# Contributing to Kitten Tricks

We would love for you to contribute to Kitten Tricks and help make it ever better together! :rocket:

- [Code of Conduct](#coc)

- [Question or Problem?](#question)

- [Issues and Bugs](#issue)

- [Feature Requests](#feature)

- [Submission Guidelines](#submit-pr)

- [Coding Rules](#rules)

- [Commit Message Guidelines](#commit)

## <a name="coc"></a> Code of Conduct

Help us keep Kitten Tricks open and inclusive. Please read and follow our [Code of Conduct][coc].

## <a name="question"></a> Got a Question or Problem?

Please do not open issues for general support questions as we want to keep GitHub issues for bug reports and feature requests. You've got much better chances of getting your question answered on [StackOverflow][stackoverflow] where the questions should be tagged with tag `Kitten Tricks`.

StackOverflow is a much better place to ask questions since:

- there are thousands of people willing to help on StackOverflow

- questions and answers stay available for public viewing so your question / answer might help someone else

- StackOverflow's voting system assures that the best answers are prominently visible.

To save your and our time, we will be systematically closing all the issues that are requests for general support and redirecting people to StackOverflow.

## <a name="issue"></a> Found an Issue?

If you find a bug in the source code or a mistake in the documentation, you can help us by [submitting an issue](#submit-issue) to our [GitHub Repository][github]. Including an issue reproduction is the absolute best way to help the team quickly diagnose the problem. Screenshots and error stack traces are also helpful.

Please follow this simple checklist before submitting:

\* If you have a question about using Kitten Tricks, please ask on the [StackOverflow][stackoverflow].

\* It is required that you clearly describe the steps necessary to reproduce the issue you are running into. Although we would love to help our users as much as possible, diagnosing issues without clear reproduction steps is extremely time-consuming and simply not sustainable.

\* The issue list of this repository is exclusively for bug reports and feature requests. Non-conforming issues will be closed immediately.

\* Issues with no clear steps to reproduce will not be triaged. If an issue is labeled with "needs info" and receives no further replies from the author of the issue for more than 5 days, it will be closed.

\* If you think you have found a bug, or have a new feature idea, please start by making sure it hasn't already been [reported][issues]. You can search through existing issues to see if there is a similar one reported. Include closed issues as it may have been closed with a solution.

\* Next, [create a new issue](#submit-issue) that thoroughly explains the problem. Please fill out the populated issue form before submitting the issue.

## <a name="feature"></a> Want a Feature?

You can \*request\* a new feature by [submitting an issue](#submit-issue) to our [GitHub Repository][github]. If you would like to \*implement\* a new feature, please submit an issue with a proposal for your work first, to be sure that we can use it. Please consider what kind of change it is:

\* For a \*\*Major Feature\*\*, first open an issue and outline your proposal so that it can be

discussed. This will also allow us to better coordinate our efforts, prevent duplication of work,

and help you to craft the change so that it is successfully accepted into the project.

\* \*\*Small Features\*\* can be crafted and directly [submitted as a Pull Request](#submit-pr).

### <a name="submit-issue"></a> Submitting an Issue Before you submit an issue, search the archive, maybe your question was already answered.

If your issue appears to be a bug, and hasn't been reported, open a new issue. Help us to maximize the effort we can spend fixing issues and adding new features by not reporting duplicate issues. Please make sure to fill out the populated issue form before submitting the issue.

You can file new issues by providing the information [here][new\_issue].

### <a name="submit-pr"></a> Submitting a Pull Request (PR)

Before you submit your Pull Request (PR) consider the following guidelines:

\* Search [GitHub][pulls] for an open or closed PR

that relates to your submission. You don't want to duplicate effort.

\* Make your changes in a new git branch:

```shell

git checkout -b my-fix-branch master

```

\* Read the [developer documentation][dev-doc].

\* Create your patch, \*\*including appropriate test cases\*\*.

\* Follow our [Coding Rules](#rules).

\* Test your changes.

\* Commit your changes using a descriptive commit message that follows our [commit message conventions](#commit). Adherence to these conventions is necessary because release notes are automatically generated from these messages.

```shell

git commit -a

```

Note: the optional commit `-a` command line option will automatically "add" and "rm" edited files.

