Structuring the Development of Software for QPix:

Version control, bureaucracy, releases, packages, etc.

Dave Elofson, Mike Kelsey, Dave Toback
Texas A&M University
April 29, 2022

*https://github.com/Q-Pix/docs/wiki

Outline

- Philosophy of Software Development Structure
  - Version control & Bureaucracy
  - Releases, packages & tags
- Applying the philosophy to QPix software development

Version Control and Package management

Version Control is the way we standardize dynamic software across many users

- Each version points to a specific point in the software's development
- Any user that uses the same version, should be able to recreate the result using the same input
  - Any user should be able to list the software version used, and peers can check the work by using the same software version
  - If someone gets different results with a different version, there is no validity to the discrepancy
    - Comparing different versions is like comparing apples and oranges
- Version control gives us a documented followable set of changes and ways to revert back to a standardized version if need be

Package Management

- Grouping all software into Packages has the advantage of standardization and putting checks and controls on software development
- Without package management, any random user could make any edits to the software that they saw fit and claim that the results confirm/disprove prior results, without the new software ever being vetted or approved.
- Ensures that all edits/additions to software are compatible with the rest of the package and the packages are compatible with the release

Version control without bureaucracy is chaos

- People usually choose one of two different ways to develop software:
  - Individual
    - One person is developing, testing and releasing the software
  - Team - (WE ARE HERE)
    - Multiple collaborators working on multiple changes to a package, all at once
- The current standard for coding professionals is to have tagged versions of Packages with the content, and tagged Releases which specify a specific set of versions of each Package which are known to work together
- The management standard is to have layers of bureaucrats. Each developer self-certifies that what they put on the develop branch works. The package manager double checks and certifies before merging develop with master, and the release manager does the same with all the packages together
  - This ensures edits/additions to a package are made in the proper manner with proper documentation, and the package that is released for use remains working as changes are made

Roles and Responsibilities

Release Manager

- Ensures that all the proposed tagged versions of all Packages work together and produce expected/desired results before creating a new release
- Liaises with Package Managers

Package Manager

- Ensures that the tagged version of the Package proposed for the Release works correctly, and works with the other Packages.
- Supervises Developers

Developers

- Work on developing and maintaining the code.
- They have lots of freedom to try things
- Need to get package into shape, and tag for approval by the Package Manager before it is incorporated into a Package, and eventually the Release

Structure of a Package

In each repository, you should find at least 2 branches:

- Master
  - Tagged version
  - Everything should work right out of the box
  - The package manager is responsible for merging develop onto master and creating a new release
- Develop
  - Anything that is on develop at the time of a merge and release will be included, therefore if it is on develop, it should be able to be built and run

You may also see additional branches:

- Bugfix
  - Used to edit the code to fix a problem
- Feature
  - Used to edit the code to add something new

You will also see tags in each repository:

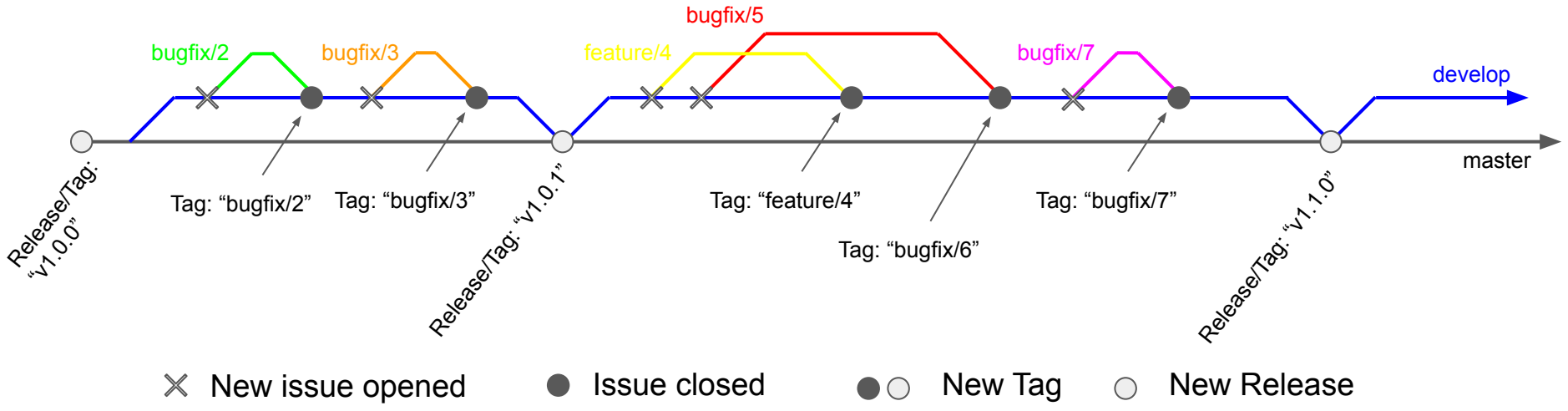
- Tags are like named save points/check points
- Used to document version numbers
- Used to document changes like a merge

