Git

October 3, 2019

1 Git and Git-Hub

At some point in your research you will likely have a massive hard drive crash, or realize you introduced a bug in your code and want to revert it back to its working state (perhaps from several weeks before). Or you may end up working on a bit of code with other people. In any of these cases version control saves the day. This document with give you an introduction to `Git` and `Git-Hub` as a version control solution.

1.1 Before you start

- Install `git`: https://git-scm.com/downloads
- Install `git-completion`: https://github.com/bobthecow/git-flow-completion/wiki/Install-Bash-git-completion (optional)
- Create a `Git-Hub` account: https://github.com/
- Set up ssh key with `Git-Hub` account: https://help.github.com/articles/generating-an-ssh-key/ (optional)
- Set up Two-Factor Authentication for `Git-Hub`: https://help.github.com/articles/about-two-factor-authentication/ (optional)

1.2 Useful documentation

- The Git Book

1.3 Making a repo on Git-Hub

When you click on the "New repository" button on Git-Hub you will see the following:

Git-Hub new repo

Here you can enter in some information about your repo:

- Repository name: the name of you repo and this will also be in the url for the repo
- Description: Some test to start off your README file
- Public/Private repo: public repos are free, as a student private ones are too (see https://education.github.com/pack)
- Initalize README: If checked this will place a README.md file in your new repo. The README is a markdown file that is displayed on the repo's home page.
- Add .gitignore: A .gitignore file tells git what files to ignore in a repo (e.g. .pyc files), there are useful templates for several languages you can start with. If you are coding in python I would suggest picking it from the dropdown list.
- Add a license: **Do this!** You should always add a license to you code, if you are unsure what license to use click the i button and read up on the different types (http://choosealicense.com/). I would recommend either the MIT or Apache 2.0 license for your work.

1.3.1 Clone a local copy

Now that you have a new repo online it is time to get a local copy on your computer. Open a terminal and cd to the directory you want your repo to live. On Git-Hub click the "Clone or download" button, if you set up an ssh key click the "Use SSH" link, if you did not, make sure you use the "Use HTTPS" link. Here is an example of an ssh link:
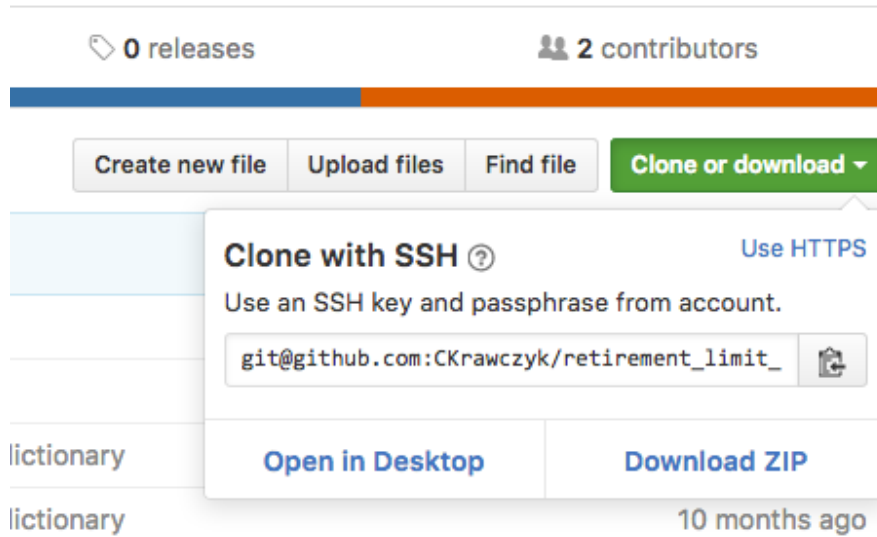
Git-Hub clone via ssh

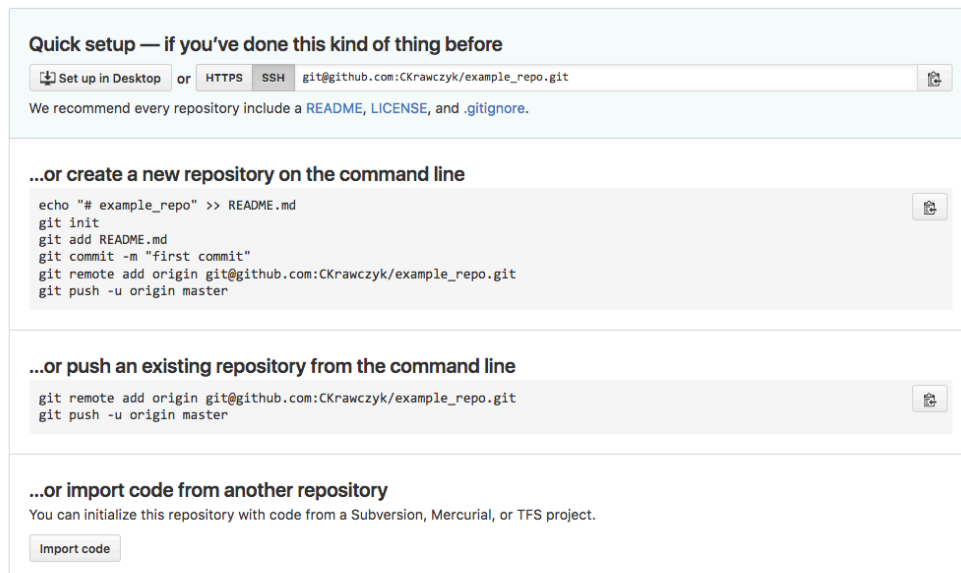Copy the link provided and go back to your terminal and type the command:

```
git clone <link you copied>
```

