Package ‘testthat’

May 22, 2015

Version 0.10.0

Title Unit Testing for R

Description A unit testing system designed to be fun, flexible and easy to set up.

URL https://github.com/hadley/testthat

BugReports https://github.com/hadley/testthat/issues

Depends R (>= 3.1.0), methods

Imports digest, crayon

Suggests devtools

License MIT + file LICENSE

Collate 'auto-test.r' 'colour-text.r' 'compare.r' 'context.r'
   'describe.r' 'evaluate-promise.r' 'expect-that.r'
   'expectation.r' 'expectations-equality.R'
   'expectations-matches.R' 'expectations-old.R' 'expectations.r'
   'make-expectation.r' 'mock.r' 'reporter.r' 'reporter-check.R'
   'reporter-list.r' 'reporter-minimal.r' 'reporter-multi.r'
   'reporter-rstudio.R' 'reporter-silent.r' 'reporter-stop.r'
   'reporter-summary.r' 'reporter-tap.r' 'reporter-teamcity.r'
   'reporter-xxx.r' 'test-example.R' 'test-files.r'
   'test-package.r' 'test-results.r' 'test-that.r' 'traceback.r'
   'utils.r' 'watcher.r'

NeedsCompilation yes

Author Hadley Wickham [aut, cre],
RStudio [cph]

Maintainer Hadley Wickham <hadley@rstudio.com>

Repository CRAN

Date/Publication 2015-05-22 22:39:27
R topics documented:

auto_test ................................................. 3
auto_test_package ....................................... 4
CheckReporter-class ...................................... 4
compare ................................................... 5
context ................................................... 6
describe .................................................. 6
equivalence ............................................... 8
evaluate_promise ......................................... 9
expect-compare ........................................... 10
expect_equal_to_reference ............................... 11
expect_is .................................................. 12
expect_named ............................................. 13
expect_null ............................................... 14
expect_true ............................................... 15
fail ....................................................... 16
ListReporter-class ....................................... 16
make_expectation ......................................... 17
matching-expectations ................................. 17
MinimalReporter-class ................................... 19
MultiReporter-class ...................................... 19
not ......................................................... 20
Reporter-class ........................................... 20
RstudioReporter-class ................................... 20
setup_test_dir ........................................... 21
SilentReporter-class ..................................... 21
skip ....................................................... 22
StopReporter-class ...................................... 22
succeed .................................................... 23
SummaryReporter-class ................................. 23
TapReporter-class ....................................... 24
TeammcityReporter-class ............................... 24
testthat ................................................... 24
testthat_results .......................................... 25
test_dir .................................................... 26
test_examples ............................................ 26
test_file ................................................... 27
test_package ............................................. 27
test_that .................................................. 28
watch ...................................................... 29
with_mock ................................................ 29

Index ...................................................... 31
auto_test

| auto_test | Watches code and tests for changes, rerunning tests as appropriate. |

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 = "summary", env = test_env())

Arguments

code_path  path to directory containing code

test_path  path to directory containing tests

reporter  test reporter to use

e   environment in which to execute test suite.

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  Watches a package for changes, rerunning tests as appropriate.

Description
Watches a package for changes, rerunning tests as appropriate.

Usage
auto_test_package(pkg = ".", reporter = "summary")

Arguments
pkg  path to package
reporter  test reporter to use

See Also
  auto_test for details on how method works

CheckReporter-class  Check reporter: 13 line summary of problems

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.

Arguments
  ...  Arguments used to initialise class
**Description**

`compare` is similar to `all.equal()`, but shows you examples of where the failures occurred.