Releases

A release is a special case of a tag

- For a single package, a release is a tag on the master branch made when everything has been fully vetted and everything is ready for use by non-developers (general users)
- For a multi-package release, it is a tag that marks out a new set of versions of the packages that have been tested and are known to work together
    - For QPix, we have set up a special ReleaseBuilder package to handle releases, which automatically checks out and builds the package tags designated for release. In this case, “The Release” is a tag on the master branch of the ReleaseBuilder package

The vision - A simple example

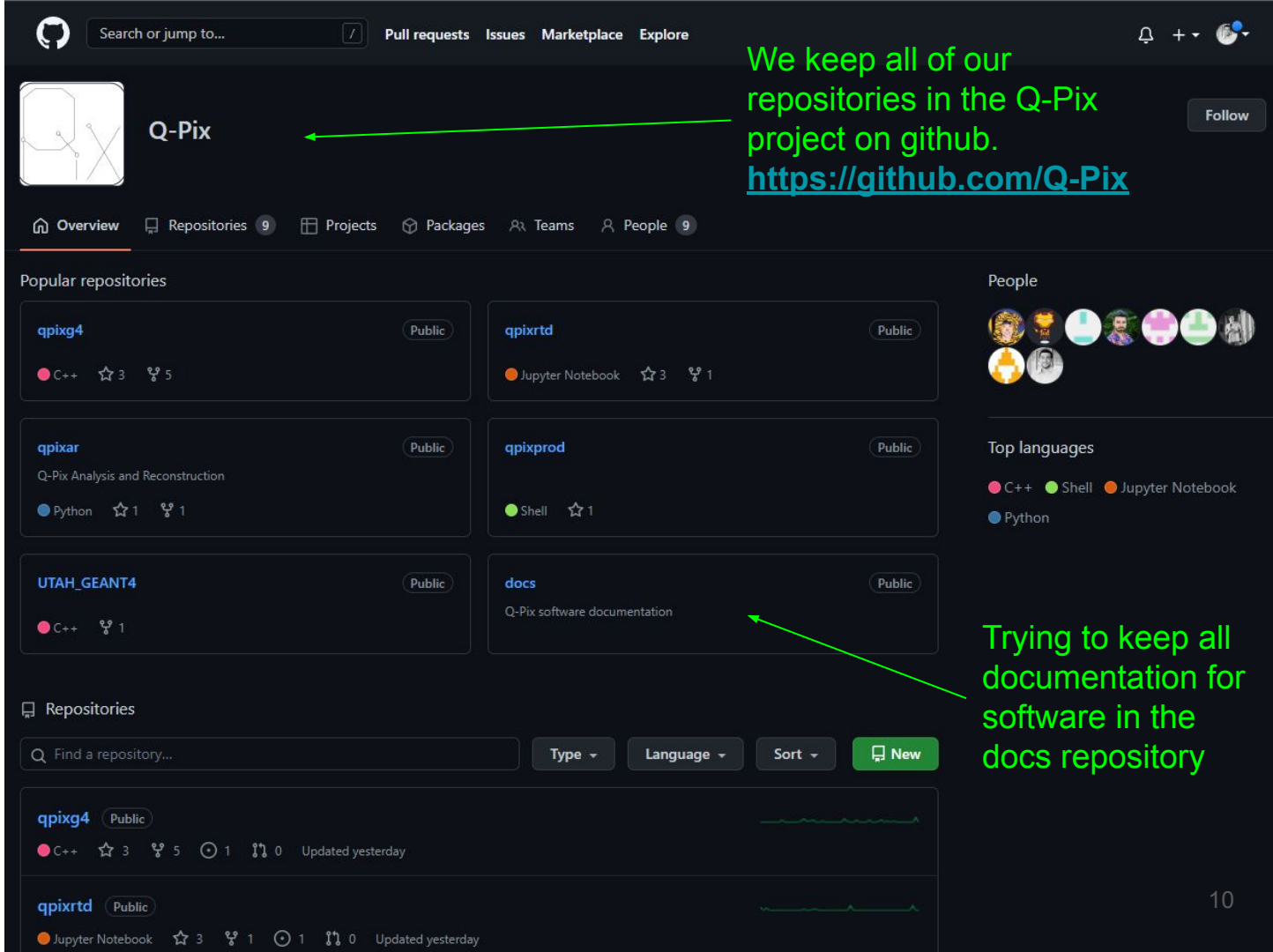


This is an example repository. The goal of every package is to have a path that is clean and clear, where each release version is obvious.

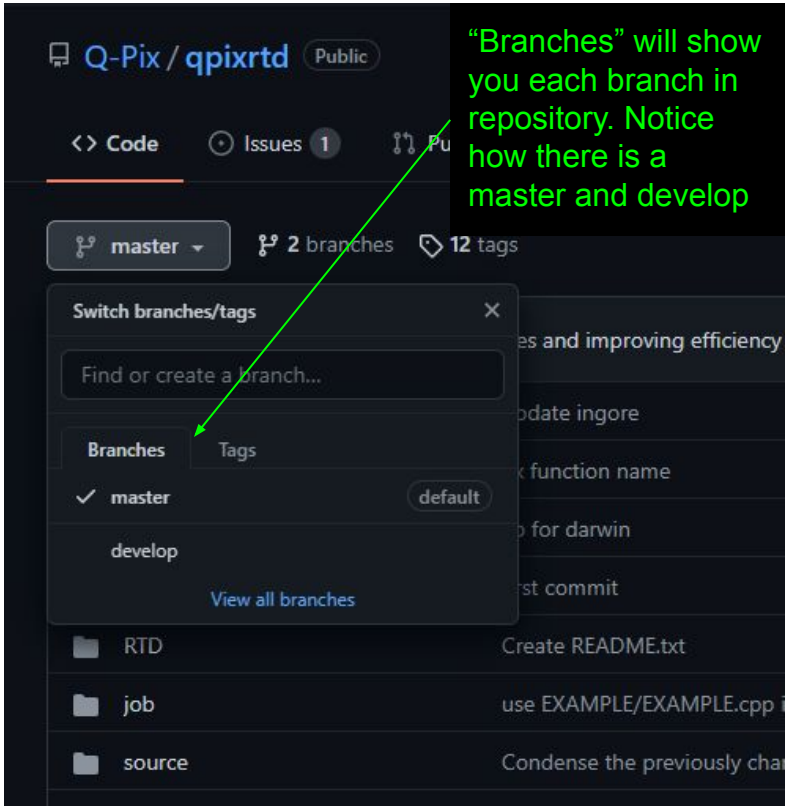
- Every edit/addition is well-documented with an issue and a tag
- The "story" of the repository can be understood from the documentation
- Enforces a single line of development (we don't go back to previous releases and patch it. The new release replaces the old one, and users should move forward)

