GitWiki

Release 0.0.1

JeffCube

Oct 17, 2021

CONTENTS

GitWiki is a wiki containing information on various topics concerning Git

Note: This project is under active development.

CHAPTER

ONE

GITHUB

1.1 Managing Multiple GitHub Accounts

1.1.1 Common Issues After Creating Another Account

Lets say you have just created a second github account you may run into the following hurdles:

Failing to Clone Your Private Repository

You decide you want to create a private repository on GitHub. Once it is created you want to start work so you begin by cloning the repository.

```
$   git clone https://github.com/<username>/private_repo_test.git
```

Suddenly you are greeted with this error.

> **Error:** Cloning into 'private_repo_test'... remote: Repository not found. fatal: repository 'https://github.com/<username>/private_repo_test.git/' not found

Assuming the url is correct, this message means do not have permission to access the repository.

Failing to Push to Your Public Repository

Instead of creating a private repository you may want to begin using your new account by creating a new public repository. After creating the repository cloning the repository goes without a hitch.

```
$   git clone https://github.com/<new_account_name>/public_repo_test.git
```

Alternatively maybe the repository is public and cloning is successful:

```
Cloning into 'public_repo_test'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
```

However after you make a new commit and try to push to the repository you are greeted with a new error:

```
$ git commit -m "new commit"
$ git push origin main
```

Error: remote: Permission to <new_account_name>/public_repo_test.git denied to <some_name>. fatal: unable to access 'https://github.com/<new_account_name>/public_repo_test.git/': The requested URL returned error: 403

In the error message the <some_name> gives you the hint that you are attempting to push to the repository without the credentials of your newly created github account. You may have tried to change your local git configurations to reference the correct username and email as so:

```
$ git config --local user.name "<new_account_name>"
$ git config --local user.email "<new_account_email>"
```

But you will still run into the same error and even will display the same <some_name> in the error message. this hints at what needs to happen to have the correct credentials when accessing repositories on your local machine.

1.1.2 How to Work With Multiple GitHub Accounts

The following instructions are created from * Tuts+ Code tutorial * This stack overflow link * GitHub's SSH key documentation

Note: These instructions were created when working on Windows 10

Note: Before starting this process you should view your ssh configuration file `~/.ssh/config` to see all of the ssh keys you have set any prior configurations. You should also visit `~/.ssh/` to see all of the keys you have already created. Keep these in mind so you do not accidentally overwrite any keys when you create new ones. If `~/.ssh/` does not exist then that means you likely have not created any ssh keys on your device yet.

Generate SSH Keys For Your Accounts

Create an ssh key for one of your GitHub accounts by running

```
$ ssh-keygen -t ed25519 -C "your_email@example.com"
```

Note:

- The -t option specifies with the type of key to be generated. GitHub currently recommends Ed25519
- The -C option specifies a comment. For RSA1 keys the comment field is for convenience to help the user identify the key.
- If you are using a legacy system that doesn't support the Ed25519 algorithm, use:

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

You will then be asked where to save the ssh key files:

```
Generating public/private ed25519 key pair.
Enter file in which to save the key (/c/Users/example_user/.ssh/id_ed25519)
```

You can press enter to use the default location. In this case `/c/Users/example_user/.ssh/id_ed25519` saves the public key to `/c/Users/example_user/.ssh/id_ed25519.pub` and the private key to `/c/Users/example_user/.ssh/id_ed25519`. However, because you are working with multiple github accounts I would recommend that you set the filename yourself to something involving the account name (So in this case `/c/Users/example_user/.ssh/id_ed25519_<account_name>`).

Next you will be asked for a passphrase:

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

You can choose to type no passphrase or enter one. By entering a passphrase you will be prompted to enter a passphrase each time you use the ssh key. You can also make it so you have one super password so you won't have to reenter your passphrase every time you use your SSH keys (See Working with SSH key passphrases).

After creating your key we need to add it to the ssh-agent. Start the agent by running:

```
$eval "$(ssh-agent -s)"
```

Next add your newly created key:

```
$ ssh-add ~/.ssh/id_ed25519_<account_name>
```

Finally add your SSH key to our account on GitHub by following GitHub's Instructions

Creating an SSH Key Config File

After creating and adding ssh keys to all of your GitHub accounts in *Generate SSH Keys For Your Accounts* we can now create a configuration file that will allow us to specify our credentials when the time comes to access our remote repositories.

1. If `~/.ssh/config` does not exist, create it.
2. Edit the file to look like the following:

Note: This config file assumes we have 2 accounts (`user_name_0`, `user_name_1`) and that we have created and saved their private keys to `~/.ssh/id_ed25519_user_name_0` and `~/.ssh/id_ed25519_user_name_1`

```
# github account: user_name_0
Host github-user_name_0
   HostName github.com
   IdentityFile ~/.ssh/id_ed25519_user_name_0
   IdentitiesOnly yes

# github account: user_name_1
Host github-user_name_1
   HostName github.com
   IdentityFile ~/.ssh/id_ed25519_user_name_1
   IdentitiesOnly yes
```

1. Test your connection by running

```
$ ssh -T git@github-user_name_0
$ ssh -T git@github-user_name_1
```

With each command, you may see this kind of warning:

> **Warning:** The authenticity of host 'github.com (192.30.255.112)' can't be established. RSA key fingerprint is **********. Are you sure you want to continue connecting (yes/no/[fingerprint])?

Type yes and you should see the following:

```
Hi user_name_0! You've successfully authenticated, but GitHub does not provide␣
↪shell access.
Hi user_name_1! You've successfully authenticated, but GitHub does not provide␣
↪shell access.
```

1.1.3 Using Your SSH Config

Now that you have setup your SSH keys and configurations, we can now overcome the common errors we saw earlier in *Common Issues After Creating Another Account*

Cloning Public / Private Repositories

Now when cloning a public or private repository you can use your new ssh configurations.

Instead of the default HTTPS clone:

```
git clone https://github.com/user_name_0/private_repo_test.git
```

We can use the SSH clone with the ssh host `github-user_name_0` we defined earlier in the configuration file:

```
git clone git@github-user_name_0:user_name_0/private_repo_test.git
```

You will now be able to also push and pull from the remote repository.

Fixing Push Errors for Existing Repositories

In the case where you have cloned a public repository without specifying your SSH keys you may encounter the following error when trying to push to it (`$ git push origin main`):

Error: remote: Permission to <user_name_0>/public_repo_test.git denied to <some_name>. fatal: unable to access 'https://github.com/<user_name_0>/public_repo_test.git/': The requested URL returned error: 403

To fix this change the url of the `origin` of your repository:

```
$ git remote set-url origin git@github-user_name_0:user_name_0/public_repo_test.git
```

Now you will be able to push and pull from the remote repository.

Managing Commit Authors

When you start committing changes across multiple repositories, you may want to make sure that your commits are associated with the correct github accounts. To do this navigate to inside each of your repositories and then run:

```
$ git config --local user.name "<user name>"
$ git config --local user.email <user_email>
```

This way every commit you make in each repository will be associated with the name and email address you specify for that specific repository.