Open Source Tips

Table of Contents

Abstract

This book contains some common DOs & DON'Ts for Open Source software.

Preface

The Open Source community is thriving. Each day the number of Open Source projects grows, as does the army of contributors that maintain them. While this is exciting for the industry, it can be daunting as a developer new to the community. This book aims to provide some tips for newcomers to help them avoid the pitfalls of Open Source development and learn from the community's collective wisdom.

As the ancient proverb goes, "*Time and tide [and technology] wait for no man*". And to the best of our ability, neither will this book. Remember to check the version number for updates! We're currently on v0.1.14.

We would love your help in keeping this book updated. Your comments, suggestions and pull requests are most welcome. You can find the repository on GitHub: https://github.com/eddiejaoude/book-open-source-tips.

If you have any questions, please contact the author, Eddie Jaoude on https://twitter.com/eddiejaoude.

Introduction

Open Source is dominating the software industry. Its champions include well known organisations like Facebook, Twitter, Netflix, LinkedIn and Google (Android/Chrome), but more significantly, an army of passionate individual developers around the world. Their efforts have impacted almost every part of computer science, culminating in millions of open source projects, with billions of lines of code!

While this abundant ecosystem has been of huge benefit to the whole industry, it can also make it difficult for newcomers to know where to start. If you're a newcomer, you might be faced with questions such as *"How can I contribute to the Open Source community?"* Or, *"How do I choose between so many competing projects?"*. The following DOs and DON'Ts aim to address some of those basic questions, plus some pointers for aspiring Open Source developers.

Let's dive right in.

TIP	Projects not made public at the beginning are at higher risk of having private credentials committed in the history. Therefore it is highly recommended to make projects public from the start, stating they are not finished is not an excuse as they will never be finished. If public from day one, then the right mindset is used and thus reduces the risk.

DOs

Readme file

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

Example README.md

```
# Name of Project

Include any project badges (e.g. CI) here at the top.

Description of project & its goals.

Also include what it does not do.


## Screenshots

Nothing says more than screenshots.


## Dependencies

Any dependencies required by the project.


## Installation

How to install on Mac...

How to install on Linux...

How to install on Windows...


## Usage

How to use the project


## Contribution SetUp

If people would like to contribute, what steps should they take.

Overview here and a link to the `CONTRIBUTION.md` file with more details
```

```
## Code of Conduct

Overview of Code of Conduct, with link to `CODE_OF_CONDUCT.MD`


## Change log / Release history

Major version & breaking changes


## Meta data

Any other useful information.
```

Code block

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

Syntax highlighting

You can add an optional language identifier to enable syntax highlighting in your fenced code block.

For example, to syntax highlight Ruby code:

```
```ruby
require 'redcarpet'
markdown = Redcarpet.new("Hello World!")
puts markdown.to_html
```
```

```ruby
require 'redcarpet'
markdown = Redcarpet.new("Hello World!")
puts markdown.to_html
```

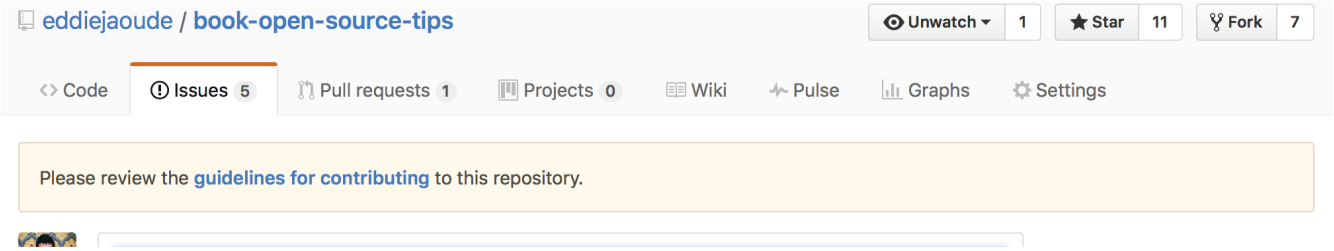
GitHub syntax highlighting

Contribution file

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

> Often times open source projects place a CONTRIBUTING file in the root directory. It explains how a participant should do things like format code, test fixes, and submit patches. Here is a fine example from puppet and

> another one from factory_girl_rails. From a maintainer's point of view, the document succinctly communicates how best to collaborate. And for a contributor, one quick check of this file verifies their submission follows the maintainer's guidelines.
>
> — GitHub, GitHub Contributing Guidelines



GitHub Contributing Guidelines

Explore open source projects

Remember to explore other open source projects out there to see how other projects have been successfully managed, and what their outcomes looked like.

