# Why a Coding Workflow?

The FEAR coding workflow aims to provide a mechanism for ensuring code is designed to meet its objectives, is easily accessible by all team members, and is adequately tested and reviewed. Additionally, the workflow follows that used by GitHub and many companies, which will prepare the FEAR programming team for future projects beyond FRC.

The coding workflow contains the following high-level steps:

1. Identify work to be done and create a story
2. Begin a new programming checklist and design form(s)
3. Create a branch for the topic
4. Implement the story and perform daily commits
5. Test the newly implemented code and perform a code review
6. Perform a pull request
7. Merge pull request into master branch
8. Complete and merge a second development topic (optional)
9. Create a release

# Identifying work to be done and creating a story

The work performed by the programming team should be broken down into “topics”, or functionality which can be implemented by the team in one week’s time. In order to avoid complexity, plan to execute no more than two simultaneous topics per week on a particular project.

* Work with a mentor to identify a topic that can be implemented and tested within a week's time.
* Create a Post-it for the topic and put it on the ***Planned*** section of the scrum board.

# Beginning a new programming checklist and design form(s)

The programming checklists and design forms help to ensure all necessary programming steps have been taken, helping to guarantee code quality.

* Download and print a new programming checklist for each new topic. Complete the Story Details section of the programming checklist.
* Download and print design form(s) for your feature. Depending on the complexity of the feature, you may need multiple design forms.
* Complete the design forms before coding starts.
* Move the Post-it to the ***WIP*** section of the scrum board.

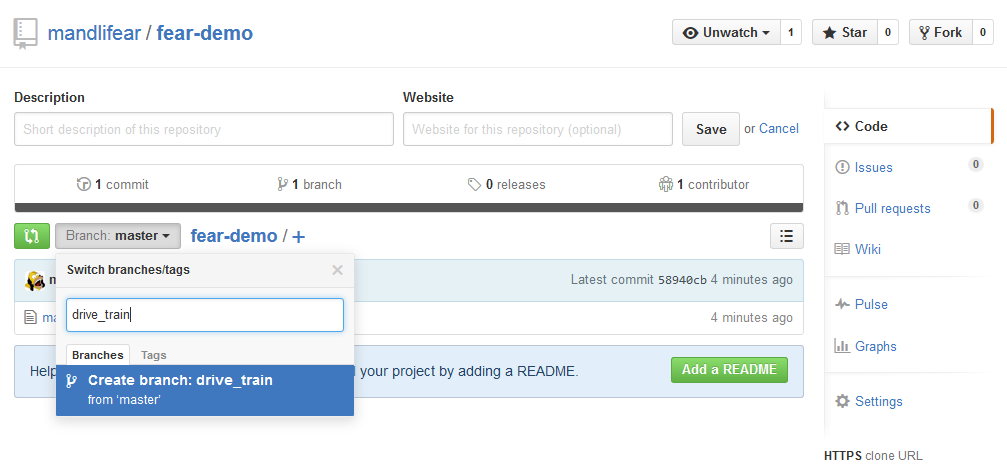
Links to the current forms can be found on the home page of the Programming Manual in GitHub.

# Creating a branch for the topic(s)

Separate branches should be created for each of the topics to be worked on. Working on branches allows us to develop and test new features without affecting the master branch. As a result, **the master branch stays clean, and contains only quality, released code**.

