Fork me on GitHub

Sinatra Recipes

Community contributed recipes and techniques

Select a topic ▾

Development ¶ ↑

Here we will cover utilities and techniques best used when developing your Sinatra application, such as organizing your application or using Bundler for development.

- Better Errors
- Bundler
- I18n
- Shotgun

Did we miss something?

It's very possible we've left something out, that's why we need your help!This is a community driven project after all. Feel free to fork the project and send us a pull request to get your recipe or tutorial included in the book.

See the README for more details.

Top

Sinatra Recipes

Community contributed recipes and techniques

Fork me on GitHub

Select a topic

Chapters

Bundler ¶ ↑

Whether you need a specific gem and version, or a set of gems for a certain environment; Bundler is a fantastic tool for managing your applications dependencies.

Gemfiles ¶ ↑

Gemfiles are the source of your bundle, used by bundler to determine what gems to install and require in different situations. It's important to understand the difference between the Gemfile and Gemfile.lock; Gemfiles are where you specify the actual gems required by your application, and the Gemfile.lock is a definition of all the required gems and the exact versions used by your application. As it's necessary for other developers to know exactly what versions of third party libraries you're using, the Gemfile.lock is recommended to be checked into source control.

Gemcutter

In most cases you're going to require gems from the official rubygems repository. Here's an example Gemfile for an application that uses Sinatra as a main dependency and RSpec for testing:

```
# define our source to loook for gems
source "http://rubygems.org/"

# declare the sinatra dependency
gem "sinatra"

# setup our test group and require rspec
```

```
group :test do
  gem "rspec"
end

# require a relative gem version
gem "i18n", "~> 0.4.1"
```

Git

Bundler also supports the installation of gems through git, so long as the repository contains a valid gemspec for the gem you're trying to install.

```
# lets use sinatra edge
gem "sinatra", :git => "http://github.com/sinatra/sinatra.git"

# and lets we use the rspec 2.0 release candidate from git
group :test do
  gem "rspec", :git => "http://github.com/rspec/rspec.git",
    :tag => "v2.0.0.rc"
end

# as well as i18n from git
gem "i18n", :git => "http://github.com/svenfuchs/i18n.git"
```

Commands (CLI)¶ ↑

Bundle is the command line utility provided with Bundler to install, update and manage your bundle. Here's a quick overview of some of the most common commands.

Installing

```
# Install specified gems from your Gemfile and Gemfile.lock
bundle install
```

```
# Inspect your bundle to see if you've met your applications requirements
bundle check
```

```
# List all gems in your bundle
bundle list
```

```
# Show source location of a specific gem in your bundle
bundle show [gemname]
```

```
# Generate a skeleton Gemfile to start your path to using Bundler
bundle init
```

Updating

Updating your bundle will look in the given repositories for the latest versions available. This will bypass your

Gemfile.lock and check for completely new versions. Alternatively you can specify an individual gem to update, and bundler will only update that gem to the latest version available in the specified repository.

```
# Update all gems specified to the latest versions available
bundle update
```

```
# Update just i18n to the latest gem version available
bundle update i18n
```

Requiring

Bundler provides two main ways to use your bundle in your application, `Bundler.setup` and `Bundler.require`. `Setup` basically tells Ruby all of your gems loadpaths, and `require` will load all of your specified gems.

Requiring `bundler/setup` is the same as calling `Bundler.setup` yourself, and is the recommended method in the gembundler documentation.

```ruby
# If you're using Ruby 1.9 you'll need to specifically load rubygems
require 'rubygems'

# and now load bundler with your dependencies load paths
require 'bundler/setup'

# next you'll have to do the gem requiring yourself
require 'sinatra'
require 'i18n'
```

Now if say you skip the last step, and just auto require gems from your groups

```ruby
require 'rubygems'
require 'bundler/setup'

# this will require all the gems not specified to a given group (default)
# and gems specified in your test group
Bundler.require(:default, :test)
```

Resources ¶ ↑

- Bundler's Purpose and Rationale – For a longer explanation of what Bundler does and how it works
- Gemfile Manual
- CLI Manual – Basic command line utilities provided with Bundler
- Gems from Git repositories – Using git repositories with your Gemfile
- Using Groups – Using groups with bundler
- bundle install manual
- bundle update manual
- bundle package manual
- bundle exec manual
- bundle config manual

Did we miss something?

