251 252 253
do

IP=172.30.0.$I

echo Ejecutando el comando en el equipo $IP
ssh root@$IP $@

done
```

Usando exec-todos.sh

- Ahora puedo ejecutar estos útiles comandos de administración de forma muy rápida:
  1. Para ver la versión del kernel de cada uno de los equipos
     ```
     ./exec-todos.sh uname -a
     ```
  2. Para crear un archivo vacío en /tmp/prueba.txt en cada uno de los equipos
     ```
     ./exec-todos.sh touch /tmp/prueba.txt
     ```
  3. Para borrar /tmp/prueba.txt de cada uno de los equipos
     ```
     ./exec-todos.sh rm -fv /tmp/prueba.txt
     ```
  4. Instalar Thunderbird en todos los equipos
     ```
     ./exec-todos.sh yum -y install thunderbird
     ```
  5. Para apagar todos los equipos `./exec-todos.sh poweroff`
- Y así sucesivamente. Las capacidades de administración son, como puede comprobarse, casi ilimitadas

Mejorando exec-todos.sh

- Nuestro script exec-todos.sh aún tiene varias limitaciones como por ejemplo el hecho de que si quiero variar el rango de equipos en los que se ejecuta, tengo que editar el script. Eso se puede solucionar mejorándolo un poco
- Vamos a pedir dos parámetros más: el rango de equipos en el que se ejecutará el comando.

exec-todos-rango.sh - sin tildes deliberadamente

```
#!/bin/bash

# En primer lugar comprobamos si el número de parámetros introducido es 3
if [ $# != 3 ]
then
echo Uso: ./exec-todos-rango.sh primer_equipo ultimo_equipo comando_a_ejecutar

else
# En este caso haremos un seq entre los parametros pasados por linea de comandos
for I in $(seq $1 $2)
do
IP=172.30.0.$I
echo Ejecutando el comando en el equipo $IP
ssh root@$IP $3
done

fi
```

- Podríamos usar nuestro nuevo script con un comando de este estilo

```
./exec-todos-rango.sh 100 200 "uname -a"
```

Mejorando aún más exec-todos.sh

- Para hacerlo aún más cómodo, podemos incluso mejorar exec-todos haciendo que funcione en modo gráfico y que pida todos los datos anteriores por pantalla
- Para ello usaremos `zenity` que es una manera sencillísima de hacer frontends gráficos para los scripts
- Tendremos que instalar zenity con `yum -y install zenity`

`exec-todos-gui.sh` - sin tildes deliberadamente

```
#!/bin/bash

INICIO=$(zenity --entry --text="Introduce el equipo inicial del rango")
FIN=$(zenity --entry --text="Introduce el equipo final del rango")
COMANDO=$(zenity --entry --text="Introduce el comando a ejecutar")

for I in $(seq $INICIO $FIN)
do
IP=172.30.0.$I
echo Ejecutando el comando en el equipo $IP
ssh root@$IP $COMANDO
done
```

- Probemos entonces la versión gráfica de exec todos con `./exec-todos-gui.sh`

Lanzando scripts en máquina remota

- A veces lo que deseamos es iniciar scripts completos en la máquina remota, sin necesidad de copiarlos a ella
- Crea **en el servidor** el siguiente script de ejemplo

prueba.sh

```
echo Hola mundo
uptime
echo Adios mundo
```

- Podemos probar el script localmente ejecutando

```
./prueba.sh
```

- Pero si lo que deseamos es ejecutar el script en una máquina remota llamaremos a ssh de la siguiente manera:

```
ssh root@172.30.0.251 "bash -s" < ./prueba.sh
```

- De este modo podemos programar los scripts deseados en el servidor, y ejecutarlos en los clientes directamente

Incluyendo en el kickstart la clave pública del servidor

- La capacidad de administración remota en masa de todo un parque de equipos con este modelo es extremadamente útil, pero hemos tenido que insertar manualmente las claves una vez instalados los equipos vía kickstart
- Lo ideal sería que el propio kickstart copiase directamente la clave pública del servidor para que así tras instalar las máquinas clientes vía kickstart, ya fueran inmediatamente administrables de forma remota, lo cual sería un auténtico *sysadmin heaven*
- Esto no es nada difícil de conseguir: en primer lugar copiaremos la clave pública a un lugar de donde el instalador kickstart lo pueda obtener (p.ej. la raíz del servidor web)

```
cp /root/.ssh/id_dsa.pub /var/www/html
```

- Posteriormente añadamos a la sección %post del fichero ks.cfg las siguientes líneas

Sección %post de ks.cfg

