**Universidad nacional Autónoma de Nicaragua Unan-León**

Un dibujo de un personaje animado

Descripción generada automáticamente con confianza baja

**Ingeniería en Telemática**

**Facultad De Ciencia y Tecnologia**

**Componente: Software como servicio**

**Docente: Ervin Montes**

**Tema:**

**Guía 3: Trabajando con Rspec y Ruby**

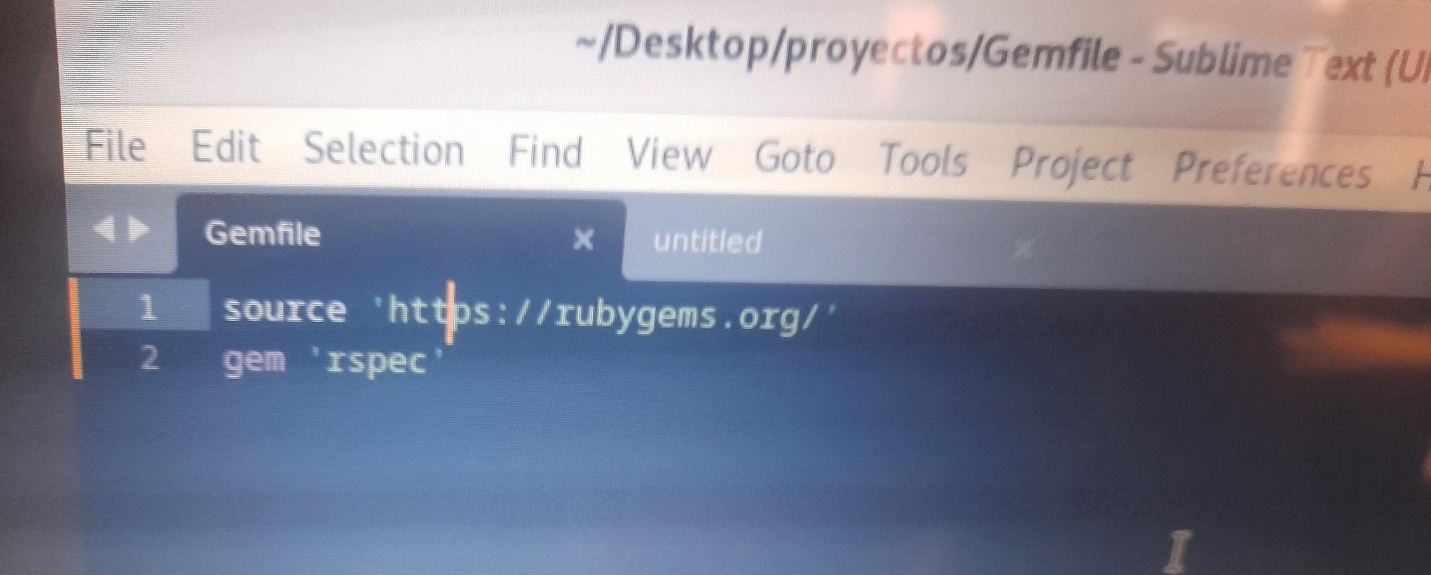
**Elaborado por:**

**Lenuel Gastón Pereira Hernández 19-03199-0**

**Ingrid Valeria Ruiz Ulloa 21-00483-0**

# 

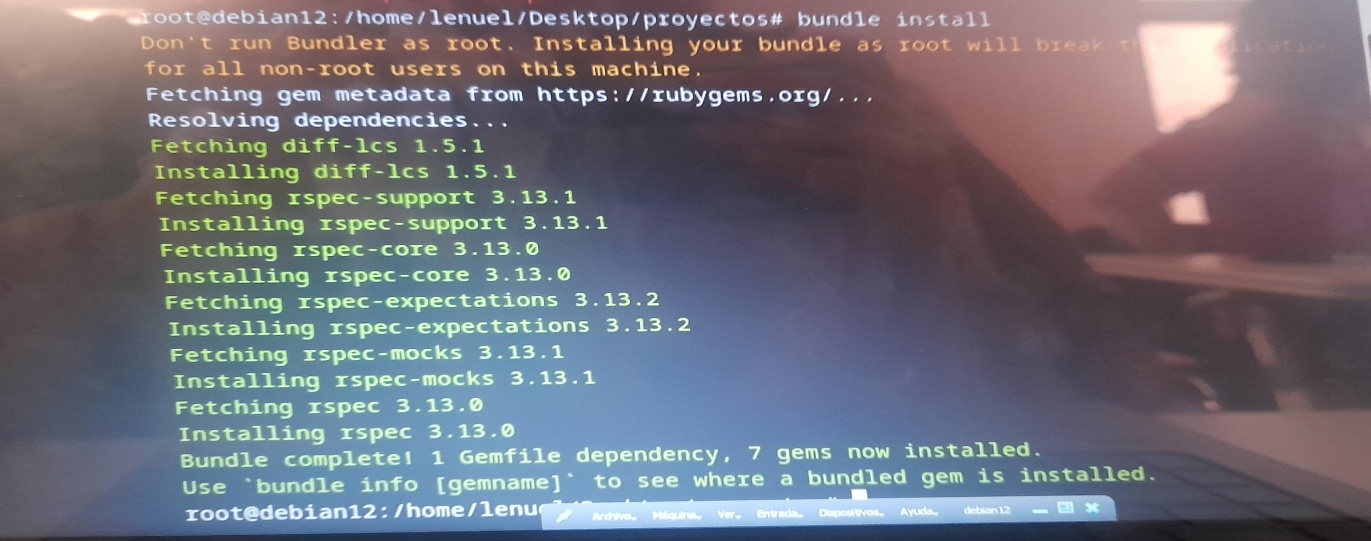
1. Crear un directorio donde almacenar el proyecto.
   1. Crear un archivo llamado Gemfile dentro del directorio creado anteriormente, este archivo es donde se agregará la gema que necesita el proyecto, en este caso solo se utilizará la gema rspec para realizar los tests al código.
