

Rails BugMash Guide



Introducing the BugMash

The idea is simple: RailsBridge has a lot of energy. [The Rails Lighthouse](#) has a lot of open tickets. With the help of some Rails Core team members, RailsBridge has been helping to cut down the number of open tickets, encourage more people to get involved with the Rails source, and have some fun.

Our mechanism for doing this is the RailsBridge BugMash. During the BugMash, we gather together over a weekend to work on Rails.

The Two Types of BugMash

For 2010, we're experimenting with two different types of BugMash. We'll still hold some BugMashes that are focused primarily on bug eradication, but with the exciting work being done on Rails 3.0, we're also going to spend some time just getting acquainted with and providing feedback on the new version.

Some Bridgers have had success (and a lot of fun) by working in face-to-face teams, but there's no need to be there to be a part of it. We also get together in the #railsbridge room on IRC, and people who are familiar with the Rails internals are available to help point you in the right direction. We do everything we can to make it easy to start contributing to Rails.

New Version BugMash

In January 2010, we'll be holding a BugMash with a focus specifically on Rails 3.0. The goals for this BugMash are:

1. Install the current Rails 3 code on as many computers as possible.
2. Run the Rails 3 tests and make sure they pass, or report test failures.
3. Create a new Rails 3 application and explore the new features.
4. Migrate an existing application to Rails 3 and report pain points.

5. Upgrade an existing plugin or gem to work with Rails 3.

Bug-Swatting BugMash

Sometimes, we attack the Lighthouse backlog with a focus on tidying things up:

1. Confirm that the bug can be reproduced
2. If it can't be reproduced, try to figure out what information would make it possible to be reproduced
3. If it can be reproduced, add the missing pieces: better repro instructions, a failing patch, and/or a patch that applies cleanly to the current Rails source
4. Bring promising tickets to the attention of the Core team



Where do I Start?

- If you're experienced with the Rails source code, read through the BugMash Cheat Sheet for tips.
- If you need to get things set up, skip forward to the Pre-Flight Checklist.
- Either way, browse through the rest of this Guide for a variety of hints and tips.

BugMash Cheat Sheet

Here are a few handy shortcuts for BugMashers.

Generating a new Rails application

You can generate a quick sample application from the Rails 2.3.5 source code tree by

```
cd railties
rake dev
```

This will create a new application named rails at the root of your Rails repository. (**NOTE:** This functionality is not available on the master branch).

You can also generate a sample application to use by calling the rails command directly out of your source tree:

```
cd /parent/of/new/app
ruby /path/to/rails/railties/bin/rails appname
cd appname/vendor
ln -s /path/to/rails/ ./rails
```

This will build a new application and then symlink your checked-out rails tree into the vendor/rails folder.

If you're working with the master branch (Rails 3.0), you also need to bundle the necessary gems into your new application. Edit the Gemfile at the root of your new application, and add these lines at the top (*without changing what's already there*):

```
directory "vendor/rails", :glob => "{*/,}*.*gemspec"
git "git://github.com/rails/arel.git"
git "git://github.com/rails/rack.git"
```

Save the file and run the bundler in the root of your new application:



```
gem bundle
```

Now you should be able to run `script/server` or `script/console` and be working with a Rails 3.0 application.

Testing Rails

Just go to the folder where you've cloned Rails and run rake:

```
cd rails
rake
```

Tests not passing? Don't panic! If you're on the cutting-edge master branch, the problem may be the code, not you. Check <http://ci.rubyonrails.org> to see how the official continuous integration server for Rails is seeing things.

