Package ‘test.assessr’

March 2, 2026

Title Assessing Package Test Reliability and Quality

Version 1.1.1

Description A reliable and validated tool that calculates unit test coverage for R packages with standard testing frameworks and non-standard testing frameworks.

License GPL (>= 2)

BugReports `https://github.com/Sanofi-Public/test.assessr/issues`

Depends R (>= 4.1.0)

Imports callr, checkmate, covr, dplyr, fs, jsonlite, pkgload, remotes, rlang, rmarkdown, RUnit, stringr, testthat (>= 3.0.0), tidyr, utils, withr

Suggests devtools, DT, here, kableExtra, knitr, methods, R6, S7, roxygen2, tidyselect, tools, mockery

Encoding UTF-8

RoxygenNote 7.3.3

Config/testthat/edition 3

Config/build/clean-inst-doc false

VignetteBuilder knitr

NeedsCompilation no

Author Edward Gillian [cre, aut] (ORCID: <`https://orcid.org/0000-0003-2732-5107`>),
Hugo Bottois [aut] (ORCID: <`https://orcid.org/0000-0003-4674-0875`>),
Paulin Charliquart [aut],
Andre Couturier [aut],
Sanofi [cph, fnd]

Maintainer Edward Gillian <`edward.gillian-ext@sanofi.com`>

Repository CRAN

Date/Publication 2026-03-02 22:20:02 UTC

Contents

`check_pkg_tests_and_snaps`

Check for Testing Infrastructure and Snapshot Files

Description

This function inspects an R package source tree and detects the presence of common testing frameworks (‘testthat‘, ‘testit‘, base R tests, BioGenerics/Bioconductor-style tests) as well as snapshot files used for golden testing.

Usage

```
check_pkg_tests_and_snaps(pkg_source_path)
```

Arguments

`pkg_source_path`	Character. Path to the root of the package source.

Value

A list containing logical indicators and file counts describing the test configuration of the package. The list includes:

has_testthat Logical. Whether ‘tests/testthat/‘ exists.

has_testit Logical. Whether ‘tests/testit/‘ exists.

has_tests_base Logical. Whether base R test files exist in ‘tests/‘.

has_BioG_test Logical. Whether BioGenerics-style tests exist in ‘inst/tests/‘.

bioc_unit_tests_dir Character. Path to BioGenerics unit test directory (if any).

bioc_run_ut_path Character. Path to BioGenerics ‘run_unitTests.R‘ (if any).

has_snaps Logical. Whether ‘_snaps/‘ exists inside ‘tests/testthat/‘.

n_golden_tests Integer. Number of snapshot test files inside ‘_snaps/‘.

n_test_files Integer. Number of ‘test-*.R‘ files in ‘tests/testthat/‘.

This function always returns a value. It does **not** perform side effects other than reading the package directory structure.

Examples

```
# Adjust CRAN repo (example only)
r <- getOption("repos")
r["CRAN"] <- "http://cran.us.r-project.org"
old <- options(repos = r)

# Example package contained in test.assessr
dp <- system.file(
  "test-data",
  "test.package.0001_0.1.0.tar.gz",
  package = "test.assessr"
)

# Set up package
install_list <- set_up_pkg(dp)
pkg_source_path <- install_list$pkg_source_path

# Install package locally (ensures correct test paths)
install_package_local(pkg_source_path)

# Detect tests and snapshots
test_pkg_data <- check_pkg_tests_and_snaps(pkg_source_path)

# Restore options
options(old)
```

`extract_short_path` *Extract the Last Two Path Components from a File Path*

Description

This helper function takes any full file path and extracts only the last two components, such as ‘"R/add.R"‘. It supports both forward slashes (‘/‘) and backslashes (‘\‘) to accommodate Windows, macOS, and Linux paths.

Usage

```
extract_short_path(long_file_name)
```

Arguments

`long_file_name` Character string. A full file path using ‘/‘ or ‘\‘ as separators.

Details

Trailing separators are preserved where meaningful (e.g., ‘"R/"‘ when the input ends with a slash). Empty path components are removed except when required to detect a trailing separator.

Value

A character scalar containing the last two components of the path joined with a forward slash—for example ‘"R/add.R"‘—or a single component if only one exists. The returned value is always of class `character`.

Output Meaning

This function is intended for display and logging purposes, where only the tail portion of a full file path is meaningful. It does not check for file existence; it simply processes the string supplied by the user.

