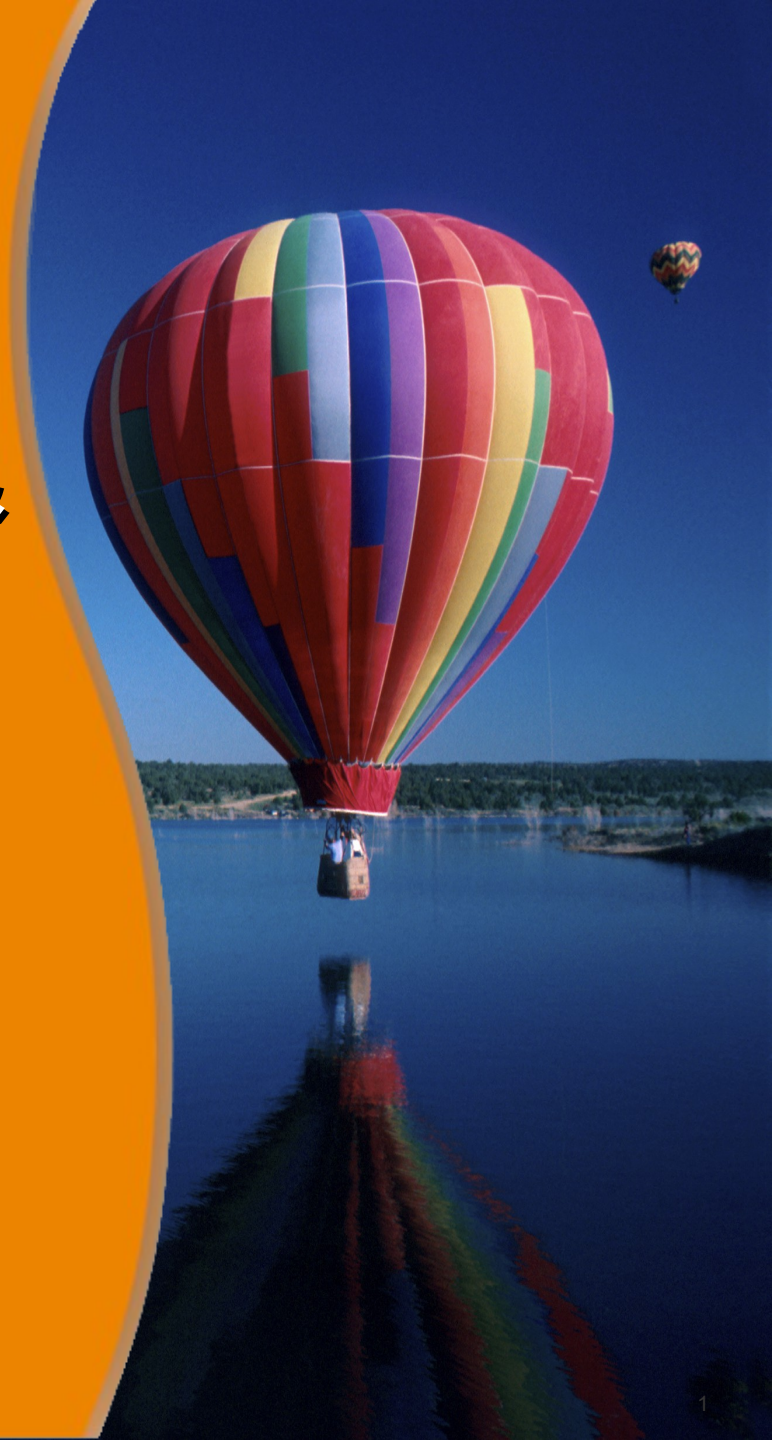
Rails Environment & Configuration



Sang Shin
JPassion.com
"Code with Passion!"

Topics

- Gem and RubyGem
- Gemfile and Gemfile.lock
- Bundler
- Environments
- Configuration
- Initializers
- Rake

Gem & RubyGem (per system)

What is Gem and RubyGem?

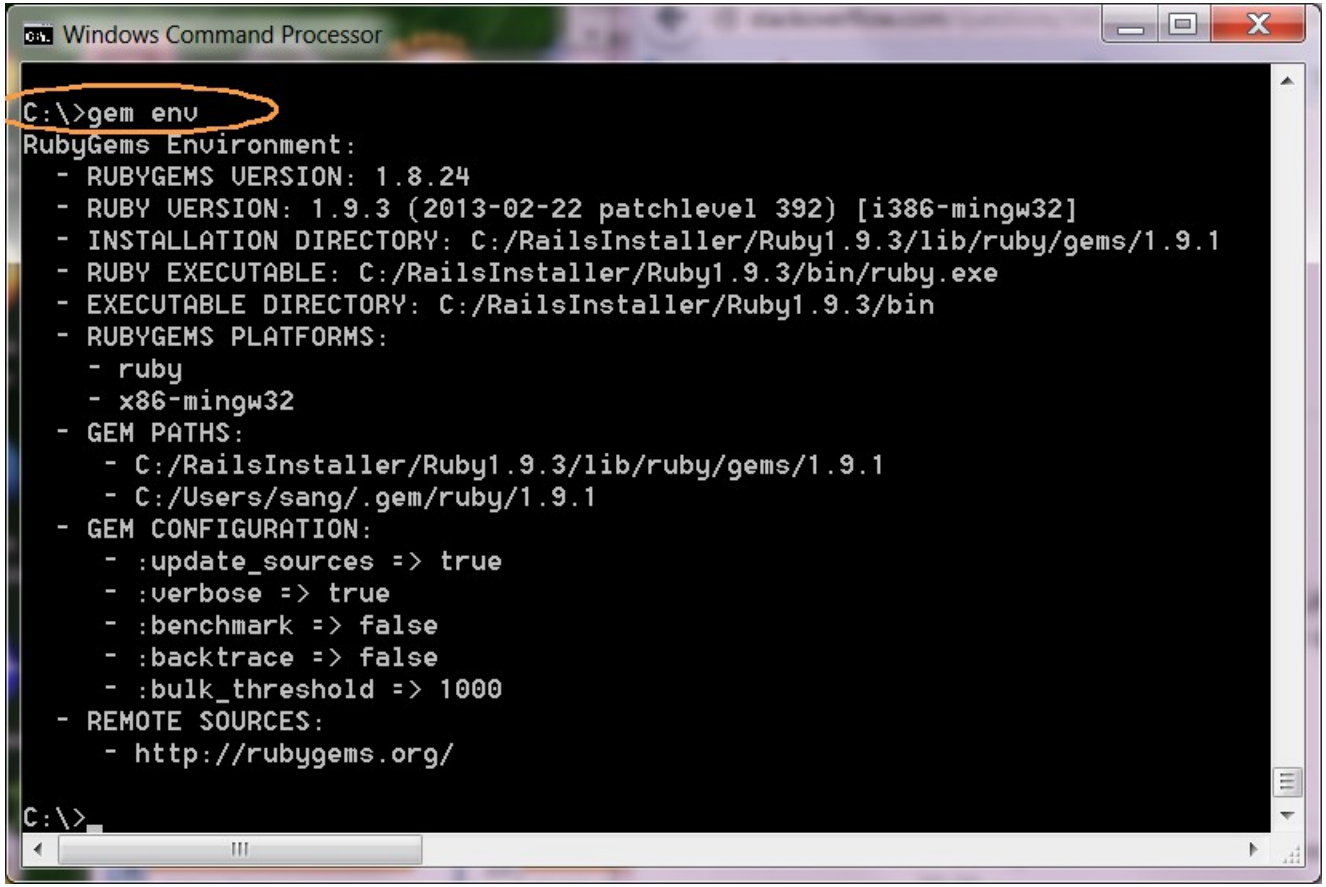
- Gem is a standard package format for Ruby libraries
  - In the same way “jar” and “war” are standard package formats for Java libraries
- In Ruby/Rails world, a feature is built as a gem and there are tons of 3rd-party gems you can use right away
  - Security, testing, performance, ...
- You can create your own custom gems or extend existing gems
- RubyGem is package manager for the Ruby programming language
  - Manage the installation/uninstallation of gems (per your system)
  - RubyGem comes with Ruby 1.9+
  - You interact with RubyGem with “gem” command

