FREE eBook

LEARNING Secure Shell

Free unaffiliated eBook created from
Stack Overflow contributors.

#ssh

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: secure-shell

It is an unofficial and free Secure Shell ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Secure Shell.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with Secure Shell

Remarks

The SSH (Secure Shell) is a cryptographic protocol for point-to-point communication over the insecure network (Internet). It obsoletes the old protocols used in the old days (`rlogin`, `rsh`, `telnet`).

It is commonly used to connect to remote servers, virtual machines or containers in data center or in your private cloud (google compute engine, AWS, ...). But it is also commonly used in conjunction with `git` for accessing and updating your repository securely and easy using public keys instead of passwords.

It should also mention any large subjects within ssh, and link out to the related topics. Since the Documentation for ssh is new, you may need to create initial versions of those related topics. TODO

Versions

Version	Release notes	Release Date
OpenSSH 7.3p1	Latest version	2016-08-01
OpenSSH 7.2p2		2016-03-09
OpenSSH 7.1p2		2016-01-14
OpenSSH 7.1		2015-08-21
OpenSSH 7.0		2015-11-08
OpenSSH 6.9		2015-07-01
OpenSSH 6.8		2015-03-18

Examples

Installation or Setup

Free version of SSH protocol implementation, OpenSSH is available in all the Linux distributions. It consists of the server and client packages.

Installation

On Debian-based Linux, you can install `openssh` using

```
# apt-get install openssh-server openssh-client
```

On RHEL/CentOS you need to use `yum`:

```
# yum install openssh-server openssh-clients
```

Current Fedora is using `dnf` instead of `yum`.

On Arch Linux, use pacman:

```
# pacman -S openssh
```

On OSX, the `openssh` should be already installed.

If you want to use more recent version, you need to install the `openssh` from brew:

```
# brew install openssh --with-brewed-openssl --with-keychain-support
```

TODO Instructions for Windows

Setup

The `openssh` client does not need any special setup and is ready to use just after installation. You can try that running `ssh remote`, where the `remote` is the remote host running `ssh` server.

The `openssh` server is usually started after the installation and default setup is applied. If not, you can start it on `systemd` based systems using

On Debian-based Linux with systemd:

```
# systemctl start ssh
```

On RHEL/CentOS/Fedora and Arch Linux:

```
# systemctl start sshd
```

Or on upstart systems using

```
# service sshd start
```

Configuration

The `openssh` have configuration files under `/etc/ssh/`. The client is also reading client configuration in `~/.ssh/config`. The server is using a file `sshd_config`, which contains most of the default values and contains simple key-value pairs. Example:

```
Protocol 2
PasswordAuthentication yes
ChallengeResponseAuthentication no
UsePAM yes
AcceptEnv LANG LC_CTYPE LC_NUMERIC LC_TIME LC_COLLATE LC_MONETARY
X11Forwarding yes
Subsystem    sftp    /usr/libexec/openssh/sftp-server
```

Creating your SSH key

You can create your ssh key using ssh-keygen, it's a program that is part of the ssh installation. To do so just run it and follow the instructions on screen.

Here's an example:

```
$ ssh-keygen
Generating public/private rsa key pair.
```

The default directory where you ssh key pair will be saved is inside the `.ssh` folder in your home directory (you can change this by specifying a valid path) and the default name for the keypair is `id_rsa` for the private key and `id_rsa.pub` for the public key:

```
Enter file in which to save the key (/home/nasreddine/.ssh/id_rsa): /home/my_folder/my_ssh_key
```

You can protect your SSH key from unauthorized use by entering a password. This is optional but it's recommended that you use a passphrase. Note that, like with any other command program, when entering your passphrase it will not show anything on screen but it is being recorded:

```
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
```

Once you enter your passphrase ssh-keygen will generate a key and save it to the path you chose:

```
Your identification has been saved in /home/my_folder/my_ssh_key.
Your public key has been saved in /home/my_folder/my_ssh_key.pub.
```

We're done. Now our ssh key is ready for use.

