# Github/Git Primer

Tyler Hague

# Why Use Github?

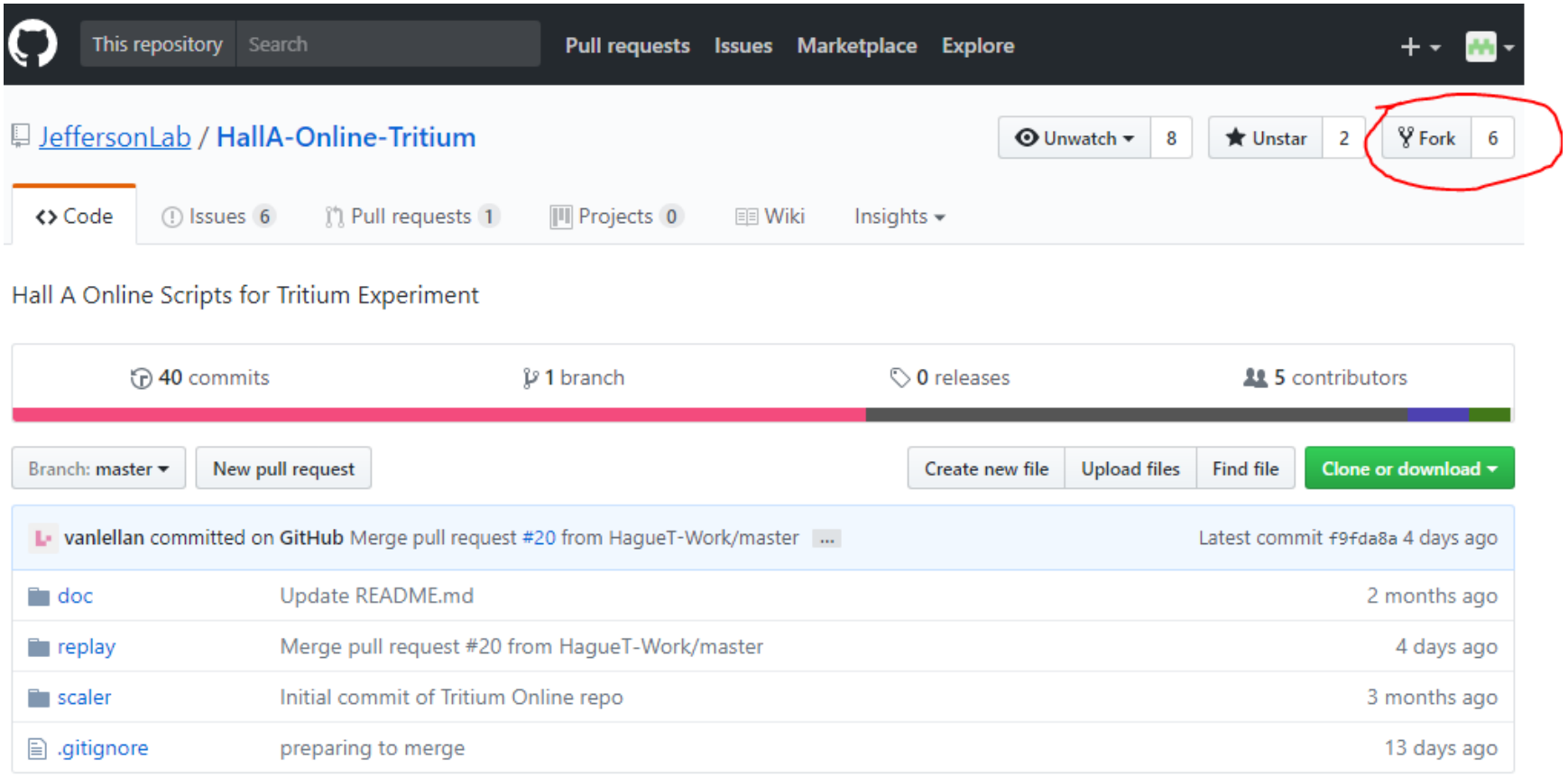
- Github keeps all of our code up to date in one place
- Github tracks changes so we can see what is being worked on
- Github has issue tracking for keeping up with what needs work
- Github has collaborative tools for conflict prevention

# What workflow should we follow?

- The master repository is our “official” code. This is where we keep working code and should not be used for any tests.
- To work on the code, make a fork and work there.
- For ‘tests’, create branches to avoid affecting your functional code. These can later be merged into the master code.
- When the work you set out to do is complete, create a pull request to put it in the master repository.

