Package 'covr'

December 5, 2016

Title Test Coverage for Packages

Version 2.2.1

Description Track and report code coverage for your package and (optionally) upload the results to a coverage service like 'Codecov' (http://codecov.io) or 'Coveralls' (http://coveralls.io). Code coverage is a measure of the amount of code being exercised by a set of tests. It is an indirect measure of test quality and completeness. This package is compatible with any testing methodology or framework and tracks coverage of both R code and compiled C/C++/FORTRAN code.

URL https://github.com/jimhester/covr

BugReports https://github.com/jimhester/covr/issues

Depends R (>= 3.1.0), methods

Imports stats, utils, jsonlite, rex, httr, crayon, withr (>= 1.0.2), memoise

Suggests R6, knitr, rmarkdown, shiny (>= 0.11.1), htmltools, htmlwidgets (>= 0.7), DT (>= 0.2), testthat, rstudioapi (>= 0.2), devtools, xml2 (>= 1.0.0), parallel

License MIT + file LICENSE

LazyData true

VignetteBuilder knitr

RoxygenNote 5.0.1

NeedsCompilation yes

Author Jim Hester [aut, cre], Willem Ligtenberg [ctb]

Maintainer Jim Hester <james.f.hester@gmail.com>

Repository CRAN

Date/Publication 2016-12-05 18:28:47

R topics documented:

`codecov` *Run covr on a package and upload the result to codecov.io*

Description

Run covr on a package and upload the result to codecov.io

Usage

```
codecov(..., coverage = NULL, base_url = "https://codecov.io",
  token = NULL, commit = NULL, branch = NULL, quiet = TRUE)
```

Arguments

`...`	arguments passed to `package_coverage`
`coverage`	an existing coverage object to submit, if `NULL`, `package_coverage` will be called with the arguments from `...`
`base_url`	Codecov url (change for Enterprise)
`token`	a codecov upload token, if `NULL` the environment variable 'CODECOV_TOKEN' is used.
`commit`	explicitly set the commit this coverage result object corresponds to. Is looked up from the service or locally if it is `NULL`.
`branch`	explicitly set the branch this coverage result object corresponds to, this is looked up from the service or locally if it is `NULL`.
`quiet`	if `FALSE`, print the coverage before submission.

Examples

```
## Not run:
codecov(path = "test")

## End(Not run)
```

`coveralls` *Run covr on a package and upload the result to coveralls*

Description

Run covr on a package and upload the result to coveralls

Usage

```
coveralls(..., coverage = NULL, repo_token = Sys.getenv("COVERALLS_TOKEN"),
  service_name = Sys.getenv("CI_NAME", "travis-ci"), quiet = TRUE)
```

Arguments

`...`	arguments passed to `package_coverage`
`coverage`	an existing coverage object to submit, if `NULL`, `package_coverage` will be called with the arguments from `...`
`repo_token`	The secret repo token for your repository, found at the bottom of your repository's page on Coveralls. This is useful if your job is running on a service Coveralls doesn't support out-of-the-box. If set to NULL, it is assumed that the job is running on travis-ci
`service_name`	the CI service to use, if environment variable 'CI_NAME' is set that is used, otherwise 'travis-ci' is used.
`quiet`	if `FALSE`, print the coverage before submission.

`exclusions` *Exclusions*

Description

covr supports a couple of different ways of excluding some or all of a file.

Line Exclusions

The `line_exclusions` argument to `package_coverage()` can be used to exclude some or all of a file. This argument takes a list of filenames or named ranges to exclude.

Function Exclusions

Alternatively `function_exclusions` can be used to exclude R functions based on regular expression(s). For example `print\.*` can be used to exclude all the print methods defined in a package from coverage.

Exclusion Comments

In addition you can exclude lines from the coverage by putting special comments in your source code. This can be done per line or by specifying a range. The patterns used can be specified by the `exclude_pattern`, `exclude_start`, `exclude_end` arguments to `package_coverage()` or by setting the global options `covr.exclude_pattern`, `covr.exclude_start`, `covr.exclude_end`.

Examples

```
## Not run:
# exclude whole file of R/test.R
package_coverage(exclusions = "R/test.R")

# exclude lines 1 to 10 and 15 from R/test.R
package_coverage(line_exclusions = list("R/test.R" = c(1:10, 15)))

# exclude lines 1 to 10 from R/test.R, all of R/test2.R
package_coverage(line_exclusions = list("R/test.R" = 1:10, "R/test2.R"))

# exclude all print and format methods from the package.
package_coverage(function_exclusions = c("print\\.", "format\\."))

# single line exclusions
f1 <- function(x) {
  x + 1 # nocov
}

# ranged exclusions
f2 <- function(x) { # nocov start
  x + 2
} # nocov end

## End(Not run)
```

`file_coverage` *Calculate test coverage for sets of files*

Description

The files in `source_files` are first sourced into a new environment to define functions to be checked. Then they are instrumented to track coverage and the files in `test_files` are sourced.