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/nasreddine       /.ssh/id_rsa): /c/dev/my_ssh_k
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/dev/my_ssh_key.
Your public key has been saved in /c/dev/my_ssh_key.pub.
The key fingerprint is:
SHA256:j9D2KP39dA7uOaRH0V6nNtG7I2srq+fZ+57rpVbPC78 nasreddine
The key's randomart image is:
+---[RSA 2048]----+
|                 |
|                 |
|               o |
|        .      o =|
|       . S      =+|
|        + =    *.o|
|       . + o  *o+=|
|        . . ++=*O=|
|          .=+=X#E*|
+----[SHA256]-----+
```

How to SSH into a machine

In order to login to a user's account on machine with SSH you can use the command `ssh username@ip_address`. It will ask for a password. If you type the correct password, you will be connected to the shell of that user on that machine. Otherwise it will prompt for the password again.

For example

```
root@dev10:~# ssh root@10.11.50.3
root@10.11.50.3's password:
Welcome to Ubuntu 16.04 LTS (GNU/Linux 4.4.0-31-generic x86_64)

Last login: Fri Jul 22 18:33:27 2016 from 10.11.50.10
root@dev2:~#
```

If you want to use a specific ssh key to connect to a machine, use `ssh -i /path/to/ssh_secret_key username@host`

When you are connecting to a machine for the very first time, it will ask you to verify the fingerprint of the target machine. This is a security mechanism for avoiding a man-in-the-middle attack. You can see the fingerprint of the target machine by issuing this command in the target machine.

```
ssh-keygen -l -E md5 -f /etc/ssh/ssh_host_ecdsa_key.pub
```

You can type "yes" if both are same. It will proceed to password prompt.

Example:

```
root@dev10:~# ssh root@10.11.50.3
```

```
The authenticity of host '10.11.50.3 (10.11.50.3)' can't be established.
ECDSA key fingerprint is dd:a3:de:cd:5b:01:cd:0b:b6:bc:b3:09:c2:c8:1a:68.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.11.50.3' (ECDSA) to the list of known hosts.
root@10.11.50.3's password:

Last login: Fri Jul 22 17:45:09 2016 from 10.11.1.71
root@dev2:~#
```

Adding your public key to the list of server user's authorized keys

In order to `ssh` into a server your identity's public key has to be added to the list of trusted keys. Most commonly this is done per-user:

```
ssh-copy-id -i ~/.ssh/<identity>.pub <user>@<hostname>
```

Which can be also done manually:

```
cat ~/.ssh/<identity>.pub | ssh <user>@<hostname> 'cat >> ~/.ssh/authorized_keys'
```

After doing that you should be able to log in without need to provide user's password when passing the identity file to the `ssh` call.

Connecting from script using password

When you really need to script `ssh` connection, piping the password into the `ssh` command does not work (`echo passw0rd | ssh host`). It is because the password is not read from standard input, but directly from TTY (teleprinter, teletypewriter, Teletype for historical reasons).

But there is `sshpass` tool which works around this problem. It can read the password from parameter, file or environment variable. But note that none of these options does not satisfy the security requirements for a passwords!

```
$ sshpass -p passw0rd ssh host
$ sshpass -f /secret/filename ssh host
$ SSHPASS=passw0rd sshpass -e ssh host
```

The command line options can be seen by other users in `ps` (during runtime it is masked, but not during start time and you can't rely on it):

```
... 23624  6216 pts/5    Ss   Aug30   0:00  \_ /bin/bash
... 12812  1988 pts/5    S+   08:50   0:00  |   \_ sshpass -p passw0rd ssh host
... 45008  5796 pts/15   Ss+  08:50   0:00  |       \_ ssh host
```

Note, that environemnet variables of a process are also accessible by other processes on the system using `/proc/PID/environ` file.

Finally, storing the password in the file might look like the best possible idea, but still using keys as described in the other examples is preferred way to use `ssh`.

Config File

OpenSSH config files are used for configuration that should be applied every time the ssh client is run. Most command line options are possible to put in the config files.

OpenSSH uses configuration from the following sources in order:

1. Command line options
2. User's configuration file `~/.ssh/config`
3. System wide configuration file `/etc/ssh/ssh_config`

Configuration options are listed one by one in the config files.