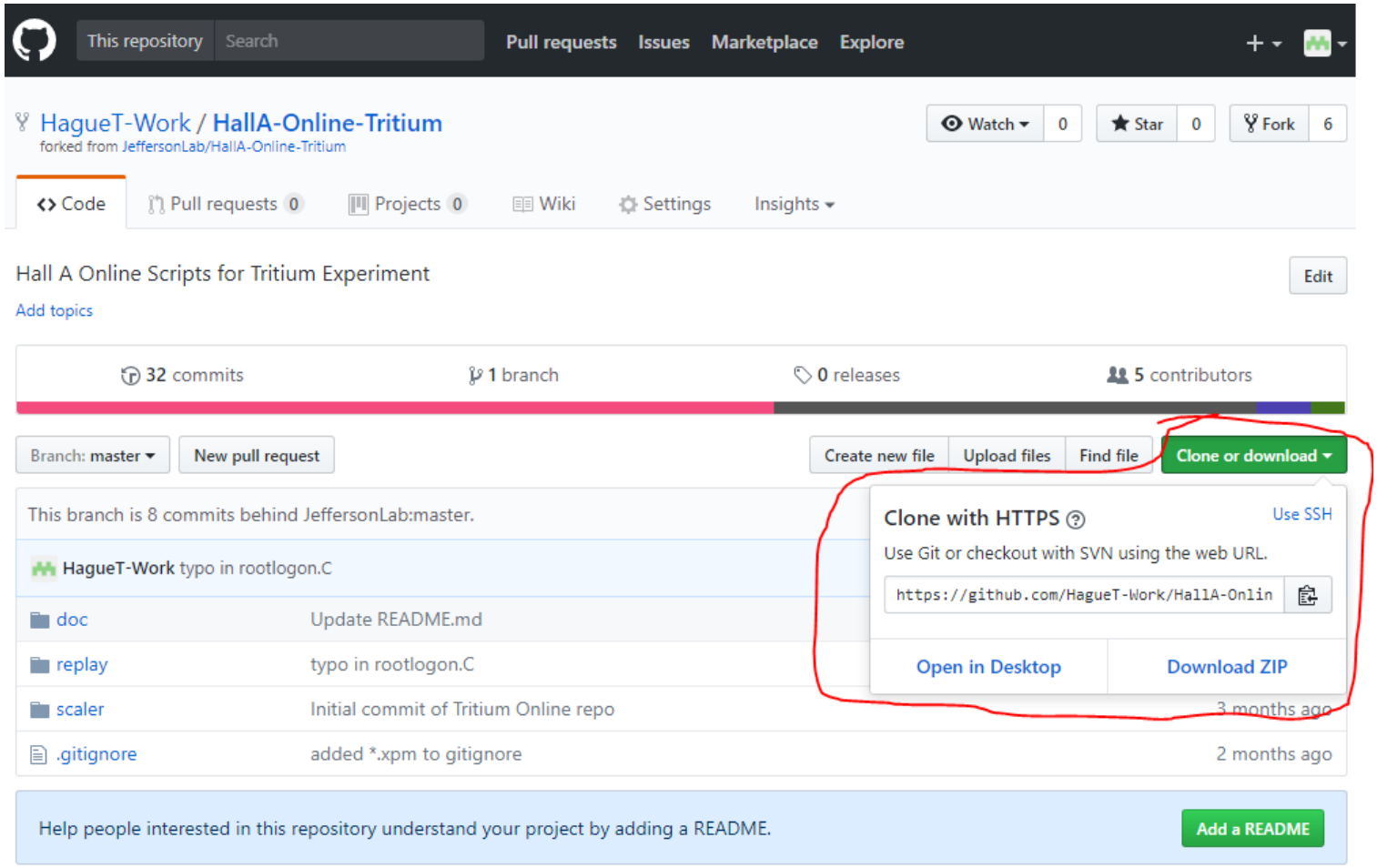
# How do I do this?!

