THE MINITEST
COOKBOOK

SAMPLE RECIPE

Continuous Testing with Guard

In order to use tests effectively, we need to view them as a guide that tells us when something breaks and allows us to make corrections before we get too far down the wrong road. To do that, we need to be running tests often - every few lines of code written is not a bad rule of thumb to follow.

This is also the reason why we need our tests to be fast. If your test suite runs in 20 seconds, you'll run it whenever you make a meaningful change to your code, and that will tell you if you if everything is still as it should be. But if your tests take 20 minutes to run, you might only run them once a day, and by then you might have spent hours literally breaking your application. By the time you figure this out and begin trying to fix the problem, you might have written several hundred lines of code. Try finding the problem now.

Guard is a Ruby gem that runs a background process which watches the files in your project for changes and tests to see whether your most recent changes have broken something. It only takes a few minutes to set it up, and once done, you can just set it and forget it and stop switching from your editor to a terminal every time you make a change and need to run your tests - a huge time saver in the long run.

Guard uses a configuration file (the `Guardfile`) that maps files in your project to tests that should be re-run when those files change. It's nice because you're able to tell Guard that you want only want to run one test when you change a Ruby source file in your `lib/` directory but that a change to `test_helper.rb` should run all tests.

Guard maintains a plugin for Minitest that makes setting up a snap. If you're working on a Ruby gem project, you'll want to add the guard-minitest gem to your gemspec as a development dependency by adding the following line:

```
spec.add_development_dependency "guard-minitest"
```

If you're working on a non-gem project, just add the gem to your `Gemfile` with the line:

```
gem 'guard-minitest'
```

In both cases, update your bundle by running `bundle` from the command line.

Next, you'll create a default `Guardfile` by entering:

```
guard init
```

The template used for the default `Guardfile` is part of the guard-minitest gem and contains a lot of extra stuff that you won't need here. You can remove all the comments and cut the file down to just the following:

```
guard :minitest do
  watch(%r{^test/(.*)\/?test_(.*)\.rb$})
  watch(%r{^lib/(.*/)?([^/]+)\.rb$})     { |m| "test/#{m[1]}test_#{m[2]}.rb" }
  watch(%r{^test/test_helper\.rb$})      { 'test' }
end
```

Let's break down what each of these lines is doing for us:

- Line 1 activates the guard-minitest plugin which acts as an adapter that translates Guard events into Minitest tests run or messages displayed to the user.
- Line 2 tells Guard to watch all test files under the `test/` directory for changes and to re-run files that have changed.
- Line 3 uses a regular expression to map the files under the `lib/` directory to corresponding test files under the `test/` directory. This time, we pass a regular expression with two capture groups: one for the directory path under `lib/` and another for the filename excluding the `.rb` extension. The resulting Ruby `MatchData` object is passed as a variable to the block argument which uses the matched text to yield the path to the test.
- Line 4 maps the `test/test_helper.rb` file using a regular expression (which could just as easily be a string) and indicates that Guard should run all files under `test/` when it detects a change.

You may want to tweak or add to the watches in the `Guardfile` depending on the structure of your project, but for most typical gems and standalone Ruby applications and scripts, this should be enough to get you started. Fire up Guard by going back to the terminal and typing:

```
guard
```

And now you should be off and running. Guard gives you a console application (using the Pry gem) that allows you to see the results of tests when they are run and to interact with the running process in a number of ways. (Type `h` for a full list of available options, and there are a lot of them.) A few of the more useful ones I found were:

- `<return>` or `all` or `a` - runs all tests
- `pause` or `p` – toggles the process listening for project changes
- `notification` or `n` – toggles desktop notifications for systems that support them
- `wtf?` - shows the backtrace of the most recently thrown exception
- `exit` or `e` – exits the Guard process

Resources:

- Guard – website, GitHub repo
- Guard::Minitest – Github repo