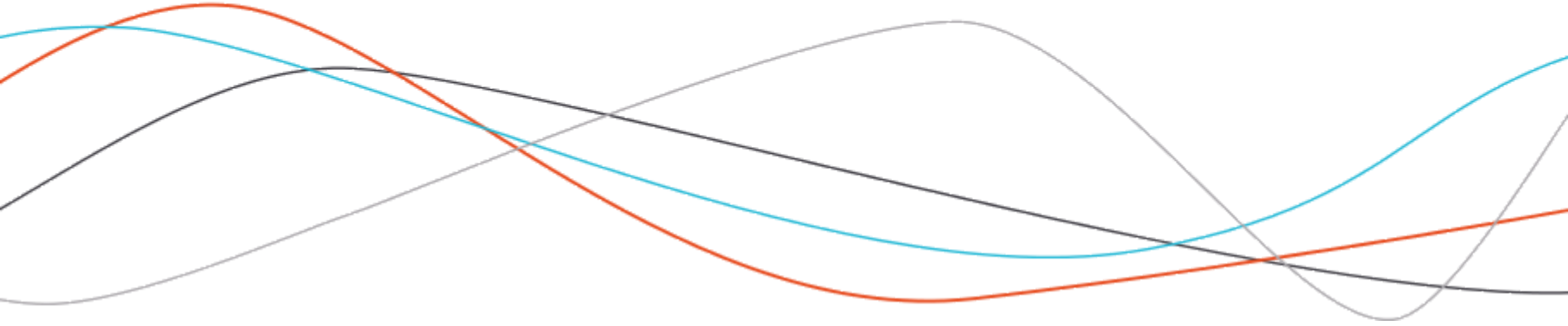
Building a package that lasts

Part 5: Test and code coverage

Everything which is not tested will, at last, break.



Why automate code testing?

- To save time!
- Work serenely with your coworkers
- Transfer the project
- Guarantee the stability on the long run

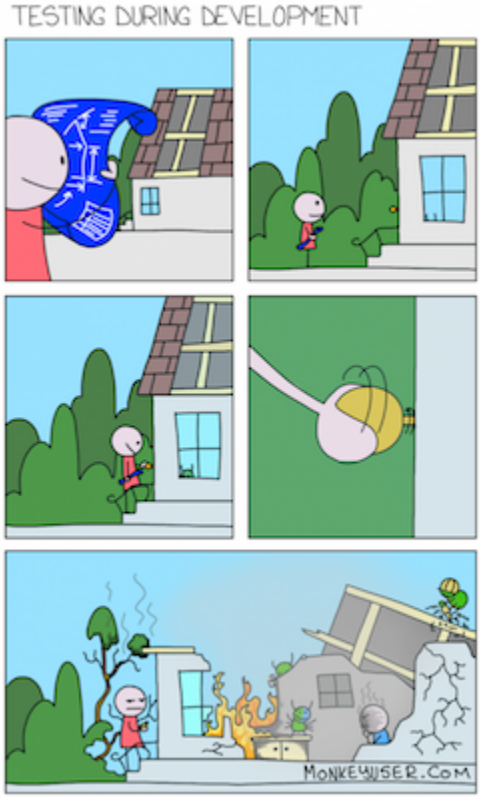
Thanks to `{testthat}`, we can automate the tests.

-> Allows bugs to be detected before they happen, and guarantees the validity of the code.

```
install.packages("testthat")
usethis::use_test("myfunction")
```

Creates a test/testthat folder, adds {testthat} to the `Suggests` of the DESCRIPTION, and creates test/testthat.R (do not touch it).

Write tests before it's too late



Use test in a "defensive programming" approach to prevent bugs.

Don't trust yourself in 6 months.

Be sure to send in production a package with minimum bugs.

Good news: **you're already writing test**, you just didn't know that before.