https://github.com/JeffersonLab/HallA-Online-Tritium



# Download your fork

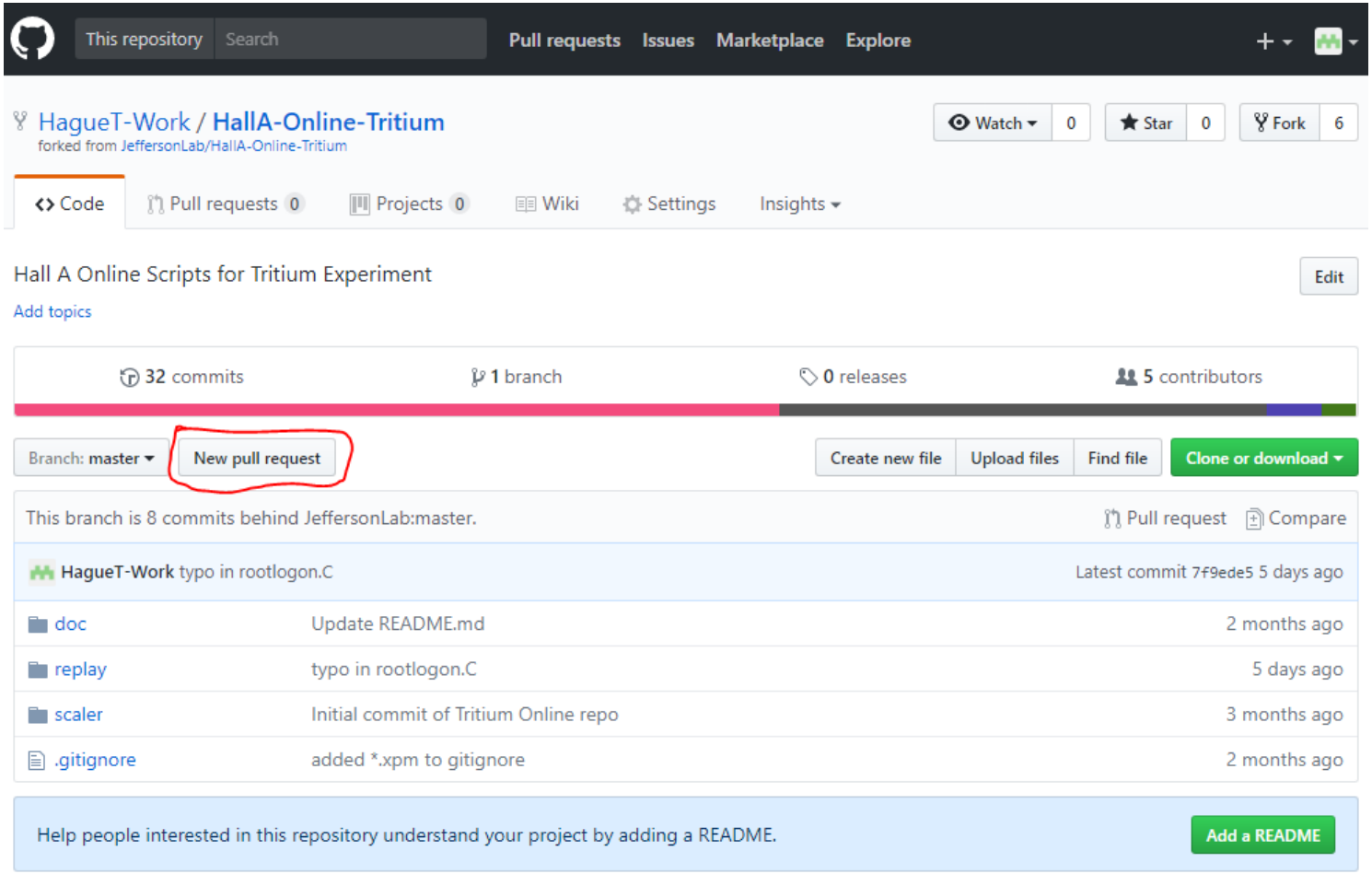
Use the command: “git clone <YOUR_GIT_URL>”



