SSH Basics and the SSH Config File

Posted in SSH, Tutorials on February 8, 2016

Secure Shell(SSH) and the FreeBSD OpenSSH server are two things I absolutely miss on a standard Windows Installation. SSH is so well integrated into plenty of Gnu/Linux tools that not having them on Windows is a pain.

ssh starcat

Fortunately, using Putty, Putty tools and Cygwin can help you out to make using Windows a lot easier, if you find Windows annoying as a Linux favoring person.

Table of Contents

SSH Basics

First you're going to use SSH will be probably via a terminal emulator either on Mac, Linux or Putty.

cat hacker
Wait, Let Me Google That For You

SSH is a remote login program.

SSH is used to establish a secure and encrypted connection generally between two computers for executing remote commands on the remote server. It can also be used to make secure connections between computers, such as a tunnel between a database server and a web server.

TLDP.org provides an excellent online book on bash, bash basics and bash scripting. I'm also working my way through it.

You can look up most commands' manuals on a shell terminal with the "MAN" command. Looking up SSH brings you the description, options and examples of how it works.

Simply type "man ssh" into your terminal and you'll find out how advanced SSH really is. You can also read it seems most of the manual pages on linux.die.net. Here is their ssh page.

Example SSH commands and use cases

On the most basic levels, SSH provides you with a secure connection to run commands on a remote computer. Therefore, you need access to the remote computer to set up a remote account with your password.

Usually, you can ask an administrator or your host to do it for you. It really depends on your rights on the remote server.

You specify your credentials on the remote host in "user@remote.host".

Basic example for a home laptop on your LAN:

<pre>ssh homeuser@192.168.0.100</pve>

Note that the commands have different parts:

-ssh: the command you summon

-homeuser: your username on the remote host
-@192.168.0.100: This is the address of the remote computer you want to log in. This can also be a hostname. A hostname is just a human readable name that humans can use to connect to computers. Computers don't care about hostnames, all they want are IP addresses. Computers use DNS resolution to match hostnames(eg. howtogeeks.com or arstechnica.com) to IP addresses(192.168.0.100 or 173.252.120.68).

Then SSH will create a connection to your home laptop. An IP address is used in the above example(192.168.0.100 or 69.163.153.176) to tell SSH where to connect to.

On establishing the connection, your SSH client will be asked for a password. Simply enter it, and you'll be logged in.

Using a RSA SSH key file for authentication

If you're still new to the Linux world, you're going to find many opinions on using passwords for SSH authentication is a bad idea. Why?

Because Linux servers with OpenSSH on the internet are constantly attacked by automated malicious bots looking to exploit system resources on servers and vulnerable computers.

People tend to have to use really weak passwords and silly usernames. I watched my authentication log and that gives you a sense of what usernames to avoid.

I've collected some examples for you in a file. You'll want to avoid such usernames. These have been collected and sorted from <pre>sudo lastb | cut -c1-9 | sort | uniq</pre> command. It's not an exhaustive list and only ~9 characters long.

Here's the example:

<pre>ssh -i id_rsa homeuser@192.168.0.100</pre>

The "-i id_rsa" flag tells ssh to use id_rsa as a private key on the remote computer.

Here is what an id_rsa key looks like. Also, here's what its public key looks like. These keys will never be used by me, so feel free to study them, but don't use them.

In order for key authentication to work, the homeuser on 192.168.0.100 will need to have the corresponding public key for the "id_rsa" private key file you specified.

You can replace id_rsa with anything you want, you just need to tell ssh where to find it. An ideal folder is your home .ssh folder in "~/.ssh/id_rsa".

You can create ssh-keys on Debian with:

<pre>
user@debian:~/tmp$ ssh-keygen -t rsa -b 4096 -f mykey
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in mykey.
Your public key has been saved in mykey.pub.
The key fingerprint is:
SHA256:Q3hWS4K3jL0DrI62pSpK6FCaWRR67CcM3aLgTvA5SgQ user@debian
The key's randomart image is:
+---[RSA 4096]----+
|E .   .. o |
|.+ o ...+ . |
|=.* ...=+.. |
|=O o +++ |
|.+X .. .S. |
|=B +.   o. |
|Bo o.   . |
|= oo. |
|=+o. |
+----[SHA256]-----+

</pre>

ssh-keygen is the command used. The -t flag tells it what kind of key to make(available on debian: [-t dsa | ecdsa | ed25519 | rsa | rsa1]). The -b tells how many bits should be in the key. I create 4096 bit keys, but the standard is 2048 and is considered to be secure. There's said to be no reason to create larger than 2048 bit keys by these sources. The -f flag tells ssh-keygen where to store the key. It can be a simple name, but then it will be stored in your current working directory(find it with "pwd").

You should choose a stong password you can remember. The above key link shows what's an unencrypted private key like.

You generally shouldn't use passwordless ssh keys to log in to important machines as anyone getting a copy of your key file will be able to use it to log in to your servers.

I admit that I use passwordless ssh keys to automate backup tasks between servers as I just don't know how you can decrypt automatically your keyfile on SSH based file transfers, such as rsync transfers.

On windows, you can use puttygen to generate your public/private key pair. It's also great to convert your key pairs from Putty keys to Openssh keys. Many windows software that implements SSH or SFTP relies on and uses putty's .ppk formats, like WinSCP.

SSH Verbose Connection

If you're making new changes to your ssh configuration file, are connecting to new hosts or just having issues connecting to other computers, ssh offers debugging functionality with the -v flag.

