
Jeweler maintenance has now shifted to Fred Mitchell. I am now maintaining both Jeweler and its fork, *juwelier*. I will keep Jeweler at least functional with the latest Ruby releases, but put new features in *Juwelier*. Your input on this is more than welcome.

Jeweler: Craft the perfect RubyGem

chat on [gitter](#)

Jeweler provides the noble ruby developer with two primary features:

- a library for managing and releasing RubyGem projects
- a scaffold generator for starting new RubyGem projects

PLEASE NOTE that if you are starting afresh, please use the successor *Juwelier* I (Fred Mitchell, [flajann2](#)) will be maintaining both Jeweler and *Juwelier*, but will be adding new features to *Juwelier*, and eventually “merge” this one into *Juwelier* after some namespace issues are dealt with.

build unknown coverage 89% coverage 89% maintainability B

Hello, world

Use RubyGems to install the heck out of jeweler to get started:

```
1 $ gem install jeweler
```

With jeweler installed, you can use the `jeweler` command to generate a new project. For the most basic use, just give it a name:

```
1 $ jeweler hello-gem
```

This requires some Git configuration (like name, email, GitHub account, etc), but `jeweler` will prompt along the way.

Your new `hello-gem` gem is ready in the `hello-gem` directory. Take a peek, and you’ll see several files and directories

- `Rakefile` setup for jeweler, running tests, generating documentation, and releasing to [rubygems.org](#)
- `README.rdoc` with contribution guidelines and copyright info crediting you
- `LICENSE` with the MIT licensed crediting you
- `Gemfile` with development dependencies filled in

-
- `lib/hello-gem.rb` waiting for you to code
 - `test/` containing a (failing) shoulda test suite shoulda

More jeweler options

The `jeweler` command supports a lot of options. Mostly, they are for generating baked in support for this test framework, or that.

Check out `jeweler --help` for the most up to date options.

Hello, rake tasks

Beyond just editing source code, you'll be interacting with your gem using `rake` a lot. To see all the tasks available with a brief description, you can run:

```
1 $ rake -T
```

You'll need a version before you can start installing your gem locally. The easiest way is with the `version:write` Rake task. Let's imagine you start with 0.1.0

```
1 $ rake version:write MAJOR=0 MINOR=1 PATCH=0
```

You can now go forth and develop, now that there's an initial version defined. Eventually, you should install and test the gem:

```
1 $ rake install
```

The `install` rake task builds the gem and `gem install`s it. You're all set if you're using RVM, but you may need to run it with `sudo` if you have a system-installed ruby:

```
1 $ sudo rake install
```

Releasing

At last, it's time to ship it! Make sure you have everything committed and pushed, then go wild:

```
1 $ rake release
```

This will automatically:

- Generate `hello-gem.gemspec` and commit it
- Use `git` to tag `v0.1.0` and push it

-
- Build `hello-gem-0.1.0.gem` and push it to rubygems.org

`rake release` accepts `REMOTE`(default: `origin`), `LOCAL_BRANCH`(default: `master`), `REMOTE_BRANCH`(default: `master`) and `BRANCH`(default: `master`) as options.

```
1 $ rake release REMOTE=upstream LOCAL_BRANCH=critical-security-fix
  REMOTE_BRANCH=v3
```

This will tag and push the commits on your local branch named `critical-security-fix` to branch named `v3` in remote named `upstream` (if you have commit rights on `upstream`) and release the gem.

```
1 $ rake release BRANCH=v3
```

If both remote and local branches are the same, use `BRANCH` option to simplify. This will tag and push the commits on your local branch named `v3` to branch named `v3` in remote named `origin` (if you have commit rights on `origin`) and release the gem.

Version bumping

It feels good to release code. Do it, do it often. But before that, bump the version. Then release it. There's a few ways to update the version:

```
1 # version:write like before
2 $ rake version:write MAJOR=0 MINOR=3 PATCH=0
3
4 # bump just major, ie 0.1.0 -> 1.0.0
5 $ rake version:bump:major
6
7 # bump just minor, ie 0.1.0 -> 0.2.0
8 $ rake version:bump:minor
9
10 # bump just patch, ie 0.1.0 -> 0.1.1
11 $ rake version:bump:patch
```

Then it's the same `release` we used before:

```
1 $ rake release
```

Customizing your gem

If you've been following along so far, your gem is just a blank slate. You're going to need to make it colorful and full of metadata.

You can customize your gem by updating your `Rakefile`. With a newly generated project, it will look something like this:

```
1 require 'jeweler'
2 Jeweler::Tasks.new do |gem|
3   # gem is a Gem::Specification... see http://guides.rubygems.org/
4     specification-reference/ for more options
5   gem.name = "whatwhatwhat"
6   gem.summary = %Q{TODO: one-line summary of your gem}
7   gem.description = %Q{TODO: longer description of your gem}
8   gem.email = "josh@technicalpickles.com"
9   gem.homepage = "http://github.com/technicalpickles/whatwhatwhat"
10  gem.authors = ["Joshua Nichols"]
11 end
12 Jeweler::RubygemsDotOrgTasks.new
```

It's crucial to understand the `gem` object is just a `Gem::Specification`. You can read up about it at guides.rubygems.org/specification-reference. This is the most basic way of specifying a gem, Jeweler-managed or not. Jeweler just exposes this to you, in addition to providing some reasonable defaults, which we'll explore now.