Does it rings a bell?

```
my_awesome_function <- function(a, b){
  res <- a + b
  return(res)
}
# Works
my_awesome_function(1, 2)
# Doesn't work
my_awesome_function("a","b")
```

Test that

In the test/ folder is a testthat.R file and a testthat folder. In this folder, you'll find .R files of the following form:

```
test-my_function.R
```

- One test file per (big) function, with general contexts.
- Break down the tests in this file by type.

These tests will be performed during `devtools::check()` (which also performs other tests), or with `devtools::test()` (Ctr/Cmd + Shift + T).

Test that

Each file is composed of a series of tests in this format:

```
context("global info")

test_that("details series 1",
          {
         test1a
         test1b
 })

test_that("details series 2",
          {
         test2a
         test2b
 })
```

Test functions

Your test functions start with `expect_*`. They take two elements: the first is the actual result, the second is the expected result. If the test is not passed, the function returns an error. If the test passes, the function returns nothing.

```
library(testthat)

expect_equal(10, 10)

a <- sample(1:10, 1)
b <- sample(1:10, 1)
expect_equal(a+b, 200)
```

```
Erreur : a + b not equal to 200.
1/1 mismatches
[1] 11 - 200 == -189
```

Expectations

```r
library(testthat)
grep("^expect", ls("package:testthat"), value = TRUE)
```

```
#>  [1] "expect"                    "expect_condition"
#>  [3] "expect_cpp_tests_pass"     "expect_equal"
#>  [5] "expect_equal_to_reference" "expect_equivalent"
#>  [7] "expect_error"              "expect_failure"
#>  [9] "expect_false"              "expect_gt"
#> [11] "expect_gte"                "expect_identical"
#> [13] "expect_is"                 "expect_known_failure"
#> [15] "expect_known_hash"         "expect_known_output"
#> [17] "expect_known_value"        "expect_length"
#> [19] "expect_less_than"          "expect_lt"
#> [21] "expect_lte"                "expect_match"
#> [23] "expect_message"            "expect_more_than"
#> [25] "expect_named"              "expect_null"
#> [27] "expect_output"             "expect_output_file"
#> [29] "expect_reference"          "expect_s3_class"
#> [31] "expect_s4_class"           "expect_setequal"
```

"Skip" a test

If you want to skip a test (if the code depends on a web connection, an API, etc...), use the `skip_if_not()` function.

```
library(httr)
url <- "http://numbersapi.com/42"
test_that("API test", {
  skip_if_not(curl::has_internet(), "No internet connection")
  res < content(GET(url))
  expect_is(url, "character")
})
```

`testthat:::skip_if_not_installed()` skips a test if a package is not installed.

There are other functions to skip a test under particular conditions, such as `skip_on_os()`, to prevent from testing on specific operating systems.

Create your own test

You can also create your own tests in `test_that()`:

```
plop <- function(class) {
  structure(1:10, class = class)
}

expect_plop <- function(object, class){
  expect_is(object, class)
}

test_that("Class well assigned", {
  a <- plop("ma_class")
  expect_plop(a, "ma_classe")
})
```

Launch tests

```r
grep("^test", ls("package:testthat"), value = TRUE)
```

```
#>  [1] "test_check"    "test_dir"      "test_env"      "test_example"
#>  [5] "test_examples" "test_file"     "test_package"  "test_path"
#>  [9] "test_rd"       "test_that"
```

Launch tests

```
devtools::check()

...
✔ |  12       | adverbs
✔ |   7       | test-utils.R
✔ |  22       | test-warn.R
✔ |  18       | test-any-all-none.R

══ Results ═══════════════════════════════════════════════════════
Duration: 0.8 s

OK:       192
Failed:   3
Warnings: 0
Skipped:  0
```

R CMD check

To test the code more globally, in the command line (i.e. in the terminal): R CMD check.

Or simply the `devtools::check()` function in your R session.

More tests are performed with `check` than with `devtools::test()`, which "only" performs the tests in the test folder.

This command runs around 50 different tests.

Is performed when you click the "Check" button on the Build tab of RStudio.

