# OpenSSH


## Asegurando ssh

Al conectar desde un sistema con **OpenSSH** otro sistema puede ser que necesitemos añadir ciertas opciones cada vez, un ejemplo muy claro es el puerto dónde el sistema destino escucha el **SSH** (y en cada sistema puede ser distinto).

Otra mejora es cambiar el ***port 22*** por otro puerto mayor a ***1024***, que el usuario ***root*** no pueda loguearse por ssh, etc.

***$ vi /etc/ssh/sshd_config***

*Port 2050*
*…*
*# Si tengo mas de una placa le indico la IP por donde quiero que escuche.*
*ListenAddress 192.168.1.1*
*...*
*Protocol 2*
*…*
*PermitRootLogin no*
*…*
*AllowUsers pablo juan*

o si queremos grupos :

*AllowGroups users*

También se puede denegar usuarios :

*DenyUsers jose pedro*

Si queremos ejecutar aplicaciones gráficas remotas en nuestro desktop, habilitamos :

*X11Fordwarding yes*
*X11DisplayOffset 10*
*X11UseLocalhost yes*

Una vez habilitada la opción, agregamos el parámetro -***X*** para indicarle que habilite la ejecución de programas con interfaz gráfica de usuario al conectarnos por ***SSH*** en la línea de comandos.

Configurar inactividad de sesión intervalo de espera

Usuario puede acceder al servidor a través de ***ssh*** y se puede establecer un intervalo de tiempo de

espera ideal para evitar desatendida sesión ***SSH***. Abrir ***sshd_config*** y asegurarnos de que los valores se configuran los siguientes:

***$ vi /etc/ssh/sshd_config***

*ClientAliveInterval 300*
*ClientAliveCountMax 0*

Se va a configurar un intervalo de tiempo de espera en segundos (***300 segundos = 5 minutos***). Después de este intervalo ha pasado, el usuario de inactividad será automáticamente expulsado (que se lee la sesión).

Utilizar TCP Wrappers y iptables

Para eso editamos el archivo ***/etc/hosts.allow*** *y le indicamos las ip que pueden entrar.*

***$ vi /etc/hosts.allow***

*sshd : 192.168.1.2 172.16.23.12*

En el firewall incorporamos las siguientes lineas :

*-A INPUT -s 192.168.0.0/24 -m state --state NEW -p tcp --dport 22 -j ACCEPT*

Un programa muy interesante entre tantos es el programa ***denyhosts***, que nos permite defendernos de ataques de usuarios hacking/cracking.

***$ apt-get install denyhosts***

1. El archivo de configuración por defecto es ***/etc/denyhosts.conf***.

2. También es necesario crear / actualizar una lista blanca en ***/etc/hosts.allow***. Por ejemplo, si tienes IP fija asignada por el ISP, introduzca en este archivo. Puede agregar todos los hosts importante que nunca que desea bloquear.

Editamos el archivo ***/etc/denyhosts***.

***$ vi /etc/denyhosts.conf***

*ADMIN_EMAIL = usuario@correo-electronico*

Verificar que los archivos para Debian estén así :

*SECURE_LOG = /var/log/auth.log*
*HOSTS_DENY = /etc/hosts.deny*
*BLOCK_SERVICE = sshd*
*LOCK_FILE = /var/run/denyhosts.pid*
*…*

Luego restauramos el servicio :

***$ /etc/init.d/denyhosts restart***

Una herramienta para leer los logs es ***logwatch*** o ***logcheck.***

***$ apt-get install logwach***
***$ logwatch***

```
 ###################### Logwatch 7.3.6 (05/19/07) ######################
      Processing Initiated: Wed Oct 12 12:25:54 2011
      Date Range Processed: yesterday
                          ( 2011-Oct-11 )
                          Period is day.
      Detail Level of Output: 0
      Type of Output/Format: stdout / text
      Logfiles for Host: debian
 ##################################################################################

 ---------- dpkg status changes Begin -----------

 Upgraded:
   libssl0.9.8 0.9.8o-4squeeze1 => 0.9.8o-4squeeze2
   openssl 0.9.8o-4squeeze1 => 0.9.8o-4squeeze2

 ---------- dpkg status changes End -----------


 ---------- Kernel Begin -----------


 WARNING:  Kernel Errors Present
   [    5.962787] PM: Error -22 checking ima ...:  1 Time(s)
   [    9.106352] Error: Driver 'pcspkr' ...:  1 Time(s)

 ---------- Kernel End -----------

 ---------- SSHD Begin -----------

 SSHD Killed: 1 Time(s)

 SSHD Started: 2 Time(s)

 Users logging in through sshd:
   root:
     192.168.0.101: 1 time

 Received disconnect:
   11: disconnected by user : 1 Time(s)
 ---------- SSHD End -----------
 ---------- Disk Space Begin -----------

 Filesystem            Size  Used Avail Use% Mounted on
 rootfs                7.0G  1.7G  5.0G  25% /
```

