Getting Started with Cucumber

Introduction

Adding Cucumber to a Project

Note: This tutorial assumes you have a modern version of Ruby and RubyGems installed. If you don't, try using the Ruby Version Manager to install it.

To get started, install Cucumber from RubyGems with the `gem` command:

```
$ gem install cucumber
```

This will install the Cucumber module along with any of its dependencies. Next, enter your project directory and create a folder called `features`, and a subdirectory of it called `step_definitions`, and another called `support`.

```
$ mkdir -p features/step_definitions
$ mkdir features/support
```

If your project doesn't have a `Gemfile` in the root directory, create one. To the exist `Gemfile` or newly created one, add the following lines:

```
group :test do
    gem 'cucumber'
    gem 'rspec-expectations'
end
```

In the previously created `support` directory, create a file called `env.rb`, and add the following lines:

```
# features/support/env.rb
# <add any additional imports here>
require 'rspec/expectations'
World(RSpec::Matchers)
```

You've now got Cucumber installed and a skeleton test apparatus set up for your project. You should now run `bundle exec cucumber` from the root directory of your project (the folder containing the `features` directory) to make sure it's installed correctly, though it won't find any tests yet. If all is well, you should see output like the following:

```
$ bundle exec cucumber
0 scenarios
0 steps
0m0.000s
```

Creating Features

In order to use Cucumber meaningfully, you need to describe the features in your project that you would like to test. Cucumber allows you to describe fairly high level features as in an English-like syntax, much like a use case, and then test the implementations that fulfill those use cases.

Feature files end with a `.feature` extension and live in your project's `features` directory. To get started, create a feature file in your features directory. Feature files can have any name you like, but for now try calling it `tutorial1.feature`.

Before filling out the feature file, lets first look at an example of a simple feature description from the Cucumber official documentation.

Feature Syntax

```
Feature: Serve coffee
  In order to earn money
  Customers should be able to
  buy coffee at all times

  Scenario: Buy last coffee
    Given there are 1 coffees left in the machine
    And I have deposited 1$
    When I press the coffee button
    Then I should be served a coffee
```

The first thing to note about this format is that it's entirely human readable. The first line of a feature file, `Feature: Serve coffee` in this example, is a header that indicates the name of the feature. This name will appear in the test output with Cucumber is run on your project. The next three lines are actually free-form text; they are their to assist human readers of the feature and help to explain the business case for the feature. It's recommended that these three lines contain a purpose, the user of the feature, and the general action the feature performs.

Following the free-form section are one or more Scenarios. Scenarios are written in a Given/When/Then format that *is* parsed by Cucumber. These lines describe the preconditions for the specific aspect of the feature to be exercised, the trigger, and the expected output. Rather than being generic, these lines can describe a single specific example, such as what should happen when a single dollar is inserted into the machine.

Lets describe a feature for a simple calculator that allows a user to average integers.

Open `tutorial1.feature` and enter the following lines, referring to the above syntax explanation if necessary. Remember that the first three lines following the `Feature:` header are not parsed by Cucumber.

```
Feature: Sum grades
  In order to discover what my total score is
  As a student learning Cucumber who is bad at math
  I want to be able to average a list of my grades

Scenario:
  Given that I have a calculator
    and that I have entered 90 for my first grade
    and that I have entered 85 for my second grade
    and that I have entered 98 for my third grade
  When I press the average button
  Then the calculator should output 91 as my average
```

Once you finished editing `tutorial1.feature`, save the file, return to the root of your project directory, and run `bundle exec cucumber`. You should see output similar to the following:

```
my_project/ $ bundle exec cucumber

Feature: Average grades
In order to discover what my total score is
As a student learning Cucumber who is bad at math
I want to be able to average a list of my grades

Scenario:                                          # features/tutorial1.feature
:6
    Given that I have a calculator                   # features/tutorial1.featu
re:7
    And that I have entered 90 for my first grade    # features/tutorial1.featu
re:8
    And that I have entered 85 for my second grade   # features/tutorial1.featu
re:9
    And that I have entered 98 for my third grade    # features/tutorial1.featu
re:10
    When I press the average button                  # features/tutorial1.featu
re:11
    Then the calculator should output 91 as my average # features/tutorial1.featu
re:12

1 scenario (1 undefined)
6 steps (6 undefined)
0m0.002s

You can implement step definitions for undefined steps with these snippets:

Given(/^that I have a calculator$/) do
  pending # express the regexp above with the code you wish you had
```

```
end

Given(/^that I have entered (\d+) for my first grade$/) do |arg1|
  pending # express the regexp above with the code you wish you had
end

<...>

If you want snippets in a different programming language,
just make sure a file with the appropriate file extension
exists where cucumber looks for step definitions.
```