Test with rhub

{rhub} is a package that allows you to test for several OS:

```r
library(rhub)
ls("package:rhub")
```

```
#>  [1] "check"                 "check_for_cran"
#>  [3] "check_on_centos"       "check_on_debian"
#>  [5] "check_on_fedora"       "check_on_linux"
#>  [7] "check_on_macos"        "check_on_ubuntu"
#>  [9] "check_on_windows"      "check_with_rdevel"
#> [11] "check_with_roldrel"    "check_with_rpatched"
#> [13] "check_with_rrelease"   "check_with_sanitizers"
#> [15] "check_with_valgrind"   "last_check"
#> [17] "list_my_checks"        "list_package_checks"
#> [19] "list_validated_emails" "platforms"
#> [21] "rhub_check"            "rhub_check_for_cran"
#> [23] "rhub_check_list"       "validate_email"
```

Test with rhub

```r
devtools::install_github("r-hub/rhub")
# or
install.packages("rhub")
# verify your email
library(rhub)
validate_email()
```

```
— Choose email address to (re)validate (or 0 to exit)

1:    colin@thinkr.fr
2:    New email address

Selection: 1
Please check your emails for the r-hub access token.
Token:
```

Test with rhub

```
rhub::check()
```

Which platforms are supported?

```
rhub::platforms()
```

```
#> debian-gcc-devel:
#>   Debian Linux, R-devel, GCC
#> debian-gcc-patched:
#>   Debian Linux, R-patched, GCC
#> debian-gcc-release:
#>   Debian Linux, R-release, GCC
#> fedora-clang-devel:
#>   Fedora Linux, R-devel, clang, gfortran
#> fedora-gcc-devel:
#>   Fedora Linux, R-devel, GCC
#> linux-x86_64-centos6-epel:
#>   CentOS 6, stock R from EPEL
#> linux-x86_64-centos6-epel-rdt:
```

Test with rhub

```
> rhub::check()

— Choose build platform

 1: Debian Linux, R-devel, GCC (debian-gcc-devel)
 2: Debian Linux, R-patched, GCC (debian-gcc-patched)
 3: Debian Linux, R-release, GCC (debian-gcc-release)
 4: Fedora Linux, R-devel, clang, gfortran (fedora-clang-devel)
 5: Fedora Linux, R-devel, GCC (fedora-gcc-devel)
 6: CentOS 6, stock R from EPEL (linux-x86_64-centos6-epel)
 7: CentOS 6 with Redhat Developer Toolset, R from EPEL (linux-x86_64-centos6-epel-rdt)
 8: Debian Linux, R-devel, GCC ASAN/UBSAN (linux-x86_64-rocker-gcc-san)
 9: macOS 10.11 El Capitan, R-release (experimental) (macos-elcapitan-release)
10: macOS 10.9 Mavericks, R-oldrel (experimental) (macos-mavericks-oldrel)
11: Oracle Solaris 10, x86, 32 bit, R-patched (experimental) (solaris-x86-patched)
12: Ubuntu Linux 16.04 LTS, R-devel, GCC (ubuntu-gcc-devel)
13: Ubuntu Linux 16.04 LTS, R-release, GCC (ubuntu-gcc-release)
14: Ubuntu Linux 16.04 LTS, R-devel with rchk (ubuntu-rchk)
15: Windows Server 2008 R2 SP1, R-devel, 32/64 bit (windows-x86_64-devel)
16: Windows Server 2008 R2 SP1, R-oldrel, 32/64 bit (windows-x86_64-oldrel)
17: Windows Server 2008 R2 SP1, R-patched, 32/64 bit (windows-x86_64-patched)
18: Windows Server 2008 R2 SP1, R-release, 32/64 bit (windows-x86_64-release)

Selection: 19
Enter an item from the menu, or 0 to exit
Selection: 10

-  Building package
-  Uploading package
-  Preparing build, see status at
   http://builder.r-hub.io/status/attempt_0.1.1.tar.gz-94cfafe4db974ff4b99759ce65f47823
-  Build started

>
```