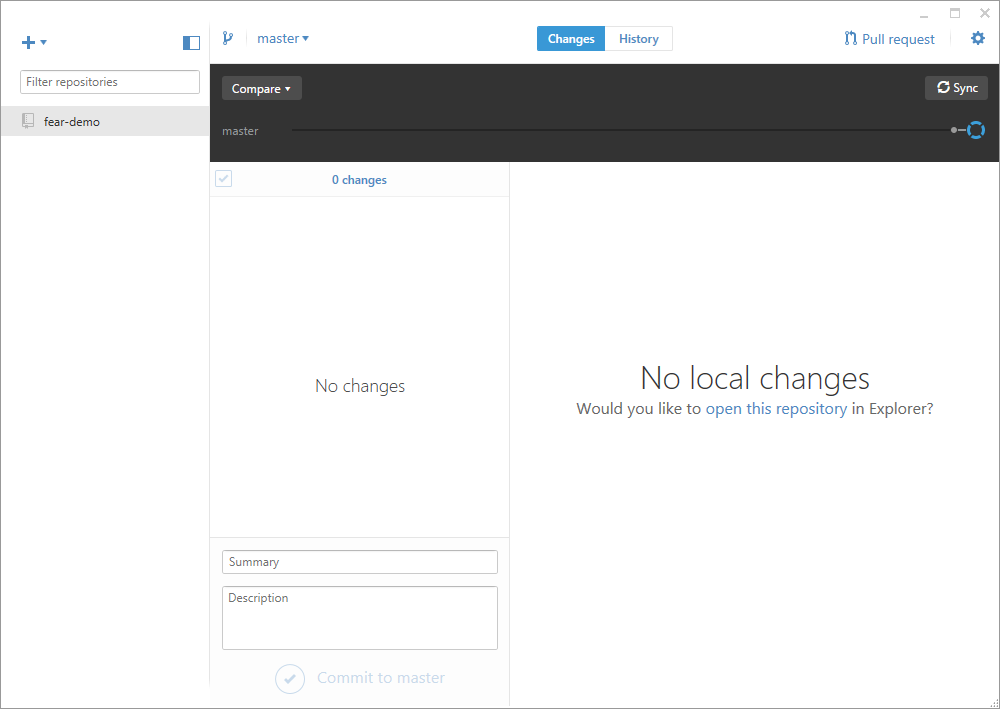
The remainder of the document details how to use GitHub and GitHub Desktop to do code development on separate topic branches. The example assumes two teams are working on features; one on the drivetrain, and a second on the game mech.

* Create new topic branches ***drive\_train*** and ***game\_mech*** from the master branch in GitHub. *Always create new branches on GitHub instead of GitHub Desktop to guarantee they are created from the latest version of the master branch.*

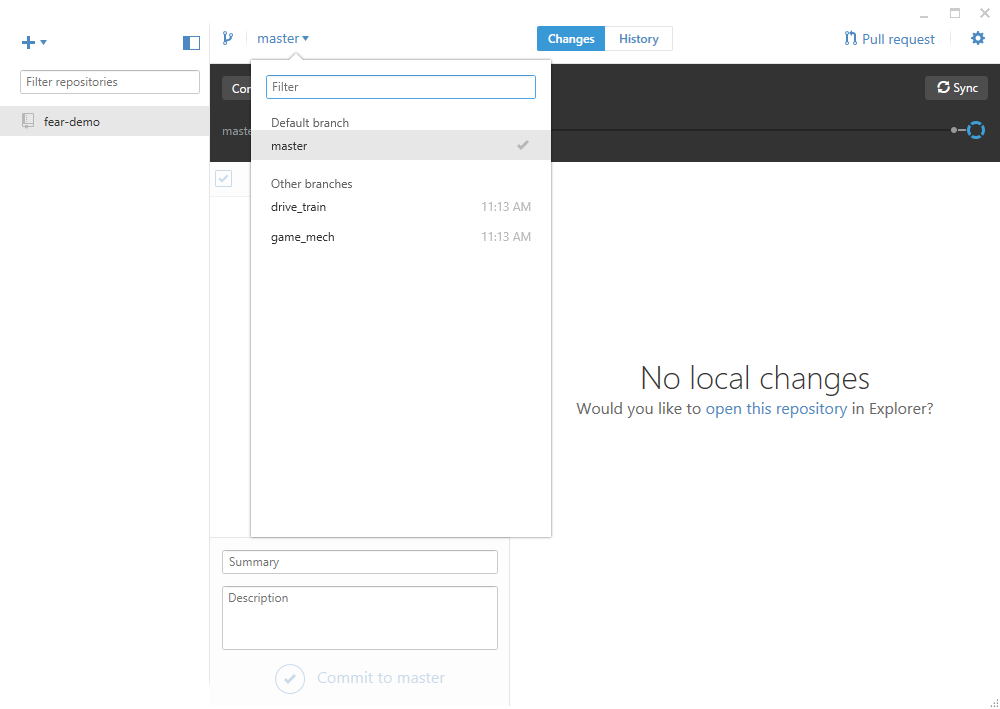


|  |
| --- |
| *For those who prefer the command line:*  git checkout master  git pull  git branch <new branch name>  git push <new branch name> |

* From GitHub Desktop, make sure the correct repository is selected on the left-hand side and click the Sync button to synchronize your local copy of the repo with the one on GitHub.