What about Rails?

- Rails itself is a gem
  - With dependencies on many other gems
- Special gems – gems that provide commands that can be executed at the command line
  - rails
  - bundler
  - rake

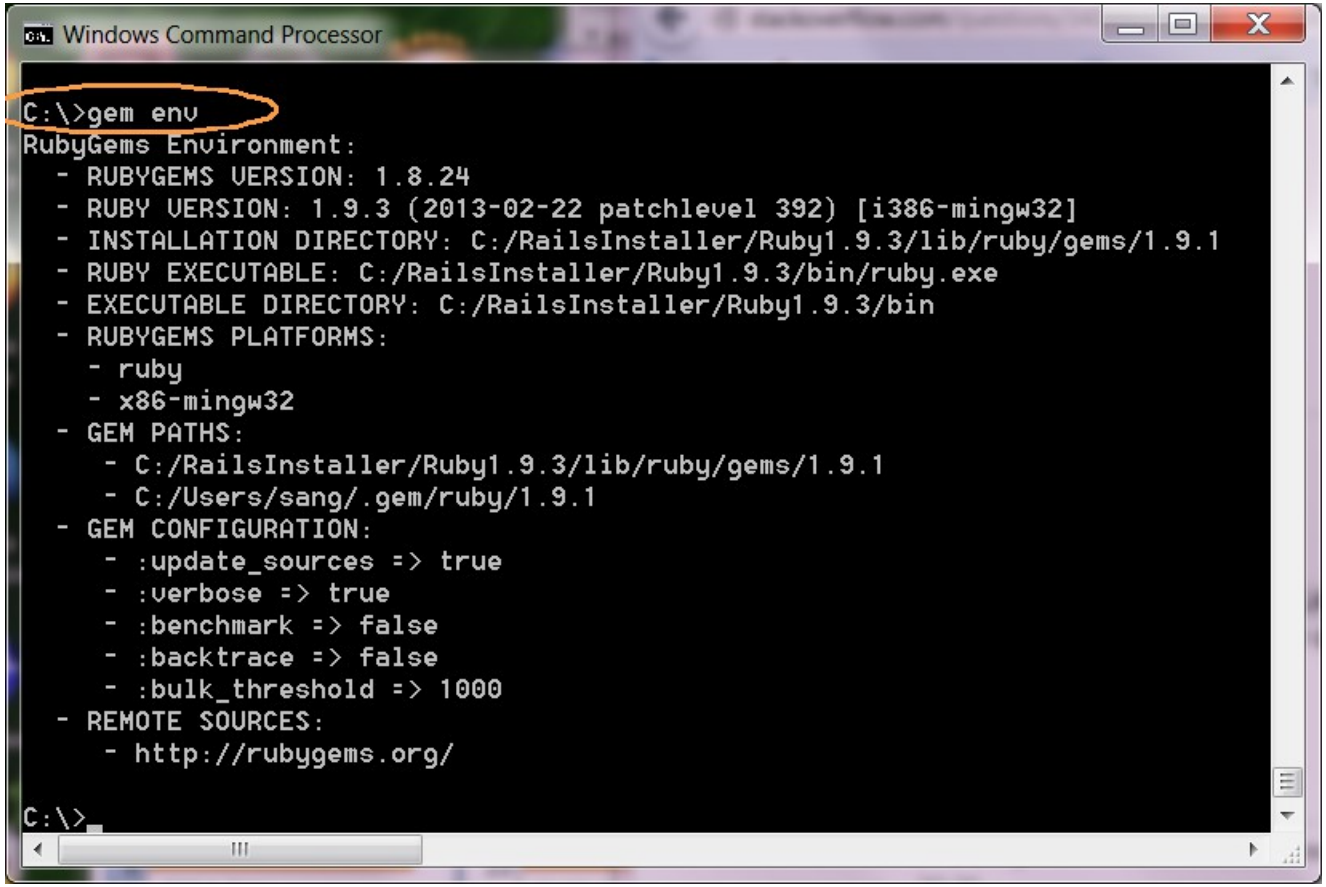
Gem Environment

- “gem env”



Gem Environment

- “gem env”



“gem” commands (1)

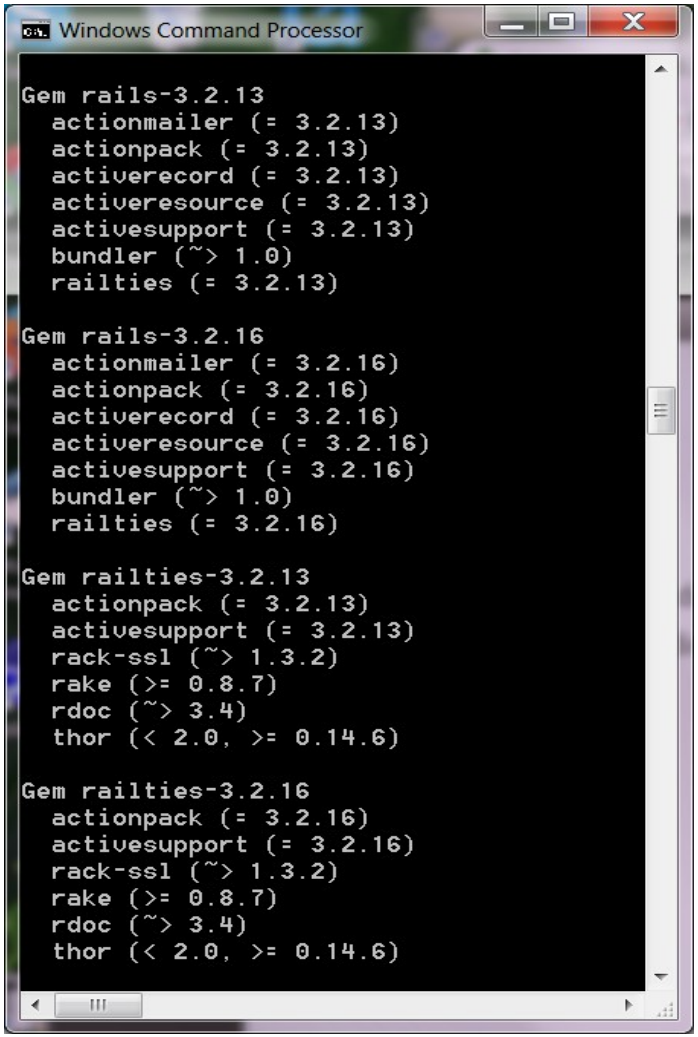
- “gem env”
  - Display gem environment information
- “gem install <gem-name>”
  - Installs <gem-name> gem
  - In general, for a particular Rails app, you will want to use “bundle install” with “Gemfile” to install all required gems in a single command instead of installing each gem individually using “gem install ..” command
- “gem uninstall <gem-name>”
  - Uninstalls <gem-name> gem

