

# Package ‘testthat’

April 24, 2019

**Title** Unit Testing for R

**Version** 2.1.1

**Description** Software testing is important, but, in part because it is frustrating and boring, many of us avoid it. 'testthat' is a testing framework for R that is easy learn and use, and integrates with your existing 'workflow'.

**License** MIT + file LICENSE

**URL** <http://testthat.r-lib.org>, <https://github.com/r-lib/testthat>

**BugReports** <https://github.com/r-lib/testthat/issues>

**Depends** R (>= 3.1)

**Imports** cli, crayon (>= 1.3.4), digest, magrittr, methods, praise, R6 (>= 2.2.0), rlang (>= 0.3.0), withr (>= 2.0.0)

**Suggests** covr, curl (>= 0.9.5), devtools, knitr, rmarkdown, usethis, vctrs (>= 0.1.0), xml2

**VignetteBuilder** knitr

**Encoding** UTF-8

**RoxygenNote** 6.1.1

**Collate** 'auto-test.R' 'capture-condition.R' 'capture-output.R' 'colour-text.R' 'compare.R' 'compare-character.R' 'compare-numeric.R' 'compare-time.R' 'context.R' 'describe.R' 'evaluate-promise.R' 'example.R' 'expect-comparison.R' 'expect-equality.R' 'expect-inheritance.R' 'expect-invisible.R' 'expect-known.R' 'expect-length.R' 'expect-logical.R' 'expect-named.R' 'expect-null.R' 'expect-output.R' 'reporter.R' 'expect-self-test.R' 'expect-setequal.R' 'expect-that.R' 'expect-vector.R' 'expectation.R' 'expectations-matches.R' 'make-expectation.R' 'mock.R' 'old-school.R' 'praise.R' 'quasi-label.R' 'recover.R' 'reporter-check.R' 'reporter-debug.R' 'reporter-fail.R' 'reporter-junit.R' 'reporter-list.R' 'reporter-location.R' 'reporter-minimal.R' 'reporter-multi.R' 'stack.R' 'reporter-progress.R' 'reporter-rstudio.R' 'reporter-silent.R' 'reporter-stop.R' 'reporter-summary.R' 'reporter-tap.R' 'reporter-teamcity.R'

'reporter-zzz.R' 'skip.R' 'source.R' 'teardown.R'  
 'test-compiled-code.R' 'test-directory.R' 'test-example.R'  
 'test-files.R' 'test-path.R' 'test-that.R' 'traceback.R'  
 'try-again.R' 'utils-io.R' 'utils.R' 'watcher.R'

**NeedsCompilation** yes

**Author** Hadley Wickham [aut, cre],  
 RStudio [cph, fnd],  
 R Core team [ctb] (Implementation of utils::recover())

**Maintainer** Hadley Wickham <hadley@rstudio.com>

**Repository** CRAN

**Date/Publication** 2019-04-23 22:40:03 UTC

## R topics documented:

auto_test . . . . .	3
auto_test_package . . . . .	4
CheckReporter . . . . .	4
comparison-expectations . . . . .	5
DebugReporter . . . . .	6
describe . . . . .	6
equality-expectations . . . . .	7
expect . . . . .	9
expect_cpp_tests_pass . . . . .	10
expect_invisible . . . . .	10
expect_known_output . . . . .	11
expect_length . . . . .	12
expect_match . . . . .	13
expect_named . . . . .	14
expect_null . . . . .	15
expect_setequal . . . . .	16
expect_vector . . . . .	17
fail . . . . .	18
FailReporter . . . . .	18
inheritance-expectations . . . . .	19
JUnitReporter . . . . .	20
ListReporter . . . . .	20
LocationReporter . . . . .	21
logical-expectations . . . . .	21
MinimalReporter . . . . .	22
MultiReporter . . . . .	23
output-expectations . . . . .	23
ProgressReporter . . . . .	26
RstudioReporter . . . . .	27
SilentReporter . . . . .	27
skip . . . . .	28
StopReporter . . . . .	29

SummaryReporter . . . . .	30
TapReporter . . . . .	31
TeamcityReporter . . . . .	31
teardown . . . . .	32
test_dir . . . . .	32
test_examples . . . . .	34
test_file . . . . .	35
test_path . . . . .	36
test_that . . . . .	36
use_catch . . . . .	37

<b>Index</b>	<b>40</b>
--------------	-----------

---

auto_test	<i>Watches code and tests for changes, rerunning tests as appropriate.</i>
-----------	--

---

## Description

The idea behind `auto_test()` is that you just leave it running while you develop your code. Everytime you save a file it will be automatically tested and you can easily see if your changes have caused any test failures.

## Usage

```
auto_test(code_path, test_path, reporter = default_reporter(),
          env = test_env(), hash = TRUE)
```

## Arguments

code_path	path to directory containing code
test_path	path to directory containing tests
reporter	test reporter to use
env	environment in which to execute test suite.
hash	Passed on to <code>watch()</code> . When <code>FALSE</code> , uses less accurate modification time stamps, but those are faster for large files.

## Details

The current strategy for rerunning tests is as follows:

- if any code has changed, then those files are reloaded and all tests rerun
- otherwise, each new or modified test is run

In the future, `auto_test()` might implement one of the following more intelligent alternatives:

- Use codetools to build up dependency tree and then rerun tests only when a dependency changes.
- Mimic ruby's autotest and rerun only failing tests until they pass, and then rerun all tests.

**See Also**

[auto\\_test\\_package\(\)](#)

---

auto_test_package	<i>Watches a package for changes, rerunning tests as appropriate.</i>
-------------------	---

---

**Description**

Watches a package for changes, rerunning tests as appropriate.

**Usage**

```
auto_test_package(pkg = ".", reporter = default_reporter(),
  hash = TRUE)
```

**Arguments**

pkg	path to package
reporter	test reporter to use
hash	Passed on to <a href="#">watch()</a> . When FALSE, uses less accurate modification time stamps, but those are faster for large files.

**See Also**

[auto\\_test\(\)](#) for details on how method works

---

CheckReporter	<i>Check reporter: 13 line summary of problems</i>
---------------	--

---

**Description**

R CMD check displays only the last 13 lines of the result, so this report is design to ensure that you see something useful there.

**Usage**

```
CheckReporter
```

**Format**

An object of class R6ClassGenerator of length 24.

**See Also**

Other reporters: [DebugReporter](#), [FailReporter](#), [ListReporter](#), [LocationReporter](#), [MinimalReporter](#), [MultiReporter](#), [ProgressReporter](#), [Reporter](#), [RstudioReporter](#), [SilentReporter](#), [StopReporter](#), [SummaryReporter](#), [TapReporter](#), [TeamcityReporter](#)

---

comparison-expectations

*Expectation: is returned value less or greater than specified value?*

---

## Description

Expectation: is returned value less or greater than specified value?

## Usage

```
expect_lt(object, expected, label = NULL, expected.label = NULL)
```

```
expect_lte(object, expected, label = NULL, expected.label = NULL)
```

```
expect_gt(object, expected, label = NULL, expected.label = NULL)
```

```
expect_gte(object, expected, label = NULL, expected.label = NULL)
```

## Arguments

object	Object to test. Supports limited unquoting to make it easier to generate readable failures within a function or for loop. See <a href="#">quasi_label</a> for more details.
expected	Single numeric value to compare.
label	Used to customise failure messages. For expert use only.
expected.label	Used to customise failure messages. For expert use only.
...	For <code>expect_equal()</code> and <code>expect_equivalent()</code> , passed on <a href="#">compare()</a> , for <code>expect_identical()</code> passed on to <a href="#">identical()</a> . Used to control the details of the comparison.

