# Acceso a máquinas usando el SSH sin contraseña

**Sergio Talens-Oliag**

InfoCentre

<stalens@infocentre.gva.es>

---

En este documento se explica como acceder a servidores Unix utilizando el **ssh** sin emplear contraseñas y como limitar este acceso para que sólo sea posible ejecutar programas concretos.

---

# Introducción

En lo que sigue supondremos que las máquinas servidoras y clientes emplean el openssh (versión 3.5 o superior, disponible en http://www.openssh.org/).

El **ssh** proporciona varios métodos de acceso sin contraseña; el que describiremos aquí se basa en el uso de criptografía de clave pública para identificar a los usuarios.

En los apartados siguientes explicaremos como generar el par de claves (identidades), como instalarlas en los distintos equipos y como restringir el acceso para que sólo se permita la ejecución de programas concretos en función de la clave.

---

# Generación de claves

Para generar los pares de claves se empleará el programa **ssh-keygen**. Este programa permite crear tres tipos distintos de claves:

1. Clave RSA para el protocolo SSH 1 (opción `-t rsa1`)
2. Clave RSA para el protocolo SSH 2 (opción `-t rsa`
3. Clave DSA para el protocolo SSH 2 (opción `-t dsa`

En los ejemplos emplearemos claves RSA para el protocolo SSH 2.

Por ejemplo si queremos crear una clave dentro del directorio `~/sshids/` haremos lo siguiente:

```
boo:~$ test -d $HOME/sshids || mkdir $HOME/sshids
boo:~$ ssh-keygen -t rsa -f $HOME/sshids/idrsa-1
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase): Enter
Enter same passphrase again: Enter
Your identification has been saved in /home/sto/sshids/idrsa-1.
Your public key has been saved in /home/sto/sshids/idrsa-1.pub.
The key fingerprint is:
fd:0a:85:ff:e9:0a:14:94:40:e2:47:bd:af:ee:13:84 sto@boo
```

**Nota:** En principio dejamos la clave sin contraseña, ya que este sistema se va a utilizar sin terminales (p. ej. desde scripts que se lanzan con **cron**).

De cualquier modo hay que indicar que es conveniente cifrar la clave pública siempre que sea posible (por ejemplo cuando se utiliza este tipo de autentificación con usuarios que se conectan al **shell** en modo interactivo).

Después de la ejecución del lo anterior tendremos la clave pública en el fichero `$HOME/sshids/idrsa-1.pub` y la privada en `$HOME/sshids/idrsa-1`

---

# Instalación en un servidor

Para poder acceder al servidor sin contraseña deberemos añadir la clave pública al fichero `~/.ssh/authorized_keys` del usuario remoto:

```
boo:~$ scp $HOME/sshids/idrsa-1.pub user@server:
user@server's password:
******
id_rsa-1.pub         100% |*****************************|   217       00:00
user@server:~$ test -d $HOME/.ssh || mkdir $HOME/.ssh
user@server:~$ cat $HOME/idrsa-1.pub >> $HOME/.ssh/authorized_keys
user@server:~$ rm $HOME/idrsa-1.pub
```

Adicionalmente deberemos verificar que el directorio y el fichero de claves autorizadas sólo están accesibles para el usuario, ya que el **ssh** puede ignorar el fichero si no tiene los permisos adecuados:

```
user@server:~$ chmod 0700 $HOME/.ssh/
user@server:~$ chmod 0600 $HOME/.ssh/authorized_keys
user@server:~$ exit
```

Para probar si todo funciona bien intentaremos entrar en el servidor con la clave recién instalada:

```
boo:~$ ssh user@server -i $HOME/identity.pub
user@server:~$ hostname
server
user@server:~$ exit
```

E incluso probando a ejecutar un comando remoto:

```
boo:~$ ssh user@server -i $HOME/sshids/idrsa-1 'echo "Entré ..."'
Entré ...
```

Si no podemos entrar habrá que comprobar si en el servidor está habilitado el acceso con clave pública para la versión del protocolo que estamos empleando.

