**SimpleCov [Build Status](http://travis-ci.org/colszowka/simplecov) [Dependency Status](https://gemnasium.com/colszowka/simplecov) [Code Climate](https://codeclimate.com/github/colszowka/simplecov)**

**Code coverage for Ruby 1.9**

* [Source Code](https://github.com/colszowka/simplecov)
* [API documentation](http://rubydoc.info/gems/simplecov/frames)
* [Changelog](https://github.com/colszowka/simplecov/blob/master/CHANGELOG.md)
* [Rubygem](http://rubygems.org/gems/simplecov)
* [Mailing List](https://groups.google.com/forum/#!forum/simplecov)
* [Continuous Integration](http://travis-ci.org/colszowka/simplecov)

[You can support the development of SimpleCov via Pledgie - thanks for your help](http://www.pledgie.com/campaigns/18379)

SimpleCov is a code coverage analysis tool for Ruby 1.9. It uses [1.9's built-in Coverage](http://www.ruby-doc.org/stdlib-1.9.3/libdoc/coverage/rdoc/Coverage.html) library to gather code coverage data, but makes processing its results much easier by providing a clean API to filter, group, merge, format and display those results, thus giving you a complete code coverage suite that can be set up with just a couple lines of code.

In most cases, you'll want overall coverage results for your projects, including all types of tests, cucumber features etc. SimpleCov automatically takes care of this by caching and then merging results when generating reports, so your report actually includes coverage across your test suites and thereby gives you a better picture of blank spots.

The official formatter of SimpleCov is packaged as a separate gem called [simplecov-html](https://github.com/colszowka/simplecov-html" \o "SimpleCov HTML Formatter Source Code @ GitHub) but will be installed and configured automatically when you launch SimpleCov. If you're curious, you can find it [on Github, too](https://github.com/colszowka/simplecov-html).

**Getting started**

1. Add SimpleCov to your Gemfile and bundle install:

gem 'simplecov', :require **=>** **false**, :group **=>** :test

1. Load and launch SimpleCov **at the very top** of your test/test\_helper.rb (*or spec\_helper.rb, cucumber env.rb, or whatever your preferred test framework uses*):

require 'simplecov'

SimpleCov**.**start

*# Previous content of test helper now starts here*

**Note:** If SimpleCov starts after your application code is already loaded (via require), it won't be able to track your files and their coverage! The SimpleCov.start **must** be issued **before any of your application code is required!**

SimpleCov must be running in the process that you want the code coverage analysis to happen on. When testing a server process (i.e. a JSON API endpoint) via a separate test process (i.e. when using Selenium) where you want to see all code executed by therails server, and not just code executed in your actual test files, you'll want to add something like this to the top ofscript/rails:

**if** ENV**[**'RAILS\_ENV'**]** **==** 'test'

require 'simplecov'

SimpleCov**.**start 'rails'

puts "required simplecov"

**end**

1. Run your tests, open up coverage/index.html in your browser and check out what you've missed so far.
2. Add the following to your .gitignore file to ensure that coverage results are not tracked by Git (optional):

coverage

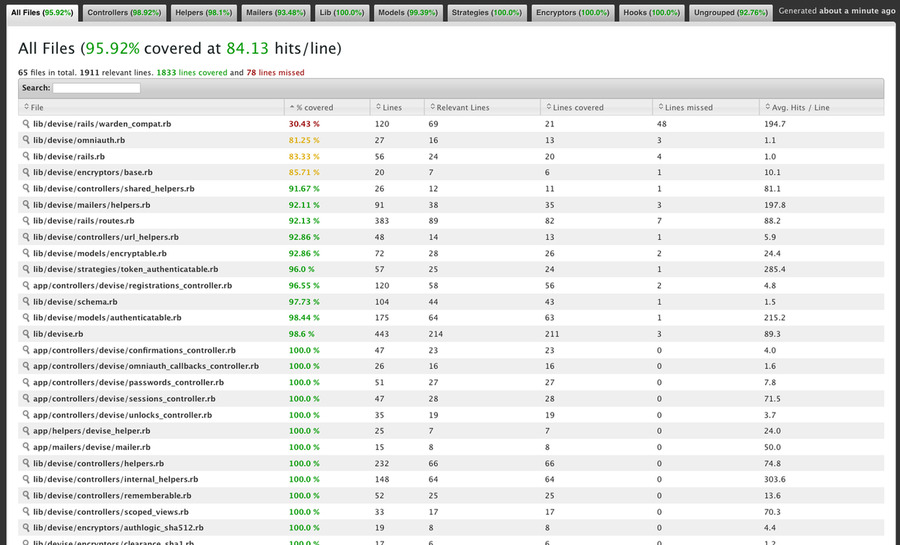
If you're making a Rails application, SimpleCov comes with a built-in configurations (see below for information on profiles) which will get you started with groups for your Controllers, Views, Models and Helpers. To use it, the first two lines of your test\_helper should be like this:

require 'simplecov'