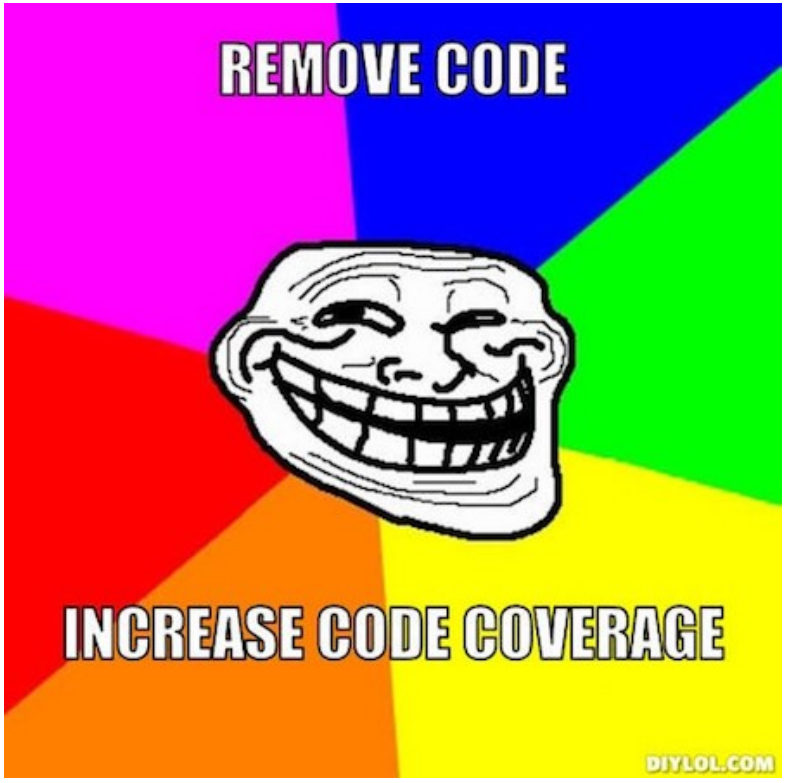
```
r-hub   Builder                                                    Email not verified
123  #> * checking for code/documentation mismatches ... OK
124  #> * checking Rd \usage sections ... OK
125  #> * checking Rd contents ... OK
126  #> * checking for unstated dependencies in examples ... OK
127  #> * checking installed files from 'inst/doc' ... OK
128  #> * checking files in 'vignettes' ... OK
129  #> * checking examples ... OK
130  #> * checking for unstated dependencies in 'tests' ... OK
131  #> * checking tests ...
132  #> Running 'testthat.R'
133  #> OK
134  #> * checking for unstated dependencies in vignettes ... OK
135  #> * checking package vignettes in 'inst/doc' ... OK
136  #> * checking running R code from vignettes ...
137  #> 'attempt.Rmd' using 'UTF-8' ... OK
138  #> NONE
139  #> * checking re-building of vignette outputs ... OK
140  #> * checking PDF version of manual ... OK
141  #> * DONE
142  #> Status: OK
143  #> Saving artifacts
144  #> Cleaning up user and home directory
145  #> SSH: Connecting from host [Rhubs-Mac-2.local]
146  #> SSH: Connecting with configuration [files] ...
147  #> SSH: Disconnecting configuration [files] ...
148  #> SSH: Transferred 0 file(s)
149  #> Build step 'Send files or execute commands over SSH' changed build result to SUCCESS
150  #> Pinging https://builder.r-hub.io/build/SUCCESS/attempt_0.1.1.tar.gz-94cfafe4db974ff4b99759ce65f47823/2018-01-16T07:31:53Z
151  #> {"status":"ok"}
152  #> Finished: SUCCESS

Terms of use · R consortium
```

Code Coverage

What is code coverage?