Testing Active Record

If you wish to test patches on Active Record, you'll have to generate test databases for MySQL. Otherwise, most of the Active Record tests won't pass. You should run these tasks in the **activerecord** directory.

```
rake mysql:build_databases
```

There is also a PostgreSQL task for doing the same job given you have PostgreSQL installed on your system.

```
rake postgresql:build_databases
```

When you're done testing patches, you can delete the generated databases.

```
rake mysql:drop_databases  
rake postgresql:drop_databases
```

There are also useful tasks for deleting and regenerating databases if you need to refresh.

```
rake mysql:rebuild_databases  
rake postgresql:rebuild_databases
```

If you're writing new tests for ActiveRecord, please try to reuse the existing test models instead of adding new ones.

Testing specified frameworks only

Running the whole test suite takes a lot of time? You can run tests in the individual Rails frameworks also. Just cd into the library you wish to test and rake test.

```
cd activesupport  
rake test
```

Testing Active Record

Running rake test for Active Record will run tests for MySQL, SQLite3 and PostgreSQL. To run test individually based on different adapters:

```
rake test_mysql  
rake test_postgresql
```

```
rake test_sqlite3
```

You should test all three of these widely-used database adapters if you're contributing to Active Record. See `rake -T` for all the adapters Rails supports, as this is only a fraction of them.

Testing Individual Files

Better yet, you can test a separate file for a speed boost.

```
rake test TEST=test/ordered_options_test.rb
```

If testing ActiveRecord and you've changed the schema, you have to initialize the database before running the test:

```
rake test_mysql TEST=test/cases/aaa_create_tables_test.rb # update the schema
rake test_mysql
TEST=test/cases/associations/has_many_through_associations_test.rb
```

Working with Rails and git

Getting the Rails source:

```
git clone git://github.com/rails/rails.git
cd rails
git checkout -b 2-3-stable origin/2-3-stable # this will leave you on the 2-3-stable
branch
(you can also do: git checkout -t origin/2-3-stable # this will also leave you on the
2-3-stable branch)
```

Working on the master (3.0) branch:

```
git checkout master
```

Working on the 2-3-stable branch:

```
git checkout 2-3-stable
```

Creating your own feature branch:

```
git checkout -b my_feature_branch
```

Apply a patch:

```
git apply <patch file>
```

Creating a patch:

```
git checkout master
```

```
git checkout -b my_feature_branch
```

```
(write and test code)
git commit -a -m "This is my great patch"
git checkout master
git pull
git checkout my_feature_branch
git rebase master
rake (to be sure tests still patch)
git format-patch master --stdout > my_great_patch.diff
```

Patching both master and 2-3-stable:

First, follow above to create a patch for master. If the same patch applies cleanly to 2-3-stable, just say so in the Lighthouse ticket and the committer will apply it to both. Otherwise, you'll need to generate a separate patch for 2-3-stable (assuming that this issue should be patched on both branches). The first thing to try is just cherry-picking your patch over to 2-3-stable:

```
git checkout -b my-feature-2-3 2-3-stable
git cherry-pick <revision of change made to master>
rake test
```

```
git format-patch 2-3-stable --stdout > my_great_patch_for_rails23.diff
```

As a last resort, you can write a completely separate patch for 2.3:

```
git checkout 2-3-stable
git checkout -b my_feature_branch
(write and test code)
git commit -a -m "This is my great patch"
git checkout 2-3-stable
git pull
git checkout my_feature_branch
git rebase 2-3-stable
rake (to be sure tests still patch)
git format-patch 2-3-stable --stdout > my_great_23_patch.diff
```

NOTE

Please keep in mind the core workflow here for Rails itself:

- All active development happens on master
- Changes targeting the stable release are made to master and flow **back** to 2.3
- Development on 2.3 with forward port to master causes log jams and is strongly discouraged

Editing someone else's patch:

First, apply the existing patch:

```
git checkout stable
git apply <patch file>
```

Then if you need to make changes to the patch (perhaps because Rails has moved on), follow this advice from core:

For updating others' patches: preserving authorship is a common courtesy we encourage. The simplest is to assign authorship of the fixed commit to the original author and sign off on it. You can `git commit --author "Foo Bar <foobar@example.com>" --signoff` to make a commit this way, or even `git commit --amend --author ... --signoff` to change the author and add a signoff to the previous commit.

