Test-Driven Development for Ruby and Rails

For Ruby 4.0.8

Source: http://guides.rubyonrails.org/v4.0.8/testing.html

Table of Contents

assert_template(expected = nil, message=nil)
assert_select_email
assert_select_encoded
css_select(selector)or css_select(element, selector)

Integration Testing Helpers

https?
https!
host!
redirect?
follow_redirect!
request_via_redirect(http_method, path, [parameters], [headers])
post_via_redirect(path, [parameters], [headers])
get_via_redirect(path, [parameters], [headers])
patch_via_redirect(path, [parameters], [headers])
put_via_redirect(path, [parameters], [headers])
delete_via_redirect(path, [parameters], [headers])
open_session

Rake Tasks for Running Tests

rake test
rake test:controllers
rake test:functionals
rake test:helpers
rake test:integration
rake test:mailers
rake test:models
rake test:units
rake test:all
rake test:all:db

Assertions

Assertion	Purpose
`assert( test, [msg] )`	Ensures that `test` is true.
`assert_not( test, [msg] )`	Ensures that `test` is false.
`assert_equal( expected, actual, [msg] )`	Ensures that `expected == actual` is true.
`assert_not_equal( expected, actual, [msg] )`	Ensures that `expected != actual` is true.
`assert_same( expected, actual, [msg] )`	Ensures that `expected.equal?(actual)` is true.
`assert_not_same( expected, actual, [msg] )`	Ensures that `expected.equal?(actual)` is false.
`assert_nil( obj, [msg] )`	Ensures that `obj.nil?` is true.
`assert_not_nil( obj, [msg] )`	Ensures that `obj.nil?` is false.
`assert_match( regexp, string, [msg] )`	Ensures that a string matches the regular expression.

`assert_no_match( regexp, string, [msg] )`	Ensures that a string doesn't match the regular expression.
`assert_in_delta( expecting, actual, [delta], [msg] )`	Ensures that the numbers `expected` and `actual` are within `delta` of each other.
`assert_not_in_delta( expecting, actual, [delta], [msg] )`	Ensures that the numbers `expected` and `actual` are not within `delta` of each other.
`assert_throws( symbol, [msg] ) { block }`	Ensures that the given block throws the symbol.
`assert_raises( exception1, exception2, ... ) { block }`	Ensures that the given block raises one of the given exceptions.
`assert_nothing_raised( exception1, exception2, ... ) { block }`	Ensures that the given block doesn't raise one of the given exceptions.
`assert_instance_of( class, obj, [msg] )`	Ensures that `obj` is an instance of `class`.
`assert_not_instance_of( class, obj, [msg] )`	Ensures that `obj` is not an instance of `class`.

`assert_kind_of( class, obj, [msg] )`	Ensures that `obj` is or descends from `class`.
`assert_not_kind_of( class, obj, [msg] )`	Ensures that `obj` is not an instance of `class` and is not descending from it.
`assert_respond_to( obj, symbol, [msg] )`	Ensures that `obj` responds to `symbol`.
`assert_not_respond_to( obj, symbol, [msg] )`	Ensures that `obj` does not respond to `symbol`.
`assert_operator( obj1, operator, [obj2], [msg] )`	Ensures that `obj1.operator(obj2)` is true.
`assert_not_operator( obj1, operator, [obj2], [msg] )`	Ensures that `obj1.operator(obj2)` is false.
`assert_send( array, [msg] )`	Ensures that executing the method listed in `array[1]` on the object in `array[0]` with the parameters of `array[2 and up]` is true. This one is weird eh?
`flunk( [msg] )`	Ensures failure. This is useful to explicitly mark a test that isn't finished yet.

