- [Filing an issue](#filing-an-issue)

- [Must read](#must-read)

- [Good to know](#good-to-know)

- [How to start contributing](#how-to-start-contributing)

- [Before you start…](#before-you-start)

- [Must read](#must-read)

- [Pull request](#pull-request)

- [How to open a pull request](#how-to-open-a-pull-request)

- [How to deal with large amounts of merge conflicts](#merge-conflicts)

- [Git Commit Guidelines](#commit)

- [Commit Message Format](#commit-message-format)

- [Header](#header)

- [Type](#type)

- [Scope](#scope)

- [Subject](#subject)

- [Body](#body)

- [Reviewing](#reviewing)

- [Testing PRs](#testing-prs)

- [Git config](#git-config)

- [Coding guidelines](#coding-guidelines)

Note that you don't need to know all of the `CONTRIBUTING.md` – it is there to

help you with things as you go, and make things easier, not harder.

Skim through it, and when you will be doing something that relevant section

will apply to, just go back to it and read in more detail about what is the

best course of action. You don't even need to memorize the section – after all,

it still will be there next time you might need it. `:-)`

# Filing an issue

### Must read

\* If you aren't sure, you can ask on the

[\*\*IRC channel\*\*](https://webchat.freenode.net/?channels=qtox) or read our

[\*\*wiki\*\*](https://github.com/qTox/qTox/wiki) first.

\* Do a quick \*\*search\*\*. Others might have already reported the issue.

\* Write in \*\*English\*\*!

\* Provide \*\*version\*\* information (you can find version numbers in menu

`Settings ? About`):

```

OS:

qTox version:

Commit hash:

toxcore:

Qt:

```

\* Provide \*\*steps\*\* to reproduce the problem, it will be easier to pinpoint the

fault.

\* \*\*Screenshots\*\*! A screenshot is worth a thousand words. Just upload it.

[(How?)](https://help.github.com/articles/file-attachments-on-issues-and-pull-requests)

### Good to know

\* \*\*Patience\*\*. The dev team is small and resource limited. Devs have to find

time, analyze the problem and fix the issue, it all takes time. :clock3:

\* If you can code, why not become a \*\*contributor\*\* by fixing the issue and

opening a pull request? :wink:

\* Harsh words or threats won't help your situation. What's worse, your complaint

will (very likely) be \*\*ignored\*\*. :fearful:

# How to start contributing

## Before you start…

Before you start contributing, first decide for a specific topic you want to

work on. Pull requests, which are spanning multiple topics (e.g. "general qTox

code cleanup") or introduce fundamental architectural changes are rare and

require additional attention and maintenance. Please also read the following

simple rules we need to keep qTox a "smooth experience" for everybody involved.

## Must read:

\* Use [\*\*commit message format\*\*](#commit-message-format).

\* Read our [\*\*coding guidelines\*\*](#coding-guidelines).

\* Keep the title \*\*short\*\* and provide a \*\*clear\*\* description about what your

pull request does.

\* Provide \*\*screenshots\*\* for UI related changes.

\* Keep your git commit history \*\*clean\*\* and \*\*precise\*\* by continuously

rebasing/amending your PR. Commits like `xxx fixup` are not needed and

rejected during review.

\* Commit message should state not only what has been changed, but also why a

change is needed.

\* If your commit fixes a reported issue (for example #4134), add the following

message to the commit `Fixes #4134.`. [Here is an

example](https://github.com/qTox/qTox/commit/87160526d5bafcee7869d6741a06045e13d731d5).

## Pull request

\*PR = Pull request\*

Ideally for simple PRs (most of them):

\* One topic per PR

\* One commit per PR

\* If you have several commits on different topics, close the PR and create one

PR per topic

\* If you still have several commits, squash them into only one commit

\* Amend commit after making changes (`git commit --amend path/to/file`)

\* Rebase your PR branch on top of upstream `master` before submitting the PR

For complex PRs (big refactoring, etc):

\* Squash only the commits with uninteresting changes like typos, docs

improvements, etc… and keep the important and isolated steps in different

commits.