Here are 10 examples of open source projects to start you off:

- 24 Pull Requests
- dwyl - do what you love
- Founders & Coders
- Hack Brexit
- We Rock Tech
- Women Who Hack For Non Profits
- Elixir Koans
- Codebar
- Rails Girls Summer of Code
- Vote America

TIP The more open source projects you have explored, the more you'll see what practices have worked and what have not. This will equip you with more knowledge on how you'd like to run your project in practice, based on existing projects you've come across.

Code of Conduct

Your community needs to feel safe, diverse & inclusive. Make sure you have a Code of Conduct for your project & community. Read more about Contributor Covenant - A Code of Conduct for Open Source Projects

> Open Source has always been a foundation of the Internet, and with the advent of social open source networks this is more true than ever. But free, libre, and open source projects suffer from a startling lack of diversity, with dramatically low participation by women, people of color, and other marginalized populations.
>
> — Contributor Covenant, Brief overview of the problem

> An easy way to begin addressing this problem is to be overt in our openness, welcoming all people to contribute, and pledging in return to value them as human beings and to foster an atmosphere of kindness, cooperation, and understanding.
>
> — Contributor Covenant, Brief overview of the solution

GitHub template files

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

> To add an Issue template to a repository create a file called ISSUE_TEMPLATE in the root directory. A file extension is optional, but Markdown files (.md) are supported. Markdown support makes it easy to add things like headings, links, @-mentions, and task lists to your templates.
>
> — GitHub, GitHub Issue Template

> Pull Request templates follows the same pattern: add a file called PULL_REQUEST_TEMPLATE to the root directory of your repository.
>
> — GitHub, GitHub Pull Request Template

> If you're worried about the added clutter in the root directory of your project, we also added support for a .github/ folder. You can put CONTRIBUTING.md, ISSUE_TEMPLATE.md, and PULL_REQUEST_TEMPLATE.md files in .github/ and everything will work as expected.
>
> — GitHub, GitHub Hidden Directory for Templates

Full details from GitHub for helping people contribute to your project

Licensing

It is not required to select a license, however by doing so, you are selecting "No License" which will default you to the T&Cs of GitHub. GitHub have created a great informational website to help you choose a license

Examples below from "Choose an open source license":

> The MIT License is a permissive license that is short and to the point. It lets people do anything they want with your code as long as they provide attribution back to you and don't hold you liable.
>
> — Choose a License, MIT License

NOTE jQuery, .NET Core, and Rails use the MIT License.

> The Apache License 2.0 is a permissive license similar to the MIT License, but also provides an express grant of patent rights from contributors to users.
>
> — Choose a License, Apache License 2.0

NOTE Android, Apache, and Swift use the Apache License 2.0.

> The GNU GPLv3 is a copyleft license that requires anyone who distributes your code or a derivative work to make the source available under the same terms, and also provides an express grant of patent rights from contributors to users.
>
> — Choose a License, GNU GPLv3

NOTE Bash, GIMP, and Privacy Badger use the GNU GPLv3.

> When you make a creative work (which includes code), the work is under exclusive copyright by default. Unless you include a license that specifies otherwise, nobody else can use, copy, distribute, or modify your work without being at risk of take-downs, shake-downs, or litigation. Once the work has other contributors (each a copyright holder), "nobody" starts including you.
>
> — Choose a License, No License

CAUTION As a consumer, if you find software that doesn't have a license, that generally means you have no permission from the creators of the software to use, modify,

> or share the software. Although a code host such as GitHub may allow you to view and fork the code, this does not imply that you are permitted to use, modify, or share the software for any purpose. Your options are: Ask the maintainers nicely to add a license, Don't use the software, Negotiate a private license with a lawyer.

Git commit

Git 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

Commit message

- Limit the subject line to `50` characters
- Do not end the subject line with a period `.`.
- If more than `50` characters use the body (description)
- Wrap the body at `72` characters.
- Use the body to explain `why` not `how`, this can been seen in the code

More information Git Commit

TIP Before doing the commit, check the `git diff` to make sure you are not committing anything you do not want to, or should not be.

Little and often

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

Working frequently on your project gives your community confidence that you believe in your project & are in it for the long term not just that moment.

Plan ahead

Create tasks today ready in preparation 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.

TIP	Plan a little ahead, not too much as things will change

GitHub Issue / Task

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.

Include diagrams, screenshots, sub tasks & anything visual to help describe the issue & changes.

