Git Level 2 - Github and Branches

Copyright

What's in this page?

- Creating your github account
- Making your first repository
- Branches and pull requests

Introduction

Welcome back! In the last episode we got started with a local github repository, but coding by yourself is no fun (well it is but it's more fun with others!). In this tutorial we're going to learn how we can become a part of a larger software community by sharing our code on github.

Note about master vs main

Recently github has changed the default branch name from "master" to "main". Thus some commands may not work exactly as written; you might need to substitute master for main or vice-versa.

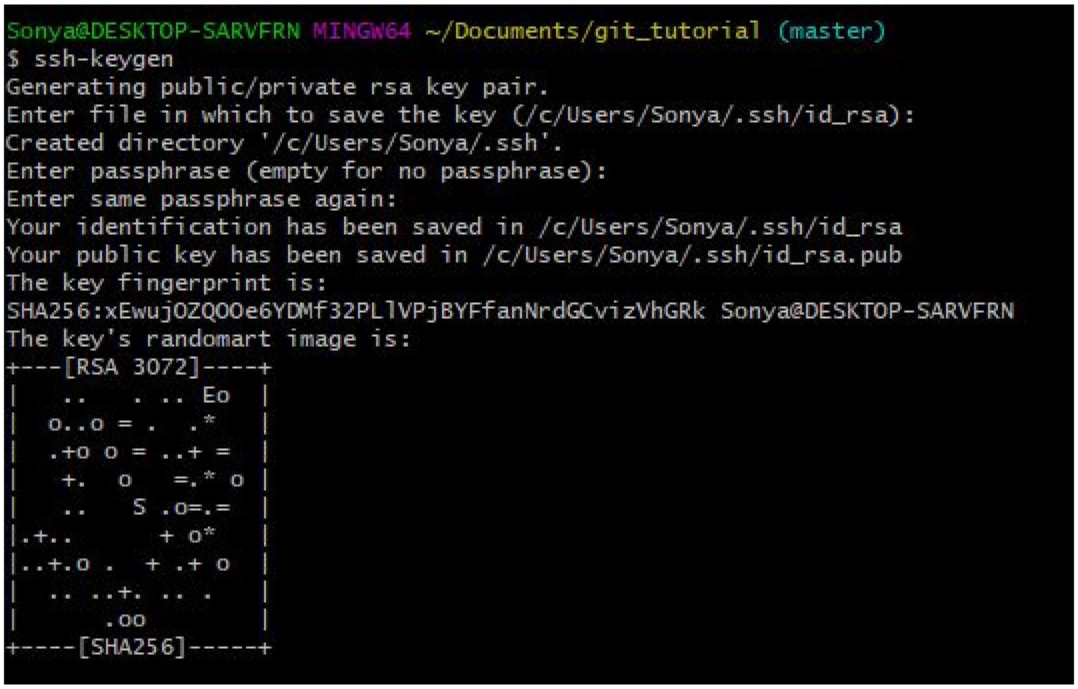
Create your github account and add ssh keys

Create an account and generate ssh keys

First of all we want to create our github account! Head on over to github.com and do that right away.

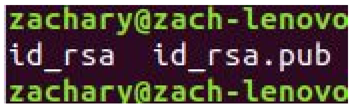
Another super important set-up step is to create your ssh keys. An ssh key consists of two parts-a public key and private key. When stuff is encoded with the public key, it can only be decoded by the private key. This allows github to verify your identity. Without an ssh key, you won't be able to upload stuff to GitHub as easily.

Go ahead and open up your terminal (remember how to do that from part one?). Generate your ssh key by typing “ssh-keygen” and then just keep pressing enter to accept the defaults. You should get an output like below



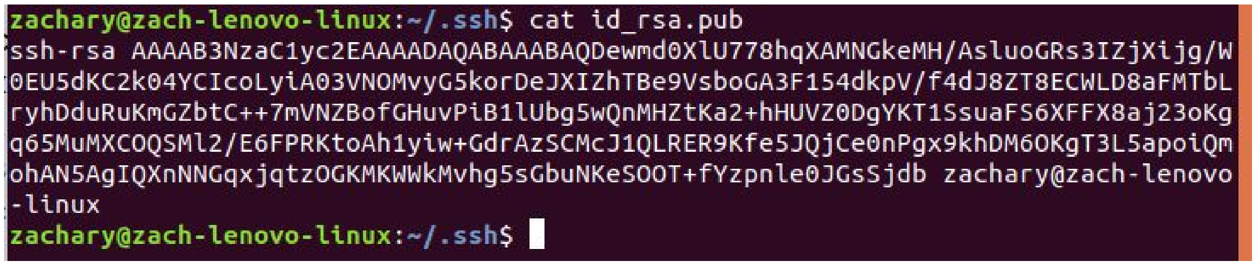
```
Sonya@DESKTOP-SARVFRN MINGW64 ~/Documents/git_tutorial (master)
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/Sonya/.ssh/id_rsa):
Created directory '/c/Users/Sonya/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/Sonya/.ssh/id_rsa
Your public key has been saved in /c/Users/Sonya/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:xEwujOZQOOe6YDMf32PLlVPjBYFfanNrdGCvizVhGRk Sonya@DESKTOP-SARVFRN
The key's randomart image is:
+---[RSA 3072]----+
|   ..   . .. Eo  |
|  o..o = .  .*   |
|  .+o o = ..+ =  |
|   +.  o   =.* o |
|   ..   S .o=.=  |
|.+..      + o*   |
|..+.o .  + .+ o  |
|  .. ..+. .. .   |
|      .oo        |
+----[SHA256]-----+
```

This will first make a new hidden folder in your home directory called “.ssh”. We can change into that folder with cd ~/.ssh. Type “ls” to show all the contents.



```
zachary@zach-lenovo
id_rsa  id_rsa.pub
zachary@zach-lenovo
```

Notice how we have two files? The “id_rsa” file is just a text file containing your private key… don’t share this with anyone! The id_rsa.pub file contains our public key. This is the one we want to share with github. We can quickly see the contents of this text file by typing “cat id_rsa.pub”. By the way, “cat” is just another one of those command (like cd or ls). It will show the contents of one or more text files on the screen.

