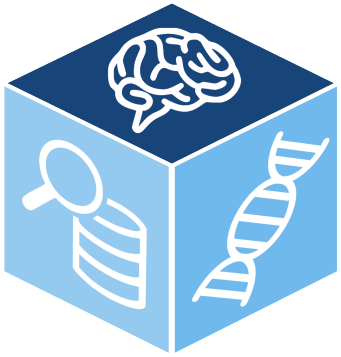


D3b

Center for Data-Driven Discovery in Biomedicine

Developer Handbook

July 16, 2018

Contents

1 Welcome

2 Creating Issues

Issues should be created If a repository has an *Issue Template*, make sure to follow it!

3 Finding Issues

3.1 Labels

Repositories across the D3b Github organization that have frequent user feedback and bug reports should use our standard set of labels to tag issues shown in Figure 1. Labels act as a means of filtering categories of issues and providing summaries of effort completed in release notes. Users that are not part of the organization may not add labels themselves, so please add labels as you see necessary to new issues. The help wanted label is a special label that is displayed in repository summaries on Github (See the *11 Issues need help* in Figure 4). Use it to mark issues that are self-contained and do not require in-depth knowledge of how the code base works. This tag is useful for guiding new members or external contributors looking to get involved to issues that will be easy for them to accomplish.

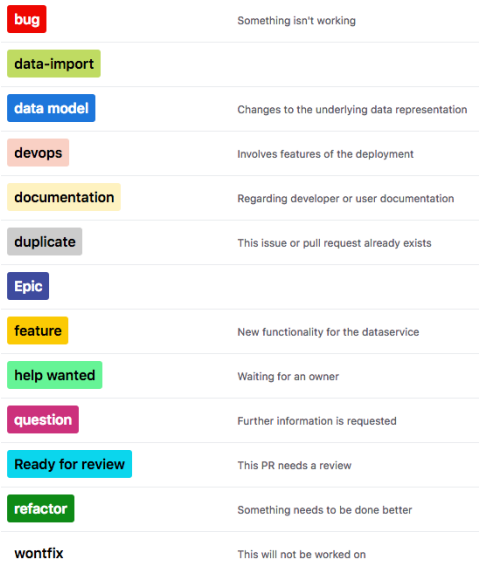
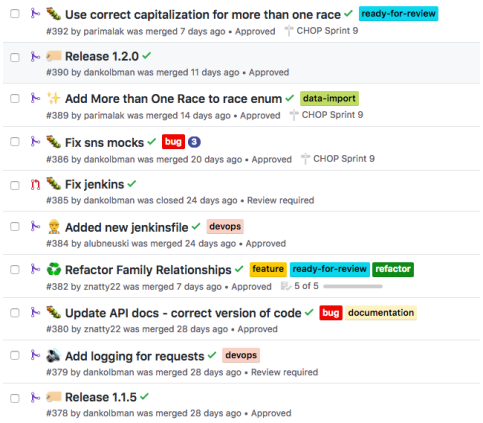
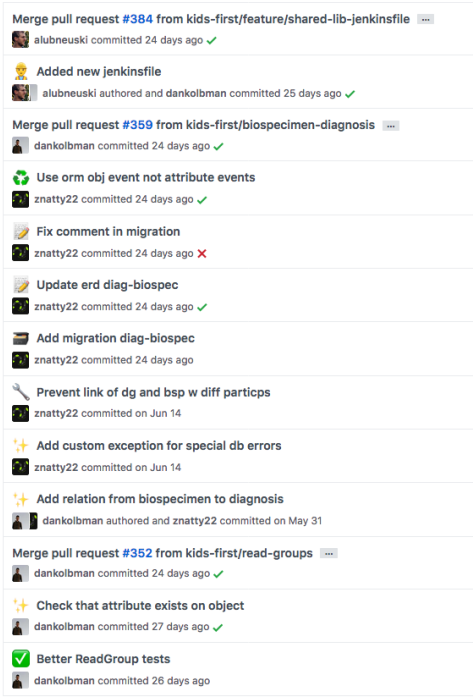


Figure 1: Standard Repository labels with descriptions and colors



(a) Closed pull requests in Github



(b) A slice of the dataservice commit history in Github.

Figure 2: Use of emojis in the dataservice.

4 Creating a Pull Request

4.1 Commit Messages

Commit messages are vital in being able to navigate a code base's history. There are many resources providing best practices for commit messages [1, 4].

We require commits to be prefixed with a single emoji that appropriately summarizes the the gist of the commit. This is helpful in allowing one to scan through the commit log visually inside of Github (Figure 2b) or be able to perform summary analysis inside of release notes or when analyzing git activity as a whole. When adding an emoji to the commit message, use the colon-ated version as opposed to the actual unicode symbol, eg: *:sparkles:*. For inspiration of what emoji may be most appropriate, see the gitmoji guide [2].

4.2 Pull Request Titles

Pull request titles should begin with an emoji, similar to commit messages. This provides similar benefits to commit message emojis where past pull requests may quickly be scanned and summarized (Figure 2a).