Another scenario is building on an incomplete patch. Rather than apply the patch and commit it as yourself, apply it as the original author. Then do subsequent commits as yourself. This preserves the full history and authorship.



Bot commands

During the BugMash, we run a bot on the #railsbridge channel on IRC. You can communicate with the bot to track your progress:

!work <num> or !working <num> - You start working on a ticket with this ID. Will return an error if ticket does not exist, or is not bugmashable.

!stop <num> or !stopworking <num> - You stop working on a ticket with this ID. Will return an error if you're not working on it, ticket does not exist or is not bugmashable.

!gimme - Tells you about a random ticket from lighthouse that nobody is (yet) working on. To start work on it, use the **!work** command.

!review <num> - Marks the ticket for bugmash-review in Lighthouse.

!unreview <num> - Removes the bugmash-review tag on the ticket in Lighthouse.

!me - Private messages you telling you the number of tickets you're working on and lists them off.

Pre-flight Checklist

The only real requirement for participating in a BugMash is enthusiasm. But there are some things you can do in advance to make it a more productive and rewarding experience.

Required Software

You'll need to set up a Rails development environment. Here are the pieces that you should install before bugmashing:

- **Ruby.** In general, Ruby 1.8.7 at a reasonably current patch level is fine for working on Rails. However, if you want to help check on cross-Ruby-version issues, you'll need to have multiple copies of Ruby and a way to switch between them. We recommend the excellent [rvm](#) for this purpose.
- **RubyGems.** You need to be on the cutting edge to work with the Rails source: version 1.3.5 or later. Run `gem --version` to check your version and, if necessary, `gem update --system` to upgrade.
- **Rails.** You presumably have some version of Rails installed, but to participate, you'll need the Rails source. And because there are still many Rails 2.3.x issues to look at, you should install both Rails 3.0 (the master branch) and Rails 2.3 (the 2-3-stable branch) on your test machine. Here's how:

```
git clone git://github.com/rails/rails.git
cd rails
git branch --track 2-3-stable origin/2-3-stable
```

- **Mocha.** The Rails tests depend on mocha for mocks, so `gem install mocha`. Make sure you have mocha 0.9.7 or higher, or you'll see a whole bunch of mystery errors due to a change in the way that Rails and mocha interact.



- **Bundler.** The Rails master branch now depends on [Bundler](#). You'll need to do this to pack up some of the gems that the tests want to see:

```
[sudo] gem install bundler  
gem bundle
```

TIP: If you see “Could not find a Gemfile to use” then bundler doesn't think it is needed. Check (1) that you're on the master branch, not 2-3-stable and (2) that you're in the root rails folder.

Bundler will create a /vendor folder underneath the root rails folder. To re-do the bundling, just delete this folder and run `gem bundle` again.

- **Rack-Test gem:** `gem install rack-test`. (This is bundled for Rails 3, but you'll need it installed separately to test Rails 2.3).
- **MySQL.** Having a working MySQL install is the bare minimum to run the tests on Rails.
- **PostgreSQL and sqlite.** If you're planning to work on Active Record issues, you'll need to run the AR tests across MySQL, PostgreSQL, and sqlite at the very minimum.

Accounts

- You'll need an account with Lighthouse to comment on [Rails issues](#) or to upload tests or patches. Sign up [here](#)
- You don't need a [Github account](#) unless you start doing such extensive surgery that you want to have your own fork of the Rails repo. (Note that the Rails team only accepts patches, not pull requests)

Optional Components

- If you'd like to help the JRuby-on-Rails story, you'll need to have [JRuby](#) installed. Note that the Relevance switcher mentioned above can do this for you.
- If you're brave enough to test ActiveRecord with JDBC connections and databases, [this ticket](#) will point you in the right direction.

