# Contributor's Guide

Welcome and thank you for your interest in contributing to the QuantConnect Lean open source project. This document aims to describe the preferred workflow contributors should follow when contributing new source code to the project. This Git workflow is inspired greatly by the [irON-Parsers Contributors Guide](https://github.com/structureddynamics/irON-Parsers/wiki/Collaboration%3A-git-development-workflow).

# Contributing

## Who is a Collaborator?

A collaborator is someone with write access to the QuantConnect Lean repository. Collaborators merge pull requests from contributors.

## Who is a Contributor?

A contributor can be anyone! It could be you. Continue reading this section if you wish to get involved and contribute back to the QuantConnect Lean open source project!

## Code Style and Testing

Code reviewers will be expecting to see code that follows Microsoft's C# guidelines. There are a few resources available [here](https://msdn.microsoft.com/en-us/library/czefa0ke(VS.71).aspx) and [here](https://msdn.microsoft.com/en-us/library/ff926074.aspx).

As a point of consistency, we use soft tabs of four spaces to ensure files render correctly in everyone's environment/diff tools.

All pull requests must be accompanied by units tests. If it is a new feature, the tests should highlight expected use cases as well as edge cases, if applicable. If it is a bugfix, there should be tests that expose the bug in question.

## Guidelines for Framework Modules Contributions

Contributions of [Algorithm Framework](https://www.quantconnect.com/docs/algorithm-framework/overview) Modules needs to follow certain extra patterns, since QuantConnect users can use them in their algorithms.

Generally modules should do one focused, specific role well. For example, combining risk control logic with [notifications](https://www.quantconnect.com/docs/live-trading/notifications) or placing orders outside execution models violates the general programming rule 'separation of concerns'. Keep each module doing one specific task and if you want to consider additional functionality add event handlers that users can bind to from their Algorithm instance.

By default production code should be silent unless there is a fatal exception. Because of this, [logging or debugging](https://www.quantconnect.com/docs/algorithm-reference/logging-and-debug) is not allowed inside LEAN framework modules. Additional [charting](https://www.quantconnect.com/docs/algorithm-reference/charting) inside the module consumes the resources and should not be included in a module as well.

## Initial Setup

\* Setup a [GitHub](https://github.com/) account

\* [Fork](https://help.github.com/articles/fork-a-repo/) the [repository](https://github.com/QuantConnect/Lean) of the project

\* Clone your fork locally

```bash

$ git clone https://github.com/username/Lean.git

```

\* Navigate to the QuantConnect Lean directory and add the upstream remote

```bash

$ cd Lean

$ git remote add upstream https://github.com/QuantConnect/Lean.git

```

The remote upstream branch links your fork of Lean with our master copy, so when you perform a `git pull --rebase` you'll be getting updates from our repository.

## Keeping your master up-to-date!

Now that you've defined the `remote upstream branch`, you can refresh your local copy of master with the following commands:

```bash

$ git checkout master

$ git pull --rebase

```

This will checkout your local master branch and then merge changes in from the remote upstream branch. We use [rebase](https://www.atlassian.com/git/tutorials/rewriting-history/git-rebase) to reduce noise from merge commits.

# Branching Model

If unfamiliar with git branches, please read this [short guide on branches](https://www.atlassian.com/git/tutorials/using-branches/).

The following names will be used to differentiate between the different repositories:

\* \*\*upstream\*\* - The 'official' QuantConnect Lean [repository](https://github.com/QuantConnect/Lean.git) (what is on QuantConnect's GitHub account)

\* \*\*origin\*\* - Your fork of the official repository on GitHub (what is on your GitHub account)

\* \*\*local\*\* - This will be your local clone of \*\*origin\*\* (what is on your computer)

As a \*\*contributor\*\* you will push your completed \*\*local\*\* topic branch to \*\*origin\*\*. As a \*\*contributor\*\* you will pull your updates from \*\*upstream\*\*. As a \*\*collaborator\*\* (write-access) you will merge branches from \*\*contributors\*\* into \*\*upstream\*\*.

## Primary Branch

The upstream repository holds a single primary branch that we maintain:

\* \*\*upstream/master\*\* - This is the where main development takes place

## Topic Branches

Topic branches are for contributors to develop bug fixes and new features so that they can be easily merged to \*\*master\*\*. They must follow a few simple rules for consistency:

\* Must branch off from \*\*master\*\*

\* Must be merged back into \*\*master\*\*

\* Consider using the GitHub issue number in the branch name

Topic branches should exist in your \*\*local\*\* and \*\*origin\*\* repositories only. Submitting a pull request will request a merge from your topic branch to our \*\*upstream/master\*\* branch.

## Working on topic branches

First create a new branch for the work you'd like to perform. When naming your branch, please use the following convention: `bug-<issue#>-<description>` or `feature-<issue#>-<description>`:

```bash

$ git checkout -b bug-123-short-issue-description

Switched to a new branch 'bug-123-short-issue-description'

```

Now perform some work and commit changes. Always review your changes before committing

```bash

$ git status

$ git diff

$ git add --all

$ git commit

```

You can push your changes to your fork's master branch using:

```bash

$ git push origin master

```

When committing, be sure to follow [best practices](https://github.com/erlang/otp/wiki/Writing-good-commit-messages) writing good commit descriptions.

After performing some work you'll want to merge in changes from the \*\*upstream/master\*\*. You can use the following two commands in order to assist upstream merging:

```bash

$ git fetch upstream

$ git rebase upstream/master bug-123-short-issue-description

```

The `git fetch upstream` command will download the \*\*upstream\*\* repository to your computer but not merge it. The `rebase upstream/master bug-123-short-issue-description` command will [rebase](https://www.atlassian.com/git/tutorials/rewriting-history/git-commit--amend) your changes on top of \*\*upstream/master\*\*. This will make the review process easier for \*\*collaborators\*\*.

> CAUTION Please note that once you have pushed your branch remotely you MUST NOT rebase!

If you need to merge changes in after pushing your branch to \*\*origin\*\*, use the following:

```bash

$ git pull upstream/master

```

When topic branches are finished and ready for review, they should be pushed back to \*\*origin\*\*.

```bash

$ git push origin bug-123-short-issue-description

To git@github.com:username/Lean.git

\* [new branch] bug-123-short-issue-description -> bug-123-short-issue-description

```

Now you're ready to send a [pull request](https://help.github.com/articles/using-pull-requests/) from this branch to \*\*upstream/master\*\* and update the GitHub issue tracker to let a collaborator know that your branch is ready to be reviewed and merged. If extra changes are required as part of the review process, make those changes on the topic branch and re-push. First re-checkout the topic branch you made your original changes on:

```bash

$ git checkout bug-123-short-issue-description

```

Now make responses to the review comments, commit, and re-push your changes:

```bash

$ git add --all

$ git commit

$ git push

```