Package 'testit'

November 2, 2026

Type Package

Title A Simple Package for Testing R Packages

Version 1.3

Description A minimal, dependency-free testing framework for R packages. Write tests as simple R expressions that return TRUE, using assert() for assertions (with informative error messages on failure), has_error() / has_warning() / has_message() for testing conditions, and test_pkg() to run all tests with full access to internal (non-exported) package functions. Snapshot testing via Markdown files is also supported.

License MIT + file LICENSE

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

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

Encoding UTF-8

Config/roxygen2/version 8.1.1

NeedsCompilation no

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

Maintainer Yihui Xie <xie@yihui.name>

Repository CRAN

Date/Publication 2026-11-02 19:10:02 UTC

Contents

assert 2
has_match 3
has_message 4
test_pkg 5

Index **8**

`assert` *Assert that conditions are true, with an informative failure message*

Description

Test that one or more conditions are `TRUE`. If any condition fails, an error is raised with the `fact` message, making it easy to identify which test failed and why. This is the primary function for writing tests with **testit**.

The infix operator `%==%` is a shortcut for `identical()` that provides helpful diagnostics on failure. `x %==% y` returns `TRUE` if `x` and `y` are identical, and `FALSE` otherwise. When used inside `assert()`, a failing `%==%` comparison will display both values via `str()` so you can see exactly what differed.

Usage

```
assert(fact, ...)

x %==% y
```

Arguments

`fact`	A character string describing what is being tested. This message is shown when an assertion fails, so make it descriptive (e.g., `'log() returns correct values'`). If `fact` is not a character string, it is treated as a test expression (i.e., the message is optional).
`...`	An R expression wrapped in `{}`; see Details.
`x, y`	Two R objects to be compared for identity.

Details

The recommended usage is to pass a single expression wrapped in `{}` as the second argument. Inside `{}`, any **statement-level** sub-expression wrapped in parentheses `()` is treated as a test condition – its value is checked and must be `TRUE`. Parentheses used for grouping within a larger expression (e.g., `(a + b) * c`) are not checked. Sub-expressions *without* parentheses are ordinary R code (e.g., variable assignments or setup steps) and are never checked.
`()` tests work inside `if`, `for`, `while`, and `repeat` bodies, as well as inside a `{}` block passed as an argument to a function call (e.g. `xfun::in_dir(dir, { ... })`). Internally, `assert()` walks the expression tree and transforms statement-level `()` into checks before evaluating the entire block in one frame (so `on.exit()` works as expected).

Value

Invisible `NULL` if all conditions pass. If any condition fails, an error is signaled that includes the `fact` message and the expression that failed. For `%==%`, `TRUE` or `FALSE`.

Note

Key differences from `stopifnot()`:

- `assert()` shows your custom `fact` message on failure, making errors easier to diagnose.
- `logical(0)` (empty logical) is treated as a failure, not a pass.
- All conditions are evaluated even if earlier ones fail; all failures are reported together in a single error message.

Examples

```
library(testit)
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)
})

# () works inside control structures too
assert('conditional test', {
  if (requireNamespace('base', quietly = TRUE)) (1 + 1 == 2)
})
```

---

has_match *Test whether a string matches a regular expression*

Description

Check whether `x` matches the regular expression `pattern`. This is intended for use inside `assert()`: unlike `grepl()`, which only returns `TRUE`/`FALSE` and hides the input on failure, a failing `has_match()` shows the actual string so you can see what was produced.

Usage

```
has_match(
  x,
  pattern,
  negate = FALSE,
  fixed = inherits(pattern, "AsIs"),
  perl = !fixed,
  ...
)
```

Arguments

x	A character vector. Multiple elements are collapsed into a single string (separated by newlines) before matching.
pattern	A regular expression. By default, matching uses Perl mode with the "dotall" flag enabled, so . also matches newlines, which is convenient for searching multiline text such as rendered HTML. Wrap the pattern in I() to match it literally, as a shorthand for fixed = TRUE.
negate	If TRUE, test that x does *not* match pattern. As a shorthand, negating the pattern itself (has_match(x, !pattern)) sets this to TRUE. Prefer either form over negating the whole call (!has_match(x, p)), since negate also makes the failure message show x.
fixed	Whether to match pattern literally instead of as a regular expression. Defaults to TRUE when pattern is wrapped in I(), so has_match(x, I(p)) is a shorthand for has_match(x, p, fixed = TRUE).
perl	Whether to use Perl-compatible regular expressions (which also enables the dotall flag). Defaults to FALSE for a fixed search, which uses neither.
...	Additional arguments passed to grepl() (e.g., ignore.case = TRUE).

