

ssh sustituye a rsh

Después de la r viene la s

Enrique Galdú

galdu@si.uji.es

Tabla de contenidos

Introducción.....	3
Deshabilitar los comandos "r"	3
Instalar y configurar SSH.....	4
Comandos básicos de ssh	5
Métodos de autenticación	8
Gestión de claves	9
Ejecución de comandos en modo no interactivo.....	11
Redirección de puertos	13
Uso del SSH en otras aplicaciones.....	14
Conclusión	16
Bibliografía	16

SSH (Secure SHell) es un reemplazo seguro para programas de acceso remoto y transmisión de archivos como **telnet**, **rsh**, **rnp** y **ftp**.

Mientras que los programas clásicos transmiten los datos en claro, *SSH* utiliza métodos de autenticación por clave pública para establecer una conexión encriptada segura entre el cliente y el servidor.

Las conexiones X11 y puertos TCP/IP también pueden ser redirigidos sobre la conexión segura. Esta redirección de puertos se puede aprovechar para hacer seguras otras aplicaciones.

Este documento se basa en `OpenSSH` que es la implementación del protocolo *SSH* de `OpenBSD`. `OpenSSH` está disponible para muchas de las distribuciones de Linux y UNIX.

También existen clientes de `ssh` para windows como Putty¹ o SecureCRT²

Los ejemplos que se incluyen en este documento se corresponden con la distribución Red Hat Linux release 9 (Shrike).

Introducción

En el uso diario de Linux es frecuente la ejecución de comandos en máquinas remotas. Tradicionalmente esta necesidad se resolvía con los comandos “r” esto es: **rlogin**, **rsh** y **rnp**.

Estos comandos lanzan un shell en la máquina remota y permiten al usuario ejecutar comandos. El usuario debe usar una cuenta en la máquina remota, por lo que debe pasar por los métodos de autenticación. Los comandos r usan la autenticación simple de usuario y contraseña, y utilizan una conexión en texto claro, por lo que estas pueden ser interceptadas por la red.

La autenticación para los comandos “r” también se puede controlar desde algunos archivos de configuración como son:

- `/etc/hosts.equiv`: a nivel de sistema, equivalencia entre usuarios de distintas máquinas. Se puede evitar la introducción de contraseñas.
- `$.HOME/.rhosts`: a nivel de usuario, permite el acceso a usuarios de otras máquinas sin utilizar contraseña.

Para poder utilizar aplicaciones gráficas al usar comandos “r” se necesita usar los comandos que controlan el acceso al servidor X. El comando **xhost** permite el acceso a un host, y **xauth** permite dar el acceso a un usuario determinado mediante el intercambio de cookies. Este es un ejemplo del comando que se ha de ejecutar para permitir el acceso a un usuario remoto:

```
$xauth extract - $DISPLAY | rsh otherhost xauth merge -
```

Estas funcionalidades se pueden obtener de una manera sencilla mediante *SSH*, con la ventaja de que utiliza criptografía de clave pública.

SSH ofrece la autenticación mediante usuario y contraseña, así como la autenticación mediante clave pública. Los datos están encriptados en la conexión con lo que se evita que las contraseñas o la información intercambiada pueda ser interceptada por otros usuarios.

Así que avanzamos de la r a la s, comenzando por deshabilitar los comandos “r”.

ssh sustituye a rsh

Deshabilitar los comandos “r”

Los comandos “r” se controlan desde el demonio **inet**, para deshabilitarlos puedes comentar sus entradas en el fichero `/etc/inetd.conf`. Si tu distribución utiliza **Xinetd** puedes comprobar si están deshabilitados con el siguiente comando:

```
[root@localhost root]# chkconfig --list|grep -E "rlogin|shell|exec"
    kshell: off
    rexec:  off
    rlogin: off
```

En nuestro caso ya están deshabilitados. Si alguno de ellos no lo estuviera puedes conseguirlo con el comando:

```
[root@localhost root]# chkconfig rlogin off
```

Instalar y configurar SSH

Puedes obtener el software de `OpenSSH` de <http://www.openssh.com/>³, allí encontrarás el fuente y versiones compiladas para diferentes sistemas operativos.