* If you click on the branch selection drop down menu, you can see that the newly created branches are present:

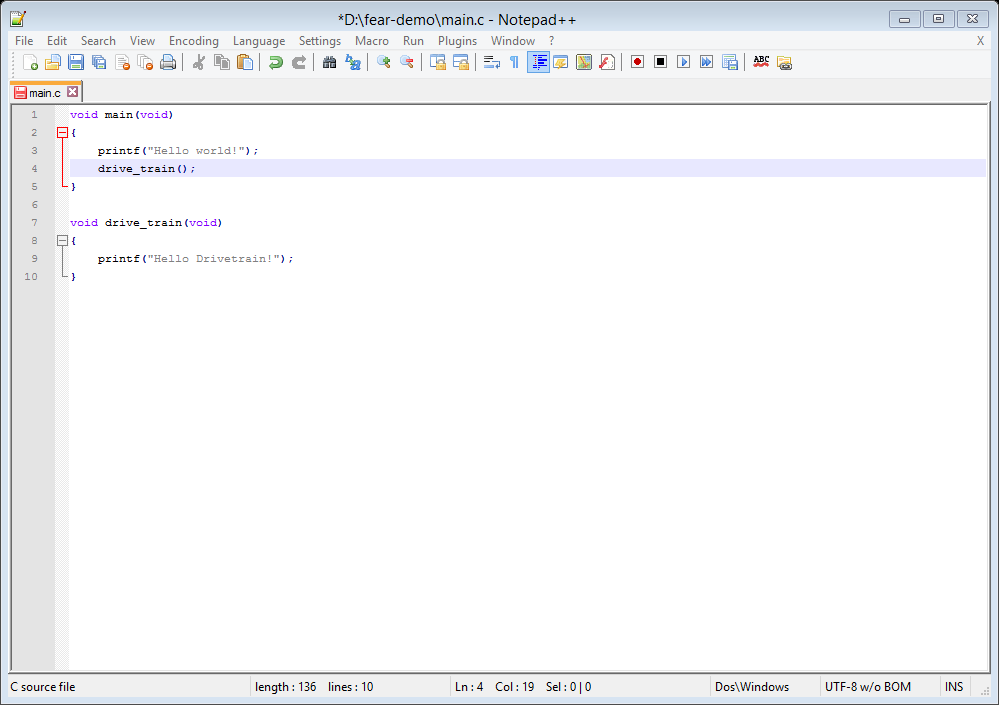


# Implementing the story and performing daily commits

* From the branch selection drop down menu, select the topic branch you will work on. For this example, let’s start with ***drive\_train***.

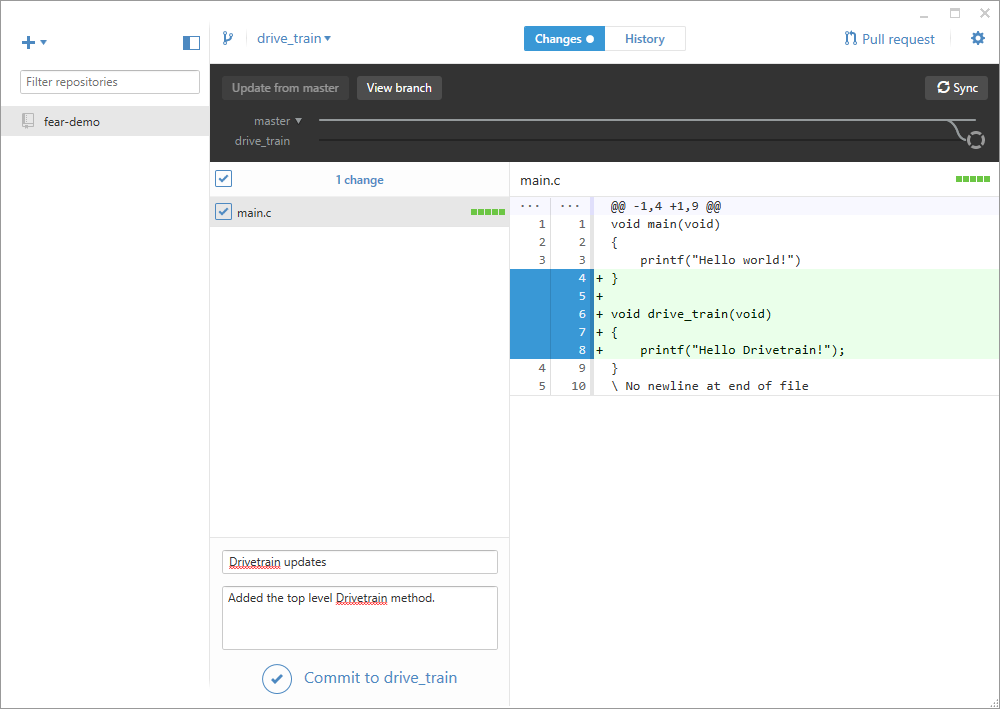
|  |
| --- |
| *For those who prefer the command line:*  git checkout <branch name> |