*---------- Disk Space End ----------*

*######################## Logwatch End ##########################*

Para verificar el archivo de configuración ejecutamos :

***$ sshd -t***

**Criptografía pública**

***OpenSSH*** dispone de varios métodos para verificar la identidad de un usuario remoto, uno de ellos es el uso de contraseñas de usuarios, pero otro de los métodos se basa en la autenticación RSA, en donde se dispone de un juego de llaves privada/pública que garantiza la identidad de un usuario intentando conectarse al equipo remoto. El juego de llaves tiene la propiedad de que lo que se encripta con una, sólo se puede desencriptar con la otra, pero sin embargo, a partir de la llave pública no se puede derivar la llave privada.

Para asegurarse de que un usuario es quien dice ser por medio de criptografía pública ***OpenSSH*** emplea el siguiente procedimiento:

- Primero, el servidor genera un número aleatorio que encripta con la llave pública del usuario y le envía el resultado al cliente, si el usuario es quien dice ser entonces no tendrá ningún problema en desencriptarlo.
- El cliente aplicará una transformación predefinida a dicho número y lo enviará de vuelta al servidor, el cual, revisará dicho resultado y de ser correcto, dará acceso al cliente.

Un sistema parecido al anterior es el que se emplea también para verificar que el servidor al que se está conectando es el correcto y no estamos siendo víctimas de un ataque de tipo ***Monkey In the Middle***.

**Creación de certificados con openssh**

Mostraremos la forma de poder entrar a varios servidores sin que nos pida alguna contraseña sino con la creación de certificados.

- Creamos el certificado en el cliente.

***$ ssh-keygen -t rsa***

*Generating public/private rsa key pair.*
*Enter file in which to save the key (/home/pablo/.ssh/id_rsa):*
*Enter passphrase (empty for no passphrase):*
*Enter same passphrase again:*
*Your identification has been saved in /home/pablo/.ssh/id_rsa.*
*Your public key has been saved in /home/pablo/.ssh/id_rsa.pub.*
*The key fingerprint is:*
*3c:d8:d4:a5:a7:81:34:c2:8c:ab:8d:79:d5:8e:f7:af pablo@condor*
*The key's randomart image is:*
*+--[ RSA 2048]----+*
| +. o . |
| . oo + o |

```
|    . .o + .    |
|   . .=.  +     |
|   = ..oS .     |
|  + o . o.      |
|  .  . .        |
|        .       |
|        Eo.     |
+-----------------+
```

- Luego copiamos el archivo generado ***id_rsa.pub*** (llave pública al servidor).

  ***$ ssh-copy-id -i .ssh/id_rsa.pub root@192.168.0.107***

  *The authenticity of host '192.168.0.107 (192.168.0.107)' can't be established.*
  *RSA key fingerprint is 6e:b5:d0:1c:18:f4:f8:62:aa:0b:08:fa:b8:4f:47:0d.*
  *Are you sure you want to continue connecting (yes/no)? yes*
  *Warning: Permanently added '192.168.0.107' (RSA) to the list of known hosts.*
  *root@192.168.0.107's password:*
  *Now try logging into the machine, with "ssh 'root@192.168.0.107'", and check in:*

  *.ssh/authorized_keys*

  *to make sure we haven't added extra keys that you weren't expecting.*

Esto lo que hace en el servidor destino (***192.168.0.107***) es agregar la clave publica ***id_rsa.pub*** al archivo ***authorized_keys*** que se encuentra dentro del ***home*** del usuario ***.ssh***.

Si lo hacemos a mano :

***$ scp ~/.ssh/id_rsa.pub root@192.168.0.107:/root***
***$ ssh root@192.168.0.107***
***$ mkdir .ssh***
***$ chmod 700 .ssh***
***$ cat id_rsa.pub >> .ssh/authorized_keys***

- Por último probamos la conexión y veremos que no nos pide contraseña.

