**Contributing to Ruby on Rails**

This guide covers ways in which *you* can become a part of the ongoing development of Ruby on Rails.

After reading this guide, you will know:

* **How to use GitHub to report issues.**
* **How to clone master and run the test suite.**
* **How to help resolve existing issues.**
* **How to contribute to the Ruby on Rails documentation.**
* **How to contribute to the Ruby on Rails code.**

Ruby on Rails is not "someone else's framework." Over the years, hundreds of people have contributed to Ruby on Rails ranging from a single character to massive architectural changes or significant documentation - all with the goal of making Ruby on Rails better for everyone. Even if you don't feel up to writing code or documentation yet, there are a variety of other ways that you can contribute, from reporting issues to testing patches.

As mentioned in [Rails README](https://github.com/rails/rails/blob/master/README.md), everyone interacting in Rails and its sub-projects' codebases, issue trackers, chat rooms, and mailing lists is expected to follow the Rails [code of conduct](http://rubyonrails.org/conduct/).

**http://guides.rubyonrails.org/images/chapters_icon.gifChapters**

1. [**Reporting an Issue**](http://guides.rubyonrails.org/contributing_to_ruby_on_rails.html#reporting-an-issue)
   * [Creating a Bug Report](http://guides.rubyonrails.org/contributing_to_ruby_on_rails.html#creating-a-bug-report)
   * [Create an Executable Test Case](http://guides.rubyonrails.org/contributing_to_ruby_on_rails.html#create-an-executable-test-case)
   * [Special Treatment for Security Issues](http://guides.rubyonrails.org/contributing_to_ruby_on_rails.html#special-treatment-for-security-issues)
   * [What about Feature Requests?](http://guides.rubyonrails.org/contributing_to_ruby_on_rails.html#what-about-feature-requests-questionmark)
2. [**Helping to Resolve Existing Issues**](http://guides.rubyonrails.org/contributing_to_ruby_on_rails.html#helping-to-resolve-existing-issues)
   * [Verifying Bug Reports](http://guides.rubyonrails.org/contributing_to_ruby_on_rails.html#verifying-bug-reports)
   * [Testing Patches](http://guides.rubyonrails.org/contributing_to_ruby_on_rails.html#testing-patches)
3. [**Contributing to the Rails Documentation**](http://guides.rubyonrails.org/contributing_to_ruby_on_rails.html#contributing-to-the-rails-documentation)
4. [**Translating Rails Guides**](http://guides.rubyonrails.org/contributing_to_ruby_on_rails.html#translating-rails-guides)
5. [**Contributing to the Rails Code**](http://guides.rubyonrails.org/contributing_to_ruby_on_rails.html#contributing-to-the-rails-code)
   * [Setting Up a Development Environment](http://guides.rubyonrails.org/contributing_to_ruby_on_rails.html#setting-up-a-development-environment)
   * [Clone the Rails Repository](http://guides.rubyonrails.org/contributing_to_ruby_on_rails.html#clone-the-rails-repository)
   * [Bundle install](http://guides.rubyonrails.org/contributing_to_ruby_on_rails.html#bundle-install)
   * [Running an Application Against Your Local Branch](http://guides.rubyonrails.org/contributing_to_ruby_on_rails.html#running-an-application-against-your-local-branch)
   * [Write Your Code](http://guides.rubyonrails.org/contributing_to_ruby_on_rails.html#write-your-code)
   * [Benchmark Your Code](http://guides.rubyonrails.org/contributing_to_ruby_on_rails.html#benchmark-your-code)
   * [Running Tests](http://guides.rubyonrails.org/contributing_to_ruby_on_rails.html#running-tests)
   * [Warnings](http://guides.rubyonrails.org/contributing_to_ruby_on_rails.html#warnings)
   * [Updating the CHANGELOG](http://guides.rubyonrails.org/contributing_to_ruby_on_rails.html#updating-the-changelog)
   * [Updating the Gemfile.lock](http://guides.rubyonrails.org/contributing_to_ruby_on_rails.html#updating-the-gemfile-lock)
   * [Sanity Check](http://guides.rubyonrails.org/contributing_to_ruby_on_rails.html#sanity-check)
   * [Commit Your Changes](http://guides.rubyonrails.org/contributing_to_ruby_on_rails.html#commit-your-changes)
   * [Update Your Branch](http://guides.rubyonrails.org/contributing_to_ruby_on_rails.html#update-your-branch)
   * [Fork](http://guides.rubyonrails.org/contributing_to_ruby_on_rails.html#fork)
   * [Issue a Pull Request](http://guides.rubyonrails.org/contributing_to_ruby_on_rails.html#issue-a-pull-request)
   * [Get some Feedback](http://guides.rubyonrails.org/contributing_to_ruby_on_rails.html#get-some-feedback)
   * [Iterate as Necessary](http://guides.rubyonrails.org/contributing_to_ruby_on_rails.html#iterate-as-necessary)
   * [Older Versions of Ruby on Rails](http://guides.rubyonrails.org/contributing_to_ruby_on_rails.html#older-versions-of-ruby-on-rails)
6. [**Rails Contributors**](http://guides.rubyonrails.org/contributing_to_ruby_on_rails.html#rails-contributors)