Code coverage is the proportion of code that is launched when you run your package tests.



Local code coverage

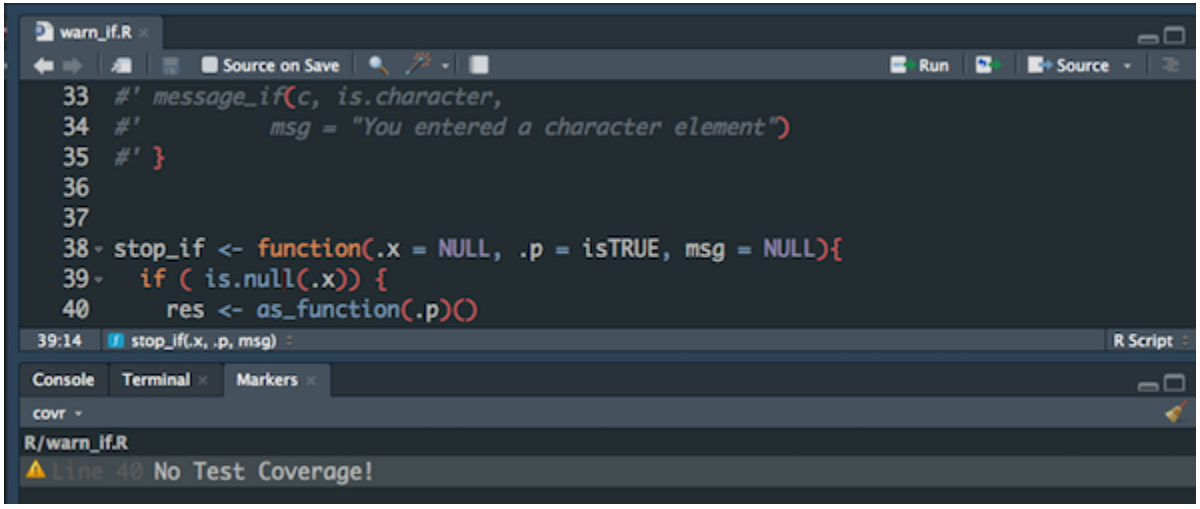
```
covr::package_coverage()
```

```
> covr::package_coverage()
attempt Coverage: 99.40%
R/warn_if.R: 98.33%
R/if.R: 100.00%
R/try_catch.R: 100.00%
R/utils.R: 100.00%
>
```

Local code coverage

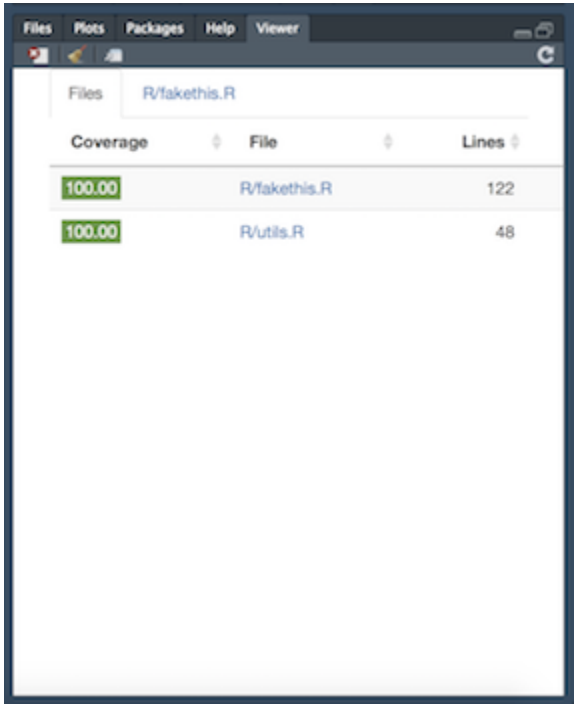
Which part of my code are not covered by tests?

```
my_coverage <- covr::package_coverage()
covr::zero_coverage()
```