* Navigate to the repository on the development PC and make code changes. In this case we are adding the ***drive\_train()*** method to ***main.c***.



* When the changes are complete, navigate back to GitHub Desktop.
* Select the ***Changes*** button at the top of the window, and then select the filename. The right-hand side of the window will show the changes and additions.
* Enter a meaningful summary and description of what changes were made, and click ***Commit to drive\_train*** to commit the changes to the local GIT repository on the development machine.

|  |
| --- |
| *For those who prefer the command line:*  git add <filename>  git commit |



* Click the ***Sync*** button to push the changes up to GitHub.

|  |
| --- |
| *For those who prefer the command line:*  git push origin HEAD |

* For normal development, continue to make additional changes to the code, and **make sure to commit and sync your changes at the end of each working session**. By doing so, your teammate can easily pick up your work.

**Good communication is key for when developing code in a team setting! Make sure your commit messages are detailed, and you keep track of your progress in the design notebook.**

# Testing the newly implemented code and performing a code review

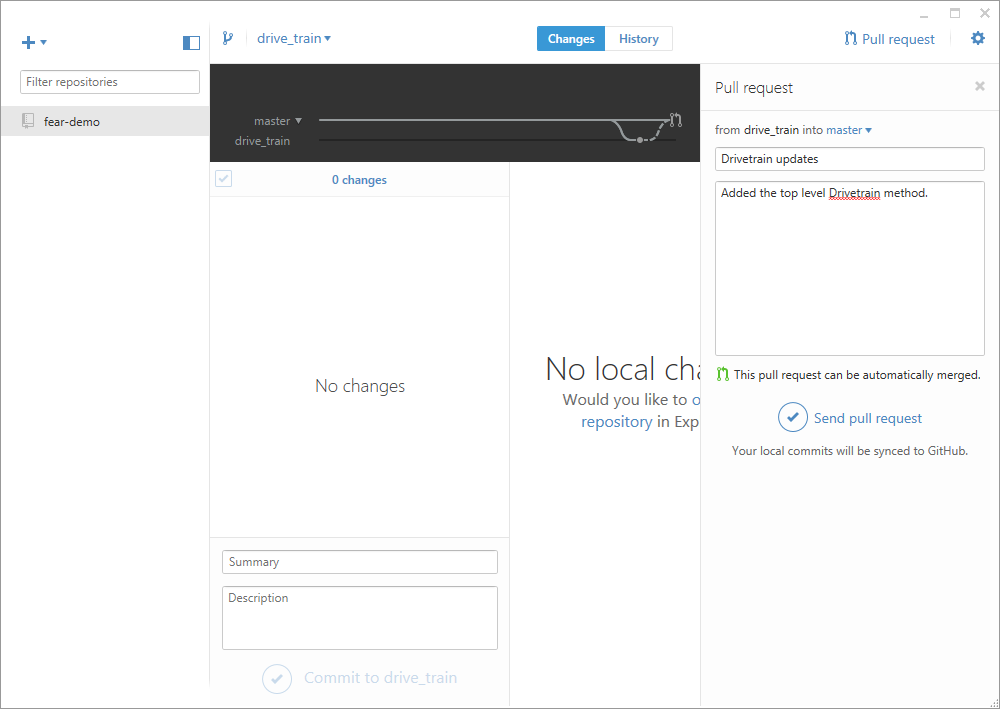
Once the implementation is complete, it must be tested to that specified in the design forms, after which a code review must be performed.

