3.5.1 What we learned in this chapter

- For a third time, we went through the full procedure of creating a new Rails application from scratch, installing the necessary gems, pushing it up to a remote repository, and deploying it to production.
- The `rails` script generates a new controller with `rails generate controller ControllerName <optional action names>`.
- New routes are defined in the file `config/routes.rb`.
- Rails views can contain static HTML or embedded Ruby (ERb).
- Automated testing allows us to write test suites that drive the development of new features, allow for confident refactoring, and catch regressions.
- Test-driven development uses a "Red, Green, Refactor" cycle.
- Rails layouts allow the use of a common template for pages in our application, thereby eliminating duplication.

3.6 Advanced testing setup

This optional section describes the testing setup used in the Ruby on Rails Tutorial screencast series. There are two main elements: an enhanced pass/fail reporter (Section 3.6.1), and an automated test runner that detects file changes and automatically runs the corresponding tests (Section 3.6.2). The code in this section is advanced and is presented for convenience only; you are not expected to understand it at this time.

The changes in this section should be made on the master branch:

```
$ git checkout master
```

3.6.1 minitest reporters

Although many systems, including the cloud IDE, will show the appropriate colors for RED and GREEN test suites, adding *minitest reporters* lends a degree of pleasant polish to the test outputs, so I recommend adding the code in Listing 3.46 to your test helper file,[17] thereby making use of the `minitest-reporters` gem included in Listing 3.2.

Listing 3.46: Configuring the tests to show RED and GREEN.
test/test_helper.rb

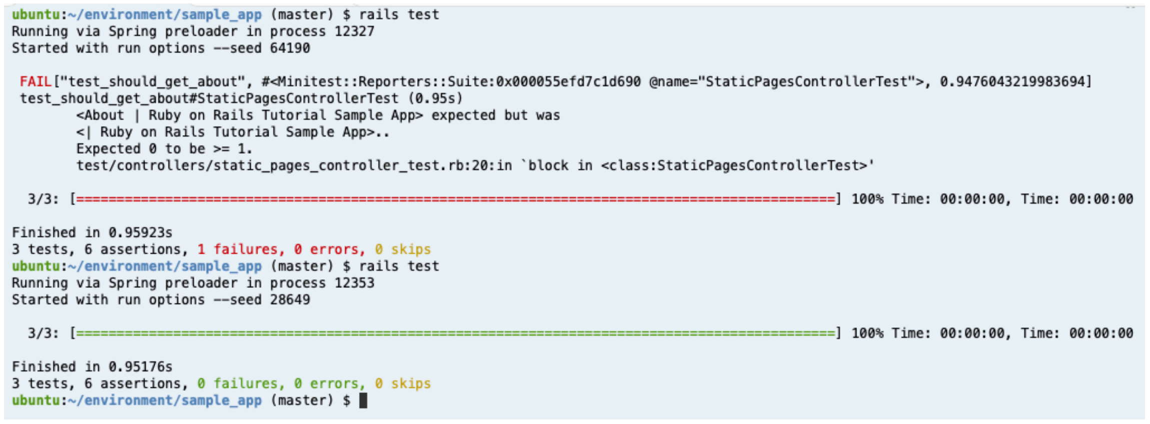
```
ENV['RAILS_ENV'] ||= 'test'
require_relative '../config/environment'
require 'rails/test_help'
require "minitest/reporters"
Minitest::Reporters.use!

class ActiveSupport::TestCase
  # Run tests in parallel with specified workers
  parallelize(workers: :number_of_processors)

  # Setup all fixtures in test/fixtures/*.yml for all tests in alphabetical order.
  fixtures :all

  # Add more helper methods to be used by all tests here...
end
```

The resulting transition from RED to GREEN in the cloud IDE appears as in Figure 3.10.

