Open Source Tips

By Eddie Jaoude

Table of Contents

Abstract

This book contains tips for DOs & DONTs of Open Source software.

Preface

Open Source is dominates the software industry. With so many projects out there from well known organisations like Facebook, Twitter, NetFlix etc to passionate individuals around the world. With so much choice, how do you share your work & how do you decide who's work to use?

As with technology that keep on changing, this book will need continual updating. Suggestions & pull requests are welcome. Currently on version v0.1.42.master.

Source Repository is on GitHub https://github.com/eddiejaoude/book-open-source-tips

Introduction

With so many popular Open Source projects & many people contributing to Open Source, it would be beneficial to capture the pros & cons. Open Source not only benefits the community but also the authors as they get feedback on their project, from a higher overview or deeper technical level, more exposure & marketing, testing & bug fixing etc.

TIP This book will be continually updated, please check the version 0.1.42.master. Any question please contact the author Eddie Jaoude on https://twitter.com/eddiejaoude.

DOs

Recommended…

Readme file

Documentation is usually left to last. Start every project with at least a `README.md` with some basic information, for example a QuickStart guide, if you change features or functionality, try at least update this with your commits.

Code blocks

Use code highlighting within code blocks in documentation to make it easier to read.

Contribution file

One of the main benefits of an Open Source project & its community is Contributions. Lower the barrier to entry with a `CONTRIBUTORS.md` file in the root of your Open Source project. Read more about GitHub Contribution file

Code of Conduct

Your community needs to feel safe, diverse & included. Make sure you have a Code of Conduct for your project & community. Read more about Open Code of Conducts

GitHub template files

Issue & Pull Request templates really help keeping the project consistent & reminds people not to leave out certain useful information.

Licensing

It is not required to select a license, however it is very useful to do so. GitHub have created an informational website to help you choose a license

Commits

Commits should be small and atomic, be it a single change or small feature. This will make your commit messages easier to write and and changes will be grouped logically. There are many benefits to this:

- Looking back through the history will be clear & easy to understand
    - if you want to find something
    - undo / remove some work

- Automate the changelog generation as part of your build for tag & package etc

Little and often

Steady projects not only look more stable, but are generally more successful & are better for your health. Trying doing a little every week.

Plan ahead

Create or check tasks today ready for tomorrow. There are two benefits of this, allowing you tomorrow to immediately hit the ground running and to digest the tasks overnight as you might make some final tweaks.

Issues / Tasks

Keep these small. Tackling a large piece of work is always daunting and more difficult to find the time. The smaller the tasks, the more likely it is to be done and the lower risk it will be. But remember the task still needs to provide value to the project.

Respond to Issues and Pull Requests

Always respond to Issues and Pull Requests in a timely fashion, ideally within 48 hours (even if it is with a comment acknowledging you have at least read the issue).

Pull Requests

If you spent the time doing the work, make sure you add a description to your your Pull Request to make it easier for the reviewer to digest your work. Raise an Issue first so a plan of action can be discussed before you begin the work and to remind yourself of the goals set out in that task.

Reviews

Even if it is only you on the project, try to raise Pull Requests & get a friend to review it. This approach is invaluable as a second pair of eyes often picks up oversights.

Automated tests & Continuous Integration

Automate everything! This helps lower the barrier to entry & increase repeatability.

- Automated tests have many benefits & give confidence in the state & quality of the application:
    - Unit tests are great for design & architecture of functionality
    - Integration tests are great for the touch points
    - End-to-end tests are great for full appliaction testing & simulate the user
- Run the automated tests on Continuous Integration (CI) (for example TravisCI)

- When automated tests are successful, deploy the application aka Continuous Delivery (CD)

Note: This includes database schema migrations, asset building and anything that is required by the end product

Get a Prototype to your users quickly

Get feedback as soon as possible. Quick and dirty prototypes in front of some of your users will give you instant feedback and direction. Remember to make it clear it is a prototype.

Work at your optimal time

People work better at different times of the day and night so find your most efficient and optimal time. Even if that is 11pm at night or 4am, try utilise your most productive time.

Never enough time

We all have the same 24 hours in a day available to us. It’s what you do with it that counts. Try to find a small amount of time per day, even 10 minutes when you are on the toilet - yes you heard me right "on the toilet". Multi tasking in that situation is possible, but trying to work while watching TV is very unproductive.

Make it open source (public) from day 1

I do NOT understand people who say "I will make it public when it is finished". First of all, it will never be finished, secondly you then do not gain all the benefits from Open Source.

Leave the codebase better than you found it

Many code repositories (mostly Closed Source ones) go from bad to worse. Open Source projects tend to do the complete opposite due to it being in the public eye. Even the smallest of improvements add up and really help - no improvement is too small.

DONTs

NOT recommended…

Big bang projects

Dont try to complete the project in a manic weekend, then ignore it for the next year. This is not good for your health nor does it look good for your project from the community’s view.

God commits

God commits or big bang commits are useless & confusing. These are terrible for you & the community for so many reasons.

God Pull Requests

Similar to "god commits", god pull requests are a bad idea. Reviewing god or big bang pull requests are not only annoying & painful, but leave room for skim reviewing & therefore mistakes. A pull request should be a single feature. No one ever complained that a Pull Request was too small.