# How to Contribute to Abseil

We'd love to accept your patches and contributions to this project. There are

just a few small guidelines you need to follow.

NOTE: If you are new to GitHub, please start by reading [Pull Request

howto](https://help.github.com/articles/about-pull-requests/)

## Contributor License Agreement

Contributions to this project must be accompanied by a Contributor License

Agreement. You (or your employer) retain the copyright to your contribution,

this simply gives us permission to use and redistribute your contributions as

part of the project. Head over to <https://cla.developers.google.com/> to see

your current agreements on file or to sign a new one.

You generally only need to submit a CLA once, so if you've already submitted one

(even if it was for a different project), you probably don't need to do it

again.

## Contribution Guidelines

Potential contributors sometimes ask us if the Abseil project is the appropriate

home for their utility library code or for specific functions implementing

missing portions of the standard. Often, the answer to this question is "no".

We’d like to articulate our thinking on this issue so that our choices can be

understood by everyone and so that contributors can have a better intuition

about whether Abseil might be interested in adopting a new library.

### Priorities

Although our mission is to augment the C++ standard library, our goal is not to

provide a full forward-compatible implementation of the latest standard. For us

to consider a library for inclusion in Abseil, it is not enough that a library

is useful. We generally choose to release a library when it meets at least one

of the following criteria:

\* \*\*Widespread usage\*\* - Using our internal codebase to help gauge usage, most

of the libraries we've released have tens of thousands of users.

\* \*\*Anticipated widespread usage\*\* - Pre-adoption of some standard-compliant

APIs may not have broad adoption initially but can be expected to pick up

usage when it replaces legacy APIs. `absl::from\_chars`, for example,

replaces existing code that converts strings to numbers and will therefore

likely see usage growth.

\* \*\*High impact\*\* - APIs that provide a key solution to a specific problem,

such as `absl::FixedArray`, have higher impact than usage numbers may signal

and are released because of their importance.

\* \*\*Direct support for a library that falls under one of the above\*\* - When we

want access to a smaller library as an implementation detail for a

higher-priority library we plan to release, we may release it, as we did

with portions of `absl/meta/type\_traits.h`. One consequence of this is that

the presence of a library in Abseil does not necessarily mean that other

similar libraries would be a high priority.

### API Freeze Consequences

Via the

[Abseil Compatibility Guidelines](https://abseil.io/about/compatibility), we

have promised a large degree of API stability. In particular, we will not make

backward-incompatible changes to released APIs without also shipping a tool or

process that can upgrade our users' code. We are not yet at the point of easily

releasing such tools. Therefore, at this time, shipping a library establishes an

API contract which is borderline unchangeable. (We can add new functionality,

but we cannot easily change existing behavior.) This constraint forces us to

very carefully review all APIs that we ship.

## Coding Style

To keep the source consistent, readable, diffable and easy to merge, we use a

fairly rigid coding style, as defined by the

[google-styleguide](https://github.com/google/styleguide) project. All patches

will be expected to conform to the style outlined

[here](https://google.github.io/styleguide/cppguide.html).

## Guidelines for Pull Requests

\* If you are a Googler, it is preferable to first create an internal CL and

have it reviewed and submitted. The code propagation process will deliver

the change to GitHub.

\* Create \*\*small PRs\*\* that are narrowly focused on \*\*addressing a single

concern\*\*. We often receive PRs that are trying to fix several things at a

time, but if only one fix is considered acceptable, nothing gets merged and

both author's & review's time is wasted. Create more PRs to address

different concerns and everyone will be happy.

\* For speculative changes, consider opening an [Abseil

issue](https://github.com/abseil/abseil-cpp/issues) and discussing it first.

If you are suggesting a behavioral or API change, consider starting with an

[Abseil proposal template](ABSEIL\_ISSUE\_TEMPLATE.md).

\* Provide a good \*\*PR description\*\* as a record of \*\*what\*\* change is being

made and \*\*why\*\* it was made. Link to a GitHub issue if it exists.

\* Don't fix code style and formatting unless you are already changing that

line to address an issue. Formatting of modified lines may be done using

`git clang-format`. PRs with irrelevant changes won't be merged. If

you do want to fix formatting or style, do that in a separate PR.

\* Unless your PR is trivial, you should expect there will be reviewer comments

that you'll need to address before merging. We expect you to be reasonably

responsive to those comments, otherwise the PR will be closed after 2-3

weeks of inactivity.

\* Maintain \*\*clean commit history\*\* and use \*\*meaningful commit messages\*\*.

PRs with messy commit history are difficult to review and won't be merged.

Use `rebase -i upstream/master` to curate your commit history and/or to

bring in latest changes from master (but avoid rebasing in the middle of a

code review).

\* Keep your PR up to date with upstream/master (if there are merge conflicts,

we can't really merge your change).

\* \*\*All tests need to be passing\*\* before your change can be merged. We

recommend you \*\*run tests locally\*\* (see below)

\* Exceptions to the rules can be made if there's a compelling reason for doing

so. That is - the rules are here to serve us, not the other way around, and

the rules need to be serving their intended purpose to be valuable.

\* All submissions, including submissions by project members, require review.

## Running Tests

If you have [Bazel](https://bazel.build/) installed, use `bazel test

--test\_tag\_filters="-benchmark" ...` to run the unit tests.

If you are running the Linux operating system and have

[Docker](https://www.docker.com/) installed, you can also run the `linux\_\*.sh`

scripts under the `ci/`(https://github.com/abseil/abseil-cpp/tree/master/ci)

directory to test Abseil under a variety of conditions.

## Abseil Committers

The current members of the Abseil engineering team are the only committers at

present.

## Release Process

Abseil lives at head, where latest-and-greatest code can be found.