Lo mas probable es que tu distribución ya incluya este paquete. En el caso de RedHat los paquetes básicos son:

- **openssh**: contiene los ficheros básicos necesarios para el servidor y clientes ssh.
- **openssh-server**: contiene el demonio **sshd**, permite a los clientes ssh establecer conexiones seguras con el sistema.
- **openssh-clients**: contiene los clientes que permiten establecer conexiones con el servidor.

Aviso

necesitas tener instalado `OpenSSL`

El demonio sshd

El demonio **sshd** es el programa que espera conexiones de red de los clientes ssh, controla la autenticación y ejecuta el comando requerido. El puerto por defecto en el que escucha es el 22 y su fichero de configuración es `/etc/ssh/sshd_config`.

Otras opciones a destacar son:

- `X11Forwarding yes | no` : habilitar o deshabilitar la redirección X
- `PasswordAuthentication yes | no` : especifica si deseamos utilizar la autenticación básica

En esta configuración se indica también la ruta en la que encontrar las claves que identifican nuestro servidor. Estas son la base de la autenticación mediante clave pública y los valores por defecto son:

- `HostKey /etc/ssh/ssh_host_key`
- `HostKey /etc/ssh/ssh_host_rsa_key`
- `HostKey /etc/ssh/ssh_host_dsa_key`

Estas claves generales al sistema, junto con su correspondiente clave pública, se crean al instalar el servidor mediante el comando **ssh-keygen**.

Los clientes ssh

Los programas que permiten al usuario utilizar el protocolo *SSH* son **scp**, **sftp** y **ssh**. Se pueden configurar opciones generales al sistema en el fichero `/etc/ssh/ssh_config` como por ejemplo:

- `ForwardX11` yes|no : habilitar o deshabilitar la redirección X
- `PasswordAuthentication` yes|no : especifica si deseamos utilizar la autenticación básica en nuestros clientes.

En él se indican las rutas para obtener las claves públicas y privadas de cada usuario:

- `IdentityFile` `~/.ssh/identity`
- `IdentityFile` `~/.ssh/id_rsa`
- `IdentityFile` `~/.ssh/id_dsa`

Estas entradas indican que las claves privada y publica de cada usuario se encontrarán en el directorio `.ssh` del HOME del usuario. En este directorio se encuentra también el fichero `authorized_keys2` que controla la autenticación mediante claves, como veremos más adelante.

Comandos básicos de ssh

SSH es un programa que permite acceder a otro ordenador a través de la red, ejecutar comandos en la máquina remota y mover ficheros entre dos máquinas. Provee autenticación y comunicaciones seguras sobre canales inseguros. Es un reemplazo de **rlogin**, **rsh** y **rcp**.

Vamos a ver estos usos básicos del *SSH*.

ssh

El comando **ssh** ofrece comunicación encriptada y segura entre dos sistemas sobre una red no segura. Este comando reemplaza al **telnet**, **rlogin**, **rsh**.

Para iniciar una sesión en otra máquina usando **ssh**:

```
[usuario1@localhost usuario1]$ ssh usuario1@servidor.dominio.es
The authenticity of host 'servidor.dominio.es (192.168.0.2)' can't be established.
RSA key fingerprint is 97:4f:66:f5:96:ba:6d:b2:ef:65:35:45:18:0d:cc:29.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'servidor.dominio.es' (RSA) to the list of known hosts.
usuario1@servidor.dominio.es's password:
[usuario1@servidor.dominio.es usuario1]$
```

Nota: La primera vez que realizas la conexión debes aceptar la firma del otro host. De esta manera se establece una relación de confianza que se traduce en archivar la clave pública de este servidor en el fichero `~/.ssh/known_hosts`.

