# Some general guidelines

## Before you submit an issue please read [issue submitting guidelines](https://github.com/springfox/springfox/wiki/Issue-Submission)

### Here are some great resources

[Github Flow](https://guides.github.com/introduction/flow/index.html)

[Creating Effective Pull Requests](http://codeinthehole.com/writing/pull-requests-and-other-good-practices-for-teams-using-github/)

Here is an example of [a great pull request](https://github.com/bitly/dablooms/pull/19)

### Before starting to work on a PR

- Create an issue with the problem description \*\*\_BEFORE\_\*\* creating a PR directly. Provide examples if possible. This

gives an opportunity for the library authors to prioritize the work stream as well.

- Describe the approach you'd like to use to solve the problem. This way the library authors will have a better idea

when it comes to reviewing the PR. Its also a good way to vet ideas out, in case there is a simpler solution.

### Unit Testing

\_A lot of effort has gone into refactoring and getting this library into a state where it is thoroughly unit tested\_. This has really paid off and we want to maintain the same diligence with unit testing for as long as this library is maintained. All pull requests must have accompanying unit tests. PR's without unit tests will, most likely, be rejected. The unit testing framework of choice is spock. There are lots of examples in the existing code base and the existing contributors are more than happy to help out with unit testing. \*\*Make sure there are tests that cover the change\*\*.

### Keeping PR process efficient

While working on the PR, here are some suggestions to make submitting a PR fun for everyone!

- Before starting on a feature or a bug, if you see some cleanup opportunities upfront do them all and group these

into one or more logical commits. Definitely if there are whitespace changes I get them out of the way. That way the

code reviewer can focus on reviewing what has actually changed, without having to open up the IDE.

- Never mix class rename/move changes with changes to the class itself.

- Break up your commits into chunks of work. Group logical changes together into one commit. This gives a history and

context as to what the thought processes is or how you arrived at the final solution

- If you encounter significant cleanup required mid-way through your work try to isolate that as a separate commit.

This is not to say ever single minute cleanup needs to be in its own commit, just the large chunks of cleanup.

- Finally clean up all the merge commits by rebasing against the source repo often.

Along the lines of the \_agile manifesto\_:

- Prefer separate commits over one big change

- Prefer isolating code cleanup/formatting issues on a separate commit over mixing it with the commit that addresses

an issue

- Prefer not squashing commits/re-writing history after it has been reviewed

- Prefer squashing commits like "fixed broken test", "oops I forgot to add this before I pushed it" to having the

commit noise

- Prefer squashing merge commits for sure