# Contributing to Reek

We welcome any and all contributions to Reek!

If what you’re proposing requires significant work discuss it beforehand

as an issue – it’s much easier for us to guide you towards a good

solution and less frustrating for you than doing a lot of work

only to find us suggesting large parts should be rewritten.

Don’t hesitate to offer trivial fixes (spelling, better naming

ideas, etc.) – we’ll let you know if you’re overdoing it. :)

## Reporting Issues

Search [all existing (open \_and\_ closed)

issues](https://github.com/troessner/reek/issues?q=is%3Aissue)

for a possible previous report; comment there if the issue is not

actually resolved properly or you have any additional information.

Include the steps to reproduce the issue,

the expected outcome and the actual outcome.

Include as much information as possible: the exact Reek

invocation that you use, Reek’s config and version, Ruby

version, Ruby platform (MRI, JRuby, etc.), operating system.

Try to provide a minimal example that reproduces the issue.

Extra kudos if you can write it as a failing test. :)

## Contributing features, bugfixes, documentation

### Getting started

Fork Reek, then clone it, make sure you have

[Bundler](http://bundler.io) installed, install dependencies

and make sure all of the existing tests pass:

```

git clone git@github.com:…username…/reek.git

cd reek

gem install bundler

bundle

bundle exec rake

```

Once you’re sure your copy of Reek works create your own feature branch from our "master" branch:

```

git checkout -b your\_feature\_or\_fix\_name

```

Make sure you have read our [style guide](docs/Style-Guide.md) before you

start contributing.

Then start hacking and add new tests which make sure that your new feature works or

demonstrate that your fix was needed.

### RSpec Specs

Reek uses [Rspec](http://rspec.info/) for unit and functional testing.

We're trying to follow [betterspecs](http://betterspecs.org/). We're not using

RSpec's

[shared examples](https://www.relishapp.com/rspec/rspec-core/docs/example-groups/shared-examples)

because we find them rather harming than helpful. You can find an excellent

cheat sheet on how to write idiomatic Rspec

[here](http://www.rubypigeon.com/posts/rspec-core-cheat-sheet).

We do not use the popular "foo" / "bar" naming when it comes to the question

"how to come up with good example names?". Instead, we use the

[military alphabet](https://en.wikipedia.org/wiki/NATO\_phonetic\_alphabet) in

ascending order which means that we would write this

```Ruby

class Foo

def bar(baz)

baz.quux

end

end

```

rather like this:

```Ruby

class Alfa

def bravo(charlie)

charlie.delta

end

end

```

We also aim to make our specs as self-reliant as possible. They should not rely on external sample

or configuration files since that makes it very hard to reason about them and introduces a tight coupling

which makes refactoring more difficult.

Unfortunately our current specs are a bit misleading in this regard since they are often doing the

opposite, relying on external sample and configuration files (indicated by using

some of the constants defined in `samples/paths.rb`).

Our goal is to reverse this development.

### Cucumber Features

Reek uses [Cucumber](https://cucumber.io/) with

[Aruba](https://github.com/cucumber/aruba) for integration tests. Keep the

following in mind when writing cucumber features.

#### What to test

Not everything needs a cucumber feature. We try to limit cucumber features to

things that really require end-to-end testing of Reek's behavior. In

particular, this means individual smell detectors should not have their own

scenarios.

#### TTY output checks

Some default behaviors of Reek depend on whether the output is a TTY, for

example output coloring and the progress bar. Because under Aruba stdout is

\*not\* a TTY, Reeks default behavior in the scenarios is different in this

regard than if it were run in a terminal.

#### Failing Cucumber Scenarios

If there is a failing scenario and you cannot figure out why it is failing,

just run the failing scenario: `bundle exec cucumber

features/failing\_scenario.feature:line`. By doing so Aruba will leave its set

up in the `tmp/aruba` directory. You can then `cd` into this directory and run

Reek the same way the cucumber scenario actually ran it. This way you can debug

scenario failures that can be very opaque sometimes.

### Writing new smell detectors

Please see [our separate guide](docs/How-To-Write-New-Detectors.md) for this.

### Creating your pull request

We care a lot about [good commit

messages](http://tbaggery.com/2008/04/19/a-note-about-git-commit-messages.html).

Once you’re happy with your feature / fix – or want to

share it as a work-in-progress and request comments – once

again make sure all of the tests pass. This will also run

[RuboCop](https://github.com/bbatsov/rubocop) – fix any

offences RuboCop finds (or discuss them in the pull request):

```

bundle exec rake

```

Once your code is ready for review push it to your repository:

```

git push -u origin

```

Then go to your GitHub fork and [make a pull

request](https://help.github.com/articles/creating-a-pull-request/)

to the original repository.

### Review and Fixes

Try to gauge and let us know in the pull request whether what

you propose is a backward-compatible bugfix and should go into the

next patch release, is a backward-compatible feature and should go

into the next minor release, or has to break backward-compatibility

and so needs to wait for the next major release of Reek. See also our

versioning policy below.

Once your PR is open someone will review it, discuss the details (if

needed) and either merge right away or ask for some further fixes.

GitHub doesn’t have a good way to amend pull requests from external

repositories. In cases where it’s easier to make a direct fix

to your pull request (rather than describing it in the comments)

we’ll either open a separate pull request to the branch in your

repository or ask you to give us commit access to your repository.

If there were any fixes to your pull request we’ll ask you to

[squash](http://git-scm.com/book/en/v2/Git-Tools-Rewriting-History#Squashing-Commits)

all of the commits into one:

```

git rebase -i master

# squash squash squash

git push -f origin

```

## Versioning policy

We are following [semantic versioning](http://semver.org/).

If you're working on a change that is breaking backwards-compatibility

just go ahead with your pull request like normal. We'll discuss this then in

the pull request and help you to point your pull request to the right branch.

### Releasing Reek

In this example we assume the current version is 5.3.1 and you want to update to 5.3.2.

\* Create a branch with a name like "prepare-v.5.3.2"

\* Update the version in `lib/reek/version.rb`

\* List all relevant changes in `CHANGELOG.md`

\* Update the version number in our cucumber features, otherwise the build will fail. You can do this quite easily via

```Bash

find features/ -type f -exec sed -i '' 's/v5.3.1/v5.3.2/g' {} +

```

\* Push the branch, create a pull request, have it reviewed and merged

\* Pull the latest master and then do a

```Bash

bundle exec rake release

```