FPLLL

Contributing

Martin R. Albrecht

2017/07/06

Outline

Overview

All contributions to `fplll`

- are peer-reviewed[1]
- are automatically tested using `make check`[2]
- must follow the coding style
- are checked for test coverage[3]

Fpylll is not quite there yet.

---

[1] This is a lie, some quick fixes are sometimes sneaked through directly

[2] `https://travis-ci.org/fplll/fplll`

[3] `https://codecov.io/gh/fplll/fplll`

Communication

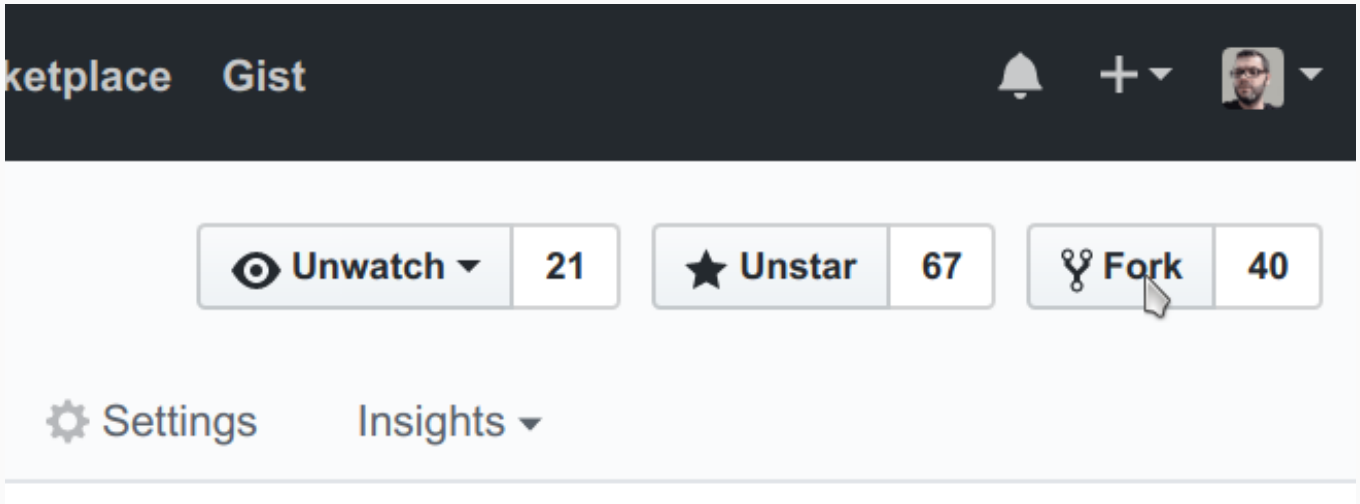
Channels

GitHub	`https://github.com/fplll/fplll/issues`
Mailing List	`https://groups.google.com/forum/#!forum/fplll-devel`
Slack	`https://fplll.slack.com`

It is good for an open-source project to have discussions in public. It shows to others that the project is alive and accessible.

Setup

Fork on GitHub



- Fork `fplll` from `https://github.com/fplll/fplll`
- Fork `fpylll` from `https://github.com/fplll/fpylll`

Clone your Fork Locally

Clone your git repo where my-github-name is your account name on GitHub:

```
$ git clone git@github.com:my-github-name/fplll.git
```

Then run

```
$ ./autogen.sh
$ ./configure
$ make
$ make check
```

as usual.

Reporting Bugs

Reporting Bugs

- `https://github.com/fplll/fplll/issues`.
- `https://groups.google.com/forum/#!forum/fplll-devel`.

GitHub is preferred, also for developers

Topic Branches and Pull Requests

Why I

- Isolate each topic or feature into a “topic branch“.
- Commits allow control over how small individual changes are made to the code.
- Branches
  - group a set of commits together that are related to one feature.
  - isolate different efforts when you might be working on multiple topics at the same time.
- While it takes some experience to get the right feel about how to break up commits, a topic branch should be limited in scope to a single issue.

Why II

- GitHub syncs a pull request to a specific branch.
- Thus, branches are the only way that you can submit more than one fix at a time.
- If you submit a pull from your `master` branch, you cannot make any more commits to your `master` branch without those getting added to the pull.

How I

```
$ git checkout -b fix-broken-thing
Switched to a new branch 'fix-broken-thing'
```

Names

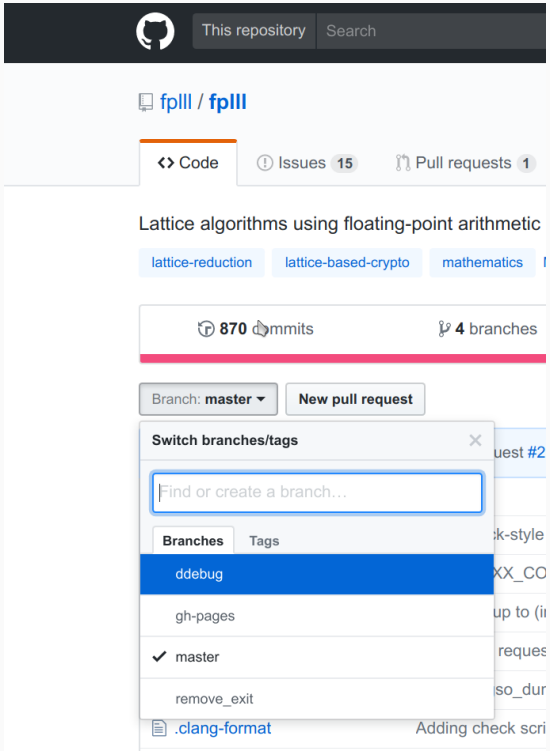
Use a sufficiently verbose name for your branch so it is clear what it is about.