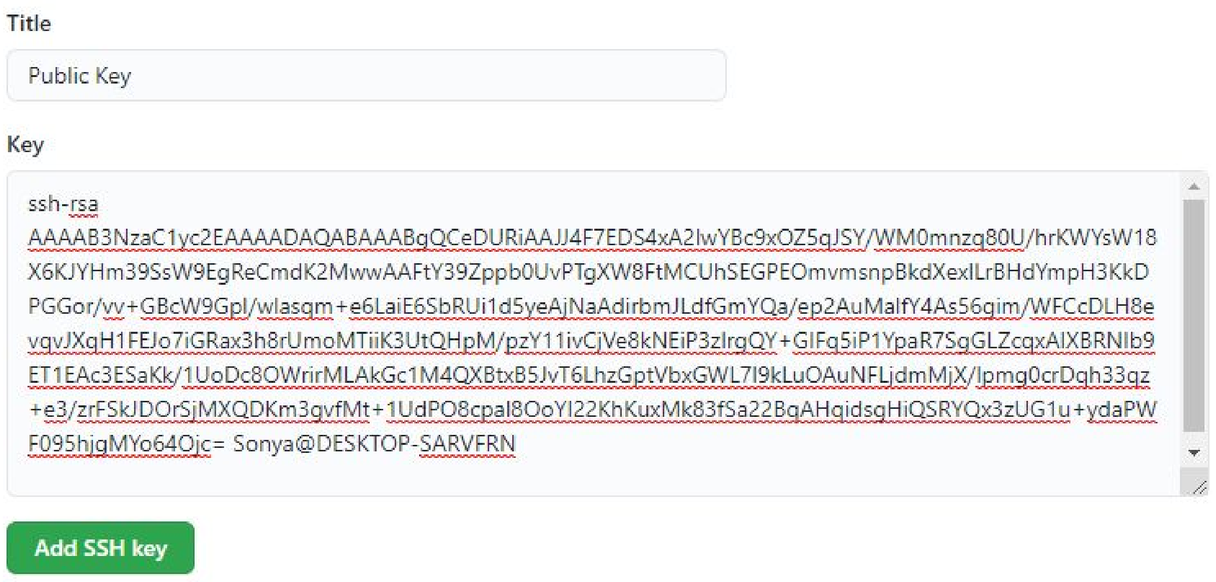


```
zachary@zach-lenovo-linux:~/.ssh$ cat id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDewmd0XlU778hqXAMNGkeMH/AsluoGRs3IZjXijg/W
0EU5dKC2k04YCIcoLyiA03VNOMvyG5korDeJXIZhTBe9VsboGA3F154dkpV/f4dJ8ZT8ECWLD8aFMTbL
ryhDduRuKmGZbtC++7mVNZBofGHuvPiB1lUbg5wQnMHZtKa2+hHUVZ0DgYKT1SsuaFS6XFFX8aj23oKg
q65MuMXCOQSMl2/E6FPRKtoAh1yiw+GdrAzSCMcJ1QLRER9Kfe5JQjCe0nPgx9khDM6OKgT3L5apoiQm
ohAN5AgIQXnNNGqxjqtzOGKMKWWkMvhg5sGbuNKeSOOT+fYzpnle0JGsSjdb zachary@zach-lenovo
-linux
zachary@zach-lenovo-linux:~/.ssh$ 
```

See! Here’s my public key. Copy your public key (not mine) and head on over to github.

Adding your ssh keys to GitHub

Now that we’re on github, hit your little profile icon in the top right corner and go down to settings. Now hit “SSH and GPG keys” and then click “New SSH key”. Paste the public key we generated before, give it a name and add your key!