Examples

```
extract_short_path("pkg/R/add.R")
extract_short_path("C:\\\\projects\\\\mypkg\\\\R\\\\helper.R")
```

`generate_test_report` *Generate HTML Report for Package Test Assessment*

Description

Generates an HTML report for the package test assessment results using rmarkdown.

Usage

```
generate_test_report(test_results, output_dir = NULL)
```

Arguments

`test_results`	List containing the results from get_package_coverage function.
`output_dir`	(required) Character string path to an existing directory where the report will be saved.

Value

Path to the generated HTML report.

Examples

```
## Not run:
test_results <- get_package_coverage()
# Always provide a directory; tempdir() is convenient in examples.
generate_test_report(test_results, output_dir = tempdir())

## End(Not run)
```

`get_package_coverage` *get package test coverage*

Description

simplified input to assess package for test coverage

Usage

```
get_package_coverage(path = NULL)
```

Arguments

`path`	(optional) path of locally stored package source code

Value

An object of class `"coverage"` as produced by `test.assessr::run_covr_modes()`. This object is a structured list containing detailed test coverage information, including:

- **File-level coverage:** percentage of lines covered in each file.
- **Line-by-line coverage:** number of executions for each line.
- **Overall coverage:** aggregated coverage across the package.
- **Metadata:** source file paths, expressions, and summary stats.

The resulting object can be printed, summarized, or passed to `test.assessr::generate_test_report()` to produce a human-readable test coverage report.

Returns `NULL` if the package cannot be installed or if the specified path does not exist.

Examples

```
# Example STF package included in test.assessr
pkg_source_path <- system.file(
  "test-data",
  "test.package.0001_0.1.0.tar.gz",
  package = "test.assessr"
)

# Run get_package_coverage
get_package_coverage <- get_package_coverage(pkg_source_path)
```

get_pkg_name *Get Package Name for Display*

Description

Extracts a display-friendly package name from either a file path or a filename. The function removes directory components (if present) and then returns the substring up to the first underscore or hyphen. This is useful for converting paths or tarball names into a clean package identifier.

Usage

```
get_pkg_name(input_string)
```

Arguments

`input_string`	Character string. A package filename or a path containing the filename (e.g., `"mypkg_1.0.0.tar.gz"` or `"/path/to/mypkg_1.0.0.tar.gz"`).

Value

A character scalar containing the cleaned package name.

The returned object is always of class `character` and corresponds to the portion of the filename before the first underscore or hyphen.

Output Meaning

The value represents a human-readable package name extracted from a file path or filename. It does not validate whether the extracted name corresponds to an installed or existing package—only that it conforms to the expected tarball naming convention.

Examples

```
pkg_source_path <- "/home/user/R/test.package.0001_0.1.0.tar.gz"
pkg_disp_1 <- get_pkg_name(pkg_source_path)
print(pkg_disp_1)

pkg <- "TxDb.Dmelanogaster.UCSC.dm3.ensGene_3.2.2.tar.gz"
pkg_disp_2 <- get_pkg_name(pkg)
print(pkg_disp_2)
```

`install_package_local` *Install a Package from a Local Source Directory*

Description

Attempts to install an R package from a local source directory using `remotes::install_local()`. The function reports on whether installation succeeded, whether the package was already installed, or whether the provided source path does not exist.

Usage

```
install_package_local(pkg_source_path)
```

Arguments

`pkg_source_path`	Character string. Path to the local package source directory (e.g., an unpacked package or extracted tarball path).

Details

The display name of the package is derived from the input path using `get_pkg_name()`.

Value

A logical value indicating whether the package is installed after running the function.

The returned object is always of class `logical`:

- `TRUE` — The package is already installed or was successfully installed.
- `FALSE` — Installation failed or the path does not exist.

Output Meaning

`TRUE` does not necessarily imply that the installation occurred during this function call—it may also mean the package was already installed.

`FALSE` indicates a failure to install or an invalid path. All diagnostic messages are printed via `message()` for user visibility.

Examples

```
## Not run:
results <- install_package_local("pkg_source_path")
print(results)

## End(Not run)
```

`run_coverage` *Run Coverage and Return Structured Coverage Results*

Description

This function executes code coverage analysis for a package using the ‘covr‘ framework. It is typically used after a package has been installed locally and test files are available. The function runs coverage in an isolated process (using ‘callr‘) and returns a structured summary of overall and file-level coverage.

