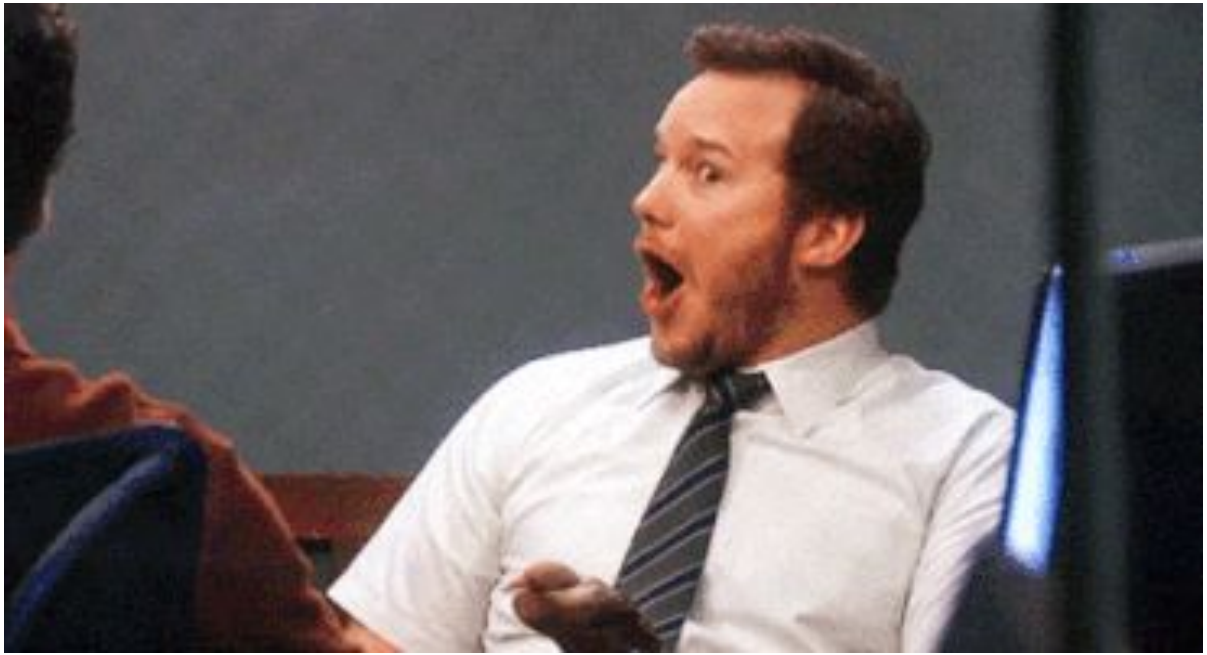
Policing ruby with RuboCop

Panuza Parajuli

Full stack software developer at Houzz

What if you could write a sloppy code and have ruby fix it for you automatically?



RuboCop

- RuboCop is a Ruby code style checker (linter) and formatter based on the community-driven Ruby Style Guide.
- RuboCop is extremely flexible.
- In practice RuboCop supports pretty much every (reasonably popular) coding style that you can think of.

Cops

Each individual check on RuboCop are defined as **cops**, and each one is responsible for detecting a specific offense.

- **Style cops**
- **Layout cops**
- **Lint cops**
- **Metric cops**
- **Naming cops**
- **Security cops**
- **Bundler cops**

Style Cops

Detect code that works but does not meet a consistent style.

For example:

Puts if !test

```
[→  blog git:(master) × rubocop app/controllers/test_rubocop_controller.rb
Inspecting 1 file
C

Offenses:

app/controllers/test_rubocop_controller.rb:6:5: C: [Correctable] Style/NegatedIf: Favor unless over if for negative conditions.
    puts if !test
    ^^^^^^^^^^^^

1 file inspected, 1 offense detected, 1 offense autocorrectable
```

Layout cops

Catch issues related to formatting, such as the use of white space.