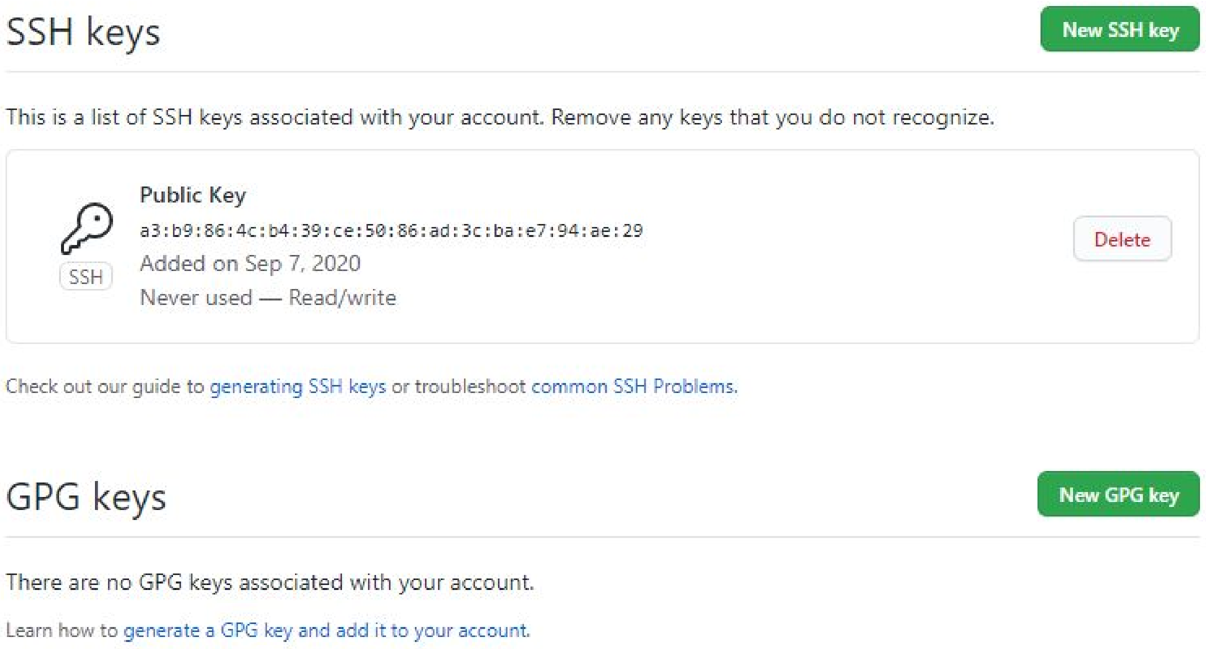


Title

Public Key

Key

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABgQCeDURiAAJJ4F7EDS4xA2lwYBc9xOZ5qJSY/WM0mnzq80U/hrKWYsW18
X6KJYHm39SsW9EgReCmdK2MwwAAFtY39Zppb0UvPTgXW8FtMCUhSEGPEOmvmsnpBkdXexlLrBHdYmpH3KkD
PGGor/vv+GBcW9Gpl/wlasqm+e6LaiE6SbRUi1d5yeAjNaAdirbmJLdfGmYQa/ep2AuMalfY4As56gim/WFCcDLH8e
vqvJXqH1FEJo7iGRax3h8rUmoMTiiK3UtQHpM/pzY11ivCjVe8kNEiP3zIrgQY+GIFq5iP1YpaR7SgGLZcqxAIXBRNIb9
ET1EAc3ESaKk/1UoDc8OWrirMLAkGc1M4QXBtxB5JvT6LhzGptVbxGWL7I9kLuOAuNFLjdmMjX/lpmg0crDqh33qz
+e3/zrFSkJDOrSjMXQDKm3gvfMt+1UdPO8cpal8OoYI22KhKuxMk83fSa22BqAHqidsqHiQSRYQx3zUG1u+ydaPW
F095hjgMYo64Ojc= Sonya@DESKTOP-SARVFRN
```


SSH keys

This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.

Public Key
a3:b9:86:4c:b4:39:ce:50:86:ad:3c:ba:e7:94:ae:29
Added on Sep 7, 2020
Never used — Read/write

Check out our guide to generating SSH keys or troubleshoot common SSH Problems.

GPG keys

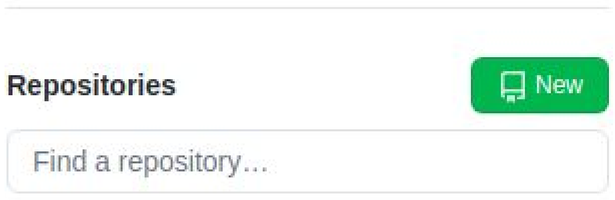
There are no GPG keys associated with your account.

Learn how to generate a GPG key and add it to your account.

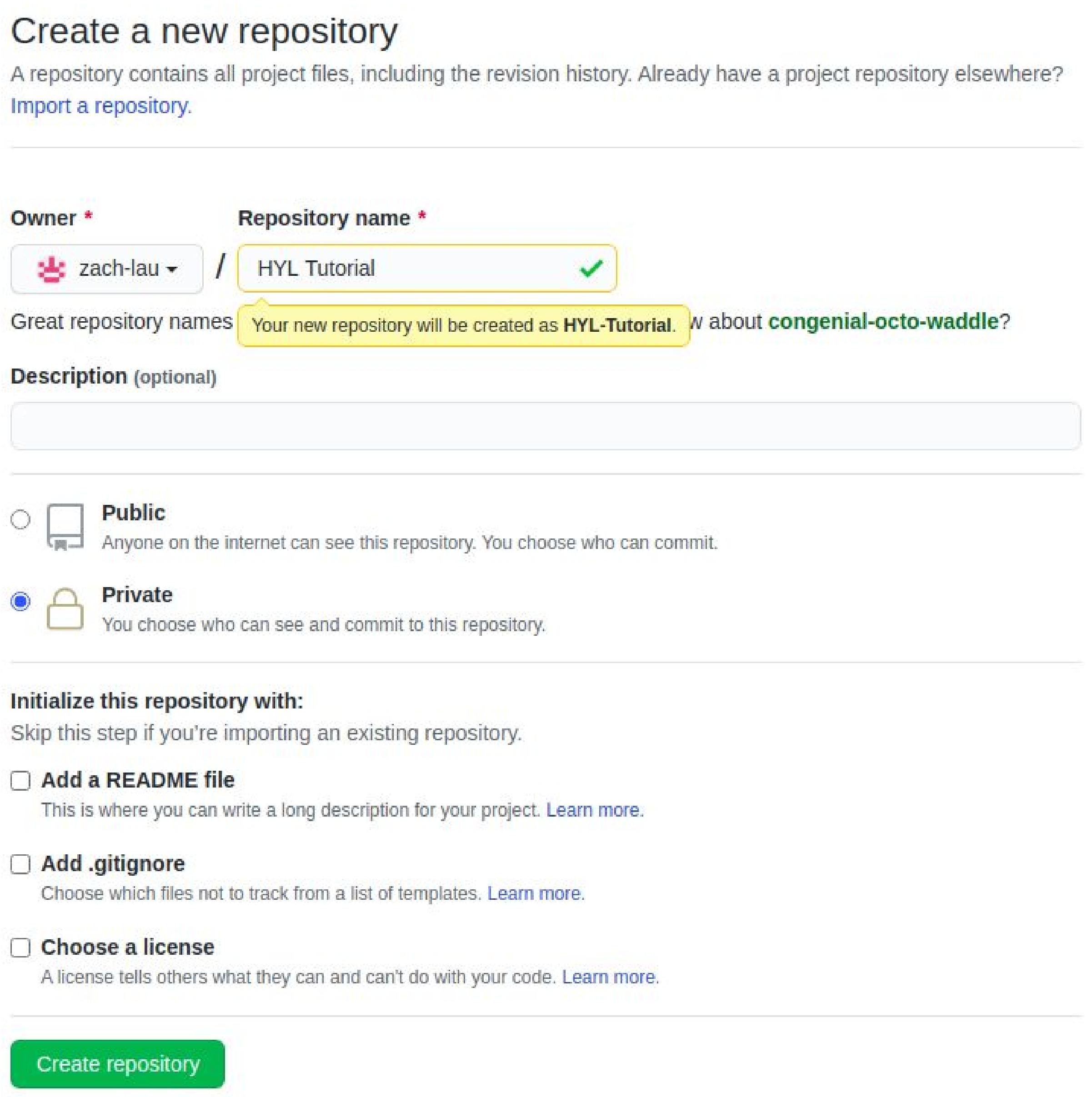
Now you should be good to go… you've connected your ssh keys with your github account.

Making your own repository

At this point we might want to share our amazing hello world project with the rest of the world. First go back to your home page (you can click the github logo in the top left corner). Then look for where it says repositories and there should be a "new" button right there.



This will give us a bunch of options.



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Owner * Repository name *

zach-lau / HYL Tutorial

Great repository names Your new repository will be created as **HYL-Tutorial**. w about **congenial-octo-waddle**?

Description (optional)

( ) **Public**
Anyone on the internet can see this repository. You choose who can commit.

(●) **Private**
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

☐ **Add a README file**
This is where you can write a long description for your project. Learn more.

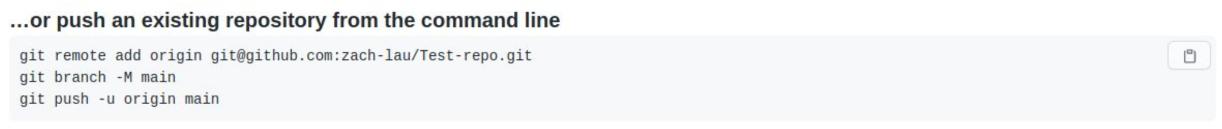
☐ **Add .gitignore**
Choose which files not to track from a list of templates. Learn more.

☐ **Choose a license**
A license tells others what they can and can't do with your code. Learn more.

Create repository

IMPORTANT: don't click any of the boxes at the bottom since we already have a repository (the Hello World we made in part one).

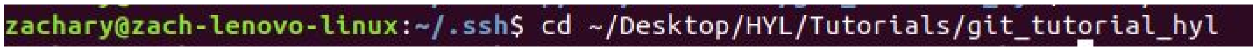
Now we're going to want to hit "Create repository" and then type the commands below into your git bash/terminal.



...or push an existing repository from the command line

