# Contributing to TimescaleDB

We appreciate any help the community can provide to make TimescaleDB better!

You can help in different ways:

\* Open an [issue](https://github.com/timescale/timescaledb/issues) with a

bug report, build issue, feature request, suggestion, etc.

\* Fork this repository and submit a pull request

For any particular improvement you want to make, it can be beneficial to

begin discussion on the GitHub issues page. This is the best place to

discuss your proposed improvement (and its implementation) with the core

development team.

Before we accept any code contributions, Timescale contributors need to

sign the [Contributor License Agreement](https://cla-assistant.io/timescale/timescaledb) (CLA). By signing a CLA, we can

ensure that the community is free and confident in its ability to use your

contributions.

## Getting and building TimescaleDB

Please follow our README for [instructions on installing from source](https://github.com/timescale/timescaledb/blob/master/README.md#option-3---from-source).

## Style guide

Before submitting any contributions, please ensure that it adheres to

our [Style Guide](docs/StyleGuide.md).

## Code review workflow

\* Sign the [Contributor License Agreement](https://cla-assistant.io/timescale/timescaledb) (CLA) if you're a new contributor.

\* Develop on your local branch:

\* Fork the repository and create a local feature branch to do work on,

ideally on one thing at a time. Don't mix bug fixes with unrelated

feature enhancements or stylistical changes.

\* Hack away. Add tests for non-trivial changes.

\* Run the [test suite](#testing) and make sure everything passes.

\* When committing, be sure to write good commit messages. Stylistically,

we use commit message titles in the imperative tense, e.g., `Add

merge-append query optimization for time aggregate`. In the case of

non-trivial changes, include a longer description in the commit message

body explaining and detailing the changes. That is, a commit message

should have a short title, followed by a empty line, and then

followed by the longer description.

\* Push your changes to an upstream branch:

\* Make sure that each commit in the pull request will represent a

logical change to the code, will compile, and will pass tests.

\* Rebase your local feature branch against master (`git fetch origin`,

then `git rebase origin/master`) to make sure you're

submitting your changes on top of the newest version of our code.

\* When finalizing your PR (i.e., it has been approved for merging),

aim for the fewest number of commits that

make sense. That is, squash any "fix up" commits into the commit they

fix rather than keep them separate. Each commit should represent a

clean, logical change and include a descriptive commit message.

\* Push your commit to your upstream feature branch: `git push -u <yourfork> my-feature-branch`

\* Create and manage pull request:

\* [Create a pull request using GitHub](https://help.github.com/articles/creating-a-pull-request).

If you know a core developer well suited to reviewing your pull

request, either mention them (preferably by GitHub name) in the PR's

body or [assign them as a reviewer](https://help.github.com/articles/assigning-issues-and-pull-requests-to-other-github-users/).

\* If you get a test failure in Travis CI, check them in the [Travis CI

build log](https://travis-ci.org/timescale/timescaledb).

\* Address feedback by amending your commit(s). If your change contains

multiple commits, address each piece of feedback by amending that

commit to which the particular feedback is aimed.

\* The PR is marked as accepted when the reviewer thinks it's ready to be

merged. Most new contributors aren't allowed to merge themselves; in

that case, we'll do it for you.

## Testing

Every non-trivial change to the code base should be accompanied by a

relevant addition to or modification of the test suite.

Please check that the full test suite (including your test additions

or changes) passes successfully on your local machine \*\*before you

open a pull request\*\*.

If you are running locally:

```bash

# Use Debug build mode for full battery of tests

./bootstrap -DCMAKE\_BUILD\_TYPE=Debug

cd build && make

make installcheck

```

All submitted pull requests are also automatically

run against our test suite via [Travis CI](https://travis-ci.org/timescale/timescaledb)

(that link shows the latest build status of the repository).

## Advanced Topics

### Testing on Windows

Currently our CI infrastructure only ensures that TimescaleDB builds on

Windows, but does not run regression tests due to differences between

Unix-based systems and Windows. We do run these tests before releases

manually, and it would be a bonus if you could test at least non-trivial

contributions for Windows. This involves setting up a remote Windows machine

with TimescaleDB and a Unix-based (e.g., macOS or Linux) machine

to serve as the client. To set up the Windows machine, build from source:

```bash

./bootstrap -DCMAKE\_BUILD\_TYPE=Debug

cmake --build ./build --config Debug

cmake --build ./build --config Debug --target install

```

Then on the client machine:

```bash

./bootstrap -DCMAKE\_BUILD\_TYPE=Debug -DTEST\_PGHOST=ip\_addr\_of\_Win\_machine

cd build && make

make installchecklocal

```