For example:

```ruby
class TestRubocopController < ApplicationController
  def test
    puts 'new'

  end
end
```

```
[➜  blog git:(master) ✗ rubocop app/controllers/test_rubocop_controller.rb
Inspecting 1 file
C

Offenses:

app/controllers/test_rubocop_controller.rb:7:1: C: [Correctable] Layout/EmptyLinesAroundMethodBody: E
xtra empty line detected at method body end.
app/controllers/test_rubocop_controller.rb:7:1: C: [Correctable] Layout/TrailingWhitespace: Trailing
whitespace detected.

1 file inspected, 2 offenses detected, 2 offenses autocorrectable
```

Lint cops

Detect possible errors in your code that doesn't make sense and is probably wrong.

Example:

```ruby
class TestRubocopController < ApplicationController
  def test(id)
    puts 'new'
  end
end
```

```
app/controllers/test_rubocop_controller.rb:5:18: W: [Correctable] Lint/UnusedMethodArgument: Unused m
ethod argument - id. If it's necessary, use _ or _id as an argument name to indicate that it won't be
 used. You can also write as new_method(*) if you want the method to accept any arguments but don't c
are about them.
  def new_method(id)
                 ^^

1 file inspected, 1 offense detected, 1 offense autocorrectable
```

Metric cops

Deals with issues related to source code measurements such as class length and method length.

Example:

```ruby
def really_long_method_name_that_does_not_looks_good
  # Too many lines
```

```
Inspecting 1 file
C

Offenses:

app/controllers/test_rubocop_controller.rb:5:3: C: Metrics/MethodLength: Method has too many lines. [18/10]
  def really_long_method_name_that_does_not_looks_good ...
  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

1 file inspected, 1 offense detected
```

Naming cops

These cops are concerned with naming conventions.

Example:

```ruby
def get_name
  puts 'test'
end
```

```
Inspecting 1 file
C

Offenses:

app/controllers/test_rubocop_controller.rb:5:7: C: Naming/AccessorMethodName: Do not prefix reader method names with get_.
  def get_name
      ^^^^^^^^

1 file inspected, 1 offense detected
```

Security cops

Checks for implementations of the `hash` method which combine values using custom logic instead of delegating to `Array#hash`. They help with catching potential security issues.

Example:

```ruby
def name
  JSON.load['{}']
end
```

```
→  blog git:(master) ✗ rubocop app/controllers/test_rubocop_controller.rb
Inspecting 1 file
C

Offenses:

app/controllers/test_rubocop_controller.rb:6:10: C: [Correctable] Security/JSONLoad: Prefer JSON.pars
e over JSON.load.
    JSON.load('{}')
         ^^^^

1 file inspected, 1 offense detected, 1 offense autocorrectable
→  blog git:(master) ✗
```

Bundler cops

Check for bad practices in Bundler files (such as `Gemfile`).

Example:

```
group :development do
  # Access an interactive console on exception pages or by calling 'console' anywhere in the code.
  gem 'listen', '>= 3.0.5', '< 3.2'
  gem 'rubocop-rails'
  gem 'rubocop'
  gem 'rubocop-rake'
```

```
Inspecting 1 file
C

Offenses:

Gemfile:50:3: C: [Correctable] Bundler/OrderedGems: Gems should be sorted in an alphabetical order wi
thin their section of the Gemfile. Gem rubocop should appear before rubocop-rails.
  gem 'rubocop'
  ^^^^^^^^^^^^

1 file inspected, 1 offense detected, 1 offense autocorrectable
```

Benefits:

- You can **track silly errors quickly** without burrowing through many lines of the error stack
- You can **aid your review process by smashing all the errors and style deficiencies before they get pushed to a remote branch**. This helps to avoid warnings and complaints from your colleague with an eye for detail.

Installation & Configuration

RuboCop's installation is pretty standard:

```
$ gem install rubocop
```

If you'd rather install RuboCop using `bundler`, add a line for it in your `Gemfile` (but set the `require` option to `false`, as it is a standalone tool):

```
gem 'rubocop', require: false
```

RuboCop can be configured through a *.rubocop.yml* file placed at the root of your project.