## See Also

Other expectations: [equality-expectations](#), [expect\\_length](#), [expect\\_match](#), [expect\\_named](#), [expect\\_null](#), [inheritance-expectations](#), [logical-expectations](#), [output-expectations](#)

## Examples

```
a <- 9
expect_lt(a, 10)

## Not run:
expect_lt(11, 10)

## End(Not run)

a <- 11
expect_gt(a, 10)
```

```
## Not run:
expect_gt(9, 10)

## End(Not run)
```

---

DebugReporter	<i>Test reporter: start recovery.</i>
---------------	---------------------------------------

---

### Description

This reporter will call a modified version of [recover\(\)](#) on all broken expectations.

### Usage

```
DebugReporter
```

### Format

An object of class R6ClassGenerator of length 24.

### See Also

Other reporters: [CheckReporter](#), [FailReporter](#), [ListReporter](#), [LocationReporter](#), [MinimalReporter](#), [MultiReporter](#), [ProgressReporter](#), [Reporter](#), [RstudioReporter](#), [SilentReporter](#), [StopReporter](#), [SummaryReporter](#), [TapReporter](#), [TeamcityReporter](#)

---

describe	<i>describe: a BDD testing language</i>
----------	---

---

### Description

A simple BDD DSL for writing tests. The language is similiar to RSpec for Ruby or Mocha for JavaScript. BDD tests read like sentences and it should thus be easier to understand what the specification of a function/component is.

### Usage

```
describe(description, code)
```

### Arguments

description	description of the feature
code	test code containing the specs

## Details

Tests using the describe syntax not only verify the tested code, but also document its intended behaviour. Each describe block specifies a larger component or function and contains a set of specifications. A specification is defined by an it block. Each it block functions as a test and is evaluated in its own environment. You can also have nested describe blocks.

This test syntax helps to test the intended behaviour of your code. For example: you want to write a new function for your package. Try to describe the specification first using describe, before you write any code. After that, you start to implement the tests for each specification (i.e. the it block).

Use describe to verify that you implement the right things and use `test_that()` to ensure you do the things right.

## Examples

```
describe("matrix()", {
  it("can be multiplied by a scalar", {
    m1 <- matrix(1:4, 2, 2)
    m2 <- m1 * 2
    expect_equivalent(matrix(1:4 * 2, 2, 2), m2)
  })
  it("can have not yet tested specs")
})

# Nested specs:
## code
addition <- function(a, b) a + b
division <- function(a, b) a / b

## specs
describe("math library", {
  describe("addition()", {
    it("can add two numbers", {
      expect_equivalent(1 + 1, addition(1, 1))
    })
  })
  describe("division()", {
    it("can divide two numbers", {
      expect_equivalent(10 / 2, division(10, 2))
    })
    it("can handle division by 0") #not yet implemented
  })
})
```

**Description**

- `expect_identical()` compares values with `identical()`.
- `expect_equal()` compares values with `all.equal()`
- `expect_equivalent()` compares values with `all.equal()` and `check.attributes = FALSE`
- `expect_reference()` compares the underlying memory addresses.

**Usage**

```
expect_equal(object, expected, ..., info = NULL, label = NULL,
             expected.label = NULL)
```

```
expect_equivalent(object, expected, ..., info = NULL, label = NULL,
                  expected.label = NULL)
```

```
expect_identical(object, expected, info = NULL, label = NULL,
                 expected.label = NULL, ...)
```

```
expect_reference(object, expected, info = NULL, label = NULL,
                 expected.label = NULL)
```

**Arguments**

`object`, `expected`

Computation and value to compare it to.

Both arguments supports limited unquoting to make it easier to generate readable failures within a function or for loop. See [quasi\\_label](#) for more details.

`...` For `expect_equal()` and `expect_equivalent()`, passed on `compare()`, for `expect_identical()` passed on to `identical()`. Used to control the details of the comparison.

`info` Extra information to be included in the message. This argument is soft-deprecated and should not be used in new code. Instead see alternatives in [quasi\\_label](#).

`label`, `expected.label`

Used to customise failure messages. For expert use only.

**See Also**

`expect_setequal()` to test for set equality.

Other expectations: [comparison-expectations](#), [expect\\_length](#), [expect\\_match](#), [expect\\_named](#), [expect\\_null](#), [inheritance-expectations](#), [logical-expectations](#), [output-expectations](#)

**Examples**

```
a <- 10
expect_equal(a, 10)

# Use expect_equal() when testing for numeric equality
sqrt(2) ^ 2 - 1
```



```

expect_equal(sqrt(2) ^ 2, 2)
# Neither of these forms take floating point representation errors into
# account
## Not run:
expect_true(sqrt(2) ^ 2 == 2)
expect_identical(sqrt(2) ^ 2, 2)

## End(Not run)

# You can pass on additional arguments to all.equal:
## Not run:
# Test the ABSOLUTE difference is within .002
expect_equal(10.01, 10, tolerance = .002, scale = 1)

## End(Not run)

# Test the RELATIVE difference is within .002
x <- 10
expect_equal(10.01, expected = x, tolerance = 0.002, scale = x)

# expect_equivalent ignores attributes
a <- b <- 1:3
names(b) <- letters[1:3]
expect_equivalent(a, b)

```

---

expect

*The building block of all expect\_ functions*

---

## Description

Call this function when writing your own expectations. See vignette("custom-expectation") for details.

## Usage

```
expect(ok, failure_message, info = NULL, srcref = NULL)
```

## Arguments

ok	TRUE or FALSE indicating if the expectation was successful.
failure_message	Message to show if the expectation failed.
info	Character vector continuing additional information. Included for backward compatibility only and new expectations should not use it.
srcref	Location of the failure. Should only needed to be explicitly supplied when you need to forward a srcref captured elsewhere.

## Value

An expectation object. Signals the expectation condition with a `continue_test` restart.

---

expect\_cpp\_tests\_pass *Test Compiled Code in a Package*

---

**Description**

Test compiled code in the package `package`. See [use\\_catch\(\)](#) for more details.

**Usage**

```
expect_cpp_tests_pass(package)
```

**Arguments**

`package`            The name of the package to test.

**Note**

A call to this function will automatically be generated for you in `tests/testthat/test-cpp.R` after calling `use_catch()`; you should not need to manually call this expectation yourself.

---

expect\_invisible            *Expectation: does expression return visibly or invisibly?*

---

**Description**

Use this to test whether a function returns a visible or invisible output. Typically you'll use this to check that functions called primarily for their side-effects return their data argument invisibly.

**Usage**

```
expect_invisible(call, label = NULL)
```

```
expect_visible(call, label = NULL)
```

**Arguments**

`call`                A function call.

`label`               Used to customise failure messages. For expert use only.

**Value**

The evaluated `call`, invisibly.

**Examples**

```
expect_invisible(x <- 10)
expect_visible(x)

# Typically you'll assign the result of the expectation so you can
# also check that the value is as you expect.
greet <- function(name) {
  message("Hi ", name)
  invisible(name)
}
out <- expect_invisible(greet("Hadley"))
expect_equal(out, "Hadley")
```

---

expect\_known\_output     *Expectations: is the output or the value equal to a known good value?*

---

**Description**

For complex printed output and objects, it is often challenging to describe exactly what you expect to see. `expect_known_value()` and `expect_known_output()` provide a slightly weaker guarantee, simply asserting that the values have not changed since the last time that you ran them.