```
git remote add origin git@github.com:zach-lau/Test-repo.git
git branch -M main
git push -u origin main
```

But first, open up your terminal and change to the directory where you made your project from last time.



```
zachary@zach-lenovo-linux:~/.ssh$ cd ~/Desktop/HYL/Tutorials/git_tutorial_hyl
```

Now there's a lot going on here, so let's break it down.

Step 1: git remote add origin git@github.com:zach-lau/SUAV-Tutorial-2020.git

This step is adding the "remote" repository. A remote repository is just a repository on a server like github. This line is basically saying that user git at the domain github.com has a repository by the name of "zach-lau/SUAV-Tutorial-2020.git". Obviously you would substitute your username for the "zach-lau". The intent is that everything we do on our computer we will copy over to this server so that we can access it anytime anywhere.

The "origin" is just a kind of label that we have for this specific remote server. We can actually add more than one remote and by using "origin" we don't have to type out the whole url every time we reference it. You can name the remote whatever you want, but origin is most common.

Step 2: git branch -M main

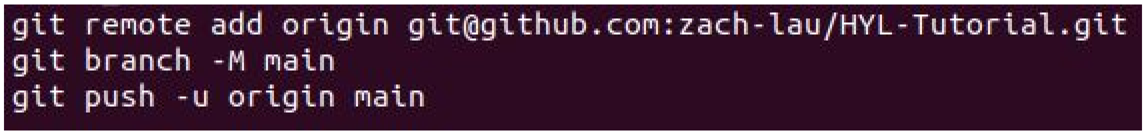
This step is creating a branch named main since GitHub is expecting a main branch to be the default on the remote.

If you don't know what a branch is don't worry! I'll explain shortly.

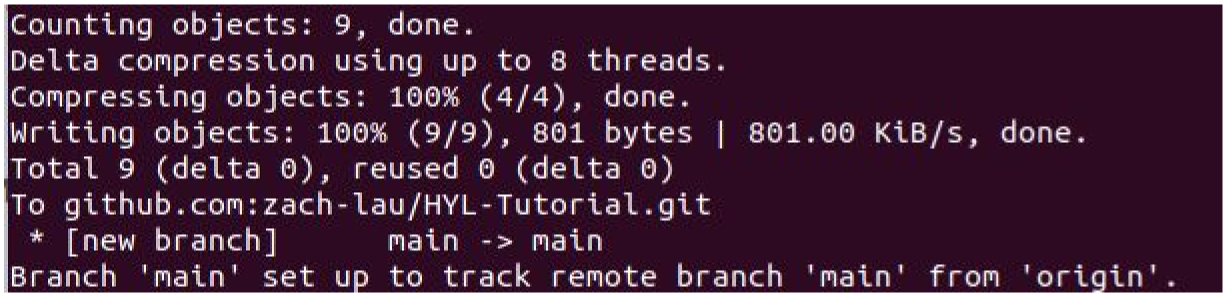
Step 3: git push -u origin main

Finally we use the "git push" command. This will copy everything from our local repository onto the remote. At this point we can go back to the github website and we should be able to see our code (refresh if it isn't there).

Type each of those steps into git and you should see something like

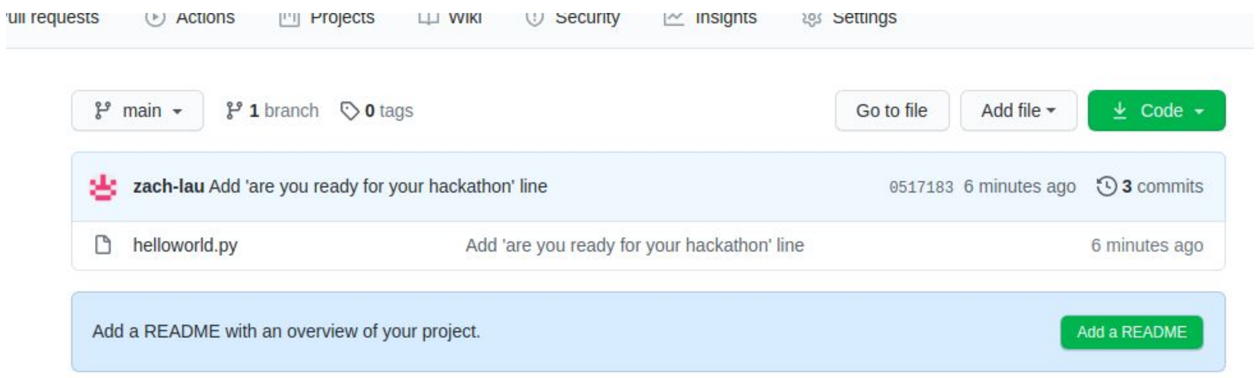