3.6.2 Automated tests with Guard

One annoyance associated with using the **`rails test`** command is having to switch to the command line and run the tests by hand. To avoid this inconvenience, we can use *Guard* to automate the running of the tests. Guard monitors changes in the filesystem so that, for example, when we change the

[17]The code in Listing 3.46 mixes single- and double-quoted strings. This is because **`rails new`** generates single-quoted strings, whereas the minitest reporters documentation uses double-quoted strings. This mixing of the two string types is common in Ruby; see Section 4.2.1 for more information.



```
ubuntu:~/environment/sample_app (master) $ rails test
Running via Spring preloader in process 12327
Started with run options --seed 64190

 FAIL["test_should_get_about", #<Minitest::Reporters::Suite:0x000055efd7c1d690 @name="StaticPagesControllerTest">, 0.9476043219983694]
 test_should_get_about#StaticPagesControllerTest (0.95s)
        <About | Ruby on Rails Tutorial Sample App> expected but was
        <| Ruby on Rails Tutorial Sample App>..
        Expected 0 to be >= 1.
        test/controllers/static_pages_controller_test.rb:20:in `block in <class:StaticPagesControllerTest>'

  3/3: [============================================================================================] 100% Time: 00:00:00, Time: 00:00:00

Finished in 0.95923s
3 tests, 6 assertions, 1 failures, 0 errors, 0 skips
ubuntu:~/environment/sample_app (master) $ rails test
Running via Spring preloader in process 12353
Started with run options --seed 28649

  3/3: [============================================================================================] 100% Time: 00:00:00, Time: 00:00:00

Finished in 0.95176s
3 tests, 6 assertions, 0 failures, 0 errors, 0 skips
ubuntu:~/environment/sample_app (master) $ ▌
```

Figure 3.10: Going from RED to GREEN in the cloud IDE.

static_pages_controller_test.rb file, only those tests get run. Even better, we can configure Guard so that when, say, the **home.html.erb** file is modified, the **static_pages_controller_test.rb** automatically runs.

The **Gemfile** in Listing 3.2 has already included the `guard` gem in our application, so to get started we just need to initialize it:

```
$ bundle exec guard init
Writing new Guardfile to /home/ec2-user/environment/sample_app/Guardfile
00:51:32 - INFO - minitest guard added to Guardfile, feel free to edit it
```

We then edit the resulting **Guardfile** so that Guard will run the right tests when the integration tests and views are updated, which will look something like Listing 3.47. For maximum flexibility, I recommend using the version of the **Guardfile** listed in the reference application, which if you're reading this online should be identical to Listing 3.47:

- Reference **Guardfile** at railstutorial.org/guardfile

Listing 3.47: A custom **Guardfile**.

```
# Defines the matching rules for Guard.
guard :minitest, spring: "bin/rails test", all_on_start: false do
```

```
  watch(%r{^test/(.*)/?(.*)_test\.rb$})
  watch('test/test_helper.rb') { 'test' }
  watch('config/routes.rb') { interface_tests }
  watch(%r{app/views/layouts/*}) { interface_tests }
  watch(%r{^app/models/(.*?)\.rb$}) do |matches|
    "test/models/#{matches[1]}_test.rb"
  end
  watch(%r{^app/controllers/(.*?)_controller\.rb$}) do |matches|
    resource_tests(matches[1])
  end
  watch(%r{^app/views/([^/]*?)/.*\.html\.erb$}) do |matches|
    ["test/controllers/#{matches[1]}_controller_test.rb"] +
    integration_tests(matches[1])
  end
  watch(%r{^app/helpers/(.*?)_helper\.rb$}) do |matches|
    integration_tests(matches[1])
  end
  watch('app/views/layouts/application.html.erb') do
    'test/integration/site_layout_test.rb'
  end
  watch('app/helpers/sessions_helper.rb') do
    integration_tests << 'test/helpers/sessions_helper_test.rb'
  end
  watch('app/controllers/sessions_controller.rb') do
    ['test/controllers/sessions_controller_test.rb',
     'test/integration/users_login_test.rb']
  end
  watch('app/controllers/account_activations_controller.rb') do
    'test/integration/users_signup_test.rb'
  end
  watch(%r{app/views/users/*}) do
    resource_tests('users') +
    ['test/integration/microposts_interface_test.rb']
  end
end

# Returns the integration tests corresponding to the given resource.
def integration_tests(resource = :all)
  if resource == :all
    Dir["test/integration/*"]
  else
    Dir["test/integration/#{resource}_*.rb"]
  end
end

# Returns all tests that hit the interface.
def interface_tests
  integration_tests << "test/controllers/"
end

# Returns the controller tests corresponding to the given resource.
```

```
def controller_test(resource)
  "test/controllers/#{resource}_controller_test.rb"
end

# Returns all tests for the given resource.
def resource_tests(resource)
  integration_tests(resource) << controller_test(resource)
end
```