**Usage**

```
expect_known_output(object, file, update = TRUE, ..., info = NULL,
  label = NULL, print = FALSE, width = 80)
```

```
expect_known_value(object, file, update = TRUE, ..., info = NULL,
  label = NULL)
```

```
expect_known_hash(object, hash = NULL)
```

**Arguments**

object	Computation and value to compare it to. Both arguments supports limited unquoting to make it easier to generate readable failures within a function or for loop. See <a href="#">quasi_label</a> for more details.
file	File path where known value/output will be stored.
update	Should the file be updated? Defaults to TRUE, with the expectation that you'll notice changes because of the first failure, and then see the modified files in git.
...	For <code>expect_equal()</code> and <code>expect_equivalent()</code> , passed on <a href="#">compare()</a> , for <code>expect_identical()</code> passed on to <a href="#">identical()</a> . Used to control the details of the comparison.
info	Extra information to be included in the message. This argument is soft-deprecated and should not be used in new code. Instead see alternatives in <a href="#">quasi_label</a> .
label	Used to customise failure messages. For expert use only.

print	If TRUE and the result of evaluating code is visible this will print the result, ensuring that the output of printing the object is included in the overall output
width	Number of characters per line of output. This does not inherit from <code>getOption("width")</code> so that tests always use the same output width, minimising spurious differences.
hash	Known hash value. Leave empty and you'll be informed what to use in the test output.

## Details

These expectations should be used in conjunction with `git`, as otherwise there is no way to revert to previous values. `git` is particularly useful in conjunction with `expect_known_output()` as the diffs will show you exactly what has changed.

Note that known values updates will only be updated when running tests interactively. R CMD check clones the package source so any changes to the reference files will occur in a temporary directory, and will not be synchronised back to the source package.

## Examples

```
tmp <- tempfile()

# The first run always succeeds
expect_known_output(mtcars[1:10, ], tmp, print = TRUE)

# Subsequent runs will succeed only if the file is unchanged
# This will succeed:
expect_known_output(mtcars[1:10, ], tmp, print = TRUE)

## Not run:
# This will fail
expect_known_output(mtcars[1:9, ], tmp, print = TRUE)

## End(Not run)
```

---

expect\_length

*Expectation: does a vector have the specified length?*

---

## Description

Expectation: does a vector have the specified length?

## Usage

```
expect_length(object, n)
```

**Arguments**

object	Object to test. Supports limited unquoting to make it easier to generate readable failures within a function or for loop. See <a href="#">quasi_label</a> for more details.
n	Expected length.

**See Also**

Other expectations: [comparison-expectations](#), [equality-expectations](#), [expect\\_match](#), [expect\\_named](#), [expect\\_null](#), [inheritance-expectations](#), [logical-expectations](#), [output-expectations](#)

**Examples**

```
expect_length(1, 1)
expect_length(1:10, 10)

## Not run:
expect_length(1:10, 1)

## End(Not run)
```

---

expect_match	<i>Expectation: does string match a regular expression?</i>
--------------	---

---

**Description**

Expectation: does string match a regular expression?

**Usage**

```
expect_match(object, regexp, perl = FALSE, fixed = FALSE, ...,
             all = TRUE, info = NULL, label = NULL)
```

**Arguments**

object	Object to test. Supports limited unquoting to make it easier to generate readable failures within a function or for loop. See <a href="#">quasi_label</a> for more details.
regexp	Regular expression to test against.
perl	logical. Should Perl-compatible regexps be used?
fixed	logical. If TRUE, pattern is a string to be matched as is. Overrides all conflicting arguments.
...	Arguments passed on to base: <code>:grep1</code> <b>ignore.case</b> if FALSE, the pattern matching is <i>case sensitive</i> and if TRUE, case is ignored during matching.

	<b>useBytes</b> logical. If TRUE the matching is done byte-by-byte rather than character-by-character. See ‘Details’.
all	Should all elements of actual value match regexp (TRUE), or does only one need to match (FALSE)
info	Extra information to be included in the message. This argument is soft-deprecated and should not be used in new code. Instead see alternatives in <a href="#">quasi_label</a> .
label	Used to customise failure messages. For expert use only.

### Details

expect\_match() is a wrapper around [grepl\(\)](#). See its documentation for more detail about the individual arguments.

### See Also

Other expectations: [comparison-expectations](#), [equality-expectations](#), [expect\\_length](#), [expect\\_named](#), [expect\\_null](#), [inheritance-expectations](#), [logical-expectations](#), [output-expectations](#)

### Examples

```
expect_match("Testing is fun", "fun")
expect_match("Testing is fun", "f.n")

## Not run:
expect_match("Testing is fun", "horrible")

# Zero-length inputs always fail
expect_match(character(), ".")

## End(Not run)
```

---

expect_named	<i>Expectation: does object have names?</i>
--------------	---

---

### Description

You can either check for the presence of names (leaving expected blank), specific names (by supplying a vector of names), or absence of names (with NULL).

### Usage

```
expect_named(object, expected, ignore.order = FALSE,
             ignore.case = FALSE, info = NULL, label = NULL)
```

**Arguments**

object	Object to test. Supports limited unquoting to make it easier to generate readable failures within a function or for loop. See <a href="#">quasi_label</a> for more details.
expected	Character vector of expected names. Leave missing to match any names. Use NULL to check for absence of names.
ignore.order	If TRUE, sorts names before comparing to ignore the effect of order.
ignore.case	If TRUE, lowercases all names to ignore the effect of case.
info	Extra information to be included in the message. This argument is soft-deprecated and should not be used in new code. Instead see alternatives in <a href="#">quasi_label</a> .
label	Used to customise failure messages. For expert use only.
...	Other arguments passed on to <a href="#">has_names()</a> .

**See Also**

Other expectations: [comparison-expectations](#), [equality-expectations](#), [expect\\_length](#), [expect\\_match](#), [expect\\_null](#), [inheritance-expectations](#), [logical-expectations](#), [output-expectations](#)

**Examples**

```
x <- c(a = 1, b = 2, c = 3)
expect_named(x)
expect_named(x, c("a", "b", "c"))

# Use options to control sensitivity
expect_named(x, c("B", "C", "A"), ignore.order = TRUE, ignore.case = TRUE)

# Can also check for the absence of names with NULL
z <- 1:4
expect_named(z, NULL)
```

---

expect_null	<i>Expectation: is an object NULL?</i>
-------------	--

---

**Description**

This is a special case because NULL is a singleton so it's possible to check for it either with `expect_equal(x, NULL)` or `expect_type(x, "NULL")`.

**Usage**

```
expect_null(object, info = NULL, label = NULL)
```

**Arguments**

object	Object to test. Supports limited unquoting to make it easier to generate readable failures within a function or for loop. See <a href="#">quasi_label</a> for more details.
info	Extra information to be included in the message. This argument is soft-deprecated and should not be used in new code. Instead see alternatives in <a href="#">quasi_label</a> .
label	Used to customise failure messages. For expert use only.

**See Also**

Other expectations: [comparison-expectations](#), [equality-expectations](#), [expect\\_length](#), [expect\\_match](#), [expect\\_named](#), [inheritance-expectations](#), [logical-expectations](#), [output-expectations](#)

**Examples**

```
x <- NULL
y <- 10

expect_null(x)
show_failure(expect_null(y))
```

---

expect_setequal	<i>Expectation: do two vectors contain the same values?</i>
-----------------	---

---

**Description**

- `expect_setequal(x, y)` tests that every element of `x` occurs in `y`, and that every element of `y` occurs in `x`.
- `expect_mapequal(x, y)` tests that `x` and `y` have the same names, and that `x[names(y)]` equals `x`.