  ***$ ssh root@192.168.0.107***

Si por alguna razón no funciona entonces deberíamos revisar el archivo ***/et/ssh/sshd_config*** y descomentar estas lineas:

**RSAAuthentication yes**
**PubkeyAuthentication yes**

Reiniciamos el servicio **/etc/init.d/ssh restart** y probamos de nuevo y ya debería funcionar sin problemas.

La primera opción (***RSAAuthentication***) sirve para indicar cuando se permitirá autenticación ***RSA***, está opción esta habilitada por defecto.

La segunda opción (**PubkeyAuthentication**) es la que especifica si se podrán usar llaves públicas para demostrar la autenticidad de un usuario. Si su valor es ***yes*** como en el ejemplo, entonces se podrán emplear las llaves públicas, si por el contrario su valor es *no*, entonces el uso de llaves

públicas quedará prohibido.

Tamaño de las llaves públicas

Por defecto, ***ssh-keygen*** genera llaves de ***2048 bits***, cuanto más grande sea una llave, más segura será. En la actualidad, ***ssh-keygen*** admite que las llaves tengan un mínimo de ***512 bits*** y aunque en la página del manual no se indique, el máximo son ***32768 bits***, pero este último valor podría cambiar a medida que la potencia de los equipos informáticos se incremente. No obstante, con la liberación de la versión ***4.3*** de ***OpenSSH***, sus desarrolladores decidieron fijar el tamaño de las llaves DSA a ***1024 bits***, pudiendo alterarse tan sólo el tamaño para las llaves RSA.

Por lo general el número de bits para la llave que escoge ***ssh-keygen*** por defecto es suficiente, pero para quienes prefieran otros valores, se puede especificar el tamaño de la llave con el parámetro ***-b*** seguido del número de bits que se desea que tenga la llave. Un ejemplo para generar una llave ***RSA*** de ***4096 bits*** podría ser el siguiente:

***$ ssh-keygen -t rsa -b 4096***

## Cambiar la frase clave de una llave privada

Puede ser que queremos cambiar la frase con la que una llave privada fue encriptada, o en el caso de que la llave privada no estuviese encriptada, querer encriptarla. Para conseguir este objetivo podemos invocar al programa ssh-keygen con el parámetro ***-p***, veamos un ejemplo:

***$ ssh-keygen -p***

*Enter file in which the key is (/root/.ssh/id_rsa):*

*Key has comment '/root/.ssh/id_rsa'*

*Enter new passphrase (empty for no passphrase):*

*Enter same passphrase again:*

*Your identification has been saved with the new passphrase.*

En primer lugar nos pide la frase con la que está encriptada (old passphrase), a continuación, nos pide que introduzcamos la nueva frase (new passphrase), entre paréntesis, nos indica que si no ponemos nada, la llave privada quedará sin encriptar (empty for no passphrase). Después nos pide que repitamos la frase, para asegurarse de que no hemos cometido errores al escribirla la primera vez y finalmente, graba la llave privada encriptada con la nueva frase**.**

## Agente ssh

Una de las ventajas de emplear llaves públicas frente al uso de contraseñas de usuario es que no tenemos que recordar nada más que una única frase, la frase con la que hemos cifrado nuestra llave privada. Eso es un avance, pero ***OpenSSH*** dispone de una herramienta que nos puede evitar el trámite de tener que andar escribiendo dicha frase cada vez que establezcamos una nueva conexión. Esta herramienta se llama ***ssh-agent*** y tiene la capacidad de recordar las llaves privadas que tenemos.

Al ejecutar el agente ***ssh*** (***ssh-agent***), este crea un ***socket UNIX*** y establece la variable de entorno ***SSH_AUTH_SOCK*** con el nombre del ***socket***. Por razones de seguridad los permisos del ***socket***

son ajustados para que tan sólo el usuario actual pueda acceder al ***socket***. Además, el agente también crea la variable de entorno ***SSH_AGENT_PID*** y establece su valor con su ***PID*** (identificador de programa).

Cuando el cliente de ***SSH*** necesita autenticar a un usuario, lo primero que hace es mirar si existe la variable de entorno ***SSH_AUTH_SOCK***, de ser así, la usa para establecer una conexión con el agente, el agente no le pasa la llave privada al cliente de ***SSH***, sino que es el propio agente el que se encarga de realizar la autenticación, de forma que la llave privada nunca sea expuesta a los clientes.

