

Structuring the Development of Software for QPix (Part 2):

Version control, bureaucracy, releases, packages, etc.

Dave Elofson, Mike Kelsey, Dave Toback

Texas A&M University

May 13, 2022

Outline

- QUICK RECAP: Philosophy of Software Development Structure
 - Version control & Bureaucracy
 - Releases, packages & tags
- **Applying the philosophy to QPix software development**
 - **Structure of the QPix project of GitHub**
 - **Modifying the repository**
 - **Issues**
 - **Forking**
 - **Editing**
 - **Pull Requests**
 - **Cool-off Period: Vetting the soon-to-be new release**

Part 1:
Philosophy of Software Development
Structure

Recap

- Package Management and version control are necessary to maintaining order while modifying Packages
- Bureaucracy is how we make package management possible, and is vital in team settings
 - Developers modify the code, adding features and fixing bugs
 - Package Managers are responsible for maintaining the integrity of the package while incorporating the developers new modifications
 - Release Managers are responsible for working with the Package Managers to make sure all Packages work together in a new Release
- We are trying to enforce a single line of development, which is properly documented, so new releases replace old, and we don't go back to patch old releases.
- For a more detailed presentation, see the talk [Structuring the Development of Software for QPix \(Part 1\)](#) from 4/29/2022

Part 2: Putting this into practice with QPix

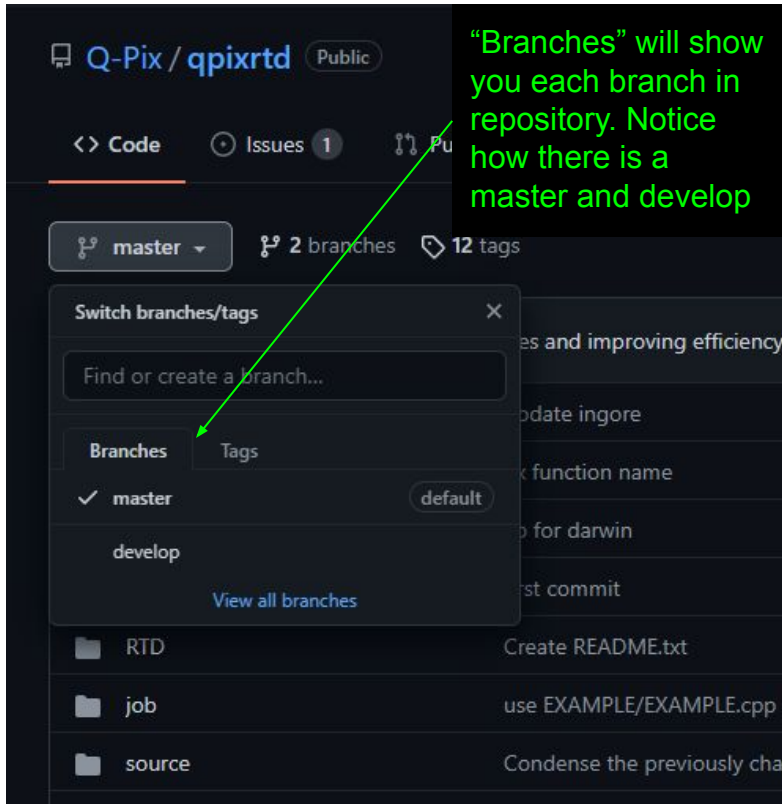
Structure of Git

The screenshot shows the GitHub profile for 'Q-Pix'. At the top, there is a search bar and navigation links for Pull requests, Issues, Marketplace, and Explore. The profile header includes the Q-Pix logo and name, with a 'Follow' button. Below the header is a navigation bar with tabs for Overview, Repositories (9), Projects, Packages, Teams, and People (9). The main content area is divided into 'Popular repositories' and 'People'. The 'Popular repositories' section displays a grid of repository cards: 'qpixg4' (C++, 3 stars, 5 forks), 'qpixrtd' (Jupyter Notebook, 3 stars, 1 fork), 'qpixar' (Python, 1 star, 1 fork), 'qpixprod' (Shell, 1 star), 'UTAH_GEANT4' (C++, 1 fork), and 'docs' (Q-Pix software documentation). A green arrow points from the top right text to the 'docs' repository card. Below the grid is a 'Repositories' section with a search bar and filters for Type, Language, and Sort, along with a 'New' button. The 'Repositories' section shows a list of repositories, including 'qpixg4' and 'qpixrtd', with their respective statistics and update times.