**Usage**

```
expect_setequal(object, expected)
```

```
expect_mapequal(object, expected)
```

**Arguments**

object	Computation and value to compare it to. Both arguments supports limited unquoting to make it easier to generate readable failures within a function or for loop. See <a href="#">quasi_label</a> for more details.
expected	Computation and value to compare it to. Both arguments supports limited unquoting to make it easier to generate readable failures within a function or for loop. See <a href="#">quasi_label</a> for more details.



**Details**

Note that `expect_setequal()` ignores names, and you will be warned if both object and expected have them.

**Examples**

```
expect_setequal(letters, rev(letters))
show_failure(expect_setequal(letters[-1], rev(letters)))

x <- list(b = 2, a = 1)
expect_mapequal(x, list(a = 1, b = 2))
show_failure(expect_mapequal(x, list(a = 1)))
show_failure(expect_mapequal(x, list(a = 1, b = "x")))
show_failure(expect_mapequal(x, list(a = 1, b = 2, c = 3)))
```

---

expect\_vector

*Expectation: does the object have vctr properties?*

---

**Description**

`expect_vector()` is a thin wrapper around `vctrs::vec_assert()`, converting the results of that function in to the expectations used by `testthat`. This means that it used the `vctrs` of `ptype` (prototype) and `size`. See details in <https://vctrs.r-lib.org/articles/type-size.html>

**Usage**

```
expect_vector(object, ptype = NULL, size = NULL)
```

**Arguments**

<code>object</code>	Object to test. Supports limited unquoting to make it easier to generate readable failures within a function or for loop. See <a href="#">quasi_label</a> for more details.
<code>ptype</code>	(Optional) Vector prototype to test against. Should be a size-0 (empty) generalised vector.
<code>size</code>	(Optional) Size to check for.

**Examples**

```
if (requireNamespace("vctrs") && packageVersion("vctrs") > "0.1.0.9002") {
  expect_vector(1:10, ptype = integer(), size = 10)
  show_failure(expect_vector(1:10, ptype = integer(), size = 5))
  show_failure(expect_vector(1:10, ptype = character(), size = 5))
}
```

fail *Default expectations that always succeed or fail.*

---

### Description

These allow you to manually trigger success or failure. Failure is particularly useful to a precondition or mark a test as not yet implemented.

### Usage

```
fail(message = "Failure has been forced")  
  
succeed(message = "Success has been forced")
```

### Arguments

message            a string to display.

### Examples

```
## Not run:  
test_that("this test fails", fail())  
test_that("this test succeeds", succeed())  
  
## End(Not run)
```

---

FailReporter *Test reporter: fail at end.*

---

### Description

This reporter will simply throw an error if any of the tests failed. It is best combined with another reporter, such as the [SummaryReporter](#).

### Usage

```
FailReporter
```

### Format

An object of class R6ClassGenerator of length 24.

### See Also

Other reporters: [CheckReporter](#), [DebugReporter](#), [ListReporter](#), [LocationReporter](#), [MinimalReporter](#), [MultiReporter](#), [ProgressReporter](#), [Reporter](#), [RstudioReporter](#), [SilentReporter](#), [StopReporter](#), [SummaryReporter](#), [TapReporter](#), [TeamcityReporter](#)

---

inheritance-expectations

*Expectation: does the object inherit from a S3 or S4 class, or is it a base type?*

---

## Description

See <https://adv-r.hadley.nz/oo.html> for an overview of R's OO systems, and the vocabulary used here.

- `expect_type(x, type)` checks that `typeof(x)` is `type`.
- `expect_s3_class(x, class)` checks that `x` is an S3 object that `inherits()` from `class`
- `expect_s4_class(x, class)` checks that `x` is an S4 object that `is()` class.

## Usage

```
expect_type(object, type)
```

```
expect_s3_class(object, class)
```

```
expect_s4_class(object, class)
```

## Arguments

<code>object</code>	Object to test. Supports limited unquoting to make it easier to generate readable failures within a function or for loop. See <a href="#">quasi_label</a> for more details.
<code>type</code>	String giving base type (as returned by <code>typeof()</code> ).
<code>class</code>	character vector of class names

## See Also

Other expectations: [comparison-expectations](#), [equality-expectations](#), [expect\\_length](#), [expect\\_match](#), [expect\\_named](#), [expect\\_null](#), [logical-expectations](#), [output-expectations](#)

## Examples

```
x <- data.frame(x = 1:10, y = "x")
# A data frame is an S3 object with class data.frame
expect_s3_class(x, "data.frame")
show_failure(expect_s4_class(x, "data.frame"))
# A data frame is built from a list:
expect_type(x, "list")

# An integer vector is an atomic vector of type "integer"
expect_type(x$x, "integer")
# It is not an S3 object
```

```
show_failure(expect_s3_class(x$x, "integer"))

# By default data.frame() converts characters to factors:
show_failure(expect_type(x$y, "character"))
expect_s3_class(x$y, "factor")
expect_type(x$y, "integer")
```

---

JUnitReporter	<i>Test reporter: summary of errors in JUnit XML format.</i>
---------------	--

---

### Description

This reporter includes detailed results about each test and summaries, written to a file (or stdout) in JUnit XML format. This can be read by the Jenkins Continuous Integration System to report on a dashboard etc. Requires the *xml2* package.

### Usage

```
JUnitReporter
```

### Format

An object of class R6ClassGenerator of length 24.

### Details

To fit into the JUnit structure, context() becomes the <testsuite> name as well as the base of the <testcase> classname. The test\_that() name becomes the rest of the <testcase> classname. The deparsed expect\_that() call becomes the <testcase> name. On failure, the message goes into the <failure> node message argument (first line only) and into its text content (full message).

Execution time and some other details are also recorded.

References for the JUnit XML format: <http://llg.cubic.org/docs/junit/>

---

ListReporter	<i>List reporter: gather all test results along with elapsed time and file information.</i>
--------------	---

---

### Description

This reporter gathers all results, adding additional information such as test elapsed time, and test filename if available. Very useful for reporting.

### Usage

```
ListReporter
```

**Format**

An object of class R6ClassGenerator of length 24.

**See Also**

Other reporters: [CheckReporter](#), [DebugReporter](#), [FailReporter](#), [LocationReporter](#), [MinimalReporter](#), [MultiReporter](#), [ProgressReporter](#), [Reporter](#), [RstudioReporter](#), [SilentReporter](#), [StopReporter](#), [SummaryReporter](#), [TapReporter](#), [TeamcityReporter](#)

---

LocationReporter	<i>Test reporter: location</i>
------------------	--------------------------------

---

**Description**

This reporter simply prints the location of every expectation and error. This is useful if you're trying to figure out the source of a segfault, or you want to figure out which code triggers a C/C++ breakpoint

**Usage**

```
LocationReporter
```

**Format**

An object of class R6ClassGenerator of length 24.

**See Also**

Other reporters: [CheckReporter](#), [DebugReporter](#), [FailReporter](#), [ListReporter](#), [MinimalReporter](#), [MultiReporter](#), [ProgressReporter](#), [Reporter](#), [RstudioReporter](#), [SilentReporter](#), [StopReporter](#), [SummaryReporter](#), [TapReporter](#), [TeamcityReporter](#)

---

logical-expectations	<i>Expectation: is the object true/false?</i>
----------------------	---

---

**Description**

These are fall-back expectations that you can use when none of the other more specific expectations apply. The disadvantage is that you may get a less informative error message.

**Usage**

```
expect_true(object, info = NULL, label = NULL)
```

```
expect_false(object, info = NULL, label = NULL)
```

**Arguments**

object	Object to test. Supports limited unquoting to make it easier to generate readable failures within a function or for loop. See <a href="#">quasi_label</a> for more details.
info	Extra information to be included in the message. This argument is soft-deprecated and should not be used in new code. Instead see alternatives in <a href="#">quasi_label</a> .
label	Used to customise failure messages. For expert use only.