**1 Reporting an Issue**

Ruby on Rails uses [GitHub Issue Tracking](https://github.com/rails/rails/issues) to track issues (primarily bugs and contributions of new code). If you've found a bug in Ruby on Rails, this is the place to start. You'll need to create a (free) GitHub account in order to submit an issue, to comment on them or to create pull requests.

Bugs in the most recent released version of Ruby on Rails are likely to get the most attention. Also, the Rails core team is always interested in feedback from those who can take the time to test *edge Rails* (the code for the version of Rails that is currently under development). Later in this guide you'll find out how to get edge Rails for testing.

**1.1 Creating a Bug Report**

If you've found a problem in Ruby on Rails which is not a security risk, do a search on GitHub under [Issues](https://github.com/rails/rails/issues) in case it has already been reported. If you are unable to find any open GitHub issues addressing the problem you found, your next step will be to [open a new one](https://github.com/rails/rails/issues/new). (See the next section for reporting security issues.)

Your issue report should contain a title and a clear description of the issue at the bare minimum. You should include as much relevant information as possible and should at least post a code sample that demonstrates the issue. It would be even better if you could include a unit test that shows how the expected behavior is not occurring. Your goal should be to make it easy for yourself - and others - to reproduce the bug and figure out a fix.

Then, don't get your hopes up! Unless you have a "Code Red, Mission Critical, the World is Coming to an End" kind of bug, you're creating this issue report in the hope that others with the same problem will be able to collaborate with you on solving it. Do not expect that the issue report will automatically see any activity or that others will jump to fix it. Creating an issue like this is mostly to help yourself start on the path of fixing the problem and for others to confirm it with an "I'm having this problem too" comment.

**1.2 Create an Executable Test Case**

Having a way to reproduce your issue will be very helpful for others to help confirm, investigate and ultimately fix your issue. You can do this by providing an executable test case. To make this process easier, we have prepared several bug report templates for you to use as a starting point:

* Template for Active Record (models, database) issues: [gem](https://github.com/rails/rails/blob/master/guides/bug_report_templates/active_record_gem.rb) / [master](https://github.com/rails/rails/blob/master/guides/bug_report_templates/active_record_master.rb)
* Template for Action Pack (controllers, routing) issues: [gem](https://github.com/rails/rails/blob/master/guides/bug_report_templates/action_controller_gem.rb) / [master](https://github.com/rails/rails/blob/master/guides/bug_report_templates/action_controller_master.rb)
* Generic template for other issues: [gem](https://github.com/rails/rails/blob/master/guides/bug_report_templates/generic_gem.rb) / [master](https://github.com/rails/rails/blob/master/guides/bug_report_templates/generic_master.rb)

These templates include the boilerplate code to set up a test case against either a released version of Rails (\*\_gem.rb) or edge Rails (\*\_master.rb).