La sintaxis básica del comando **ssh** es:

```
ssh user@hostname [command]
```

El comando es opcional. Si se especifica en lugar de obtener un shell se ejecuta el comando en la máquina remota.

Por ejemplo podríamos hacer un `ls` en la máquina remota y observar su salida:

```
ssh usuario1@servidor.dominio.es ls
```

O realizar alguna operación mas elaborada como realizar una copia en local de un directorio remoto, como en el ejemplo:

```
ssh usuario1@servidor.dominio.es "tar cf - /home/usuario1" |\
tar xvf -
```

Una de las funcionalidades que le da mayor potencia al **ssh** es la redirección de las X. Si observas la variable de entorno `DISPLAY` observarás que tiene la forma `localhost:n.n`, esta permite que al abrir cualquier aplicación gráfica su salida se redirija al display del cliente.

```
[usuario1@localhost usuario1]$ ssh usuario1@servidor.dominio.es
[usuario1@servidor usuario1]$ echo $DISPLAY
localhost:11.0
[usuario1@servidor usuario1]$ xeyes&
[usuario1@servidor usuario1]$
```

scp

El comando **scp** permite copiar ficheros entre dos máquinas. Utiliza **ssh** para la transmisión de la información, por lo que ofrece la misma seguridad que el **ssh**. De la misma manera utiliza los métodos de autenticación de **ssh**. Este comando reemplaza al **r****cp**, **f****t****p**.

Este es un ejemplo de uso del **scp** para copiar desde la máquina local a una remota:

```
[usuario1@localhost]scp /tmp/file usuario1@servidor.dominio.es:/tmp
```

También podemos copiar ficheros entre dos máquinas remotas:

```
[usuario1@localhost]scp usuario1@anotherhost:/tmp/file \
usuario1@servidor.dominio.es:/tmp
```

La sintaxis del comando es:

```
scp [-pqrvc46] [-F ssh_config] [-S program] [-P port] [-c cipher]
[-i identity_file] [-o ssh_option] [[user@]host1:]file1 [...]
[[user@]host2:]file2
```

Puedes consultar las opciones en la página man de **scp**, estas son las más habituales:

- **-p**: conserva las propiedades del archivo. Permisos del archivo, fecha de última de modificación.
- **-r**: copia recursiva de directorios

La sintaxis para especificar el origen o destino de los archivos tiene la forma **[[user@]host:]file** donde:

- **user**: es el usuario de la máquina. Si no se especifica es el actual.
- **host**: es la máquina origen o destino del archivo. Si no se informa es la máquina local.
- **file**: fichero o directorio a copiar. Por defecto es el directorio HOME del usuario. En caso de ser un directorio deberás especificar la opción **-r**.

sftp

El comando **sftp** transfiere archivos entre máquinas de forma interactiva.

Los comandos interactivos son similares al clásico **ftp**:

```
[usuario1@localhost usuario1]$ sftp servidor.dominio.es
Connecting to servidor.dominio.es...
usuario1@servidor's password:
sftp> help
Available commands:
cd path                Change remote directory to 'path'
lcd path               Change local directory to 'path'
chgrp grp path        Change group of file 'path' to 'grp'
chmod mode path       Change permissions of file 'path' to 'mode'
chown own path        Change owner of file 'path' to 'own'
help                  Display this help text
get remote-path [local-path] Download file
lls [ls-options [path]] Display local directory listing
ln oldpath newpath    Symlink remote file
mkdir path            Create local directory
lpwd                  Print local working directory
ls [path]             Display remote directory listing
lumask umask          Set local umask to 'umask'
mkdir path            Create remote directory
put local-path [remote-path] Upload file
pwd                   Display remote working directory
exit                  Quit sftp
quit                  Quit sftp
rename oldpath newpath Rename remote file
rmdir path            Remove remote directory
rm path               Delete remote file
symlink oldpath newpath Symlink remote file
version               Show SFTP version
!command              Execute 'command' in local shell
!                      Escape to local shell
?                      Synonym for help
sftp>
```

ssh sustituye a *rsh*

Un ejemplo de uso:

```
[usuario1@localhost]sftp usuario1@servidor.dominio.es
sftp> get fichero
```

Métodos de autenticación

SSH puede utilizar varios métodos de autenticación y hay archivos que controlan los permisos para estos métodos.

