# Contributing to Shoulda Matchers

Although we are always happy to make improvements to Shoulda Matchers, we also

welcome changes and improvements from the community!

Have a fix for a problem you've been running into or an idea for a new feature

you think would be useful? Here's what you need to do:

1. [Read and understand the Code of Conduct](#code-of-conduct).

1. Fork this repo and clone your fork to somewhere on your machine.

1. [Ensure that you have a working environment](#setting-up-your-environment).

1. Read up on the [architecture of the gem](#architecture), [how to run

tests](#running-tests), and [the code style we use in this

project](#code-style).

1. Cut a new branch and write a failing test for the feature or bugfix you plan

on implementing.

1. [Make sure your branch is well managed as you go

along](#managing-your-branch).

1. [Update the inline documentation if you're making a change to the

API](#documentation).

1. [Refrain from updating the changelog.](#a-word-on-the-changelog)

1. Push to your fork and submit a pull request.

1. [Ensure that the test suite passes on Travis and make any necessary changes

to your branch to bring it to green.](#continuous-integration)

Although we maintain Shoulda Matchers in our free time, we try to respond to

contributions in a timely manner. Once we look at your pull request, we may give

you feedback. For instance, we may suggest some changes to make to your code to

fit within the project style or discuss alternate ways of addressing the issue

in question. Assuming we're happy with everything, we'll then bring your changes

into master. Now you're a contributor!

---

## Code of Conduct

If this is your first time contributing, please read the [Code of Conduct]. We

want to create a space in which everyone is allowed to contribute, and we

enforce the policies outline in this document.

[Code of Conduct]: https://thoughtbot.com/open-source-code-of-conduct

## Setting up your environment

The setup script will install all dependencies necessary for working on the

project:

```bash

bin/setup

```

## Architecture

This project follows the typical structure for a gem: code is located in `lib`

and tests are in `spec`.

### Matchers

All of the matchers are broken up by the type of example group they apply to:

\* `{lib,spec/unit}/shoulda/matchers/action\_controller\*` for ActionController

matchers

\* `{lib,spec/unit}/shoulda/matchers/active\_model\*` for ActiveModel matchers

\* `{lib,spec/unit}/shoulda/matchers/active\_record\*` for ActiveRecord matchers

\* `{lib,spec/unit}/shoulda/matchers/independent\*` for matchers that can be used

in any example group

There are other files in the project, of course, but these are likely the ones

you'll be most interested in.

### Tests

In addition, tests are broken up into two categories:

\* `spec/unit` — low-level tests for individual matchers (you're probably

interested in these)

\* `spec/acceptance` — high-level tests to ensure that the gem works in Rails

projects, plain Ruby projects, etc. (these do not need to get updated often)

A word about the tests, by the way: they're admittedly the most complicated part

of this gem, and there are a few different strategies that we've introduced at

various points in time to set up objects for tests across all specs, some of

which are old and some of which are new. The best approach for writing tests is

probably to copy an existing test in the same file as where you want to add a

new test.

## Code style

We follow a derivative of the [unofficial Ruby style guide] created by the

Rubocop developers. You can view our Rubocop configuration [here], but here are

some key differences:

\* Use single quotes for strings.

\* When breaking up methods across multiple lines, place the `.` at the end of

the line instead of the beginning.

\* Don't use conditional modifiers (i.e. `x if y`); place the beginning and

ending of conditionals on their own lines.

\* Use an 80-character line-length except for `describe`, `context`, `it`, and

`specify` lines in tests.

\* For arrays, hashes, and method arguments that span multiple lines, place a

trailing comma at the end of the last item.

\* Collection methods are spelled `detect`, `inject`, `map`, and `select`.

[unofficial Ruby style guide]: https://github.com/rubocop-hq/ruby-style-guide

[here]: .rubocop.yml

## Running tests

### A word about Appraisals

Because the gem supports multiple Ruby and Rails versions, we have to be able

to test against them as well.

Oftentimes it is enough to use whatever Ruby version you have on your computer,

but if you are fixing a bug that targets a specific Ruby version, you'll want to

make sure to switch to that using your Ruby manager of choice before you run any

tests.

If you are fixing a bug for a specific \_Rails\_ version, then you'll want to use

a specific "appraisal" when running a test. [Appraisals] are Gemfiles centered

around a Rails version. What this means is that when running tests, you have to

specify which one to use. You do this by using this command:

```bash

bundle exec appraisal <appraisal name> ...

```

You can find the available list of appraisals by running:

```bash

bundle exec appraisal list

```

[Appraisals]: https://github.com/thoughtbot/appraisal

### Unit tests

Unit tests are the most common kind of tests in the gem. They exercise matcher

code file by file in the context of a real Rails application. This application

is created and loaded every time you start running tests.

To run a unit test, you might say something like:

```bash

bundle exec appraisal rails\_5\_2 rspec spec/unit/shoulda/matchers/active\_model/validate\_inclusion\_of\_matcher\_spec.rb

```

### Acceptance tests

The acceptance tests exercise matchers in the context of a real Ruby or Rails

application. Unlike unit tests, this application is set up and torn down for

each test.

To run an acceptance test, you might say something like:

```bash

bundle exec appraisal rails\_5\_2 rspec spec/acceptance/rails\_integration\_spec.rb

```

### All tests

In order to run all of the tests, simply run:

```bash

bundle exec rake

```

## Managing your branch

\* Use well-crafted commit messages, providing context if possible. [Tim Pope's

guide] was a wonderful piece on this topic when it came out and we still find

it to be helpful even today.

\* Squash "WIP" commits and remove merge commits by rebasing your branch against

`master`. We try to keep our commit history as clean as possible.

[Tim Pope's guide]: https://tbaggery.com/2008/04/19/a-note-about-git-commit-messages.html

## Documentation

As you navigate the codebase, you may notice certain classes and methods that

are prefaced with inline documentation. This documentation is written and

generated using [YARD][yard] and is published [online][rubydocs].

[rubydocs]: https://matchers.shoulda.io/docs

[yard]: https://github.com/lsegal/yard

We ensure that the documentation is up to date before we issue a release, but

sometimes we don't catch everything. So if your changes end up extending or

updating the API, it helps us greatly to update the docs at the same time.

## A word on the changelog

You may also notice that we have a changelog in the form of

[CHANGELOG.md](CHANGELOG.md). You may be tempted to include changes to this in

your branch, but don't worry about this — we'll take care of it!

## Continuous integration

While running `bundle exec rake` is a great way to check your work, this command

will only run your tests against the latest supported Ruby and Rails version.

Ultimately, though, you'll want to ensure that your changes work in all possible

environments. We make use of [Travis][travis] to do this work for us. Travis

will kick in after you push up a branch or open a PR. It takes 20-30 minutes to

run a complete build, which you are free to

[monitor as it progresses][shoulda-matchers-on-travis].

[shoulda-matchers-on-travis]: https://travis-ci.org/thoughtbot/shoulda-matchers

What happens if the build fails in some way? Don't fear! Click on a failed job

and scroll through its output to determine the cause of the failure. You'll want

to make changes to your branch and push them up until the entire build is green.

It may take a bit of time, but overall it is worth it and it helps us immensely!

[travis]: https://travis-ci.org/