- If you do a great deal of work with Lighthouse, you may find the OS X application [Lighthouse Keeper](#) useful.

TIP: Windows users can find an installation guide over at the [devChix](#) wiki. (You can skip steps 11 and 12 for setting up Heroku.) But beware! In past BugMashes, Windows users have found it very difficult to get the Rails tests working. If at all possible, you should install a Linux virtual machine and use that instead of trying to test on native Windows.

Official BugMash hours

Rails contributors are located all over the world, so we define an extended weekend for the BugMash. BugMashes run from Saturday noon in New Zealand (00:00:00 GMT) to Sunday midnight on the US West coast (07:00:00 GMT). That gives everyone who wants to be involved plenty of time to participate.

BugMash on IRC

We try to have experienced Rails developers and core team members available on IRC for as many hours during the BugMash as humanly possible. If you're having trouble getting started, want to brainstorm about a particular ticket, or can help other people out, please come by and join us. We'll be hanging out on the #railsbridge channel on Freenode IRC. The RailsBridge [IRCGuide](#) can help you get connected.

In addition to real live human beings, we also have a bot who hangs out in the channel and helps us keep track of who's working on what.

LightHouse Mechanics

Before each BugMash, some of the Rails core team members go through the open issues in Lighthouse and add the [bugmash](#) tag to tickets that they're especially interested in seeing tackled. This allows you to use a Lighthouse query to find interesting tickets. You're not required to stick to those tickets, but they're a good starting point.

Note that some of the tagged tickets are marked as “stale” or “incomplete,” and Lighthouse will show them with a strikethrough in list view. *These are still fair game for the BugMash!* One of the BugMash purposes is to take a second look at some of the issues that have been in a holding pattern for a long time.

Scoring! Prizes!

Every BugMash comes with prizes, and we have a scoring system that keeps track of your effort. Here's a quick rundown on the scoring:

- 25 pts for every +/- 1 (awarded to the commenter not the ticket creator)
- 50 pts for every verified/not reproducible (awarded to the commenter)
- 50 pts for a new ticket
- 100 pts for supplying a test case/patch
- 1000 pts for every changeset

Quick example:

hardbap opens ticket "HasOneThroughAssociation should not be a child of HasManyThroughAssociation", +50

includes a world class patch, +100 for hardbap

mikeg verifies the patch, +50 for mikeg

hobbs +1, +25 for hobbs

hardbap's patch is committed to core with gusto by lifo, +1000 for hardbap

hardbap's total for the ticket = 1150

mikeg's total for the ticket = 50

hobbs's total for the ticket = 25

For every 100 points, you get 1 ticket in the lottery to distribute the prizes.

Continuous Integration

There is an official [Continuous Integration](http://ci.rubyonrails.org) environment set up for the Rails builds at <http://ci.rubyonrails.org>. It runs the entire suite of Rails tests on multiple machines, against multiple databases and Ruby interpreters. The builds will start automatically on new commits (no need to click "build now"), and should be kept green. If you have questions or problems, or if you suspect a test is failing due to a misconfiguration in the CI environment, contact thewoolleyman+railsci@gmail.com. Remember that this environment is still beta, so it may be down occasionally.

Scoring Tips

- To up or down vote a ticket use "+1" and "-1". Your comment should also include a brief explanation of your vote.
- When verifying a patch or bug use "verified" or "not reproducible".
- If you've included a patch make sure your comment includes the phrase "I've attached a patch."
- Changeset points will be awarded when the patch is committed to the Rails core.
- You can check your score at the [official scoreboard](#) and if we've botched your score please [let us know](#)
- Good luck!

Bugmash.com

Some information about the special strings in Lighthouse messages that Bugmash.com is looking for:

1. To up or down vote a ticket use "+1" and "-1". Your comment should also include a brief explanation of your vote.
2. When verifying a patch or bug use "verified" or "not reproducible".
3. If you've included a patch make sure your comment includes the phrase "I've attached a patch."
4. Changeset points will be awarded when the patch is committed to the Rails core.