“gem” commands (2)

- “gem list --local”
  - List gems that are previously downloaded
- “gem list –remote”
  - List available gems from “rubygems.org” website
- “gem search <gem-name> --remote”
  - Search for <gem-name>
- “gem dependency”
  - Show you all gems with their dependencies
- “gem dependency -R”
  - Show you all gems with their dependencies
  - Also shows gems with reverse dependencies (which gems have dependency on the gems)

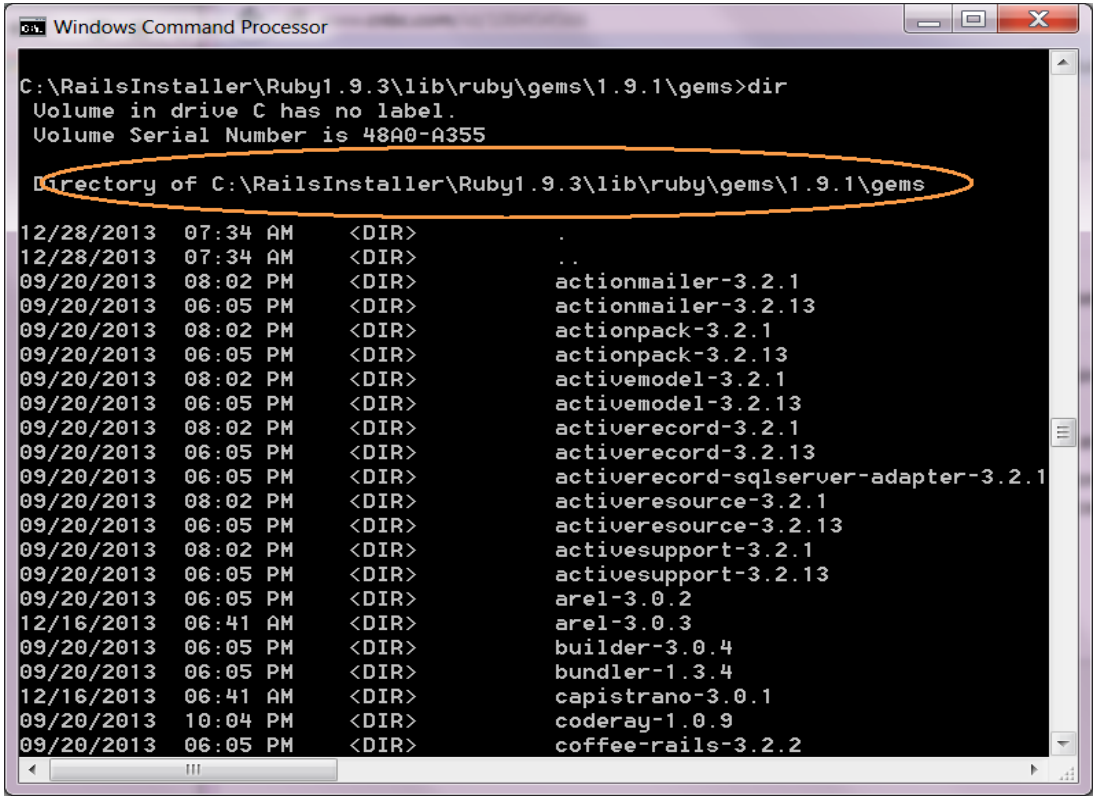
“rails” gem & its dependencies

- “gem dependency”



Where Gems are stored?

- <Ruby-Home>/lib/ruby/gems/<version>/gems
- <User-Home>/.gem/ruby/1.9.1



3rd Party Gems

- http://rubygems.org/
  - Default Ruby gems host (4300 gems hosted Jan. 2014)
- https://www.ruby-toolbox.com/
  - Show available gems based on categories
  - Gems in the same category can be listed in the order of their popularity

Lab:

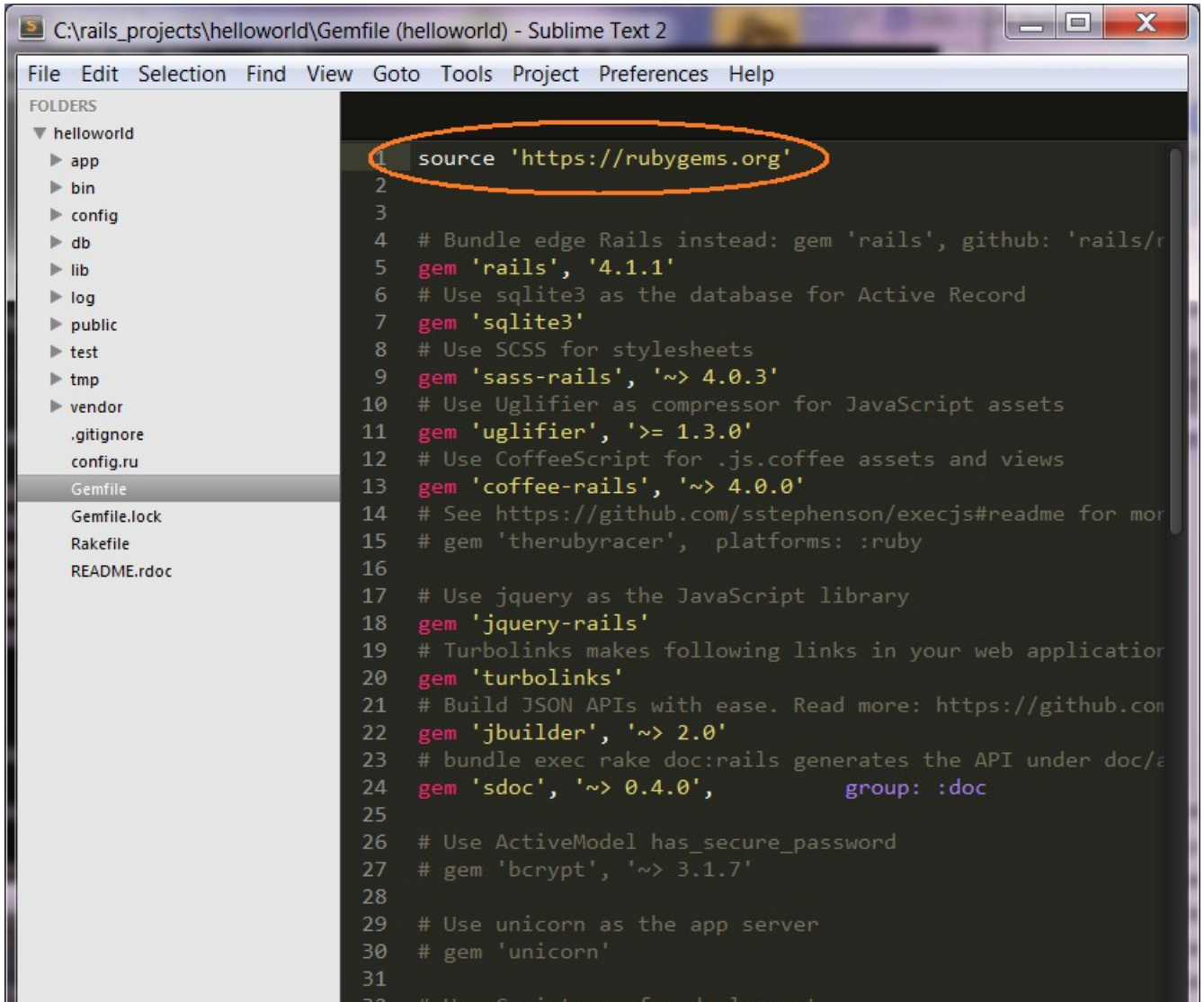
Exercise 1: Gem & RubyGem 5524_rails4_configuration.zip