   2. Abrir el archivo Gemfile y agregar las siguientes líneas, para poder integrar rspec al proyecto.



source 'https://rubygems.org/'

1. En el terminal, ubicarse dentro del directorio de trabajo y escribir los siguientes comandos.
   1. Instalar la gema rspec, que es la que se utiliza para trabajar con TDD y que servirá para elaborar los tests.

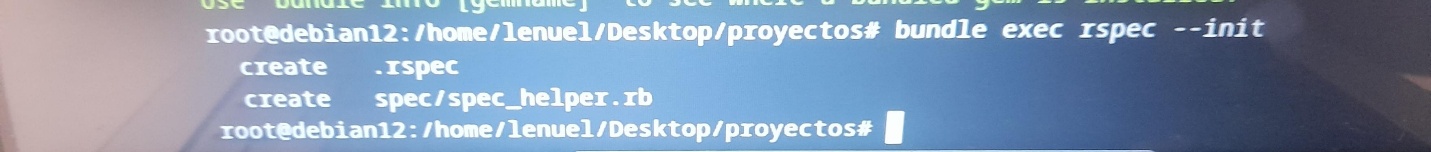
$ bundle install

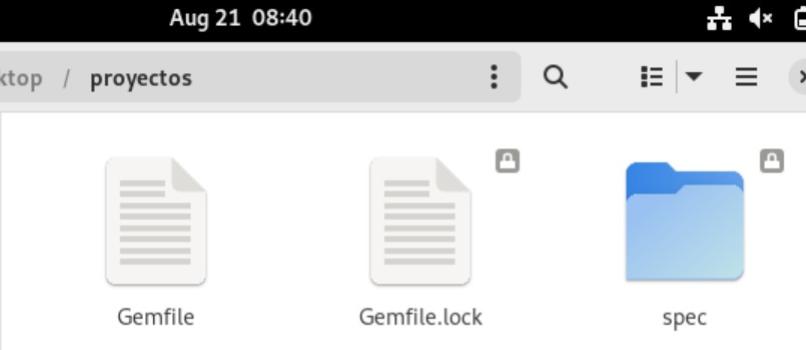


* 1. Ejecutar rspec.

Se crearán automáticamente dos archivos: .rspec y spec/spec\_helper.rb que son los archivos que utiliza rspec para correr los tests.

