FREE eBook

LEARNING

rspec

Free unaffiliated eBook created from
Stack Overflow contributors.

#rspec

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: rspec

It is an unofficial and free rspec ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official rspec.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with rspec

Remarks

RSpec is a BDD tool used to specify and test Ruby programs. It is used primarily to specify and test classes and methods, i.e. for unit testing.

The rspec gem is just a meta-gem which brings in the three parts of RSpec. Those three parts are also a way to structure this documentation.

- rspec-core provides RSpec's way of structuring and running tests: the `rspec` command-line executable, the `describe`, `context` and `it` methods, shared examples, etc. It is documented in the RSpec Core topic.
- rspec-expectations provides RSpec's support for expecting test results: the `expect`/`to` expectation syntax and RSpec's built-in matchers. (It also provides the deprecated `should` expectation syntax.) It is documented in the RSpec Expectations topic.
- rspec-mocks provides RSpec's support for test doubles: `double`, `allow`, `expect`, `receive`, `have_received`, etc. It is documented in the RSpec Mocks topic.

There is also the rspec-rails gem, which extends RSpec with support for testing the types of classes used in Rails applications, and with support for writing feature specs (acceptance tests) which test the application from the user's point of view.

Official documentation for RSpec and rspec-rails is here: https://www.relishapp.com/rspec

Examples

Installing RSpec

The most common way to install the RSpec gem is using Bundler. Add this line to your application's `Gemfile`:

```
gem 'rspec'
```

And then execute `bundle` to install the dependencies:

```
$ bundle
```

Alternatively, you can install the gem manually:

```
$ gem install rspec
```

After installing the gem, run the following command:

```
rspec --init
```

This will create a `spec` folder for your tests, along with the following config files:

- a `spec` directory into which to put spec files
- a `spec/spec_helper.rb` file with default configuration options
- an `.rspec` file with default command-line flags

A simple RSpec example

In greeter.rb (wherever that goes in your project):

```
class Greeter
  def greet
    "Hello, world!"
  end
end
```

In spec/greeter_spec.rb:

```
require_relative '../greeter.rb'

RSpec.describe Greeter do
  describe '#greet' do
    it "says hello" do
      expect(Greeter.new.greet).to eq("Hello, world!")
    end
  end
end
```

So our file structure looks like:

```
$ tree .
.
├── greeter.rb
└── spec
    └── greeter_spec.rb

1 directory, 2 files
```

Output

```
$rspec greeter_spec.rb
Finished in 0.00063 seconds (files took 0.06514 seconds to load)
1 example, 0 failures
```

In RSpec terminology, the file is a "spec" of `Greeter` and the `it` block is an "example". The line with `expect` is an expectation. If the expectation is met, nothing happens and the test passes. If not, the test fails.

This example also shows that `describe` blocks can be nested, in this case to convey that the `greet` method is part of the `Greet` class. The `#` in `#greet` is only a convention to show that `greet` is an instance method (as opposed to '.' for a class method). RSpec doesn't interpret the string at all, so you could use a different string or omit that `describe` block entirely.

Read Getting started with rspec online: https://riptutorial.com/rspec/topic/2017/getting-started-with-rspec

Chapter 2: RSpec Core

Examples

Running examples with a given tag

Adding tags to "describe" or "it" blocks allows you to run only those examples with a given tag. Use the `--tag` (or `-t`) option to run examples that match a specified tag. The tag can be a simple name or a name:value pair.

- If a simple name is supplied, only examples with `:name => true` will run. For example, `rspec <spec_file> --tag smoke` would run the example tagged with "Smoke".

```
describe '#Tests' do
  it 'runs the smoke test', :smoke => true do
  end

  it 'runs the regression tests', :regression => true do
  end

  it 'runs the acceptance tests', :acceptance => true do
  end
end
```

- If a `name:value` pair is given, examples with `name => value` will run,where value is always a string. For example, `rspec <spec_file> --tag testId:101` would run the example tagged with `testId` "101".

```
describe '#Tests' do
  it 'runs the test with id 99', :testId => 99 do
  end

  it 'runs the test with id 101', :testId => 101 do
  end
end
```

Read RSpec Core online: https://riptutorial.com/rspec/topic/5672/rspec-core

Chapter 3: RSpec Expectations

Introduction

RSpec::Expectations lets you express expected outcomes on an object using an example-based DSL syntax.

Remarks

This topic gives examples of how to expect test results in RSpec using `expect .to` and the many built-in matchers.

This functionality is provided by the rspec-expectations gem.

Examples

Basic Usage

Given a `class` as follows:

```
class Cube
  attr_reader :height, :width, :depth

  def initialize(args)
    @height = args[:height] || args[:y] || 1
    @width  = args[:width]  || args[:x] || 1
    @depth  = args[:depth]  || args[:z] || 1
  end

  def volume
    height * width * depth
  end
end
```

The following example passes if `cube.volume` equals 60 and fails if it doesn't. It uses the most commonly used built-in matcher, `eq`, which just tests for equality.

```
RSpec.describe Cube do
  it "calculates it's volume" do
    cube = Cube.new(x: 3, y: 4, z: 5)
    expect(cube.volume).to eq(60)
  end
end
```

Read RSpec Expectations online: https://riptutorial.com/rspec/topic/4304/rspec-expectations

Chapter 4: RSpec Matcher

Introduction

rspec-expectations ships with a number of built-in matchers. Each matcher can be used with expect(..).to or expect(..).not_to to define positive and negative expectations respectively on an object.