En el caso de claves RSA con SSH 2 deberemos comprobar que en el fichero `sshd_config` esté habilitado el protocolo ssh2 (opción `Protocol`) y la autentificación con claves (opción `PubkeyAuthentication yes`)

Si aun así no funciona podemos ejecutar el cliente **ssh** con la opción *-v* y mirar los logs en el servidor.

---

# Restricciones en función de la identidad remota

Para controlar lo que una identidad remota puede ejecutar en el servidor deberemos modificar el fichero `$HOME/.ssh/authorized_keys` perteneciente al usuario del servidor.

En el ejemplo que estamos siguiendo el contenido del fichero sería algo como lo siguiente:

```
user@server:~$ cat $HOME/.ssh/authorized_keys
  ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAIEAw9SSGl4yB/sux2bdNAkWyJRSNkbdUK
  sofjPcBTHsRqgqSyZp7MuG9/IU+C9VhaGBa5XMaZwUPiyL0DoJg8lAtK2+GgaXvW/F
  7MoLPcUpVGzaYJ0izT2KAFd+3X9JCeOqjgeM7cqzY2zrXE9161E+gB8lqckkiYP+8m
  b9qtQI8lk= sto@boo
```

Hay que indicar que cada clave debe ir en una sola línea, independientemente de como se visualice en este documento.

Para limitar lo que se puede hacer con esta clave se pueden añadir opciones delante de la clave, las opciones se separan entre si utilizando comas y de la clave usando espacios.

Algunas de las opciones más útiles son las siguientes:

`no-port-forwarding`, `no-X11-forwarding`, `no-agent-forwarding`

> Bloquean el uso de las opciones de redireccionamiento de puertos, del protocolo X11 y del agente de identificación. En general es conveniente que una cuenta restringida incluya estas opciones.

`nopty`

> Deshabilita la reserva de un terminal para la ejecución del comandos. Es recomendable incluir esta opción siempre que lo que vayamos a ejecutar no lo necesite.

`environment="VARIABLE=valor"`

> Se pueden añadir tantas opciones `environment` como se quiera y sirven para definir variables de entorno para la identidad remota.

`from="lista"`

> Opción que permite limitar desde que direcciones se acepta esta identidad. La lista de máquinas va separada por comas y permite el uso de expresiones regulares simples (`*` para representar una subcadenas, `?` para representar un carácter cualquiera y `!` para negar el acceso desde una máquina concreta)

`command="instrucción del shell"`

> Esta opción permite especificar el comando a ejecutar si se conecta un usuario con esta identidad. En caso de que el usuario intentase lanzar un comando propio este se le pasa al nuestro en la variable de entorno `SSH_ORIGINAL_COMMAND`.

Para una lista completa se puede consultar la página de manual del **sshd**.

Empleando estas opciones podemos controlar de modo muy fino qué se puede ejecutar en el servidor.

Por ejemplo, si sólo queremos que la clave anterior se pueda utilizar desde la máquina `boo.infocentre.gva.es`, no pueda hacer reenvíos ni usar terminales pondremos lo siguiente delante de la clave:

```
user@server:~$ cat $HOME/.ssh/authorized_keys
from="boo.infocentre.gva.es",no-port-forwarding,no-X11-forwarding,
no-agent-forwarding,no-pty ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAIEAw9S
SGl4yB/sux2bdNAkWyJRSNkbdUKsofjPcBTHsRqgqSyZp7MuG9/IU+C9VhaGBa5XMa
ZwUPiyL0DoJg8lAtK2+GgaXvW/F7MoLPcUpVGzaYJ0izT2KAFd+3X9JCeOqjgeM7cq
zY2zrXE9161E+gB8lqckkiYP+8mb9qtQI8lk= sto@boo
```

Con esto conseguimos que la clave sólo sea útil para ejecutar comandos que no necesiten terminal desde la máquina `boo.infocentre.gva.es`.