```
mkdir /root/.ssh
wget -O /root/.ssh/authorized_keys http://172.30.0.3/id_dsa.pub
```

- Finalmente prueba a autoinstalar vía kickstart otro cliente y comprueba que te puedes conectar a él directamente. Sysadmin heaven!

Ejemplos de scripts útiles en administración remota

- El objetivo principal de esta sección es probar a ejecutar estos scripts en nuestros servidores o en máquinas remotas usando lo que hemos aprendido con ssh
- Se han elegido a modo de ejemplo un conjunto de scripts para realizar tareas que un administrador necesita habitualmente como
  - Creación de archivos y directorios
  - Parcheado de archivos (añadir líneas, sustituir líneas)
  - Hacer listados, informes de uso, etc...
  - Descomprimir archivos
  - Descargar archivos vía http
  - Arreglar permisos
- Lo importante será por tanto comprender el script para aprender vía ejemplos y saber extender estas ideas a nuestras necesidades futuras a la hora de administrar equipos en individuales o en masa
- En algunos casos también se muestran ejemplos de comandos individuales muy útiles en administración remota
- Comprueba el funcionamiento de los scripts (o comandos individuales) ejecutándolos en una máquina cliente de prueba (bien por ssh desde el servidor o bien localmente)

Ejemplos de scripts útiles en administración remota

Creación de 100 usuarios con una password por defecto

```
#!/bin/bash

for I in $(seq 1 100)
do

NOMBUSUARIO=usuario_$I

echo Creando usuario $NOMBUSUARIO
useradd $NOMBUSUARIO
echo 'abc12345' | passwd --stdin $NOMBUSUARIO

done
```

Ejemplos de scripts útiles en administración remota

Creación de un archivo de instrucciones en los escritorios de los anteriores usuarios

```
#!/bin/bash

for I in $(seq 1 100)
do

NOMBUSUARIO=usuario_$I
FICHEROINSTR=/home/$NOMBUSUARIO/Desktop/instrucciones.txt

echo Creando archivo para $NOMBUSUARIO

mkdir /home/$NOMBUSUARIO/Desktop

echo "Instrucciones de uso del sistema" > $FICHEROINSTR
echo "--------------------------------" >> $FICHEROINSTR
echo "bla bla bla..." >> $FICHEROINSTR

done
```

Ejemplos de scripts útiles en administración remota

Idem que el anterior, pero en una versión más limpia

```
#!/bin/bash

for I in $(seq 1 100)
do

NOMBUSUARIO=usuario_$I
FICHEROINSTR=/home/$NOMBUSUARIO/Desktop/instrucciones.txt

echo Creando archivo para $NOMBUSUARIO

mkdir /home/$NOMBUSUARIO/Desktop

cat << EOF > $FICHEROINSTR
Instrucciones de uso del sistema
--------------------------------
bla bla bla...
EOF

done
```

Ejemplos de scripts útiles en administración remota

Añadir una línea al /etc/fstab, para montar un nuevo disco

```
#!/bin/bash

mkdir /NUEVODISCO

echo "/dev/hdb1   /NUEVODISCO   auto  defaults 0 0" >> /etc/fstab
```

Parchear una línea de /etc/fstab (buscar/reemplazar usando sed)

```
sed -i 's/NUEVODISCO/NEWDISK/g' /etc/fstab
```

Ejemplos de scripts útiles en administración remota

Listar los 10 archivos que más ocupan en /usr/share/doc

```
ls -lR /usr/share/doc | sort -k 5 -n | tail -n 10
```

Listar los 10 directorios que más ocupan en /usr/share/doc

```
du /usr/share/doc | sort -n | tail -n 10
```

Bajar un archivo de un servidor web

```
wget http://172.30.0.3/ejemplo.tar.gz
```

Extraer *al vuelo* el anterior archivo (útil también para parchear)

```
wget -O - http://172.30.0.3/ejemplo.tar.gz | tar -xzvf -
```

Contar el número de veces que se ha loggeado el usuario root en la máquina

```
last | grep root | wc -l
```

Ejemplos de scripts útiles en administración remota

Arreglar los permisos de todos los usuarios en /home (poniendo modo 700 y uid:gid el propio usuario)

```
#!/bin/bash

for USUARIO in $(ls -1 /home)
do

echo Aplicando permisos a $USUARIO

chmod 700 /home/$USUARIO
chown $USUARIO:$USUARIO /home/$USUARIO

done
```

Ejemplos de scripts útiles en administración remota

Hacer una copia de seguridad del /home del servidor en el cliente remoto (hay que ejecutarlo en el servidor)

- Es útil conocer también que ssh es un comando que soporta todo tipo de redirecciones de salida y entrada estándar. Esto es extremadamente potente y nos permite hacer trabajos tan complicados a priori como este en un sencillo comando
- En el servidor se hace un tar de home, pero para evitar crear el archivo intermedio y luego copiarlo al cliente, se envía por salida estándar y ésta se redirige a la entrada estándar de un ssh que lo único que hace es copiar dicha entrada estándar (que es un archivo comprimido) de forma efectiva a un fichero