We keep all of our repositories in the Q-Pix project on github.
<https://github.com/Q-Pix>

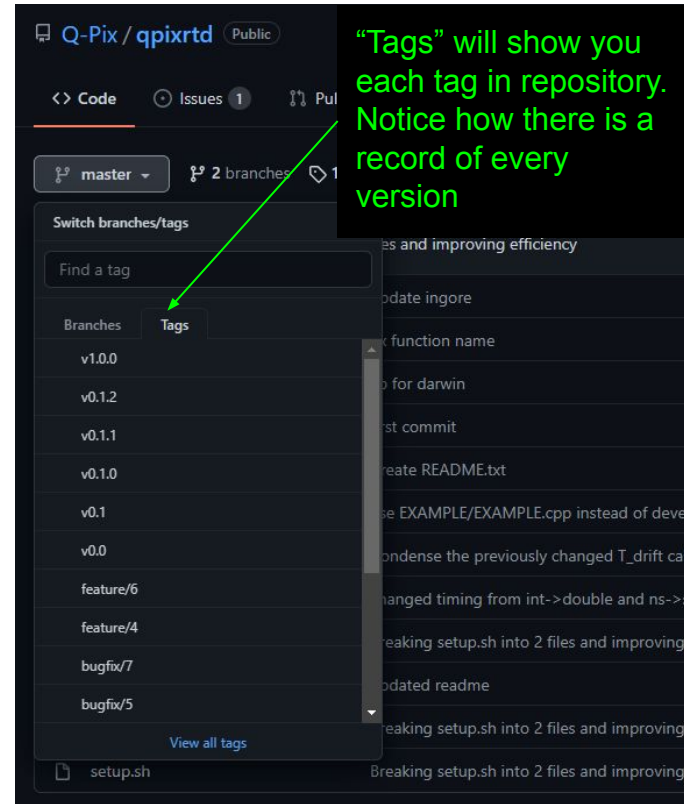
Trying to keep all documentation for software in the docs repository

Structure of a repository



“Branches” will show you each branch in repository. Notice how there is a master and develop

The screenshot shows the GitHub repository interface for 'Q-Pix / qpixrtd'. The 'Code' tab is active. Below the repository name, there are buttons for 'Code', 'Issues 1', and 'Pul'. A dropdown menu for 'Switch branches/tags' is open, showing 'Find or create a branch...' and a list of branches: 'master' (checked and marked 'default') and 'develop'. A green arrow points from the text to the 'Branches' tab in the dropdown.



“Tags” will show you each tag in repository. Notice how there is a record of every version

The screenshot shows the same GitHub repository interface. The 'Switch branches/tags' dropdown menu is open, and the 'Tags' tab is selected. A list of tags is displayed, including 'v1.0.0', 'v0.1.2', 'v0.1.1', 'v0.1.0', 'v0.1', 'v0.0', 'feature/6', 'feature/4', 'bugfix/7', and 'bugfix/5'. A green arrow points from the text to the 'Tags' tab in the dropdown.

Modifying the repository (A high level overview)

- **Forking the repository**
 - All modifications will be done on your personal forked repository and then transferred to the group repository
 - Once you (properly) fork the repository once, you won't need to do it again
- **Opening an issue**
 - Every time you want to modify the repository (either to add a cool feature or to fix a bug) you need to start by opening an issue explaining what and why
- **Modifying the code**
- **Submitting a pull request**
 - This is how you ask to have your modifications from your forked repository incorporated into the group repository and then into the new version