Usage

```
run_coverage(pkg_source_path, timeout = Inf)
```

Arguments

`pkg_source_path`	Character. Path to the installed package directory from which coverage should be computed.
`timeout`	Numeric. Timeout (in seconds) passed to `callr::r_safe()` when running coverage. This limits the maximum time allowed for executing the tests underlying the coverage analysis.

Details

The function invokes covr’s coverage evaluation in a clean R session and extracts both:

- total coverage: percentage of lines covered across the package, and
- function/file-level coverage: coverage data for individual files.

It is used internally by higher-level functions such as `run_covr_modes()` and skip-aware coverage wrappers in the Standard Testing Framework (STF).

Value

A named list containing:

total_cov Numeric. Aggregated coverage percentage for the package.

res_cov A list containing file-level or function-level coverage results as returned by ‘covr‘. This includes per-file coverage, errors, and diagnostic notes if present.

Returns `NULL` if coverage could not be computed.

Examples

```
# Save and adjust CRAN mirror for reproducibility
r <- getOption("repos")
old <- options(repos = r)
r["CRAN"] <- "http://cran.us.r-project.org"
options(repos = r)

# Example package from test.assessr
dp <- system.file(
  "test-data",
  "test.package.0001_0.1.0.tar.gz",
  package = "test.assessr"
)

# Set up package source directory
install_list <- set_up_pkg(dp)
pkg_source_path <- install_list$pkg_source_path

# Install locally to enable testing and coverage
package_installed <- install_package_local(pkg_source_path)

if (isTRUE(package_installed)) {
  coverage_results <- run_coverage(pkg_source_path)
}

# Restore user's original repository settings
options(old)
```

`run_covr_modes` *Run Coverage Analysis with Test Detection*

Description

This function inspects the test configuration of an R package and runs code coverage analysis using any available testing framework, including ‘testthat‘, ‘testit‘, base R test scripts, or Bioconductor-style tests. If no recognised testing configuration is found, a default zero-coverage result is returned.

Usage

```
run_covr_modes(pkg_source_path, covr_timeout = 60)
```

Arguments

`pkg_source_path`	Character. Path to the root directory of the package source.
`covr_timeout`	Numeric. Timeout in seconds for running coverage analysis. Default is 60.

Value

A named list containing coverage results, package metadata, and test configuration details. The returned list includes (but is not limited to):

pkg_name Character. Package name extracted from the DESCRIPTION file.

pkg_ver Character. Package version.

date_time Timestamp of when the analysis was run.

executor User or environment running the analysis.

sysname, version, release, machine System metadata.

r_version R version used during analysis.

test_framework_type Character. Detected testing framework type.

covr_list A nested list containing:

- **total_cov** Numeric. Aggregated coverage percentage.
- **res_cov** File-level and line-level coverage details.
- **errors, notes** Any warnings or notes detected during testing.

test_pkg_data A list describing the test configuration of the package (presence of testthat, testit, base tests, snapshots, etc.).

This function always returns a value. When no supported testing framework is detected, a default object with zero coverage and diagnostic information is returned.

Examples

```
dp <- system.file("test-data",
  "test.package.0001_0.1.0.tar.gz",
  package = "test.assessr")

# set up package
install_list <- set_up_pkg(dp)

package_installed <- install_list$package_installed
pkg_source_path <- install_list$pkg_source_path

# install package locally to ensure test works
package_installed <- install_package_local(pkg_source_path)
package_installed <- TRUE

covr_mode_list <- run_covr_modes(pkg_source_path)
```

`set_up_pkg` *Create Information on Local Package Installation*

Description

This function unpacks a package tarball using `unpack_tarball()` and returns information about whether the unpacked directory exists. It is a lightweight preparatory step used before attempting a local package installation.

Usage

```
set_up_pkg(dp)
```

Arguments

`dp`	Character string. Path to a package tarball or package directory.

Value

A named list with the following elements:

- `package_installed` — Logical. `TRUE` if the unpacked package directory exists, otherwise `FALSE`.
- `pkg_source_path` — Character string giving the unpacked package source directory, or `""` if unpacking failed.

The returned object is always a base R `list`. It contains no side effects besides calling `unpack_tarball()` and checking filesystem paths.

Output Meaning

A value of `package_installed` = `TRUE` indicates that the unpacked directory exists on disk and can be used for local installation.

`package_installed` = `FALSE` indicates either:

- `unpack_tarball()` returned an empty result, or
- the unpacked directory does not exist on disk.

Examples

```
## Not run:
set_up_pkg(path/to/package, "mypackage")

## End(Not run)
```

Index