Uncle John's Bathroom Reader

Dependencies, Gems and Bundler

by Jean Baptiste Semver

Writing Ruby is pretty easy but chances are you're going to want to share your work or even use something written by somebody else. In this issue of **Uncle John's Bathroom Reader**, we're going to learn about dependency management with **gems** and the infamous **bundler** tool.

All dependency resolution begins at the **gemspec** file. This file allows gems to specify what other gems they are required to have. In specifying dependencies you can specify the version of your dependency in four different ways:

```
Gem::Specification.new do |s|
  # Any version available
  s.add_dependency('json')

  # Any version greater than a specific release
  s.add_dependency('httparty', ['> 0.7'])

  # Any point release of this major version (e.g. 1.1 and 1.2, but not 2.0)
  s.add_dependency('capybara', ['~> 1.0'])

  # This specific release of the gem
  s.add_dependency('sinatra', ['1.3.2'])
end
```

The gemspec is a useful way for a gem to specify dependencies, but for a project such as a web application, you will want to use bundler and a **`Gemfile`**. The Gemfile uses the same version syntax as the gemspec file for specifying project dependencies.

```
source :rubygems

gem 'json'
gem 'httparty', '> 0.7'
gem 'capybara', '~> 1.0'
gem 'sinatra', '1.3.2'
```

A Gemfile also will allow you to specify dependencies that are not published to a Gem repository, such as a gem on your local system or even in a remote Git repository!

```
gem 'openbanana', :path => '~/gems/openbanana'
gem 'cucumber', :git => 'git://github.com/cucumber/cucumber.git'
```

This is a useful mechanism for testing pre-release or development versions of gems with your projects, but it requires you preface commands with **`bundle exec`** in order to pull the dependencies into Ruby properly (e.g. **`bundle exec rake -T`**). There's not much more to dependency management in Ruby (for better or worse) but between gemspecs and bundler there is enough flexibility to meet most of your needs.

That's it for this issue of **Uncle John's Bathroom Reader**, now go wash your hands.

`Gemfile.lock`:

When you first run **`bundle install`**, Bundler will create a **`Gemfile.lock`**. This file contains the specific versions of all the gems resolved when Bundler ran. Future invocations of **`bundle install`** will use these exact same versions. This ensures all environments are running the same gems; for apps which are deployed, the Gemfile.lock should be committed to the repository.

To update a gem, executing **`bundle update mygem`** will resolve the dependencies, install **`mygem`** and only update the Gemfile.lock.

Because of the Gemfile.lock, it's **usually** unnecessary to pin specific versions of gems in the **`Gemfile`** itself.

`Gemfile Groups`:

Bundler allows the specification of "groups" inside of the Gemfile. Consider these syntax sugar. When **`bundle install`** is run, *all* gems must be available, otherwise the command will fail.

Wanted:

Would you consider yourself an expert on error handling in Java or Ruby? We'd love to see an issue on the gotchas surrounding exceptions in both languages! Contact **ujbr@**lookout.com if you're interested in writing on the subject!

Full Gemfile documentation:
http://gembundler.com/man/gemfile.5.html

Gem Specification reference:
http://docs.rubygems.org/read/chapter/20

Send comments and suggestions for future issues to: **ujbr@lookout.com**