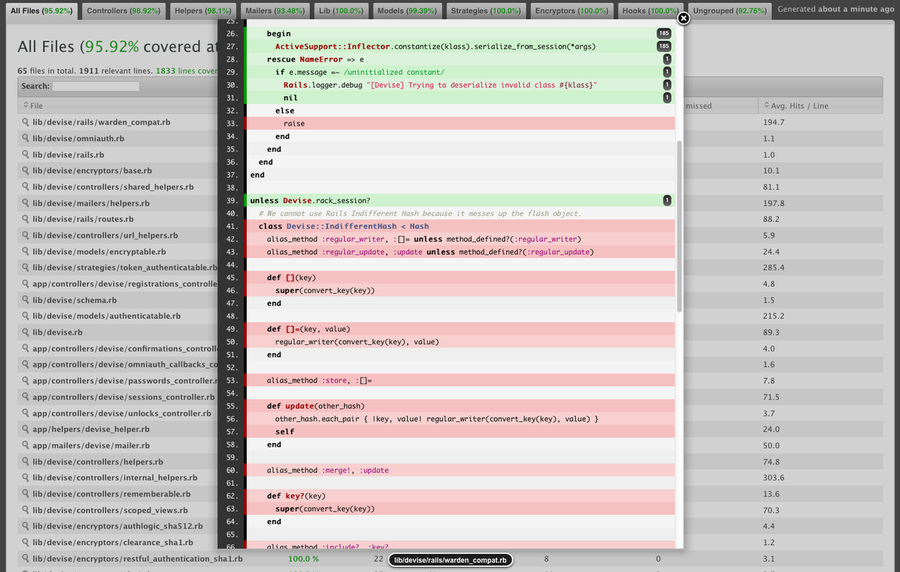
SimpleCov**.**start 'rails'

**Example output**

**Coverage results report, fully browsable locally with sorting and much more:**

[](https://github-camo.global.ssl.fastly.net/5d9b874a778392ff6180d297faa362f0b32fcd62/687474703a2f2f636f6c737a6f776b612e6769746875622e636f6d2f73696d706c65636f762f6465766973655f726573756c742d302e352e332e706e67)

**Source file coverage details view:**

[](https://github-camo.global.ssl.fastly.net/3cb7252450587d575bca65e27f20107b1986d67e/687474703a2f2f636f6c737a6f776b612e6769746875622e636f6d2f73696d706c65636f762f6465766973655f736f757263655f66696c652d302e352e332e706e67)

**Use it with any framework!**

Similarily to the usage with Test::Unit described above, the only thing you have to do is to add the simplecov config to the very top of your Cucumber/RSpec/whatever setup file.

Add the setup code to the **top** of features/support/env.rb (for Cucumber) or spec/spec\_helper.rb (for RSpec). Other test frameworks should work accordingly, whatever their setup file may be:

require 'simplecov'

SimpleCov**.**start 'rails'

You could even track what kind of code your UI testers are touching if you want to go overboard with things. SimpleCov does not care what kind of framework it is running in, it just looks at what code is being executed and generates a report about it.