# What do I do when I've made changes I want to keep?

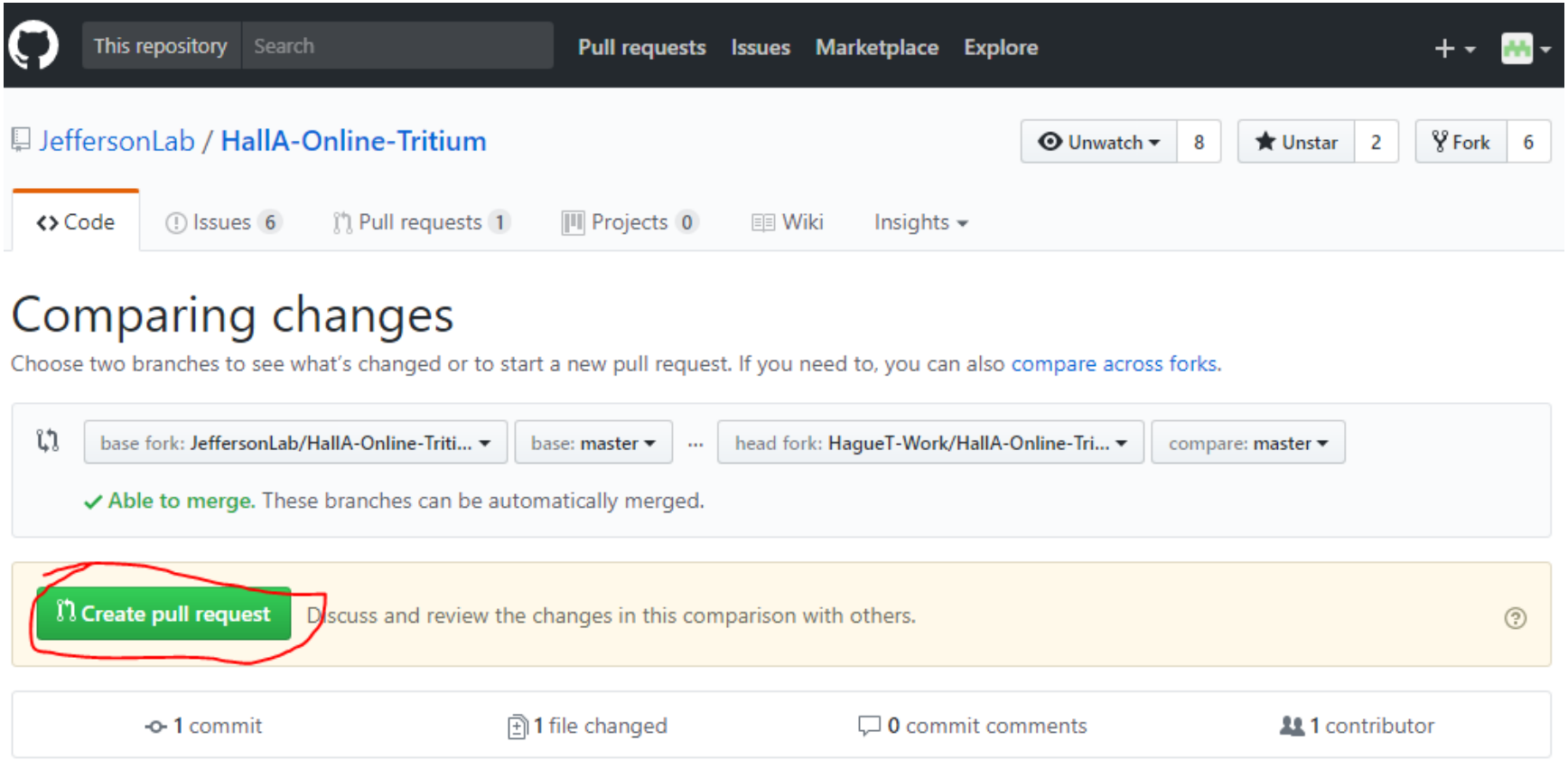
- Commit your work
  - "git add" – select the individual files that you would like to have included in the next commit (or use "--all")
  - "git commit" optional flag "--all"
  - This will open a text editor to provide a commit message
    - First line should be a brief overall description of what you did
    - On the next line you can begin a detailed description of the work if necessary
  - This creates a local commit that you can reference and track
- Use "git push" to move your local commits to your github fork
  - You will be prompted for your github login info