Iniciando el agente

Hay varias formas de cargar el agente entre una de ella es :

***$ ssh-agent***

*SSH_AUTH_SOCK=/tmp/ssh-PgJvc11615/agent.11615; export SSH_AUTH_SOCK;*
*SSH_AGENT_PID=11616; export SSH_AGENT_PID;*
*echo Agent pid 11616;*

La salida anterior corresponde a una shell ***Bourne*** y no es más que los comandos que habría que ejecutar para establecer las variables de entorno ***SSH_AUTH_SOCK*** y ***SSH_AGENT_PID***, en el caso de emplear una shell derivada del ***csh*** la salida sería ligeramente distinta.

El programa ***ssh-agent*** intenta averiguar el tipo de shell que se está usando, no obstante, se le puede pasar la opción -***c*** para indicar una shell derivada de ***csh***, o la opción -***s*** para que muestre los comandos apropiados para una shell derivada del ***Bourne***.

Añadir llaves al agente

El agente nada más iniciarse no contiene ninguna llave, para agregarlas se emplea la herramienta ***ssh-add***, si se ejecuta sin argumentos intenta añadir los archivos ***~/.ssh/id_rsa***, ***~/.ssh/id_dsa*** y **~/.ssh/identity**:

***$ ssh-add***

Enter passphrase for /home/pablo/.ssh/id_rsa:
Identity added: /home/pablo/.ssh/id_rsa (/home/pablo/.ssh/id_rsa)

En el caso anterior ***ssh-add*** primero encuentra el archivo ***~/.ssh/id_rsa*** y pide la frase con la que está cifrada la llave que contiene, entonces añade dicha llave al agente, después, localiza el archivo ***~/.ssh/id_dsa*** que contiene otra llave, pero al estar cifrada con la misma frase que la primera llave, no vuelve a pedir la frase, si no que la descifra y la añade directamente al agente.

Listar las llaves que hay en el agente

En caso de querer ver que llaves contiene el agente, podemos pasarle la opción -***l*** a la herramienta ***ssh-add***, y obtendremos una salida como la siguiente:

***$ ssh-add -l***

*2048 3c:d8:d4:a5:a7:81:34:c2:8c:ab:8d:79:d5:8e:f7:af /home/pablo/.ssh/id_rsa (RSA)*

Para mostrar las claves públicas utilizamos la opción ***-L***, lo cual puede ser útil para añadirlas al archivo ***authorized_keys*** de una máquina remota :

***$ ssh-add -L***

*ssh-rsa*
*AAAAB3NzaC1yc2EAAAADAQABAAABAQCmmdA5yDOLCv5fPxPp7vVVHKh70kV*
*n7noTCtkJwgovsqGtk8j1LH7FCDTsvDP4XFdSMNZJ/EgK5PWbjt0LRdt7jWI+dZro*
*p/7Sf3AYVAMGIf9//CMq4uKtUDru/JvN2HZe7geey1jgqkdTTMtSCtSQeefp1Vhk1fap*
*dB8mqYFwGOvBs/OFryp/XI4TM+RLhyr20oYqe99LPUuPx/qSwo4dzsqtoZGO+0Mh*
*/hrOZFp4pkEJibWlXO8+0O6l65PP+Uwj0vG+Tw3s1r9JfNgyyqYrgDyOHKmu6beQ*
*LlIWgeqUkVVYIgTmoR3CpXDSdtGweEMh/Q79VS9G6AFJkRC+aS49*
*/home/pablo/.ssh/id_rsa*

**Nota**: Cada llave aparece en una línea, pero como cada línea es muy larga, he eliminado parte del contenido de la llave pública, sustituyéndolo por puntos suspensivos para que así quepa bien en la pantalla.