```
git remote add origin git@github.com:zach-lau/HYL-Tutorial.git
git branch -M main
git push -u origin main
```



```
Counting objects: 9, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (9/9), 801 bytes | 801.00 KiB/s, done.
Total 9 (delta 0), reused 0 (delta 0)
To github.com:zach-lau/HYL-Tutorial.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
```

And on github we'll be able to see



zach-lau Add 'are you ready for your hackathon' line 0517183 6 minutes ago

helloworld.py Add 'are you ready for your hackathon' line 6 minutes ago

Add a README with an overview of your project.

Branches - what are they?

Now you've probably seen me mention branches a few times. But what is a branch? I think it's most useful to think of branches when we have multiple people working on a project.

Let's say that Alice and Bob are working on the same project. Hence they start out with the same version of the code. Alice goes and implements some change on her copy of the code and Bob implements a different change. Now their codebases have diverged!... or we could say they have branched apart.

The problem occurs when they try to upload their code to the remote repository. Who has the true version of the code? Well we can avoid this problem for now by storing both copies, so Alice's code will be on one branch and Bob's code will be on another branch. We'll have to combine them eventually but in the meantime we'll have each change implemented on it's own branch.

What does this allow us to do? Well with branches we can easily share multiple different versions of the codebase and quickly hop back and forth! That way Alice can check out Bob's change and Bob can check out Alice's change. Nifty eh?

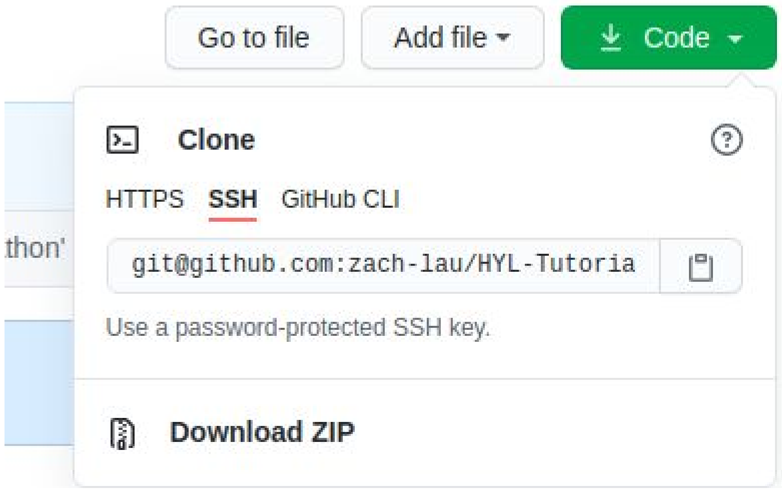
Let’s see exactly how it all works by diving into one of the UAV test spaces.

Clone your first repository

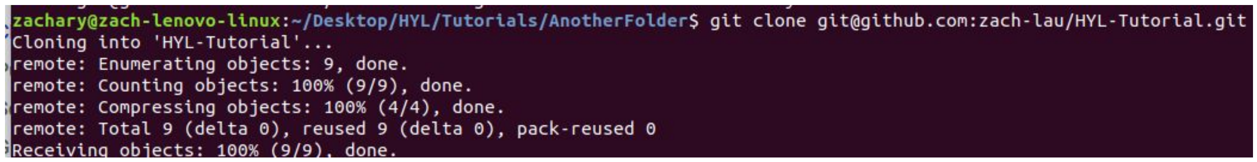
Ok ok.. now for the fun part. ~~Stealing other people's code!~~ Working on collaborative projects!

To start working on the project we first have to get our own local copy. The next part of the tutorial can be done either on a friend’s laptop, or in a different folder on your own laptop.

To clone a repository, go to your repository, then hit the “Code” button in the top right and then choose SSH. If you don’t choose SSH you won’t be able to push your changes. Copy that url.


Go to file
Add file
Code
Clone
HTTPS
SSH
GitHub CLI
thon'
git@github.com:zach-lau/HYL-Tutoria
Use a password-protected SSH key.
Download ZIP

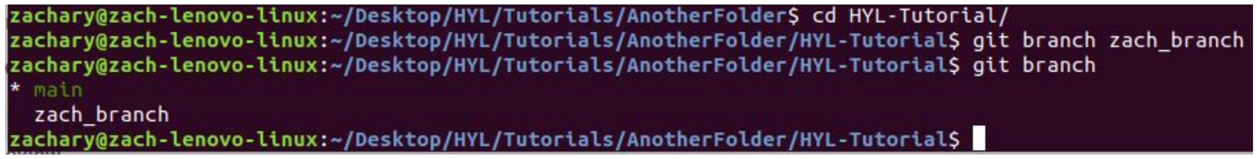
Now open a terminal and navigate to wherever you want to put the files from this repository. By typing “git clone <url you just copied>”, you can make your own copy.



```
zachary@zach-lenovo-linux:~/Desktop/HYL/Tutorials/AnotherFolder$ git clone git@github.com:zach-lau/HYL-Tutorial.git
Cloning into 'HYL-Tutorial'...
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 9 (delta 0), reused 9 (delta 0), pack-reused 0
Receiving objects: 100% (9/9), done.
```

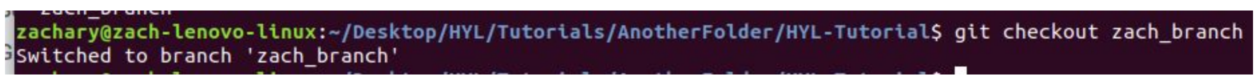
Before we start working we want to make our own branch-if you’re ever working on a project with other people NEVER DO YOUR WORK ON main. To make a branch type “git branch *mybranchname* main”. Choose a unique name for your branch. By typing main here we are saying that we are branching off of main. We can have branches off other branches but that’s less common.

If we just type git branch we'll be able to see all the branches. The star indicates which branch we are currently on.