**Notes on specific frameworks and test utilities**

For some frameworks and testing tools there are quirks and problems you might want to know about if you want to use SimpleCov with them. Here's an overview of the known ones:

|  |  |  |
| --- | --- | --- |
| **Framework** | **Notes** | **Issue #** |
| **Test/Unit 2** | Test Unit 2 used to mess with ARGV, leading to failure to detect the test process name in SimpleCov. test-unit releases 2.4.3+ (Dec 11th, 2011) should have this problem resolved. | [SimpleCov #45](https://github.com/colszowka/simplecov/issues/45) &[Test/Unit #12](https://github.com/test-unit/test-unit/pull/12) |
| **Spork** | Because of the how Spork works internally (using preforking) there used to be trouble when using SimpleCov with it, but that apparently has been resolved with a specific configuration strategy. See [this](https://github.com/colszowka/simplecov/issues/42#issuecomment-4440284) comment. | [SimpleCov #42](https://github.com/colszowka/simplecov/issues/42#issuecomment-4440284) |
| **parallel\_tests** | SimpleCov does not detect parallel\_test automatically yet but can be taught to do so with a simple workaround explained at Issue #64. | [SimpleCov #64](https://github.com/colszowka/simplecov/issues/64) |
| **Riot** | A user has reported problems with the coverage report using the riot framework. If you experience similar trouble please follow up on the related Github issue. | [SimpleCov #80](https://github.com/colszowka/simplecov/issues/80) |
| **RubyMine** | The [RubyMine IDE](https://www.jetbrains.com/ruby/) has built-in support for SimpleCov's coverage reports, though you might need to explicitly set the output root using `SimpleCov.root('foo/bar/baz')` | [SimpleCov #95](https://github.com/colszowka/simplecov/issues/95) |

**Configuring SimpleCov**

[Configuration](http://rubydoc.info/gems/simplecov/SimpleCov/Configuration) settings can be applied in three formats, which are completely equivalent:

* The most common way is to configure it directly in your start block:

SimpleCov**.**start **do**

some\_config\_option 'foo'

**end**

* You can also set all configuration options directly:

SimpleCov**.**some\_config\_option 'foo'

* If you do not want to start coverage immediately after launch or want to add additional configuration later on in a concise way, use:

SimpleCov**.**configure **do**

some\_config\_option 'foo'

**end**

Please check out the [Configuration](http://rubydoc.info/gems/simplecov/SimpleCov/Configuration) API documentation to find out what you can customize.

**Using .simplecov for centralized config**

If you use SimpleCov to merge multiple test suite results (i.e. Test/Unit and Cucumber) into a single report, you'd normally have to set up all your config options twice, once in test\_helper.rb and once in env.rb.

To avoid this, you can place a file called .simplecov in your project root. You can then just leave the require 'simplecov' in each test setup helper and move the SimpleCov.start code with all your custom config options into .simplecov:

*# test/test\_helper.rb*

require 'simplecov'

*# features/support/env.rb*

require 'simplecov'

*# .simplecov*

SimpleCov**.**start 'rails' **do**

*# any custom configs like groups and filters can be here at a central place*

**end**

**Filters**

Filters can be used to remove selected files from your coverage data. By default, a filter is applied that removes all files OUTSIDE of your project's root directory - otherwise you'd end up with a billion of coverage reports for source files in the gems you are using.

Of course you can define your own to remove things like configuration files, tests or whatever you don't need in your coverage report.

**Defining custom filters**

You can currently define a filter using either a String (that will then be Regexp-matched against each source file's path), a block or by passing in your own Filter class.

**String filter**

SimpleCov**.**start **do**

add\_filter "/test/"

**end**

This simple string filter will remove all files that match "/test/" in their path.

**Block filter**

SimpleCov**.**start **do**

add\_filter **do** **|**source\_file**|**

source\_file**.**lines**.**count **<** 5

**end**

**end**

Block filters receive a SimpleCov::SourceFile instance and expect your block to return either true (if the file is to be removed from the result) or false (if the result should be kept). Please check out the RDoc for SimpleCov::SourceFile to learn about the methods available to you. In the above example, the filter will remove all files that have less then 5 lines of code.

**Custom filter class**

**class** **LineFilter** **<** SimpleCov::Filter

**def** **matches?**(source\_file)

source\_file**.**lines**.**count **<** filter\_argument

**end**

**end**

SimpleCov**.**add\_filter LineFilter**.**new(5)

Defining your own filters is pretty easy: Just inherit from SimpleCov::Filter and define a method 'matches?(source\_file)'. When running the filter, a true return value from this method will result in the removal of the given source\_file. The filter\_argument method is being set in the SimpleCov::Filter initialize method and thus is set to 5 in this example.

**Groups**

You can separate your source files into groups. For example, in a rails app, you'll want to have separate listings for Models, Controllers, Helpers, Libs and Plugins. Group definition works similar to Filters (and indeed also accepts custom filter classes), but source files end up in a group when the filter passes (returns true), as opposed to filtering results, which exclude files from results when the filter results in a true value.

