madrid-rb

Febrero 2018

«Introducción al testing (con rspec)»

Patrocinado por

BeBanjo ProRuby ASP gems B'IMOTION

cabify LEXTREND the cocktail® TRIVE

Bienvenidos a **madrid-rb**

Grupo de usuarios de ruby de madrid

Último jueves de cada mes (en general)

http://www.madridrb.com

Testing 101

Introducción al testing

Yo

Fernando Martínez a.k.a. @oinak

Mantengo rubyonrails.org.es

Ayudé con conferenciaror.es

Trabajo en returnly.com *(we're hiring!)*

@agustincnc: «el taliban de los tests»

demoscopia

¿Quién hace tests?

¿Quién hace tests *automáticos*?

¿Quién *empieza* por tests automáticos?

¿Por qué testear?

siempre hay alguien que <u>testea</u>

el cliente, tras pagar

tu, tras deployar

tu, tras programar

tu, mientras programas

Everybody has a testing environment.
Some people are lucky enough enough
to have a totally separate environment...
to run production in.
@stahnma

«**todo el mundo** tiene un entorno de pruebas.
algunos afortunados tienen, un entorno *aparte*...
para ejecutar producción»

¿Cómo testear?

Érase una vez...

La trágica historia del estudiante de informática que no conocía cvs

clase mínima de ruby:

Almacena un nobre de usuario para saludarlo

```
g = Greeter.new("Nombre")
g.say
# => Hola Nombre!
```

clase mínima de ruby:

```ruby
class Greeter
  def initialize(name = "World")
    @name = name
  end

  def say
    "Hello #{@name}"
  end
end
```

prueba mínima:

```
$ ruby -r'./examples/greeter.rb' -e 'puts Greeter.new("Fer").say'
Hello Fer
```

¿está bien?

guardar salida para comprobar

¿cual fue el error?

¿efectos secundarios?

prueba mínima en irb:

```ruby
require './examples/greeter.rb'

greet    = Greeter.new("Fer")
expected = "Hola Fer!"
got      = greet.say

if got == expected
  puts 'ok'
else
  puts "ERROR, Expected: #{expected}, got #{got}"
end
```

```
> require './examples/greeter.rb'
=> true
> greet    = Greeter.new("Fer")
=> #<Greeter:0x000055abdd9fcd40 @name="Fer">
> expected = "Hola Fer!"
=> "Hola Fer!"
> got = greet.say
=> "Hello Fer"
> if got == expected
?>   puts 'ok'
?> else
?>   puts "ERROR, Expected: #{expected}, got #{got}"
?> end
```

```
ERROR, Expected: Hola Fer!, got Hello Fer
```

D.R.Y.

¿Cómo testear (una clase)?

test mínimo (minitest):

```
require 'minitest/autorun'            # "soy un test"
require 'my_obj'                      # carga lo que vas a probar

class ObjTest < Minitest::Test        # convención: TuClaseTest

  def test_what                       # ejecutará todos los test_xxxx

    o = Obj.new("Value")              # 1. prepara lo necesario

    result = o.my_method              # 2. ejecuta tu codigo

    assert_equal(result, "Expected") # 3. compara con lo esperado

    o.destroy                         # 4. dejalo como lo encontraste
  end
end
```

test mínimo (minitest):

```ruby
require 'minitest/autorun' # to test a single file
require 'greeter' # test subject

class GreeterTest < Minitest::Test
  def test_say
    greeter = Greeter.new("Ada")

    assert_equal(greeter.say, "Hello Ada!")
  end
end
```

prueba automática (minitest)

```
$ ruby -I./examples ./examples/greeter_test.rb
Run options: --seed 5216

# Running:

F

Finished in 0.000568s, 1759.7114 runs/s, 1759.7114 assertions/s.

  1) Failure:
GreeterTest#test_say [./examples/greeter_test.rb:8]:
Expected: "Hello Ada"
  Actual: "Hello Ada!"

1 runs, 1 assertions, 1 failures, 0 errors, 0 skips
```

test mínimo (rspec):

Primero hacemos `gem install rspec`

```ruby
require 'greeter' # test subject

RSpec.describe Greeter do
  describe "#say" do
    it "returns 'Hello Name!'" do
      greeter = Greeter.new("Ada")

      expect(greeter.say).to eq("Hello Ada!")
    end
  end
end
```

prueba automática (rspec)

```
$ rspec -Iexamples/ examples/greeter_spec.rb
F

Failures:

  1) Greeter#say returns 'Hello Name!'
     Failure/Error: expect(greeter.say).to eq("Hello Ada!")

       expected: "Hello Ada!"
            got: "Hello Ada"

       (compared using ==)
     # ./examples/greeter_spec.rb:8:in `block (3 levels) in <top (required)>'

Finished in 0.01217 seconds (files took 0.08172 seconds to load)
1 example, 1 failure

Failed examples:

rspec ./examples/greeter_spec.rb:5 # Greeter#say returns 'Hello Name!'
```