Example:
<pre>ssh -v homeuser@192.168.0.150</pre>

This will print out what ssh does and where it meets issues. My most frequent issues were:

- the private key path was wrong(file specified doesn't exist)
- wrong port
- wrong host or IP(Yes, it happens! Once I had mistyped an IP address(178 instead of 172 and complained at my host why it's not working)
- wrong username
- root cannot login(Oops, I disabled root login.)
- Connection errors, PATH errors or file not found errors on remote hosts when using rrsync(Restricted Rsync, excellent guide here.)

Further examples will be provided below.

SSH Configuration File and Tweaking It

The SSH configuration is my favorite when it comes to ssh. Initially, it doesn't exist. But SSH knows that by default the configuration can be found in ~/.ssh/ folder as config file(~/.ssh/config).

Organizing the SSH config file

Now the nice part about it is that you can comment it all the way you want. Want to remind yourself later what is that server doing there? Oh yeah, that's the company's linux email server.

Your comment should start with "#".

Example:

<pre>
#
These are my log in details for my workplace
#
</pre>

Everything after a "#" on a line is ignored.

Therefore, you can create nice blocks for each server you have, even noting down their hostname and probably their IP-s if you care.

My current config file has 718 lines. Most of them are comments though for readability and organization.

Working with your SSH config file

You can simply open up and/or create your ssh config file right. I prefer mousepad and nano on debian. You can call mousepad from the terminal as well:

<pre>mousepad ~/.ssh/config</pre>

The SSH configuration file is read(if it exists) when ssh makes a connection. It parses the config file and uses the details you've specified.

Okay, but what's the point?

Writing, typing out and generally remembering what the hostname, IP, username and private key path of a remote host connection when you've got more than a few servers is just something you want to avoid.

Instead, you can collect these attributes inside the config file and just simply call them whatever you want. Work, home, library, wifes computer?

It's built up by what I call Host blocks. Each host has a "Hostname" or IP(this is their address, hostnames are better.). A "User" attribute, an "Identityfile /home/user/.ssh/id_rsa" flag to specify a key to be used. And last a "Port" to specify which port should be used.

Yours could look like this:

<pre>

Host work
Hostname ssh.myworkplace.com
User alexsmith
Identityfile /home/alex/.ssh/id_rsa_work
Port 22

Host home
Hostname justasubdomain.mydomain.com
User alex
Identityfile /home/alex/.ssh/id_rsa_home
Port 40000

Host library
Hostname ourlibrary.org
User randomuser9876
Identityfile /home/alex/.ssh/id_rsa_libr
Port 15000

Host wifes
Hostname 192.168.0.50
User alex
Identityfile /home/alex/.ssh/id_rsa_wife
Port 22

</pre>

After setting your own servers' details up in your ssh config file, now you can finally use it! I recommend debugging(with the -v flag) the connection to see if the ssh private key file is found and that there's nothing wrong.

```
ssh -v work
```

or later just ssh work.

And that's it! No more:

```
ssh -i "~/.ssh/id_rsa_work" alexsmith@ssh.myworkplace.com -p 22
```

Remote command execution

Now that you've got your ssh connections simplified, what if you just want to run a static command every time?

It's possible. Example:

```
ssh work "bash ~/backup-script.sh"
```

This will run

```
bash ~/backup-script.sh
```

on the remote computer and close the connection.

You can honestly do whatever you want.

Sudo apt-get update or upgrade or install a new package.

I'd recommend automating update with cron daily or weekly and upgrade every once in a while. If it's a critical system, read up on the change logs of the packages that are to be upgraded.

Sometimes packages change and file change places. Or libraries will start to work differently.

One example was dist-upgrading debian jessie to debian testing(9). Nano(the cli editor) stopped working. I had to make a symlink to the new location so that nano from the cli can still work.

Additional values to be added:

I've had issues with ssh time outing more often than I like, so I found these values to be extremely good:

<pre>

Host *
ServerAliveInterval 30
ServerAliveCountMax 30

</pre>

ServerAliveInterval: http://unix.stackexchange.com/questions/3026/what-do-the-options-serveraliveinterval-and-clientaliveinterval-in-sshd-conf

ServerAliveCountMax: http://linux.ioerror.us/2013/09/serveralivecountmax-in-ssh/

You can also specify Ciphers that ssh should use. Example:

<pre>

Ciphers blowfish-cbc,aes128-ctr,aes192-ctr,aes256-ctr,arcfour256,arcfour128,aes128-cbc,3des-cbc

</pre>

SSH tunneling inside

SSH tunneling is one of those extremely useful and safe ways to bypass networking issues, limitations and often firewalls.

Travel around much? You know about those gas station and hotel "free" wifi right? Let's just say it's easy to bypass their silly pay-walls and possibly any tracking javascript they might otherwise infect into your browser.

Example of a Host block in the SSH config file:

```
#Example Port forward
#Host sshtunnel
#    HostName webserver.example.com
#    IdentityFile ~/.ssh/bob.example.key
#    LocalForward 9906 127.0.0.1:80
#    User bob
```

When connecting with

```
ssh sshtunnel
```

, a local port will be forwarded to the remote host's port 80. Thus, for example, you could go to your browser at http://localhost:9906 and it'd connect to the remote website which is listening on the remote server at 127.0.0.1:80.

Further reading on SSH tunnels:

This redditor has 4 excellent posts on ssh tunnels I've still got to read:

1. http://www.everythingcli.org/ssh-tunnelling-for-fun-and-profit-local-vs-remote/
2. http://www.everythingcli.org/ssh-tunnelling-for-fun-and-profit-tunnel-options/
3. http://www.everythingcli.org/ssh-tunnelling-for-fun-and-profit-autossh/
4. http://www.everythingcli.org/ssh-tunnelling-for-fun-and-profit-ssh-config/

Share This Page With Your Friends!

Tags: ssh, ssh basics, ssh config

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

POST COMMENT

About me

- About me
- Contact

Legal

- Legal