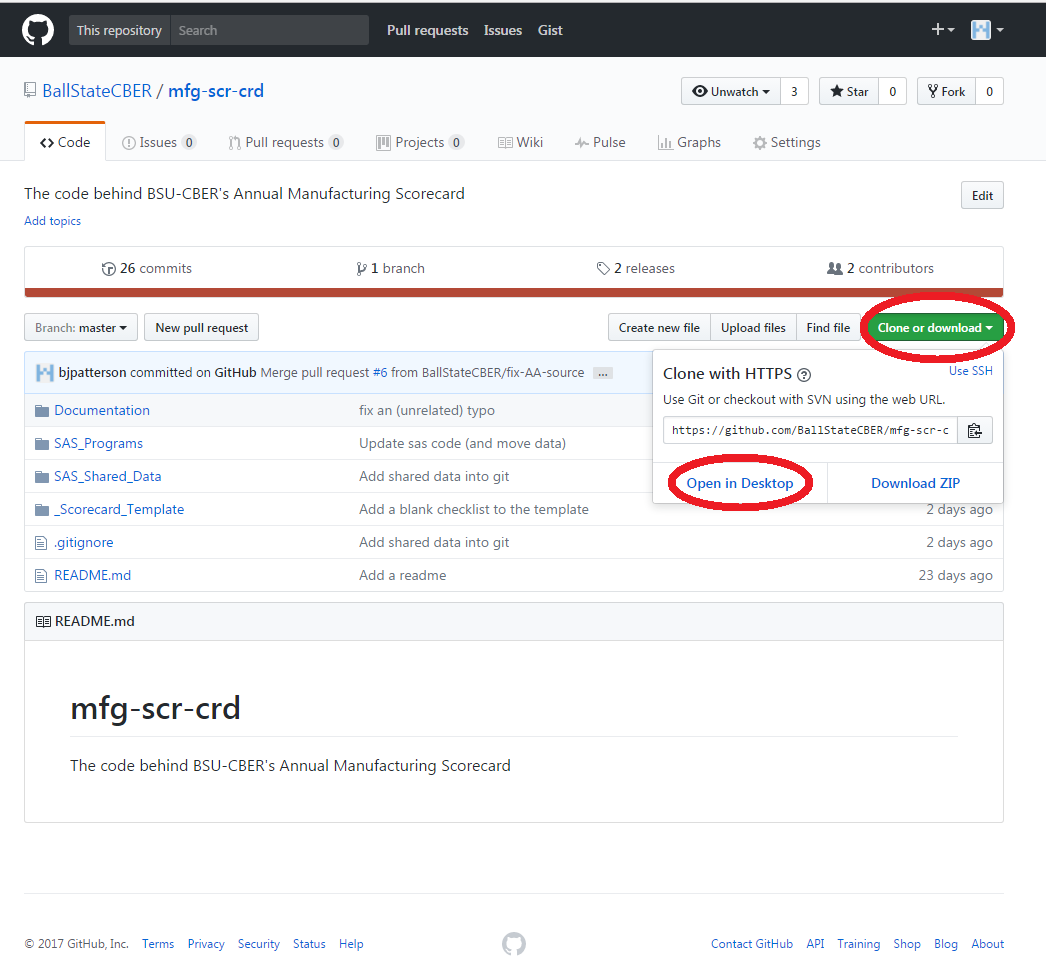
Making Changes to Scorecard Code

If you ever need to make changes to the Manufacturing Scorecard code, it is important that you follow proper “git-flow” procedure. This will keep the codebase easy to manage, and ensure that we have a proper history of changes that have been made.

# Getting Started

1. Download and install the GitHub Desktop client. (<https://desktop.github.com/>)
   1. (You will need an administrative login to complete this installation.)
2. Get appropriate GitHub login credentials. (Ask a supervisor what’s appropriate.)
   1. This may mean creating a private account, or using an existing student account.
3. Navigate to the scorecard code repository on github.com
4. Download the latest version of the repository using the GitHub Desktop client.
   1. (If your browser gives you a popup asking about opening a program, hit “OK”.)



# Creating a “Feature Branch”

It’s a best practice to ***never* modify the main code directly**. Instead, we make a copy of the code (called a “branch” in git terminology), modify the copy, and when we are satisfied with our changes, we “merge” the new branch back into the main code.

This allows us to test new changes to code without disrupting what already exists. (Branching also allows multiple people to work on the code at once without disturbing each other, which is less important for our purposes, but extremely useful in large production environments.)