Simply copy the content of the appropriate template into a .rb file and make the necessary changes to demonstrate the issue. You can execute it by running ruby the\_file.rb in your terminal. If all goes well, you should see your test case failing.

You can then share your executable test case as a [gist](https://gist.github.com/), or simply paste the content into the issue description.

**1.3 Special Treatment for Security Issues**

Please do not report security vulnerabilities with public GitHub issue reports. The [Rails security policy page](http://rubyonrails.org/security) details the procedure to follow for security issues.

**1.4 What about Feature Requests?**

Please don't put "feature request" items into GitHub Issues. If there's a new feature that you want to see added to Ruby on Rails, you'll need to write the code yourself - or convince someone else to partner with you to write the code. Later in this guide you'll find detailed instructions for proposing a patch to Ruby on Rails. If you enter a wish list item in GitHub Issues with no code, you can expect it to be marked "invalid" as soon as it's reviewed.

Sometimes, the line between 'bug' and 'feature' is a hard one to draw. Generally, a feature is anything that adds new behavior, while a bug is anything that causes incorrect behavior. Sometimes, the core team will have to make a judgement call. That said, the distinction generally just affects which release your patch will get in to; we love feature submissions! They just won't get backported to maintenance branches.

If you'd like feedback on an idea for a feature before doing the work to make a patch, please send an email to the [rails-core mailing list](https://groups.google.com/forum/?fromgroups#!forum/rubyonrails-core). You might get no response, which means that everyone is indifferent. You might find someone who's also interested in building that feature. You might get a "This won't be accepted." But it's the proper place to discuss new ideas. GitHub Issues are not a particularly good venue for the sometimes long and involved discussions new features require.

**2 Helping to Resolve Existing Issues**

As a next step beyond reporting issues, you can help the core team resolve existing issues. If you check the [issues list](https://github.com/rails/rails/issues) in GitHub Issues, you'll find lots of issues already requiring attention. What can you do for these? Quite a bit, actually:

**2.1 Verifying Bug Reports**

For starters, it helps just to verify bug reports. Can you reproduce the reported issue on your own computer? If so, you can add a comment to the issue saying that you're seeing the same thing.

If an issue is very vague, can you help narrow it down to something more specific? Maybe you can provide additional information to help reproduce a bug, or help by eliminating needless steps that aren't required to demonstrate the problem.

If you find a bug report without a test, it's very useful to contribute a failing test. This is also a great way to get started exploring the source code: looking at the existing test files will teach you how to write more tests. New tests are best contributed in the form of a patch, as explained later on in the "Contributing to the Rails Code" section.

Anything you can do to make bug reports more succinct or easier to reproduce helps folks trying to write code to fix those bugs - whether you end up writing the code yourself or not.

**2.2 Testing Patches**

You can also help out by examining pull requests that have been submitted to Ruby on Rails via GitHub. To apply someone's changes you need first to create a dedicated branch:

|  |
| --- |
| $ git checkout -b testing\_branch |

Then you can use their remote branch to update your codebase. For example, let's say the GitHub user JohnSmith has forked and pushed to a topic branch "orange" located at <https://github.com/JohnSmith/rails>.

|  |
| --- |
| $ git remote add JohnSmith https://github.com/JohnSmith/rails.git  $ git pull JohnSmith orange |

After applying their branch, test it out! Here are some things to think about:

* Does the change actually work?
* Are you happy with the tests? Can you follow what they're testing? Are there any tests missing?
* Does it have the proper documentation coverage? Should documentation elsewhere be updated?
* Do you like the implementation? Can you think of a nicer or faster way to implement a part of their change?