¿Cómo testear una Gema?

```
$ bundle gem my_gem
Creating gem 'my_gem'...
Do you want to generate tests with your gem?
Type 'rspec' or 'minitest' to generate those test files now and in
the future. rspec/minitest/(none):
```

gema con minitest

```
my_gem_mini/
├── bin
│   ├── console
│   └── setup
├── CODE_OF_CONDUCT.md
├── Gemfile
├── lib
│   ├── my_gem_mini
│   │   └── version.rb
│   └── my_gem_mini.rb
├── LICENSE.txt
├── my_gem_mini.gemspec
├── Rakefile
├── README.md
└── test
    ├── my_gem_mini_test.rb
    └── test_helper.rb
```

gema con rspec

```
my_gem_rspec/
├── bin
│   ├── console
│   └── setup
├── CODE_OF_CONDUCT.md
├── Gemfile
├── lib
│   ├── my_gem_rspec
│   │   └── version.rb
│   └── my_gem_rspec.rb
├── LICENSE.txt
├── my_gem_rspec.gemspec
├── Rakefile
├── README.md
└── spec
    ├── my_gem_rspec_spec.rb
    └── spec_helper.rb
```

gemspec comun

```ruby
Gem::Specification.new do |spec|
  spec.name          = "my_gem_rspec"
  spec.version       = MyGemRspec::VERSION
  spec.authors       = ["Fernando Martínez"]
  spec.email         = ["me@oinak.com"]

  spec.summary       = %q{TODO: Write a short summary, because RubyGems requires one.}
  spec.description   = %q{TODO: Write a longer description or delete this line.}
  spec.homepage      = "TODO: Put your gem's website or public repo URL here."
  spec.license       = "MIT"

  # Prevent pushing this gem to RubyGems.org. To allow pushes either set the 'allowed_push_host'
  # to allow pushing to a single host or delete this section to allow pushing to any host.
  if spec.respond_to?(:metadata)
    spec.metadata["allowed_push_host"] = "TODO: Set to 'http://mygemserver.com'"
  else
    raise "RubyGems 2.0 or newer is required to protect against " \
      "public gem pushes."
  end
  #...
end
```

gemspec minitest

```ruby
Gem::Specification.new do |spec|
  spec.name          = "my_gem_mini"
  #...
  spec.files         = `git ls-files -z`.split("\x0").reject do |f|
    f.match(%r{^(test|spec|features)/})
  end
  spec.bindir        = "exe"
  spec.executables   = spec.files.grep(%r{^exe/}) { |f| File.basename(f) }
  spec.require_paths = ["lib"]

  spec.add_development_dependency "bundler", "~> 1.16"
  spec.add_development_dependency "rake", "~> 10.0"
  spec.add_development_dependency "minitest", "~> 5.0"
end
```

gemspec rspec

```ruby
Gem::Specification.new do |spec|
  spec.name          = "my_gem_rspec"
  #...
  spec.files         = `git ls-files -z`.split("\x0").reject do |f|
    f.match(%r{^(test|spec|features)/})
  end
  spec.bindir        = "exe"
  spec.executables   = spec.files.grep(%r{^exe/}) { |f| File.basename(f) }
  spec.require_paths = ["lib"]

  spec.add_development_dependency "bundler", "~> 1.16"
  spec.add_development_dependency "rake", "~> 10.0"
  spec.add_development_dependency "rspec", "~> 3.0"
end
```

Configurar

test_helper.rb

```ruby
$LOAD_PATH.unshift File.expand_path("../../lib", __FILE__)
require "my_gem_mini"

require "minitest/autorun"
```

spec_helper.rb

```ruby
require "bundler/setup"
require "my_gem_rspec"

RSpec.configure do |config|
  # Enable flags like --only-failures and --next-failure
  config.example_status_persistence_file_path = ".rspec_status"

  # Disable RSpec exposing methods globally on `Module` and `main`
  config.disable_monkey_patching!

  config.expect_with :rspec do |c|
    c.syntax = :expect
  end
end
```

Ejecutar

Rakefile (minitest)

```ruby
require "bundler/gem_tasks"
require "rake/testtask"

Rake::TestTask.new(:test) do |t|
  t.libs << "test"
  t.libs << "lib"
  t.test_files = FileList["test/**/*_test.rb"]
end

task :default => :test
```

Rakefile (rspec)

```ruby
require "bundler/gem_tasks"
require "rspec/core/rake_task"

RSpec::Core::RakeTask.new(:spec)

task :default => :spec
```

¿Qué testear?

¿Qué testear?