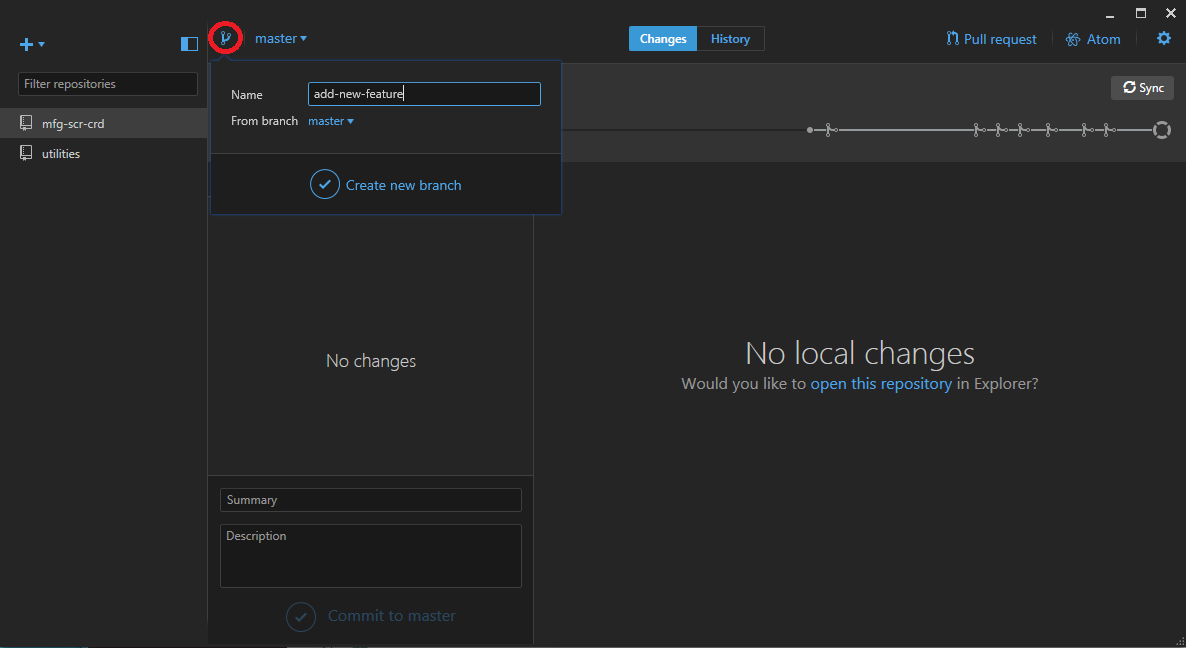
It’s *also* a best practice to **only make *closely related changes* in a single branch**.

For example, if you want to add a new data source to the Human Capital scorecard that requires changes to several files, a single feature branch makes sense. If you’re adding multiple related datasets, a single branch might still be acceptable.

If you wanted to add a new data category, change the output files from CSVs to Excel documents, and fix a bug that was causing the code to crash, you should really open a separate branch for each of these changes. (This keeps the code easy to review.

**To create a branch in the GihHub client:**

1. Select the repository you wish to work on (on the left hand side).
2. Click the branch icon (highlighted in the image below).
3. Name the branch something informative (for example “add-inflation-index”)
   1. Note that there are some naming restrictions. In particular, names can’t contain spaces.
4. Make sure you’re branching off of the correct location.
   1. (Generally, you’ll want to branch from “master”, which is the “production” code.)
5. Hit the “Create new branch” button.