Examples

Equality matchers

compare using eq (==)

```
RSpec.describe "a string" do
  it "is equal to another string of the same value" do
    expect("this string").to eq("this string")
  end

  it "is not equal to another string of a different value" do
    expect("this string").not_to eq("a different string")
  end
end

RSpec.describe "an integer" do
  it "is equal to a float of the same value" do
    expect(5).to eq(5.0)
  end
end
```

When I run `rspec` then the output should contain "3 examples, 0 failures"

compare using ==

```
RSpec.describe "a string" do
  it "is equal to another string of the same value" do
    expect("this string").to be == "this string"
  end

  it "is not equal to another string of a different value" do
    expect("this string").not_to be == "a different string"
  end
end

RSpec.describe "an integer" do
  it "is equal to a float of the same value" do
    expect(5).to be == 5.0
  end
end
```

When I run `rspec` then the output should contain "3 examples, 0 failures"

compare using `eql (eql?)`

```
 RSpec.describe "an integer" do
  it "is equal to another integer of the same value" do
    expect(5).to eql(5)
  end

  it "is not equal to another integer of a different value" do
    expect(5).not_to eql(6)
  end

  it "is not equal to a float of the same value" do
    expect(5).not_to eql(5.0)
  end
end
```

When I run `rspec` then the output should contain "3 examples, 0 failures"

compare using `equal (equal?)`

```
RSpec.describe "a string" do
  it "is equal to itself" do
    string = "this string"
    expect(string).to equal(string)
  end

  it "is not equal to another string of the same value" do
    expect("this string").not_to equal("this string")
  end

  it "is not equal to another string of a different value" do
    expect("this string").not_to equal("a different string")
  end
end
```

When I run `rspec` then the output should contain "3 examples, 0 failures"

compare using be `(equal?)`

```
RSpec.describe "a string" do
  it "is equal to itself" do
    string = "this string"
    expect(string).to be(string)
  end

  it "is not equal to another string of the same value" do
    expect("this string").not_to be("this string")
  end

  it "is not equal to another string of a different value" do
    expect("this string").not_to be("a different string")
  end
end
```

When I run `rspec` then the output should contain "3 examples, 0 failures"

Read RSpec Matcher online: https://riptutorial.com/rspec/topic/10762/rspec-matcher

Chapter 5: RSpec Mocks

Remarks

This topic documents RSpec's support for test doubles (stubs, mocks, etc.). That support is provided by the rspec-mocks gem.

Examples

Stubbing with allow

The following example uses `allow` and `receive` to stub a `Cart`'s call to a `CreditCardService` so that the example doesn't have to wait for a network call or use a credit card number that the processor knows about.

```
class Cart
  def check_out
    begin
      transaction_id = CreditCardService.instance.validate credit_card_number, total_price
      order = Order.new
      order.items = cart.items
      order
    rescue CreditCardService::ValidationFailedError
      # handle the error
    end
  end
end

describe Cart do
  describe '#check_out' do
    it "places an order" do
      allow(CreditCardService.instance).
        to receive(:validate).with("1234567812345678", 3700).and_return("transaction_id")
      cart = Cart.new
      cart.items << Item.new("Propeller beanie", 3700)
      order = cart.check_out
      expect(order.transaction_id).to eq("transaction_id")
    end
  end
end
```

`with` is optional; without it, any arguments are accepted. `and_return` is optional too; without it the stub returns `nil`.

Mocking by setting a message expectation with expect

The following example uses `expect` and `receive` to mock an `Order`'s call to a `CreditCardService`, so that the test passes only if the call is made without having to actually make it.

```
class Order
```

```
  def cancel
     CreditCardService.instance.refund transaction_id
  end
end

describe Order do
  describe '#cancel' do
    it "refunds the money" do
      order = Order.new
      order.transaction_id = "transaction_id"
      expect(CreditCardService.instance).to receive(:refund).with("transaction_id")
      order.cancel
    end
  end
end
```

In this example the mock is on the return value of `CreditCardService.instance`, which is presumably a singleton.

`with` is optional; without it, any call to `refund` would satisfy the expectation. A return value could be given with `and_return`; in this example it is not used, so the call returns `nil`.

Read RSpec Mocks online: https://riptutorial.com/rspec/topic/5678/rspec-mocks

Credits

S. No	Chapters	Contributors
1	Getting started with rspec	adarsh, Ashish Bista, Community, Dave Schweisguth, gmuraleekrishna, Mark Huk, mbigras, Scott Matthewman, Simone Carletti, Srikanth Gurram, Vishnu Y S
2	RSpec Core	Dave Schweisguth, Srikanth Gurram
3	RSpec Expectations	Ashish Bista, Dave Schweisguth, Midwire
4	RSpec Matcher	Faruk Hossain
5	RSpec Mocks	Dave Schweisguth