It's very possible we've left something out, that's why we need your help!This is a community driven project after all. Feel free to fork the project and send us a pull request to get your recipe or tutorial included in the book.

See the README for more details.

Top

Sinatra Recipes

Community contributed recipes and techniques

Fork me on GitHub

Select a topic

Chapters

BetterErrors ¶ ↑

better_errors is a Rack middleware that replaces the standard error page with a more useful one. It provides:

- Full stack trace
- Source code inspection for all stack frames (with highlighting)
- Local and instance variable inspection
- Live REPL on every stack frame

You can install it using the `gem` command:

```
gem install better_errors
```

or with `bundler` (recommended):

```
group :development do
  gem 'better_errors'
  # uncomment this for more advanced features:
  # gem 'binding_of_caller'
end
```

Then, you can use it as follows:

```
require 'sinatra'
require 'better_errors'

# Just in development!
configure :development do
  use BetterErrors::Middleware
```

```
    # you need to set the application root in order to abbreviate filenames
    # within the application:
    BetterErrors.application_root = File.expand_path('..', __FILE__)
  end

  get '/' do
    raise 'Oops! See you at the better_errors error page!'
  end
```

More information¶ ↑

For more information please look at the better_errors README.

Did we miss something?

It's very possible we've left something out, that's why we need your help!This is a community driven project after all. Feel free to fork the project and send us a pull request to get your recipe or tutorial included in the book.

See the README for more details.

Top

Fork me on GitHub

Sinatra Recipes

Community contributed recipes and techniques

Select a topic

Chapters

Bundler ¶ ↑

Whether you need a specific gem and version, or a set of gems for a certain environment; Bundler is a fantastic tool for managing your applications dependencies.

Gemfiles ¶ ↑

Gemfiles are the source of your bundle, used by bundler to determine what gems to install and require in different situations. It's important to understand the difference between the Gemfile and Gemfile.lock; Gemfiles are where you specify the actual gems required by your application, and the Gemfile.lock is a definition of all the required gems and the exact versions used by your application. As it's necessary for other developers to know exactly what versions of third party libraries you're using, the Gemfile.lock is recommended to be checked into source control.

Gemcutter

In most cases you’re going to require gems from the official rubygems repository. Here’s an example Gemfile for an application that uses Sinatra as a main dependency and RSpec for testing:

```
# define our source to loook for gems
source "http://rubygems.org/"

# declare the sinatra dependency
gem "sinatra"

# setup our test group and require rspec
```

```
group :test do
  gem "rspec"
end

# require a relative gem version
gem "i18n", "~> 0.4.1"
```

Git

Bundler also supports the installation of gems through git, so long as the repository contains a valid gemspec for the gem you're trying to install.

```
# lets use sinatra edge
gem "sinatra", :git => "http://github.com/sinatra/sinatra.git"

# and lets we use the rspec 2.0 release candidate from git
group :test do
  gem "rspec", :git => "http://github.com/rspec/rspec.git",
    :tag => "v2.0.0.rc"
end

# as well as i18n from git
gem "i18n", :git => "http://github.com/svenfuchs/i18n.git"
```

Commands (CLI)¶ ↑

Bundle is the command line utility provided with Bundler to install, update and manage your bundle. Here's a quick overview of some of the most common commands.

Installing

```
# Install specified gems from your Gemfile and Gemfile.lock
bundle install
```

```
# Inspect your bundle to see if you've met your applications requirements
bundle check
```

```
# List all gems in your bundle
bundle list
```

```
# Show source location of a specific gem in your bundle
bundle show [gemname]
```

```
# Generate a skeleton Gemfile to start your path to using Bundler
bundle init
```

Updating

Updating your bundle will look in the given repositories for the latest versions available. This will bypass your

Gemfile.lock and check for completely new versions. Alternatively you can specify an individual gem to update, and bundler will only update that gem to the latest version available in the specified repository.

```
# Update all gems specified to the latest versions available
bundle update
```

```
# Update just i18n to the latest gem version available
bundle update i18n
```

Requiring

Bundler provides two main ways to use your bundle in your application, `Bundler.setup` and `Bundler.require`. `Setup` basically tells Ruby all of your gems loadpaths, and `require` will load all of your specified gems.