Autenticación mediante usuario/contraseña

Es la autenticación básica. Se puede habilitar o deshabilitar en `/etc/ssh/sshd_config` y `/etc/ssh/ssh_config`.

Autenticación basada en host/usuario

Como en los comandos “*r*” se puede configurar el acceso a *ssh* mediante ficheros que especifican desde que usuario y máquina se permite.

Estos ficheros son:

- `/etc/ssh/shosts.equiv`: con el mismo funcionamiento que `/etc/hosts.equiv`
- `$/HOME/.shosts`: a nivel de usuario, como el fichero `$/HOME/.rhost`

Autenticación mediante claves

Para obtener el máximo partido a *SSH* podemos utilizar su capacidad de autenticación mediante clave pública y privada.

Para ello el cliente debe generar sus claves privada y pública, compartiendo esta última con el servidor para poder identificarse. Una vez hecho esto las conexiones se podrán establecer sin necesidad de utilizar el esquema clásico de usuario y contraseña.

Nota: Un mensaje encriptado con la pública sólo puede desencriptarse con la correspondiente clave privada. *OpenSSH* utiliza estas propiedades de los algoritmos de clave pública y privada para realizar la autenticación sin intercambio de contraseñas.

Estos son los pasos a seguir para poder utilizar esta autenticación.

1. Generar la clave en el cliente.

```
[usuario1@localhost usuario1]$ ssh-keygen -t dsa
Generating public/private dsa key pair.
Enter file in which to save the key (/home/usuario1/.ssh/id_dsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/usuario1/.ssh/id_dsa.
Your public key has been saved in /home/usuario1/.ssh/id_dsa.pub.
```

```
The key fingerprint is:  
1c:bb:8c:da:c5:a4:db:9f:bb:4d:64:86:a8:29:85:05 usuario1@localhost.localdomain  
[usuario1@localhost usuario1]$
```

La passphrase si se rellena tiene un comportamiento similar a la contraseña, será solicitada al utilizar esta clave. Deberá ser una frase, en la que se pueden incluir espacios en blanco y signos de puntuación.

En nuestro caso hemos decidido dejarla vacía para así poder utilizar este método de autenticación en procesos no interactivos. Puedes utilizar una frase de paso si no confías en la seguridad de tu máquina.

Nota: La passphrase se puede cambiar con el comando `ssh-keygen -p`.

El tipo de la clave se especifica mediante el parámetro `-t`, y puede ser:

- `rsa1` para `ssh v1`
- `rsa, dsa` para `ssh v2`

En el ejemplo se ha utilizado el tipo `dsa`, y se ha guardado en los archivos por defecto. Estos son `$HOME/.ssh/id_dsa` para la clave privada y `$HOME/.ssh/id_dsa.pub` para la clave pública.

La clave privada es la que nos identifica, por lo que debe ser accesible únicamente por el usuario propietario.

2. Compartir la clave pública

La clave pública debe ser incluida en el fichero `$HOME/.ssh/authorized_keys2` de cada máquina en la que deseemos utilizar la autenticación por clave pública.

Para realizar este intercambio podemos utilizar directamente el `ssh` desde la máquina en que hemos generado nuestra clave.

```
[usuario1@localhost usuario1]$ ssh usuario@servidor.dominio.es \  
    'cat >> .ssh/authorized_keys2' < .ssh/id_dsa.pub
```

Ahora ya podemos utilizar los comandos de `ssh` sin utilizar el usuario y contraseña, confiando únicamente en las claves pública y privada.