* Complete and fill out any applicable items in the ***Implementation***, ***Testing*** and ***Code Review*** sections of the Programming Checklist for the topic.

# Performing a pull request

A pull request indicates to the rest of the team your branch is ready to go, and can be merged into the master.

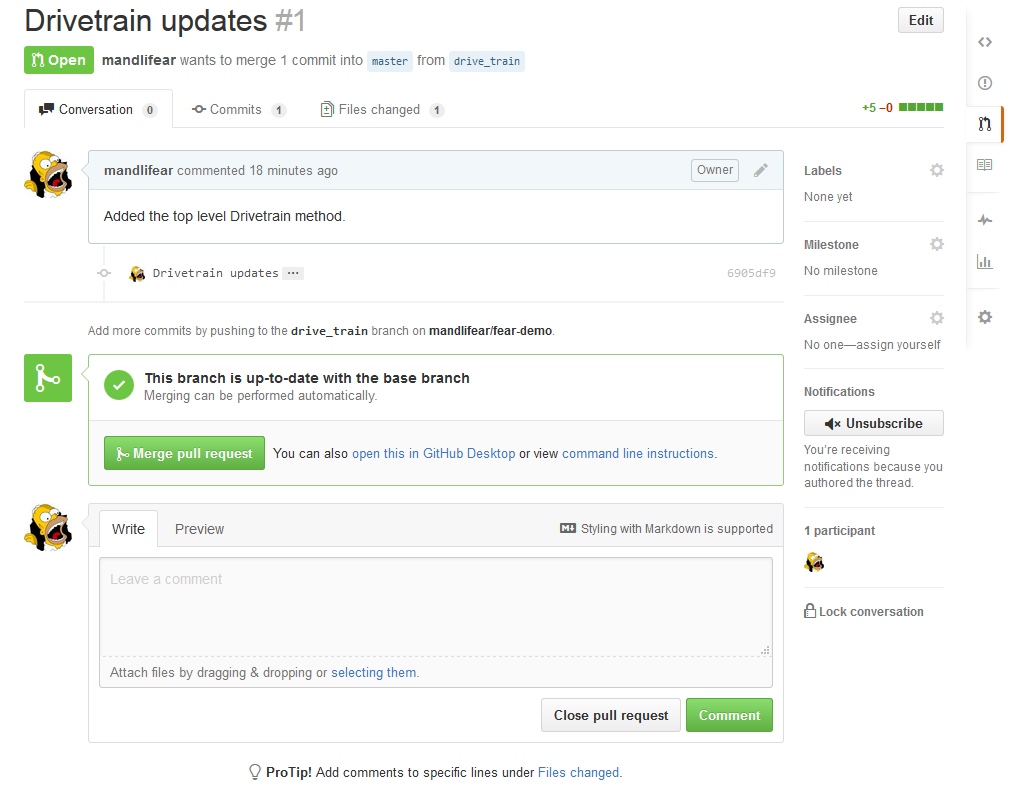
* Once the code on the topic branch has been tested and reviewed, create a pull request with the details of what changes are in your branch.



# Merging a pull request into master branch

Once a pull requests has been created, the department head merges the pull request into the master branch in GitHub.

* In GitHub, navigate to the ***Pull requests*** tab on the right-hand side and select ***Drivetrain updates***
* Note that all pull requests should indicate that ***This branch is up-to-date with the base branch*** and ***Merging can be performed automatically***. If not, the pull request must be closed, the topic branch resynchronized with master, retested, reviewed, and another pull request be created (covered in later steps).
* If the department head and mentor agree the pull request is ready to be merged, click the ***Merge pull request*** button followed by ***Confirm merge***.



* Congratulations, the first pull request has been merged!

# Completing and merging a second development topic

As stated previously, this example includes a second development topic, ***game\_mech***. This section details how a second branch is merged in. If the team is only working on a single topic, the steps below need not be performed.

Similar to the drivetrain, the game mech development team also commits and synchronizes code changes at least once per session. However, since the drivetrain changes were first to complete, and are already in the master branch, the ***game\_mech*** branch needs to pull in the new changes from ***master***, address any merge conflicts, test, code review and finally do a pull request.