Local code coverage

> Using the addin



codecov.io

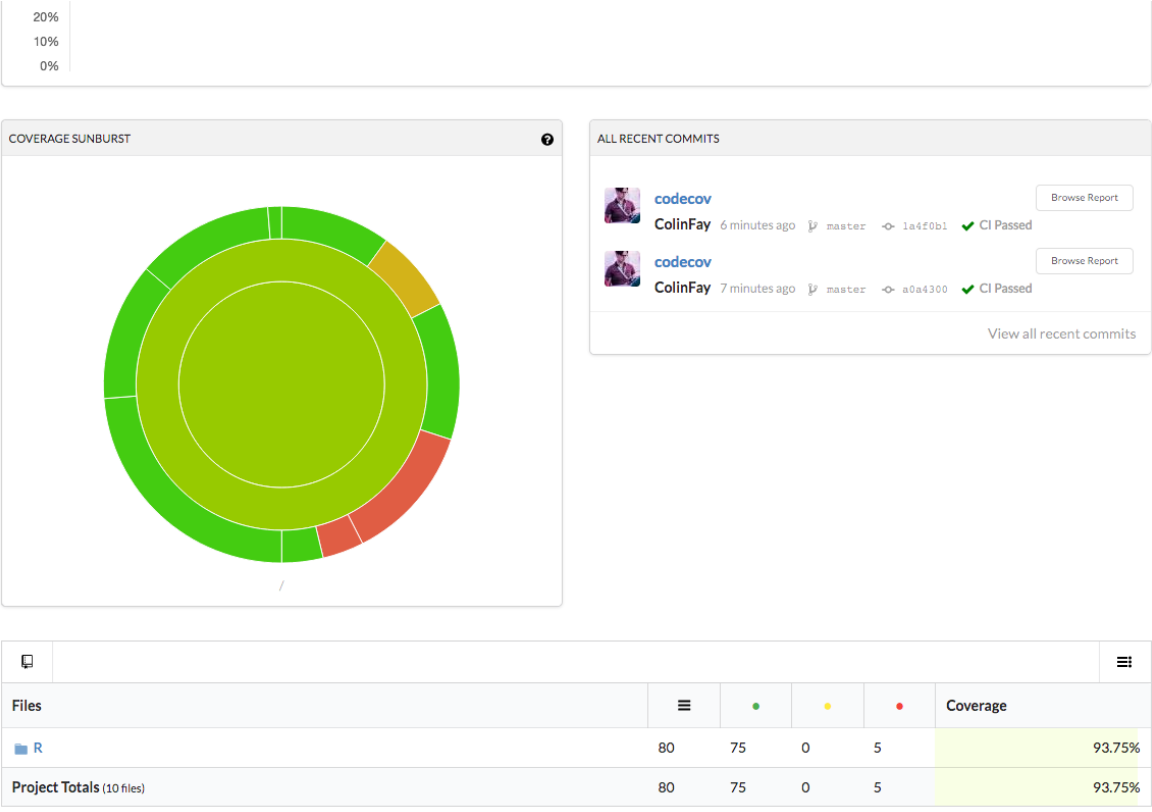
```r
usethis::use_coverage()
```

```
> covr::package_coverage()
attempt Coverage: 99.40%
R/warn_if.R: 98.33%
R/if.R: 100.00%
R/try_catch.R: 100.00%
R/utils.R: 100.00%
>
```

