

Package ‘testit’

December 5, 2018

Type Package

Title A Simple Package for Testing R Packages

Version 0.9

Description Provides two convenience functions `assert()` and `test_pkg()` to facilitate testing R packages.

License GPL

URL <https://github.com/yihui/testit>

BugReports <https://github.com/yihui/testit/issues>

Suggests rstudioapi

RoxygenNote 6.1.1

Encoding UTF-8

NeedsCompilation no

Author Yihui Xie [aut, cre] (<<https://orcid.org/0000-0003-0645-5666>>),
Steven Mortimer [ctb]

Maintainer Yihui Xie <xie@yihui.name>

Repository CRAN

Date/Publication 2018-12-05 18:10:03 UTC

R topics documented:

| | |
|------------------------------------|---|
| <code>assert</code> | 2 |
| <code>has_warning</code> | 4 |
| <code>test_pkg</code> | 4 |

| | |
|--------------|----------|
| Index | 6 |
|--------------|----------|

`assert`*Assertions with an optional message*

Description

The function `assert()` was inspired by `stopifnot()`. It emits a message in case of errors, which can be a helpful hint for diagnosing the errors (`stopifnot()` only prints the possibly truncated source code of the expressions).

The infix operator `%==%` is simply an alias of the `identical()` function to make it slightly easier and intuitive to write test conditions. `x %==% y` is the same as `identical(x, y)`. When it is used inside `assert()`, a message will be printed if the returned value is not `TRUE`, to show the values of the LHS (`x`) and RHS (`y`) via `str()`, which can be helpful for you to check why the assertion failed.

Usage

```
assert(fact, ...)
```

```
x %==% y
```

Arguments

| | |
|-------------------|---|
| <code>fact</code> | a message for the assertions when any of them fails; treated the same way as expressions in <code>...</code> if it is not a character string, which means you do not have to provide a message to this function |
| <code>...</code> | any number of R expressions; see Details |
| <code>x, y</code> | two R objects to be compared |

Details

There are two ways to write R expressions in the `...` argument.

The first way is a single R expression wrapped in `{}` and passed as a single argument. This expression may contain multiple sub-expressions. A sub-expression is treated as a test condition if it is wrapped in `()` (meaning its value will be checked to see if it is a logical vector containing any `FALSE` values), otherwise it is evaluated in the normal way and its value will not be checked. If the value of the last sub-expression is logical, it will also be treated as a test condition.

The second way is a series of R expressions (each expression is passed as an individual argument) that return vectors of `TRUE`'s (if `FALSE` is returned anywhere, an error will show up).

Value

For `assert()`, invisible `NULL` if all expressions returned `TRUE`, otherwise an error is signalled and the user-provided message is emitted. For `%==%`, `TRUE` or `FALSE`.

Note

The internal implementation of `assert()` is different with the `stopifnot()` function in R **base**: (1) the custom message fact is emitted if an error occurs; (2) `assert()` requires the logical values to be non-empty (`logical(0)` will trigger an error); (3) if `...` contains a compound expression in `{}` that returns `FALSE` (e.g., `if (TRUE) {1+1; FALSE}`), the first and the last but one line of the source code from `deparse()` are printed in the error message, otherwise the first line is printed; (4) the arguments in `...` are evaluated sequentially, and `assert()` will signal an error upon the first failed assertion, and will ignore the rest of assertions.

Examples

```
## The first way to write assertions -----

assert("T is bad for TRUE, and so is F for FALSE", {
  T = FALSE
  F = TRUE
  (T != TRUE) # note the parentheses
  (F != FALSE)
})

assert("A Poisson random number is non-negative", {
  x = rpois(1, 10)
  (x >= 0)
  (x > -1) # () is optional because it's the last expression
})

## The second way to write assertions -----

assert("one equals one", 1 == 1)
assert("seq and : produce equal sequences", seq(1L, 10L) == 1L:10L)
assert("seq and : produce identical sequences", identical(seq(1L, 10L), 1L:10L))

# multiple tests
T = FALSE
F = TRUE
assert("T is bad for TRUE, and so is F for FALSE", T != TRUE, F != FALSE)

# a mixture of tests
assert("Let's pray all of them will pass", 1 == 1, 1 != 2, letters[4] == "d",
      rev(rev(letters)) == letters)

# logical(0) cannot pass assert(), although stopifnot() does not care
try(assert("logical(0) cannot pass", 1 == integer(0)))
stopifnot(1 == integer(0)) # it's OK!

# a compound expression
try(assert("this if statement returns TRUE", if (TRUE) {
  x = 1
  x == 2
}))
```

```
# no message
assert(!FALSE, TRUE, is.na(NA))
```

| | |
|-------------|---|
| has_warning | <i>Check if an R expression produces warnings or errors</i> |
|-------------|---|

Description

The two functions `has_warning()` and `has_error()` check if an expression produces warnings and errors, respectively.

Usage

```
has_warning(expr)

has_error(expr, silent = !interactive())
```

Arguments

| | |
|---------------------|---|
| <code>expr</code> | an R expression |
| <code>silent</code> | logical: should the report of error messages be suppressed? |

Value

A logical value.

Examples

```
has_warning(1 + 1)
has_warning(1:2 + 1:3)

has_error(2 - 3)
has_error(1 + "a")
has_error(stop("err"), silent = TRUE)
```

| | |
|----------|--|
| test_pkg | <i>Run the tests of a package in its namespace</i> |
|----------|--|

Description

The main purpose of this function is to expose the namespace of a package when running tests, which allows one to use non-exported objects in the package without having to resort to the triple colon `:::` trick.

Usage

```
test_pkg(package, dir = "testit")
```

Arguments

| | |
|---------|--|
| package | the package name |
| dir | the directory of the test files; by default, it is the directory 'testit/' under the current working directory |

Details

The tests are assumed to be under the 'testit/' directory by default, and this function also looks for the 'tests/testit/' directory under the package installation directory when the user-provided dir does not exist. The test scripts must be named of the form 'test-*.R'; other R scripts will not be treated as test files (but may also be useful, e.g. you can [source\(\)](#) them in tests).

For R CMD check, this means the test R scripts ('test-*.R' are under 'pkg_root/tests/testit/'. The R scripts are executed with [sys.source](#) in the namespace of the package to be tested; when an R script is executed, the working directory is the same as the directory containing this script, and all existing objects in the test environment will be removed before the code is executed.

Value

NULL. All test files are executed, unless an error occurs.

Note

All test scripts ('test-*.R') must be encoded in UTF-8 if they contain any multibyte characters.

See Also

The [testthat](#) package (much more sophisticated).

Examples

```
## Not run:  
test_pkg("testit")  
  
## End(Not run)
```

Index

`:::`, [4](#)

`%==%` (assert), [2](#)

assert, [2](#)

deparse, [3](#)

has_error (has_warning), [4](#)

has_warning, [4](#)

identical, [2](#)

source, [5](#)

stopifnot, [2](#)

str, [2](#)

sys.source, [5](#)

test_pkg, [4](#)