```
zachary@zach-lenovo-linux:~/Desktop/HYL/Tutorials/AnotherFolder$ cd HYL-Tutorial/
zachary@zach-lenovo-linux:~/Desktop/HYL/Tutorials/AnotherFolder/HYL-Tutorial$ git branch zach_branch
zachary@zach-lenovo-linux:~/Desktop/HYL/Tutorials/AnotherFolder/HYL-Tutorial$ git branch
* main
  zach_branch
zachary@zach-lenovo-linux:~/Desktop/HYL/Tutorials/AnotherFolder/HYL-Tutorial$
```

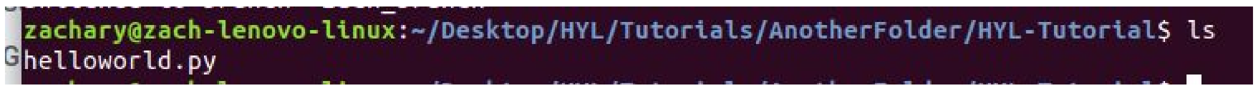
We want to be on "zach_branch" so let's change to it with "git checkout zach_branch"



```
zachary@zach-lenovo-linux:~/Desktop/HYL/Tutorials/AnotherFolder/HYL-Tutorial$ git checkout zach_branch
Switched to branch 'zach_branch'
```

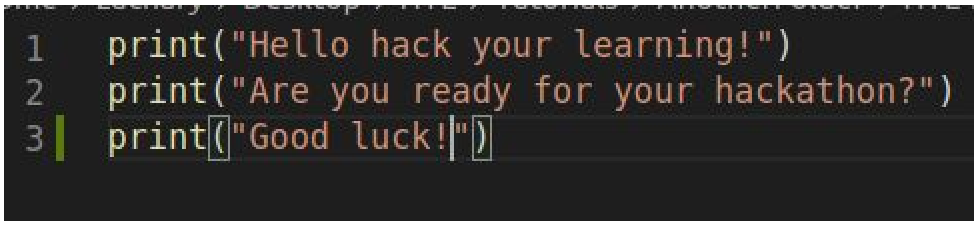
Note that you'll often see these two steps combined into one by "git checkout -b branchname main". Since it defaults to branching off of "main" we can omit the main at the end as well to get "git checkout -b branchname".

Look… we can see the file that we made earlier.



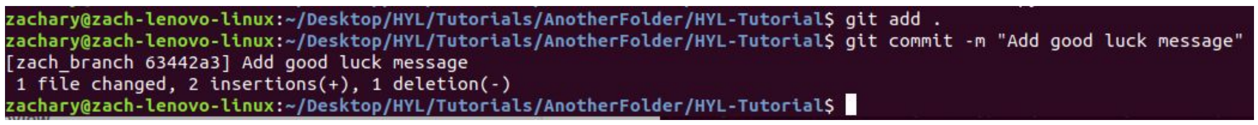
```
zachary@zach-lenovo-linux:~/Desktop/HYL/Tutorials/AnotherFolder/HYL-Tutorial$ ls
helloworld.py
```

Let's make a change.



```
print("Hello hack your learning!")
print("Are you ready for your hackathon?")
print("Good luck!")
```

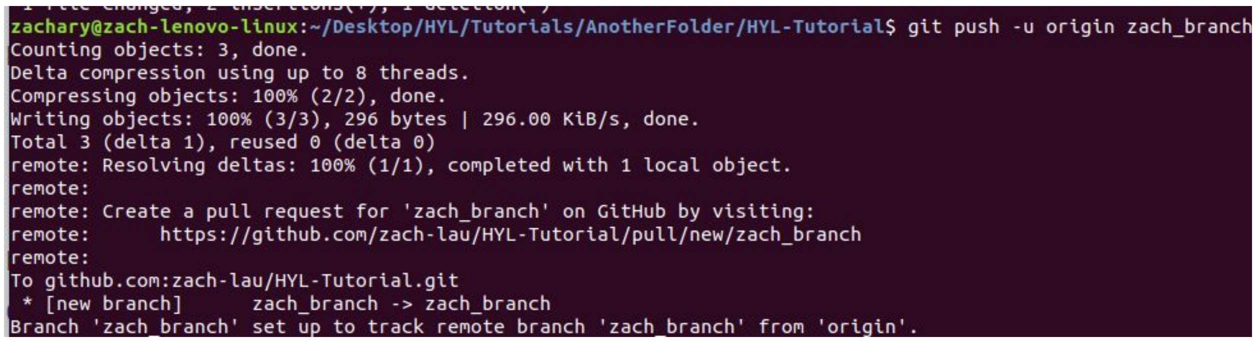
We can add and commit our change just as we did in part 1.



```
zachary@zach-lenovo-linux:~/Desktop/HYL/Tutorials/AnotherFolder/HYL-Tutorial$ git add .
zachary@zach-lenovo-linux:~/Desktop/HYL/Tutorials/AnotherFolder/HYL-Tutorial$ git commit -m "Add good luck message"
[zach_branch 63442a3] Add good luck message
 1 file changed, 2 insertions(+), 1 deletion(-)
zachary@zach-lenovo-linux:~/Desktop/HYL/Tutorials/AnotherFolder/HYL-Tutorial$
```

The -m flag allows us to add a message without opening up a text editor.

Then upload our change to github by using "git push -u origin <mybranchname>"

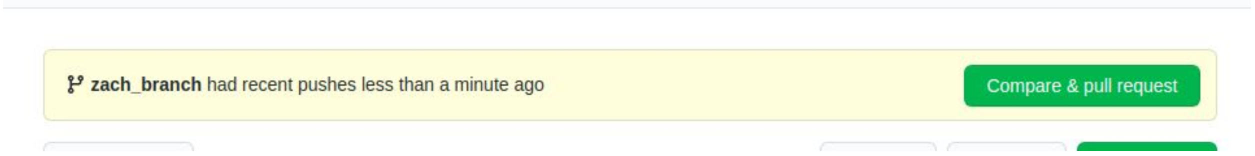