Once you're happy that the pull request contains a good change, comment on the GitHub issue indicating your approval. Your comment should indicate that you like the change and what you like about it. Something like:

*I like the way you've restructured that code in generate\_finder\_sql - much nicer. The tests look good too.*

If your comment simply reads "+1", then odds are that other reviewers aren't going to take it too seriously. Show that you took the time to review the pull request.

**3 Contributing to the Rails Documentation**

Ruby on Rails has two main sets of documentation: the guides, which help you learn about Ruby on Rails, and the API, which serves as a reference.

You can help improve the Rails guides by making them more coherent, consistent or readable, adding missing information, correcting factual errors, fixing typos, or bringing them up to date with the latest edge Rails.

You can either open a pull request to [Rails](http://github.com/rails/rails) or ask the [Rails core team](http://rubyonrails.org/core) for commit access on docrails if you contribute regularly. Please do not open pull requests in docrails, if you'd like to get feedback on your change, ask for it in [Rails](http://github.com/rails/rails) instead.

Docrails is merged with master regularly, so you are effectively editing the Ruby on Rails documentation.

If you are unsure of the documentation changes, you can create an issue in the [Rails](https://github.com/rails/rails/issues) issues tracker on GitHub.

When working with documentation, please take into account the [API Documentation Guidelines](http://guides.rubyonrails.org/api_documentation_guidelines.html) and the [Ruby on Rails Guides Guidelines](http://guides.rubyonrails.org/ruby_on_rails_guides_guidelines.html).

As explained earlier, ordinary code patches should have proper documentation coverage. Docrails is only used for isolated documentation improvements.

To help our CI servers you should add [ci skip] to your documentation commit message to skip build on that commit. Please remember to use it for commits containing only documentation changes.

Docrails has a very strict policy: no code can be touched whatsoever, no matter how trivial or small the change. Only RDoc and guides can be edited via docrails. Also, CHANGELOGs should never be edited in docrails.

**4 Translating Rails Guides**

We are happy to have people volunteer to translate the Rails guides into their own language. If you want to translate the Rails guides in your own language, follows these steps:

* Fork the project (rails/rails).
* Add a source folder for your own language, for example: *guides/source/it-IT* for Italian.
* Copy the contents of *guides/source* into your own language directory and translate them.
* Do NOT translate the HTML files, as they are automatically generated.

To generate the guides in HTML format cd into the *guides* directory then run (eg. for it-IT):

|  |
| --- |
| $ bundle install  $ bundle exec rake guides:generate:html GUIDES\_LANGUAGE=it-IT |

This will generate the guides in an *output* directory.

The instructions are for Rails > 4. The Redcarpet Gem doesn't work with JRuby.

Translation efforts we know about (various versions):

* **Italian**: <https://github.com/rixlabs/docrails>
* **Spanish**: <http://wiki.github.com/gramos/docrails>
* **Polish**: <http://github.com/apohllo/docrails/tree/master>
* **French** : <http://github.com/railsfrance/docrails>
* **Czech** : <https://github.com/rubyonrails-cz/docrails/tree/czech>
* **Turkish** : <https://github.com/ujk/docrails/tree/master>
* **Korean** : <https://github.com/rorlakr/rails-guides>
* **Simplified Chinese** : <https://github.com/ruby-china/guides>
* **Traditional Chinese** : <https://github.com/docrails-tw/guides>
* **Russian** : <https://github.com/morsbox/rusrails>
* **Japanese** : <https://github.com/yasslab/railsguides.jp>

**5 Contributing to the Rails Code**

**5.1 Setting Up a Development Environment**

To move on from submitting bugs to helping resolve existing issues or contributing your own code to Ruby on Rails, you *must* be able to run its test suite. In this section of the guide you'll learn how to setup the tests on your own computer.

**5.1.1 The Easy Way**

The easiest and recommended way to get a development environment ready to hack is to use the [Rails development box](https://github.com/rails/rails-dev-box).

**5.1.2 The Hard Way**

In case you can't use the Rails development box, see [this other guide](http://guides.rubyonrails.org/development_dependencies_install.html).

