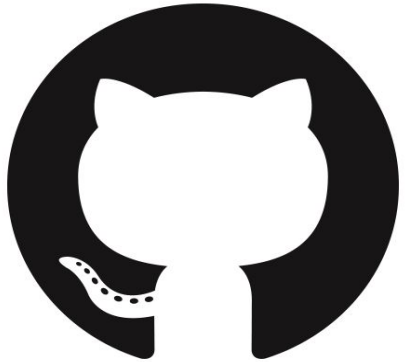
Git Version Control
GitHub website

Chantilly Robotics (Team 612)

Version control

- Organize code development
- Logs code changes for easier development
  - Especially with multiple coders
- Git is a version control scheme



Git development process

Branch → Edit → Commit → Merge

- Create a code **branch** off of the main codebase
- Edit code locally
- **Commit** updates to your **branch**
- Submit **pull requests** to **merge** your **branch** with the main codebase

Branches

- A snapshot of the code base that you can edit
- You can edit code in your **branch**
  - If your **branch** becomes outdated you can **pull** from master into your **branch**
- **Branch** naming conventions
  - `dev-` for features
  - `fix-` for bugfixes

Commits

- Use to save changes to your **branch** after you edit
- **Stage** changed files you want to **commit**
- After **staging**, **commit** files to your branch
- **Commits** have a text summary and description
  - Summaries should be short but descriptive
- NEVER **commit** changes to master. Ever.
  - Even if the breadfish tells you to

Merging and pull requests

- Submit **pull requests** when you want to **merge** your code with master
- **Pull requests** are reviewed before being **merged**
- Your **branch** must be up-to-date to be able to submit **pull requests**

Merge conflicts

- Sometimes your changes may be conflicting with other changes in master
  - Usually because master was updated since you created the branch
- **Merge conflict** messages will show your change and its conflict
  - Look for the word HEAD and a hexadecimal string
- The changes that have been made to master need to be kept; you need to change *your* code
- **Pulling** from master may overwrite your changes, so back them up before **pulling**
- How best to avoid **merge conflicts**
  - **Pull** from master before editing your **branch**
  - **Commit** changes in a timely manner