It's important to keep amount of changes in the PR small, since smaller PRs are

easier to review and merging them is quicker. PR diff shouldn't exceed `300`

changed lines, unless it has to.

## How to open a pull request

1. Fork the qTox repository on Github to your existing account.

2. Open a Terminal and do the following steps:

```bash

# Go to a directory of your choice, where the qTox directory will be created:

cd /to/the/directory

# Clone the forked repo:

git clone git@github.com:<YOUR\_USER>/qTox.git

# Add the "upstream" remote to be able to fetch from the qTox upstream repository:

git remote add upstream https://github.com/qTox/qTox.git

# Fetch from the "upstream" repository

git fetch upstream

# Point the local "master" branch to the "upstream" repository

git branch master --set-upstream-to=upstream/master

```

You're now all set to create your first pull request! Hooray! :)

Still in Terminal, do the following steps to actually create the pull request:

```bash

# Fetch from the "upstream" repository:

git fetch upstream master:master

# Checkout a local branch on up-to-date "master" and give it a sane name, e.g.:

git checkout -b feat/brandnew-feature master

```

Now do your changes and commit them by your heart's desire. When you think

you're ready to push for the first time, do the following:

```bash

# Push to the new upstream branch and link it for synchronization

git push -u origin feat/brandnew-feature

# From now on, you can simply…

git push

# ...to your brand new pull request.

```

That's it! Happy contributing!

<a name="merge-conflicts" />

## How to deal with large amounts of merge conflicts

Usually you want to avoid conflicts and they should be rare. If conflicts

appear anyway, they are usually easy enough to solve quickly and safely.

However, if you find yourself in a situation with large amounts of merge

conflicts, this is an indication that you're doing something wrong and you

should change your strategy. Still… you probably don't want to throw away and

lose all your valuable work. So don't worry, there's a way to get out of that

mess. The basic idea is to divide the conflicts into smaller – easier to solve

– chunks and probably several (topic) branches. Here's a little "Rule of Thumb"

list to get out of it:

1. Split your commit history into topic related chunks (by

rebasing/cherry-picking "good" commits).

2. Split "API" and "UI" (widget related) changes into separate commits.

3. Probably split PR into several smaller ones.

In addition it helps to regularly keep rebasing on the upstream repository's

recent master branch. If you don't have the upstream remote in your repo, add

it as described in [How to open a pull request](#how-to-open-a-pull-request).

~~~bash

# If not on PR branch, check it out:

git checkout my/pr-branch

# Now fetch master ALWAYS from upstream repo

git fetch upstream master:master

# Last, rebase PR branch onto master…

git rebase -i master

# …and, if everything's clear, force push to YOUR repo (your "origin" Git remote)

git push -f

~~~

## Good to know

\* \*\*Search\*\* the pull request history! Others might have already implemented

your idea and it could be waiting to be merged (or have been rejected

already). Save your precious time by doing a search first.

\* When resolving merge conflicts, do `git rebase <target\_branch\_name>`, don't

do `git pull`. Then you can start fixing the conflicts. [Here is a good

explanation](https://www.atlassian.com/git/tutorials/merging-vs-rebasing).

<a name="commit" />

## Git Commit 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 qTox change log\*\* using

[clog-cli](https://github.com/clog-tool/clog-cli).

### Commit Message Format

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

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

```

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

<BLANK LINE>

<body>

```

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

header is also optional.

#### Header

The header must be a short (72 characters or less) summary of the changes made.

#### 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, etc), but change the style to a more appropriate one

\* \*\*refactor\*\*: A code change that only improves code readability and reduces

complexity, without changing any functionality

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

\* \*\*revert\*\*: Reverts a previous commit

\* \*\*test\*\*: Adding missing tests

\* \*\*chore\*\*: Changes to the build process or auxiliary tools and libraries such

as documentation generation

##### 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.

#### Scope

The scope could be anything specifying place of the commit change. Note that

"place" doesn't necessarily mean location in source code.