**Details**

Attributes are ignored.

**See Also**

[is\\_false\(\)](#) for complement

Other expectations: [comparison-expectations](#), [equality-expectations](#), [expect\\_length](#), [expect\\_match](#), [expect\\_named](#), [expect\\_null](#), [inheritance-expectations](#), [output-expectations](#)

**Examples**

```
expect_true(2 == 2)
# Failed expectations will throw an error
## Not run:
expect_true(2 != 2)

## End(Not run)
expect_true(!(2 != 2))
# or better:
expect_false(2 != 2)

a <- 1:3
expect_true(length(a) == 3)
# but better to use more specific expectation, if available
expect_equal(length(a), 3)
```

---

MinimalReporter

*Test reporter: minimal.*

---

**Description**

The minimal test reporter provides the absolutely minimum amount of information: whether each expectation has succeeded, failed or experienced an error. If you want to find out what the failures and errors actually were, you'll need to run a more informative test reporter.

**Usage**

```
MinimalReporter
```

**Format**

An object of class R6ClassGenerator of length 24.

**See Also**

Other reporters: [CheckReporter](#), [DebugReporter](#), [FailReporter](#), [ListReporter](#), [LocationReporter](#), [MultiReporter](#), [ProgressReporter](#), [Reporter](#), [RstudioReporter](#), [SilentReporter](#), [StopReporter](#), [SummaryReporter](#), [TapReporter](#), [TeamcityReporter](#)

---

MultiReporter

*Multi reporter: combine several reporters in one.*

---

**Description**

This reporter is useful to use several reporters at the same time, e.g. adding a custom reporter without removing the current one.

**Usage**

MultiReporter

**Format**

An object of class R6ClassGenerator of length 24.

**See Also**

Other reporters: [CheckReporter](#), [DebugReporter](#), [FailReporter](#), [ListReporter](#), [LocationReporter](#), [MinimalReporter](#), [ProgressReporter](#), [Reporter](#), [RstudioReporter](#), [SilentReporter](#), [StopReporter](#), [SummaryReporter](#), [TapReporter](#), [TeamcityReporter](#)

---

output-expectations

*Expectation: does code produce output/message/warning/error?*

---

**Description**

Use `expect_output()`, `expect_message()` and `expect_warning()` to match specified outputs. Use `expect_error()` or `expect_condition()` to match individual errors or conditions. Use `expect_silent()` to assert that there should be no output of any type.

**Usage**

```
expect_output(object, regexp = NULL, ..., info = NULL, label = NULL,
              width = 80)
```

```
expect_error(object, regexp = NULL, class = NULL, ..., info = NULL,
             label = NULL)
```

```
expect_condition(object, regexp = NULL, class = NULL, ...,
                 info = NULL, label = NULL)
```

```
expect_message(object, regexp = NULL, ..., all = FALSE, info = NULL,
             label = NULL)
```

```
expect_warning(object, regexp = NULL, ..., all = FALSE, info = NULL,
              label = NULL)
```

```
expect_silent(object)
```

**Arguments**

object	Object to test. Supports limited unquoting to make it easier to generate readable failures within a function or for loop. See <a href="#">quasi_label</a> for more details.
regexp	regular expression to test against. If NULL, the default, asserts that there should be an output, a message, a warning, or an error, but does not test for specific value. If NA, asserts that there should be no output, messages, warnings, or errors.
...	Arguments passed on to <code>expect_match</code> <b>all</b> Should all elements of actual value match <code>regexp</code> (TRUE), or does only one need to match (FALSE) <b>perl</b> logical. Should Perl-compatible regexps be used? <b>fixed</b> logical. If TRUE, <code>pattern</code> is a string to be matched as is. Overrides all conflicting arguments.
info	Extra information to be included in the message. This argument is soft-deprecated and should not be used in new code. Instead see alternatives in <a href="#">quasi_label</a> .
label	Used to customise failure messages. For expert use only.
width	Number of characters per line of output. This does not inherit from <code>getOption("width")</code> so that tests always use the same output width, minimising spurious differences.
class	Instead of supplying a regular expression, you can also supply a class name. This is useful for "classed" conditions.
all	For messages and warnings, do all need to match the <code>regexp</code> (TRUE), or does only one need to match (FALSE)

**Details**

Note that warnings are captured by a custom signal handler: this means that `options(warn)` has no effect.



**Value**

The first argument, invisibly. If `expect_error()` captures an error, that is returned instead of the value.

**See Also**

Other expectations: [comparison-expectations](#), [equality-expectations](#), [expect\\_length](#), [expect\\_match](#), [expect\\_named](#), [expect\\_null](#), [inheritance-expectations](#), [logical-expectations](#)

**Examples**

```
# Output -----
str(mtcars)
expect_output(str(mtcars), "32 obs")
expect_output(str(mtcars), "11 variables")

# You can use the arguments of grepl to control the matching
expect_output(str(mtcars), "11 VARIABLES", ignore.case = TRUE)
expect_output(str(mtcars), "$ mpg", fixed = TRUE)

# Messages -----

f <- function(x) {
  if (x < 0) message("*x* is already negative")
  -x
}
expect_message(f(-1))
expect_message(f(-1), "already negative")
expect_message(f(1), NA)

# You can use the arguments of grepl to control the matching
expect_message(f(-1), "*x*", fixed = TRUE)
expect_message(f(-1), "NEGATIVE", ignore.case = TRUE)

# Warnings -----

f <- function(x) {
  if (x < 0) warning("*x* is already negative")
  -x
}
expect_warning(f(-1))
expect_warning(f(-1), "already negative")
expect_warning(f(1), NA)

# You can use the arguments of grepl to control the matching
expect_warning(f(-1), "*x*", fixed = TRUE)
expect_warning(f(-1), "NEGATIVE", ignore.case = TRUE)

# Errors -----

f <- function() stop("My error!")
expect_error(f())
expect_error(f(), "My error!")
```

```
# You can use the arguments of grepl to control the matching
expect_error(f(), "my error!", ignore.case = TRUE)

# Silent -----
expect_silent("123")

f <- function() {
  message("Hi!")
  warning("Hey!!")
  print("OY!!!")
}
## Not run:
expect_silent(f())

## End(Not run)
```

---

ProgressReporter

*Test reporter: interactive progress bar of errors.*

---

## Description

This reporter is a reimagining of [SummaryReporter](#) designed to make the most information available up front, while taking up less space overall. It is the default reporting reporter used by [test\\_dir\(\)](#) and [test\\_file\(\)](#).

## Usage

ProgressReporter

## Format

An object of class R6ClassGenerator of length 24.

## Details

As an additional benefit, this reporter will praise you from time-to-time if all your tests pass.

## See Also

Other reporters: [CheckReporter](#), [DebugReporter](#), [FailReporter](#), [ListReporter](#), [LocationReporter](#), [MinimalReporter](#), [MultiReporter](#), [Reporter](#), [RstudioReporter](#), [SilentReporter](#), [StopReporter](#), [SummaryReporter](#), [TapReporter](#), [TeamcityReporter](#)

---

RstudioReporter	<i>Test reporter: RStudio</i>
-----------------	-------------------------------

---

**Description**

This reporter is designed for output to RStudio. It produces results in any easily parsed form.

**Usage**

RstudioReporter

**Format**

An object of class R6ClassGenerator of length 24.