```
# This is a comment.

# Parameter can be specified like this, separated with white space.
StrictHostKeyChecking ask

# Or parameter key and value may be separated with white space and =.
ForwardX11 = yes

# The parameter value can be quoted if it contains white space.
IdentityFile "/file system/path with/white space"
```

The full list of possible config parameters is available here.

One of the most useful features of the config file is that it can be sectioned based on host name or address. In this way you can have different configurations for different hosts.

```
# Based on host name.

Host host1.domain.com
    User user1

Host host2.domain.com
    User user2

# Or wildcard matching name or ip.

Host *elastic-cloud.com 10.201.4.?
    User user3
```

SSH-Agent

If you have set a long passphrase and do not wish to keep entering it every time you want to connect to the server, you can use SSH-Agent to store your passphrase while you are logged in on your computer.

Start the ssh-agent in the background:

```
eval "$(ssh-agent -s)"
# Agent pid 59566
```

And add your key to the ssh-agent, you will be prompted to enter your passphrase:

```
ssh-add ~/.ssh/matrix.ac
# Enter passphrase for /home/osaka/.ssh/matrix.ac:
# Identity added: /home/osaka/.ssh/matrix.ac (/home/osaka/.ssh/matrix.ac)
```

Done. Now connect with `ssh user@matrix.ac` and you should not have to enter your passphrase. You can extend this by using programs like gnome-keyring/seahorse, keychain and other key managers.

Read Getting started with Secure Shell online: https://riptutorial.com/ssh/topic/2070/getting-started-with-secure-shell

Chapter 2: Debugging ssh problems

Examples

Private key is not accepted (OpenSSH clients debug)

By default, most of the information is hidden from the user. You can use `-v` switches to get a verbose log of the connection attempt, which will usually pinpoint the problem by showing why the behavior is different than you expect.

Let's assume you are connecting to the server `example.com` using `ssh` (or other OpenSSH client like `sftp` or `scp`) and your private key is not accepted by the server and the server asks for the password (or rejects the connection):

```
$ ssh example.com
user@example.com's password:
```

Try to run the `ssh` with `-vvv` switches, which will write out all the debug messages. It will be a lot of information, but after some time, it is quite easy to understand that:

```
$ ssh -vvv example.com
```

The most common problem is that the key is not in the expected location. You can expect similar lines to show up, complaining about missing files. Checking that your file was really read is a good start:

```
debug1: identity file /home/username/.ssh/id_dsa type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/username/.ssh/id_dsa-cert type -1
```

We can continue to the authentication part. The key might be offered, but rejected by the server, because of problem with server configuration. The log might look somehow like this:

```
debug3: preferred publickey,keyboard-interactive,password
debug3: authmethod_lookup publickey
debug3: remaining preferred: keyboard-interactive,password
debug3: authmethod_is_enabled publickey
debug1: Next authentication method: publickey
debug1: Offering RSA public key: username@localhost
debug3: send_pubkey_test
debug3: send packet: type 50
debug2: we sent a publickey packet, wait for reply
debug3: receive packet: type 51
debug1: Authentications that can continue: publickey,gssapi-keyex,gssapi-with-mic,password
```

REMOTE HOST IDENTIFICATION HAS CHANGED!

The common error using `ssh` is to see the error like

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
SHA256:L5ri/Xdgpuals893ej1z5F1wlg1n2YNeBf/tsABX+QQ.
Please contact your system administrator.
Add correct host key in /Users/username/.ssh/known_hosts to get rid of this message.
Offending RSA key in /Users/username/.ssh/known_hosts:12
RSA host key for *IP address* has changed and you have requested strict checking.
Host key verification failed.
```

This means that you connected to the same server before and it was identified using different host keys. If you are aware you changed the server keys, reinstalled the server or the server administrator announced some changes, it is usually ok to remove the old key and let the `ssh` to store its new.