Project information

```
1 gem.name = "whatwhatwhat"
```

Every gem has a name. Among other things, the gem name is how you are able to `gem install` it. Reference

```
1 gem.summary = %Q{TODO: one-line summary of your gem}
```

This is a one line summary of your gem. This is displayed, for example, when you use `gem list --details` or view it on rubygems.org.

```
1 gem.description = %Q{TODO: longer description of your gem}
```

Description is a longer description. Scholars ascertain that knowledge of where the description is used was lost centuries ago.

```
1 gem.email = "josh@technicalpickles.com"
```

This should be a way to get a hold of you regarding the gem.

```
1 gem.homepage = "http://github.com/technicalpickles/whatwhatwhat"
```

The homepage should have more information about your gem. The jeweler generator guesses this based on the assumption your code lives on GitHub, using your Git configuration to find your GitHub

username. This is displayed by `gem list --details` and on rubygems.org.

```
1 gem.authors = ["Joshua Nichols"]
```

Hey, this is you, the author (or me in this case). The `jeweler` generator also guesses this from your Git configuration. This is displayed by `gem list --details` and on rubygems.org.

Files

The quickest way to add more files is to `git add` them. `Jeweler` uses your Git repository to populate your gem's files by including added and committed and excluding `.gitignored`. In most cases, this is reasonable enough.

If you need to tweak the files, that's cool. `Jeweler` populates `gem.files` as a `Rake::FileList`. It's like a normal array, except you can `include` and `exclude` file globs:

```
1 gem.files.exclude 'tmp' # exclude temporary directory
2 gem.files.include 'lib/foo/bar.rb' # explicitly include lib/foo/bar.rb
```

If that's not enough, you can just set `gem.files` outright

```
1 gem.files = Dir.glob('lib/**/*.rb')
```

Dependencies

Dependencies let you define other gems that your gem needs to function. `gem install your-gem` will install your-gem's dependencies along with it, and when you use your-gem in an application, the dependencies will be made available. Use `gem.add_dependency` to register them. Reference

```
1 gem.add_dependency 'nokogiri'
```

This will ensure a version of `nokogiri` is installed, but it doesn't require anything more than that. You can provide extra args to be more specific:

```
1 gem.add_dependency 'nokogiri', '= 1.2.1' # exactly version 1.2.1
2 gem.add_dependency 'nokogiri', '>= 1.2.1' # greater than or equal to
  1.2.1, ie, 1.2.1, 1.2.2, 1.3.0, 2.0.0, etc
3 gem.add_dependency 'nokogiri', '>= 1.2.1', '< 1.3.0' # greater than or
  equal to 1.2.1, but less than 1.3.0
4 gem.add_dependency 'nokogiri', '~> 1.2.1' # same thing, but more
  concise
```

When specifying which version is required, there's a bit of the conundrum. You want to allow the most versions possible, but you want to be sure they are compatible. Using `>= 1.2.1` is fine most of the time, except until the point that 2.0.0 comes out and totally breaks backwards the API. That's when it's good to use `~> 1.2.1`, which requires any version in the 1.2 family, starting with 1.2.1.

Executables

Executables let your gem install shell commands. Just put any executable scripts in the `bin/` directory, make sure they are added using `git`, and Jeweler will take care of the rest.

When you need more finely grained control over it, you can set it yourself:

```
1 gem.executables = ['foo'] # note, it's the file name relative to `bin`  
  /`, not the project root
```

Versioning

We discussed earlier how to bump the version. The rake tasks are really just convenience methods for manipulating the `VERSION` file. It just contains a version string, like `1.2.3`.

`VERSION` is a convention used by Jeweler, and is used to populate `gem.version`. You can actually set this yourself, and Jeweler won't try to override it:

```
1 gem.version = '1.2.3'
```

A common pattern is to have this in a version constant in your library. This is convenient, because users of the library can query the version they are using at runtime.

```
1 # in lib/foo/version.rb  
2 class Foo  
3   module Version  
4     MAJOR = 1  
5     MINOR = 2  
6     PATCH = 3  
7     BUILD = 'pre3'  
8  
9     STRING = [MAJOR, MINOR, PATCH, BUILD].compact.join('.')  
10   end  
11 end  
12  
13 # in Rakefile  
14 require 'jeweler'  
15 require './lib/foo/version.rb'  
16 Jeweler::Tasks.new do |gem|  
17   # snip
```

```
18   gem.version = Foo::Version::STRING
19 end
```

Rake tasks

Jeweler lives inside of Rake. As a result, they are dear friends. But, that friendship doesn't interfere with typical Rake operations.

That means you can define your own namespaces, tasks, or use third party Rake libraries without cause for concern.

Contributing to Jeweler

- Check out the latest master to make sure the feature hasn't been implemented or the bug hasn't been fixed yet
- Ask on the mailing list for feedback on your proposal, to see if somebody else has done it.
- Check out the issue tracker to make sure someone already hasn't requested it and/or contributed it
- Fork the project
- Start a feature/bugfix branch
- Commit and push until you are happy with your contribution
- Make sure to add tests for the feature/bugfix. This is important so I don't break it in a future version unintentionally.
- Please try not to mess with the Rakefile, version, or history. If you want to have your own version, or is otherwise necessary, that is fine, but please isolate it to its own commit so I can cherry-pick around it.

Copyright

Copyright (c) 2008-2010 Josh Nichols. Copyright (c) 2016 Fred Mitchell. See LICENSE for details.