# I’m ready to move my code to the master repository. Now what?



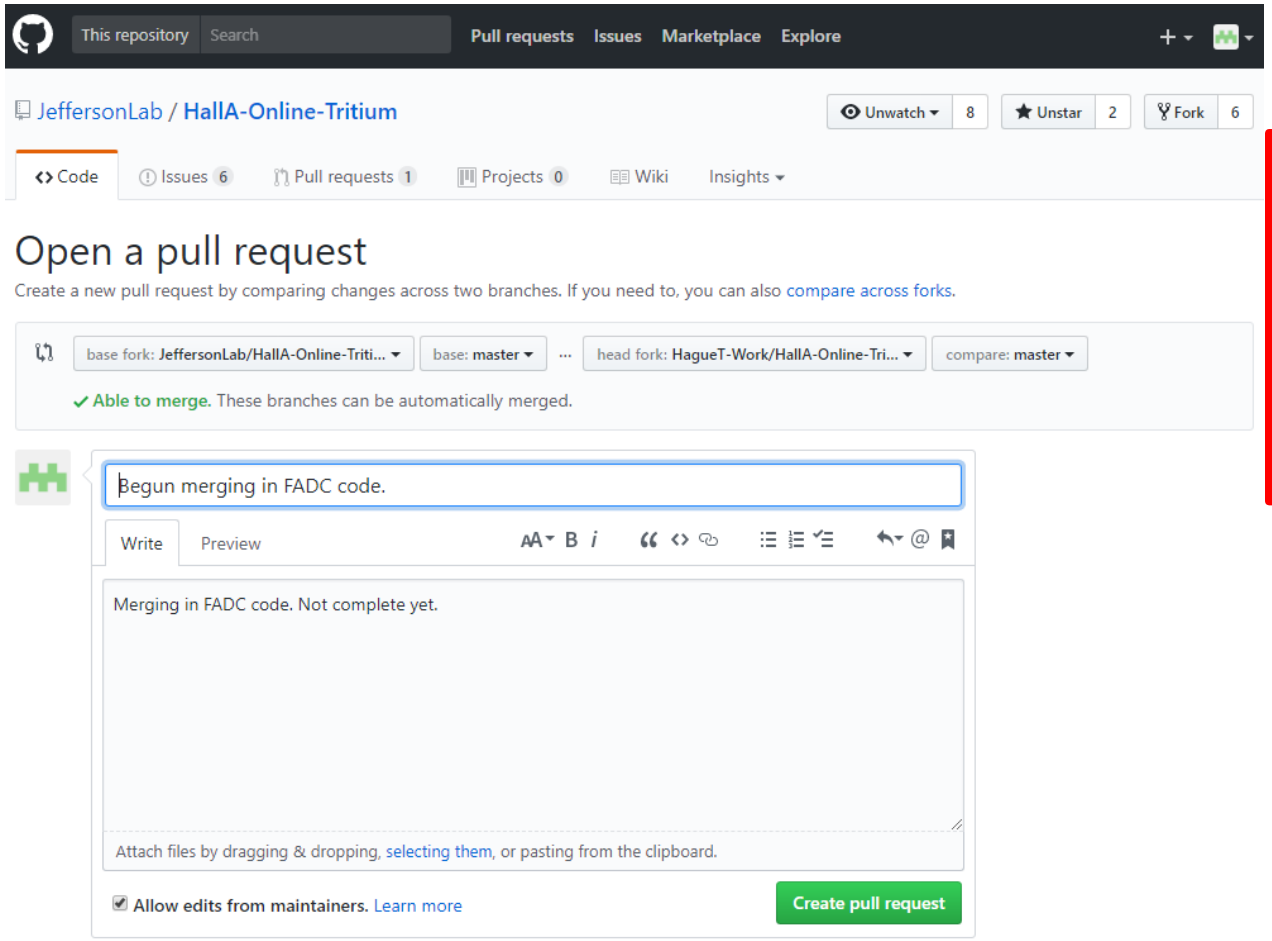
# I’m ready to move my code to the master repository. Now what?

Lower on the screen it will show the differences between the current code and your code



# I’m ready to move my code to the master repository. Now what?

It will ask you for a pull request message. This should describe what your changes accomplish. This will then be sent off to the maintainer of the repository (Evan) for approval.




*Note that I’m not actually making this pull request. As stated in the message, this work is not complete. Don’t put incomplete code in the main repository.