Unitarios (una pieza)

Funcionales (interacción entre piezas)

Integración (interacción entre sistemas)

Acceptación (punto de vista de usuario)

sí, es una simplificación gruesa, ahorraos el tuit

¿Qué testear (rails way)?

Modelos (tocando DB)

Controlador (abstrayendo muchas piezas, rutas, vistas...)

Integración (default en rails 5, ahora menos lentos)

System, benchmark...

¿Qué testear (unit)?

	Incoming	**Outgoing**
Query	Response [1]	Stub (test at provider) [3]
Command	State (Response) [2]	Call is made (mock/verify) [4]

1. `full_name` en una persona
2. `compact!` en una colección
3. `Time.now` que usaremos en un timestamp
4. `Slack.notify(channel: "#alert",txt: "Error: #{msg}")`

1. incoming query

- Como lo que hemos visto
- Llamar a un método público y
- Comparar el retorno con lo esperado

2. incoming command

```
class Cart
  def scan(code)
    @products << Product.new(code)
  end

  def total
    sub_total - discounts
  end

  #...
end
```

Y si no hay método `products`?

2. incoming command (cont)

Opciones:

No testear "si es privado no le importa a nadie"

Efectos "testear lo que ves"

```
send(:privado), get_instance_variable("@foo")
```

3. outgoing query

- **este** test no debería fallar
- no ejecutar código de más
- respuesta conocida

3. outgoing query (stub)

```
product = Product.new(expire_date: Date.new(2010,5,5))

def Date.today
  new(2010, 5, 6)
end

assert_equal(product.expired?, true)
```

`Date` se queda modificado

hay que *arreglarlo* a mano

3. outgoing query (stub/minitest)

```ruby
# add 'require "minitest/mock"' to test_helper.rb
class PersonTest < Minitest::Test
  def setup
    @product = Product.new(expire_date: Date.new(2010,5,5))
    @expired_date = Date.new(2010,5,6)
  end

  def test_expired_after_exired_date
    Time.stub(:now, @expired_date) do
      assert_equal(@product.expired?, true)
    end
  end
end
```

método queda restaurado al terminar el bloque

3. outgoing query (stub/spec)

```
describe Product do # Rspec.describe Product do
  describe ".expired?" do
    subject{ Product.new(expire_date: Date.new(2010,5,5)) }

    context "after expire date" do
      let(:expired_date){ Date.new(2010,5,6) }
      before do
        allow(Date).to receive(:today).and_return(expired_date)
      end

      it "is expired" do
        expect(subject).to be_expired # checks with and without '?'
      end
    end
  end
end
```

3. outgoing query (stub/minispec)

```ruby
# add 'require "minitest/spec"' to test_helper.rb
describe Product do # class ProductTest < Minitest::Spec
  describe ".expired?" do
    subject{ Product.new(expire_date: Date.new(2010,5,5)) }

    describe "after expire date" do
      let(:expired_date){ Date.new(2010,5,6) }
      before do
        allow(Date).to receive(:today).and_return(expired_date)
      end

      it "is expired" do
        expect(subject).must_be :expired?
      end
    end
  end
end
```

4. outgoing command

comprobar que cambiamos el mundo

no cómo, sino qué

clase con dependencias

```
class DepOne
  def initialize(foo)
    @foo = foo
  end

  def run
    if @foo.save
      publish
    end
  end

  private

  def publish
    Notification.new(@foo, Time.now).deliver
  end
end
```

test deps (outgoing command/rspec)

```
describe "sends notification" do
  let(:foo){ double() }
  subject { DepOne.new(foo) }

  before do
    expect_any_instance_of(Notification).to receive(:deliver).
      and_return(true)

    expect(foo).to_receive(:save).and_return(true)
  end

  it "returns the specified value" do
    expect(subject.run).to eq(true)

    # auto verifies mocks
  end
end
```

se cura con una inyeccion

```ruby
class DepOne
  def initialize(foo, notifier: Notification )
    @foo = foo
    @notifier = notifier
  end

  def run
    if @foo.save
      publish
    end
  end

  private

  def publish
    notifier.deliver(@foo, Time.now)
  end
end
```

test deps (outgoing command/minispec)

```
describe "sends notification" do
  let(:foo){ Minitest::Mock.new }
  let(:notifier){ Minitest::Mock.new }
  subject { DepOne.new(foo) }

  before do
    notifier.expect(:publish, notification, [Time, foo])
    foo.expect(:save, true)
  end

  it "returns the specified value" do
    expect(subject.run).to eq(true)

    foo.verify # it was called the appropiate number of times
  end
end
```

deberes

Mocks and stubs *Fowler & Meszaros*

Gracias 🍕 !

Patrocinado por

BeBanjo ProRuby ASP gems B'IMOTION

cabify LEXTREND the cocktail® TRIVE