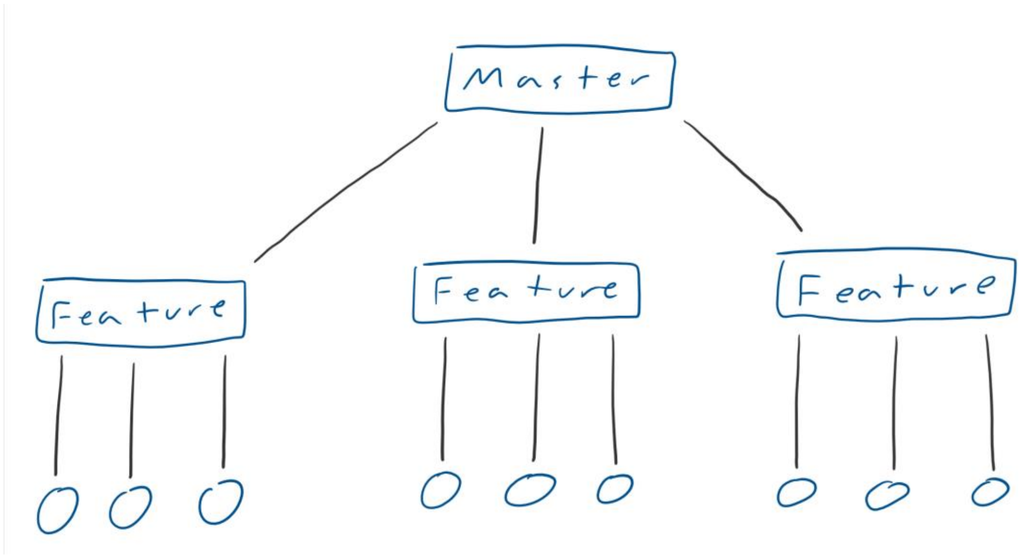
Background:

This is a brief overview of using branches within GitHub. I'm still learning many of these things myself and I'm sure there are many other features that can be taken advantage of here. That being said, please share any further advice or suggestions you may have. Branches are a way to make separate instances of the master branch. Each branch begins as a copy of the master file and gets added to and changed while the master branch remains untouched. This allows the master branch to be made up of only code that is working and ready for deployment while the other branches in the repository are set up to be used for developing code and getting it to a finalized state. This is often used in a commercial setting to allow companies to put out updates without the added code of things that are still a work in progress. I found it helpful to think of the branches as a hierarchy like the one pictured below:



Here the master branch is broken into multiple feature branches. These are branches created to work on a specific project that will later be added into the master branch once they are fully functional. Below that, we have the third layer of branches. These circles were drawn to represent the individual people working on each feature. Each person has their own individual branch which functions as a place for them to upload and backup code that they don't want to lose but that isn't quite ready to move up into the feature branch. It is important to note that each branch along with their contents is visible to all contributors meaning that even if you upload code to your own personal branch it can be seen by everyone. It is also worth mentioning that code will not just flow up the branches, often code will be copied back and forth between the feature branch and the personal branches.

Throughout this walkthrough I'm assuming that you have created and your own local repository, set up git, and are now familiar with the pull, push, and commit commands. If you aren't quite there, make sure to check out Mary's "github101" as it is a very helpful step by step walkthrough of those.

Creating a Branch:

1. First, navigate to your local repository. This is what it looks like on my machine:

```
$ cd Coding/HTCeramics
```

2. Then, use one of the following commands to create your branch. Give it a name reflective of its purpose. Ex/ if this is a personal branch, include the feature and your name while if this is a feature branch just use the name of the feature.

   # To create a new branch that is a copy of the branch you are currently in:

   ```
   $ git branch name_of_new_branch
   ```

   # To create a new branch that is a copy of a branch you are not currently in

   ```
   $ git branch name_of_new_branch name_of_branch_to_copy
   ```

Workflow Commands:

1. The first thing I first step is to navigate to the directory in which you have the HTCeramics folder stored. To do this use the "cd" command. This might look something like this:

   ```
   $ cd Coding/HTCeramics
   ```

2. Once you are in the right directory use the pull command to get all the latest changes from the remote/online GitHub repository.

   ```
   $ git pull
   ```

3. Next, we need to select which branch we would like to work on. This is done using the checkout command which selects the desired branch (peak_fitting_rikhof in this case) from the local repository on your computer. This does nothing with the online repository. You will also notice that files displayed in your file navigator will change to reflect those within the branch you are working on.

   ```
   $ git checkout peak_fitting_rikhof
   ```

4. Now that we are looking at the branch we would like to be working in, it might be good idea to run the merge command in order to merge the changes between this branch and another one. This is not necessary if you are not planning on pushing the files up to your remote personal branch but rather the remote feature branch. I that is the case you can skip this step and just work in your local feature branch, uploading to the remote one once finished.

   ```
   $ git merge branch_to_merge_with
   ```

5. We can now go ahead and begin working on code, being sure to save it in the HTCeramics folder.
6. Once we have made changes to the code that we would like to upload, the following command will let us see which of these changes git has noticed. After running this, the files that it lists in red are those that git has detected changes in but has not yet started tracking and saving those changes. If we were to switch branches at this point, those changes would be deleted and lost forever.

   ```
   $ git status
   ```

7. The following command will add all of the detected changes to a local database where git can track them.

   $ git add .

8. We now need to commit these changes to our local repository for them to be fully saved on our machine. -m in the following command allows you to add a message that will go along with your commit in order to give others an idea of what this change does, once you push the file to the remote repository.

   $ git commit –m "type your message here"

9. The final step is to push our changes from our working branch in our local repository up to the corresponding branch on the remote repository. Just make sure to push your changes to whichever branch best reflects the stage of development that the code is now at.

   $ git push –u origin peak_fitting_rikhof

<u>Pull Requests:</u>

A "pull request" is a way to merge the changes between 2 remote branches through the GitHub website. This is generally used when a feature branch has been finished and is now ready to be added to the master branch. This is a fairly simple process that the GitHub website will walk you through after you click the button labeled "pull request" on the main page. Once you have created a pull request it can be reviewed by other people before it goes through. It also allows you to see each of the changes side by side in each file between the 2 branches you are merging.