How to set up SSH on your Bitbucket account

1 Introduction

Recently, you may have seen the following when pushing and pulling from your Bitbucket account:

```
remote: You are using an account password for Git over HTTPS.
remote: Beginning March 1, 2022, users are required to use app passwords
remote: for Git over HTTPS.
remote: To avoid any disruptions, change the password used in your Git
    client
remote: to an app password.
remote: Note, these credentials may have been automatically stored in your
     Git client
remote: and/or a credential manager such as Git Credential Manager (GCM).
remote: More details:
remote: https://bitbucket.org/blog/deprecating-atlassian-account-password
-for-bitbucket-api-and-git-activity
```

Now this doesn't necessarily (nothing will break), just that when this is enabled it will make pushing/pulling more difficult. Bitbucket is just "increasing" security of cloned repositories over `https` (the quotations because security isn't actually increased). What they're doing is requiring you to have a repository specific password that you enter every time you push and pull from your remote master branch. These passwords will be generated from your Bitbucket account online, if you have a Github account then it is similar to the personal access tokens they require.

The benefit of these repository-specific passwords is being able to control exactly what someone contributing to a repository is able to do. More so than the read/write/admin permissions you can grant contributors. Really though, `SSH` keys are the most secure, the additional benefit of learning how to set one up is that it's how (almost) every company makes their employees set up their git accounts on the companies' repository.

2 One SSH key

If you don't have a current `SSH` key set-up, then this is where you'll want to be. Note that I will be writing the commands I use for a Unix operating system (so Linux and Mac), the steps will remain the same on Windows but the commands might vary slightly.

1. Generate your `SSH` key. Type the following command into your terminal

```
ssh-keygen
```

This does exactly what it sounds like, you can specify the algorithm to use and the bit size of the resulting key if you would like, but I'm not going to go into how to do that that here, just know that it exists if you want the option. Your computer should say something along these lines.

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/usr/.ssh/id_rsa):
```

(`usr` here will be replaced by the username on your computer). Now if you currently don't have an `SSH` key on your computer, then this default name is what you want, it is where the git client automatically looks for `SSH` keys. Hit enter and you will see an additional line saying

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

Unless I am doing something for a company, I leave this blank, but if you do decide to enter a passphrase, then you will be required to enter it every time you push and pull from whatever Bitbucket account will be using this key. (Unless of course you have set up git's caching mechanism, but if you're going to set that up, why enter a passphrase in the first place?). Hit enter and you have completed step 1.

2. Now that you have generated your SSH key, you need to add it to your Bitbucket account. We will first copy it then go over to Bitbucket and add it. If you navigate to the folder in which you created your `SSH` key `cd ~ /.ssh` then you should see (at least) two files, `id_rsa` and `id_rsa.pub`, the former carries the private key information. DO NOT give this to anyone. This is how git verifies that you are actually you, even if someone has your public key (stored in the latter file), they won't be able to use it since they don't have the private key that verifies your identity. Type the following command into your terminal, which copies your public key to your clipboard

```
cat ~/.ssh/id_rsa.pub | xclip -sel clip
```

of course if you named your `SSH` key something different, replace the name in the command above. Alternatively, if you do not have `xclip` installed you can just enter the following command

```
cat ~/.ssh/id_rsa.pub
```

which will display the public key to your terminal so you can copy it manually the usual way.

3. Next, you need to add your public key to your Bitbucket account, to do this go to Bitbucket, on the bottom left hand corner there is a small circle with your Bitbucket profile, click that, and go to `personal settings`. Now on the left hand side, under security, go to `SSH keys`. Finally, click `add key`. Give your key a label, and then paste what we copied in previous step into the second, large text box.

4. You are almost finished, the final step is to update the URLs used your local repositories. I will assumed you want to update your volume 2 repository to push/pull via SSH, but the procedure is the same for any repository. To do this, we first need to get the appropriate URL. This is done by going to your volume 2 repository on Bitbucket, clicking on clone, changing the drop down menu to SSH, and copying the given URL (don't copy the the git clone part). The URL should look like git@bitbucket.org:<username>/<reponame>.git. Finally, go to the corresponding volume 2 local repository in your terminal and type

```
git remote set-url origin git@bitbucket.org:<username>/<reponame>.git
```

You have now successfully set up your first SSH key! Only step 4 needs to be repeated to set up SSH for any other Bitbucket repositories. Luckily the act of pushing and pulling remains the same. You can still use git add <file name> (and it's variations) and git commit -m "<message>" and git push/pull (and it's variations) the same way as you did before. The only thing changing is the behind-the-scenes procedure that your computer performs.

3 Multiple SSH Keys

If you are currently doing work for a company on your local machine, or perhaps you also have an SSH key set-up for a Github/Gitlab account, then you'll want to follow these instructions. You want to keep SSH keys separate, but when the git client is verifying identity, they need to know which private key to check. The procedure to set all this up is similar to the procedure above with one additional step.

1. Generate your SSH key by typing the following command into your terminal

