# Contributing to Gitlab

Please take a moment to review this document in order to make the contribution

process easy and effective for everyone involved!

## Using the issue tracker

You can use the issues tracker for:

\* [bug reports](#bug-reports)

\* [feature requests](#feature-requests)

\* [submitting pull requests](#pull-requests)

Use [Stackoverflow](http://stackoverflow.com/) for questions and personal support requests.

## Bug reports

A bug is a \_demonstrable problem\_ that is caused by the code in the repository.

Good bug reports are extremely helpful - thank you!

Guidelines for bug reports:

1. \*\*Use the GitHub issue search\*\* &mdash; check if the issue has already been

reported.

2. \*\*Check if the issue has been fixed\*\* &mdash; try to reproduce it using the

`master` branch in the repository.

3. \*\*Isolate and report the problem\*\* &mdash; ideally create a reduced test

case.

Please try to be as detailed as possible in your report. Include information about

your Ruby, Gitlab client and GitLab instance versions. Please provide steps to

reproduce the issue as well as the outcome you were expecting! All these details

will help developers to fix any potential bugs.

Example:

> Short and descriptive example bug report title

>

> A summary of the issue and the environment in which it occurs. If suitable,

> include the steps required to reproduce the bug.

>

> 1. This is the first step

> 2. This is the second step

> 3. Further steps, etc.

>

> Any other information you want to share that is relevant to the issue being

> reported. This might include the lines of code that you have identified as

> causing the bug, and potential solutions (and your opinions on their

> merits).

## Feature requests

Feature requests are welcome. But take a moment to find out whether your idea

fits with the scope and aims of the project. It's up to \*you\* to make a strong

case to convince the community of the merits of this feature.

Please provide as much detail and context as possible.

## Contributing Documentation

Code documentation has a special convention: it uses [YARD](http://yardoc.org/)

formatting and the first paragraph is considered to be a short summary.

For methods say what it will do. For example write something like:

```ruby

# Reverses the contents of a String or IO object.

#

# @param [String, #read] contents the contents to reverse

# @return [String] the contents reversed lexically

def reverse(contents)

contents = contents.read if contents.respond\_to? :read

contents.reverse

end

```

For classes, modules say what it is. For example write something like:

```ruby

# Defines methods related to groups.

module Groups

```

Keep in mind that the documentation notes might show up in a summary somewhere,

long texts in the documentation notes create very ugly summaries. As a rule of thumb

anything longer than 80 characters is too long.

Try to keep unnecessary details out of the first paragraph, it's only there to

give a user a quick idea of what the documented "thing" does/is. The rest of the

documentation notes can contain the details, for example parameters and what

is returned.

If possible include examples. For example:

```ruby

# Gets information about a project.

#

# @example

# Gitlab.project(3)

# Gitlab.project('gitlab')

#

# @param [Integer, String] id The ID or name of a project.

# @return [Gitlab::ObjectifiedHash]

def project(id)

```

This makes it easy to test the examples so that they don't go stale and examples

are often a great help in explaining what a method does.

## Pull requests

Good pull requests - patches, improvements, new features - are a fantastic

help. They should remain focused in scope and avoid containing unrelated

commits.

\*\*IMPORTANT\*\*: By submitting a patch, you agree that your work will be

licensed under the license used by the project.

If you have any large pull request in mind (e.g. implementing features,

refactoring code, etc), \*\*please ask first\*\* otherwise you risk spending

a lot of time working on something that the project's developers might

not want to merge into the project.

Please adhere to the coding conventions in the project (indentation,

accurate comments, etc.) and don't forget to add your own tests and

documentation. When working with git, we recommend the following process

in order to craft an excellent pull request:

1. [Fork](https://help.github.com/articles/fork-a-repo/) the project, clone your fork,

and configure the remotes:

```sh

# Clone your fork of the repo into the current directory

git clone https://github.com/<your-username>/gitlab

# Navigate to the newly cloned directory

cd gitlab

# Assign the original repo to a remote called "upstream"

git remote add upstream https://github.com/NARKOZ/gitlab

```

2. If you cloned a while ago, get the latest changes from upstream:

```bash

git checkout master

git pull upstream master

```

3. Create a new topic branch (off of `master`) to contain your feature, change,

or fix.

\*\*IMPORTANT\*\*: Making changes in `master` is discouraged. You should always

keep your local `master` in sync with upstream `master` and make your

changes in topic branches.

```sh

git checkout -b <topic-branch-name>

```

4. Commit your changes in logical chunks. Keep your commit messages organized,

with a short description in the first line and more detailed information on

the following lines. Feel free to use Git's

[interactive rebase](https://help.github.com/articles/about-git-rebase/)

feature to tidy up your commits before making them public.

5. Make sure all the tests are still passing.

```sh

rake

```

6. Push your topic branch up to your fork:

```sh

git push origin <topic-branch-name>

```

7. [Open a Pull Request](https://help.github.com/articles/using-pull-requests/)

with a clear title and description.

8. If you haven't updated your pull request for a while, you should consider

rebasing on master and resolving any conflicts.

\*\*IMPORTANT\*\*: \_Never ever\_ merge upstream `master` into your branches. You

should always `git rebase` on `master` to bring your changes up to date when

necessary.

```sh

git checkout master

git pull upstream master

git checkout <your-topic-branch>

git rebase master

```

Thank you for your contributions!