**See Also**

Other reporters: [CheckReporter](#), [DebugReporter](#), [FailReporter](#), [ListReporter](#), [LocationReporter](#), [MinimalReporter](#), [MultiReporter](#), [ProgressReporter](#), [Reporter](#), [SilentReporter](#), [StopReporter](#), [SummaryReporter](#), [TapReporter](#), [TeamcityReporter](#)

---

SilentReporter	<i>Test reporter: gather all errors silently.</i>
----------------	---

---

**Description**

This reporter quietly runs all tests, simply gathering all expectations. This is helpful for programmatically inspecting errors after a test run. You can retrieve the results with the `expectations()` method.

**Usage**

SilentReporter

**Format**

An object of class R6ClassGenerator of length 24.

**See Also**

Other reporters: [CheckReporter](#), [DebugReporter](#), [FailReporter](#), [ListReporter](#), [LocationReporter](#), [MinimalReporter](#), [MultiReporter](#), [ProgressReporter](#), [Reporter](#), [RstudioReporter](#), [StopReporter](#), [SummaryReporter](#), [TapReporter](#), [TeamcityReporter](#)

---

skip	<i>Skip a test.</i>
------	---------------------

---

### Description

This function allows you to skip a test if it's not currently available. This will produce an informative message, but will not cause the test suite to fail.

### Usage

```
skip(message)

skip_if_not(condition, message = deparse(substitute(condition)))

skip_if(condition, message = deparse(substitute(condition)))

skip_if_not_installed(pkg, minimum_version = NULL)

skip_if_offline(host = "r-project.org")

skip_on_cran()

skip_on_os(os)

skip_on_travis()

skip_on_appveyor()

skip_on_ci()

skip_on_bioc()

skip_if_translated()
```

### Arguments

message	A message describing why the test was skipped.
condition	Boolean condition to check. <code>skip_if_not()</code> will skip if FALSE, <code>skip_if()</code> will skip if TRUE.
pkg	Name of package to check for
minimum_version	Minimum required version for the package
host	A string with a hostname to lookup
os	Character vector of system names. Supported values are "windows", "mac", "linux" and "solaris".

## Details

skip\* functions are intended for use within `test_that()` blocks. All expectations following the skip\* statement within the same `test_that` block will be skipped. Test summaries that report skip counts are reporting how many `test_that` blocks triggered a skip\* statement, not how many expectations were skipped.

## Helpers

`skip_if_not()` works like `stopifnot()`, generating a message automatically based on the first argument.

`skip_if_offline()` skips tests if an internet connection is not available using `curl::nslookup()`.

`skip_on_cran()` skips tests on CRAN, using the `NOT_CRAN` environment variable set by devtools.

`skip_on_travis()` skips tests on travis by inspecting the `TRAVIS` environment variable.

`skip_on_appveyor()` skips tests on appveyor by inspecting the `APPVEYOR` environment variable.

#' `skip_on_ci()` skips tests on continuous integration systems by inspecting the CI environment variable.

`skip_on_bioc()` skips tests on Bioconductor by inspecting the `BBS_HOME` environment variable.

`skip_if_not_installed()` skips a tests if a package is not installed or cannot be loaded (useful for suggested packages). It loads the package as a side effect, because the package is likely to be used anyway.

## Examples

```
if (FALSE) skip("No internet connection")

## The following are only meaningful when put in test files and
## run with `test_file`, `test_dir`, `test_check`, etc.

test_that("skip example", {
  expect_equal(1, 1L) # this expectation runs
  skip('skip')
  expect_equal(1, 2) # this one skipped
  expect_equal(1, 3) # this one is also skipped
})
```

---

StopReporter

*Test reporter: stop on error.*

---

## Description

The default reporter, executed when `expect_that` is run interactively. It responds by `stop()`ping on failures and doing nothing otherwise. This will ensure that a failing test will raise an error.

## Usage

StopReporter

**Format**

An object of class R6ClassGenerator of length 24.

**Details**

This should be used when doing a quick and dirty test, or during the final automated testing of R CMD check. Otherwise, use a reporter that runs all tests and gives you more context about the problem.

**See Also**

Other reporters: [CheckReporter](#), [DebugReporter](#), [FailReporter](#), [ListReporter](#), [LocationReporter](#), [MinimalReporter](#), [MultiReporter](#), [ProgressReporter](#), [Reporter](#), [RstudioReporter](#), [SilentReporter](#), [SummaryReporter](#), [TapReporter](#), [TeamcityReporter](#)

---

SummaryReporter	<i>Test reporter: summary of errors.</i>
-----------------	--

---

**Description**

This is a reporter designed for interactive usage: it lets you know which tests have run successfully and as well as fully reporting information about failures and errors.

**Usage**

SummaryReporter

**Format**

An object of class R6ClassGenerator of length 24.

**Details**

You can use the `max_reports` field to control the maximum number of detailed reports produced by this reporter. This is useful when running with `auto_test()`

As an additional benefit, this reporter will praise you from time-to-time if all your tests pass.

**See Also**

Other reporters: [CheckReporter](#), [DebugReporter](#), [FailReporter](#), [ListReporter](#), [LocationReporter](#), [MinimalReporter](#), [MultiReporter](#), [ProgressReporter](#), [Reporter](#), [RstudioReporter](#), [SilentReporter](#), [StopReporter](#), [TapReporter](#), [TeamcityReporter](#)

---

TapReporter	<i>Test reporter: TAP format.</i>
-------------	-----------------------------------

---

**Description**

This reporter will output results in the Test Anything Protocol (TAP), a simple text-based interface between testing modules in a test harness. For more information about TAP, see <http://testanything.org>

**Usage**

TapReporter

**Format**

An object of class R6ClassGenerator of length 24.

**See Also**

Other reporters: [CheckReporter](#), [DebugReporter](#), [FailReporter](#), [ListReporter](#), [LocationReporter](#), [MinimalReporter](#), [MultiReporter](#), [ProgressReporter](#), [Reporter](#), [RstudioReporter](#), [SilentReporter](#), [StopReporter](#), [SummaryReporter](#), [TeamcityReporter](#)

---

TeamcityReporter	<i>Test reporter: Teamcity format.</i>
------------------	--

---

**Description**

This reporter will output results in the Teamcity message format. For more information about Teamcity messages, see <http://confluence.jetbrains.com/display/TCD7/Build+Script+Interaction+with+TeamCity>

**Usage**

TeamcityReporter

**Format**

An object of class R6ClassGenerator of length 24.

**See Also**

Other reporters: [CheckReporter](#), [DebugReporter](#), [FailReporter](#), [ListReporter](#), [LocationReporter](#), [MinimalReporter](#), [MultiReporter](#), [ProgressReporter](#), [Reporter](#), [RstudioReporter](#), [SilentReporter](#), [StopReporter](#), [SummaryReporter](#), [TapReporter](#)

---

teardown	<i>Run code on setup/teardown</i>
----------	-----------------------------------

---

### Description

Code in a `setup()` block is run immediately in a clean environment. Code in a `teardown()` block is run upon completion of a test file, even if it exits with an error. Multiple calls to `teardown()` will be executed in the order they were created.

### Usage

```
teardown(code, env = parent.frame())
```

```
setup(code, env = parent.frame())
```

### Arguments

`code`            Code to evaluate

`env`             Environment in which code will be evaluated. For expert use only.

### Examples

```
## Not run:

tmp <- tempfile()
setup(writeLines(tmp, "some test data"))
teardown(unlink(tmp))

## End(Not run)
```

---

test_dir	<i>Run all tests in directory or package</i>
----------	--

---

### Description

Use `test_dir()` for a collection of tests in a directory; use `test_package()` interactively at the console, and `test_check()` inside of R CMD check.

