Automating your developer workflow for faster feedback loops

Sabrina Gannon

Who am I?

► I'm a senior Ruby on Rails developer who just wrapped a round of job seeking.

Who am I?

- ► I'm a senior Ruby on Rails developer who just wrapped a round of job seeking.
- ► I really enjoy having lots of projects on the go, both technical and non technical.

Who am I?

- I'm a senior Ruby on Rails developer who just wrapped a round of job seeking.
- ► I really enjoy having lots of projects on the go, both technical and non technical.
- ▶ I struggle with both getting started on new tasks as well as staying focused.

Why this topic?

▶ Automation is extensible and reduces context switching.

Why this topic?

- ▶ Automation is extensible and reduces context switching.
- ► Writing small utilities is a learning opportunity (fun!)

Why this topic?

- Automation is extensible and reduces context switching.
- Writing small utilities is a learning opportunity (fun!)
- Faster feedback loops mean less mental overhead needing to run a command yourself.

Why not this topic

▶ You don't need to be more productive so you can do more work.

Why **not** this topic?

- ▶ You don't need to be more productive so you can do more work.
- ▶ Not everything needs to be automated.

Let's look at some examples!

Let's say I want to write my slides in a markdown file, and generate a PDF output of those slides to present with.

I can do that using the following command using the Pandoc document converter:

pandoc -t beamer slides.md -o slides.pdf

Rerunning the command starts to feel tiresome; I'm still learning Pandoc and want to see my changes quickly, but I have to rerun the command in my terminal every time.

The feedback loop I'm working with is:

1. Make a change

The feedback loop I'm working with is:

- 1. Make a change
- 2. Regenerate PDF

The feedback loop I'm working with is:

- 1. Make a change
- 2. Regenerate PDF
- 3. See the effect of my changes.

If I can always have the PDF version available when I make a change to my Markdown file, that would remove a manual step that slows down the feedback loop.

The feedback loop I want is:

1. Make a change

The feedback loop I want is:

- 1. Make a change
- 2. See the effect of my changes.

Installing the Guard gem

Let's install Guard by adding a Gemfile to the project directory with the following:

```
# Gemfile
source "https://rubygems.org"
gem "guard"
gem "guard-shell"
```

We can then install our gems with bundle exec

Initializing the Guard gem

Guard is configured using a Guardfile, which is generated by running the command *guard init*.

However, because I'm using the shell plugin to run my shell command, what I actually type in to initialize my file is:

guard shell init

Configuring the Guard gem

Here's what the complete Guardfile looks like:

```
# Guardfile
# m[0] is the entire filename string "slides.md"
# uses Ruby's File utilities to remove the
# '.md' file extension when naming the pdf
guard :shell do
  watch(/.*\.md$/) do |m|
    `pandoc -t beamer #{m[0]} -o #{File.basename(m[0], Fil
  end
end
```

Running the Guard gem

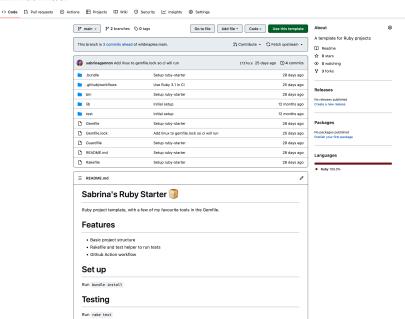
Run bundle exec guard to start watching files in the project directory.

When a markdown file is changed, the following output runs in my terminal:

21:31:53 - INFO - Guard is now watching at '/Users/sabrina, [1] guard(main)>

Template Ruby project with all of my favourite gems

sabrinagannon/ruby-starter Public template



```
# Guardfile
guard 'rake', :task => 'test' do
watch(%r{^lib/(.+)\.rb})
watch(%r{^test/(.+)\.rb})
end
```

```
# Rakefile
require "rake/testtask"

Rake::TestTask.new(:test) do |t|
t.libs << "test"
t.test_files = FileList["test/**/*_test.rb"]
end

task :default => :test
```

▶ Q: What if I don't need to run my entire test suite on every change?

- Q: What if I don't need to run my entire test suite on every change?
- ► A1: Configure guard to run only the associated test file for an edited file.

- Q: What if I don't need to run my entire test suite on every change?
- ► A1: Configure guard to run only the associated test file for an edited file.
- ▶ A2: Use a tool like the Focus gem to run a single test at a time.

Snippet to load gems inline in a single file Ruby script

```
#!/usr/bin/env ruby
require "bundler/inline"
gemfile do
  source "https://rubygems.org"
  gem "pry"
end
require "pry"
# Your ruby script
Credit to Matheus Richard @matheusrich's tweet
```

How developer automations can be useful

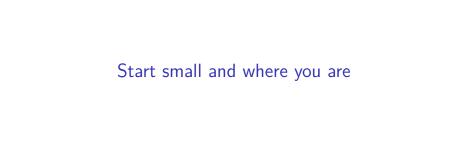
▶ Prepping for technical interviews.

How developer automations can be useful

- Prepping for technical interviews.
- ▶ "Test Driven Development" workflows.

How developer automations can be useful

- Prepping for technical interviews.
- "Test Driven Development" workflows.
- Automating tedious work to free up time in your daily life and aid with other simple tasks.



Where you can go from here

► Editor integrations (ex. VSCode task runner)

Where you can go from here

- Editor integrations (ex. VSCode task runner)
- Github Actions and similar platforms can run automated commands (ex. Continuous integration for personal projects)

Where you can go from here

- Editor integrations (ex. VSCode task runner)
- Github Actions and similar platforms can run automated commands (ex. Continuous integration for personal projects)
- Turning an automation into something available on a webpage (ex. I did this with a plant watering tracker)

References

```
https://graceful.dev/courses/acapa/
https://pandoc.org/MANUAL.html
https://github.com/guard/guard
https://github.com/guard/guard-shell
https://github.com/sabrinagannon/ruby-starter
https://twitter.com/matheusrich/status/1536429683591585792
https://github.com/guard/guard-shell#check-syntax-of-a-ruby-
file
```

Thank you!

Slides are available at:

https://github.com/sabrinagannon/automating-your-dev-workflow

I'm online on Wnbrb slack or @deusmxsabrina on Twitter if you'd like to share automations you use!