# Contributing to Serenity

When contributing to the Serenity operating system, please be sure that the change(s) you wish to make are in line with the project vision. If you are unsure about this, open an issue first, so we can discuss it. If you are already confident that it's a good fit, you can proceed directly to a Pull Request.

Everyone is welcome to work on the project, and while we have lots of fun, it's a serious kind of fun. :^)

## Communication

The easiest way to get in touch is by joining the `#serenityos` channel on the Freenode IRC network.

## Issue policy

Unlike many other software projects, SerenityOS is not concerned with gaining the largest possible userbase. Its target audience is its own developers. As such, we have limited interest in feature requests from non-contributors.

That said, please do file any bugs you find, keeping the following in mind:

\* One issue per bug. Putting multiple things in the same issue makes both discussion and completion unnecessarily complicated.

\* No build issues (or other support requests.) If the Travis CI is green, the build problem is most likely on your side. Work it out locally, or ask on IRC.

## Feature policy

As a labor of love, Serenity is strongly user-focused. When proposing or building a new feature, you should start by imagining how a person would use it. Like a tree falling in the woods with nobody around, does a feature with no user impact really provide value?

Maybe your feature involves a GUI application, or maybe it's more suited to a CLI program. Either way, a useful exercise is to imagine what the user interface would be for your idea. Let's say you want to implement a fan speed monitor in the kernel because you recently read about how the related hardware interfaces work - rather than diving right into the kernel implementation, it would be good to imagine how that might be exposed to a user. You might want to build a simple taskbar widget, or maybe a command-line program, or even something as simple as a `/proc` interface, and fill it with mock data to begin with. Whatever your interface is, it should "fit" with the rest of the system, and it should provide some utility to a user.

Sometimes there is no obvious user-facing interface for a feature. In those cases you should still think about the impact that your feature or change will have on a user. If possible, you should devise a way to measure your feature - maybe it reduces memory consumption, maybe it results in fewer page faults, maybe it makes more efficient use of network resources. Whatever that metric is, try to capture it, and provide a summary in your proposal or pull request. If your numbers win, you have a good chance of getting your feature into the system. If your numbers don't win, you have valuable data that you can use to improve your feature.

If you can think of no user-facing interface for your feature, and no user impact at all, please do feel free to start a discussion about it. But please don't spend your precious time building a feature or implementing a change if there's no way for someone to use it and no way to quantify the overall improvement.

## Code submission policy

Nobody is perfect, and sometimes we mess things up. That said, here are some good dos & dont's to try and stick to:

\*\*Do:\*\*

\* Write in idiomatic Serenity C++20, using the `AK` containers in all code.

\* Conform to the project coding style found in [CodingStyle.md](https://github.com/SerenityOS/serenity/blob/master/Documentation/CodingStyle.md). Please use `clang-format` (version 10 or later) to automatically format C++ files.

\* Choose expressive variable, function and class names. Make it as obvious as possible what the code is doing.

\* Split your changes into separate, atomic commits.

\* Make sure your commits are rebased on the master branch.

\* Wrap your commit messages at 72 characters.

\* The first line of the commit message should have the format "Category: Brief description of what's being changed". The "category" can be a subdirectory, but also something like "POSIX compliance" or "ClassName". Whatever seems logical.

\* Write your commit messages in proper English, with care and punctuation.

\* Squash your commits when making revisions after a patch review.

\*\*Don't:\*\*

\* Submit code that's incompatible with the project licence (2-clause BSD.)

\* Touch anything outside the stated scope of the PR.

\* Iterate excessively on your design across multiple commits.

\* Use weasel-words like "refactor" or "fix" to avoid explaining what's being changed.

\* Include commented-out code.

\* Write in C.