In your own code, you can use `is_testing()` to determine if code is being run as part of a test and `testing_package()` to retrieve the name of the package being tested. You can also check the underlying env var directly `identical(Sys.getenv("TESTTHAT"), "true")` to avoid creating a run-time dependency on `testthat`.



**Usage**

```
test_dir(path, filter = NULL, reporter = default_reporter(),
  env = test_env(), ..., encoding = "unknown", load_helpers = TRUE,
  stop_on_failure = FALSE, stop_on_warning = FALSE, wrap = TRUE)

test_package(package, filter = NULL, reporter = check_reporter(), ...,
  stop_on_failure = TRUE, stop_on_warning = FALSE)

test_check(package, filter = NULL, reporter = check_reporter(), ...,
  stop_on_failure = TRUE, stop_on_warning = FALSE, wrap = TRUE)

is_testing()

testing_package()
```

**Arguments**

path	Path to directory containing tests.
filter	If not NULL, only tests with file names matching this regular expression will be executed. Matching be performed on the file name after it has been stripped of "test-" and ".R".
reporter	Reporter to use to summarise output. Can be supplied as a string (e.g. "summary") or as an R6 object (e.g. <code>SummaryReporter\$new()</code> ). See <a href="#">Reporter</a> for more details and a list of built-in reporters.
env	Environment in which to execute the tests. Expert use only.
...	Additional arguments passed to <code>grepl()</code> to control filtering.
encoding	Deprecated. All files now assumed to be UTF-8.
load_helpers	Source helper files before running the tests? See <code>source_test_helpers()</code> for more details.
stop_on_failure	If TRUE, throw an error if any tests fail.
stop_on_warning	If TRUE, throw an error if any tests generate warnings.
wrap	Automatically wrap all code within <code>test_that()</code> ? This ensures that all expectations are reported, even if outside a test block.
package	Name of installed package.

**Value**

A list of test results.

**Test files**

For package code, tests should live in `tests/testthat`.

There are four classes of `.R` files that have special behaviour:

- Test files start with `test` and are executed in alphabetical order.
- Helper files start with `helper` and are executed before tests are run and from `devtools::load_all()`.
- Setup files start with `setup` and are executed before tests, but not during `devtools::load_all()`.
- Teardown files start with `teardown` and are executed after the tests are run.

## Environments

Each test is run in a clean environment to keep tests as isolated as possible. For package tests, that environment that inherits from the package's namespace environment, so that tests can access internal functions and objects.

## R CMD check

To run `testthat` automatically from R CMD check, make sure you have a `tests/testthat.R` that contains:

```
library(testthat)
library(yourpackage)

test_check("yourpackage")
```

## Examples

```
test_dir(testthat_examples(), reporter = "summary")
test_dir(testthat_examples(), reporter = "minimal")
```

---

test_examples	<i>Test package examples</i>
---------------	------------------------------

---

## Description

These helper functions make it easier to test the examples in a package. Each example counts as one test, and it succeeds if the code runs without an error.

## Usage

```
test_examples(path = "../..")

test_example(path)

test_rd(rd)
```

## Arguments

path	For <code>test_examples()</code> , path to directory containing Rd files. For <code>test_example()</code> , path to a single Rd file. Remember the working directory for tests is <code>tests/testthat</code> .
rd	A parsed Rd object, obtained from <code>tools::Rd_db()</code> or otherwise.

---

test_file	<i>Run all tests in specified file</i>
-----------	--

---

### Description

Execute code in the specified file, displaying results using a reporter. Use this function when you want to run a single file's worth of tests. You are responsible for ensuring that the functions to test are available in the global environment.

### Usage

```
test_file(path, reporter = default_reporter(), env = test_env(),
  start_end_reporter = TRUE, load_helpers = TRUE,
  encoding = "unknown", wrap = TRUE)
```

### Arguments

path	Path to file.
reporter	Reporter to use to summarise output. Can be supplied as a string (e.g. "summary") or as an R6 object (e.g. <code>SummaryReporter\$new()</code> ). See <a href="#">Reporter</a> for more details and a list of built-in reporters.
env	Environment in which to execute the tests. Expert use only.
start_end_reporter	Should the reporters <code>start_reporter()</code> and <code>end_reporter()</code> methods be called? For expert use only.
load_helpers	Source helper files before running the tests? See <a href="#">source_test_helpers()</a> for more details.
encoding	Deprecated. All files now assumed to be UTF-8.
wrap	Automatically wrap all code within <a href="#">test_that()</a> ? This ensures that all expectations are reported, even if outside a test block.

### Value

Invisibly, a list with one element for each test.

### Examples

```
path <- testthat_example("success")
test_file(path, reporter = "minimal")

# test_file() invisibly returns a list, with one element for each test.
# This can be useful if you want to compute on your test results.
out <- test_file(path, reporter = "minimal")
str(out[[1]])
```

---

test_path	<i>Locate file in testing directory.</i>
-----------	--

---

**Description**

This function is designed to work both interactively and during tests, locating files in the tests/testthat directory

**Usage**

```
test_path(...)
```

**Arguments**

... Character vectors giving path component.

**Value**

A character vector giving the path.

---

test_that	<i>Create a test.</i>
-----------	-----------------------

---

**Description**

A test encapsulates a series of expectations about small, self-contained set of functionality. Each test is contained in a [context](#) and contains multiple expectations.

**Usage**

```
test_that(desc, code)
```

**Arguments**

desc	test name. Names should be kept as brief as possible, as they are often used as line prefixes.
code	test code containing expectations

**Details**

Tests are evaluated in their own environments, and should not affect global state.

When run from the command line, tests return NULL if all expectations are met, otherwise it raises an error.

**Examples**

```

test_that("trigonometric functions match identities", {
  expect_equal(sin(pi / 4), 1 / sqrt(2))
  expect_equal(cos(pi / 4), 1 / sqrt(2))
  expect_equal(tan(pi / 4), 1)
})
# Failing test:
## Not run:
test_that("trigonometric functions match identities", {
  expect_equal(sin(pi / 4), 1)
})

## End(Not run)

```

---

use\_catch

*Use Catch for C++ Unit Testing*


---

**Description**

Add the necessary infrastructure to enable C++ unit testing in R packages with **Catch** and `testthat`.

**Usage**

```
use_catch(dir = getwd())
```

**Arguments**

`dir`                    The directory containing an R package.

**Details**

Calling `use_catch()` will:

1. Create a file `src/test-runner.cpp`, which ensures that the `testthat` package will understand how to run your package's unit tests,
2. Create an example test file `src/test-example.cpp`, which showcases how you might use **Catch** to write a unit test,
3. Add a test file `tests/testthat/test-cpp.R`, which ensures that `testthat` will run your compiled tests during invocations of `devtools::test()` or `R CMD check`, and
4. Create a file `R/catch-routine-registration.R`, which ensures that R will automatically register this routine when `tools::package_native_routine_registration_skeleton()` is invoked.

C++ unit tests can be added to C++ source files within the `src` directory of your package, with a format similar to R code tested with `testthat`. Here's a simple example of a unit test written with `testthat` + **Catch**:

```
context("C++ Unit Test") {
  test_that("two plus two is four") {
    int result = 2 + 2;
    expect_true(result == 4);
  }
}
```

When your package is compiled, unit tests alongside a harness for running these tests will be compiled into your R package, with the C entry point `run_testthat_tests()`. `testthat` will use that entry point to run your unit tests when detected.

