Chapter 3 Rails Tutorial Notes - Mostly Static Pages

3.1 Setup

These steps are used when configuring fresh apps:

- To create the new app, we use the command *rails _<version#>_ new <appname>*
- Switch to the new app's directory with the cd command: *cd <appname>/*
- Update the Gemfile with any gems needed by our app (Listing 3.2)
- Install & include our specified gems with *bundle install --without production*

Note: *bundle update* can be used to get the latest version of your previously used gems.

- Initialize our git repository: *git init, git add -A, git commit -m "Init repo"*
- Suggestion: pdate the contents of README.md (Listing 3.3)
- Create a new repository and push it for the first time:
    - *git remote add origin git@bitbucket.org:<username>/sample_app.git*
    - *git push -u origin --all*
- Add a hello action to the App controller, and set the route route (Listings 3.4-3.5)

3.2 Static Pages

Goal: Create a set of actions and views that comprise of static HTML.

- Checkout a topic branch: *git checkout -b static-pages*

Create a StaticPages controller using the command in Listing 3.6.
Box 3.1 outlines a few useful ways of undoing unwanted actions in Ruby.
Start up a rails server with *rails server -b $IP -p $PORT* and check the URL /static_pages/home to make sure that you can view the raw home view.
The home view action is empty so it goes straight to rendering the view, which can be observed at app/views/static_pages/home.html.erb.
These static pages can be customized with no knowledge of Rails (Listings 3.11-3.12)

3.3 Getting Started with Testing

Reasons that testing can be helpful include:

- Regression prevention - something that was working before suddenly is broken.
- Code refactoring - structural changes to code can be made without affecting function.
- Client simulation - developers act as client, which helps determine the app's design and interface.

Box 3.3 points out some guidelines on when to test. It is suggested to do controller and model tests first before doing any testing across models/views/controllers.
The Rails scaffolding automatically creates test files for us.
To run our test suite in order to verify these tests pass, use the command *rails test*.
The main testing cycle is:

- (RED) writing a failing test first.
- (GREEN) writing app code to get it to pass our test(s).
- (Refactor) refactoring if needed to clean up code.

Green means passing whereas red means failing (error messages will appear in red)
The results of failed tests often give detailed messages which may contain fix instructions or can be googled to fix.

3.4 Slightly Dynamic Pages

Goal: Make the Home/Help/About pages have page titles that change per page using the testing cycle.
(Red) Start by making tests for page titles. The *assert_select* method can test for the presence of types of html tags; these are put in the controller test for each page's test. Once it's clear the tests fail, put title elements with content into all three pages (Listings 3.26-3.28)
The *setup* method is handy for testing since it can eliminate repetition in test files (Listing 3.30)
Don't repeat code! Make sure you DRY(Don't Repeat Yourself)
Embedded Ruby(ERB) is introduced along with the provide function (Listing 3.31)

- *<% ... %>* executes the code inside, while *<%= ... %>* executes it and inserts the result.

Rails comes with a layout file called application.html.erb that can be used as a common template for pages.
Once the <title> is put in /views/layouts/application.html.erb, each of our pages can be cleaned (Listings 3.36-3.38)
These pages now have much less duplication going on.
Change the app's root route from *root 'application#hello'* to *root 'static_pages#home'.*

3.5 Conclusion

Commit the changes made in this chapter, merge them back to master branch, push the code to repo and deploy:
git add -A
git commit -m "Finish static pages"
git checkout master
git merge static-pages
git push
rails test
git push heroku

3.6 Advanced Testing Setup (from tutorial screencast)

Setup consists of 2 main elements:

1. An enhanced pass/fail reporter
2. An automated test runner for detecting changes and running corresponding tests

Changes should be made on master branch: *git checkout master*
Guard is a file system monitoring tool that detects changes and automates test running. Guard will only run tests that are applicable to the files that have changed. Initialize our guard gem with the command *bundle exec guard init.*
Edit the new Guardfile with Listing 3.45 so that Guard will run the right tests at the right times.
To avoid Spring server/Git conflicts, add the spring/ directory to the .gitignore file (Figures 3.9-3.11)
Then modify .gitignore with the code in Listing 3.46.
(Box 3.4) Notes on Unix Processes:

- *ps aux* will show all processes on your system.
- To filter by type, you can pattern-match with a Unix pipe (|). For example, *ps aux | grep spring*
- The results describe characteristics of the processes. The first number in each result is the process' id.
- The kill command for a process with id 12241 would be *kill -15 12241*
- 2 techniques for killing Spring server processes are *spring stop* and *pkill -15 -f spring*

Open a new terminal tab and run *Guard* at the command line with *bundle exec guard.*
To run all tests you can press enter at the guard> prompt. To exit Guard, use ctrl + D.
Lastly, add any changes made and make a commit :
git add -A
git commit -m "Complete advanced setup"