Usage:

Running rubocop with no arguments will check all Ruby source files in the current directory:

```
$ rubocop
```

Alternatively you can specify a list of files and directories to check:

```
$ rubocop app spec lib/something.rb
```

You can auto-correct offenses with rubocop -a:

```
$ rubocop -a
```

RuboCop can do way more. Use -h to view all available command-line options.

Suppressing Rubocop Errors

Although RuboCop is great tool, it can yield false positives from time to time or suggest fixing the code in a way that is detrimental to the intent of the programmer.

You can mention the individual cops or departments to be disabled, as shown below:

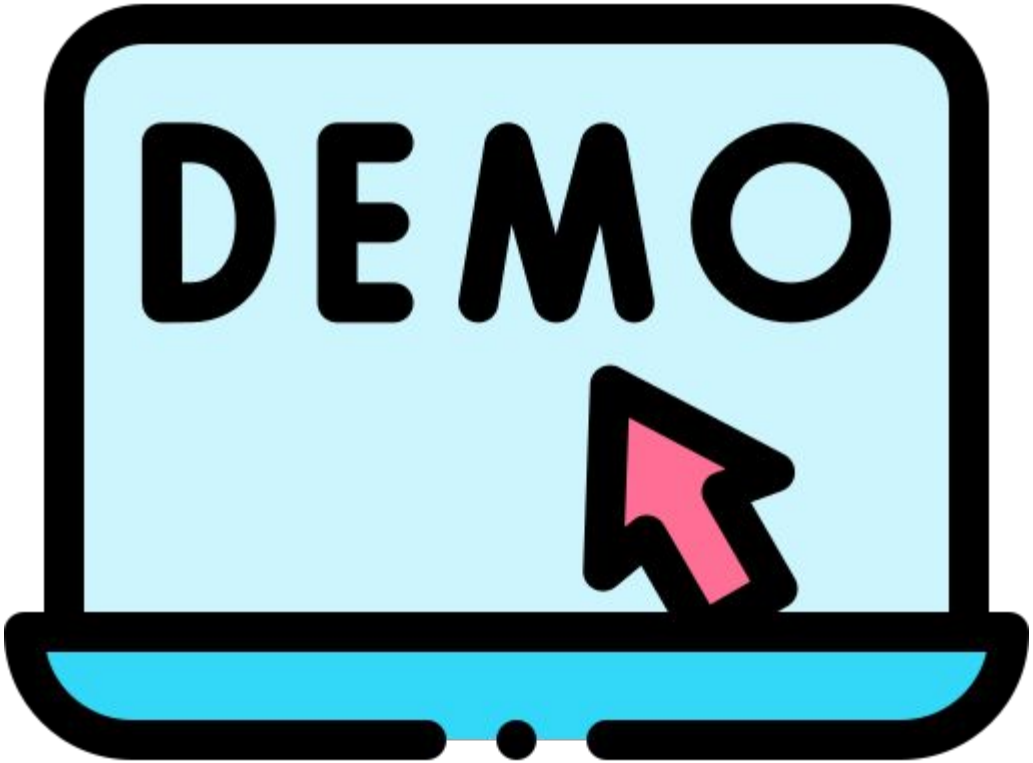
```
# rubocop:disable Layout/LineLength, Style
```

[..]

Or you can disable all cops for a section of the code in one fell swoop:

```
# rubocop:disable all
```

[..]


DEMO

Conclusion

RuboCop code brings **so many benefits to a code base**, especially in the context of a **team of developers.** Even if you don't like being told how to format your code, you need to keep in mind that **rubocop** isn't just for you.

It is also for the other people you collaborate with so that everyone can stick to the same conventions, thus eliminating the drawbacks of dealing with multiple coding styles in the same project.

Alternative Auto Formatters

Prettier

Prettier started out as an opinionated code formatter for JavaScript, but it now supports many other languages, including Ruby.

RubyFmt

RubyFmt is a brand-new code formatter that's written in Rust and currently under active development.

Any Questions?

References

https://docs.rubocop.org/rubocop/index.html

https://www.honeybadger.io/blog/linting-formatting-ruby/

https://www.netguru.com/blog/how-rubocop-can-ease-your-code