Eliminar llaves del agente

También podemos eliminar llaves del agente empleando la herramienta ***ssh-add*** con la opción ***-d*** seguida de la llave pública que le corresponde a la llave privada que queremos que el agente borre de su memoria. Por ejemplo, para que olvide nuestra llave RSA:

***$ ssh-add -d .ssh/id_rsa***

*Identity removed: .ssh/id_rsa (.ssh/id_rsa.pub)*

No obstante, también podemos eliminar todas las llaves que tenga el agente empleando la opción ***-D***, ejemplo:

***$ ssh-add -D***

```
All identities removed.
```

Bloquear el agente

Cuando no necesitamos usar el agente, pero no queremos que se olvide de nuestras llaves, como en el caso de dejar el equipo en el que estamos trabajando para ir a tomar un cafetín, podemos optar por bloquear el agente, para ello tan sólo hace falta pasarle la opción ***-x*** a la herramienta ***ssh-add***, la cual nos pedirá una contraseña con la que luego podremos desbloquear el agente:

***$ ssh-add -x***

```
Enter lock password:
Again:
Agent locked.
```

Luego, cuando necesitemos volver a usar el agente, podemos desbloquearlo ejecutando la herramienta ***ssh-add*** con la opción ***-X***, ***ssh-add*** entonces nos pedirá la contraseña para desbloquear el agente:

***$ ssh-add -X***

```
Enter lock password:
Agent unlocked.
```

Reenviar el agente

Probablemente más de una vez tengamos que acceder a un equipo remoto que se encuentra dentro de una red privada (***LAN***), pero tendremos que hacerlo a través de un router, primero conectando al router, y después conectando al equipo remoto. Estableciendo así dos conexiones, una entre el equipo local y el router, y la otra entre el router y el equipo remoto dentro de la LAN.

El agente ***ssh*** será capaz de autenticarnos en la primera conexión sin ningún problema, pero para que nos autentique en la segunda, este deberá ser reenviado a través de la primera conexión. Por defecto, ***OpenSSH*** no está configurado para reenviar el agente, con lo que habrá que ejecutar el cliente con la opción ***-A***, por ejemplo, para conectar con el router y reenviarle el agente:

***$ ssh-add -A IP-SERVIDOR***

Ahora, si miramos las variables de entorno, podemos apreciar como en la sesión que hemos abierto en el router, efectivamente el agente se ha reenviado:

***$ echo $SSH_AUTH_SOCK***

*/tmp/ssh-sQwzt23661/agent.23661*

¿Cómo funciona esto realmente? Bien, al iniciar la sesión interactiva en el router, el cliente de ***SSH*** crea un ***socket UNIX*** y establece la variable ***SSH_AUTH_SOCK*** apuntando hacia el ***socket***. Cuando se ejecuta otro cliente de ***SSH*** para conectar desde el router a otro equipo, el nuevo cliente de ***SSH*** se conecta a ese ***socket***, y el primer cliente de ***SSH*** reenvía todo lo que recibe por ese ***socket*** hacia el agente que se esta ejecutando en nuestro equipo local.

Gracias a este mecanismo, podemos hacer que nuestro agente nos siga autenticando en el resto de equipos a los que nos conectemos, de alguna manera, es como si el agente pudiese viajar con nosotros. Por ejemplo, si desde la sesión interactiva del router queremos conectarnos a un equipo dentro de la LAN llamado ***SERVIDOR-REMOTO***, podremos poner algo como lo siguiente:

***$ ssh -A SERVIDOR-REMOTO***

***$ echo $SSH_AUTH_SOCK***
*/tmp/ssh-OsvWx13346/agent.13346*

Se puede apreciar, que se ha creado un nuevo ***socket UNIX*** y se ha establecido la variable ***SSH_AUTH_SOCK*** apuntando hacia el, lo cual permitirá que nuestro agente, que esta ejecutándose en nuestro equipo local, pueda seguir autenticándonos en las conexiones que realicemos desde ***SERVIDOR-REMOTO***.

Tiempo de vida para llaves

Se puede hacer que el agente recuerde las llaves durante un tiempo determinado, a partir del cual, el agente las olvidará. Por defecto, el agente no olvida las llaves nunca (mientras se siga ejecutando, o no se le fuerce a olvidarlas), pero se le puede especificar la opción ***-t*** seguida del tiempo de vida que se quiere que tengan las llaves:

***$ ssh-agent -t 60 bash***

El comando anterior inicia un nuevo shell bash con el agente, y establece el límite de vida de las llaves por defecto en ***60 segundos***. Pero también se puede establecer un tiempo límite de vida para cada llave al añadirla con el comando **ssh-add**, la forma de hacerlo, es la misma, especificando la opción -***t*** seguida del tiempo límite de vida que se quiere que tenga la llave, por ejemplo:

```
$ ssh-add -t 120 remote
```

De esta forma, pasados dos minutos, el agente eliminará la llave **remote** de su memoria y no podremos seguir empleándola para autenticarnos a no ser que se la añadamos de nuevo al agente.