Si además queremos limitar la ejecución a un comando concreto podemos introducir la opción `command`, que hará que cuando el usuario remoto conecte con esa identidad se ejecute lo que nosotros pongamos en lugar de lo solicitado por el cliente. En el entorno de ese programa se incluye la variable `SSH_ORIGINAL_COMMAND` con el contenido del comando enviado por el cliente:

```
user@server:~$ cat $HOME/.ssh/authorized_keys
command="$HOME/.ssh/logrcmd",no-port-forwarding,no-X11-forwarding,
no-agent-forwarding,no-pty ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAIEAw9S
SGl4yB/sux2bdNAkWyJRSNkbdUKsofjPcBTHsRqgqSyZp7MuG9/IU+C9VhaGBa5XMa
ZwUPiyL0DoJg8lAtK2+GgaXvW/F7MoLPcUpVGzaYJ0izT2KAFd+3X9JCeOqjgeM7cq
zY2zrXE9161E+gB8lqckkiYP+8mb9qtQI8lk= sto@boo
```

Aunque el uso de la opción `command` es muy útil, tiene unas cuantas pegas:

- No siempre sabemos cual es el comando que ejecutamos en la máquina remota (p. ej. al invocar el **scp** o cuando usamos el **ssh** como sustituto del **rsh** en programas como el **rsync** o el **cvs**).
- Aunque sepamos cual es el comando que se invoca en la máquina remota, si lo ponemos en el `command` de una identidad, siempre que conectemos con ella se ejecutará ese programa, independientemente de lo que se solicite desde nuestro lado.

Para solucionar el primer punto podemos emplear dos técnicas:

- Ejecutar el **ssh** con la opción *-v* y buscar en la salida la línea `debug1: Sending command: XXXX`.

  Por ejemplo para aplicar esta técnica con el **rsync** podemos exportar la variable `RSYNC_RSH` con el valor `ssh -v` o invocarlo añadiendo la opción *--rsh='ssh -v'*.

- Empleando un `script` auxiliar que nos guarde un histórico de los comandos que se solicitan y los ejecute:

  ```
  #!/bin/sh
  LOGFILE="$HOME/.ssh/rcmd.log"
  TIMESTAMP=`date +"%Y/%m/%d-%H:%M:%S"`
  echo "[$TIMESTAMP]: '$SSH_ORIGINAL_COMMAND'" >> $LOGFILE
  exec $SSH_ORIGINAL_COMMAND
  ```

La segunda pega se puede solventar utilizando un script similar al de logging de comandos que valide el valor de la variable de entorno `SSH_ORIGINAL_COMMAND` y sólo lo ejecute si la identidad está autorizada.

Un ejemplo que permitiría la ejecución de los comandos **date** y **uptime** podría ser el siguiente:

```
#!/bin/sh
LOGCMD="$HOME/.ssh/rcmd.log"
```

```
LOGERR="$HOME/.ssh/rcmd.err"
TIMESTAMP=`date +"%Y/%m/%d-%H:%M:%S"`
case "$SSH_ORIGINAL_COMMAND" in;
  'date'|'uptime')
    CMD="$SSH_ORIGINAL_COMMAND"
  ;;
  *)
    echo "[$TIMESTAMP]: '$SSH_ORIGINAL_COMMAND'" >> $LOGERR
    exit 0
  ;;
esac
echo "[$TIMESTAMP]: '$SSH_ORIGINAL_COMMAND'" >> $LOGCMD
exec $CMD
```

En el ejemplo no se emplean expresiones regulares para el valor del `SSH_ORIGINAL_COMMAND` por que complica la gestión, en principio lo mejor es que si se emplea esta técnica se intente limitar lo más posible los comandos a ejecutar.

Por último indicar que además de la limitación para identidades concretas se debe revisar la configuración general del servidor **ssh** y los tipos de acceso configurados.