Requiring `bundler/setup` is the same as calling `Bundler.setup` yourself, and is the recommended method in the gembundler documentation.

```
# If you're using Ruby 1.9 you'll need to specifically load rubygems
require 'rubygems'

# and now load bundler with your dependencies load paths
require 'bundler/setup'

# next you'll have to do the gem requiring yourself
require 'sinatra'
require 'i18n'
```

Now if say you skip the last step, and just auto require gems from your groups

```
require 'rubygems'
require 'bundler/setup'

# this will require all the gems not specified to a given group (default)
# and gems specified in your test group
Bundler.require(:default, :test)
```

Resources ¶ ↑

- Bundler's Purpose and Rationale – For a longer explanation of what Bundler does and how it works
- Gemfile Manual
- CLI Manual – Basic command line utilities provided with Bundler
- Gems from Git repositories – Using git repositories with your Gemfile
- Using Groups – Using groups with bundler
- bundle install manual
- bundle update manual
- bundle package manual
- bundle exec manual
- bundle config manual

Did we miss something?

It's very possible we've left something out, that's why we need your help!This is a community driven project after all. Feel free to fork the project and send us a pull request to get your recipe or tutorial included in the book.

See the README for more details.

Top

Sinatra Recipes

Community contributed recipes and techniques

Fork me on GitHub

Select a topic

Chapters

How can I internationalize my application? {#i18n}¶ ↑

We will rely on the `i18n` gem to handle internationalisation of strings and objects, and to manage fallbacks on available locales

```
require 'i18n'
require 'i18n/backend/fallbacks'
```

The following configuration is necessary on I18n so that:

- it can fallback on other locales if the requested one is not available (ie: translation does not exist).
- all the translations are read from YAML files located in the `locales` directory

```
configure
  I18n::Backend::Simple.send(:include, I18n::Backend::Fallbacks)
  I18n.load_path, Dir[File.join(settings.root, 'locales', '*.yml')]
  I18n.backend.load_translations
end
```

Now we need to choose the locale that the user wants. There are several solutions (and some can even be mixed together): browser preference, specific URL, dedicated subdomain, cookies/session management. Only the first three will be shown below:

Browser preference (requir`rack-contrib`) ¶ ↑

```
use Rack::Locale
```

Specific URL¶ ↑

```
before '/:locale/*' do
  I18n.locale       =       params[:locale]
  request.path_info = '/' + params[:splat ][0]
end
```

Dedicated subdomain¶ ↑

```
before do
  if (locale = request.host.split('.')[0]) != 'www'
    I18n.locale = locale
  end
end
```

We have all the necessary information to deliver to the user texts/pages in their native language. And for that we will need to select strings and templates according to the desired locale.

Selection of localized strings/objects is easy as it only requires use of standard methods from `I18n`

```
I18n.t(:token)
I18n.l(Time.now)
```

For rendering the templates matching the desired locale, we need to extend the `find_template` method. It needs to select the first template matching the user locale (or at least one acceptable fallback). To help in the selection process templates stored in the `views` directory are suffixed by the name of the locale.

```
helpers do
  def find_template(views, name, engine, &block)
    I18n.fallbacks[I18n.locale].each { |locale|
      super(views, "#{name}.#{locale}", engine, &block) }
    super(views, name, engine, &block)
  end
end
```

Did we miss something?

It's very possible we've left something out, that's why we need your help!This is a community driven project after all. Feel free to fork the project and send us a pull request to get your recipe or tutorial included in the book.

See the README for more details.

Top

Fork me on GitHub

Sinatra Recipes

Community contributed recipes and techniques

Select a topic

Chapters

Shotgun ¶ ↑

Shotgun will actually restart your application on every request. This has the advantage over other reloading techniques of always producing correct results. However, since it actually restarts your application, it is rather slow compared to the alternatives. Moreover, since it relies on `fork`, it is not available on Windows and JRuby.

Usage is rather simple:

```
gem install shotgun # run only once, to install shotgun
shotgun my_app.rb
```

If you want to run a modular application, create a file named `config.ru` with similar content:

```
require File.expand_path '../my_app.rb', __FILE__
run MyApp
```

And run it by calling `shotgun` without arguments.

The `shotgun` executable takes arguments similar to those of the `rackup` command, run `shotgun --help` for more information.

Did we miss something?

It's very possible we've left something out, that's why we need your help!This is a community driven project after all. Feel free to fork the project and send us a pull request to get your recipe or tutorial included in the book.

See the README for more details.

Top