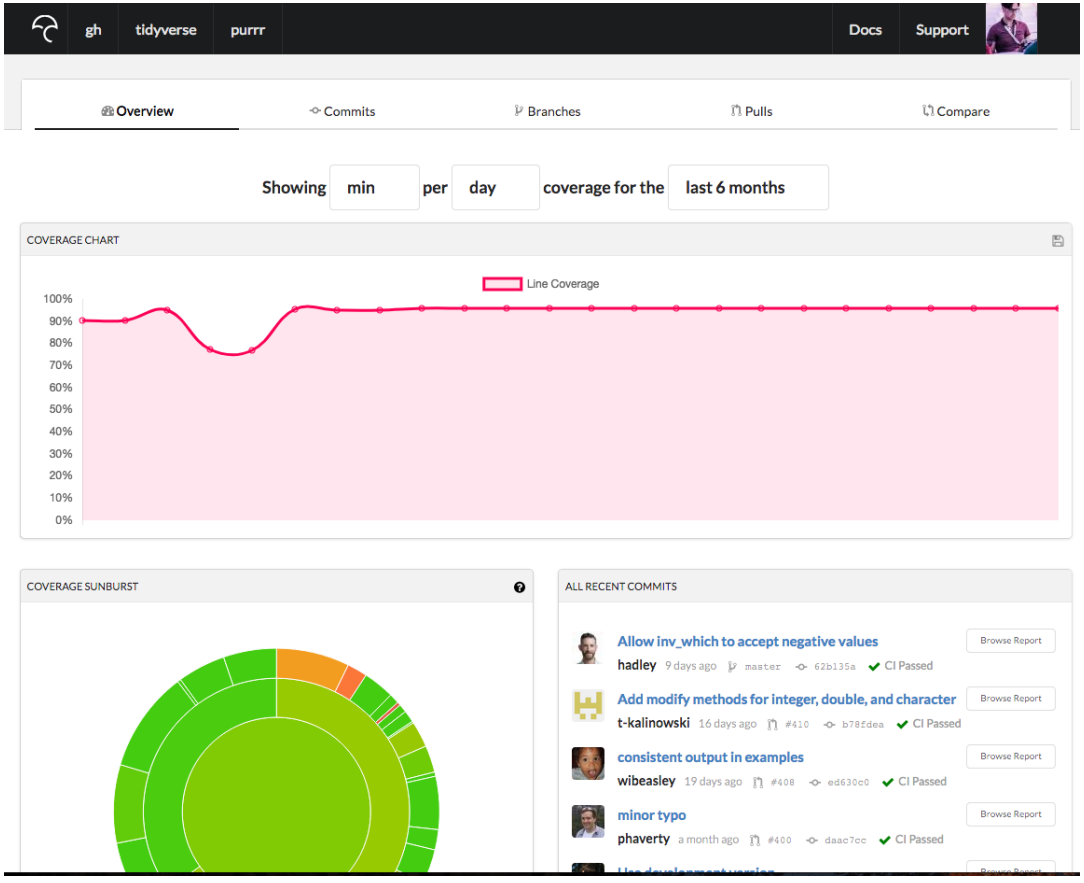
Code cov is a service that is used with Travis (we'll see Travis in the next chapterr), and allows to know the amount of code covered by the tests online.

The `{usethis}` function creates the appropriate yaml, and inserts in your clipboard some code to paste in your travis yaml.

codecov.io


20%
10%
0%
COVERAGE SUNBURST
ALL RECENT COMMITS
codecov
ColinFay 6 minutes ago master 1a4f0b1 CI Passed
Browse Report
codecov
ColinFay 7 minutes ago master a0a4300 CI Passed
Browse Report
View all recent commits
Files
Coverage
R 80 75 0 5 93.75%
Project Totals (10 files) 80 75 0 5 93.75%

codecov.io


gh
tidyverse
purrr
Docs
Support
Overview
Commits
Branches
Pulls
Compare
Showing
min
per
day
coverage for the
last 6 months
COVERAGE CHART
Line Coverage
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
COVERAGE SUNBURST
ALL RECENT COMMITS
Allow inv_which to accept negative values
hadley 9 days ago master 62b135a CI Passed
Browse Report
Add modify methods for integer, double, and character
t-kalinowski 16 days ago #410 b78fdea CI Passed
Browse Report
consistent output in examples
wibeasley 19 days ago #408 ed630c0 CI Passed
Browse Report
minor typo
phaverty a month ago #400 daac7cc CI Passed
Browse Report

Let's practice !