En el caso de utilizar una passphrase no vacía se solicitaría al realizar la autenticación:

```
[usuario1@localhost usuario1]$ ssh usuario1@servidor.dominio.es  
Enter passphrase for key '/home/usuario1/.ssh/id_dsa':  
[usuario1@servidor usuario1]$
```

Gestión de claves

Para la autenticación mediante claves, OpenSSH ofrece un método para cachear las claves y no tener que introducir la passphrase mas que una vez. Este funcionamiento se obtiene mediante el **ssh-agent**.

Este agente se puede arrancar al inicio de la sesión, y el resto de conexiones con **ssh** utilizarán este agente para realizar la autenticación, que será entonces transparente al usuario.

Al agente se añaden las claves que se desea usar mediante el comando **ssh-add**, si existe passphrase está será la única vez que se solicita y se comprueba en local. Cuando se necesite usar esta clave el agente se encarga de realizar la autenticación con la otra máquina.

Vamos a ver este proceso en detalle.

ssh-agent

Lanzamos el agente, este se queda en background pero informa de las variables de entorno que nos serán útiles.

```
[usuario1@localhost usuario1]$ ssh-agent
SSH_AUTH_SOCK=/tmp/ssh-XXln2tGm/agent.3216; export SSH_AUTH_SOCK;
SSH_AGENT_PID=3217; export SSH_AGENT_PID;
echo Agent pid 3217;
[usuario1@localhost usuario1]$
```

La salida del ssh-agent pueden lanzarse en el shell para tener disponibles las variables de entorno que utilizarán las llamadas a comandos *SSH*. Esta operación se puede realizar en un sólo comando de esta manera:

```
[usuario1@localhost usuario1]$ eval $(ssh-agent)
Agent pid 3229
```

Sugerencia: puedes añadir estas líneas a tu `.bash_profile` para que se ejecuten al inicio de la sesión.

Lo comprobamos:

```
[usuario1@localhost usuario1]$ set |grep SSH
SSH_AGENT_PID=3229
SSH_ASKPASS=/usr/libexec/openssh/gnome-ssh-askpass
SSH_AUTH_SOCK=/tmp/ssh-XX5cnIut/agent.3228 (1)
[usuario1@localhost usuario1]$
```

(1) Path del socket unix que se utiliza para dialogar con el agente.

ssh-add

Una vez tenemos el agente de autenticación en marcha, deberemos añadir las claves que deseamos usar. En nuestro caso la que hemos generado en `id_dsa`.

```
[usuario1@localhost usuario1]$ ssh-add .ssh/id_dsa
Enter passphrase for .ssh/id_dsa:
```

```
Identity added: .ssh/id_dsa (.ssh/id_dsa)
[usuario1@localhost usuario1]$
```

Si la clave tiene passphrase se solicita en este momento, y se utilizará para todas las sesiones que se abran a partir de este momento sin necesidad de volver a introducirla.

keychain

El agente tiene el inconveniente de que sólo se puede usar en las sesiones en las que estén definidas las variables de entorno que hemos comentado. Si se añade el comando **ssh-agent** en el inicio de la sesión el efecto será que tendremos un agente por cada sesión.

Para solucionar este comportamiento puedes utilizar el script keychain que se puede obtener de <http://www.gentoo.org/projects/keychain/>. Este script se incluye en el inicio de la sesión y lanza el **ssh-agent** una sola vez en el sistema.

Ejecución de comandos en modo no interactivo.

La autenticación mediante claves se puede utilizar para ejecutar comandos sobre equipos remotos desde el cron.

Es recomendable que este tipo de comandos se realicen desde un usuario destinado sólo a estas tareas.

Para ello utilizaremos el campo de opciones para la clave pública en el fichero `$HOME/.ssh/authorized_keys2`.