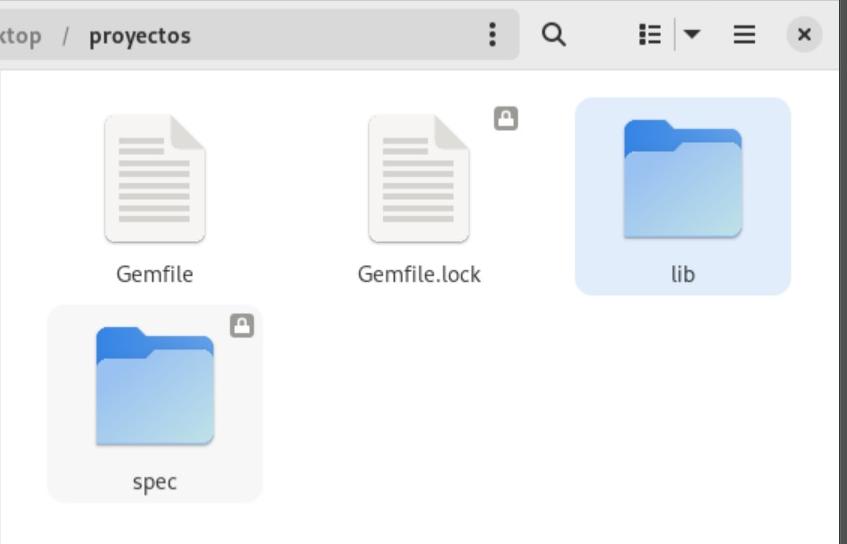
$ bundle exec rspec --init



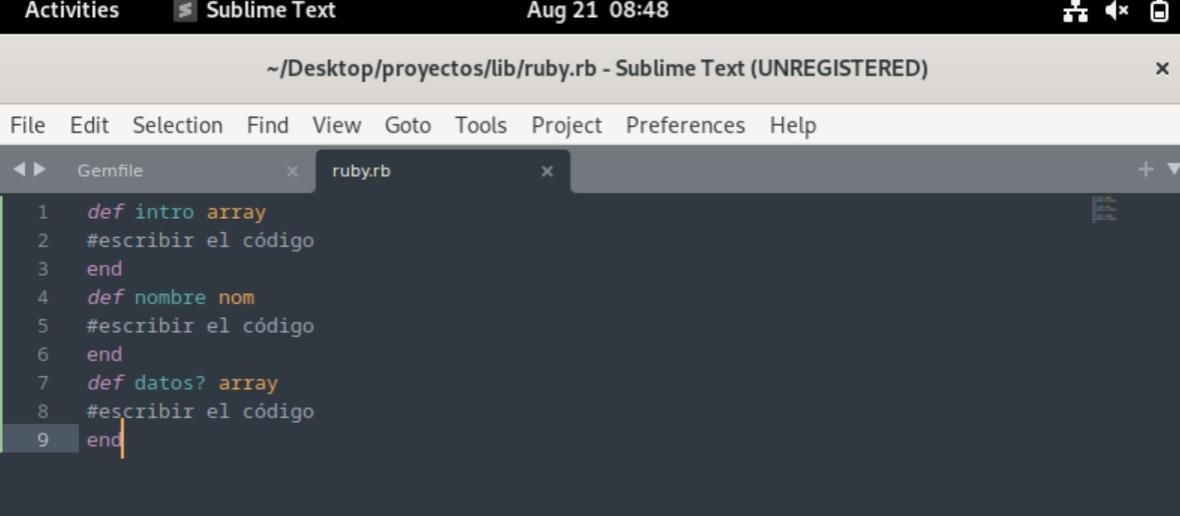


3.