How II

When you are ready to generate a pull request, either for preliminary review or for consideration of merging into the project, you must first push your local topic branch back up to GitHub:

```
$ git push origin fix-broken-thing
```

How III



- Select your topic branch from this list, and then click the ”Pull request” button.
- You can add a comment about your branch.
- If this in response to a submitted issue, link to that issue in this initial comment.
- The maintainers will be notified of your pull request and it will be reviewed.

How iv

- You can continue to add commits to your topic branch (and push them up to GitHub) either if you see something that needs changing, or in response to a reviewer's comments.
- If a reviewer asks for changes, you do not need to close the pull and reissue it after making changes.
- Just make the changes locally, push them to GitHub, then add a comment to the discussion section of the pull request.

Pull upstream changes into your fork regularly I

Pull upstream changes from `master` into your fork on a regular basis.

- Putting in a days of hard work into a pull request only to have it rejected because it has diverged too far from master sucks.
- To pull in upstream changes:

```
$ git remote add upstream https://github.com/fplll/fplll.git
$ git fetch upstream master
```

- Check logs if you actually want the changes before merging:

```
$ git log upstream/master
```

- Then merge the changes that you fetched:

```
$ git merge upstream/master
```

How to Get your Pull Request Accepted

Run tests!

Before you submit a pull request, run tests:

```
$ make check
```

These checks are also run on Travis-CI automatically for every pull request. Nothing failing tests will be accepted.[4]

[4] `https://travis-ci.org/fplll/fplll`, `https://travis-ci.org/fplll/fpylll`

If you add Code, add Tests I

Code that isn't tested is broken.

If you add Code, add Tests II

Keep your tests simple.

- Complex tests end up requiring their own tests.
- We would rather see duplicated assertions across test methods then cunning utility methods that magically determine which assertions are needed at a particular stage.

Explicit is better than implicit.

If you add Code, add Tests III

- The nature of `fplll` means that sometimes it is hard to properly test the behaviour of a change quickly.
- Running BKZ for several minutes takes way too long for a test.
- In this case, we should at least test that a particular piece of code compiles and runs.

Discussion

Should we have `make check-long`?

Keep your Pull Requests limited to a Single Issue

Pull requests should be as small/atomic as possible.

Coding Conventions I

- `fplll` is written in C++11
- We try to make use of its modern features to make the library readable.
- Keep your code as clean and straightforward as possible.
- Code is written for the consumption by compilers and for the consumption by human beings.
- By making code clear and easy to understand, others can build on it and fix issues should they arise.

Coding Conventions II

Our naming convention is close to Python's naming convention.

- Classes are in `CamelCase`.
- Functions, methods, parameters and local variables in `lower_case` .
- Curly braces go on the next line and we prefer explicit curly braces, e.g.

```
if (foo)
{
  do_something_good();
}
```

instead of:

```
if (foo)
  do_something_bad();
```

Coding Conventions III

The pixel shortage is over. We want to see:

- `package` instead of `pkg`
- `grid` instead of `g`
- `my_function_that_does_things` instead of `mftdt`

Enforcement of the Coding Convention I

- The coding convention is enforced throughout the whole project.
- In particular, the code of every pull request has to strictly adhere to the coding convention, and the Travis build will error when it is not the case.
- Automatic formatting can (and should) be performed by the command

```
$ make check-style
```

Enforcement of the Coding Convention II

In order to improve readability, some situations might require manual formatting. Clang-format includes a comment trigger to locally disable the formatting.

```
int formatted_code;
// clang-format off
    void    unformatted_code  ;
// clang-format on
void formatted_code_again;
```

Attribution

- Do not forget to add yourself as a contributor in `README.md` if you make a non-trivial contribution.
- You may want to claim copyright in the copyright headers of each file.

Documentation

Doxygen

fplll uses doxygen with a bootstrap theme to generate API documentation. To produce API documentation run

```
$ doxygen Doxyfile
```

- Our documentation is served at `https://fplll.github.io/fplll/` using GitHub pages.
- To update the documentation, check out the **`gh-pages`** branch and update the html files in there.

GitHub.io

Doxygen writes its outputs to `doc/html`, you can arrange it that this directory holds the `gh-pages` branch of the fplll repository:

```
$ cd doc
$ git clone -b gh-pages git@github.com::<my-github-name>/fplll.git html
$ cd ..
```

- Now, whenever you run `doxygen` it will write its outputs to a directory which holds the right branch.
- If you push it to your remote, you can then check it at `http://my-github-name.github.io/fplll`.

Clean

Before generating documentation with Doxygen to push to `https://fplll.github.io/fplll/` please run

```
$ make maintainer-clean
```

in the fplll root directory.

- This removes **`config.h`** and **`fplll_config.h`**, i.e. it prevents your local, machine-specific configuration to be pushed as part of the official API documentation.
- Review the changes in the **`gh-pages`** branch before committing and pushing.

Thank You