4.3 Keeping the Commit Log Tidy

In addition to best practices outlined in Section 4.1, care should be taken to avoid introducing unhelpful commits such as merges of master into a feature branch. Often, a feature branch will become stale during development as other features are merged into master. Instead of using the *Update* button in Github to merge the latest master into the feature branch, one should opt to rebase the branch on the latest master. This will avoid adding many *Merged branch master* commits into a feature branches commit history.

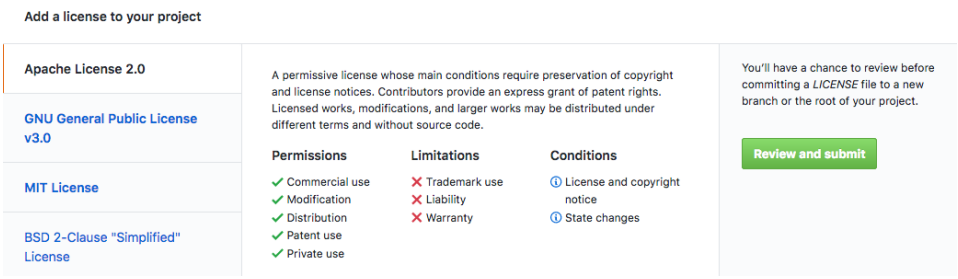


Figure 3: The Apache 2.0 license is available by default from the Github web interface.

4.4 Labeling Pull Requests

4.5 Requesting Reviews

5 Review Process

5.1 Code Review

5.2 Testing

5.3 Status Checks

6 Software Release

7 New Repositories

7.1 Naming

7.2 Description

7.3 Tags

7.4 Protected Branches

7.5 README

7.6 License

Kids First uses the Apache 2.0 [3] license exclusively for all public repositories. The file should be placed in the root of the directory with the name *LICENSE*. The Apache 2.0 license may be easily added from the root repository view on Github using *add new file* and naming it LICENSE. This will present a *Choose license template* button to the right of the file name where the Apache 2.0 License may be chosen to automatically populate the file (See Fig 3).

7.7 Config Repository

In addition to creating the main code repository, each code repository that is to be deployed must have a second, *private* repository that contains the infrastructure and CI/CD strategy for that code base. The repository for the config *must match the name of the code repository suffixed with -config*. This is a rule of thumb that is followed throughout so that the configuration may be discovered automatically by other scripts. In addition to the naming standard, the repository should be *private*. It is best practice to keep the infrastructure obscured and prevent external contributions as the config contains code that is often run at elevated privileges in our environments.

8 Environments

Often, entire software stacks are deployed concurrently in many different *environments*. This provides us with a handful of benefits:

- Assurance that there is always a stable deployment for end-users
- Provides a safe sandbox to develop new features in isolation
- Insures the boilerplate to re-deploy a new environment exists

We divide our environments into three primary stages that are tightly couple to our Continuous Integration

8.1 dev

The development environment (dev) is intended as sandbox for new features and code bases. It is the most unstable environment with no guarantees on data availability. When new code is pushed to a feature branch, it automatically triggers its CI pipeline to test, build, and, deploy to the development environment. The branch must pass through testing and deployment successfully to before it is allowed to be merged to insure that it will successfully deploy to the next stage.

8.2 qa

The QA environment provides a mostly-stable environment where new features may be used, evaluated, and tested before being released. This is is where integration test-suites may be run to ensure that all services cooperate nicely. The data in QA should also be close to production quality to fully flex all features. New features enter QA after merging a feature branch into the master branch of a code base.

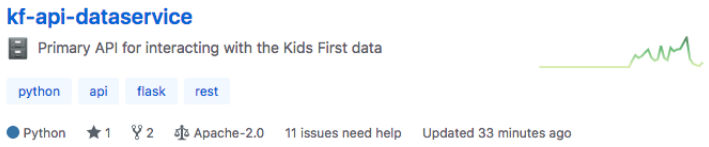


Figure 4: An exemplary repository summary including: proper naming scheme, emoji followed by short description, labels, the *Apache 2.0* license, and issues tagged with *help wanted*

8.3 prd

The production environment (prd) is the end-user environment. This is critically stable as it must be exposed to the public. To introduce features into prd, a repository must go through the release process. This includes creating a pull request for the version change, signoff on it from any potential stakeholders, merging, and finally acquiring final approval from an administrator to deploy the infrastructure and code into the environment.

References

[1] Chris Beams. Git commits. `https://chris.beams.io/posts/git-commit/`.

[2] Carlos Cuesta. gitmoji. `https://gitmoji.carloscuesta.me`.

[3] The Apache Software Foundation. Apache license version 2.0. `https://www.apache.org/licenses/LICENSE-2.0`, 2004.

[4] Stack Overflow. Standard to follow when writing git commit messages. `https://stackoverflow.com/questions/15324900/standard-to-follow-when-writing-git-commit-messages`.