1.4 Making a local repo

You can also make an existing folder into a git repo. If you plan to keep back ups of this code on Git-Hub (like you should), start by making a blank repo on Git-Hub (see above) and just give it a name. Once you do that you will see the following:

Git-Hub blank repo

This gives you the step-by-step code for initializing your repo and setting the `remote` URL correctly. Again, if you have `ssh` set up make sure you have the "SSH" button pressed, and if you don't make sure the "HTTPS" button is pressed.

1.4.1 Git without Git-Hub

You can also just use git without pushing your code up to Git-Hub. To initialize git in a directory:

```
cd <dirctory>
git init
```

2 Git setup

Before we start using git there are some basic configuration:

```
git config --global user.name "<your name>"
git config --global user.email <your email>
git config --global core.editor <your text editor (defaults to vim)>
```

Check that your configuration look correct:

```
git config --list
```

2.1 getting help

If you ever need help with a git command:

```
git help <command>
git <command> --help
```

3 Git basics

Now that you have your repo set up it is time to learn how to use version control. The basic workflow for `git` is: add/make changes to a file, stage files for a commit, commit the files to the repo, push commits to remote server (e.g. Git-Hub).

3.1 checking the status of your repo

At any time you check the status of your repo (i.e. what files have changed, and what files are staged for commit) with:

```
git status
```

This screen helpfully tells you how to "undo" staged files (more on this below).

3.2 staging files for a commit

You can stage changes for a commit by using the `add` command:

```
git add <file>
```

Or a if you want to add all files that have changed:

```
git add .
```

To unstage a file that was *never* committed before (keeps local changes):

```
git rm --cached <file>
```

To unstage a file that was committed before (keeps local changes):

```
git reset HEAD <file>
```

To undo changes made to a file (revert it back to previous commit):

```
git checkout -- <file>
```

3.3 commiting changes to a repo

After files are staged you can commit the changes using:

```
git commit
```

This will open a text editor for you to type a commit message into. This should be a short description of what things were changed in the commit. If your commit message is short you can also use:

```
git commit -m "<your commit message here>"
```

Note Once a commit has been made that version of the file will *always* be in your repo. You will always be able to return to this version of the code (this is the point of version control). Because of this **never** commit sensitive information (e.g. passwords, private data, etc...) to a public repository.

3.4 pushing changes to Git-Hub

If you have your remote repository set up (see above), you can push your commits with:

```
git push
```

3.5 pulling changes from Git-Hub

If you are working on several computers, or you are working with other people, you can pull changes down from the remote repository with:

```
git pull
```

3.6   code conflicts

If your local version of the code conflicts with the remote version of the code (e.g. the same file was edited) you will not be able to push or pull the code without resolving the conflicts. Git will indicate were the conflicts are in the code with blocks that look like:

```
<<<<<<< HEAD
nine
=======
eight
>>>>>>> branch-a
```

Edit this block to look like the version you want, than stage the file with git add. You can see where the conflicts are with git status. Once all conflicts are resolved git commit your changes.

4   Branches (and why you should use them!)

Git lets you keep several versions of you code at once via a branch system. A good habbit to get into is to only use the master branch (the default first branch) for working/finished code, and use branches to test out new features. Once the new feature is finished then it's branch is merged into the master branch.

This is very useful if multiple people are working on the same code. Each person works in their own branch, and when they are done they update their master branch to pull in the work of others, than merger their own work into master (and deal with any merge conflicts).

4.1   list all your branches

```
git branch
```

4.2   switch to a new branch

```
git checkout <branch name>
```

4.3   to make a new branch (and switch to it)

```
git checkout -b <new branch name>
```

Note Any uncommitted changes are carried over when you make a new branch. So if you started editing your master branch by mistake you can just make a new branch and not loose any of the changes you made or mess up your master branch.

4.4   merging a branch

```
git checkout master
git merge <branch name>
```

If there are any conflicts during the merge resolve them as stated above.

Note In this example we are merging the branch into master, but master is just a branch like any other (there is nothing special about master). You can merger any two branches.

[ ]: