# Contributing

Contributions from the community are essential in keeping Hibernate (any Open Source

project really) strong and successful.

# Legal

All original contributions to Hibernate are licensed under the

[GNU Lesser General Public License (LGPL)](https://www.gnu.org/licenses/old-licenses/lgpl-2.1.txt),

version 2.1 or later, or, if another license is specified as governing the file or directory being

modified, such other license. The LGPL text is included verbatim in the [lgpl.txt](lgpl.txt) file

in the root directory of the ORM repository.

All contributions are subject to the [Developer Certificate of Origin (DCO)](https://developercertificate.org/).

The DCO text is also included verbatim in the [dco.txt](dco.txt) file in the root directory of the ORM repository.

## Guidelines

While we try to keep requirements for contributing to a minimum, there are a few guidelines

we ask that you mind.

For code contributions, these guidelines include:

\* respect the project code style - find templates for [Eclipse](https://community.jboss.org/docs/DOC-16649)

and [IntelliJ IDEA](https://community.jboss.org/docs/DOC-15468)

\* have a corresponding JIRA issue and the key for this JIRA issue should be used in the commit message

\* have a set of appropriate tests. For bug reports, the tests reproduce the initial reported bug

and illustrates that the solution actually fixes the bug. For features/enhancements, the

tests illustrate the feature working as intended. In both cases the tests are incorporated into

the project to protect against regressions.

\* if applicable, documentation is updated to reflect the introduced changes

\* the code compiles and the tests pass (`./gradlew clean build`)

For documentation contributions, mainly just respect the project code style, especially in regards

to use of tabs - as mentioned above, code style templates are available for both Eclipse and IntelliJ

IDEA IDEs. Ideally these contributions would also have a corresponding JIRA issue, although this

is less necessary for documentation contributions.

## Getting Started

If you are just getting started with Git, GitHub and/or contributing to Hibernate via

GitHub there are a few pre-requisite steps to follow:

\* Make sure you have a [Hibernate JIRA account](https://hibernate.atlassian.net)

\* Make sure you have a [GitHub account](https://github.com/signup/free)

\* [Fork](https://help.github.com/articles/fork-a-repo) the Hibernate repository. As discussed in

the linked page, this also includes:

\* [Set](https://help.github.com/articles/set-up-git) up your local git install

\* Clone your fork

\* See the wiki pages for setting up your IDE, whether you use

[IntelliJ IDEA](https://community.jboss.org/wiki/ContributingToHibernateUsingIntelliJ)

or [Eclipse](https://community.jboss.org/wiki/ContributingToHibernateUsingEclipse)<sup>(1)</sup>.

## Create the working (topic) branch

Create a [topic branch](http://git-scm.com/book/en/Git-Branching-Branching-Workflows#Topic-Branches)

on which you will work. The convention is to incorporate the JIRA issue key in the name of this branch,

although this is more of a mnemonic strategy than a hard-and-fast rule - but doing so helps:

\* remember what each branch is for

\* isolate the work from other contributions you may be working on.

\_If there is not already a JIRA issue covering the work you want to do, create one.\_

Assuming you will be working from the master branch and working

on the JIRA HHH-123 : `git checkout -b HHH-123 master`

## Code

Do yo thing!

## Commit

\* Make commits of logical units.

\* Be sure to use the JIRA issue key in the commit message. This is how JIRA will pick

up the related commits and display them on the JIRA issue.

\* Make sure you have added the necessary tests for your changes.

\* Run \_all\_ the tests to assure nothing else was accidentally broken.

\* Make sure your source does not violate the checkstyles.

\_Prior to committing, if you want to pull in the latest upstream changes (highly

appreciated btw), please use rebasing rather than merging. Merging creates

"merge commits" that really muck up the project timeline.\_

## Submit

\* Push your changes to the topic branch in your fork of the repository.

\* Initiate a [pull request](http://help.github.com/articles/creating-a-pull-request)

\* Update the JIRA issue, adding a comment including a link to the created pull request

\_if the JIRA key was not used in the commit message\_.

It is important that this topic branch on your fork:

\* be isolated to just the work on this one JIRA issue, or multiple issues if they are

related and also fixed/implemented by this work. The main point is to not push

commits for more than one PR to a single branch - GitHub PRs are linked to

a branch rather than specific commits.

\* remain until the PR is closed. Once the underlying branch is deleted the corresponding

PR will be closed, if not already, and the changes will be lost.

# Notes

<sup>(1)</sup> Gradle `eclipse` plugin is no longer supported, so the recommended way to import the project in your IDE is with the proper IDE tools/plugins. Don't try to run `./gradlew clean eclipse --refresh-dependencies` from the command line as you'll get an error because `eclipse` no longer exists