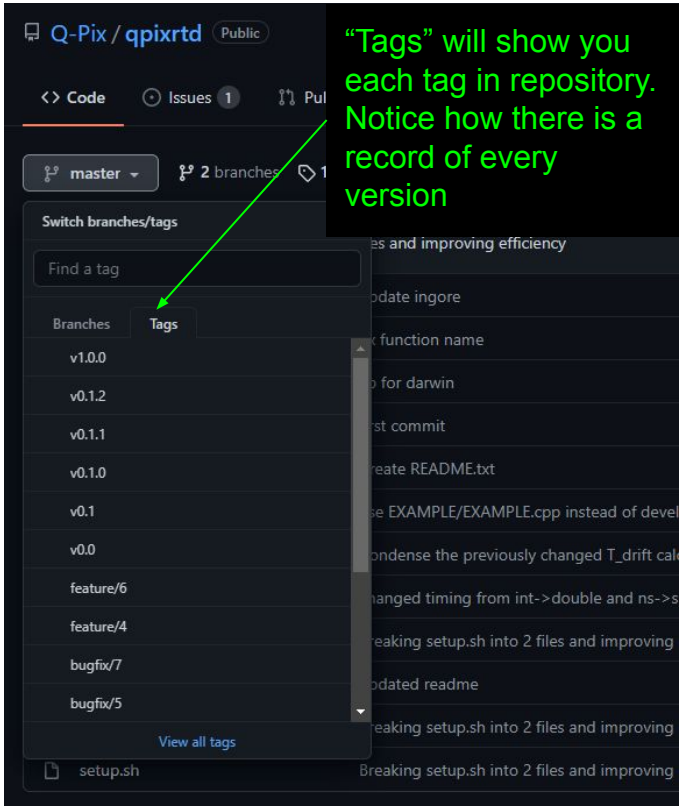
Putting this into practice with QPix

Structure of Git



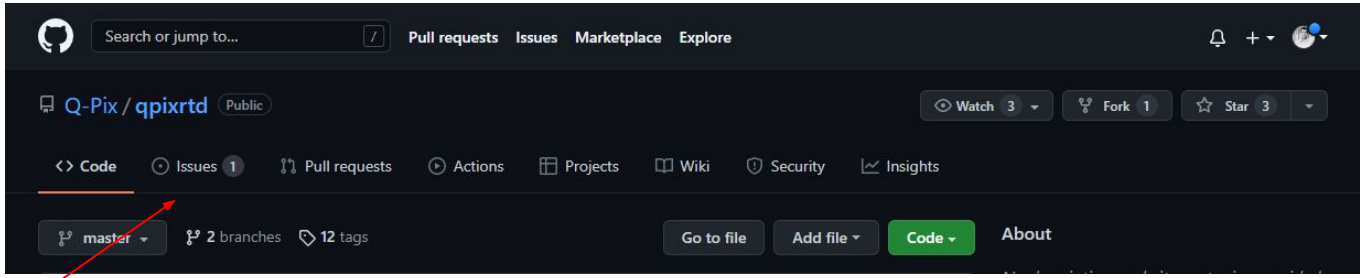
Structure of a repository

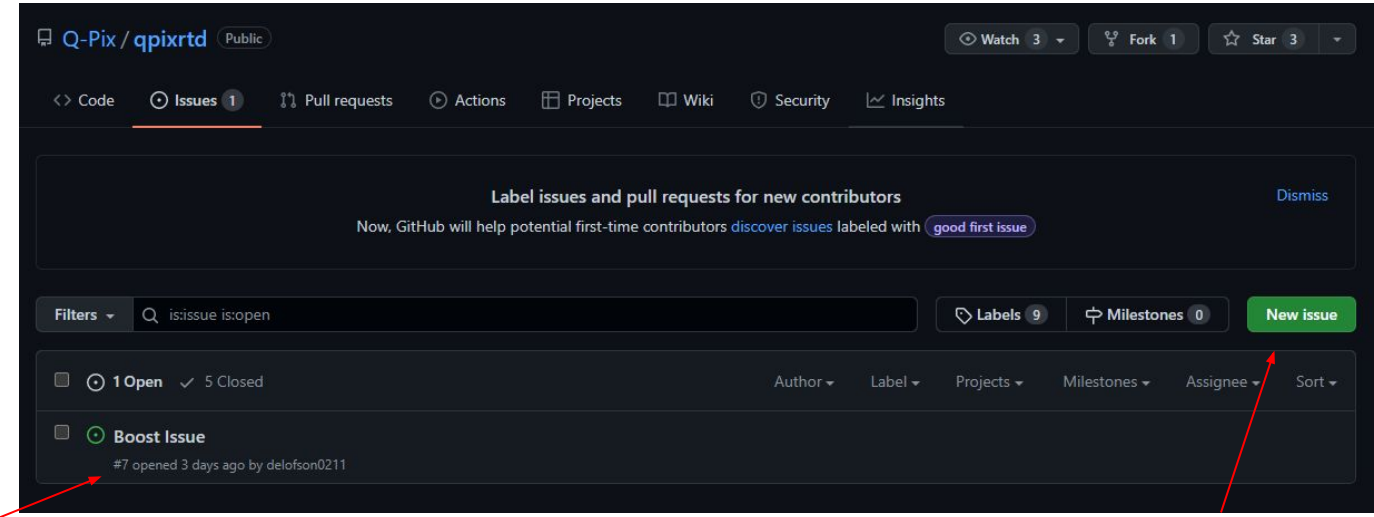




Modifying the repositories (Opening an issue)

