Help Topic: SSH Keys

Spring 2020

Michael Benjamin, mikerb@mit.edu
Department of Mechanical Engineering
MIT, Cambridge MA 02139

SSH Keys

Normally when using the secure shell utility, `ssh`, you will be prompted for your password on each interaction. While secure, this can cumbersome if such events are more frequent than say once per half hour. Certain tools, like SVN, use `ssh` as underlying utility to handle authentication. This can also lead frequent entry of your password, perhaps several times within a minute, depending on your task.

The `ssh` utility supports something called *ssh keys* to help reduce the need for frequent entry of your password, while maintaining the security of user authentication. It works by generating a public and private key pair, with one part of the pair stored on your target machine, and one part stored on your local machine. You authenticate on your local machine once, and the following ssh interactions use this key pair for authentication thereafter. This setup needs to be done once for each machine you intend to interact with regularly.

If you are interested in setting up ssh keys for gitHub, check out Section 3.2.2 in
https://oceanai.mit.edu/2.680/pmwiki/pmwiki.php?n=Lab.ClassSkillsGit

Let's get started.

Generating a Key Pair

The first step is to generate a public-private key pair. On your local machine:

```
$ ssh-keygen -t rsa -f ~/.ssh/id_rsa
```

You will be prompted (twice) for a passphrase. Choose this with the same care you would for new password for yourself.

This will create a directory locally, if you don't already have it, in your home directory called `.ssh/`. Like all hidden files or directories, you need to use the `-a` option, `ls -a`, to see the directory. The `-f /.ssh/id_rsa` option explicitly names the directory and filename of the private key. If you left this unspecified, you may be prompted for the file name, with `id_rsa` being the default value anyway. Afterwards, things should look like:

```
$ cd ~/.ssh
$ ls
id_rsa  id_rsa.pub
```

In the next step we will move the public key file, `id_rsa.pub` to the target machine. The private key file, `id_rsa`, will remain on your local machine.

Moving the Public Key to the Target Machine

The next step is to move public half of the key pair to the target machine. For most of us in the lab, the target machine is typically the oceanai server, but it can also include any or all of the robots.

The first step is to log on to the target machine and create a `.ssh/` directory in your home directory, if it doesn't already exist. (`cd; mkdir .ssh`). It may be there already if you've ever invoked `ssh` from your target machine account. This directory needs to exist before the next step, which puts the public key file there. Now that the target *location* exists, you're ready for the next step. The next step is to copy the public key to the target machine. For example, back on your local machine, using the username `joe`:

```
$ scp ~/.ssh/id_rsa.pub joe@oceanai.mit.edu:~/.ssh/authorized_keys
```

Note this file on the target machine, `authorized_keys`, contains initially one public key, looking something like:

```
ssh-rsa AAAADSDFB3NzaC1SDFkc3FMAFSDFSDFAACBAIr857SDFSDFeJ27zYZsdfs+MLemsdfsn
tHZv14dSHH14ynlLgDBVTafMA+DxPsdfsdfssdfwFezsUV/PmsdfsTasLdfLsEfgJgs2dfAsGdf7
sdfsdfwsdrfPsdfDstfIsqfja/PKJKJKJ+mG7MD3+44sdxEsdHzsdflsdfUFFDWELFASDFDSSFSK
JTYHFHFfsdfswKspdxewrfekYGAHTJu7XpXeFOTAjlkB5vRTMUC7   joe@joes_home_machine
```

This sets up access for one particular username@machine. This file on the target machine may also support different users, or different machines, or both. Each user/machine will have a separate line in the `authorized_keys` file.

Things to Note if Setting Up Access for Multiple Machines

IMPORTANT: If you are setting up access for more than one user/machine, beware that when you execute the `scp` command above that you will *overwrite* the `authorized_keys` file. Instead you should `scp` the `id_rsa.pub` file to a temporary file of a different name. For example:

```
$ scp ~/.ssh/id_rsa.pub joe@oceanai.mit.edu:~/.ssh/authorized_keys_tmp
```

After this, you will need to log on to the target machine and then append the temporary file (presumably with only one line) onto the existing `authorized_keys` file.

```
(on the target machine)
$ cd .ssh/
$ ls
authorized_keys  authorized_keys_tmp
$ cat authorized_keys_tmp >> authorized_keys
$ ls
authorized_keys  authorized_keys_tmp
$ rm -f authorized_keys_tmp
$ ls
authorized_keys
```

When all is done you can remove the `is_rsa.pub` file on your local machine.

Security - Configuring ssh keys to Expire

Some people don't like the idea that, once the ssh key password has been entered locally, it it remains valid indefinitely. What if someone gained access to your computer somehow? Couldn't they then also do further damage by logging into additional computer accounts of yours, using your ssh key previously authenticated? If this scenario haunts you, but you still want the conveniences of ssh keys, consider having them expire after a duration of time to your liking. After all, most interactions with remote machines, for say version control for example, happen in bunches. So even if you set your ssh keys to expire after five minutes, it could still save you from typing your password many times. Plus, five minutes is an awefully short time for a thief to figure out what to do with your computer. Personally I choose a timeframe of something like two hours.

The below instructions are for MacOS. The first step is to edit the following file:

```
/System/Library/LaunchAgents/org.openbsd.ssh-agent.plist
```

Look for the block of text looking like:

```
<key>ProgramArguments</key>
<array>
    <string>/usr/bin/ssh-agent</string>
    <string>-l</string>
</array>
```

Add the below indicated two lines, using whatever interval of time you choose. The below duration is 30 minutes (`30m`), you can also enter this in seconds hours, or days (`120s`, or `8h`, or `5d`).

```
<key>ProgramArguments</key>
<array>
    <string>/usr/bin/ssh-agent</string>
    <string>-l</string>
    <string>-t</string>                        <-- add this line
    <string>30m</string>                       <-- and this line
</array>
```

Then reboot. There may be ways around this, but apparently this is easiest.

The above tip is from Stack Exchange:
http://apple.stackexchange.com/questions/118223/

Controlling SSH Key Expiration from the Command Line

On MacOS, typically you will be prompted with a pop-up dialog window prompting you for your ssh key password if it has not been set. This occurs automatically if you invoke `ssh`, or a command like `svn` that uses ssh. You can also "load your identity" from the command line using the `ssh-add` command. This command can also be used to delete your identity, or control the expiration time of your identity.

For example, if you don't want to wait for your identity to time-out, you can do the following:

```
$ ssh-add -d              <-- deletes your ssh key identiy now
```

You can also add your ssh key (add your identity) from the command line instead of waiting for the pop-up window prompt. By using the `-t` option you can specify the timeout (in seconds) from the command line:

```
$ ssh-add -t 1800          <-- identity will time out in 1/2 hour
```

So if you normally have your identity never time out, or only after a very long time, and you want to change it temorarily, try the below:

```
$ ssh-add -d
$ ssh-add -t 1800
```

Run `ssh-add -h` to see more options for this command.