

SSH Tutorial: UNIX Tutorial

Last Updated Monday, 08 January 2007

SSH Tutorial

Authors: Scott D. Brown

This is my cheat-sheet for using OpenSSH. It describes how to use keys and the authorization agent. This is largely applicable to the Center UNIX machines and the Max OS/X machines (since they are really UNIX too). If you use CygWin (shell or X-Windows) on a Windows machine, this is also applicable.

1. Overview

What is SSH? SSH stands for Secure Shell. Why do I care about using SSH? The primary reason is that SSH encrypts all of your communications between two machines. Why do you want your communications encrypted? Because a hacker could intercept unencrypted communications and steal information including any passwords that you type over that connection. Although your account may not contain any information that a hacker would want, access to your account provides a hacker with another location from which to stage attacks on more desirable targets.

SSH provides two levels of protection. The first is that it encrypts your communications. In this capacity, SSH can replace programs like rlogin and telnet by giving the user a way to type passwords and other sensitive pieces of information over an unencrypted channel in a way that would be near impossible to every decrypt. The second is that SSH also has mechanisms to track the identity of remote machines (using a piece of information analogous to "fingerprints") so that a hacker cannot "spoof" a remote machine and convince you to type your password into a bogus program on a bogus machine.

The other valuable capability that SSH provides is a simplified method for performing remote logins. Rather than typing passwords for each machine, SSH uses the concept of a "keyring" like the one in your pocket. You are allowed to make "keys" of various types and then configure the "locks" on different hosts to enable and disable which "keys" can be used. The keys can be added to a "keyring" that can be used to try the "lock" on a remote machine you wish to access. This use of a "keyring" means that you don't need to manually try each key but rather supply the ring.

SSH supports the use of "Public-key encryption" as one of the authorization methods. Public-key encryption (also called asymmetric encryption) involves a pair of keys (a "public" key and a "private" key) associated with an entity that needs to authenticate its identity electronically or to sign or encrypt data. Each public key is published, and the corresponding private key is kept secret. Data encrypted

with your public key can be decrypted only with your private key. In general, to send encrypted data to someone, you encrypt the data with that person's public key, and the person receiving the encrypted data decrypts it with the corresponding private key.

SSH uses this concept to authorize access to computers. As you will see, you will make a private key on a local machine and put the corresponding public key on a remote machine. When you attempt to remotely login from your local machine, SSH will send an encrypted message generated with your private key to the remote machine which will be able to verify it with your public key. If you have the correct key, SSH will allow you to log into the remote machine without having to type your password.

Within SSH, the authorization keys we are discussing are in fact a keypair. Each "key" consists of a public and private portion which are stored in separate files. We will talk about how to create them and where they are stored in the following sections.

2. Making your SSH directory

Most users will already have a directory under their home directory called ".ssh". This is the directory where both private and public keys are stored and some additional files. Since this directory has your keys in it and you want to keep them private, it is important that you keep the permissions on this directory restricted. By default, most users will have this directory readable by all users or users in the same group. If your .ssh directory permissions look like:

```
$ ls -ld ~/.ssh
```

```
drwx--x--x 2 sdbpci 8192 Oct 21 12:39 /cis/staff/sdbpci/.ssh
```

or

```
$ ls -ld ~/.ssh
```

```
drwx--x--- 2 sdbpci 8192 Oct 21 12:39 /cis/staff/sdbpci/.ssh
```

then you need to change the permissions. Use the following syntax to the `chmod` command to change the permissions so that you and only you have read, write and execute permissions for this directory.

```
$ chmod 700 ~/.ssh
```

```
$ ls -ld ~/.ssh
```

```
drwx----- 2 sdbpci 8192 Oct 21 12:39 /cis/staff/sdbpci/.ssh
```

3. Making keys

You make keys using the "ssh-keygen" program. When you run this program it will automatically generate the random sequence used to encode the key. The program requires at least two arguments: (1) the type of encryption used to make the key and (2) how big the key is.

