Gemの作り方まとめ 普通のgem編

Feb 15th, 2014

会社の人にgemの作り方まとめてくれって言われたので標準的なgemの作り方をまとめます。 標準的な作り方なので他の人が作ったgemを読み解くヒントにもなります。

とはいえ有名なgemは(有名なgemに限って)メッチャクチャだったりするので読みづらかったりします。 歴史が古かったりすると特にね。

ジェネレータ

まずはジェネレータを使ってプロジェクトを作りましょう。 昔はいろいろあったけど最近は `bundle` コマンドで大勢が決定してる感じです。

```
bundle gem test_gem -t
```

`-t` はテストも作成するオプションです。デフォルトでrspecを使うようになってます。 その他のオプションは

```
bundle help gem
```

で確認できます。

Railsプラグインのgemを作る場合は

```
rails plugin new test_gem
```

を使う方法もあります。 別記事で詳しく書く予定です。

作成されるファイル

作られるファイルの一覧はこんなかんじです

```
 create  test_gem/Gemfile
 create  test_gem/Rakefile
 create  test_gem/LICENSE.txt
 create  test_gem/README.md
 create  test_gem/.gitignore
```

```
 create  test_gem/test_gem.gemspec
 create  test_gem/lib/test_gem.rb
 create  test_gem/lib/test_gem/version.rb
 create  test_gem/.rspec
 create  test_gem/spec/spec_helper.rb
 create  test_gem/spec/test_gem_spec.rb
 create  test_gem/.travis.yml
```

gem名について

gem名の - はディレクトリ構造に変換されます。 例えば `test-gem` という名前で作ると `lib/test_gem.rb` だったところが `lib/test/gem.rb` のような構造になります。

`Gemfile` 内で

```
1 gem 'test-gem'
```

が正しくロードできるように `lib/test-gem.rb` を作ってその中で

```
1 require 'test/gem'
```

するといいでしょう。

gemspec

gemの中を見たことがなければgemspecというファイルは初見でしょう。 gemspecファイルはgemのメタデータが書かれています。

生成された直後はこんな内容になっています。

```
1  # coding: utf-8
2  lib = File.expand_path('../lib', __FILE__)
3  $LOAD_PATH.unshift(lib) unless $LOAD_PATH.include?(lib)
4  require 'test_gem/version'
5
6  Gem::Specification.new do |spec|
7    spec.name          = "test_gem"
8    spec.version       = TestGem::VERSION
9    spec.authors       = ["HOGE"]
10   spec.email         = ["hoge@example.com"]
11   spec.description   = %q{TODO: Write a gem description}
12   spec.summary       = %q{TODO: Write a gem summary}
13   spec.homepage      = ""
14   spec.license       = "MIT"
15
16   spec.files         = `git ls-files`.split($/)
17   spec.executables   = spec.files.grep(%r{^bin/}) { |f| File.basename(f) }
18   spec.test_files    = spec.files.grep(%r{^(test|spec|features)/})
19   spec.require_paths = ["lib"]
20
21   spec.add_development_dependency "bundler", "~> 1.3"
```

```
22   spec.add_development_dependency "rake"
23   spec.add_development_dependency "rspec"
24 end
```

まずTODOのところを埋めましょう。 spec.homepage にはgithubのurlを入れましょう。

このファイルで重要なのは依存するgemの設定です。 `add_dependency` と `add_development_dependency` を使って必要なgemを追加しましょう。 Gemfileはこの中身を参照しているので変更する必要はありません。

それ以外の設定はなるべくいじらないほうがいいです。

中身の作成

まず `lib/test_gem.rb` を見てみましょう。

```
1 require "test_gem/version"
2
3 module TestGem
4   # Your code goes here...
5 end
```

注目するのは TestGem が module だということです。 ここはなるべく module のままで、実際に色々やるのは下の class に任せましょう。

例えば Client という class を作るなら、 `lib/test_gem/client.rb` を作って

```
1 module TestGem
2   class Client
3     # codes
4   end
5 end
```

な感じに書きます。 クラス構造とディレクトリ構造を一致させましょう。

それを `lib/test_gem.rb` でロードします。

```
1 require 'test_gem/version'
2 require 'test_gem/client'
3
4 module TestGem
5 end
```

ActiveSupportのautoloadを使う場合は、

```
1 require 'active_support/dependencies'
```

```
2
3 module TestGem
4   autoload :Client, 'test_gem/client'
5 end
```

という感じになります。 Railsだと Gemfile に書いたgemを自動でrequireしてくれますが、ここではしてくれないので手動でrequireしなければいけません。

テストもディレクトリ構造を一致させましょう。 最初は spec/test_gem_spec.rb しかありません。 spec/test_gem ディレクトリを作り、 その中に spec/test_gem/client_spec.rb を作るという感じです。

guard を使う場合は、

```
1 guard :rspec do
2   watch(%r{^lib/(.+)\.rb$})     { |m| "spec/#{m[1]}_spec.rb" }
3 end
```

と定義すればちょうどディレクトリ構造がぴったり合います。

実行可能なコマンドを含める

例えば rspec のような実行可能なコマンドも簡単に含めることができます。 bin ディレクトリを作り、その中に実行ファイルを入れるだけです。

コマンドなのでファイル名に .rb を付けないほうが見栄えがいいです。 例えば bin/run_test_gem というファイル名で

```
1 #!/usr/bin/env ruby
2
3 require 'test_gem'
4
5 p TestGem::Client.new
```

のような内容にすると、 run_test_gem というコマンドが使えるようになります。

ただし普段rvmを使っているので、これに関してだけは他のシステムだとどうなるかちょっと分からないです。 rvmの場合、直接コマンドを実行せずにラッパースクリプト経由になるので

```
1 TestGem::Client.new
```

だけで動いてしまいます。

ビルドとかリリースとか

ビルドやリリースはすべてrakeタスクが用意されています

```
rake -T
```

で確認しましょう。

リリースの前は `lib/test_gem/version.rb` を手で編集してバージョン番号を上げないといけません。 (これのせいでjewelerが好きだった)

上記の実行可能コマンドを実際に試してみたいときは、**対象ファイルをコミットして**

```
rake install
```

でインストールしないと実行できないというよくハマりがちな罠があります。

実際にどんなふうにクラス書いていけばいいの?

他のgemをたくさん読みましょう。

よく知られているgemの多くはとにかく**デカイ**です。 構造を把握するだけでもかなり大変です。 小奇麗にまとまったgemを探して読んでみましょう。

一番大事なこと

コードにもコミットメッセージにも **日本語を含めるべきではない** 。

もうちっとだけ続くんじゃ

- Railsプラグインの作り方
- C拡張を含むgemの作り方

Posted by masarakki Feb 15th, 2014 ruby

« 不自由なデプロイを強いられてる人のための capistrano-env ってgemを作った 山城が死んだ »

Recent Posts

- ほぼ電子書籍に移行した話
- 超チューニング祭に参加した
- 非エンジニアのためのOAuth講座
- 山城が死んだ
- Gemの作り方まとめ 普通のgem編

GitHub Repos

- snapshot
- sugoi-search
- jfroxy
- niconico-chromecast
- yabai-git-commands
- nyaruko_lang

  いつもニコニコあなたの隣に這いよる混沌ニャルラトホテプ言語ですっ

@masarakki on GitHub

Google+
