# Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

\* [Types of Contributions](#Types-of-Contributions)

\* [Contributor Setup](#Setting-Up-the-Code-for-Local-Development)

\* [Contributor Guidelines](#Contributor-Guidelines)

\* [Contributor Testing](#Testing-with-tox)

\* [Core Committer Guide](#Core-Committer-Guide)

## Types of Contributions

You can contribute in many ways:

### Create Cookiecutter Templates

Some other Cookiecutter templates to list in the [README](README.md) would be great.

If you create a Cookiecutter template, submit a pull request adding it to README.md.

### Report Bugs

Report bugs at [https://github.com/cookiecutter/cookiecutter/issues](https://github.com/cookiecutter/cookiecutter/issues).

If you are reporting a bug, please include:

\* Your operating system name and version.

\* Any details about your local setup that might be helpful in troubleshooting.

\* If you can, provide detailed steps to reproduce the bug.

\* If you don't have steps to reproduce the bug, just note your observations in as much detail as you can. Questions to start a discussion about the issue are welcome.

### Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

### Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "please-help" is open to whoever wants to implement it.

Please do not combine multiple feature enhancements into a single pull request.

Note: this project is very conservative, so new features that aren't tagged with "please-help" might not get into core. We're trying to keep the code base small, extensible, and streamlined. Whenever possible, it's best to try and implement feature ideas as separate projects outside of the core codebase.

### Write Documentation

Cookiecutter could always use more documentation, whether as part of the official Cookiecutter docs, in docstrings, or even on the web in blog posts, articles, and such.

If you want to review your changes on the documentation locally, you can do:

```bash

pip install -r docs/requirements.txt

make servedocs

```

This will compile the documentation, open it in your browser and start watching the files for changes, recompiling as you save.

### Submit Feedback

The best way to send feedback is to file an issue at [https://github.com/cookiecutter/cookiecutter/issues](https://github.com/cookiecutter/cookiecutter/issues).

If you are proposing a feature:

\* Explain in detail how it would work.

\* Keep the scope as narrow as possible, to make it easier to implement.

\* Remember that this is a volunteer-driven project, and that contributions are welcome :)

## Setting Up the Code for Local Development

Here's how to set up `cookiecutter` for local development.

1. Fork the `cookiecutter` repo on GitHub.

2. Clone your fork locally:

```bash

git clone git@github.com:your\_name\_here/cookiecutter.git

```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```bash

mkvirtualenv cookiecutter

cd cookiecutter/

python setup.py develop

```

4. Create a branch for local development:

```bash

git checkout -b name-of-your-bugfix-or-feature

```

Now you can make your changes locally.

5. When you're done making changes, check that your changes pass the tests and lint check:

```bash

pip install tox

tox

```

Please note that tox runs lint check automatically, since we have a test environment for it.

If you feel like running only the lint environment, please use the following command:

```bash

tox -e lint

```

6. Ensure that your feature or commit is fully covered by tests. Check report after regular tox run. You can also run coverage only report and get html report with statement by statement highlighting:

```bash

tox -e cov-report

```

You report will be placed to `htmlcov` directory. Please do not include this directory to your commits. By default this directory in our `.gitignore` file.

7. Commit your changes and push your branch to GitHub:

```bash

git add .

git commit -m "Your detailed description of your changes."

git push origin name-of-your-bugfix-or-feature

```

8. Submit a pull request through the GitHub website.

## Contributor Guidelines

### Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.

2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.md.

3. The pull request must pass all CI/CD jobs before being ready for review.

4. If one CI/CD job is failing for unrelated reasons you may want to create another PR to fix that first.

### Coding Standards

\* PEP8

\* Functions over classes except in tests

\* Quotes via [http://stackoverflow.com/a/56190/5549](http://stackoverflow.com/a/56190/5549)

\* Use double quotes around strings that are used for interpolation or that are natural language messages

\* Use single quotes for small symbol-like strings (but break the rules if the strings contain quotes)

\* Use triple double quotes for docstrings and raw string literals for regular expressions even if they aren't needed.

\* Example:

```python

LIGHT\_MESSAGES = {

'English': "There are %(number\_of\_lights)s lights.",

'Pirate': "Arr! Thar be %(number\_of\_lights)s lights."

}

def lights\_message(language, number\_of\_lights):

"""Return a language-appropriate string reporting the light count."""

return LIGHT\_MESSAGES[language] % locals()

def is\_pirate(message):

"""Return True if the given message sounds piratical."""

return re.search(r"(?i)(arr|avast|yohoho)!", message) is not None

```

\* Write new code in Python 3.

## Testing with tox

Tox uses py.test under the hood, hence it supports the same syntax for selecting tests.

For further information please consult the [`pytest usage docs`](http://pytest.org/en/latest/example/index.html).

To run a particular test class with tox:

```bash

tox -e py '-k TestFindHooks'

```

To run some tests with names matching a string expression:

```bash

tox -e py '-k generate'

```

Will run all tests matching "generate", test\_generate\_files for example.

To run just one method:

```bash

tox -e py '-k "TestFindHooks and test\_find\_hook"'

```

To run all tests using various versions of python in virtualenvs defined in tox.ini, just run tox:

```bash

tox

```

This configuration file setup the pytest-cov plugin and it is an additional dependency. It generate a coverage report after the tests.

It is possible to tests with some versions of python, to do this the command is:

```bash

tox -e py36,pypy3

```

Will run py.test with the python3.6 and pypy3 interpreters, for example.

## Core Committer Guide

### Vision and Scope

Core committers, use this section to:

\* Guide your instinct and decisions as a core committer

\* Limit the codebase from growing infinitely

#### Command-Line Accessible

\* Provides a command-line utility that creates projects from cookiecutters

\* Extremely easy to use without having to think too hard

\* Flexible for more complex use via optional arguments

#### API Accessible

\* Entirely function-based and stateless (Class-free by intentional design)

\* Usable in pieces for developers of template generation tools

#### Being Jinja2-specific

\* Sets a standard baseline for project template creators, facilitating reuse

\* Minimizes the learning curve for those who already use Flask or Django

\* Minimizes scope of Cookiecutter codebase

#### Extensible

Being extendable by people with different ideas for Jinja2-based project template tools.

\* Entirely function-based

\* Aim for statelessness

\* Lets anyone write more opinionated tools

Freedom for Cookiecutter users to build and extend.

\* No officially-maintained cookiecutter templates, only ones by individuals

\* Commercial project-friendly licensing, allowing for private cookiecutters and private Cookiecutter-based tools

#### Fast and Focused

Cookiecutter is designed to do one thing, and do that one thing very well.

\* Cover the use cases that the core committers need, and as little as possible beyond that :)

\* Generates project templates from the command-line or API, nothing more

\* Minimize internal line of code (LOC) count

\* Ultra-fast project generation for high performance downstream tools

#### Inclusive

\* Cross-platform and cross-version support are more important than features/functionality

\* Fixing Windows bugs even if it's a pain, to allow for use by more beginner coders

#### Stable

\* Aim for 100% test coverage and covering corner cases

\* No pull requests will be accepted that drop test coverage on any platform, including Windows

\* Conservative decisions patterned after CPython's conservative decisions with stability in mind

\* Stable APIs that tool builders can rely on

\* New features require a +1 from 3 core committers

#### VCS-Hosted Templates

Cookiecutter project templates are intentionally hosted VCS repos as-is.

\* They are easily forkable

\* It's easy for users to browse forks and files

\* They are searchable via standard Github/Bitbucket/other search interface

\* Minimizes the need for packaging-related cruft files

\* Easy to create a public project template and host it for free

\* Easy to collaborate

### Process: Pull Requests

If a pull request is untriaged:

\* Look at the roadmap

\* Set it for the milestone where it makes the most sense

\* Add it to the roadmap

How to prioritize pull requests, from most to least important:

\* Fixes for broken tests. Broken means broken on any supported platform or Python version.

\* Extra tests to cover corner cases.

\* Minor edits to docs.

\* Bug fixes.

\* Major edits to docs.

\* Features.

Ensure that each pull request meets all requirements in [checklist](https://gist.github.com/audreyr/4feef90445b9680475f2).

### Process: Issues

If an issue is a bug that needs an urgent fix, mark it for the next patch release.

Then either fix it or mark as please-help.

For other issues: encourage friendly discussion, moderate debate, offer your thoughts.

New features require a +1 from 2 other core committers (besides yourself).

### Process: Roadmap

The roadmap located [here](https://github.com/cookiecutter/cookiecutter/milestones?direction=desc&sort=due\_date&state=open)

Due dates are flexible. Core committers can change them as needed. Note that GitHub sort on them is buggy.

How to number milestones:

\* Follow semantic versioning. Look at: [http://semver.org](http://semver.org)

Milestone size:

\* If a milestone contains too much, move some to the next milestone.

\* Err on the side of more frequent patch releases.

### Process: Pull Request merging and HISTORY.md maintenance

If you merge a pull request, you're responsible for updating `AUTHORS.md` and `HISTORY.md`

When you're processing the first change after a release, create boilerplate following the existing pattern:

```md

## x.y.z (Development)

The goals of this release are TODO: release summary of features

Features:

\* Feature description, thanks to [@contributor](https://github.com/contributor) (#PR).

Bug Fixes:

\* Bug fix description, thanks to [@contributor](https://github.com/contributor) (#PR).

Other changes:

\* Description of the change, thanks to [@contributor](https://github.com/contributor) (#PR).

```

### Process: Accepting Template Pull Requests

\* Run the template to generate the project.

\* Attempt to start/use the rendered project.

\* Merge the template in.

\* Update the history file.

note: Adding a template doesn't give authors credit.

### Process: Your own code changes

All code changes, regardless of who does them, need to be reviewed and merged by someone else.

This rule applies to all the core committers.

Exceptions:

\* Minor corrections and fixes to pull requests submitted by others.

\* While making a formal release, the release manager can make necessary, appropriate changes.

\* Small documentation changes that reinforce existing subject matter. Most commonly being, but not limited to spelling and grammar corrections.

### Responsibilities

\* Ensure cross-platform compatibility for every change that's accepted. Windows, Mac, Debian & Ubuntu Linux.

\* Ensure that code that goes into core meets all requirements in this checklist: [https://gist.github.com/audreyr/4feef90445b9680475f2](https://gist.github.com/audreyr/4feef90445b9680475f2)

\* Create issues for any major changes and enhancements that you wish to make. Discuss things transparently and get community feedback.

\* Don't add any classes to the codebase unless absolutely needed. Err on the side of using functions.

\* Keep feature versions as small as possible, preferably one new feature per version.

\* Be welcoming to newcomers and encourage diverse new contributors from all backgrounds. Look at [Code of Conduct](CODE\_OF\_CONDUCT.md).

### Becoming a Core Committer

Contributors may be given core commit privileges. Preference will be given to those with:

1. Past contributions to Cookiecutter and other open-source projects. Contributions to Cookiecutter include both code (both accepted and pending) and friendly participation in the issue tracker. Quantity and quality are considered.

2. A coding style that the other core committers find simple, minimal, and clean.

3. Access to resources for cross-platform development and testing.

4. Time to devote to the project regularly.