\* Push your branch to GitHub:

```shell

git push my-fork my-fix-branch

```

\* In GitHub, send a pull request to `master`.

\* If we suggest changes then:

\* Make the required updates.

\* Rebase your branch and force push to your GitHub repository (this will update your Pull Request):

```shell

git rebase master -i

git push -f

```

That's it! Thank you for your contribution!

#### After your pull request is merged

After your pull request is merged, you can safely delete your branch and pull the changes

from the main (upstream) repository:

\* Delete the remote branch on GitHub either through the GitHub web UI or your local shell as

follows:

```shell

git push my-fork --delete my-fix-branch

```

\* Check out the master branch:

```shell

git checkout master -f

```

\* Delete the local branch:

```shell

git branch -D my-fix-branch

```

\* Update your master with the latest upstream version:

```shell

git pull --ff upstream master

```

## <a name="rules"></a> Coding Rules

To ensure consistency throughout the source code, keep these rules in mind as you are working:

\* All features or bug fixes \*\*must be tested\*\* by one or more specs (unit-tests).

\* All public API methods \*\*must be documented\*\* following JsDoc notation.

## <a name="commit"></a> Commit Message Guidelines

We have very precise rules over how our git commit messages can be formatted. This leads to \*\*more

readable messages\*\* that are easy to follow when looking through the \*\*project history\*\*. But also,

we use the git commit messages to \*\*generate the Kitten Tricks change log\*\*.

### Commit Message Format

Each commit message consists of a \*\*header\*\*, a \*\*body\*\* and a \*\*footer\*\*. The header has a special

format that includes a \*\*type\*\*, a \*\*scope\*\* and a \*\*subject\*\*:

```

<type>(<scope>): <subject>

<BLANK LINE>

<body>

<BLANK LINE>

<footer>

```

The \*\*header\*\* is mandatory and the \*\*scope\*\* of the header is optional.

Any line of the commit message cannot be longer 100 characters! This allows the message to be easier

to read on GitHub as well as in various git tools.

### Revert

If the commit reverts a previous commit, it should begin with `revert: `, followed by the header of

the reverted commit. In the body it should say: `This reverts commit <hash>.`, where the hash is

the SHA of the commit being reverted.

### Type

Must be one of the following:

\* \*\*feat\*\*: A new feature

\* \*\*fix\*\*: A bug fix

\* \*\*docs\*\*: Documentation only changes

\* \*\*style\*\*: Changes that do not affect the meaning of the code (white-space, formatting, missing

semi-colons, etc)

\* \*\*refactor\*\*: A code change that neither fixes a bug nor adds a feature

\* \*\*perf\*\*: A code change that improves performance

\* \*\*test\*\*: Adding missing tests or correcting existing tests

\* \*\*build\*\*: Changes that affect the build system, CI configuration or external dependencies

(example scopes: gulp, broccoli, npm)

\* \*\*chore\*\*: Other changes that don't modify `src` or `test` files

\* \*\*relese\*\*: Release version commit

### Scope

The scope could be anything specifying place of the commit change. For example

`menu`, `sidebar`, etc.

### Subject

The subject contains succinct description of the change:

\* use the imperative, present tense: "change" not "changed" nor "changes"

\* don't capitalize first letter

\* no dot (.) at the end

### Body

Optional. Just as in the \*\*subject\*\*, use the imperative, present tense: "change" not "changed" nor "changes".

The body should include the motivation for the change and contrast this with previous behavior.

### Footer

Optional. The footer should contain any information about \*\*Breaking Changes\*\* and is also the place to

reference GitHub issues that this commit \*\*Closes\*\*.

\*\*Breaking Changes\*\* should start with the word `BREAKING CHANGE:` with a space or two newlines.

The rest of the commit message is then used for this.

[coc]: CODE\_OF\_CONDUCT.md

[dev-doc]: DEV\_DOCS.md

[github]: https://github.com/akveo/kittenTricks

[stackoverflow]: https://stackoverflow.com/questions/tagged/kittenTricks

[issues]: https://github.com/akveo/kittenTricks/issues

[new\_issue]: https://github.com/akveo/kittenTricks/issues/new

[pulls]: https://github.com/akveo/kittenTricks/pulls