5. You can check your score at the [official scoreboard](#) and if we've botched your score please [let us know](#)

BugMash Flowchart

Not entirely sure what to do with an issue? Here's a step-by-step guide:

If the ticket has no test or patch:

I. Is it a bug?

A. Can you verify with latest Rails (2-3-stable or master)?

1. If yes, add a comment with the word “verified” to the ticket explaining how you verified
2. If no, add a comment with the phrase “not reproducible” to the ticket explaining what you tried to verify

II. Is it a feature request?

A. Could you use this in your projects?

1. If yes, add a +1 with your reasoning to the ticket
2. If no, add a -1 with your reasoning to the ticket
3. If you're not sure, hunt up an experienced person in #railsbridge to consult with

III. Can you provide a test case?

A. If yes, great, attach it to the ticket and make sure your comment includes the phrase “I’ve attached a patch.”

B. If no, consult about how to write a test

IV. Can you provide a patch?

A. If yes, great, attach it to the ticket and make sure your comment includes the phrase “I’ve attached a patch.”

B. If no, consult about how to write a patch

If the ticket has a test but no patch:

I. Does the test still fail on latest Rails (2-3-stable or master)?



- A. If yes, add a comment (which should include the word “verified”) to the ticket explaining that you checked
 - B. If no, add a comment to the ticket stating that it’s “not reproducible,” but think about what might be different for you
- II. Can you provide a patch?
- A. If yes, great, attach it to the ticket and make sure your comment includes the phrase “I’ve attached a patch.” Then ask a committer to look over the ticket and commit the patch so you get credit for it.
 - B. If no, consult about how to write a patch

If the ticket has a patch but no test:

- I. Does the patch still apply to latest Rails (2-3-stable or master)?
- A. If yes, would you find this useful?
 1. If yes, add a +1 with your reasoning to the ticket
 2. If no, add a -1 with your reasoning to the ticket
- II. Can you supply a test?
- A. If yes, create a diff including the patch and a test. Remember, the test should fail without the patch applied and pass with it applied. Attach to the ticket and make sure your comment includes the phrase “I’ve attached a test.” Then ask a committer to look over the ticket and commit the patch so you get credit for it.
 - B. If no, consult about how to write a test

If the ticket has a test and a patch:

- I. Does the patch still apply to latest Rails (2-3-stable or master)?
- A. If yes, would you find this useful?
 1. If yes, add a +1 with your reasoning to the ticket
 2. If no, add a -1 with your reasoning to the ticket
 3. If you're not sure, hunt up an experienced person in #railsbridge to consult with
 - B. If no, can you fix the patch?
 1. If yes, do so (be sure to fix the test too) and attach to the ticket and make sure your comment includes the phrase “I’ve attached a patch.” Then ask a committer to look over the ticket and commit the patch so you get credit for it.

Links

- [Pre-flight Checklist](#)
- [BugMashFlowchart](#)
- [BugMashCheatSheet](#)
- [Contributing to Rails](#)
- [Tickets tagged with 'bugmash'](#)
- There are official Rails Continuous Integration servers which test against multiple interpreters and databases. See the Continuous Integration section in the [BugMashFlowchart](#)
- The Rails [Continuous Integration Server Setup Notes](#) can be helpful in getting the various databases configured
- <http://railscasts.com/episodes/113-contributing-to-rails-with-git>
- [BugMashStats](#)

Photo Credits

- Stock.xchng user [lusi](#) (fly photo)
- Stock.xchng user [lustfish](#) (plane photo)
- Flickr user [DaseinDesign](#) (coffee flowchart photo)
- Stock.xchng user [asifthebes](#) (traffic lights photo)
- Stock.xchng user [ngould](#) (rakes photo)
- Stock.xchng user [sasan](#) (robot photo)