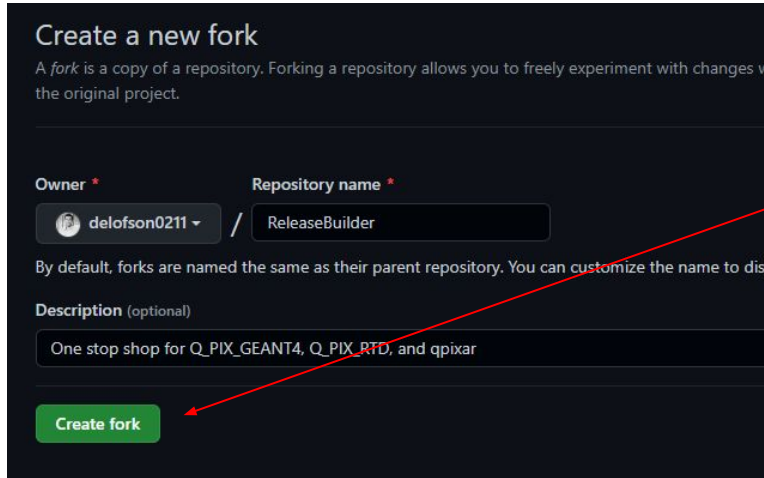
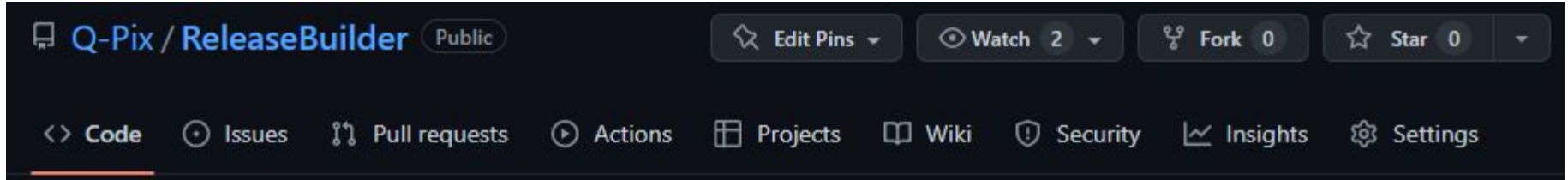
More detailed explanation of each step coming next...

Forking the repository

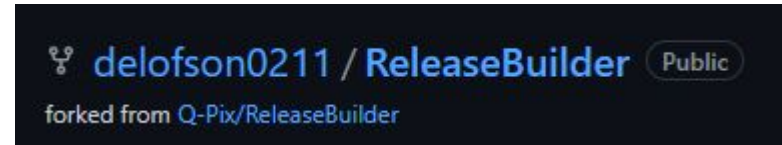
- Forking the repository is making a personal copy of the group repository for your own personal use
 - It provides a layer of protection between modifications to the code and the working versions of the Package
 - Anything you do will be isolated to your personal repository
 - In the event of a disaster, the group repository is unharmed and you can revert back to the working version of the group repository
 - You have the freedom to do what you want without affecting anyone else's version of the package
- Note:** Just because you can do what you want does not mean that you can use this to produce credible results. In order to produce credible results, you must be using a vetted version of the group repository

Creating a fork

Start creating a fork by clicking
“Fork”



It will automatically name the
fork after the parent repository.
All you need to do is click
“Create fork”



Your new forked repository name will
look like this

Properly setting up your fork

Clone your new fork locally by using

```
$ git clone git@github.com:delofson0211/ReleaseBuilder.git # creates local repository
```

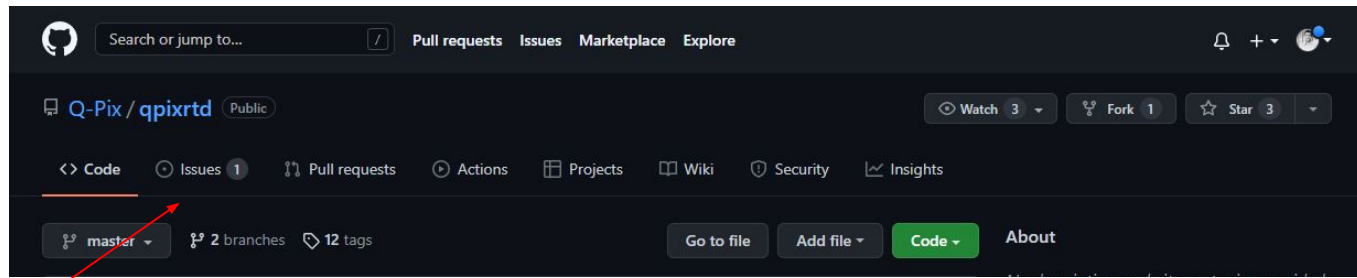
Track the original repository as a remote of the fork

```
$ git remote add -track develop upstream git@github.com:Q-Pix/ReleaseBuilder.git  
$ git fetch upstream
```