Rails-Specific Assertions

Rails adds some custom assertions of its own to the `test/unit` framework:

Assertion	Purpose
`assert_difference(expressions, difference = 1, message = nil) {...}`	Test numeric difference between the return value of an expression as a result of what is evaluated in the yielded block.
`assert_no_difference(expressions, message = nil, &amp;block)`	Asserts that the numeric result of evaluating an expression is not changed before and after invoking the passed in block.
`assert_recognizes(expected_options, path, extras={}, message=nil)`	Asserts that the routing of the given path was handled correctly and that the parsed options (given in the expected_options hash) match path. Basically, it asserts that Rails recognizes the route given by expected_options.
assert_generates(expected_path, options, defaults={}, extras = {}, message=nil)	Asserts that the provided options can be used to generate the provided path. This is the inverse of assert_recognizes. The extras parameter is used to tell the request the names and values of additional request parameters that would be in a query string.

	The message parameter allows you to specify a custom error message for assertion failures.
`assert_response(type, message = nil)`	Asserts that the response comes with a specific status code. You can specify `:success` to indicate 200-299, `:redirect`to indicate 300-399, `:missing` to indicate 404, or `:error` to match the 500-599 range. You can also pass an explicit status number or its symbolic equivalent. For more information, see full list of status codes and how their mapping works.
`assert_redirected_to(options = {}, message=nil)`	Assert that the redirection options passed in match those of the redirect called in the latest action. This match can be partial, such that `assert_redirected_to(controller: "weblog")` will also match the redirection of `redirect_to(controller: "weblog", action: "show")` and so on. You can also pass named routes such as `assert_redirected_to root_path` and Active Record objects such as `assert_redirected_to @article`.
`assert_template(expected = nil, message=nil)`	Asserts that the request was rendered with the appropriate template file.

`assert_select_email`	Allows you to make assertions on the body of an e-mail.
`assert_select_encoded`	Allows you to make assertions on encoded HTML. It does this by un-encoding the contents of each element and then calling the block with all the un-encoded elements.
`css_select(selector)`or `css_select(element, selector)`	

Integration Testing Helpers

In addition to the standard testing helpers, there are some additional helpers available to integration tests:

Helper	Purpose
`https?`	Returns `true` if the session is mimicking a secure HTTPS request.
`https!`	Allows you to mimic a secure HTTPS request.
`host!`	Allows you to set the host name to use in the next request.
`redirect?`	Returns `true` if the last request was a redirect.
`follow_redirect!`	Follows a single redirect response.
`request_via_redirect(http_method, path, [parameters], [headers])`	Allows you to make an HTTP request and follow any subsequent redirects.
`post_via_redirect(path, [parameters], [headers])`	Allows you to make an HTTP POST request and follow any subsequent redirects.

`get_via_redirect(path, [parameters], [headers])`	Allows you to make an HTTP GET request and follow any subsequent redirects.
`patch_via_redirect(path, [parameters], [headers])`	Allows you to make an HTTP PATCH request and follow any subsequent redirects.
`put_via_redirect(path, [parameters], [headers])`	Allows you to make an HTTP PUT request and follow any subsequent redirects.
`delete_via_redirect(path, [parameters], [headers])`	Allows you to make an HTTP DELETE request and follow any subsequent redirects.
`open_session`	Opens a new session instance.

Rake Tasks for Running Tests

You don't need to set up and run your tests by hand on a test-by-test basis. Rails comes with a number of commands to help in testing. The table below lists all commands that come along in the default Rakefile when you initiate a Rails project.

Tasks	Description
`rake test`	Runs all unit, functional and integration tests. You can also simply run `rake` as Rails will run all the tests by default
`rake test:controllers`	Runs all the controller tests from `test/controllers`
`rake test:functionals`	Runs all the functional tests from `test/controllers`, `test/mailers`, and `test/functional`
`rake test:helpers`	Runs all the helper tests from `test/helpers`
`rake test:integration`	Runs all the integration tests from `test/integration`
`rake test:mailers`	Runs all the mailer tests from `test/mailers`
`rake test:models`	Runs all the model tests from `test/models`

`rake test:units`	Runs all the unit tests from `test/models`, `test/helpers`, and `test/unit`
`rake test:all`	Runs all tests quickly by merging all types and not resetting db
`rake test:all:db`	Runs all tests quickly by merging all types and resetting db