**SOCK con ssh**

Puede actuar como un SOCK tanto en los protocolos SOCKS4 y SOCKS5. Esto es para el uso de internet cuando estamos restringidos. Lo bueno de un SOCKS proxy es que, a diferencia de un proxy HTTP, admite cualquier tipo de protocolo y conexión para rutear, y por ende muchas aplicaciones tienen soporte para configurar uno.

Las opciones que vamos a utilizar son :

- **-N:** No ejecutar ningún comando, simplemente deja la conexión abierta para hacer port forwarding.

- **-D [bind:]port**: Especificamos donde se pone a la escucha el **SOCKS server**.

Por ejemplo :

***# ssh -ND 1090 usuario@IP***

Puede ser que no se permita el puerto saliente ***22***, por lo que debemos cambiar al servidor que nos conectamos de puerto ***22*** a puerto ***443*** (***https***). Para eso modificamos el ***sshd_config*** del servidor al cual nos conectamos :

***# vi /etc/ssh/sshd_config***

Port 443

Verificamos que realmente este escuchando el puerto en el servidor :

***# netstat -tpan | grep LIST | grep 443***

tcp 0 0 :::443 :::* LISTEN 27538/sshd

Ahora teniendo escuchando en el puerto 443 el ssh ejecutamos el comando de la siguiente forma :

***# ssh -p 443 -ND 1090 usuario@IP***

En nuestro navegador configuramos en la parte de proxy SOCKS como ***localhost:1090***, una forma de probar que esto funciona es de la siguiente manera :

***# curl -socks5 localhost:1090 -dump systemadmin.es***

Todo el tráfico originado esta cifrado, por lo que no puede ser inspeccionado por la empresa. Desde el servidor destino hasta el site que se quiera visitar vuelve a estar claro.

## Crear túneles con SSH

***OpenSSH*** nos permite crear dos clases de túneles:

- **Locales** = En los locales se redirección un puerto de la máquina local (cliente) hacia un puerto en una máquina remota a la que el servidor tenga acceso.
- Remotos = en los túneles remotos, lo que se hace es redireccionar un puerto desde una máquina remota a la que el servidor tenga acceso hacia un puerto de la máquina local.

## Túneles locales

***La sintaxis para crear un túnel local es:***

***$ ssh -L [puerto_local]:localhost:[puerto_remoto] servidor_remoto***

Un claro ejemplo y muy sencillo para comprender, es que tengamos un servidor Web y necesitemos visualizar alguna página protegida por el servidor (por los ficheros ***.htaccess***). El problema en las páginas protegidas mediante éste método radica en que cada vez que ingresamos un usuario y contraseña, éstas viajan por la red como texto plano y puede ser interceptada por algún tercero. Usando túneles SSH podemos controlar este problema, ya que todos los datos viajan encriptados y seguros.

Para redireccionar el puerto que escucha el servidor web remoto a nuestro ordenador, ejecutamos:

***$ ssh -L 10080:localhost:80 usuario@servidor-web-remoto.com***

El parámetro ***-L*** nos permite realizar el túnel a través de un puerto local. Con esto, ya tenemos creado un Túnel entre el puerto 10080 de nuestra computadora y el puerto ***80*** que atiende el servidor Web remoto. Si tecleamos en el navegador web ***http://localhost:10080*** podremos acceder sin problemas al servidor web remoto con la seguridad de que todos los datos viajan encriptados.

Una forma fácil de configurar esto y que nos quede es la siguiente :

***$ vi ~/.ssh/config***

```
Host servidor-centrux
HostName centrux.dyndns.org
    User marcos
    Port 443
    LocalForward 8080 localhost:443
```

***$ ssh servidor-centrux***

## Túneles Remotos

La sintaxis es la siguiente:

***$ ssh -R [puerto_remoto]:localhost:[puerto_local] terminal_remoto***

Siguiendo con el ejemplo antes mencionado, imaginemos un servidor Web escuchando el puerto 80 y deseamos que una computadora remota pueda tener acceso al puerto del servidor web y en éste caso, nos encontramos del lado del servidor. Para realizar el túnel remoto ejecutamos:

$ ***ssh -R 10080:localhost:80 usuario@pc-cliente-remoto***

Usando el parámetro -***R*** podremos crear túneles remotos desde el puerto del servidor.