La clave pública tiene los siguientes campos, separados por espacios:

- opciones
- tipo de clave
- clave codificada en base64

```
[usuario1@localhost usuario1]$ cat .ssh/id_dsa.pub
ssh-dss
AAAAB3NzaC1kc3MAAACBAJRsicUj2jLS69TYC/wbzyTalmRcT3MuMmmQWq09hCupSMENGc7FigvjLFjXD5J+9EJ
usuario1@localhost.localdomain
```

Las opciones irán separadas por comas, algunas de las aceptadas son:

- **from:** permite especificar una lista de hosts desde los que se permite o no el acceso mediante clave pública. Es una opción de seguridad, para evitar dejar el acceso libre desde cualquier host a alguien que haya robado la clave privada.
- **command:** indica el comando que se ejecutará cuando se use esta clave, restringe una clave para realizar una operación determinada. El comando que especifique el usuario en su línea de comandos se ignorará.

Sugerencia: La variable de entorno `SSH_ORIGINAL_COMMAND` contiene el comando que lanza el usuario. Esta variable se puede utilizar para extraer los argumentos de la llamada.

ssh sustituye a *rsh*

- **no-pty**: evita el uso de una tty en la ejecución del comando

Sugerencia: Puedes consultar el resto de las opciones en la sección AUTHORIZED_KEYS FILE FORMAT del man de sshd

Utilizaremos esta funcionalidad para acceder, por ejemplo, a logs de apache en sistemas remotos y procesarlos mediante *webalizer*.

Para ello creamos una entrada en `$HOME/.ssh/authorized_keys2` con el siguiente formato:

```
from="IP_cliente.dominio.es",command="/usr/local/root/bin/cat_access_log",no-pty
ssh-dss AAAAB3NzaC1kc3MAAACBAKh0En5pRLIzlsYdXeN2Dy1LPZs7Wg5oGQTsfrksIskTAVm0i2ai9CovnT
saltador@cliente.dominio.es
```

en ella se especifica que al utilizar esta clave, asociada al usuario *saltador* de la máquina cliente sólo desde esta máquina, se ejecute el comando `/usr/local/root/bin/cat_access_log` sin reservar una tty.

El script asociado al comando ejecuta las siguientes acciones:

```
#!/bin/bash

if [ ! -z "${SSH_ORIGINAL_COMMAND}" ]
then
    host=$(echo ${SSH_ORIGINAL_COMMAND}|awk '{print $2}')
else
    if [ $# -lt 1 ]
    then
        echo "Usage: $(basename $0) host"
        exit 1
    else
        host=$1
    fi
fi

LOGDIR=/var/log/httpd
DATA=$(date --date="1 month ago" +%Y%m)
ACCESS=${LOGDIR}/${host}/${DATA}-access_log

if [ -f ${ACCESS} ]
then
    cat ${ACCESS}
fi
```

Este permite consultar el último `access_log` de nuestro servidor web, haciendo uso de la variable `SSH_ORIGINAL_COMMAND` que da acceso a los parámetros originales.

Ahora sólo nos queda configurar el script que se ejecutará desde el cron en la máquina cliente.

```
#!/bin/bash
COMMAND="/usr/local/root/bin/cat_access_log"
SERVER=virtualhost

ssh saltador@servidor.dominio.es "${COMMAND} ${SERVER}" | \
    webalizer - \
```

En este script se ejecuta el comando especificado con el parámetro sobre la máquina remota. La salida del comando la recoge el programa **webalizer** que, en la máquina local, procesará el log del servidor web.

Redirección de puertos

La redirección de puertos nos permite tanto encriptar comunicaciones, que de otra manera se realizarían en texto claro, como acceder a servicios que sólo están disponibles de manera local al servidor.

Si observamos las opciones avanzadas del comando **ssh**:

```
ssh [-afgknqstvxACNTX1246] [-b bind_address] [-c cipher_spec]
    [-e escape_char] [-i identity_file] [-l login_name] [-m mac_spec]
    [-o option] [-p port] [-F configfile] [-L port:host:hostport] [-R
    port:host:hostport] [-D port] hostname | user@hostname [command]
```

vemos que tenemos la posibilidad de redirigir puertos locales o puertos remotos, mediante las opciones **-L** y **-R**.

Vamos a ver un ejemplo útil para cada tipo de redirección.

Redirección puertos locales

El uso típico de esta redirección es la de tener acceso a servicios que sólo están disponibles desde la máquina remota.