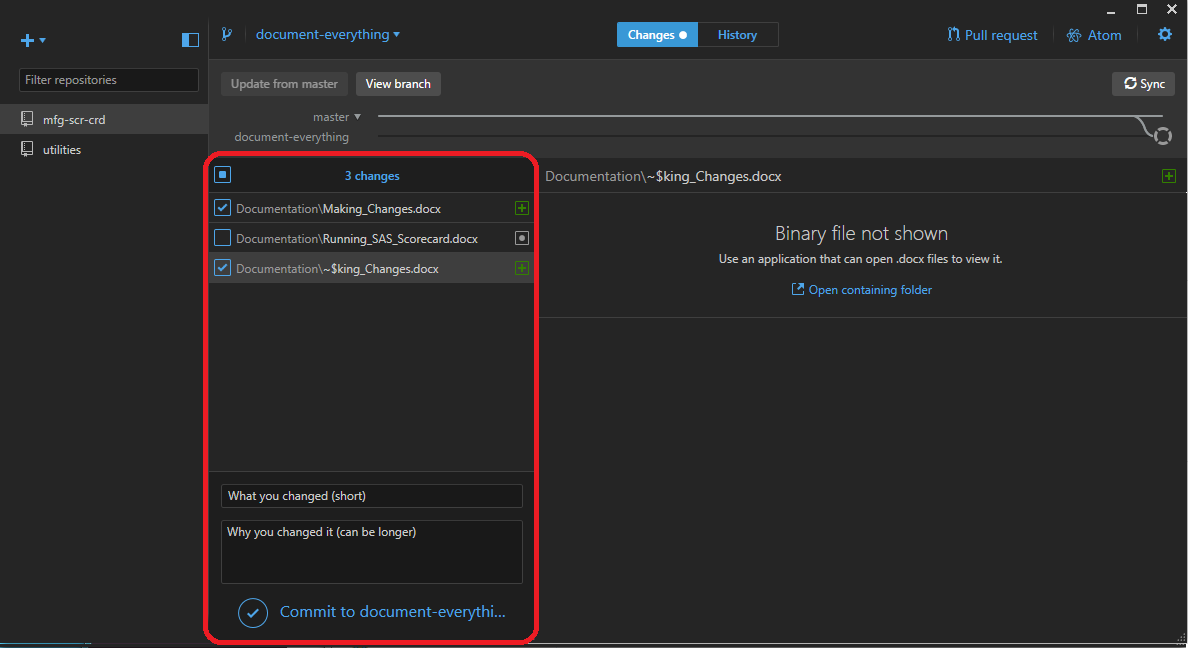
# Modifying the Code

**Once you’ve created a feature branch**, you’re ready to start modifying the code. You can safely make any changes you want (including modifying/adding/deleting files) without fear of destroying the original version.

# “Committing” Changes

After you’ve made changes, and you’re happy with your work, you need to “commit” your changes to GitHub.

In the GitHub client, select the files you wish to commit, fill out the details of the changes at the bottom of the screen (summary = what you changed, description = why you changed it), and hit the “commit” button. This saves the changes locally to your machine.



Once you’ve committed changes locally, you need to let the GitHub website know about the new changes.

# Testing the Code

Once you’ve finished modifying the code, it’s time to test the changes.

If you make changes to code, **it’s your responsibility to test that they work as intended.** You’re not finished with the new code until you’ve done a test run, and verified that everything looks OK!

The two most common problems to watch out for are:

* Code that crashes, and doesn’t finish executing
* Code that executes with unexpected results (such as *everyone* getting an “A” in a Logistics)

# Submitting a “Pull Request”

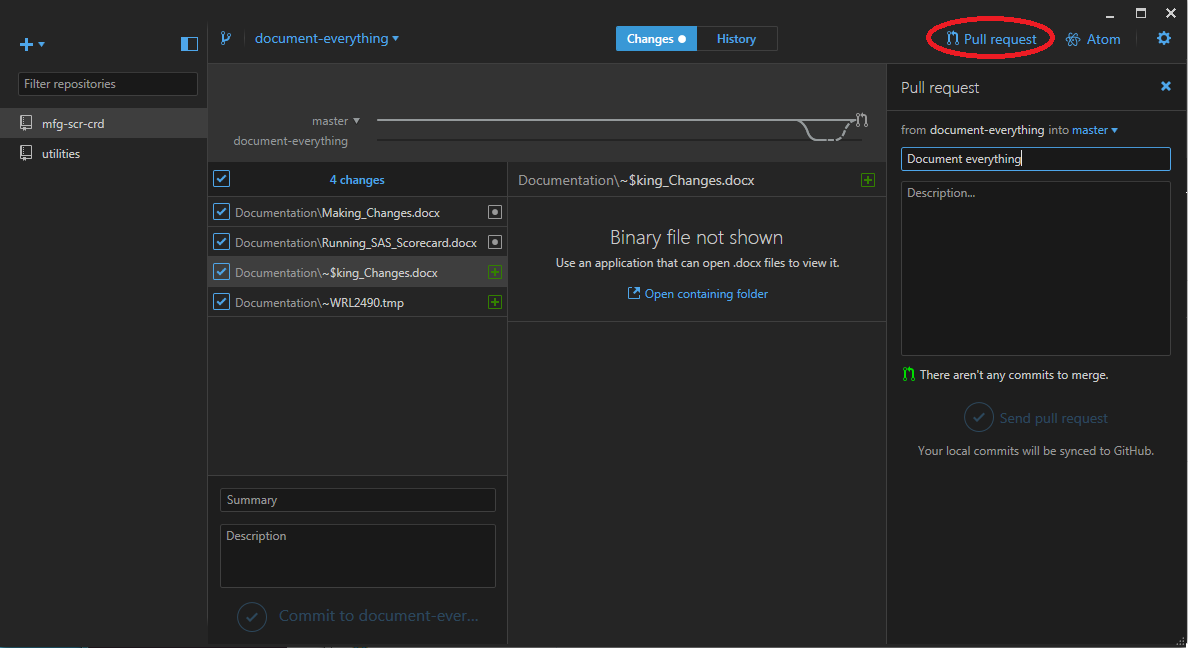
**Once you’ve tested your new code,** it’s time to merge the new changes into the main code. This is done via a “pull request”. This lets the code owners know that you’ve done some new work, and gives them some tools to view the changes you made and have a discussion about the changes before finalizing the merge.