Note: fetching upstream will sync up your fork with the latest on the develop branch of the Q-Pix version of the repository

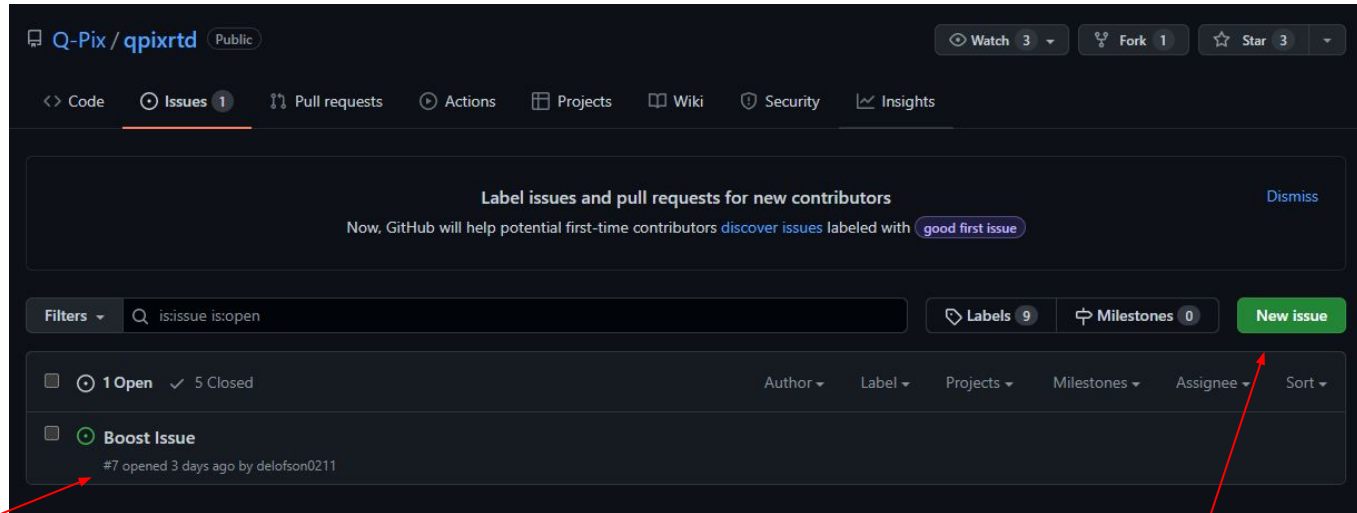
Setting up your fork properly ensures that you can stay up-to-date with the QPix repositories, and also allows you to create pull requests to modify the QPix repositories with the modifications you develop

Opening an issue



Whether it's good or bad, first open an issue and explain what is going on

You can see a list of all open issues here



Each issue is assigned a number. In this case, this is Issue #7

For your new code, you will want to open a New Issue by clicking on the button above

Opening an issue

Give the issue a good title that is clear, concise and informative

Save any commentary, elaboration or explanation for the comment

The screenshot shows the GitHub interface for creating a new issue in the repository 'Q-Pix / qpixrtd'. The 'Issues' tab is selected, showing 1 issue. The form includes a title field, a 'Write' tab, a rich text editor with a 'Leave a comment' placeholder, and a 'Submit new issue' button. The right sidebar shows settings for Assignees, Labels, Projects, Milestone, and Development. A red arrow points from the text 'Give the issue a good title that is clear, concise and informative' to the title field. Another red arrow points from 'Save any commentary, elaboration or explanation for the comment' to the comment area. A third red arrow points from 'Submit the new issue and go back to the issues page to see what issue number your issue was assigned' to the 'Submit new issue' button.

