Episode 245

New Gem With Bundler

The way that Ruby gems are created and managed has evolved over the years. Back in episode 135 [watch[1], read[2]] we used the echoe gem to create a gem; about a year later in episode 183 [watch[3], read[4]] we used Jeweler. Both of these tools offer ways to generate Gemspec files for each release of a gem and the Gemspec file is managed with that tool. This time we're going to manage the Gemspec file manually using a method that is much simpler overall.

With this method we still need to create the initial Gemspec file for each of our gems. We could do this manually or we could use a tool like Bundler to help us. Bundler is generally thought of as a way of managing an application's dependencies but it also has a little-known command for creating gems that works very well. That command is `bundle gem` and we pass it the name of the gem that we want to create. For the purposes of this episode we're going to create a simple gem called `lorem` that generates some "Lorem ipsum" text.

```
$ bundle gem lorem
      create  lorem/Gemfile
      create  lorem/Rakefile
      create  lorem/.gitignore
      create  lorem/lorem.gemspec
      create  lorem/lib/lorem.rb
      create  lorem/lib/lorem/version.rb
Initializating git repo in /Users/eifion/code/ep245/lorem
```

The `bundle gem` command creates a new directory and generates several files in it. It also initializes a git repository there. This assumes that you're using git and we'll explain why it does this shortly. Before we do that we'll walk you through some of the generated files, starting with the `lorem.gemspec` file.

---

[1] http://railscasts.com/episodes/135-making-a-gem

[2] http://asciicasts.com/episodes/135-making-a-gem

[3] http://railscasts.com/episodes/183-gemcutter-jeweler

[4] http://asciicasts.com/episodes/183-gemcutter-jeweler

/lorem/lorem.gemspec

```
# -*- encoding: utf-8 -*-
$:.push File.expand_path("../lib", __FILE__)
require "lorem/version"

Gem::Specification.new do |s|
  s.name        = "lorem"
  s.version     = Lorem::VERSION
  s.platform    = Gem::Platform::RUBY
  s.authors     = ["TODO: Write your name"]
  s.email       = ["TODO: Write your email address"]
  s.homepage    = ""
  s.summary     = %q{TODO: Write a gem summary}
  s.description = %q{TODO: Write a gem description}

  s.rubyforge_project = "lorem"

  s.files         = `git ls-files`.split("\n")
  s.test_files    = `git ls-files -- ↵
    {test,spec,features}/*`.split("\n")
  s.executables   = `git ls-files -- bin/*`.split("\n").map{ ↵
    |f| File.basename(f) }
  s.require_paths = ["lib"]
end
```

We can see straight away by looking at the TODO items that this file is designed to be edited directly. Once we complete these items we'll have a working Gemspec file. One interesting part of this file are the file attributes towards the end of the file. Each of these is determined at runtime by using the `git ls-files` command by using git to determine the files that will be included in the gem and this is why `bundle gem` creates a git repository. The good thing about this is that it will automatically inherit the behaviour of the `.gitignore` file so that temporary files and the files we don't want in our git repository (and therefore don't want in our gem) are excluded. The defaults above will work for most gems but they can always be edited if we need different behaviour.

Another thing to note is how the version number is determined. This is defined as a constant called `Lorem::VERSION` and this value is defined in another file called

`version.rb` in the `lib/lorem` directory. All this file does is define the version number.

`/lib/lorem/version.rb`

```
module Lorem
  VERSION = "0.0.1"
end
```

When it's time to update our gem to a new version we can change the version number here and republish the gem.

The other file generated in the `lib` directory is called `lorem.rb` and it's this file that's loaded when someone `requires` our gem. We can put any code we like in this file or create other files in the `lib` directory and `require` them here. For our simple Lorem gem we'll just create a class method called `ipsum` that returns some Lorem Ipsum text.

`/lib/lorem.rb`

```
module Lorem
  def self.ipsum
    "Lorem ipsum dolor sit amet, consectetur adipisicing ...."
  end
end
```

Publishing

So, we've finished our gem now and we're ready to publish the first version. Before we do we'll update the Gemspec file and replace the TODO text.