**To create a pull request:**

1. In the GitHub Desktop client, click the “Pull Request” button.
2. Fill in the details for the pull request (this should provide enough detail that someone else will have an idea of what changes you made).
3. Send the pull request.
4. Have someone else review the changes you made.
   1. (GitHub’s website has some very helpful tools for reviewing changes to code efficiently.)
5. Discuss, and make additional changes as needed.
   1. (You can continue to commit and sync new changes, and they will be added to the pull request.)
6. **Let someone else (a supervisor) merge your code when they’re satisfied that it’s complete.**

It’s another best practice to **never merge your own pull requests,** when you can avoid it. If there’s *anyone* else available to review your code, you should try to have them review it. When they’re satisfied that the changes you’ve made are correct (and easy to understand), they should merge the code for you.

This review process prevents many bugs from being introduced into the main codebase, encourages good coding style, and makes sure that someone other than you knows what changes have been made to the code.

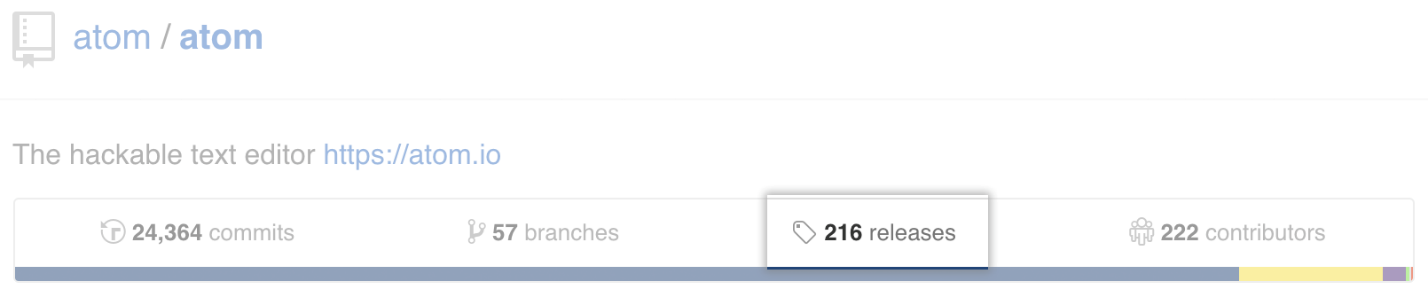
(The review process also highlights the importance of keeping your branches focused on a single change, so that reviewers can easily understand any changes you’ve made.)

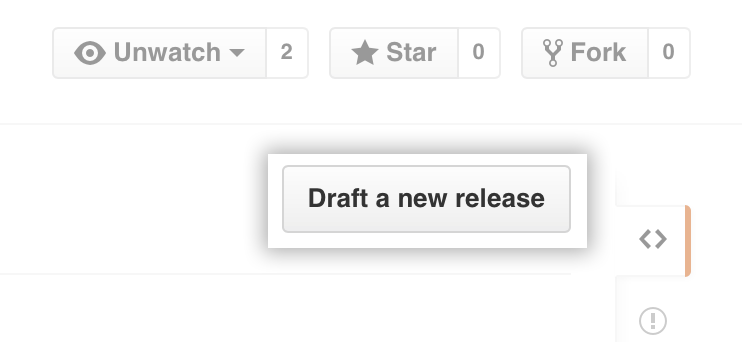
# Tagging a Release

Once the Manufacturing Scorecard has been finalized for the year, it’s a good idea to tag the version of the code that was used. **This is especially important if any changes have been made.**

This allows us to return in the future, after new changes have been made in the code, and see what the code looked like several years back (in case we needed to double check a calculation, etc.

**To tag a release in code:**

1. View the codebase online.
2. On the “Code” page, select “releases”
3. Select “Draft a new release”



1. Name the version after the year it was run (ex. `v2017`), and be sure to tag the master branch.
2. Fill out the Title and Description, paying special attention to note any changes that were made for this version of the code.
3. Publish the release