```
zachary@zach-lenovo-linux:~/Desktop/HYL/Tutorials/AnotherFolder/HYL-Tutorial$ git push -u origin zach_branch
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 296 bytes | 296.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote:
remote: Create a pull request for 'zach_branch' on GitHub by visiting:
remote:      https://github.com/zach-lau/HYL-Tutorial/pull/new/zach_branch
remote:
To github.com:zach-lau/HYL-Tutorial.git
 * [new branch]      zach_branch -> zach_branch
Branch 'zach_branch' set up to track remote branch 'zach_branch' from 'origin'.
```

The “-u” is specifying the “upstream” branch. Whenever we deal with software “upstream” usually means the most accessible, so since the server is more accessible than your local branch it is upstream. We type “origin” to say we want to push to the remote repository that we have labelled “origin” and then we type the branch name (in my case “zach_branch”).

Note: we’ll only need to add the -u flag the first time we push a branch. After that git will remember.

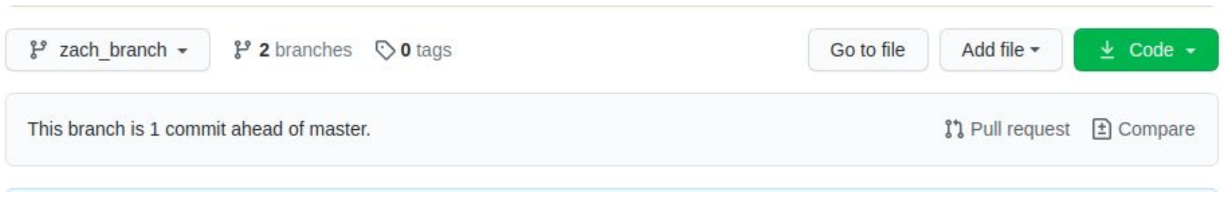
Pull Requests

That’s cool, but how do I get my code into the actual mainstream code if I don’t push to main? That’s where pull requests come in. After you push your change you might see a pop up like this:

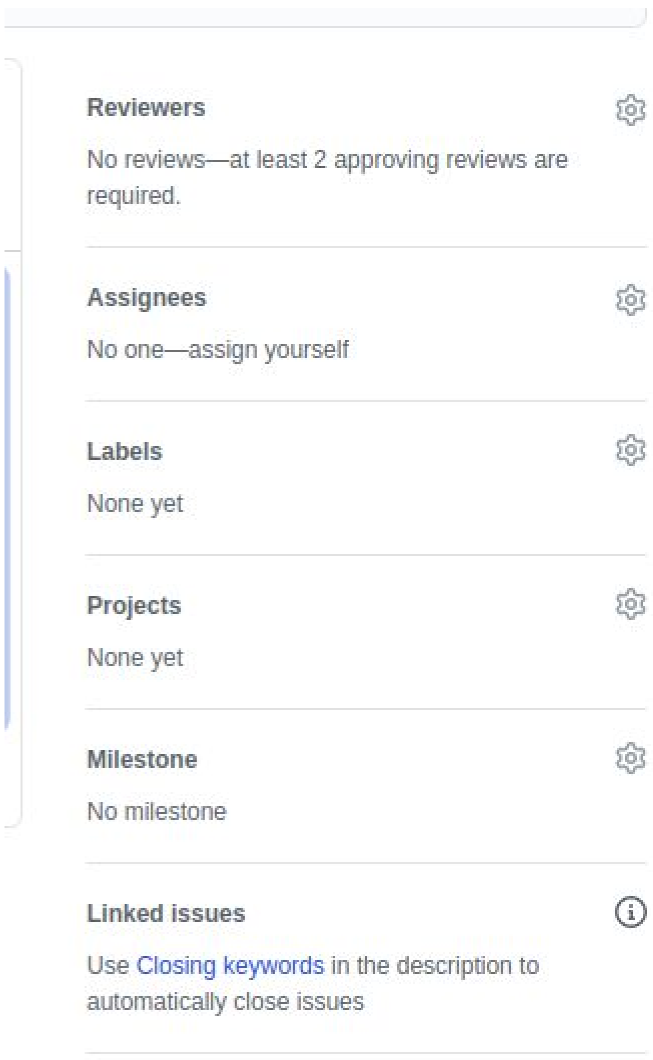


A “pull request” is basically saying… hey I want my code to go into main!

If you don’t see that button you can also create a pull request by changing to your branch in the drop down menu.



Create a pull request for the change you made. You’ll be greeted by a screen where you can add a comment. You’ll also see the following on the right hand side:



Reviewers

No reviews—at least 2 approving reviews are required.

Assignees

No one—assign yourself

Labels

None yet

Projects

None yet

Milestone

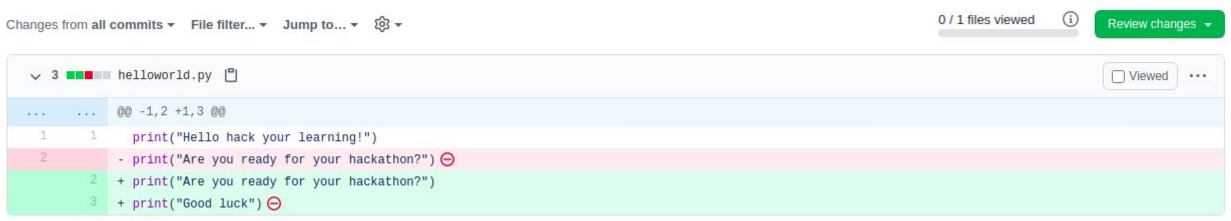
No milestone

Linked issues

Use Closing keywords in the description to automatically close issues

Here you can request a review from your teammates by clicking that little gear icon. You might want to let the reviewers know that the PR is up so they can go and review it.

If you're asked to review, you'll see this screen.



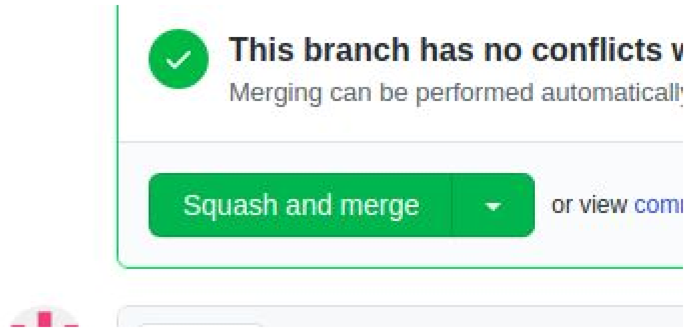
```
3 helloworld.py
@@ -1,2 +1,3 @@
  print("Hello hack your learning!")
- print("Are you ready for your hackathon?")
+ print("Are you ready for your hackathon?")
+ print("Good luck")
```