To make a key, it is easiest to change directories to the .ssh directory and then run the keygen program:

```
$ cd ~/.ssh
```

When making a key, I suggest that you use the DSA encryption algorithm and a 2048 bit key. To make a key with these attributes, run the following command:

```
$ ssh-keygen -t dsa -b 2048
```

Generating public/private dsa key pair.

Enter file in which to save the key (/cis/staff/sdbpci/.ssh/id_dsa): cis

The program will run for a little while it generates a key pair. It will then ask you for the name of the file to store the generated key in. The key I use to go from UNIX machine to UNIX machine within the CIS building I called "cis". You can call it whatever you want.

The program will then ask for the passphrase for the key. The passphrase is like a password - it should be 10-30 characters, contain a mix of cases, some numbers, and it should not contain common words or phrases, etc. But, at the same time you will need to remember this passphrase since you will need to supply it whenever you add it to your keyring. Take the same consideration when choosing a passphrase as you would your password. It is not a good idea to use the same passphrase as your password because if a hacker gets a hold of your password, then they can also add your key to their keyring and log into any machine you have authorized this key for. Take advantage of the fact that your passphrase can be longer and hence maybe easier to remember. You will be asked to repeat the passphrase to verify the spelling, etc.

After you type the passphrase, the program will exit. The output of this program will be a keypair stored in two separate files. In this example, the name I supplied the ssh-keygen program was "cis" so the filename for the private key will be "cis" and "cis.pub" for the public key (see the filename "cis" I typed after the "Enter file in which to save the key" prompt from the "ssh-keygen" program). By the way, the SSH key files are just ASCII files so you can open them up in an editor and look at them if you want. However, do not ever change the format or values of these files otherwise you will corrupt them and they will become useless.

Choosing good names for your keys will help keep things organized. You can always rename the key files just remember to rename both of them. For example, my home machine is name "offside" and runs Windows (with the Cygwin UNIX emulator) and Linux. I have a different set of DSA keys for that machine depending on whether I booted into Windows or Linux. They are named "offside_win_dsa" and "offside_linux_dsa" respectively (plus the associated public key files).

How many keys should you make? That depends of course. The model that I am using is a key for all hardwired, Center UNIX machines, another I made on my Mac Notebook, another I made on my LINUX machine at home, etc. When we see how to distribute and authorize keys, this will become clearer.

You will only need to create a key once for each machine or set of machines you use. For example, since all the Center UNIX machines have the same directories on them (including your .ssh directory), you only need 1 key for all the Suns in the building. However, you will need one for each home machine or notebook you might use.

4. Running the Authentication Agent

An authentication agent is a program that helps manage your keys for you (it is the "keyring"). You run it on your client machine (the machine you are currently on) to supply authentication keys for the remote machine. The SSH authentication agent is a program called "ssh-agent". When you start the agent, it will print to stdout a series of commands that are to be executed by your shell. The format of the commands can be controlled using the -c and -s options. If you are using csh (or tcsh), you can run the command as follows:

```
$ ssh-agent -c $SHELL
```

What this will do is start a new shell with the agent running in the background and the appropriate environment variables set. If your shell is tcsh(1), you might have problems on some machines. The ideal process to start the agent is the first process created when you log in. For example, your shell when you log in via console or the primary process starting your X-Windows session. To start one with each login shell, you could place commands in your

.login file. For an X-Windows login, you might start it in the .xsession or .xinitrc file.

Most of the user accounts in the Center have this command somewhere in the default login files. If an ssh-agent is already running you will be able to tell because there are some environment variables in your shell defined. You can check this with the following command:

```
$ setenv | grep SSH  
SSH_AGENT_PID=29970  
SSH_AUTH_SOCK=/tmp/ssh-5JfCvXcV/agent.29969
```

```
$
```

If you get the following, then an agent is not probably running:

```
$ setenv | grep SSH
```

```
$
```

Chances are if an agent is running for you now, it will always be running when you log in.

5. Adding a key to the keyring.

The authentication agent manages your keyring. You add keys to your ring by running the command "ssh-add". If you run "ssh-add" with the "-l" option it will list the keys you have added to your agent:

```
$ ssh-add -l
```

The agent has no identities.

If an agent is not running, you will get an error message indicating so.

