# Contributing

## Issues

A good issue includes a [short, self contained, correct example](http://sscce.org/) of the problem, something like:

```python

assert github.Github().get\_user("jacquev6").name == "Vincent Jacques"

```

It is even better if you provide the debug logs associated with your issue.

Enable them with `github.enable\_console\_debug\_logging` and copy them in the body of the issue.

\*\*Warning:\*\* you may want to remove some private information (authentication information is removed, but there may be private stuff in the messages)

If for any reason you are not able to do that, open your issue anyway and a maintainer will see what is needed to solve your problem.

## Pull Requests

Pull Requests should clearly describe two things:

1. The problem they attempt to solve

2. How the author went about solving the problem

Ideally, changes should be made in logical commits and tests added to improve the project's coverage of the GitHub API.

## Coding style

PyGithub adopts the black coding style and uses isort to sort imports.

To manually format the code:

```

tox -e lint

```

## Pre-commit plugin

To forget about coding style and let [pre-commit](https://pre-commit.com/#installation) fix your flake8/isort/black issue.

```

pre-commit install

```

That's it!

## Adding missing attributes for a GithubObject

```bash

$ python scripts/add\_attribute.py [class\_name] [attribute\_name] [attribute\_type]

# For example, if you want to add a `url` attribute of string type to the Commit class

# Note: adding multiple attributes you have to run the script multiple times

$ python scripts/add\_attribute.py Commit url string

```

## Deprecation warning

Before removing attributes/methods, consider adding deprecation warnings instead. The [Deprecated](https://github.com/tantale/deprecated) packages provides a handy decorator to add deprecation warnings with an optional reason.

```python

from deprecated import deprecated

@property

@deprecated

def rate(self):

pass

@deprecated(reason="Deprecated in favor of the new branch protection")

def get\_protected\_branch(self):

pass

```

## Automated tests

First you need to install the test dependencies:

```bash

pip install -r test-requirements.txt

```

Then you can run the tests through `pytest`.

Run a specific test with `pytest tests/tests\_filename.py` or `pytest tests/tests\_filename.py -k testMethod` or `pytest -k TestClass.testMethod`.

If you add a new test, for example `Issue139.testCompletion`, you have to run `pytest -k Issue139.testCompletion --record` to create the `tests/ReplayData/\*.txt` files needed for your new test.

Check them and commit them as well.

You will need a `GithubCredentials.py` file at the root of the project with the following contents:

```

login = "my\_login"

password = "my\_password" # Can be left empty if not used

oauth\_token = "my\_token" # Can be left empty if not used

jwt = "my\_json\_web\_token" # Can be left empty if not used

```

If you use 2 factor authentication on your Github account, tests that require a login/password authentication will fail.

You can use `pytest Issue139.testCompletion --record --auth\_with\_token` to use the `oauth\_token` field specified in `GithubCredentials.py` when recording a unit test interaction. Note that the `password = ""` (empty string is ok) must still be present in `GithubCredentials.py` to run the tests even when the `--auth\_with\_token` arg is used. (Also note that if you record your test data with `--auth\_with\_token` then you also need to be in token authentication mode when running the test. A simple alternative is to replace `token private\_token\_removed` with `Basic login\_and\_password\_removed` in all your newly generated ReplayData files.)

Similarly, you can use `pytest Issue139.testCompletion --record --auth\_with\_jwt` to use the `jwt` field specified in `GithubCredentials.py` to access endpoints that require JWT.

To run manual tests with external scripts that use the PyGithub package, you can install your development version with:

```

pip install --editable path/to/project

```

You may also want to investigate `tox` to run tests:

```

pip install tox

tox -epy36

```

## Build documentation locally

```

pip install -r requirements.txt

sphinx-build doc build

```

If you use tox:

```

tox -edocs

```