# If I'm working outside of the main repository, how do I keep my code up to date?

- Must connect your local (downloaded) code to the upstream (master) repository.

1. "git remote add upstream <URL_OF_MASTER_REPO>"
2. "git fetch upstream"
3. "git checkout master"
4. "git merge upstream/master"
5. "git commit --all"
6. "git push"

# What if I want to work on a side-project or test?

- Create a branch!
- A branch allows you to create a set of tracked changes separate from the master branch. The branches exist within your fork of the repository.
- Relevant commands:
  - “git branch” – shows a list of branches available on your machine
  - “git branch <name_of_branch>” – creates a branch with the name specified
  - “git checkout <name_of_branch>” – begin working on code in the specified branch. “-b” flag will create the branch if it doesn’t exist.
- If you decide that the branch is something that should be merged into the master branch:
  - “git checkout master”
  - “git merge <name_of_branch>”

# What if there are conflicts in the merge?

- Your code will enter merge mode.
- The merge failure message will tell you which files have conflicts.
- When you edit these files, you will see extra lines indicating where the conflicts are. You must manually decide which code should be kept and which should be deleted.
- After you do this, you can commit the changes and the conflict resolution is complete.

# Merge Mode Example

Person1 and Person2 have been working independently on scaler code.
Person1 decided to merge Person2's code with theirs. This has led to a conflict.

```
[me@centos7 replay]$ git checkout person1
Switched to branch 'person1'
[me@centos7 replay]$ git merge person2
Auto-merging replay/replay_tritium.C
CONFLICT (content): Merge conflict in replay/replay_tritium.C
Automatic merge failed; fix conflicts and then commit the result.
[me@centos7 replay]$
```

This message tells us which files have conflicts. In this case, we should look at "replay/replay_tritium.C" to fix what is happening. Then, the changes can be committed.

# Merge Mode Example

When we look at “replay/replay_tritium.C” we see that both person1’s code and person2’s code is shown. Since we are in branch “person1” that is the “HEAD” branch. The conflicting code is shown after the “=======“. Person1 must now choose what code to keep and commit the changes.

```
    //==================================
    //  Scalers
    //==================================
    if(bScaler){
<<<<<<< HEAD
      THaScalerEvtHandler* rscaler = new THaScalerEvtHandler("Right ","HA scaler event type 140 on R-HRS");
      gHaEvtHandlers->Add(rscaler);

      //Hi, I'm person1 and I'm working on some scaler stuff
      scalerstuff->DoPerson1Things();
=======
      //Hi, I'm person2 and I'm also changing some scaler stuff. Person1 and I did not communicate
      //about our work so we are going to have some merge conflicts. Let's see what happens.

      //THaScalerEvtHandler* rscaler = new THaScalerEvtHandler("Right ","HA scaler event type 140 on R-HRS");
      //gHaEvtHandlers->Add(rscaler);

      newscalerstuff->DoPerson2Things();
>>>>>>> person2
    }
```

*Merge conflicts will happen when many people are working on the code. The comments here are not meant to imply that communication breakdown is the only cause of merge conflicts. However, when a conflict happens those who wrote the conflicting code should communicate in order to reach an agreeable solution. Don’t simply delete the other persons code without good reason.

# Merge Mode Example

After the discussing the problem with Person 2, a solution was found. Person1 made the made the changes and completed the merge by committing them to their repository.

```
//==================================
//  Scalers
//==================================
if(bScaler){
  THaScalerEvtHandler* rscaler = new THaScalerEvtHandler("Right ","HA scaler event type 140 on R-HRS");
  gHaEvtHandlers->Add(rscaler);

  //Hi, I'm person1 and I'm working on some scaler stuff
  scalerstuff->DoPerson1Things();
  //Hi person2. I'm uncommenting the original scaler stuff because it's critical for my code.
  //I'm leaving in your scaler things because we discussed why it's necessary.
  //The merge is now complete and I can commit this to my branch.

  newscalerstuff->DoPerson2Things();
}
```

```
[me@centos7 replay]$ git commit --all
[person1 c1f0bcc] Merge branch 'person2' into person1
[me@centos7 replay]$
```

# When in doubt, ask an expert!

- Sometimes that's me.
- A lot of times that's Evan.
- If we don't know, we'll do our best to help you figure it out so that we can all better use this tool.

# Questions?