Gemfile & Gemfile.lock (per each Rails application)

What is “Gemfile”?

- Specifies all the gems (in other words, dependencies) required for a Rails app
- When you create a new Rails app through “rails new <appname>”, Rails creates a *Gemfile* for you
- You can then add, remove, update, and group gems by modifying *Gemfile*
  - For example, if you need a security gem, you add it to the Gemfile
- Every time a modification is made to the *Gemfile*, “bundle install” or “bundle update” needs to be executed
  - “bundle install” or “bundle update” will download and install a newly added gem, for example
  - At the end of “rails new <new-app>”, “bundle install” gets executed by default

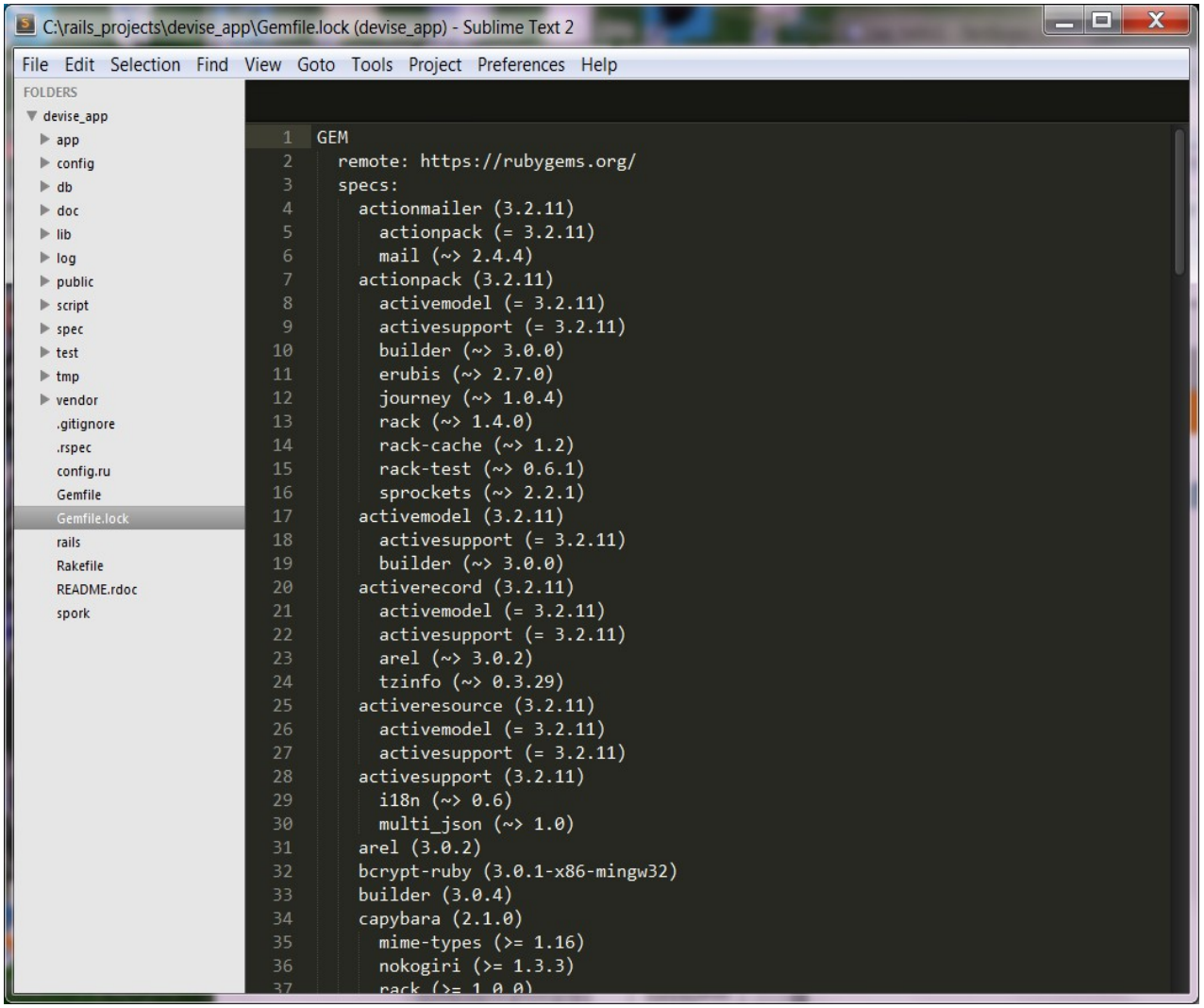
“Gemfile” Example



What is Gemfile.lock?

- When you run “bundle install”, Bundler will create *Gemfile.lock* if it does not exist already
  - *Gemfile.lock* specifies all gems used in the application along with their versions
- Bundler uses this file in all subsequent 'bundle install”, which guarantees that you always use the same exact version of gems, even as your application moves across machines
- You SHOULD check your *Gemfile.lock* into version control
  - If you do not, meaning if there is no Gemfile.lock, every machine that checks out your repository (including your production server) will resolve all dependencies again, which could result in different versions of third-party code being used if any of the gems in the *Gemfile* or any of their dependencies have been updated

Gemfile.lock Example



Bundler

What is a Bundler?