Podemos realizar una conexión **ssh** a la máquina remota y redirigir este servicio para utilizarlo de manera local.

La sintaxis es:

```
ssh [-L port:host:hostport] hostname | user@hostname
```

donde:

- **port**: será el puerto local que queremos utilizar para acceder al servicio
- **host**: máquina que ofrece el servicio
- **hostport**: puerto en que **host** ofrece el servicio
- **hostname**: máquina a la que nos conectamos, puede ser la misma que **host**

Se entenderá mejor con un ejemplo.

Voy a utilizar el servicio de proxy que se sirve en el puerto 81 de la máquina proxy.dominio.es y que sólo está disponible desde servidor.dominio.es, mediante el puerto local 8080.

```
[usuario1@localhost usuario1]$ ssh -L 8080:proxy.dominio.es:81 \
                                usuario1@servidor.dominio.es
[usuario1@servidor usuario1]$
```

De esta manera además de obtener un shell realizamos la redirección de puertos. Si no deseamos obtener el shell podemos utilizar la opción **-f** para que el comando se quede en background y **-N** para indicar que no queremos ejecutar comandos remotos.

ssh sustituye a rsh

```
[usuario1@localhost usuario1]$ ssh -fN -L 8080:proxy.dominio.es:81 \
    usuario1@servidor.dominio.es
[usuario1@localhost usuario1]$ netstat -lntp|grep 8080
tcp        0      0 127.0.0.1:8080        0.0.0.0:*              LIS-
TEN        6417/ssh
[usuario1@localhost usuario1]$ ps -fp 6417
UID          PID  PPID  C  STIME TTY          TIME CMD
usuario1     6417   1    0  19:24 ?            00:00:00 ssh -fN -L 8080:proxy.dominio.es:81
[usuario1@localhost usuario1]$
```

Como puedes observar de esta manera el comando **ssh** se queda en background y no tenemos shell en la maquina servidor. Ahora sólo nos quedaría configurar nuestro navegador para utilizar como proxy a localhost por el puerto 8080.

Sugerencia: En el ejemplo el servicio redirigido sólo puede usarse localmente. Si quieres que tu máquina acepte conexiones remotas en este puerto deberás añadir al comando la opción `-g`.

Redirección puertos remotos

Esta redirección se realiza en sentido inverso a la anterior y se utiliza para hacer accesible remotamente un puerto local o accesible localmente.

La sintaxis es:

```
ssh [-R port:host:hostport] hostname | user@hostname
```

donde:

- **port**: puerto remoto donde se ofrecerá el servicio
- **host**: máquina que ofrece el servicio, la local o una a la que tengamos acceso
- **hostport**: puerto en que **host** ofrece el servicio
- **hostname**: máquina a la que nos conectamos, puede ser la misma que **host**

Por ejemplo, este comando conseguirá que en la máquina remota se sirva en el puerto 8001, el servidor de streaming que se está sirviendo en la máquina local en el puerto 8000.

```
ssh -fN -R 8000:localhost:8001 usuario1@servidor.dominio.es
```

Uso del SSH en otras aplicaciones

SSH se puede utilizar junto con otras aplicaciones para hacer segura su comunicación o cómo un método de autenticación. Estos son algunos ejemplos.

pop+ssh

Las sesiones de correo POP suelen ser inseguras. Al establecer la sesión el usuario y contraseña viajan en texto claro, así como el contenido del correo.

Si utilizamos **ssh** para establecer estas conexiones, la información viajará encriptada y lo más importante, la autenticación también.

Para conseguirlo utilizaremos la redirección local de puertos para redirigir el puerto local 11110 al puerto 110 de nuestro servidor de pop.

```
[user1@localhost user1]$ssh -fN -L 11110:popserver:110 popserver
```

De esta manera podemos configurar nuestro cliente de correo para acceder al puerto local 11110 para leer nuestro correo por POP en lugar de acceder directamente, y de manera insegura, a nuestro servidor de POP.

```
[user1@localhost user1]$telnet localhost 11110
Trying 127.0.0.1...
Connected to localhost.localdomain (127.0.0.1).
Escape character is '^]'.
+OK Qpopper (version 4.0.3) at popserver starting.
```

ppp+ssh

Mediante *PPP* y *SSH* es posible crear una red privada virtual (VPN) para crear un tunel entre dos hosts.