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

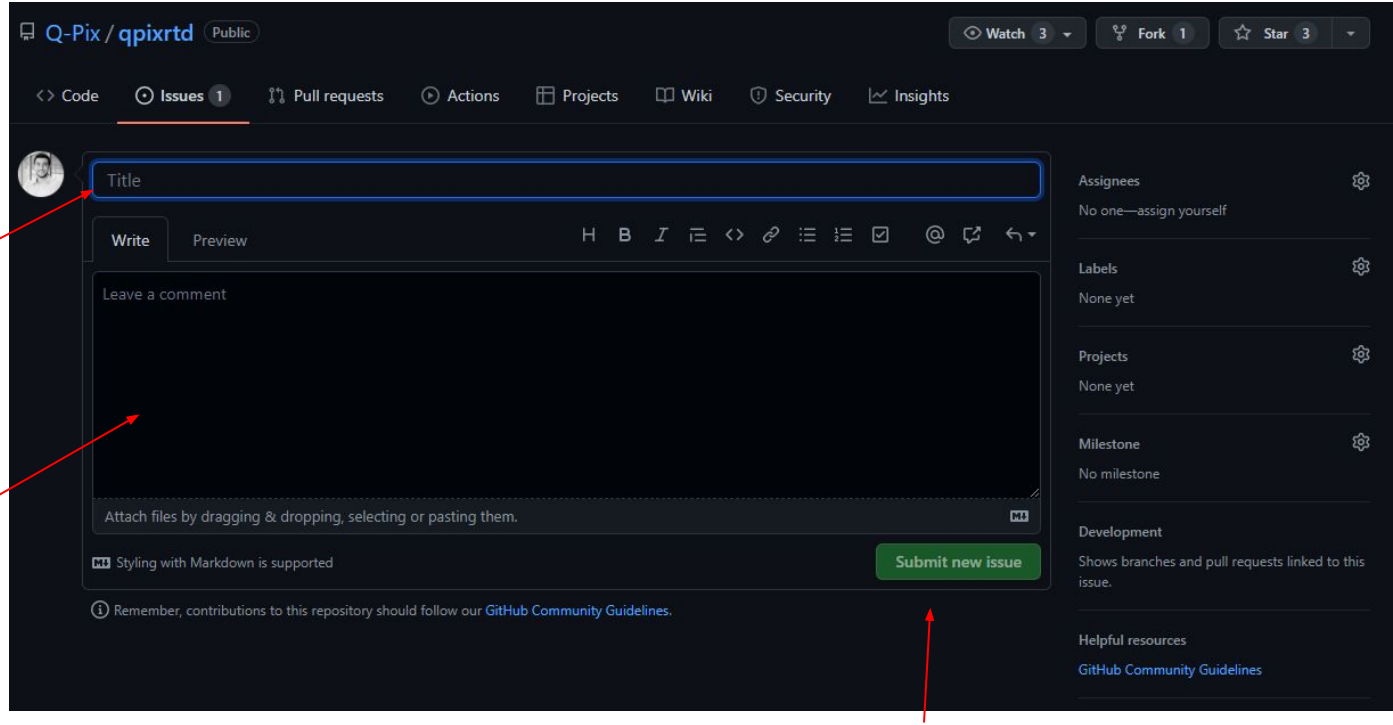
Each issue is assigned a number. In this case, this is Issue #7

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

Modifying the repositories (Opening an issue)

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

Save any commentary, elaboration or explanation for the comment



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

Modifying the repositories (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
```

Modifying the repositories (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.

Modifying the repositories (merging with develop)

Once you have tested your code, and it works and you are ready to merge it back into develop, follow these steps. Remember once it is in develop, it will be included with the next release, so make sure it works and does not cause problems.

```
$ 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
$ git checkout develop                                                # moves you to the "develop" branch
$ git pull –all                                                       # makes sure your develop branch is up to date
$ git merge -m "Describe the reason for your branch here" <branch name>   # merges your branch onto develop. In the message, answering
                                                                            why is more important than saying what you did
$ git push origin :<branch name>                                      # deletes the remote branch so you can't see it
$ git branch -d <branch name>                                         # deletes local branch
$ git tag -a -m "Describe the reason for your branch here" <branch name>  # creates a tag on the develop branch with the original
                                                                            branch name along with a description of the tag
$ git push; git push –tag                                             # pushes the develop branch and tag upstream to github
```

*By tagging develop with the name of your bugfix/feature branch, you make a way to track the merge back to the issue with an explanation of what was being done.

What next? How to make a new release?

- Close the issue. Even if you are the one who opened it, comment on how you solved the problem and close.
- Notify the package manager that you've finished
- 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
- 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)