**5.2 Clone the Rails Repository**

To be able to contribute code, you need to clone the Rails repository:

|  |
| --- |
| $ git clone https://github.com/rails/rails.git |

and create a dedicated branch:

|  |
| --- |
| $ cd rails  $ git checkout -b my\_new\_branch |

It doesn't matter much what name you use, because this branch will only exist on your local computer and your personal repository on GitHub. It won't be part of the Rails Git repository.

**5.3 Bundle install**

Install the required gems.

|  |
| --- |
| $ bundle install |

**5.4 Running an Application Against Your Local Branch**

In case you need a dummy Rails app to test changes, the --dev flag of rails new generates an application that uses your local branch:

|  |
| --- |
| $ cd rails  $ bundle exec rails new ~/my-test-app --dev |

The application generated in ~/my-test-app runs against your local branch and in particular sees any modifications upon server reboot.

**5.5 Write Your Code**

Now get busy and add/edit code. You're on your branch now, so you can write whatever you want (make sure you're on the right branch with git branch -a). But if you're planning to submit your change back for inclusion in Rails, keep a few things in mind:

* Get the code right.
* Use Rails idioms and helpers.
* Include tests that fail without your code, and pass with it.
* Update the (surrounding) documentation, examples elsewhere, and the guides: whatever is affected by your contribution.

Changes that are cosmetic in nature and do not add anything substantial to the stability, functionality, or testability of Rails will generally not be accepted (read more about [our rationales behind this decision](https://github.com/rails/rails/pull/13771#issuecomment-32746700)).

**5.5.1 Follow the Coding Conventions**

Rails follows a simple set of coding style conventions:

* Two spaces, no tabs (for indentation).
* No trailing whitespace. Blank lines should not have any spaces.
* Indent after private/protected.
* Use Ruby >= 1.9 syntax for hashes. Prefer { a: :b } over { :a => :b }.
* Prefer &&/|| over and/or.
* Prefer class << self over self.method for class methods.
* Use my\_method(my\_arg) not my\_method( my\_arg ) or my\_method my\_arg.
* Use a = b and not a=b.
* Use assert\_not methods instead of refute.
* Prefer method { do\_stuff } instead of method{do\_stuff} for single-line blocks.
* Follow the conventions in the source you see used already.

The above are guidelines - please use your best judgment in using them.

**5.6 Benchmark Your Code**

If your change has an impact on the performance of Rails, please use the [benchmark-ips](https://github.com/evanphx/benchmark-ips) gem to provide benchmark results for comparison.

Here's an example of using benchmark-ips:

|  |
| --- |
| require 'benchmark/ips'    Benchmark.ips do |x|    x.report('addition') { 1 + 2 }    x.report('addition with send') { 1.send(:+, 2) }  end |

This will generate a report with the following information:

|  |
| --- |
| Calculating -------------------------------------              addition   132.013k i/100ms    addition with send   125.413k i/100ms  -------------------------------------------------              addition      9.677M (± 1.7%) i/s -     48.449M    addition with send      6.794M (± 1.1%) i/s -     33.987M |

Please see the benchmark/ips [README](https://github.com/evanphx/benchmark-ips/blob/master/README.md) for more information.

**5.7 Running Tests**

It is not customary in Rails to run the full test suite before pushing changes. The railties test suite in particular takes a long time, and even more if the source code is mounted in /vagrant as happens in the recommended workflow with the [rails-dev-box](https://github.com/rails/rails-dev-box).

As a compromise, test what your code obviously affects, and if the change is not in railties, run the whole test suite of the affected component. If all tests are passing, that's enough to propose your contribution. We have [Travis CI](https://travis-ci.org/rails/rails) as a safety net for catching unexpected breakages elsewhere.