Add your groups with:

SimpleCov**.**start **do**

add\_group "Models", "app/models"

add\_group "Controllers", "app/controllers"

add\_group "Long files" **do** **|**src\_file**|**

src\_file**.**lines**.**count **>** 100

**end**

add\_group "Short files", LineFilter**.**new(5) *# Using the LineFilter class defined in Filters section above*

**end**

**Merging results**

Normally, you want to have your coverage analyzed across ALL of your test suites, right?

Simplecov automatically caches coverage results in your (coverage\_path)/.resultset.json. Those results will then be automatically merged when generating the result, so when coverage is set up properly for cucumber and your unit / functional / integration tests, all of those test suites will be taken into account when building the coverage report.

There are two things to note here though:

**Test suite names**

Simplecov tries to guess the name of the currently running test suite based upon the shell command the tests are running on. This should work fine for Unit Tests, RSpec and Cucumber. If it fails, it will use the shell command that invoked the test suite as a command name.

If you have some non-standard setup and still want nicely labeled test suites, you have to give Simplecov a cue what the name of the currently running test suite is. You can do so by specifying SimpleCov.command\_name in one test file that is part of your specific suite.

So, to customize the suite names on a Rails app (yeah, sorry for being Rails biased, but everyone knows what the structure of those projects is. You can apply this accordingly to the RSpecs in your Outlook-WebDAV-Calendar-Sync gem), you could do something like this:

*# test/unit/some\_test.rb*

SimpleCov**.**command\_name 'test:units'

*# test/functionals/some\_controller\_test.rb*

SimpleCov**.**command\_name "test:functionals"

*# test/integration/some\_integration\_test.rb*

SimpleCov**.**command\_name "test:integration"

*# features/support/env.rb*

SimpleCov**.**command\_name "features"

Note that this has only to be invoked ONCE PER TEST SUITE, so even if you have 200 unit test files, specifying it in some\_test.rb is fair enough.

[simplecov-html](https://github.com/colszowka/simplecov-html) prints the used test suites in the footer of the generated coverage report.

**Timeout for merge**

Of course, your cached coverage data is likely to become invalid at some point. Thus, result sets that are older than SimpleCov.merge\_timeout will not be used any more. By default, the timeout is 600 seconds (10 minutes), and you can raise (or lower) it by specifying SimpleCov.merge\_timeout 3600 (1 hour), or, inside a configure/start block, with just "merge\_timeout 3600".

You can deactivate merging altogether with SimpleCov.use\_merging false.

**Running coverage only on demand**

The Ruby STDLIB Coverage library that SimpleCov builds upon is *very* fast (i.e. on a ~10 min Rails test suite, the speed drop was only a couple seconds for me), and therefore it's SimpleCov's policy to just generate coverage every time you run your tests because it doesn't do your test speed any harm and you're always equipped with the latest and greatest coverage results.

Because of this, SimpleCov has no explicit built-in mechanism to run coverage only on demand.

However, you can still accomplish this very easily by introducing a ENV variable conditional into your SimpleCov setup block, like this:

SimpleCov**.**start **if** ENV**[**"COVERAGE"**]**

Then, SimpleCov will only run if you execute your tests like this:

COVERAGE=true rake test

**Profiles**

By default, Simplecov's only config assumption is that you only want coverage reports for files inside your project root. To save you from repetitive configuration, you can use predefined blocks of configuration, called 'profiles', or define your own.

You can then pass the name of the profile to be used as the first argument to SimpleCov.start. For example, simplecov comes bundled with a 'rails' profile. It looks somewhat like this:

SimpleCov**.**profiles**.**define 'rails' **do**

add\_filter '/test/'

add\_filter '/config/'

add\_group 'Controllers', 'app/controllers'

add\_group 'Models', 'app/models'

add\_group 'Helpers', 'app/helpers'

add\_group 'Libraries', 'lib'

add\_group 'Plugins', 'vendor/plugins'

**end**

As you can see, it's just a SimpleCov.configure block. In your test\_helper.rb, launch simplecov with:

SimpleCov**.**start 'rails'

**OR**

SimpleCov**.**start 'rails' **do**

*# additional config here*

**end**

**Custom profiles**

You can load additional profiles with the SimpleCov.load\_profile('xyz') method. This allows you to build upon an existing profile and customize it so you can reuse it in unit tests and cucumber features, for example.

