A guide to using git with SSH

IN5320

Introduction

Git is a version control system for tracking changes to source code. You work with it by fetching, committing, and pushing changes to a repository. Depending on how the repository is set-up, you might need to authenticate yourself when performing some of these actions. By default, git will prompt you for a username and password when this is the case. In the long run, this might become quite a cumbersome task.

An alternative way of authenticating yourself is to use an SSH-key. By setting up the keys and configuring your local repository to use SSH, all authenticated actions can be performed without needing to specify your username and password. Additionally, if no passphrase is supplied when generating the SSH-key, then the authentication process can happen seemlessly in the background.

What is an SSH-key?

An SSH-key is actually a pair of keys: a public key; and a private key. They are like a keyhole and its matching key. By handing someone your public key, you will later be able to prove your identity to them—authenticate yourself—using your private key. Note that you should *never* expose your private key to anyone.

How to generate an SSH-key

GitHub has a great guide on how to generate an SSH-key pair. In short, you run a single command in the terminal[1]:

```
ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
```

It will prompt you for a location to save the keys. Just press ENTER to accept the default location. Next, you'll be asked to enter a passphrase. This one is strictly not required, and you can leave it empty[2].

The program should generate two files for you: `id_rsa.pub`, the public key; and `id_rsa`, the private key. They should both be located in the `~/.ssh` directory, if you didn't change the location.

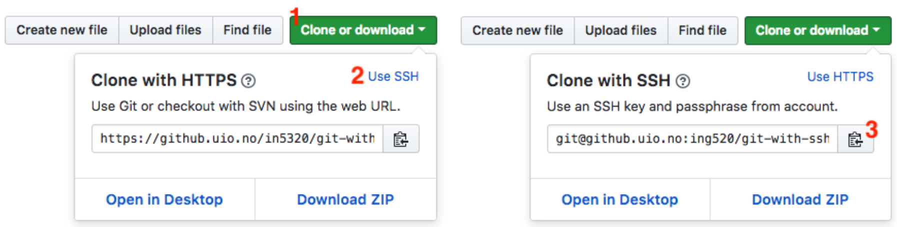
[1]Windows users will probably have to run the command in GitBash, which comes installed with git for Windows.

[2]If you don't leave it empty, git will prompt you for the passphrase on every authenticated action. An SSH-key without a passphrase is still secure, as long as your private key remains a secret.

How to use your SSH-key

Since we'll be working with the UiO version of GitHub, you'll need to log in and upload your public key there. Again, GitHub has a guide on how to add a new SSH key to your GitHub account. You can follow the instructions exactly—just make sure you carry them out on UiO GitHub instead.

Finally, you'll have to configure your local repository to use SSH. The easiest way to do this is just to clone the repository using an SSH link. On UiO GitHub, you can find this link by clicking on the green **Clone or download** button on a repository site, followed by the link **Use SSH** if it's not already the selected option.



Use Git or checkout with SVN using the web URL.

`https://github.uio.no/in5320/git-with`

Use an SSH key and passphrase from account.

`git@github.uio.no:ing520/git-with-ssh`

Figure 1: Screenshots illustrating the steps for cloning a repository with SSH.

What if you already have an SSH-key for GitHub?

Great, then you can skip the generation process and reuse that key. Or if you don't want to reuse the same key[3], it is also possible to configure your SSH agent to use different keys for different hosts. You can do that by following these steps:

1. Navigate inside the `.ssh` folder located in your home directory:

```
cd ~/.ssh
```

2. Create a file named `config` if it does not already exist:

```
touch config
```

3. Open the just created file in any text editor.

4. Add the following content:

```
Host github.com
    IdentityFile ~/.ssh/github

Host github.uio.no
    IdentityFile ~/.ssh/uio-github
```

[3] A possible security concern is that if someone gets a hold of your private key, then they will

5. Modify the `IdentityFile` paths to point to your own private keys.
6. Save the file and exit the text editor.

With the above configuration file, git should be able to automatically use the correct SSH-key for both normal GitHub and UiO GitHub[4].

have access to all of the accounts associated with that key.

[4]Actually, git just delegates the task to your SSH agent, which then figures out which key to use.