To add a key to the agent, run the "ssh-add" with the name of the private key file. For example, I generated a key called "cis" before that I want to load now :

```
$ ssh-add ~/.ssh/cis
```

You will then be prompted for the passphrase for this key. If you correctly enter the passphrase, the agent will add it to the keyring. If not, it will not be added.

If you now ask "ssh-add" to list the keys again, you should see something like this:

```
$ ssh-add -l
```

```
2048 cb:d5:62:42:59:84:a4:95:db:dd:21:3d:5f:5f:3e:7b /cis/staff/sdbpci/.ssh/cis  
(DSA)
```

The exact names and values will differ, but you should not get the same "The agent has no identities." message that you got before.

You can add more and more keys to the keyring using the "ssh-add" command and you can remove them if you like also (read the "ssh-add" man page). However, for most people within the Center, you will only need 1 or 2 keys on your ring ever.

The process of adding a key to you ring is one that will be repeated any time you log into a machine. If you log out, you ring gets destroyed.

6. Allowing public key authentication

Now you have at least one key, and you have an agent running that has that key on its ring. What can you do with it? Ideally, you use it to allow secure logins across the network. The most common use is to allow you to log in to a remote machine without providing a password (or passphrase). To do this, we need to configure a remote machine to use the SSH keys correctly.

Access to a remote machine is controlled by a file called "authorized_keys" which appears in your ".ssh" directory. This file contains public keys that are authorized to allow logins to this machine. Assuming this is your first key, this file most likely doesn't exist. To make an empty authorized key file type the following command:

```
$ cd ~/.ssh
```

```
$ touch authorized_keys
```

This will create an empty "authorized_keys" files. To add a key to the "authorized_keys" file, we need a copy of a public key. On the Center UNIX machines, your public key is already in this directory because it is the same directory on every machine. If you are configuring a machine at home, or your laptop, then you will need to FTP your key over to the directory. Let's assume that I did this and I have the public key file "cis.pub" in my .ssh directory. To add it to the "authorized_keys" file I can just use the following command:

```
$ cat cis.pub >> authorized_keys
```

This command will tack my "cis.pub" key onto the end of the "authorized_keys" file (note the append redirection with two ">" signs). If you want to add more keys just repeat the command with the name of another public key file.

When you are done, your "authorized_keys" file might look something like this where mine contains only 1 key that I (sdbpci) generated on tablesaw.cis.rit.edu.

```
$ cat authorized_keys
```

```
ssh-dss
```

```
AAAAB3NzaC1kc3MAAACBAPOi+2paXGkQMZ/9nN66oMaz57S0BEktnPsh5iJVfAQsngvbmXOGB5IKj  
D17AIRRjESAvPB0FYj6iJkiULTWE7ueGooR3bYWhM+0ZI/aV2QYVQmC14w7crVuyuim+5IJ9p00wR  
3qMhSwq4BK5At79xtvSjkJ1rF1sB5auuc5u93XAAAFQCBAAdNWCUzIKKjD6rgVkwPcqj5NewAAAIE  
AmDoTdx37nVMnXovNzjsmalhxM7MAGdAq6aLmVHhycE1/t4tuYUs+5RtB7e1CE5E4VauNCz1PLfJk  
6nUID5CkXMKil46t+qOeez7CqxIE7JKHPeC3ci4UOyHZiUZ63oPQ4EqSwrRxKT0j7HBDBk4v2VEdR  
zyeDFDwTkecOz1KwTMAAACBALfmfm3O2aISjSXSjOm+eZ8x8aq8LsVGDrPLEoylKMM2AnMx1JvOj3  
CbFp7zMLBeK4i56EGbqpNm5Fivyeq2ZZhFpPgFLNc0VRw/ZC/Rw/oaxyUj0rGQev+X9p18MDyR4km  
zjuTPdb9OhO/WOxOZArQ+zZKGAkJRt2iqff8TnZWusdbpci@tablesaw.cis.rit.edu
```

Once this key has been placed in the "authorized_keys" file, it is authorized until you delete it from this file (it is just an ASCII file you can edit in any editor to remove a key).

At this point, you are done. You have learned how to create a key, start up a keyring manager (ssh-agent), add a key to your ring, and authorize a key. Now what?