Los pasos pueden ser:

- Crear un usuario "vpn" en la máquina que hará de servidor de tuneles.
- Preparar la autenticación mediante claves entre el usuario del servidor y el del cliente, mediante el intercambio de clave pública.
- Configurar el `sudo` en la máquina servidor para que el usuario "vpn" pueda ejecutar el demonio **pppd**.

```
[root@localhost root]# tail -2 /etc/sudoers
ALL ALL= (root) NOPASSWD: /usr/local/bin/driftnet

vpn ALL= (root) NOPASSWD: /usr/sbin/pppd
```

- Instalar en el cliente el script **ppp-over-ssh** que realizará la conexión con el usuario "vpn" en el servidor.

```
#!/bin/sh
REMOTE_ACCOUNT=vpn@servidor.dominio.es
REMOTE_PPPD="sudo /usr/sbin/pppd ipcp-accept-local ipcp-accept-remote noauth"
LOCAL_PPPD="pppd silent 10.0.2.1:10.0.2.2"
$LOCAL_PPPD pty "ssh -t $REMOTE_ACCOUNT $REMOTE_PPPD"
```

- En la máquina local ejecutamos el script. Este ejecuta un **ssh** contra la máquina remota para ejecutar el **pppd** que establecerá la conexión.

ssh sustituye a rsh

De esta manera se establece un canal seguro entre las dos máquinas utilizando una interfaz ppp. Esto permite utilizar los servicios de la máquina remota para todos los usuarios de la máquina local y también en el sentido inverso.

rsync+ssh

Otro de los programas que puede hacer uso del **ssh** es **rsync**. Rsync permite mantener dos directorios de maquinas diferentes sincronizados. Su protocolo hace que se envíe únicamente las diferencias entre los archivos.

Para especificar que usamos **ssh** para conectar entre las dos máquinas usamos el parámetro `-e`:

```
rsync -e ssh -avpz directorio_local usuario@servidor.dominio.es:directorio_destino
```

Conclusión

OpenSSH puede usarse para:

- reemplazar al **telnet**, **rlogin**, **rsh** y **rcp**
- realizar copias de seguridad de manera segura a través de la red
- ejecutar comandos remotos
- acceder a recursos locales desde Internet
- realizar transferencias seguras de ficheros

Bibliografía

1. Paginas man
 - man sshd (8) - demonio, variables de entorno, formato ficheros de claves, formato de ficheros de hosts conocidos
 - man ssh (1) - comando ssh
 - man scp (1) - comando scp
 - man sftp (1) - comando sftp
 - man ssh-keygen (1) - generación de claves
 - man ssh-add (1) - añade identidades al agente de autenticación
 - man ssh-agent (1) - agente de autenticación
 - man ssh_config (5) - ficheros de configuración de usuario
 - man sshd_config (5) - ficheros de configuración del sistema
2. TLDP.org <http://www.tldp.org>
 - Linux Network Administrators Guide.
 - Securing and Optimizing Linux.

- Secure-POP+SSH
- Security-HOWTO
- VPN PPP-SSH Mini-HOWTO

3. Más información

- OpenSSH key management: <http://www-106.ibm.com/developerworks/library/l-keyc.html>

Notas

1. <http://www.chiark.greenend.org.uk/~sgtatham/putty>
2. <http://www.vandyke.com/products/securecrf/index.html>
3. <http://www.openssh.com/>
4. <http://www.gentoo.org/projects/keychain/>
5. <http://www.tldp.org>
6. <http://www-106.ibm.com/developerworks/library/l-keyc.html>

ssh sustituye a *rsh*