**5.7.1 Entire Rails:**

To run all the tests, do:

|  |
| --- |
| $ cd rails  $ bundle exec rake test |

**5.7.2 For a Particular Component**

You can run tests only for a particular component (e.g. Action Pack). For example, to run Action Mailer tests:

|  |
| --- |
| $ cd actionmailer  $ bundle exec rake test |

**5.7.3 Running a Single Test**

You can run a single test through ruby. For instance:

|  |
| --- |
| $ cd actionmailer  $ bundle exec ruby -w -Itest test/mail\_layout\_test.rb -n test\_explicit\_class\_layout |

The -n option allows you to run a single method instead of the whole file.

**5.7.4 Testing Active Record**

First, create the databases you'll need. For MySQL and PostgreSQL, running the SQL statements create database activerecord\_unittest and create database activerecord\_unittest2 is sufficient. This is not necessary for SQLite3.

This is how you run the Active Record test suite only for SQLite3:

|  |
| --- |
| $ cd activerecord  $ bundle exec rake test:sqlite3 |

You can now run the tests as you did for sqlite3. The tasks are respectively:

|  |
| --- |
| test:mysql2  test:postgresql |

Finally,

|  |
| --- |
| $ bundle exec rake test |

will now run the three of them in turn.

You can also run any single test separately:

|  |
| --- |
| $ ARCONN=sqlite3 bundle exec ruby -Itest test/cases/associations/has\_many\_associations\_test.rb |

To run a single test against all adapters, use:

|  |
| --- |
| $ bundle exec rake TEST=test/cases/associations/has\_many\_associations\_test.rb |

You can invoke test\_jdbcmysql, test\_jdbcsqlite3 or test\_jdbcpostgresql also. See the file activerecord/RUNNING\_UNIT\_TESTS.rdoc for information on running more targeted database tests, or the file ci/travis.rb for the test suite run by the continuous integration server.

**5.8 Warnings**

The test suite runs with warnings enabled. Ideally, Ruby on Rails should issue no warnings, but there may be a few, as well as some from third-party libraries. Please ignore (or fix!) them, if any, and submit patches that do not issue new warnings.

If you are sure about what you are doing and would like to have a more clear output, there's a way to override the flag:

|  |
| --- |
| $ RUBYOPT=-W0 bundle exec rake test |

**5.9 Updating the CHANGELOG**

The CHANGELOG is an important part of every release. It keeps the list of changes for every Rails version.

You should add an entry **to the top** of the CHANGELOG of the framework that you modified if you're adding or removing a feature, committing a bug fix or adding deprecation notices. Refactorings and documentation changes generally should not go to the CHANGELOG.

A CHANGELOG entry should summarize what was changed and should end with the author's name. You can use multiple lines if you need more space and you can attach code examples indented with 4 spaces. If a change is related to a specific issue, you should attach the issue's number. Here is an example CHANGELOG entry:

|  |
| --- |
| \*   Summary of a change that briefly describes what was changed. You can use multiple      lines and wrap them at around 80 characters. Code examples are ok, too, if needed:            class Foo            def bar              puts 'baz'            end          end        You can continue after the code example and you can attach issue number. GH#1234        \*Your Name\* |

Your name can be added directly after the last word if there are no code examples or multiple paragraphs. Otherwise, it's best to make a new paragraph.

**5.10 Updating the Gemfile.lock**

Some changes require the dependencies to be upgraded. In these cases make sure you run bundle update to get the right version of the dependency and commit the Gemfile.lock file within your changes.

**5.11 Sanity Check**

You should not be the only person who looks at the code before you submit it. If you know someone else who uses Rails, try asking them if they'll check out your work. If you don't know anyone else using Rails, try hopping into the IRC room or posting about your idea to the rails-core mailing list. Doing this in private before you push a patch out publicly is the "smoke test" for a patch: if you can't convince one other developer of the beauty of your code, you’re unlikely to convince the core team either.

**5.12 Commit Your Changes**

When you're happy with the code on your computer, you need to commit the changes to Git:

|  |
| --- |
| $ git commit -a |

This should fire up your editor to write a commit message. When you have finished, save and close to continue.