7. Using keys to perform remote logins as yourself

So we want to log into "goes" and I am currently logged into "drillpress" First thing I do is check if I have my "cis" key on my ring:

```
$ ssh-add -l
```

```
2048 cb:d5:62:42:59:84:a4:95:db:dd:21:3d:5f:5f:3e:7b /cis/staff/sdbpci/.ssh/cis (DSA)
```

Looks good. Now I can use the "ssh" command to log into "goes":

```
$ ssh goes.cis.rit.edu
```

```
Chester F. Carlson Center for Imaging Science
```

```
Rochester Institute of Technology
```

```
Node: goes
```

```
$
```

See how easy that was? No password.

If it is the first time you have logged into a particular machine, you might get the following prompt: The authenticity of host 'goes.cis.rit.edu (129.21.58.120)' can't be established.

RSA key fingerprint is b2:50:1f:e3:c0:ae:65:05:99:f3:de:b1:19:56:7b:fc.

Are you sure you want to continue connecting (yes/no)?

Type "yes" and you will be logged in. This warning is another level of security with SSH. It collects fingerprints of every machine. If you log into "goes.cis.rit.edu" next week and the fingerprint is different, you will get a warning. That should never happen in the Center. If it does, it might mean some hacker is trying to get you to log into a different machine that they named "goes".

8. Using keys to perform remote logins as a different user

We have a small number of accounts that we use for common tasks. The most common is the "dirsig" account. So how do I use SSH to allow me to log into "goes" as the user "dirsig" instead? Well, it is pretty easy.

A. Supply the owner of that account your public key file. In the case of the "dirsig" account, that is me. I will add your public key to the "authorized_keys" file in the ~dirsig/.ssh directory. Ideally, you would make a second keypair for this purpose. You could call it "dirsig" for example. You would only add this key to your ring when you need to log into the "dirsig" account.

B. To log into "goes" as the user "dirsig", you just change the "ssh" syntax slightly:

```
$ ssh dirsig@goes.cis.rit.edu
```

Chester F. Carlson Center for Imaging Science

Rochester Institute of Technology

Node: goes

dirsig[1]:

The following (old "rlogin" style) also works:

```
$ ssh goes.cis.rit.edu -l dirsig
```

Notice a few things using this approach instead of the "su" or "rlogin" commands. First, the only password you ever type is your passphrase for your SSH key. Once your key is on the ring you can login until the cows come home. Second, you do not know the actual password to the "dirsig" account itself. If I need to, that password can be changed without any effect on you and your key. Third, if we need to stop allowing access of a particular user into the "dirsig" account, all I have to do is delete their public key from the "authorized_keys" file and they can't get in anymore.

9. Using Secure Copy (scp) instead of FTP

The ssh tools also provide an easy way to copy files around from machine to machine. Again, in the Center all the UNIX machines have shared user directories, but if you have a home machine and you want to copy a file to it, this method would be applicable.

Let's assume you want to copy a file from a remote machine to the current directory on the machine you are currently logged into. The name of the file on the remote machine is "test1.dat" and it is in the directory "pub/test_data" under your home directory. To copy this remote file (on the machine "tablesaw.cis.rit.edu") to your current directory, use the following command:

```
$ scp tablesaw.cis.rit.edu:pub/test_data/test1.dat .
```

```
test1.dat 100% 5688 136.3KB/s 00:00
```

This will make a copy of that file in your current directory with the name "test1.dat". If you want the local copy to have a different filename, then just replace the "." argument at the end of the command with the filename that you want the local copy to have.

In general, the "scp" command has the same syntax as the standard "cp" command. The only difference is the ability to specify name of the remote machine in the filename, separated by the ":" character.

Of course you can also copy a local file over to the remote machine. In this example, I want to put a copy of "test2.dat" onto the machine "tablesaw":

```
$ scp test2.dat tablesaw.cis.rit.edu:pub/test_data
```

```
test2.dat 100% 5688 136.3KB/s 00:00
```

If you log into "tablesaw" and look in the "pub/test_data" directory, you should find a copy of "test2.dat"

[end]