- Handles dependency management in Rails 3 and Rails 4 applications
  - > Performs dependency resolution on the complete list of gems specified in the Gemfile “all at once”
  - > Solves the “dependency conflict found too late” problem of Rails 2 (In Rails 2, dependency is resolved “one at a time”)

Bundler commands (used with app)

- “bundle install” (or just “bundle”)
  - Install newly added gems specified in the “Gemfile”
- What it does
  - If this is the first time you run “bundle install” (and a “Gemfile.lock” does not exist), bundler will fetch all gems, resolve dependencies, and install them locally, then creates “Gemfile.lock”
  - If “Gemfile.lock” does exist, and you have not updated your “Gemfile”, bundler use the gems specified in the “Gemfile.lock”
  - If “Gemfile.lock” does exist, and you have updated your “Gemfile”, bundler will use the dependencies in the “Gemfile.lock” for all gems that you did not update, but will re-resolve the dependencies of gems that you did update.

Bundler commands (used with app)

- “bundle show [gemname]”
  - Displays where a bundled gem is installed
- “bundle update”
  - Install newly added gems in the Gemfile (same as “bundle install”)
  - If existing gems don't have version, it will upgrade to whatever latest (difference from “bundle install”)
  - If existing gems have version controlled with ~>, it will upgrade to the latest at the final digit, the patch version (difference from “bundle install”)
    - 'my_gem', '~> 2.1.0' - “bundle update” will check if newer version of 2.1.x is available

Bundler commands (used with app)

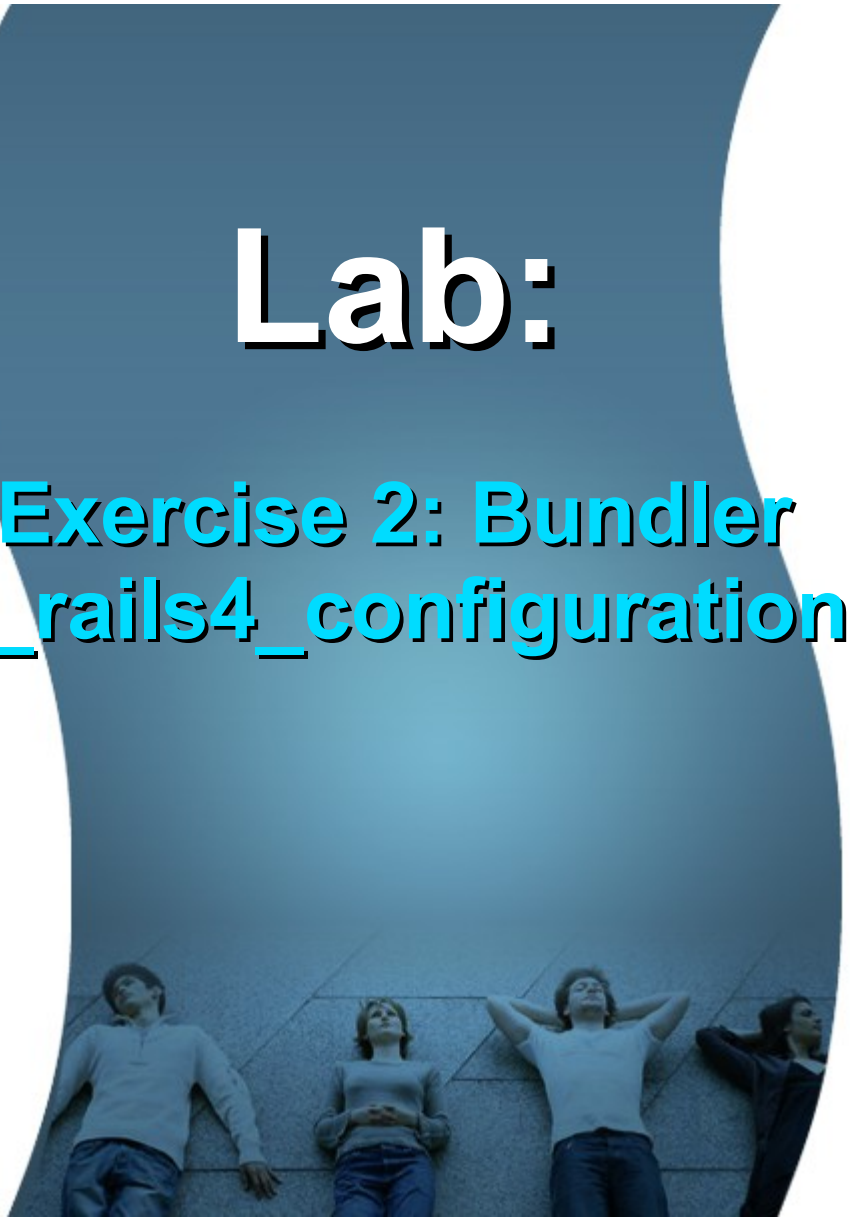
- “bundle package”
  - Package up all gems in the “vendor/cache” directory inside app
  - “bundle install” will use gems in the package instead of “rubygems.org”
  - Used to avoid external dependencies at deploy time
- “bundle outdated”
  - Show all of the outdated gems in the current bundle

“bundle exec <rails command>”

- “bundle exec <rails command>” executes a command in the context of your bundle
  - Uses the gem versions specified in your “Gemfile” file (instead of the latest gem version installed in the system)
  - Most of the time, running “bundle exec <command>” has the same results as if you just ran “<command>”, especially if you have the same gems installed system wide as in your Gemfile file
  - Using “bundle exec <command>” guarantees that the program is run with the environment specified in the Gemfile, which hopefully means it is the environment that the creators of the program want it to be run in, which hopefully means it should run correctly no matter what weird setup you have on your computer
- Example
  - bundle exec rake --tasks

Lab:

Exercise 2: Bundler
5524_rails4_configuration.zip

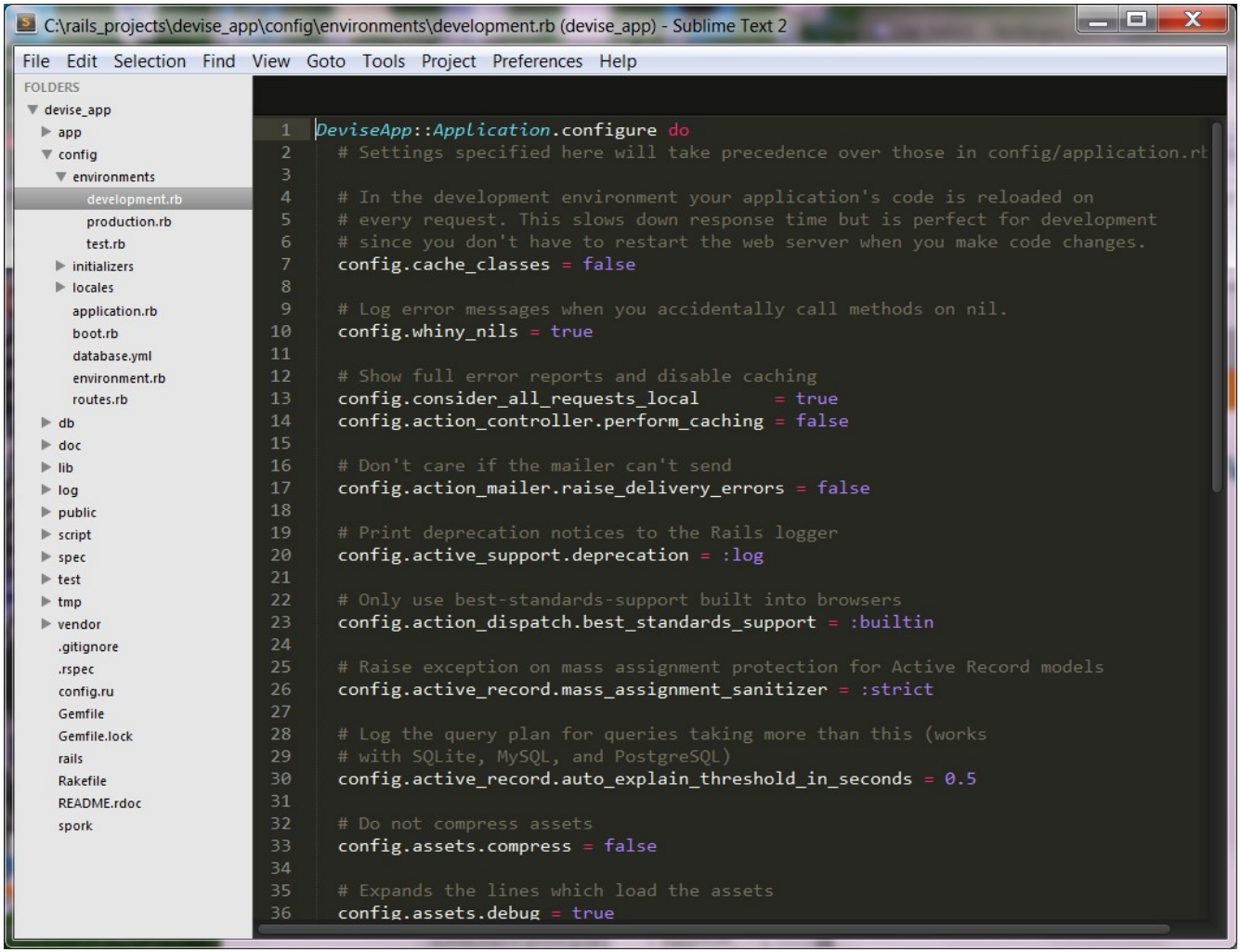


Environments

Environments

- By default Rails ships with three environments:
  - "development"
  - "test"
  - "production"
- You can create your own custom environment, for example, “staging”
  - Create a file called *config/environments/staging.rb*
- Start rails app with your custom environment
  - *rails server -e staging*
- Start Rails console with your custom environment
  - *rails console staging*

Development Environment



Lab:

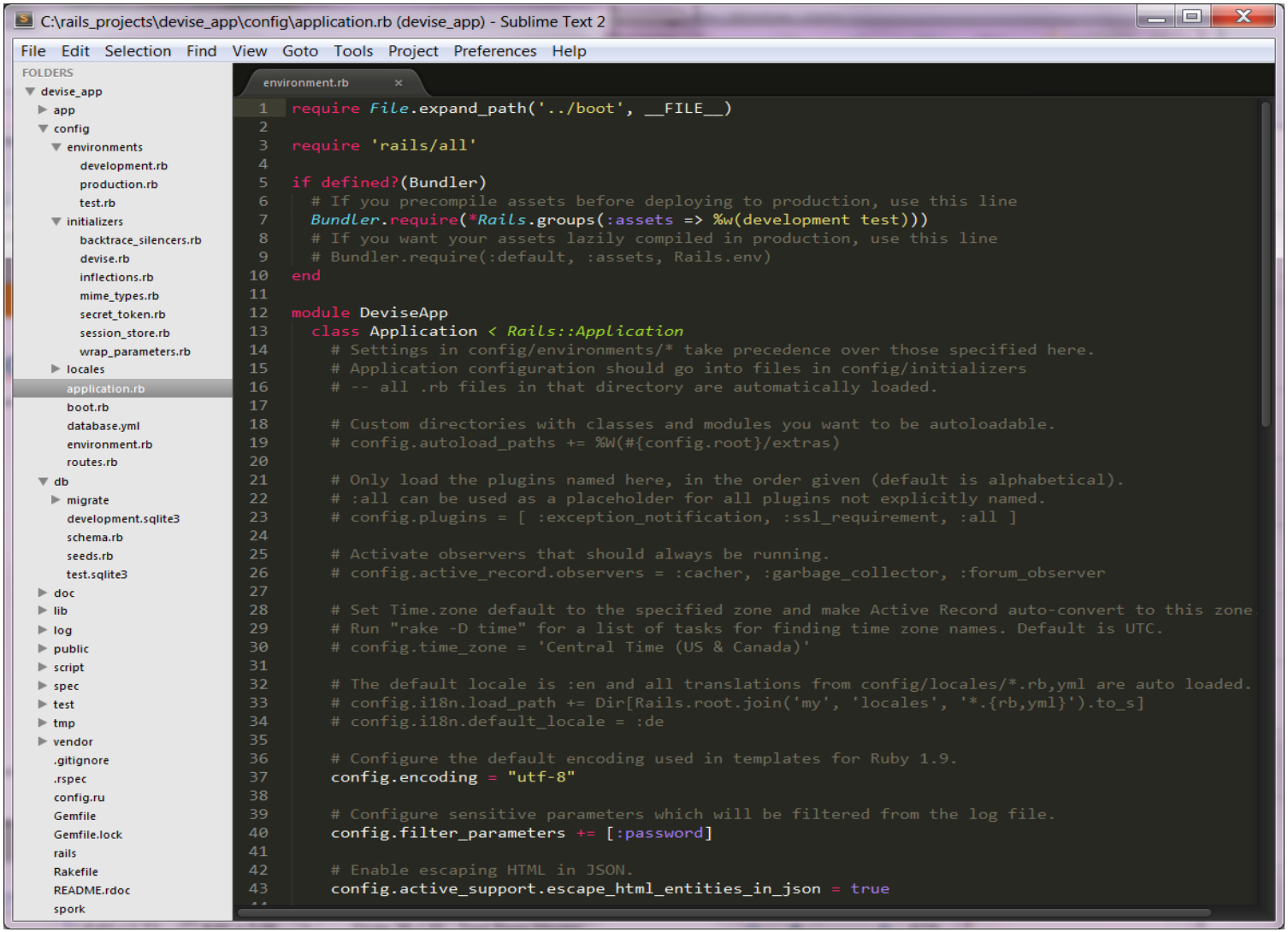
Exercise 3: Environments 5524_rails4_configuration.zip

Configuration

Location for Initialzation Code

- Rails offers four standard spots to place initialization code:
  - config/application.rb
  - Environment-specific configuration files
  - Initializers
  - After-initializers