```
ssh-keygen
```

however, now when they ask you to enter a file in which to save the key, you won't want to use the default name (keep it in the same .SSH directory though). Below is an example of something you could put.

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/usr/.ssh/id_rsa): /home/
   usr/.ssh/id_rsa_bitbucket
```

(usr here will be replaced by the username on your computer). after you hit enter, your computer will display the option for a passphrase, again, unless this is an SSH key for a company, I usually leave it blank.

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

2. Now this is where things change quite a bit from before. In the .SSH directory, we need to modify the config file to tell the git client where to look for your SSH key. If there is already a config file in this folder, then we can modify that one, if not create a new file named config (no file extension). Below is an example of what a config file could look like.

```
Host someCompany
  HostName gitlab.someCompany.com
  User git
  IdentityFile ~/.ssh/id_rsa
  IdentitiesOnly yes

Host bitbucket-personal
     HostName bitbucket.org
     User git
     IdentityFile ~/.ssh/id_rsa_bitbucket
     IdentitiesOnly yes
```

you will need to create a `Host` configuration for each current `ssh` key that you have. The `HostName` is the URL that will be using that `SSH` key, the `User` is the client that will be using it, the `IdentityFile` is where the `User` will find the corresponding private key and `IdentitiesOnly` Specifies that `SSH` should only use the identity keys configured in the `config` files, even if the ssh-agent offers more identities.

3. Luckily, these are where the differences end. From here on out, you can follow steps 2-4 in the instructions for one `SSH` key to finishing setting up your Bitbucket to use `SSH`.

4 Lab Computers

Lab computers impose a slight issue. Everything we have done before will still work. However, if you want to sit at a different lab computer then you will have to generate a new `SSH` key for that new computer. We will "mostly" get around this by using the `myacmeshare` folder that every ACME student has set up. Before getting into the procedure let me tell you the "big" picture. Similar to section 3 we will set up a `config` folder, but this time the identity file will be within your `myacmeshare` folder. That way your public and private key get copied stored on the ACME server, then when getting on a different lab machine, you just need to copy the `config` folder to the local .ssh folder.

1. if your current local copies of your repositories aren't inside the `myacmeshare` folder, delete them.

2. Generate your `SSH` key by typing the following command into your terminal

```
ssh-keygen
```

however, now when they ask you to enter a file in which to save the key, you will want to use the default name, but not in the default location. Something along the lines of

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/usr/.ssh/id_rsa): /home/
   usr/myacmeshare/.ssh/id_rsa_bitbucket
```

(`usr` here will be replaced by the username on your computer)

3. We now need to create the `config` folder which tells the git client where to find this `.ssh` key since it's not in the default location. change directory into the root directory `cd`, then list hidden files using `ls -ail` if a `.ssh` directory already exists, then enter it, if not create a new one with `mkdir .ssh`. Within this directory check to see if a `config` folder exists, if it does we will want to edit it, if not create one (no file extension). The format of the `config` file should look like

```
Host bitbucket-personal
     HostName bitbucket.org
     User git
     IdentityFile ~/myacmeshare/.ssh/id_rsa_bitbucket
     IdentitiesOnly yes
```

   Finally, copy this `config` file into your `myacmeshare` folder.

   Jut a little information about what the `config` file is doing; the `HostName` is the URL that will be using that `SSH` key, the `User` is the client that will be using it, the `IdentityFile` is where the `User` will find the corresponding private key and `IdentitiesOnly` Specifies that `SSH` should only use the identity keys configured in the `config` files, even if the ssh-agent offers more identities.

4. If you had to delete your repositories in step 1, then complete steps 2 and 3 in section 2, however now instead of step 4, complete the following sub-setp

   (a) Again, I will assume that you are updating your volume 2 repository. On bitbucket, go to your volume 2 repository and click clone, make sure the drop down menu has `ssh` selected, then on your machine change directory into your `myacmeshare` folder and paste the command you just copied. It should look something like,

   ```
   git clone git@bitbucket.org:<username>/<reponame>.git
   ```

   hit enter and you have successfully cloned your repository via `ssh`!

5. The point of us having you copy the `config` file into your `myacmeshare` folder is because now since that `config` file is on the server, if you log into a different computer, you only need to copy the `config` file to the local `.ssh` folder on that machine and your git will be properly set up.