* To pull in the changes from the master, click the ***Update from master*** button in GitHub Desktop. If there are no conflicts, GitHub Desktop will merge the code together. However, if the same sections of code were changed during game mech and drivetrain development, the conflicts will need to be resolved by hand. It is best practice to include the other team members who worked on the conflicting code, as well as a mentor, when resolving conflicts.

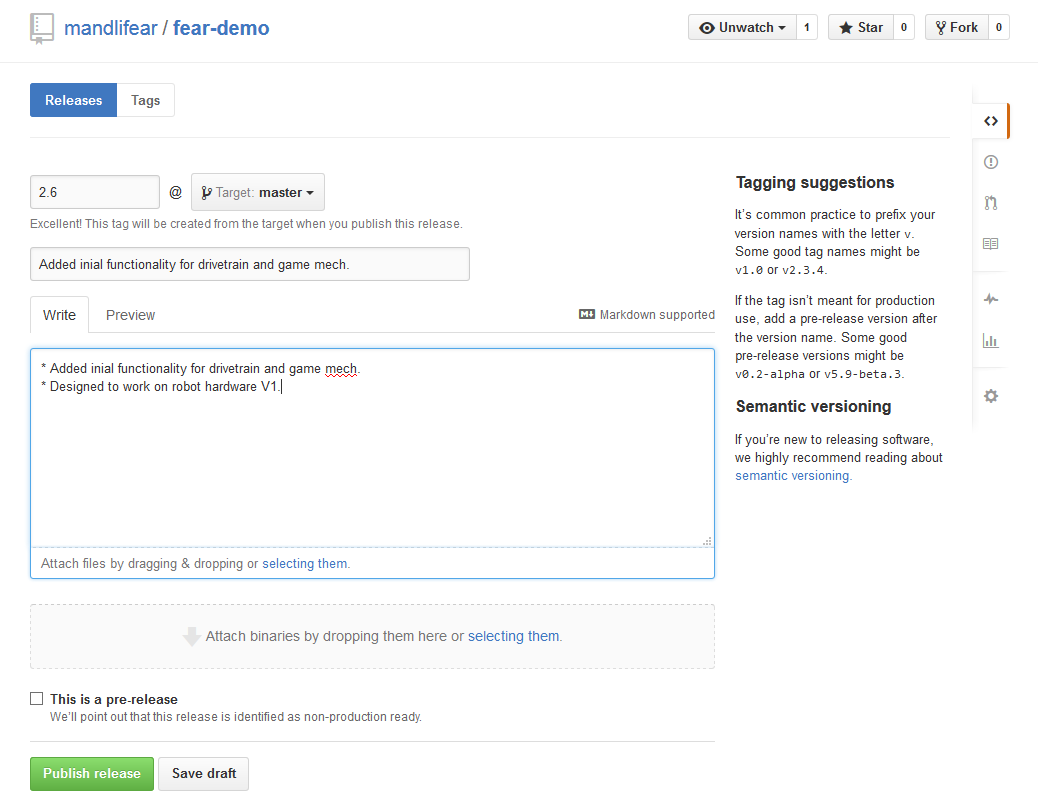
|  |
| --- |
| *For those who prefer the command line, from the game\_mech branch:*  git pull origin master  *And if there are merge conflicts:*  git mergetool  *And finally to commit and push the merged changes:*  git commit  git push origin HEAD |

* Make any required changes to the merged code to ensure everything is working.
* Test and perform a code review. If everything is correct, create a pull request, and merge the changes into the master.

# Creating a release

Once all pull requests have been merged into master, create a release. A release in GitHub labels your code so it is clear what functionality exists on a particular commit on the master branch. It also makes it easy to revert back to a previous released cut of code if need be.

* Navigate to the releases tab and click ***Create a new release***.
* In the ***tag version*** indicate the week and day, i.e. 2.7 for week 2, 7th day of the week (Saturday)
* Select the target as ***master***
* Create a meaningful title for the release, and fill in the details. Be sure to include what hardware the code was designed to run on.
* Click the ***Publish release*** button to finish.



* Complete and fill out the Merge / Release section of the Programming Checklist for the topic.
* Move the related Post-it notes to the ***Done*** section of the scrum board.