TEST::UNIT

Ruby Cheatsheet

1. `require "test/unit"`
2. `class MyTest < Test::Unit::TestCase`
3. Prepend your instance methods with `test_`
4. Use required assertions (see below)
5. For convenience optionally overwrite the `setup` and `teardown` methods, which initialize/destroy your test data

`assert``(boolean)`
Asserts that `boolean` is not `false` or `nil`.

`assert_equal``(expected, actual)`
Passes if `expected == actual`. Note that the ordering of arguments is important, since a helpful error message is generated when this one fails that tells you the values of expected and actual. To test for the exact opposite use **`assert_not_equal`**.

`assert_same``(expected, actual)`
Passes if `actual.equal? expected` - i.e. they are the same instance. To test for the exact opposite use **`assert_not_same`**.

`assert_nil``(object)`
Passes if `object` is `nil`. To test for the exact opposite use **`assert_not_nil`**.

`assert_raise``(*exceptions, &block)`
Passes if the `block` raises one of the given `exceptions`. To test for the exact opposite use **`assert_nothing_raised`**.

`assert_in_delta``(expected_float, actual_float, delta)`
Passes if `expected_float` and `actual_float` are equal within `delta` tolerance

`assert_respond_to``(object, method)`
Passes if `object.respond_to? method`.

`assert_instance_of``(klass, object)`
Passes if `object.instance_of? klass`.

`assert_kind_of``(klass, object)`
Passes if `object.kind_of? klass`.

`assert_match``(pattern, string)`
Passes if `string =~ pattern`. To test for the exact opposite use **`assert_no_match`**.

`assert_operator``(object1, operator, object2)`
Compares the `object1` with `object2` using `operator`. Passes if `object1.send(operator, object2) == true`.

`assert_send``(send_array)`
Passes if the method send returns a true value. `send_array` is composed of: 1. the receiver, 2. a method, 3. arguments to the method. Example: `assert_send ["string", :include, "str"]`

`assert_throws``(expected_symbol, &block)`
Passes if the `block` throws `expected_symbol`. To test for the exact opposite use **`assert_nothing_thrown`**.

`assert_block``(message="assert_block failed.", &block)`
Passes if the `block` yields `true`. All other assertions are based on this one.

Detailed messages

Every assert method can have an additional last argument named **`message`** which you can pass to give a more detailed failure message. This is omitted here for an easier overview.

TestHelper

You can do it like Rails and define a module in **`test_helper.rb`** which you require in each test file. This helper will require all the files that need to be tested in one central place.

Take it easy

Keep each test method short and concise. Try to not use more than one assertion at once.
This keeps your tests focused, and when something fails you immediately see why.

Custom assertions

If you find yourself doing similar assertions again and again, better write your own assert-method that uses the basic ones listed here. Place the custom assertions in the `TestHelper` and be sure to give a meaningful failure message.