## Functions

All of the functions provided by Catch are available with the `CATCH_` prefix – see [here](#) for a full list. `testthat` provides the following wrappers, to conform with `testthat`'s R interface:

Function	Catch	Description
<code>context</code>	<code>CATCH_TEST_CASE</code>	The context of a set of tests.
<code>test_that</code>	<code>CATCH_SECTION</code>	A test section.
<code>expect_true</code>	<code>CATCH_CHECK</code>	Test that an expression evaluates to true.
<code>expect_false</code>	<code>CATCH_CHECK_FALSE</code>	Test that an expression evaluates to false.
<code>expect_error</code>	<code>CATCH_CHECK_THROWS</code>	Test that evaluation of an expression throws an exception.
<code>expect_error_as</code>	<code>CATCH_CHECK_THROWS_AS</code>	Test that evaluation of an expression throws an exception of a specific class.

In general, you should prefer using the `testthat` wrappers, as `testthat` also does some work to ensure that any unit tests within will not be compiled or run when using the Solaris Studio compilers (as these are currently unsupported by Catch). This should make it easier to submit packages to CRAN that use Catch.

## Symbol Registration

If you've opted to disable dynamic symbol lookup in your package, then you'll need to explicitly export a symbol in your package that `testthat` can use to run your unit tests. `testthat` will look for a routine with one of the names:

```
C_run_testthat_tests
c_run_testthat_tests
run_testthat_tests
```

See [Controlling Visibility](#) and [Registering Symbols](#) in the **Writing R Extensions** manual for more information.

## Advanced Usage

If you'd like to write your own Catch test runner, you can instead use the `testthat::catchSession()` object in a file with the form:

```
#define TESTTHAT_TEST_RUNNER
#include <testthat.h>

void run()
{
    Catch::Session& session = testthat::catchSession();
    // interact with the session object as desired
}
```

This can be useful if you'd like to run your unit tests with custom arguments passed to the Catch session.

### Standalone Usage

If you'd like to use the C++ unit testing facilities provided by Catch, but would prefer not to use the regular testthat R testing infrastructure, you can manually run the unit tests by inserting a call to:

```
.Call("run_testthat_tests", PACKAGE = <pkgName>)
```

as necessary within your unit test suite.

### See Also

[Catch](#), the library used to enable C++ unit testing.

# Index

## \*Topic **datasets**

- CheckReporter, 4
- DebugReporter, 6
- FailReporter, 18
- JunitReporter, 20
- ListReporter, 20
- LocationReporter, 31
- MinimalReporter, 22
- MultiReporter, 23
- ProgressReporter, 26
- RstudioReporter, 27
- SilentReporter, 27
- StopReporter, 29
- SummaryReporter, 30
- TapReporter, 31
- TeamcityReporter, 31

## \*Topic **debugging**

- auto\_test, 3
- auto\_test\_package, 4

all.equal(), 8

auto\_test, 3

auto\_test(), 4, 30

auto\_test\_package, 4

auto\_test\_package(), 4

CheckReporter, 4, 6, 18, 21, 23, 26, 27, 30, 31

compare(), 5, 8, 11

comparison-expectations, 5

context, 36

curl::nslookup(), 29

DebugReporter, 4, 6, 18, 21, 23, 26, 27, 30, 31

describe, 6

equality-expectations, 7

expect, 9

expect\_condition (output-expectations),  
23

expect\_cpp\_tests\_pass, 10

expect\_equal (equality-expectations), 7

expect\_equal\_to\_reference  
(expect\_known\_output), 11

expect\_equivalent  
(equality-expectations), 7

expect\_error (output-expectations), 23

expect\_false (logical-expectations), 21

expect\_gt (comparison-expectations), 5

expect\_gte (comparison-expectations), 5

expect\_identical  
(equality-expectations), 7

expect\_invisible, 10

expect\_known\_hash  
(expect\_known\_output), 11

expect\_known\_output, 11

expect\_known\_value  
(expect\_known\_output), 11

expect\_length, 5, 8, 12, 14–16, 19, 22, 25

expect\_less\_than  
(comparison-expectations), 5

expect\_lt (comparison-expectations), 5

expect\_lte (comparison-expectations), 5

expect\_mapequal (expect\_setequal), 16

expect\_match, 5, 8, 13, 13, 15, 16, 19, 22, 25

expect\_message (output-expectations), 23

expect\_more\_than  
(comparison-expectations), 5

expect\_named, 5, 8, 13, 14, 14, 16, 19, 22, 25

expect\_null, 5, 8, 13–15, 15, 19, 22, 25

expect\_output (output-expectations), 23

expect\_output\_file  
(expect\_known\_output), 11

expect\_reference  
(equality-expectations), 7

expect\_s3\_class  
(inheritance-expectations), 19

expect\_s4\_class  
(inheritance-expectations), 19

expect\_setequal, 16



- expect\_silent (output-expectations), 23
- expect\_true (logical-expectations), 21
- expect\_type (inheritance-expectations), 19
- expect\_vector, 17
- expect\_visible (expect\_invisible), 10
- expect\_warning (output-expectations), 23
- fail, 18
- FailReporter, 4, 6, 18, 21, 23, 26, 27, 30, 31
- grepl(), 14, 33
- has\_names(), 15
- identical(), 5, 8, 11
- inheritance-expectations, 19
- inherits(), 19
- is(), 19
- is\_false(), 22
- is\_testing (test\_dir), 32
- JUnitReporter, 20
- ListReporter, 4, 6, 18, 20, 21, 23, 26, 27, 30, 31
- LocationReporter, 4, 6, 18, 21, 21, 23, 26, 27, 30, 31
- logical-expectations, 21
- MinimalReporter, 4, 6, 18, 21, 22, 23, 26, 27, 30, 31
- MultiReporter, 4, 6, 18, 21, 23, 23, 26, 27, 30, 31
- output-expectations, 23
- ProgressReporter, 4, 6, 18, 21, 23, 26, 27, 30, 31
- quasi\_label, 5, 8, 11, 13–17, 19, 22, 24
- recover(), 6
- Reporter, 4, 6, 18, 21, 23, 26, 27, 30, 31, 33, 35
- RstudioReporter, 4, 6, 18, 21, 23, 26, 27, 27, 30, 31
- setup (teardown), 32
- SilentReporter, 4, 6, 18, 21, 23, 26, 27, 27, 30, 31
- skip, 28
- skip\_if (skip), 28
- skip\_if\_not (skip), 28
- skip\_if\_not\_installed (skip), 28
- skip\_if\_offline (skip), 28
- skip\_if\_translated (skip), 28
- skip\_on\_appveyor (skip), 28
- skip\_on\_bioc (skip), 28
- skip\_on\_ci (skip), 28
- skip\_on\_cran (skip), 28
- skip\_on\_os (skip), 28
- skip\_on\_travis (skip), 28
- source\_test\_helpers(), 33, 35
- stop(), 29
- stopifnot(), 29
- StopReporter, 4, 6, 18, 21, 23, 26, 27, 29, 30, 31
- succeed (fail), 18
- SummaryReporter, 4, 6, 18, 21, 23, 26, 27, 30, 30, 31
- TapReporter, 4, 6, 18, 21, 23, 26, 27, 30, 31, 31
- TeamcityReporter, 4, 6, 18, 21, 23, 26, 27, 30, 31, 31
- teardown, 32
- test\_check (test\_dir), 32
- test\_dir, 32
- test\_dir(), 26
- test\_example (test\_examples), 34
- test\_examples, 34
- test\_file, 35
- test\_file(), 26
- test\_package (test\_dir), 32
- test\_path, 36
- test\_rd (test\_examples), 34
- test\_that, 36
- test\_that(), 7, 29, 33, 35
- testing\_package (test\_dir), 32
- tools::Rd\_db(), 34
- typeof(), 19
- use\_catch, 37
- use\_catch(), 10
- vctrs::vec\_assert(), 17
- watch(), 3, 4