A well-formatted and descriptive commit message is very helpful to others for understanding why the change was made, so please take the time to write it.

A good commit message looks like this:

|  |
| --- |
| Short summary (ideally 50 characters or less)    More detailed description, if necessary. It should be wrapped to  72 characters. Try to be as descriptive as you can. Even if you  think that the commit content is obvious, it may not be obvious  to others. Add any description that is already present in the  relevant issues; it should not be necessary to visit a webpage  to check the history.    The description section can have multiple paragraphs.    Code examples can be embedded by indenting them with 4 spaces:        class ArticlesController        def index          render json: Article.limit(10)        end      end    You can also add bullet points:    - make a bullet point by starting a line with either a dash (-)    or an asterisk (\*)    - wrap lines at 72 characters, and indent any additional lines    with 2 spaces for readability |

Please squash your commits into a single commit when appropriate. This simplifies future cherry picks and keeps the git log clean.

**5.13 Update Your Branch**

It's pretty likely that other changes to master have happened while you were working. Go get them:

|  |
| --- |
| $ git checkout master  $ git pull --rebase |

Now reapply your patch on top of the latest changes:

|  |
| --- |
| $ git checkout my\_new\_branch  $ git rebase master |

No conflicts? Tests still pass? Change still seems reasonable to you? Then move on.

**5.14 Fork**

Navigate to the Rails [GitHub repository](https://github.com/rails/rails) and press "Fork" in the upper right hand corner.

Add the new remote to your local repository on your local machine:

|  |
| --- |
| $ git remote add mine https://github.com:<your user name>/rails.git |

Push to your remote:

|  |
| --- |
| $ git push mine my\_new\_branch |

You might have cloned your forked repository into your machine and might want to add the original Rails repository as a remote instead, if that's the case here's what you have to do.

In the directory you cloned your fork:

|  |
| --- |
| $ git remote add rails https://github.com/rails/rails.git |

Download new commits and branches from the official repository:

|  |
| --- |
| $ git fetch rails |

Merge the new content:

|  |
| --- |
| $ git checkout master  $ git rebase rails/master |

Update your fork:

|  |
| --- |
| $ git push origin master |

If you want to update another branch:

|  |
| --- |
| $ git checkout branch\_name  $ git rebase rails/branch\_name  $ git push origin branch\_name |

**5.15 Issue a Pull Request**

Navigate to the Rails repository you just pushed to (e.g. <https://github.com/your-user-name/rails>) and click on "Pull Requests" seen in the right panel. On the next page, press "New pull request" in the upper right hand corner.

Click on "Edit", if you need to change the branches being compared (it compares "master" by default) and press "Click to create a pull request for this comparison".

Ensure the changesets you introduced are included. Fill in some details about your potential patch including a meaningful title. When finished, press "Send pull request". The Rails core team will be notified about your submission.

**5.16 Get some Feedback**

Most pull requests will go through a few iterations before they get merged. Different contributors will sometimes have different opinions, and often patches will need to be revised before they can get merged.

Some contributors to Rails have email notifications from GitHub turned on, but others do not. Furthermore, (almost) everyone who works on Rails is a volunteer, and so it may take a few days for you to get your first feedback on a pull request. Don't despair! Sometimes it's quick, sometimes it's slow. Such is the open source life.

If it's been over a week, and you haven't heard anything, you might want to try and nudge things along. You can use the [rubyonrails-core mailing list](http://groups.google.com/group/rubyonrails-core/) for this. You can also leave another comment on the pull request.

While you're waiting for feedback on your pull request, open up a few other pull requests and give someone else some! I'm sure they'll appreciate it in the same way that you appreciate feedback on your patches.

**5.17 Iterate as Necessary**

It's entirely possible that the feedback you get will suggest changes. Don't get discouraged: the whole point of contributing to an active open source project is to tap into the knowledge of the community. If people are encouraging you to tweak your code, then it's worth making the tweaks and resubmitting. If the feedback is that your code doesn't belong in the core, you might still think about releasing it as a gem.