En el directorio raíz del proyecto, crear una carpeta nueva llamada lib



3.1. Dentro de la carpeta lib, crear un archivo ruby.rb y agregar lo siguiente.

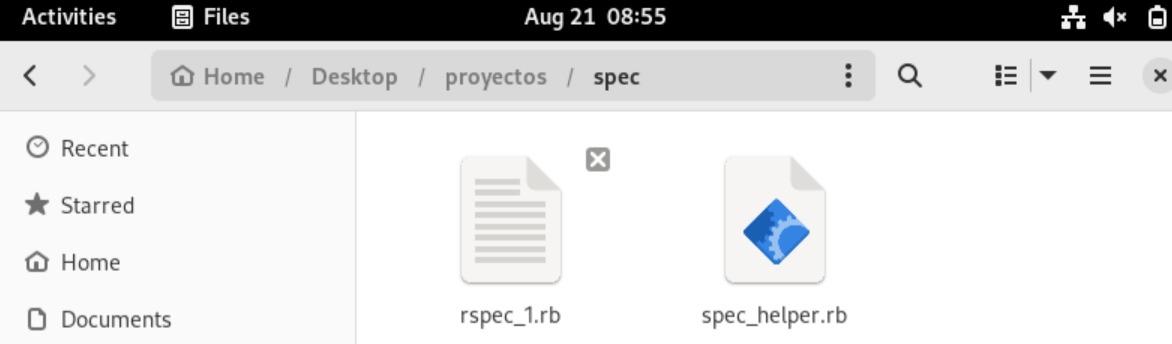


Uno de los principales motivos de crear tests, es que dan garantía de un código de mejor calidad que ha sido sometido a diferentes eventualidades y todos los posibles bugs fueron corregidos.

Como se observa, en el código anterior existen 3 métodos que serán donde se va escribir el código

Ruby en base a los tests que se aplicarán en el proyecto, mostrado a continuación.

1. Dentro de la carpeta **spec,** crear un archivo **rspec\_1.rb** y copiar el siguiente código, este código es cada uno de los tests aplicados, para escribir el código Ruby dentro de los métodos que están en el archivo **ruby.rb**.