The old key can be transparently removed using `ssh-keygen`:

```
ssh-keygen -R *IP address*
```

And next connection should ask you to verify the new fingerprint:

```
ssh192.168.0.128
The authenticity of host '192.168.0.128 (192.168.0.128)' can't be established.
ECDSA key fingerprint is SHA256:L5ri/Xdgpuals893ej1z5F1wlg1n2YNeBf/tsABX+QQ.
Are you sure you want to continue connecting (yes/no)?
```

If you are not aware of any of the above, the best is to contact your server administrator to make sure that everything is ok. If not, the potential attacker would be able to get both your authentication information and all transferred data!

Connection Refused

A "Connection Refused" error will occur if your client sends a connection request to a remote server host, and the remote host responds to say that it refuses to accept the request. The "Connection Refused" error essentially means that the computer is not accepting connections to the requested IP address and port.

"Connection refused" can be caused by a firewall which is blocking connection requests. A firewall which is configured to block connections to a particular endpoint can be set to drop connection requests--in which case the client will never get a response and will eventually timeout. Or the firewall can respond to connection connection requests with a refusal response.

Aside from firewalls, in the case of SSH, "connection refused" has a few possible causes:

- You could be using the wrong port number to connect. The standard port number for SSH is 22, but some people run the ssh service on a different port to deter unauthorized access attempts.

- You could be trying to connect to the wrong computer. You may have mistyped the hostname or IP address. Or the computer may be using a dynamically-assigned address which has changed.

- The ssh server process may not be running:

    - It may not have been started yet if the system is in the process of starting.
    - It may have been disabled; e.g. when the system is in single-user mode.
    - It may have been misconfigured, causing it to fail to start.
    - The computer may not have an SSH server set up. MS Windows systems typically don't include an SSH server. On some Linux systems, the SSH server may be an optional component. OS X includes an SSH server, but it's disabled by default.

- The SSH server process may not be listening for connections on the specific IP interface which you're trying to connect to. Most computers have at least two IP interfaces, a "localhost" interface and one or more network interfaces. Each active interface will have an IP address associated with it. An SSH server is typically configured to accept connections on any IP interface. But it can be configured to accept connections only on particular interfaces. In that case, the computer will refuse connections to an IP address which the SSH server isn't listening to, even if the connection request has the correct port.

- The server may have a backlog of connection requests to the same port. This is rare and unusual, but if the host is receiving connection requests faster than they can be handled, the host will eventually start rejecting new connection requests.

Note that, firewalls aside, "connection refused" means that you *are* communicating with the remote computer--it's just not accepting your connection request.

Connection timed out

A "Connection timed out" error occurs when the remote system does not respond to the client's attempt to open a TCP/IP connection. The most common causes include:

- A firewall is blocking the connection attempt on the port that you are using:
    - The firewall could be on the client-side, blocking outbound connections, on the server-side, blocking inbound connections, or somewhere else on the path.
    - The root problem *could* be that you are using the wrong port number.
- The host you are attempting to connect to could be offline.
- Some part of the network between the client and server could be down or disrupted.
- There could be a network routing problem.

Diagnosing the root cause of connection timeouts is difficult.

ssh_exchange_identification: read: Connection reset by peer

This error message may be produced by the OpenSSH `ssh` client. It means that the TCP connection between the client and the server was abnormally closed by the server immediately after being accepted. Common reasons for this message include:

- The SSH server process is malfunctioning--for example, it crashed.
- A firewall, router, or other network device between the client and the server is interfering with the SSH connection.

The phrases in the error message indicate specifically what has happened:

ssh_exchange_identification: 1. After an SSH client makes a connection to an SSH server, the first step in starting the SSH protocol is for the server to send its software version to the client. An error containing "ssh_exchange_identification" indicates that the error occurred immediatly after making the TCP connection, while the client was waiting for the software version from the server.