That little symbol at the end is just warning us that we didn't add a newline at the end. New code will be in green and code that was removed will be in red.

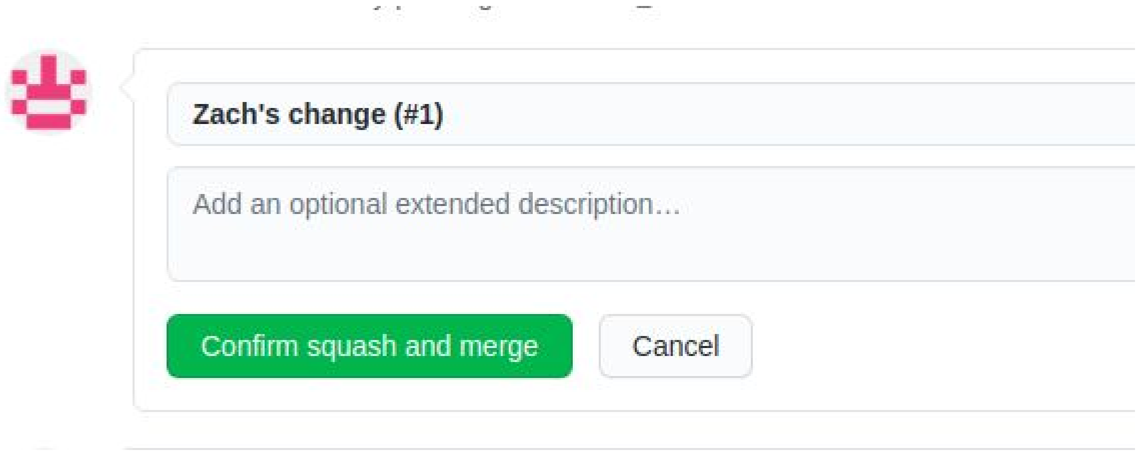
Hit the review button and you can either approve or ask for changes. It’s important that you really understand what you’re approving when you do a review. By hitting approve you’re taking partial responsibility for that section of code.

Squash and merge

Once you have your desired approvals, you can squash and merge your change! Hit the squash and merge button at the bottom:

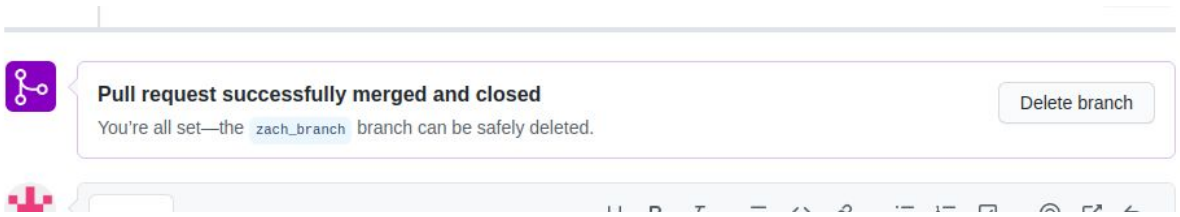

This branch has no conflicts v
Merging can be performed automaticall

And then fill out the description if desired.


Zach's change (#1)

Add an optional extended description…

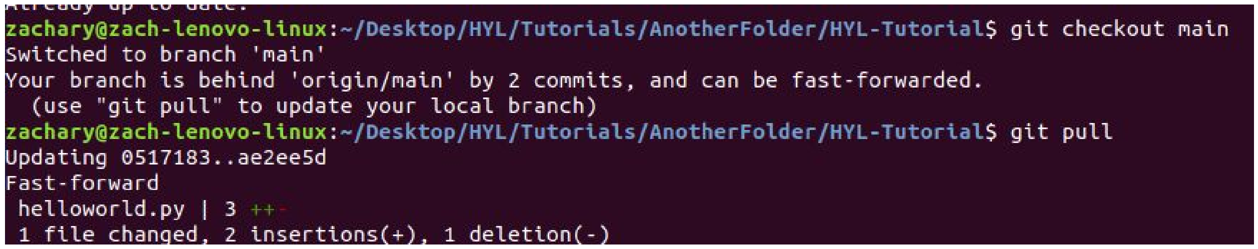
This adds your changes to the main branch as if they were done in one commit (basically squashed together). Delete the branch when you’re done to keep things clean.


Pull request successfully merged and closed
You're all set—the zach_branch branch can be safely deleted.
Delete branch

Git pull

Lastly, how do I view other peoples' changes? Remember that git is never going to update anything unless you tell it to, so when other people squash and merge their pull requests, it won't automatically update the code on your computer. To do that we use "git pull".

For example, to update our local version of the repo with the change we just merged into main we would do the following:



```
zachary@zach-lenovo-linux:~/Desktop/HYL/Tutorials/AnotherFolder/HYL-Tutorial$ git checkout main
Switched to branch 'main'
Your branch is behind 'origin/main' by 2 commits, and can be fast-forwarded.
  (use "git pull" to update your local branch)
zachary@zach-lenovo-linux:~/Desktop/HYL/Tutorials/AnotherFolder/HYL-Tutorial$ git pull
Updating 0517183..ae2ee5d
Fast-forward
 helloworld.py | 3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)
```

Review

Well that's a rap! Today we've covered

- Creating a github account and adding your ssh keys
- Creating your own repository
- Working on other repositories
- Branches and PR's

After these first two github tutorials you can start contributing to software projects left right and center!

Commands

git remote <add|remove> <name> <url>	Manage our remotes
git branch	Show branches

git branch <newname> <oldname>	Create a new branch
git checkout <branchname>	Switch to a branch
git clone <url>	Clone a repository from github
git push	Add your changes to github
git pull	Get changes from github