```
tar -czpvf - /home | ssh root@172.30.0.251 "dd of=backup.tar.gz"
```

- De esta forma se pueden hacer copias de seguridad remotas de forma muy sencilla

Anexo: Referencia rápida de bash scripting

Anexo: Referencia rápida de bash scripting

Creación y lanzamiento de scripts bash

Para hacer un script bash hay que seguir tres pasos principales

1. Escribir el script en un fichero de texto (p.ej. `script.sh`)
2. La primera línea de dicho fichero especifica el intérprete con el que se lanzará el script (bash, perl, tcsh...). En nuestro caso escribiremos en la primera línea del script lo siguiente
   `#!/bin/bash`
3. Darle permisos de ejecución al script por ejemplo con el siguiente comando: `$ chmod +x script.sh`
4. Ejecutaremos el script con el comando `$ ./script.sh`

Intenta ejecutar el siguiente script:

prueba.sh

```
#!/bin/bash

echo Hola mundo
uptime
echo Adios mundo
```

Un nuevo lenguaje de programación en 4 transparencias: Bash Scripting

Variables

- `VAR1=valor`
- `echo $VAR1` – la variable es expandida por la shell **ANTES** de la ejecución del comando

Operaciones numéricas (comando bc)

- `echo (2+8)/2 | bc`

Capturar la salida estándar de un comando en una variable

- RES=$(cualquiercomando)

Parsear (analizar) los parámetros de la línea de comandos

- `$1 , $2 , $3 , ...`
- `$@` – expande en todos los parámetros
- `$#` – expande en el número de parámetros de línea de comandos

Un nuevo lenguaje en 4 transparencias: Bash scripting

Comodines (REGEXPS)

- `*` sustituye cero o más ocurrencias
- `?` sustituye cero o una ocurrencias
- e.g. `ls /usr/bin/*edit*`
- **¡¡OJO!! LA SHELL ES LA QUE EXPANDE ANTES DE EJECUTAR LOS COMANDOS**, los comandos en general no saben de REGEXPS

El uso de las comillas

- **Comillas dobles** expande lo que hay dentro de ellas y pasan el literal completo como un único parámetro (aunque haya espacios)
  - `echo "El resultado es $VAR1"` → `El resultado es 3`
- **Comillas simples** no expanden nada
  - `echo 'El resultado es $VAR1'` → `El resultado es $VAR1`

prueba.sh

Se puede depurar un script lanzándolo así: `bash -x script.sh`

Bucles FOR

- for con una lista

```
for PLANETA in mercurio venus jupiter marte
do
echo $PLANETA
done
```

- for con REGEXP

```
for I in *.txt
do
echo $I
done
```

- for con números

```
for I in $(seq 1 10)
do
echo $I
done
```

while / if

- while

```
while [ "$VAR1" != "FIN" ]
do
...
done
```

- if

```
if [ "$VAR2" == "SI" ]
then
...
else
...
fi
```

- Hemos de respetar el espaciado exacto entre los corchetes, variables y condiciones de test
- Hay un montón de condiciones de test para estos comandos. Referencia completa en el manual de bash

Bash para el GUI

- `read -p "Estás seguro (S/N)? "`
- `zenity --info --text="Trabajo completado"`
- `zenity --entry --text="Introduce tu password"`
- `zenity --file-selection`
- `zenity --calendar`

stdin, stdout, stderr

- stdin es la entrada estándar
  - p.ej. el teclado (cuando se pide algo con un scanf o getchar)
- stdout es la salida estándar
  - e.g. la pantalla (cuando se escribe algo a pantalla con printf)
- stderr es la salida estándar de errores
  - p.ej. la pantalla (cuando un comando dice ”permiso denegado”)

Comandos útiles para administración de sistema usando pipes

- Con los pipes se puede redirigir la stdout de un comando directamente a la stdin del siguiente
- `sort`,`tail`,`head`,`uniq`,`grep`
- **awk**

Redirigir stdin, stdout y stderr

Puedes redirigir stdin, stdout y stderr del siguiente modo:

- `>` redirige stdout a un fichero (creándolo o reescribiéndolo)
- `>>` adjunta stdout a un fichero (creándolo si no existe)
- `|` redirige la stdout del comando a la izquierda hacia la stdin del comando a la derecha (tubería o pipe)

Redirigir stderr

Si quieres redirigir stderr (por ejemplo para loggear o descartar errores) tiens que usar el número 2 (identificador de stderr) antes de la redirección:

- `cualquiercomando 2>> fichero_de_registro`
- `comandodicharachero 2> /dev/null`