Value

TRUE if x matches pattern (or does not match it when negate = TRUE), otherwise FALSE.

Examples

```
library(testit)
assert('the page has a script but no style', {
  html = '<html>\n<script>x</script>\n</html>'
  (has_match(html, '<script>.*</script>'))
  (has_match(html, '<style>', negate = TRUE))
  (has_match(html, !'<style>'))  # same, via the !pattern shorthand
  (has_match(html, I('<script>x</script>')))  # literal match (fixed = TRUE)
})
```

---

has_message *Test whether an expression signals a condition*

Description

Check if evaluating an expression produces a message, warning, or error. These functions are designed to be used inside assert() to verify that code signals the expected conditions. Optionally, you can match against the condition's text to ensure the *right* message/warning/error was signaled.

Usage

```
has_message(expr, message = NULL, ...)

has_warning(expr, message = NULL, ...)

has_error(expr, message = NULL, ...)
```

Arguments

expr	An R expression to evaluate.
message	An optional string to match against the condition text. Uses fixed (literal) matching by default. If provided, the function returns TRUE only when the condition is signaled *and* the message matches.
...	Additional arguments passed to grepl() for matching (e.g., fixed = FALSE to use regex, or ignore.case = TRUE). Note that fixed = TRUE is the default.

Value

TRUE if the condition was signaled (and the message matched, if provided), FALSE otherwise.

Examples

```
has_message(message('hello'))
has_message(1 + 1)
has_message(message('hello world'), 'hello')

has_warning(1 + 1)
has_warning(1:2 + 1:3)
has_warning(1:2 + 1:3, 'longer object length')

has_error(2 - 3)
has_error(1 + 'a')
has_error(stop('err'), 'err')
has_error(stop('error occurred'), 'error')
```

---

test_pkg *Run all tests for a package*

Description

Discover and execute test files (test-*.R and test-*.md) for a package. Tests are run inside the package namespace, so you can call internal (non-exported) functions directly without the ::: operator.

Usage

```
test_pkg(package = pkg_name(), dir = NULL, filter = NULL, update = NA)
```

Arguments

package	The package name. By default, it is detected from the DESCRIPTION file.
dir	The directory containing test files. If NULL (the default), testit/ or tests/testit/ under the current working directory is used (whichever exists). You can also pass a custom path.
filter	An optional regular expression to select a subset of test files. Only files whose names match the pattern will be run. For example, filter = "plot" runs only test files with "plot" in their names.
update	Controls snapshot file behavior: • TRUE: always update snapshot files with actual output (never errors). • NA (default): update only if the file is tracked by Git (so you can review diffs before accepting). • FALSE: never update; always compare and error on mismatch.

Details

Test files are looked up in the testit/ or tests/testit/ directory by default. Files must be named test-*.R for regular tests or test-*.md for snapshot tests. Other files in the directory are ignored (but you can source() them from your tests if needed).

Helper files named helper*.R (e.g., helper.R, helper-utils.R) in the test directory are sourced before any test file runs. Objects defined in helpers are available to all tests. To share helpers across multiple test subdirectories, put them in a sibling helpers/ directory (e.g., tests/helpers/); all .R files there are sourced first, before the test subdirectory's own helpers.

Each test file runs in a clean environment (previous test objects are removed), and the working directory is set to the directory containing the test file.

See https://pkg.yihui.org/testit/#sec:snapshot-testing for more details about snapshot testing.

Value

Invisible NULL. If any tests fail, a single error is thrown at the end with all failure messages combined.

Note

You must call library(testit) before test_pkg(). Test scripts use assert() and other **testit** functions without the testit:: prefix, so the package needs to be on the search path. Without library(testit), you will get "could not find function" errors.

All test scripts must be encoded in UTF-8 if they contain multibyte characters.

When filter or update are not explicitly provided, test_pkg() checks commandArgs(TRUE) for command-line arguments: --filter=PATTERN sets the filter, and --update sets update = TRUE. This allows you to pass these options via Rscript tests/*.R --filter=PATTERN --update without modifying individual test_pkg() calls.

Examples

```
## Not run:
library(testit)
test_pkg('testit')

## End(Not run)
```

Index

%==% *(*assert*)*, 2

assert, 2
assert(), *3*, *4*, *6*

grepl(), *3–5*

has_error *(*has_message*)*, 4
has_match, 3
has_message, 4
has_warning *(*has_message*)*, 4

I(), *4*
identical(), *2*

source(), *6*
stopifnot(), *3*
str(), *2*

test_pkg, 5