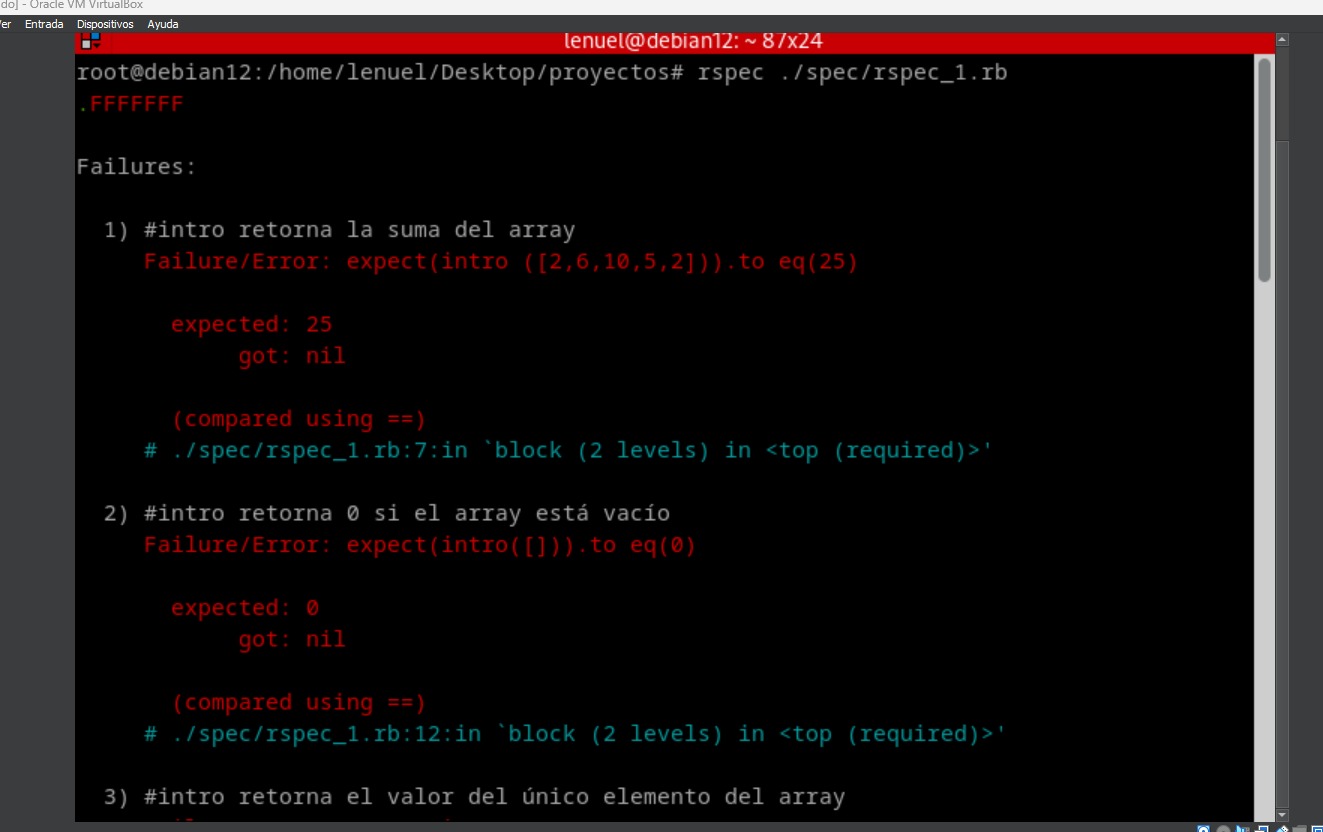


|  |
| --- |
| require "ruby.rb"    describe "#intro" do    it "Debe estar definada" do  expect{intro([1,2,3,4])}.not\_to raise\_error end    it "retorna la suma del array" do expect(intro ([2,6,10,5,2])).to eq(25) expect(intro ([15,6,10,19])).to eq(50) expect(intro ([15,6,4,5])).to eq(30) end    it "retorna 0 si el array está vacío" do expect(intro([])).to eq(0) end    it "retorna el valor del único elemento del array" do expect(intro([5])).to eq(5) end    end    describe "#nombre" do    it "Debe iniciar con Mayúscula y concatena la cadena con la palabra hello" do  expect(nombre("Pedro")).to eq("hello Pedro") expect(nombre("Juan")).to eq("hello Juan") expect(nombre("María")).to eq("hello María") end    it "No trabaja con un string vacío" do  expect(nombre("")).to eq("vacío")  end end    describe "#datos?" do    it "retorna true si el tamaño del array es > 0" do  expect(datos?([2,1,5,6])).to be true  end  it "retorna false si el tamaño del array es <= 0" do expect(datos?([])).to be false  end  end |

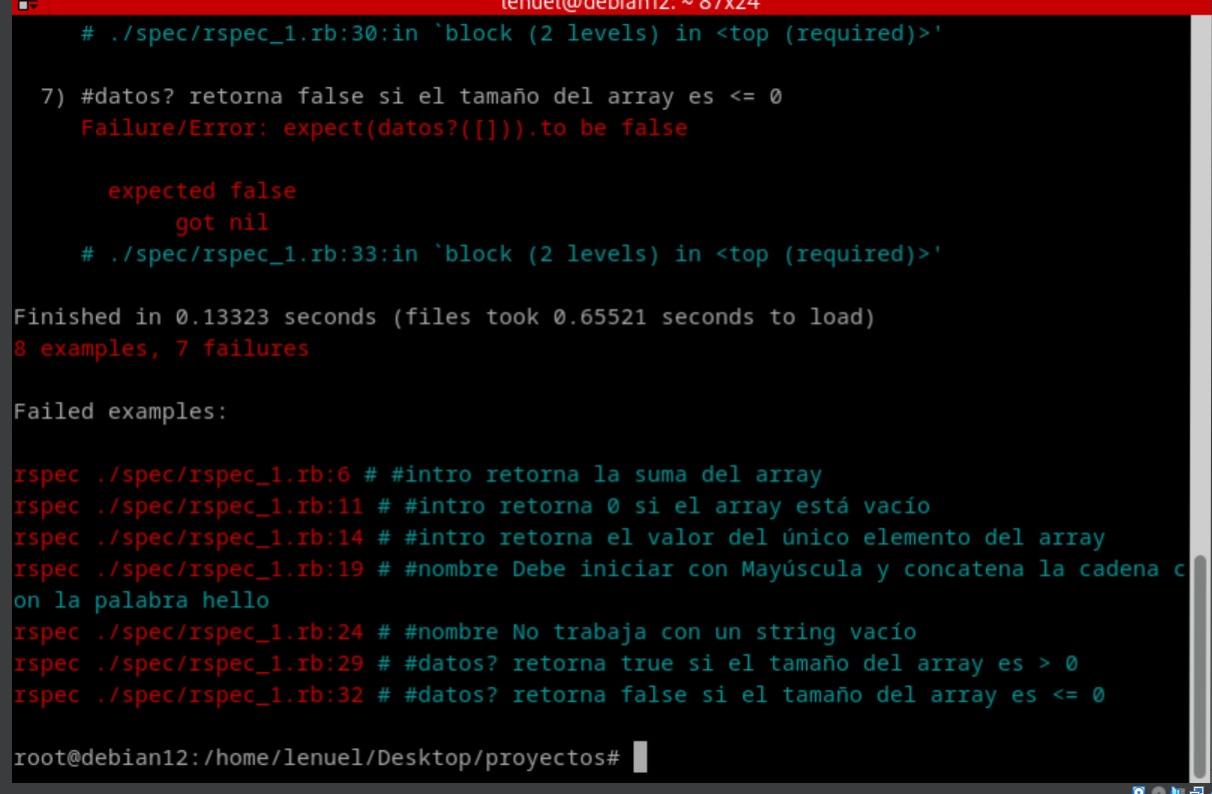
1. En el terminal, ejecutar el siguiente comando para verificar el funcionamiento de los test.