Usage

```
file_coverage(source_files, test_files, line_exclusions = NULL,
  function_exclusions = NULL, parent_env = parent.frame())
```

Arguments

source_files	Character vector of source files with function definitions to measure coverage
test_files	Character vector of test files with code to test the functions
line_exclusions	a named list of files with the lines to exclude from each file.
function_exclusions	a vector of regular expressions matching function names to exclude. Example `print\.` to match print methods.
parent_env	The parent environment to use when sourcing the files.

---

function_coverage *Calculate test coverage for a specific function.*

Description

Calculate test coverage for a specific function.

Usage

```
function_coverage(fun, code = NULL, env = NULL, enc = parent.frame())
```

Arguments

fun	name of the function.
code	expressions to run.
env	environment the function is defined in.
enc	the enclosing environment which to run the expressions.

`package_coverage` *Calculate test coverage for a package*

Description

This function calculates the test coverage for a development package on the `path`. By default it runs only the package tests, but it can also run vignette and example code.

Usage

```
package_coverage(path = ".", type = c("tests", "vignettes", "examples",
  "all", "none"), combine_types = TRUE, relative_path = TRUE,
  quiet = TRUE, clean = TRUE, line_exclusions = NULL,
  function_exclusions = NULL, code = character(), ..., exclusions)
```

Arguments

`path`	file path to the package
`type`	run the package 'tests', 'vignettes', 'examples', 'all', or 'none'. The default is 'tests'.
`combine_types`	If `TRUE` (the default) the coverage for all types is simply summed into one coverage object. If `FALSE` separate objects are used for each type of coverage.
`relative_path`	whether to output the paths as relative or absolute paths.
`quiet`	whether to load and compile the package quietly, useful for debugging errors.
`clean`	whether to clean temporary output files after running, mainly useful for debugging errors.
`line_exclusions`	a named list of files with the lines to exclude from each file.
`function_exclusions`	a vector of regular expressions matching function names to exclude. Example `print\.` to match print methods.
`code`	A character vector of additional test code to run.
`...`	Additional arguments passed to `testInstalledPackage`
`exclusions`	'Deprecated', please use 'line_exclusions' instead.

Details

This function uses `testInstalledPackage` to run the code, if you would like to test your package in another way you can set `type = "none"` and pass the code to run as a character vector to the `code` parameter.

Parallelized code using `mcparallel` needs to be use a patched `mcparallel:::mcexit`. This is done automatically if the package depends on parallel, but can also be explicitly set using the environment variable `COVR_FIX_PARALLEL_MCEXIT` or the global option `covr.fix_parallel_mcexit`.

See Also

exclusions For details on excluding parts of the package from the coverage calculations.

---

percent_coverage *Provide percent coverage of package*

Description

Calculate the total percent coverage from a coverage result object.

Usage

```
percent_coverage(x, ...)
```

Arguments

x	the coverage object returned from package_coverage
...	additional arguments passed to tally_coverage

Value

The total percentage as a numeric(1).

---

print.coverage *Print a coverage object*

Description

Print a coverage object

Usage

```
## S3 method for class 'coverage'
print(x, group = c("filename", "functions"), by = "line",
  ...)
```

Arguments

x	the coverage object to be printed
group	whether to group coverage by filename or function
by	whether to count coverage by line or expression
...	additional arguments ignored

Value

The coverage object (invisibly).

report *Display covr results using a standalone report*

Description

Display covr results using a standalone report

Usage

```
report(x, ...)

shine(x, ...)

## S3 method for class 'coverage'
report(x, file = file.path(tempdir(),
  paste0(get_package_name(x), "-report.html")), browse = interactive(), ...)
```

Arguments

x	a coverage dataset
...	Additional arguments passed to methods
file	The report filename.
browse	whether to open a browser to view the report.

Examples

```
## Not run:
x <- package_coverage()
report(x)

## End(Not run)
```

tally_coverage *Tally coverage by line or expression*

Description

Tally coverage by line or expression

Usage

```
tally_coverage(x, by = c("line", "expression"))
```

Arguments

x	the coverage object returned from package_coverage
by	whether to tally coverage by line or expression

Value

a data.frame of coverage tallied by line or expression.

---

to_cobertura *Create a Cobertura XML file*

Description

This functionality requires the xml2 package be installed.

Usage

```
to_cobertura(cov, filename = "cobertura.xml")
```

Arguments

cov	the coverage object returned from package_coverage
filename	the name of the Cobertura XML file

Author(s)

Willem Ligtenberg

---

value *Retrieve the value from an object*

Description

Retrieve the value from an object

Usage

```
value(x, ...)
```

Arguments

x	object from which to retrieve the value
...	additional arguments passed to methods

`zero_coverage` *Provide locations of zero coverage*

Description

When examining the test coverage of a package, it is useful to know if there are any locations where there is **0** test coverage.

Usage

```
zero_coverage(x, ...)
```

Arguments

`x`	a coverage object returned `package_coverage`
`...`	additional arguments passed to `tally_coverage`

Details

if used within RStudio this function outputs the results using the Marker API.

Value

A `data.frame` with coverage data where the coverage is 0.

Index