TIP	Don't forget you can use a sub task checklist on GitHub Issues Checklist. Sub task list are prepended with `[ ]` for incomplete & `[x]` for complete items.

Related

- Labels tips

Always respond - Issue and Pull Request

Always respond to Issue and Pull Request in a timely fashion, ideally within 24 hours (even if it is with a comment acknowledging you have at least read the issue & will respond fully at a later date). This manages expectations for when the contributor can expect a full response.

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.

Pull Requests should be linked to the original Issue it is trying to solve. This can be done with using `#` followed by the **Issue No**, e.g. `#123`. The Issue description will contain the information before, therefore the Pull Request description should contain the information after the changes. Include visual material too, for example diagrams & screenshots.

Remember, keep it small. For example if your changes contains a feature or bug fix & code styling changes, these should be in separate Pull Requests. Every project would rather have 10 Pull Requests, than 1 or 2 massive Pull Requests.

TIP	Pull Request should be a single feature or change.

Multiple commits in a Pull Request highlight the creation steps of the Pull Request. Do not try to do everything in a single commit.

Comments do matter, if in doubt, put it in, they can more easily be removed than added. Don't describe how, that is obviously in the code, describe why.

Review

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.

TIP	Even simple text changes might make sense to you but not to someone else. Review everything!

Automation - Tests, Continuous Integration (CI), Continuous Deployment (CD)

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 application testing & simulate the user
- It should be very easy to setup up a local environment & run all these tests
- Run the automated tests on Continuous Integration (CI) after someone has pushed their code changes to a feature branch
- 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

TIP	TravisCI is highly recommended in Open Source projects. Very easy to set up & get going, all from a simple YAML file.

Prototyping - Get a prototype to your users quickly

Get feedback as soon as possible. Get a 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.

TIP	When building a prototype, the technology is less important to choose for longevity but for speed & possibly to test out potential technology solution.

Optimal time - work when at your best

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.

Not 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.

Codebase improvements - Leave the codebase better than you found it

Many code repositories (mostly Closed Source ones) go from bad to worse. On the other hand, 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.

TIP	No improvement is too small. Make it better.

Fail fast with faster feedback loops

Feedback loops include:

1. Locally
    - Linters
    - Automated tests
2. Continuous Integration (CI)
    - Linters
    - Automated tests
    - Deployment with Continuous Delivery (CD)
3. Manual / Exploratory testing
4. …
5. All the way to the **Client** and then the **User**

The sooner feedback and/or change the more efficient it will be. Therefore on the flip side, the later the feedback and/or change the more costly it will be. Not only because it went through more steps to get there, but after the change has been made, it will have to go through all those steps again.

TIP	Fail fast!

Accessible

Make the project **Accessible**.

This has 2 areas:

Lower the barrier to entry

Allow all people, juniors to seniors to be part of the community & contribute.

- GitHub template more information
- Automated tests more information
- …

The resulting product needs to cater for all

Access by everyone regardless of disability (e.g. visually impaired) is critical to the success of any project.

For example:

- Alternative `alt` text for images
- Keyboard input as well as mouse
- Transcripts for videos
- …and much more!
- A good place to look for ideas would be https://www.w3.org/WAI/standards-guidelines/

TIP	Be inclusive

GitHub Labels

These are really useful for various reasons:

- Helps people filter their results, for example by `defect`, `idea` etc
- If people want to contribute to your project a label that states `help needed` will stand out
- Difficulty level. Contributors can filter by a level they are comfortable with. Having some sort of ranking system like beginner, intermediate and expert will help to make it easier for contributors, especially beginners to know which issues to tackle.

TIP	Have different labels but don't go label crazy, 10-20 is about right

Related

- Issues & Tasks tips

GitHub Milestones

Using **Milestones** not only gives you and your community visibility on the current goal and its progress, but also the future goals and what they contain.

TIP Align your **Release** versions with your **Milestones**

Related

- Release tips

GitHub Releases / Tags

When you are happy with the work done so far, make a **Release** so your community knows, this is a stable version. In the **release notes**, include **change log**.

TIP Align your **Release** versions with your **Milestones**

Related

- Milestone tips

Branching

Branching is important when working with a team, and even more important when working with a wider team you don't even know yet.

Protect your mainline branch (usually `master` or `develop`) and everything must go via a feature branch and a **Pull Request**. This should be the same for everyone, public contributors and approved maintainers.

TIP There are many branching strategies, one example is **Gitflow**, many people use part of Gitflow.

Related

- Pull Requests

DON’Ts

Big bang project

Don’t 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.