On the cloud IDE, there's one additional step, which is to run the following rather obscure commands to allow Guard to monitor all the files in the project:

```
$ echo fs.inotify.max_user_watches=524288 | sudo tee -a /etc/sysctl.conf
$ sudo sysctl -p
```

Once Guard is configured, you should open a new terminal (as with the Rails server in Section 1.2.2) and run it at the command line as follows (Figure 3.11):

```
$ bundle exec guard
```

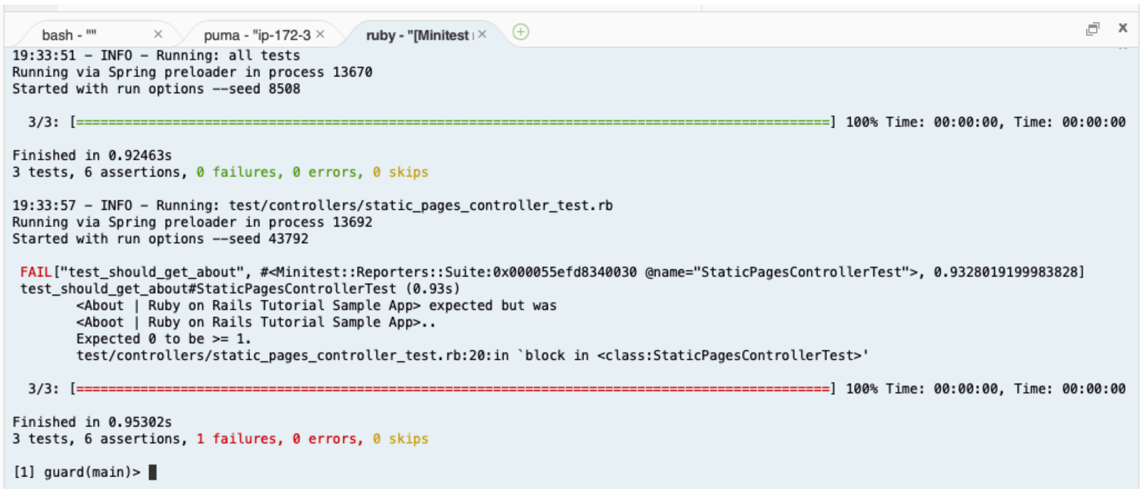
The rules in Listing 3.47 are optimized for this tutorial, automatically running (for example) the integration tests when a controller is changed. To run *all* the tests, simply hit return at the **`guard>`** prompt.

To exit Guard, press Ctrl-D. To add additional matchers to Guard, refer to the examples in Listing 3.47, the Guard README, and the Guard wiki.

If the test suite fails without apparent cause, try exiting Guard, stopping Spring (which Rails uses to pre-load information to help speed up tests), and restarting:

```
$ bin/spring stop    # Try this if the tests mysteriously start failing.
$ bundle exec guard
```

Before proceeding, you should add your changes and make a commit:



```
19:33:51 - INFO - Running: all tests
Running via Spring preloader in process 13670
Started with run options --seed 8508

  3/3: [=========================================================================================] 100% Time: 00:00:00, Time: 00:00:00

Finished in 0.92463s
3 tests, 6 assertions, 0 failures, 0 errors, 0 skips

19:33:57 - INFO - Running: test/controllers/static_pages_controller_test.rb
Running via Spring preloader in process 13692
Started with run options --seed 43792

 FAIL["test_should_get_about", #<Minitest::Reporters::Suite:0x000055efd8340030 @name="StaticPagesControllerTest">, 0.9328019199983828]
 test_should_get_about#StaticPagesControllerTest (0.93s)
        <About | Ruby on Rails Tutorial Sample App> expected but was
        <Aboot | Ruby on Rails Tutorial Sample App>..
        Expected 0 to be >= 1.
        test/controllers/static_pages_controller_test.rb:20:in `block in <class:StaticPagesControllerTest>'

  3/3: [=========================================================================================] 100% Time: 00:00:00, Time: 00:00:00

Finished in 0.95302s
3 tests, 6 assertions, 1 failures, 0 errors, 0 skips

[1] guard(main)>
```

Figure 3.11: Using Guard on the cloud IDE.

```
$ git add -A
$ git commit -m "Complete advanced testing setup"
```