*# lib/simplecov\_custom\_profile.rb*

require 'simplecov'

SimpleCov**.**profiles**.**define 'myprofile' **do**

load\_profile 'rails'

add\_filter 'vendor' *# Don't include vendored stuff*

**end**

*# features/support/env.rb*

require 'simplecov\_custom\_profile'

SimpleCov**.**start 'myprofile'

*# test/test\_helper.rb*

require 'simplecov\_custom\_profile'

SimpleCov**.**start 'myprofile'

**Customizing exit behaviour**

You can define what simplecov should do when your test suite finishes by customizing the at\_exit hook:

SimpleCov**.**at\_exit **do**

SimpleCov**.**result**.**format!

**end**

Above is the default behaviour. Do whatever you like instead!

**Minimum coverage**

You can define the minimum coverage percentage expected. SimpleCov will return non-zero if unmet.

SimpleCov**.**minimum\_coverage 90

**Maximum coverage drop**

You can define the maximum coverage drop percentage at once. SimpleCov will return non-zero if exceeded.

SimpleCov**.**maximum\_coverage\_drop 5

**Refuse dropping coverage**

You can also entirely refuse dropping coverage between test runs:

SimpleCov**.**refuse\_coverage\_drop

**Using your own formatter**

You can use your own formatter with:

SimpleCov**.**formatter **=** SimpleCov::Formatter**::**HTMLFormatter

When calling SimpleCov.result.format!, it will be invoked with SimpleCov::Formatter::YourFormatter.new.format(result), "result" being an instance of SimpleCov::Result. Do whatever your wish with that!

**Using multiple formatters**

If you want to use multiple result formats, as of SimpleCov 0.7.0 you can use the built-in MultiFormatter:

SimpleCov**.**formatter **=** SimpleCov::Formatter**::**MultiFormatter**[**

SimpleCov::Formatter**::**HTMLFormatter,

SimpleCov::Formatter**::**CSVFormatter,

**]**

**Available formatters**

Apart from the direct companion [simplecov-html](https://github.com/colszowka/simplecov-html" \o "SimpleCov HTML Formatter Source Code @ GitHub), there are other formatters available:

**[simplecov-rcov](https://github.com/fguillen/simplecov-rcov)**

*by Fernando Guillen*

"The target of this formatter is to cheat on Hudson so I can use the Ruby metrics plugin with SimpleCov."

**[simplecov-csv](https://github.com/fguillen/simplecov-csv)**

*by Fernando Guillen*

CSV formatter for SimpleCov code coverage tool for ruby 1.9+

**[simplecov-vim](https://github.com/nyarly/Simplecov-Vim)**

*by Judson Lester*

A formatter for Simplecov that emits a Vim script to mark up code files with coverage information.

**[simplecov-single\_file\_reporter](https://github.com/grosser/simplecov-single_file_reporter)**

*by*[*Michael Grosser*](http://grosser.it/)

A formatter that prints the coverage of the file under test when you run a single test file.

**[simplecov-json](https://github.com/vicentllongo/simplecov-json)**

*by Vicent Llongo*

JSON formatter for SimpleCov

**[simplecov-badge](https://github.com/matthew342/simplecov-badge)**

*by Matt Hale*

A formatter that generates a coverage badge for use in your project's readme using ImageMagick.

**Ruby version compatibility**

Only Ruby 1.9+ ships with the coverage library that SimpleCov depends upon. SimpleCov is built against various other Rubies, including Rubinius and JRuby, in [Continuous Integration](http://travis-ci.org/colszowka/simplecov), but this happens only to ensure that SimpleCov does not make your test suite crash right now. Whether SimpleCov will support JRuby/Rubinius in the future depends solely on whether those Ruby interpreters add the coverage library.

SimpleCov is built in [Continuous Integration](http://travis-ci.org/colszowka/simplecov) on 1.8.7, ree, 1.9.2, 1.9.3.

**Contributing**

See the [contributing guide](https://github.com/colszowka/simplecov/blob/master/CONTRIBUTING.md).

**Kudos**

Thanks to Aaron Patterson for the original idea for this!

**Copyright**

Copyright (c) 2010-2012 Christoph Olszowka. See MIT-LICENSE for details.