These error messages are an indication from Cucumber that it can't find a step definition for the feature. This makes sense, considering one hasn't been written yet. To rectify this situation, create a file in the `features/step_definitions` folder with a `.rb` extention, and add the following lines:

```
Given /that I have a calculator/ do
    @calculator = Calculator.new
end

Given /I have entered (\d+) for my (.*) grade/ do |grade, n|
    @calculator.add(grade.to_i)
end

When /I press the average button/ do
    @calculator.average()
end

Then /the calculator should output (\d+) as my average/ do |n|
    @calculator.grade.should == n.to_i
end
```

Each of these lines represents a step from the scenario we defined earlier. Each step definition is a function that takes a regular expression matching the natural language steps from the scenario file, with groups such as `(\d+)` called out in order to capture arguments or other variable parts of the step. As you can see here, for the grade-entering step, there are two parameters– the grade value and the ordinal number (first, second) of its position in the list. Note that parameters captured like this are strings, and they must be cast to the appropriate type if you expect them to be integers, floats, or other types.

You can see from this step definition how actual implementation can be attached to the human-readable feature description. To try it out, save the step definition file and run `bundle exec cucumber` again from the root directory.

```
my_project/ $ bundle exec cucumber
Feature: Average grades
In order to discover what my total score is
As a student learning Cucumber who is bad at math
```

```
I want to be able to average a list of my grades

  Scenario:                                          # features/tutorial1.featu
re:6
    Given that I have a calculator                   # features/step_definition
s/tutorial1.rb:1
      uninitialized constant Calculator (NameError)
      ./features/step_definitions/tutorial1.rb:2:in `/that I have a calculator/'
      features/tutorial1.feature:7:in `Given that I have a calculator'
    And that I have entered 90 for my first grade    # features/step_definition
s/tutorial1.rb:5
    And that I have entered 85 for my second grade   # features/step_definition
s/tutorial1.rb:5
    And that I have entered 98 for my third grade    # features/step_definition
s/tutorial1.rb:5
    When I press the average button                  # features/step_definition
s/tutorial1.rb:9
    Then the calculator should output 91 as my average # features/step_definition
s/tutorial1.rb:13

Failing Scenarios:
cucumber features/tutorial1.feature:6 # Scenario:

1 scenario (1 failed)
6 steps (1 failed, 5 skipped)
0m0.002s
```

As you may have expected, this time the test gets further but fails because we haven't actually implemented the `Calculator` class in question. In another Ruby source file, define the Calculator class as follows and then add a `require` statement to `env.rb` to import the file that contains it. Here's an example implementation that will pass the tests we defined:

```
class Calculator
    def initialize
        @sum = 0
    end

    def add(n)
        @sum += n
    end

    def grade
        return @grade
    end
    def grade=(val)
        @grade = val
    end

    def average()
        @grade = (@sum/3.0)
    end
```

```
end
```

This time, running Cucumber should result in a full set of passed tests, and nice bright green output lines, like the color of a cucumber.

```
Feature: Average grades
    In order to discover what my total score is
    As a student learning Cucumber who is bad at math
    I want to be able to average a list of my grades

  Scenario:                                          # features/tutorial1.featu
re:6
    Given that I have a calculator                   # features/step_defininiti
ons/tutorial1.rb:1
    And that I have entered 90 for my first grade    # features/step_defininiti
ons/tutorial1.rb:5
    And that I have entered 85 for my second grade   # features/step_defininiti
ons/tutorial1.rb:5
    And that I have entered 98 for my third grade    # features/step_defininiti
ons/tutorial1.rb:5
    When I press the average button                  # features/step_defininiti
ons/tutorial1.rb:9
    Then the calculator should output 91 as my average # features/step_defininiti
ons/tutorial1.rb:13

1 scenario (1 passed)
6 steps (6 passed)
0m0.006s
```

Additional Resources

For more help with Cucumber, check out the official site and documentation available at the following links:

- Cucumber
- Cucumber on GitHub
- Cucumber Documentation