/lorem.gemspec

```
# -*- encoding: utf-8 -*-
$:.push File.expand_path("../lib", __FILE__)
require "lorem/version"

Gem::Specification.new do |s|
  s.name        = "lorem"
  s.version     = Lorem::VERSION
  s.platform    = Gem::Platform::RUBY
  s.authors     = ["Eifion Bedford"]
  s.email       = ["eifion@asciicasts.com"]
  s.homepage    = ""
  s.summary     = %q{Lorem ipsum generator}
  s.description = %q{Simply generates lorem ipsum text.}

  s.rubyforge_project = "lorem"

  s.files         = `git ls-files`.split("\n")
  s.test_files    = `git ls-files -- {test,spec,features}/*`.split
("\n")
  s.executables   = `git ls-files -- bin/*`.split("\n").map{ |f|
File.basename(f) }
  s.require_paths = ["lib"]
end
```

Then we can run the `gem build` command and point it to our Gemspec file.

```
$ gem build lorem.gemspec
WARNING:  no homepage specified
  Successfully built RubyGem
  Name: lorem
  Version: 0.0.1
  File: lorem-0.0.1.gem
```

This command generates a `.gem` file. If we were to run the `gem push` command now that would push the gem up to RubyGems.org and publish it so that others could install it.

Updating

It really is that easy to publish a gem with a Gemspec file. When we want to release a new version then we make the necessary changes to the code and then update the version number in `version.rb`.

/lib/lorem/version.rb

```
module Lorem
  VERSION = "0.0.2"
end
```

We can then do a `gem build` and `gem push` again to build and publish the new version.

Gemfile and Rakefile

There are two more files that Bundler generated for us that we've not yet covered, the `Gemfile` and the `Rakefile`. We'll look at the `Gemfile` first. In a Rails application the `Gemfile` manages gem dependencies with Bundler. If we look inside this `Gemfile` we'll see that it has just one command: `gemspec`.

/Gemfile

```
source "http://rubygems.org"

# Specify your gem's dependencies in lorem.gemspec
gemspec
```

This command will look inside the gem's Gemspec file for the gem's dependencies and load them through Bundler and generally we don't need to worry about managing this file directly. It's better to manage the gem's dependencies inside the Gemspec file and let Bundler load them automatically through the `Gemfile`. For example let's say that we want to use RSpec to test this gem. Instead of adding a reference to RSpec in the `Gemfile` we'll add it as a development dependency in `lorem.gemspec`.

If you're not familiar with managing a Gemspec file then it's well worth taking a look at the documentation[5]. The documentation lists the various methods you can call inside the Gemspec for setting various attributes, including an `add_development_dependency` method which is want we want to use for adding RSpec.

`/lorem.gemspec`

```
# -*- encoding: utf-8 -*-
$:.push File.expand_path("../lib", __FILE__)
require "lorem/version"

Gem::Specification.new do |s|
  s.add_development_dependency "rspec"
  # Other attributes omitted
end
```

As the Gemspec file is referenced through the `Gemfile` we can run `bundle` to ensure that we have all the required gems installed.

```
$ bundle
Fetching source index for http://rubygems.org/
Using diff-lcs (1.1.2)
Using lorem (0.0.1) from source at /Users/eifion/Desktop/Dropbox/
rails/apps_for_asciicasts/ep245/lorem
Installing rspec-core (2.3.1)
Installing rspec-expectations (2.3.0)
Installing rspec-mocks (2.3.0)
Installing rspec (2.3.0)
Using bundler (1.0.7)
Your bundle is complete! It was installed into /Users/eifion/.rvm/
gems/ruby-1.9.2-p0
```

This is useful because when we publish the gem and have other developers contribute to it we can instruct them in the README file to run the `bundle` command to get the environment and all of the dependencies set up.

That's the `Gemfile` but what about the `Rakefile`, why does Bundler generate that? If we take a look at it we'll see that it adds some gem helper tasks from Bundler.

---

[5] http://docs.rubygems.org/read/chapter/20#dependencies

/Rakefile

```
require 'bundler'
Bundler::GemHelper.install_tasks
```

We can take a look at those tasks by running `rake -T`.

```
$ rake -T
(in /Users/eifion/code/ep245/lorem)
rake build    # Build lorem-0.0.2.gem into the pkg directory
rake install  # Build and install lorem-0.0.2.gem into system gems
rake release  # Create tag v0.0.2 and build
              # and push lorem-0.0.2.gem to Rubygems
```

The code in the `Rakefile` generates three tasks. These are convenience tags for the common tasks of building, installing and releasing the gem. A common workflow is to call `rake install` when a gem is set up the way we want so that we can install it on our local machine and test it out fully and then call `rake release` to tag that release and publish the gem to RubyGems.

Updating Existing Gems

Bundler provides a pretty nice way to set up a gem from scratch but what if we have an existing gem that we'd like to apply this workflow to? It's easy to apply this technique to an existing gem by copy over some of the files. The files that need to be copied are the `Gemfile`, the `.gemspec` file and the `Rakefile` if you want the rake tasks that it provides. Adding those files to any existing gem will enable us to use this same workflow to any existing gems.

Overall Bundler is a great solution for creating and managing gems. It's much simpler than the other tools we've covered in previous episodes so it's well worth considering when you're creating any new gems.