Submit the new issue and go back to the issues page to see what issue number your issue was assigned

Modifying the code (Making a new branch)

For all new code, it needs to go in a new branch. As mentioned, if it is to fix a problem, it should be titled as “bugfix/[issue number]” or if it is to add something new, it should be titled “feature/[issue number]”

This can all be done from the command line

```
$ git checkout develop           # makes "develop" your current branch
$ git pull --all                # makes sure you have latest changes
$ git checkout -b <branch>      # create new branch with your chosen name
$ git push --set-upstream origin <branch> # links local branch to remote repository
```

Make sure this is all happening in your forked repository!

Modifying the code (Adding your code)

You have now created a branch both locally and remotely that is up to date with the develop branch, and will contain all of your new code.

Bugfix and feature branches are allowed to not work. They are your code and no one will complain if there is a problem. Do not merge your code to develop unless it has been tested to work. If you need others to test your code, they can checkout your branch to test it.

Save fast, save frequently – If multiple people are contributing to the repository (which in our case, they are) code can change. It is good practice to not have your branch sitting for too long before checking back in with develop, or merging, otherwise it could get left behind and no longer be compatible with the development branch. If you have been working on a branch for a while, it may be a good idea to run `git fetch upstream` to make sure you are still up to date

Modifying the repositories (updating your branch)

As you make changes to your branch, you may want to update the remote version of your branch with your new modifications. This can be done with the following steps

1. Add - mark the files you want updated in the remote repository
 - Some files may just be modified by running your code and may not have any real significance to the package. Leave these out. Note: Adding does not actually record any changes
2. Commit
 - Records the changes that you made on your own local branch. This still does not save them.
3. Push
 - This will push the changes to the repository. By pushing to origin, you can update your changes to the remote repository. If you skip this step, you will not see your changes on github.com, nor will anyone else be able to see them.

```
$ git status # shows the status of the branch including which files to be
              committed and which to be left out
$ git add <list files to add for commit> # add any files that you purposely changed and need to be
              included in the commit
$ git commit # commits the give files
$ git push origin <branch name> # pushes the branch upstream to github
```


Modifying the repositories (Pull Requests)

Once you have tested your code, and it works and you are ready to have it checked and vetted to be added it to the develop branch of the Package. Remember once it is in develop, it will be included with the next release, so make sure it works and does not cause problems.

Pull Requests act like a safeguard against corruption in the Package. Basically these serve as one more step/check before code is merged onto the develop branch and can be included in the next tagged version.

Pull Requests

Q-Pix / ReleaseBuilder Public

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

testbranch had recent pushes less than a minute ago [Compare & pull request](#)

master 3 branches 2 tags [Go to file](#) [Add file](#) [Code](#)

delofson0211 Updated packages 4a0d3d4 13 days ago 6 commits

README.md Update README.md 20 days ago

Once you push your changes to the remote branch, you will see an option to **“Compare & pull request”**

Note: If the push didn't just happen and this option isn't available, you can do the same thing by clicking the “Pull requests” tab

Pull Requests

Branch to be merged (should be your forked version)

