# Guidelines for Contributing

As a scientific community-driven software project, PyMC3 welcomes contributions from interested individuals or groups. These guidelines are provided to give potential contributors information to make their contribution compliant with the conventions of the PyMC3 project, and maximize the probability of such contributions to be merged as quickly and efficiently as possible.

There are 4 main ways of contributing to the PyMC3 project (in descending order of difficulty or scope):

\* Adding new or improved functionality to the existing codebase

\* Fixing outstanding issues (bugs) with the existing codebase. They range from low-level software bugs to higher-level design problems.

\* Contributing or improving the documentation (`docs`) or examples (`pymc3/examples`)

\* Submitting issues related to bugs or desired enhancements

# Opening issues

We appreciate being notified of problems with the existing PyMC code. We prefer that issues be filed the on [Github Issue Tracker](https://github.com/pymc-devs/pymc3/issues), rather than on social media or by direct email to the developers.

Please verify that your issue is not being currently addressed by other issues or pull requests by using the GitHub search tool to look for key words in the project issue tracker.

# Contributing code via pull requests

While issue reporting is valuable, we strongly encourage users who are inclined to do so to submit patches for new or existing issues via pull requests. This is particularly the case for simple fixes, such as typos or tweaks to documentation, which do not require a heavy investment of time and attention.

Contributors are also encouraged to contribute new code to enhance PyMC3's functionality, also via pull requests. Please consult the [PyMC3 documentation](https://pymc-devs.github.io/pymc3/) to ensure that any new contribution does not strongly overlap with existing functionality.

The preferred workflow for contributing to PyMC3 is to fork the [GitHub repository](https://github.com/pymc-devs/pymc3/), clone it to your local machine, and develop on a feature branch.

## Steps:

1. Fork the [project repository](https://github.com/pymc-devs/pymc3/) by clicking on the 'Fork' button near the top right of the main repository page. This creates a copy of the code under your GitHub user account.

2. Clone your fork of the PyMC3 repo from your GitHub account to your local disk, and add the base repository as a remote:

```bash

$ git clone git@github.com:<your GitHub handle>/pymc3.git

$ cd pymc3

$ git remote add upstream git@github.com:pymc-devs/pymc3.git

```

3. Create a ``feature`` branch to hold your development changes:

```bash

$ git checkout -b my-feature

```

Always use a ``feature`` branch. It's good practice to never routinely work on the ``master`` branch of any repository.

4. Project requirements are in ``requirements.txt``, and libraries used for development are in ``requirements-dev.txt``. To set up a development environment, you may (probably in a [virtual environment](https://docs.python-guide.org/dev/virtualenvs/)) run:

```bash

$ pip install -r requirements.txt

$ pip install -r requirements-dev.txt

```

Alternatively, there is a script to create a docker environment for development. See: [Developing in Docker](#Developing-in-Docker).

5. Develop the feature on your feature branch. Add changed files using ``git add`` and then ``git commit`` files:

```bash

$ git add modified\_files

$ git commit

```

to record your changes locally.

After committing, it is a good idea to sync with the base repository in case there have been any changes:

```bash

$ git fetch upstream

$ git rebase upstream/master

```

Then push the changes to your GitHub account with:

```bash

$ git push -u origin my-feature

```

6. Go to the GitHub web page of your fork of the PyMC3 repo. Click the 'Pull request' button to send your changes to the project's maintainers for review. This will send an email to the committers.

## Pull request checklist

We recommended that your contribution complies with the following guidelines before you submit a pull request:

\* If your pull request addresses an issue, please use the pull request title to describe the issue and mention the issue number in the pull request description. This will make sure a link back to the original issue is created.

\* All public methods must have informative docstrings with sample usage when appropriate.

\* Please prefix the title of incomplete contributions with `[WIP]` (to indicate a work in progress). WIPs may be useful to (1) indicate you are working on something to avoid duplicated work, (2) request broad review of functionality or API, or (3) seek collaborators.

\* All other tests pass when everything is rebuilt from scratch. See

[Developing in Docker](#Developing-in-Docker) for information on running the test suite locally.

\* When adding additional functionality, provide at least one example script or Jupyter Notebook in the ``pymc3/examples/`` folder. Have a look at other examples for reference. Examples should demonstrate why the new functionality is useful in practice and, if possible, compare it to other methods available in PyMC3.

\* Documentation and high-coverage tests are necessary for enhancements to be accepted.

\* Run any of the pre-existing examples in ``docs/source/notebooks`` that contain analyses that would be affected by your changes to ensure that nothing breaks. This is a useful opportunity to not only check your work for bugs that might not be revealed by unit test, but also to show how your contribution improves PyMC3 for end users.

You can also check for common programming errors with the following

tools:

\* Code with good test \*\*coverage\*\* (at least 80%), check with:

```bash

$ pip install pytest pytest-cov coverage

$ pytest --cov=pymc3 pymc3/tests/<name of test>.py

```

\* No `pyflakes` warnings, check with:

```bash

$ pip install pyflakes

$ pyflakes path/to/module.py

```

\* No PEP8 warnings, check with:

```bash

$ pip install pycodestyle

$ pycodestyle path/to/module.py

```

\* AutoPEP8 can help you fix some of the easy redundant errors:

```bash

$ pip install autopep8

$ autopep8 path/to/pep8.py

```

## Developing in Docker

We have provided a Dockerfile which helps for isolating build problems, and local development.

Install [Docker](https://www.docker.com/) for your operating system, clone this repo, then

run `./scripts/start\_container.sh`. This should start a local docker container called `pymc3`,

as well as a [`jupyter`](http://jupyter.org/) notebook server running on port 8888. The

notebook should be opened in your browser automatically (you can disable this by passing

`--no-browser`). The repo will be running the code from your local copy of `pymc3`,

so it is good for development.

You may also use it to run the test suite, with

```bash

$ docker exec -it pymc3 bash # logon to the container

$ cd ~/pymc3/tests

$ . ./../../scripts/test.sh # takes a while!

```

This should be quite close to how the tests run on TravisCI.

If the container was started without opening the browser, you

need the notebook instances token to work with the notebook. This token can be

accessed with

```

docker exec -it pymc3 jupyter notebook list

```

## Style guide

Follow [TensorFlow's style guide](https://www.tensorflow.org/community/contribute/code\_style) or the [Google style guide](https://google.github.io/styleguide/pyguide.html) for writing code, which more or less follows PEP 8.

For documentation strings, we \*prefer\* [numpy style](https://numpydoc.readthedocs.io/en/latest/format.html) to comply with the style that predominates in our upstream dependencies.

#### This guide was derived from the [scikit-learn guide to contributing](https://github.com/scikit-learn/scikit-learn/blob/master/CONTRIBUTING.md)