$ rspec spec/rspec\_1.rb

Al ejecutar el comando anterior mostrará un conjunto de mensajes en color rojo, los cuales corresponden a cada uno de los tests que se están aplicando en el ejercicio, cuando los tests se muestran en color rojo es debido a que los tests ejecutados están fallando, en relación a la funcionalidad del código al cual se le están aplicando cada uno de ellos.



Al final de todos los mensajes, se muestra exactamente los tests que están siendo ejecutados.

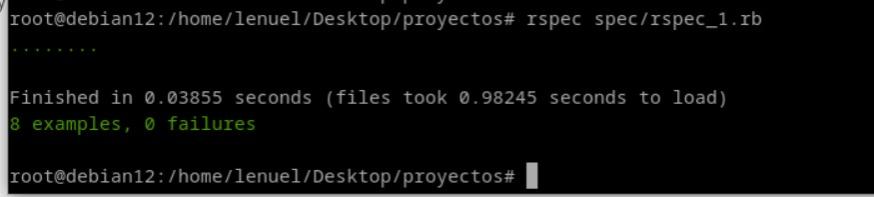


El desarrollo en base a pruebas o TDD no es más que, primero escribir los tests y correrlos esperando que fallen; que es cuando se muestran los mensajes en rojo, escribir código y volver a correr los tests, esperando que estos pasen y que se muestren los mensajes en verde que significa que el código no contiene errores o que los tests funcionan de manera correcta.

**Ejercicios propuestos para ser entregados al docente**

1. Realizar cada uno de los enunciados de la guía.

1. Escribir el código Ruby necesario dentro cada una de los métodos creados en el archivo **lib/ruby.rb** para aprobar todos los tests, y que muestre los mensajes en color verde como se observa.



**Extra**: escribir un conjunto de tests en relación a lo aprendido en la guía dentro del archivo **rspec\_1.rb,** para luego crear un método y escribir código ruby de tal manera que pueda hacer funcionar los test a como se trabajó en la guía.

3 Supongamos que tienes la siguiente clase Calculator en un archivo llamado calculator.rb:

|  |
| --- |
| class Calculator def add(a, b)  a + b  end    def subtract(a, b)  a - b end    def multiply(a, b)  a \* b  end    def divide(a, b)  raise ZeroDivisionError if b == 0 a / b end end |

Ahora, vamos a crear pruebas para esta clase utilizando RSpec. Crea un archivo llamado calculator\_spec.rb para escribir tus pruebas:

# calculator\_spec.rb

require\_relative 'calculator' # Asegúrate de ajustar la ruta según la ubicación de tu archivo calculator.rb

RSpec.describe Calculator do

|  |
| --- |
| let(:calculator) { Calculator.new }    describe "#add" do it "sums two numbers" do result = calculator.add(2, 3) expect(result).to eq(5) end end    describe "#subtract" do it "subtracts two numbers" do result = calculator.subtract(5, 3) expect(result).to eq(2) end end    describe "#multiply" do it "multiplies two numbers" do result = calculator.multiply(4, 3) expect(result).to eq(12) end end    describe "#divide" do it "divides two numbers" do result = calculator.divide(10, 2) expect(result).to eq(5) end    it "raises an error when dividing by zero" do  expect { calculator.divide(10, 0) }.to raise\_error(ZeroDivisionError) end  end end |

Deberías ver la salida de RSpec que indica si todas las pruebas pasaron o no.

