# How to contribute

We definitely welcome your patches and contributions to gRPC! Please read the gRPC

organization's [governance rules](https://github.com/grpc/grpc-community/blob/master/governance.md)

and [contribution guidelines](https://github.com/grpc/grpc-community/blob/master/CONTRIBUTING.md) before proceeding.

If you are new to github, please start by reading [Pull Request

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

If you are looking for features to work on, please filter the issues list with the label ["disposition/help wanted"](https://github.com/grpc/grpc/issues?q=label%3A%22disposition%2Fhelp+wanted%22).

Please note that some of these feature requests might have been closed in the past as a result of them being marked as stale due to there being no activity, but these are still valid feature requests.

## Legal requirements

In order to protect both you and ourselves, you will need to sign the

[Contributor License

Agreement](https://identity.linuxfoundation.org/projects/cncf).

## Cloning the repository

Before starting any development work you will need a local copy of the gRPC repository.

Please follow the instructions in [Building gRPC C++: Clone the repository](BUILDING.md#clone-the-repository-including-submodules).

## Building & Running tests

Different languages use different build systems. To hide the complexity

of needing to build with many different build systems, a portable python

script that unifies the experience of building and testing gRPC in different

languages and on different platforms is provided.

To build gRPC in the language of choice (e.g. `c++`, `csharp`, `php`, `python`, `ruby`, ...)

- Prepare your development environment based on language-specific instructions in `src/YOUR-LANGUAGE` directory.

- The language-specific instructions might involve installing C/C++ prerequisites listed in

[Building gRPC C++: Prerequisites](BUILDING.md#pre-requisites). This is because gRPC implementations

in this repository are using the native gRPC "core" library internally.

- Run

```

python tools/run\_tests/run\_tests.py -l YOUR\_LANGUAGE --build\_only

```

- To also run all the unit tests after building

```

python tools/run\_tests/run\_tests.py -l YOUR\_LANGUAGE

```

You can also run `python tools/run\_tests/run\_tests.py --help` to discover useful command line flags supported. For more details,

see [tools/run\_tests](tools/run\_tests) where you will also find guidance on how to run various other test suites (e.g. interop tests, benchmarks).

## Generated project files

To ease maintenance of language- and platform- specific build systems, many

projects files are generated using templates and should not be edited by hand.

Run `tools/buildgen/generate\_projects.sh` to regenerate. See

[templates](templates) for details.

As a rule of thumb, if you see the "sanity tests" failing you've most likely

edited generated files or you didn't regenerate the projects properly (or your

code formatting doesn't match our code style).

## Guidelines for Pull Requests

How to get your contributions merged smoothly and quickly.

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

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

at a time, but 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 issue and discussing it first.

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

[gRFC proposal](https://github.com/grpc/proposal).

- 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. PRs with irrelevant changes won't be merged. If you do

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

- If you are adding a new file, make sure it has the copyright message template

at the top as a comment. You can copy over the message from an existing file

and update the year.

- 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.

- If you have non-trivial contributions, please consider adding an entry to [the

AUTHORS file](https://github.com/grpc/grpc/blob/master/AUTHORS) listing the

copyright holder for the contribution (yourself, if you are signing the

individual CLA, or your company, for corporate CLAs) in the same PR as your

contribution. This needs to be done only once, for each company, or

individual. Please keep this file in alphabetical order.

- 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).

- If you are regenerating the projects using

`tools/buildgen/generate\_projects.sh`, make changes to generated files a

separate commit with commit message `regenerate projects`. Mixing changes

to generated and hand-written files make your PR difficult to review.

Note that running this script requires the installation of Python packages

`pyyaml` and `mako` (typically installed using `pip`) as well as a recent

version of [`go`](https://golang.org/doc/install#install).

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

We recommend you \*\*run tests locally\*\* before creating your PR to catch

breakages early on (see [tools/run\_tests](tools/run\_tests). Ultimately, the

green signal will be provided by our testing infrastructure. The reviewer

will help you if there are test failures that seem not related to the change

you are making.

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

so.

## Obtaining Commit Access

We grant Commit Access to contributors based on the following criteria:

\* Sustained contribution to the gRPC project.

\* Deep understanding of the areas contributed to, and good consideration of various reliability, usability and performance tradeoffs.

\* Contributions demonstrate that obtaining Commit Access will significantly reduce friction for the contributors or others.

In addition to submitting PRs, a Contributor with Commit Access can:

\* Review PRs and merge once other checks and criteria pass.

\* Triage bugs and PRs and assign appropriate labels and reviewers.

### Obtaining Commit Access without Code Contributions

The [gRPC organization](https://github.com/grpc) is comprised of multiple repositories and commit access is usually restricted to one or more of these repositories. Some repositories such as the [grpc.github.io](https://github.com/grpc/grpc.github.io/) do not have code, but the same principle of sustained, high quality contributions, with a good understanding of the fundamentals, apply.