Connection reset: A connection reset means the TCP connection was "abnormally closed". You can get this if the software process holding one of the TCP connection crashes. Network firewalls can be configured to use connection resets as a means to block TCP connections.

by peer means that the TCP connection was closed from the "other end" of the connection. In this case, the "other end" is the remote SSH server.

Note that this error doesn't indicate any kind of authentication failure. The server closed the TCP connection *immediately* after accepting it. The client and server have not yet exchanged any data at all. The server has not yet sent a host key to the client, and the client has not yet tried to authenticate with the server.

ssh_exchange_identification: Connection closed by remote host

This error message may be produced by the OpenSSH ssh client. It means that the TCP connection between the client and the server was closed by the server immediately after being accepted. This message generally indicates the SSH server has been configured not to accept connections from the client for some reason:

- The server may be configured to refuse connections from the client's IP address.
- The server may be configured with a limit on the number of active SSH connections.
- The client may have connected to something that's not an SSH server.

The phrases in the error message indicate specifically what has happened:

ssh_exchange_identification: 1. After an SSH client makes a connection to an SSH server, the first step in starting the SSH protocol is for the server to send its software version to the client. An error containing "ssh_exchange_identification" indicates that the error occurred immediately after making the TCP connection, while the client was waiting for the software version from the server.

Connection closed: This means the TCP connection was closed in a normal fashion. It indicates that the SSH server deliberately closed the connection. This is in contrast to "Connection reset", which could indicate that the SSH server crashed or malfunctioned.

by remote host means that the TCP connection was closed from the "other end" of the connection. In this case, the "other end" is the remote SSH server.

Note that this error doesn't indicate any kind of authentication failure. The server closed the TCP

connection *immediately* after accepting it. The client and server have not yet exchanged any data at all. The server has not yet sent a host key to the client, and the client has not yet tried to authenticate with the server.

Read Debugging ssh problems online: https://riptutorial.com/ssh/topic/2926/debugging-ssh-problems

Chapter 3: Kill unresponsive SSH session

Introduction

Sometimes your SSH session stops responding. You can easily get out of that session using the following trick.

Examples

Killing an SSH session using the Escape character

By default, the escape character is `~`.

Just go ahead and type `~.` in your opened SSH session.

After hitting `Enter` your session will end immediately.

Go ahead and try it in any session, it works regardless of the responsiveness of your session.

Read Kill unresponsive SSH session online: https://riptutorial.com/ssh/topic/8797/kill-unresponsive-ssh-session

Chapter 4: Managing remote files through ssh

Introduction

this topic should describe various ways of using ssh as a secure way to manage files on remote machines

Examples

Mounting remote directory through ssh

You can mount remote directory through ssh by using sshfs. Sshfs does not come as a default on Ubuntu, so you need to install it first by using `sudo apt-get install sshfs`. You can then mount the remote directory to your local machine like this

```
sshfs user@xxx.xxx.xxx.xxx:/remotedir /localdir
```

Note that the localdir needs to be present before trying to mount the remotedir. In case that the localdir is not empty, sshfs will complain and abort. You can force it to mount in the non empty dir by using nonempty option like this

```
sshfs -o nonempty user@xxx.xxx.xxx.xxx:/remotedir /localdir
```

Once you umount the localdir, local files will become visible again.

Moving files between servers through ssh

One way to move files between servers is by using the `scp` command. Secure copy command utilizes ssh to transfer data. The simples example for copying a file from local to remote server is

```
scp /localdir/localfile user@xxx.xxx.xxx.xxx:/remotedir/remotefile
```

Similarily, to copy file from a remote to local server would be

```
scp user@xxx.xxx.xxx.xxx:/remotedir/remotefile /localdir/localfile
```

Read Managing remote files through ssh online: https://riptutorial.com/ssh/topic/9693/managing-remote-files-through-ssh

Chapter 5: Remote commands

Examples

Hello World

To send a remote command via SSH (the SSH server needs to be running on the remote host), you can simply write the command after *user@machine*.

```
ssh alice@example.com echo 'Hello World'
```

```
Hello World
```

It returns back the output to the sender, executed on the remote host.