For example:

\* `audio` – change affects audio

\* `video` – change affects video

\* `settings` – change affects qTox settings

\* `chatform`

\* `tray` – change affects tray icon

\* `l10n` – translation update

\* `i18n` – something has been made translatable

\* `build` – change affects build system / scripts, e.g. `CMakeLists.txt`,

`simple\_make.sh`, etc.

\* `travis` – change affects Travis CI

\* `CONTRIBUTING` – change to the contributing guidelines

Since people were abusing length of the scope, it's limited to 12 characters.

If you're running into the limit, you're doing it wrong.

#### 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

A properly formed git commit subject line should always be able to complete the

following sentence:

> If applied, this commit will \_\_\_your subject line here\_\_\_

### Body

Wrap the body at 72 characters whenever possible (for example, don't modify

long links to follow this rule). 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.

The body contains (in order of appearance):

\* A detailed \*\*description\*\* of the committed changes.

\* References to GitHub issues that the commit \*\*closes\*\* (e.g., `Closes #000`

or `Fixes #000`).

\* Any \*\*breaking changes\*\*.

Include every section of the body that is relevant for your commit.

\*\*Breaking changes\*\* should start with the phrase `BREAKING CHANGE:` with a

space or two newlines. The rest of the commit message is then used for this.

## Reviewing

Currently `reviewable.io` is being used to review changes that land in qTox.

How to review:

1. Click on the `Reviewable` button in [pull request].

2. Once Reviewable opens, comment on the lines that need changes.

3. Mark as reviewed only those files that don't require any changes – this

makes it easier to see which files need to be changed & reviewed again once

change is made.

4. If pull request is good to be merged, press `LGTM` button in Reviewable.

5. Once you're done with evaluating PR, press `Publish` to make comments

visible on GitHub.

When responding to review:

1. Click on the `Reviewable` button in [pull request].

2. Once you push changes to the pull request, make drafts of responses to the

change requests.

- if you're just informing that you've made a requested change, use

`Reviewable`'s provided `Done` button.

- if you want discuss the change, write a response draft.

3. When discussion points are addressed, press `Publish` button to make

response visible on GitHub.

Note:

\* when no one is assigned to the PR, \*anyone\* can review it

\* when there are assigned people, only they can mark review as passed

### Testing PRs

The easiest way is to use [`test-pr.sh`] script to get PR merged on top of

current `master`. E.g. to get pull request `#1234`:

```bash

./test-pr.sh 1234

```

That should create branches named `1234` and `test1234`. `test1234` is what you

would want to test. If script fails to merge branch because of conflicts, fret

not, it doesn't need testing until PR author fixes merge conflicts. You might

want to leave a comment on the PR saying that it needs a rebase :smile:

As for testing itself, there's a nice entry on the wiki:

https://github.com/qTox/qTox/wiki/Testing

## Git config

\*Not a requirement, just a friendly tip. :wink:\*

It's nice when commits and tags are being GPG-signed. Github has a few articles

about configuring & signing.

https://help.github.com/articles/signing-commits-using-gpg/

And \*tl;dr\* version:

```sh

gpg --gen-key

gpg --send-keys <your generated key ID>

git config --local commit.gpgsign true

# also force signing tags

git config --local tag.forceSignAnnotated true

```

# Coding Guidelines

See [coding\_standards.md].

## Limitations

### Filesystem

Windows' unbeaten beauty and clarity:

https://msdn.microsoft.com/en-us/library/windows/desktop/aa365247%28v=vs.85%29.aspx

Symbols that should be forbidden for filenames under Windows:

`<` `>` `:` `"` `/` `\` `|` `?` `\*`

[pull request]: https://github.com/qTox/qTox/pulls

[`test-pr.sh`]: /test-pr.sh

[coding\_standards.md]: /doc/coding\_standards.md