**5.17.1 Squashing commits**

One of the things that we may ask you to do is to "squash your commits", which will combine all of your commits into a single commit. We prefer pull requests that are a single commit. This makes it easier to backport changes to stable branches, squashing makes it easier to revert bad commits, and the git history can be a bit easier to follow. Rails is a large project, and a bunch of extraneous commits can add a lot of noise.

In order to do this, you'll need to have a git remote that points at the main Rails repository. This is useful anyway, but just in case you don't have it set up, make sure that you do this first:

|  |
| --- |
| $ git remote add upstream https://github.com/rails/rails.git |

You can call this remote whatever you'd like, but if you don't use upstream, then change the name to your own in the instructions below.

Given that your remote branch is called my\_pull\_request, then you can do the following:

|  |
| --- |
| $ git fetch upstream  $ git checkout my\_pull\_request  $ git rebase -i upstream/master    < Choose 'squash' for all of your commits except the first one. >  < Edit the commit message to make sense, and describe all your changes. >    $ git push origin my\_pull\_request -f |

You should be able to refresh the pull request on GitHub and see that it has been updated.

**5.17.2 Updating pull request**

Sometimes you will be asked to make some changes to the code you have already committed. This can include amending existing commits. In this case Git will not allow you to push the changes as the pushed branch and local branch do not match. Instead of opening a new pull request, you can force push to your branch on GitHub as described earlier in squashing commits section:

|  |
| --- |
| $ git push origin my\_pull\_request -f |

This will update the branch and pull request on GitHub with your new code. Do note that using force push may result in commits being lost on the remote branch; use it with care.

**5.18 Older Versions of Ruby on Rails**

If you want to add a fix to older versions of Ruby on Rails, you'll need to set up and switch to your own local tracking branch. Here is an example to switch to the 4-0-stable branch:

|  |
| --- |
| $ git branch --track 4-0-stable origin/4-0-stable  $ git checkout 4-0-stable |

You may want to [put your Git branch name in your shell prompt](http://qugstart.com/blog/git-and-svn/add-colored-git-branch-name-to-your-shell-prompt/) to make it easier to remember which version of the code you're working with.

**5.18.1 Backporting**

Changes that are merged into master are intended for the next major release of Rails. Sometimes, it might be beneficial for your changes to propagate back to the maintenance releases for older stable branches. Generally, security fixes and bug fixes are good candidates for a backport, while new features and patches that introduce a change in behavior will not be accepted. When in doubt, it is best to consult a Rails team member before backporting your changes to avoid wasted effort.

For simple fixes, the easiest way to backport your changes is to [extract a diff from your changes in master and apply them to the target branch](http://ariejan.net/2009/10/26/how-to-create-and-apply-a-patch-with-git).

First make sure your changes are the only difference between your current branch and master:

|  |
| --- |
| $ git log master..HEAD |

Then extract the diff:

|  |
| --- |
| $ git format-patch master --stdout > ~/my\_changes.patch |

Switch over to the target branch and apply your changes:

|  |
| --- |
| $ git checkout -b my\_backport\_branch 3-2-stable  $ git apply ~/my\_changes.patch |

This works well for simple changes. However, if your changes are complicated or if the code in master has deviated significantly from your target branch, it might require more work on your part. The difficulty of a backport varies greatly from case to case, and sometimes it is simply not worth the effort.

Once you have resolved all conflicts and made sure all the tests are passing, push your changes and open a separate pull request for your backport. It is also worth noting that older branches might have a different set of build targets than master. When possible, it is best to first test your backport locally against the Ruby versions listed in .travis.yml before submitting your pull request.

And then... think about your next contribution!

**6 Rails Contributors**

All contributions, either via master or docrails, get credit in [Rails Contributors](http://contributors.rubyonrails.org/).