Interactive and screen-based commands

Many commands and programs in the remote side are screen-based (e.g. `mc`) or they need to ask password (e.g. `sudo`), to be able to run these kind of programs you can use option `-t`.

```
ssh -t alice@example.com sudo ls /
```

```
[sudo] password for alice:

bin root dev etc home lib mnt opt proc root run usr var
```

Read Remote commands online: https://riptutorial.com/ssh/topic/3297/remote-commands

Chapter 6: Reverse tunnels

Examples

OpenSSH

Creating a reverse `ssh` tunnel takes just one switch `-R` to the original command.

Command line

Let's assume you are connecting to the `example.com` as a user `guest` using a command `ssh guest@example.com`. Opening reverse tunnel can look like this:

```
ssh -R 2222:localhost:22 guest@example.com
```

It will open a port `2222` on the remote server (loopback interface only) and every connection to this port will be forwarded to your local computer ssh server (port `22`).

This also assumes that you have allowed options `AllowTcpForwarding yes` and `PermitOpen any` in your `sshd_config` on your server. Otherwise it will fail with error

```
open failed: administratively prohibited: open failed
```

If you want to allow the forwarded port to be accessible on other network addresses (than `localhost`), you need additionally to allow `GatewayPorts yes` and use a IP address or hostname or IP):

```
ssh -R 2222:example.com:22 guest@example.com
```

Configuration

Additionally, you can specify your remote port forwarding in your `~/.ssh/config` to avoid typing the same line every time you connect. Good practice might be to set up also alias to the host, which will have this forwarding, if you connect to your host frequently and don't want to initiate the port forwarding every time:

```
Host example.com-R
  Hostname example.com
  User guest
  RemoteForward 2222 localhost:22
```

and then create remote port forwarding simply using `ssh example.com-R`

Running in background

The port forwarding can be simply run in the background using switches `-N` (do not run the remote command, only the forwarding), `-f` (go to background after authentication), `-T` (disable remote TTY allocation). Putting it all together:

```
ssh -NTfR 2222:localhost:22 guest@example.com
```

Read Reverse tunnels online: https://riptutorial.com/ssh/topic/5689/reverse-tunnels

Chapter 7: Working with ssh keys

Examples

Convert PPK (PuTTY key) to OpenSSH format

You might receive from your peer private key in PPK format, which seems it does not work in OpenSSH (command-line `ssh`). The client will be asking for the passphrase, because of OpenSSH bug.

```
$ ssh -i mykey.ppk example.com
Enter passphrase for mykey.ppk:
```

You need to convert the key to OpenSSH format using PuTTYgen (command-line version):

```
puttygen mykey.ppk -o mykey.key -O private-openssh
```

Or in GUI version:

- Open PuttyGen
- Click Load
- Load your private key
- Go to **Conversions**->**Export OpenSSH** and export your private key
- Copy your private key to `~/.ssh/id_rsa`

Source: SO answer, Unix SE answer

Read Working with ssh keys online: https://riptutorial.com/ssh/topic/7861/working-with-ssh-keys

Credits

S. No	Chapters	Contributors
1	Getting started with Secure Shell	Community, dreftymac, Harikrishnan, Jakuje, jalanb, Joe Block, jPlatte, Lernkurve, Marek Skiba, mszymborski, Nasreddine, Osaka, Piotr Dobrysiak, ygram
2	Debugging ssh problems	Jakuje, Joe Block, Kenster, Martin Prikryl, Stephen C
3	Kill unresponsive SSH session	leifg
4	Managing remote files through ssh	user
5	Remote commands	deepmax, Jakuje
6	Reverse tunnels	Jakuje
7	Working with ssh keys	Jakuje