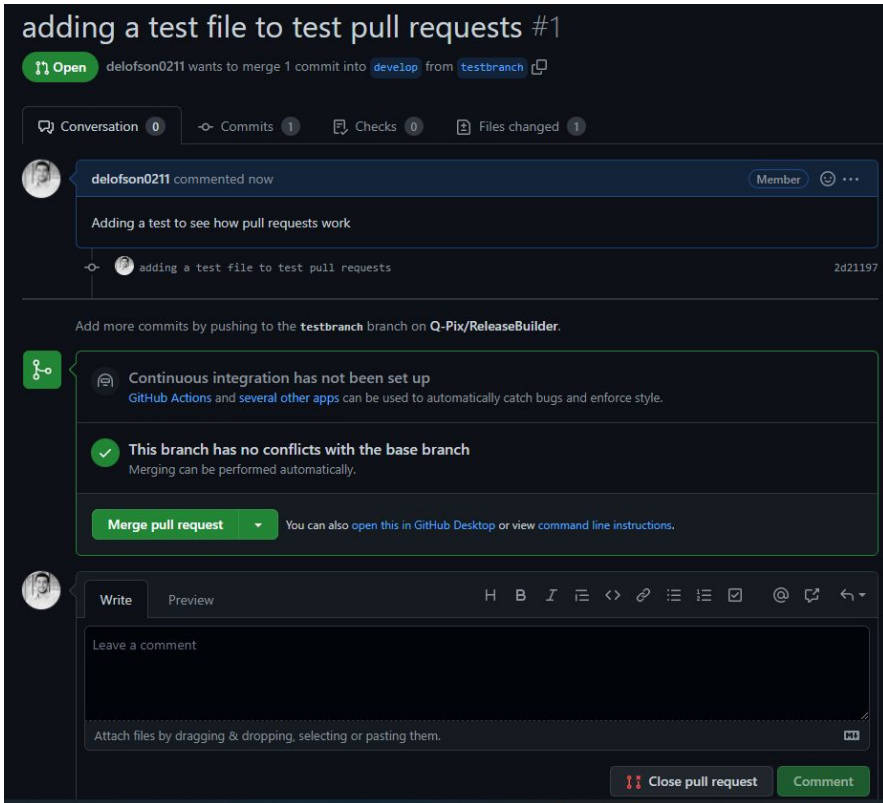
Branch to be merged onto (should be Q-Pix version of develop)

The screenshot shows the GitHub 'Open a pull request' form. At the top, it says 'Open a pull request' and 'Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).' Below this, there are two dropdown menus: 'base: develop' and 'compare: testbranch'. To the right of these is a green checkmark and the text 'Able to merge. These branches can be automatically merged.' Below the dropdowns is a text input field containing 'adding a test file to test pull requests'. Underneath the input field are 'Write' and 'Preview' tabs, followed by a rich text editor toolbar with icons for bold, italic, link, list, and other formatting options. Below the toolbar is a large text area for comments, with the placeholder text 'Leave a comment'. At the bottom of the form is a green button labeled 'Create pull request'.

Tells you if your merge is possible by checking if your bugfix/feature branch is compatible with develop

Leave a comment explaining what your modification is doing. Since it should already be associated with an open issue, also mention which issue it is related to.

Pull Requests (More for Package Manager)



The screenshot shows a GitHub pull request titled "adding a test file to test pull requests #1". The pull request is open, showing a comment from user "delofson0211" stating "Adding a test to see how pull requests work". Below the comment, there is a green box with a checkmark indicating "This branch has no conflicts with the base branch". At the bottom of the pull request, there is a green "Merge pull request" button and a "Comment" button.

Once you create the pull request, it will open a conversation.

The package manager will check that the merge is possible with no conflicts, vet the modification, and then approve if everything looks good.

- It is important to note that anyone can help with the vetting process and can contribute to the conversation
- The package manager should not merge the pull request if people are still having issues with the feature branch

What next? How to make a new release?

Once your pull request has been merged:

- Close the issue. Even if you are the one who opened it, comment on how you solved the problem and close.
- At this point, your work is done, and the ball is out of your court

-
- It is the responsibility of the package manager to make a new Master version of the package, and alert the Release manager (currently dave.elofson@tamu.edu) that it is ready to be released
 - See next page for more details
 - It is the responsibility of the Release manager to decide when to make a new release.
 - If the Release Manager finds that all the tagged Master versions of the Packages work together, the Release manager will make a new release, post it in the documentation and announce it to the group
 - (Yes it is weird because I'm the manager for all of them at the moment, but that WILL change)

Modifying the repository (Merging onto master)

- When it is time for a new release to come out, the Package Manager will create a pull request to merge develop on to master
- There will be a “cooling-off” period in which the develop branch should be checked by developers to ensure that it works properly before the merge is completed. The Package Manager will announce the creation of the pull request and the cooling off period
 - Developers can add to the conversation about the new version to comment on any bugs that they find
 - The Package Manager will explicitly state how long the cooling off period will last
- If nothing is found, or everything is fixed by the end of the cooling off period, the Package Manager will merge the pull request and tag the new version

Another resource

For a very coherent walkthrough, also look at this [link](#)

Most of the information in the walkthrough was used in the making of this talk, so if anything is unclear, it may be better in the original article