TIP	Try to be consistent and do a little every week. Your thoughts and ideas will be better over time, this will prevent you from wasting time on redoing work when you get a light bulb moment a few days later.

God commit

God commits or big bang commits are not good because they are difficul to understand and review, these can block pull requests.

Also if a commit needs to be cherry picked in/out, then it makes it very difficult.

For example if files are being reformatted, this should be done in 1 commit, not with other changes otherwise it is confusing.{

TIP	Each commit should do one single item

God Pull Request

Similar to "god commit", god pull request are a bad idea. Reviewing god or big bang pull request are not only annoying & painful, but leave room for skim reviewing & therefore mistakes. A pull request should be a single feature.

TIP	No one ever complained that a Pull Request was too small.

CV Driven Development

Most of us have heard of Test Driven Development (TDD). Do not fall in to the trap of CV Driven Development (CVDD). It is good to push your skills with new technology but do a little at a time to reduce the risk. Do not try every new technology you want to learn on the same project, this is very high risk, with the mostly likely outcome of little result & therefore frustration.

TIP	Approximately 10 - 20% of new technology per project is a good, safe & exciting amount.

Weakest Dependency

Your project is only as good as your weakest dependency. Check the dependencies you include in your project before including them, look at their:

- license
- contribution activity
- security
- contributors

TIP	Be aware of dependencies you include. It is great not to reinvent the wheel, there are usually a lot of choices available but make sure to do your research on the available libraries.

Appendix

Acronyms

Table 1. Acronyms

Acronym	Description	Notes
TDD	Test Driven Development	GOOD! Wikipedia
BDD	Behaviour Driven Development	GOOD! Wikipedia
CVDD	CV Driven Development	BAD!

Resources

TIP Suggestions welcome…read more at how to contribute

> A great way to get involved in open source is to contribute to the existing projects you're using. GitHub is home to more than 5 million open source projects. There are projects for every skill set like recipes, HTML/CSS, Ruby, Astrophysics and many more. This guide will cover what you might find in a typical project and how to make a great contribution.
>
> — Contributing to Open Source on GitHub, https://guides.github.com/activities/contributing-to-open-source/

> Open source works by having many people contribute to the creation and maintenance of software. Thing is, it works well only when people are actually contributing. Successful open source projects thrive on a wide variety of contributions from people with all levels of coding skills and commitment. If just one person fixes a compiler warning, closes a bug, or adds to the documentation, pretty soon you're talking real progress. For many people, the hardest part is just getting started. So here are some suggested ways you can begin contributing right away, at whatever level is most comfortable for you.
>
> — The Beginner's Guide to Contributing to Open Source Projects, https://blog.newrelic.com/2014/05/05/open-source_gettingstarted/

> GitHub is the home of many popular open source projects like Ruby on Rails, jQuery, Docker, Go and many others. The way people (usually) contribute to an open source project on GitHub is using pull requests. A pull request is basically a patch which includes more information and allows members to discuss it on the website.

— How to contribute to an open source project on GitHub, http://blog.davidecoppola.com/2016/11/howto-contribute-to-open-source-project-on-github/

> It’s okay to not know everything, and take one step at a time to learn something new. Don’t waste a lot of time choosing the “right” project. If you know a project or a organization with a beginner-friendly community, just start there. A huge shout-out to all the open source maintainers who have been super responsive and encourage of new contributors. You are helping newcomers navigate huge code bases and contribute in maybe a small yet meaningful ways. Your efforts are truly appreciated and needed.

— A Beginner’s Very Bumpy Journey Through The World of Open Source, https://medium.freecodecamp.com/a-beginners-very-bumpy-journey-through-the-world-of-open-source-4d108d540b39#.an82epenf

> You want to focus on the building blocks for your community before you get too deep in the code. "We’ll do it right later" doesn’t always work out well. If you don’t make governance decisions now, things can fall apart, or if you make the wrong decision, it could turn off potential community members going forward.

— How to care for the community over the code, https://opensource.com/article/16/12/community-over-code

> For many web developers, accessibility is complex and somewhat difficult. The Accessibility Project understands that and we want to help to make web accessibility easier for front end developers to implement. Blind and visually impaired make up 285,000,000 people according to the World Health Organization (June 2012) with 39,000,000 categorized as legally blind and the remaining 246,000,000 visually impaired. Deaf and hearing impaired make up 275,000,000 (2004) in the moderate-to-profound hearing impairment category.

— Web Accessibility Checklist, http://a11yproject.com/checklist.html