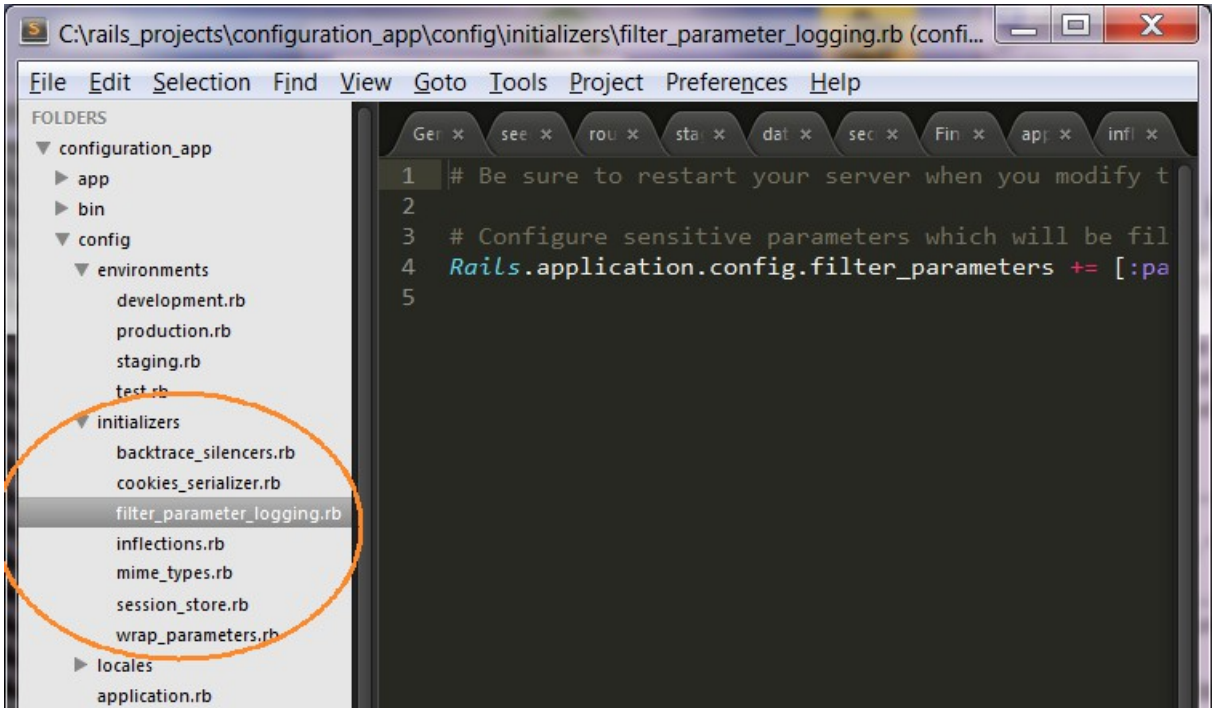
config/application.rb



Initializers

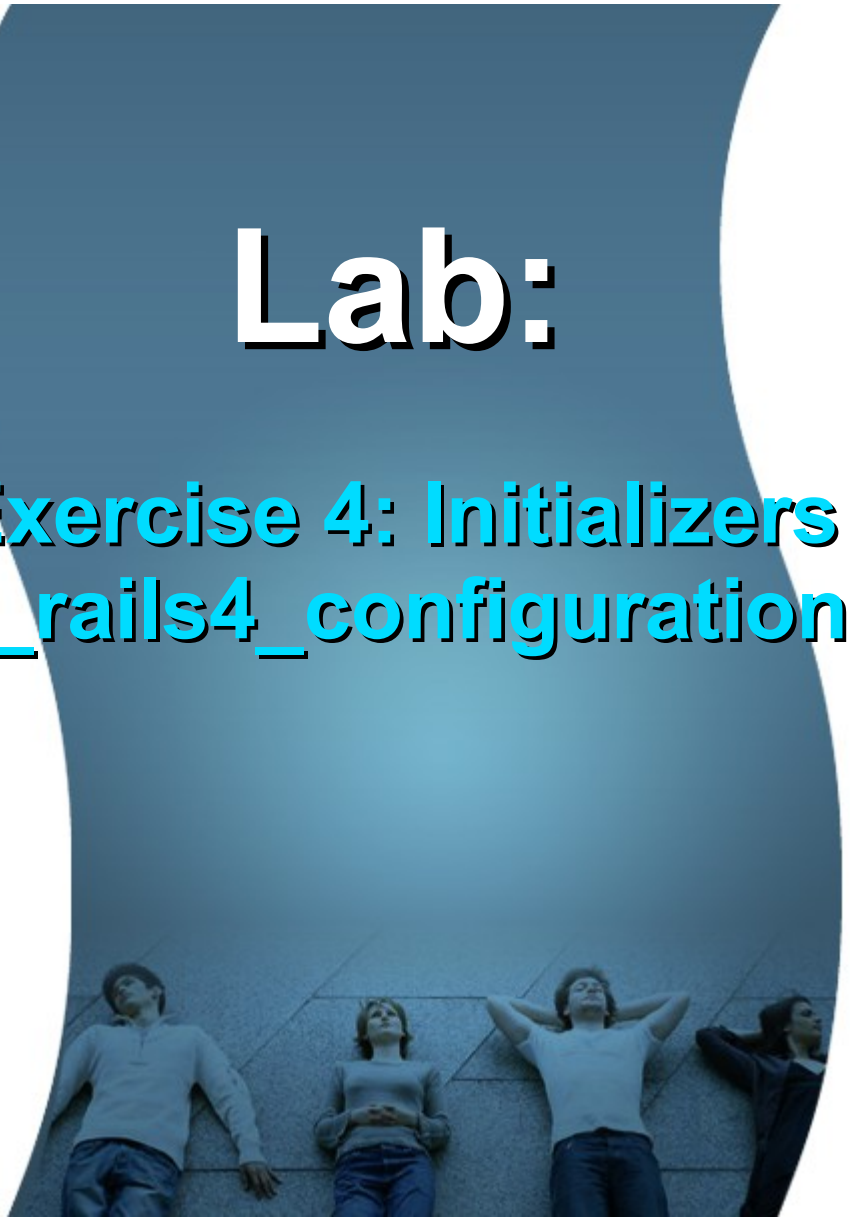
Initializer Files

- After loading the framework and any gems in your application, Rails turns to loading initializers
- An initializer is any Ruby file stored under *config/initializers* in your application – you can add your own initializers here



Lab:

Exercise 4: Initializers
5524_rails4_configuration.zip

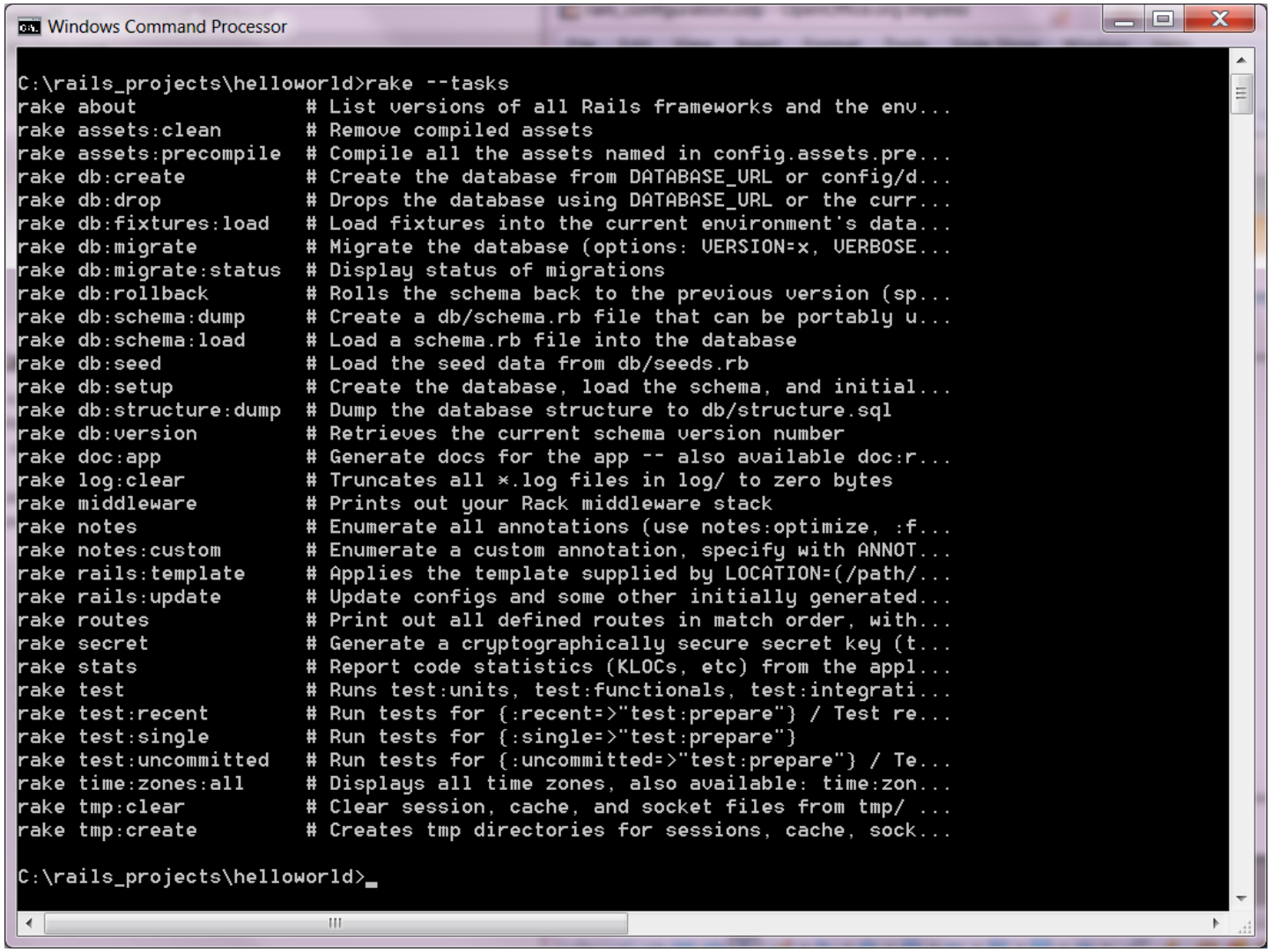


Rake

What is “Rake”?

- Rake is Ruby Make, a standalone Ruby utility that replaces the Unix utility 'make', and uses a '*Rakefile*' and *.rake* files to build up a list of tasks
- In Rails, Rake is used for common administration tasks, especially sophisticated ones that build off of each other

Ready to use “Rake”tasks



Creating Custom Rake Tasks

- Custom rake tasks have a .rake extension and are placed in ./lib/tasks directory

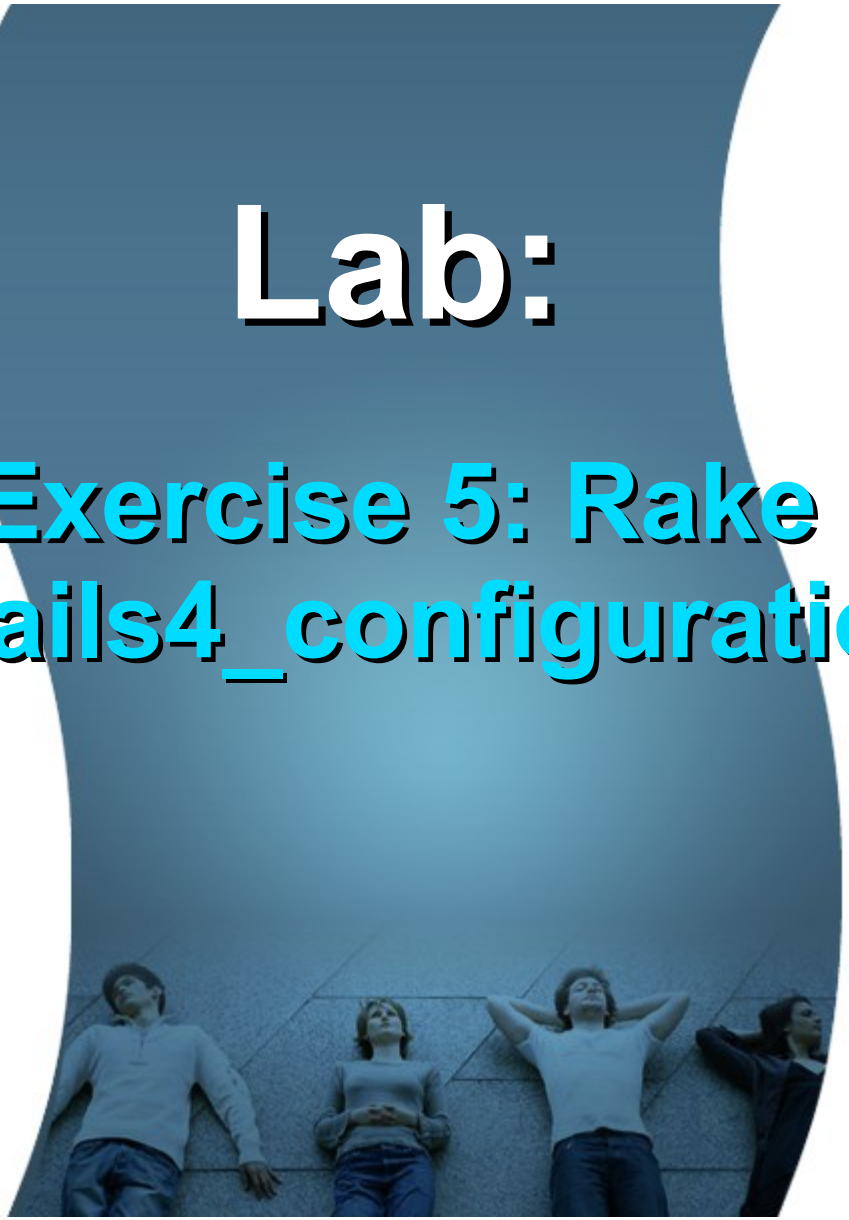
```
namespace :greeting do
  desc "Some greeting"
  task :say_hello do
    puts "Hello, JPassion.com!"
  end
  task :say_goodbye do
    puts "Goodbye, JPassion.com!!"
  end
end
```

- Then you can invoke the task

```
rake greeting:say_hello
rake greeting:say_goodbye
```

Lab:

Exercise 5: Rake
5553_rails4_configuration.zip



Code with Passion!
JPassion.com