**Usage**

```r
compare(x, y, ...)  
## Default S3 method:  
compare(x, y, ...)  
## S3 method for class 'character'  
compare(x, y, ..., max_diffs = 5, max_lines = 5,  
       width = getOption("width"))  
## S3 method for class 'numeric'  
compare(x, y, max_diffs = 10, ...)  
```

**Arguments**

- `x`, `y`  
  Objects to compare
- `...`  
  Additional arguments used to control specifics of comparison
- `max_diffs`  
  Maximum number of differences to show
- `max_lines`  
  Maximum number of lines to show from each difference
- `width`  
  Width of output device

**Examples**

```r
# Character
x <- c("abc", "def", "jih")
compare(x, x)

y <- paste0(x, "y")
compare(x, y)

compare(letters, paste0(letters, "-"))

x <- "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis cursus tincidunt auctor. Vestibulum ac metus bibendum, facilisis nisi non, pulvinar dolor. Donec pretium iaculis nulla, ut interdum sapien ultricies a."
y <- "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis cursus tincidunt auctor. Vestibulum ac metus1 bibendum, facilisis nisi non, pulvinar dolor. Donec pretium iaculis nulla, ut interdum sapien ultricies a."
compare(x, y)
```
describe(c(x, x), c(y, y))
# Numeric -----------------------------------------------

x <- y <- runif(100)
y[sample(100, 10)] <- 5
compare(x, y)

x <- y <- 1:10
x[5] <- NA
x[6] <- 6.5
compare(x, y)

# Compare ignores minor numeric differences in the same way
# as all.equal.
compare(x, x + 1e-9)

context

Describe the context of a set of tests.

Description
A context defines a set of tests that test related functionality. Usually you will have one context per file, but you may have multiple contexts in a single file if you so choose.

Usage
context(desc)

Arguments
desc description of context. Should start with a capital letter.

Examples
context("String processing")
context("Remote procedure calls")

describe describe: a BDD testing language

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.
describe

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
  })
})
equivalence

**Expectation: is the object equal to a value?**

**Description**
- `expect_identical` tests with `identical`
- `expect_equal` tests with `all.equal`
- `expect_equivalent` tests with `all.equal` and `check.attributes = FALSE`

**Usage**
```r
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)
```

**Arguments**
- `object` object to test
- `expected` Expected value
- `...` other values passed to `all.equal`
- `info` extra information to be included in the message (useful when writing tests in loops).
- `label` For full form, label of expected object used in error messages. Useful to override default (deparsed expected expression) when doing tests in a loop. For short cut form, object label. When NULL, computed from deparsed object.
- `expected.label` Equivalent of `label` for shortcut form.

**See Also**
Other expectations: `expect-compare, expect_less_than, expect_more_than; expect_equal_to_reference; expect_error, expect_match, expect_message, expect_output, expect_warning, matching-expectations; expect_false, expect_true; expect_is; expect_named; expect_null; takes_less_than`

**Examples**
```r
a <- 10
equal(a, 10)

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

Evaluate a promise, capturing all types of output.

Description

This uses `evaluate` a promise, returning the result, test, messages and warnings that the code creates in a list. It is used to evaluate code for all test that tests, ensuring that (as much as possible) any spurious output is suppressed during the testing process.

Usage

evaluate_promise(code, print = FALSE)

Arguments

code Code to evaluate. This should be an unevaluated expression.
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.
Value

A list containing

- **result**: The result of the function
- **output**: A string containing all the output from the function
- **warnings**: A character vector containing the text from each warning
- **messages**: A character vector containing the text from each message

Examples

```r
evaluate_promise({
  print("1")
  message("2")
  warning("3")
  4
})
```

---

**expect-compare**

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

Description

This is useful for ensuring returned value is below a ceiling or above a floor.

Usage

```r
expect_less_than(object, expected, ..., info = NULL, label = NULL,
  expected.label = NULL)
```

```r
expect_more_than(object, expected, ..., info = NULL, label = NULL,
  expected.label = NULL)
```

Arguments

- **object**: object to test
- **expected**: Expected value
- **...**: other values passed to `all.equal`
- **info**: extra information to be included in the message (useful when writing tests in loops).
- **label**: For full form, label of expected object used in error messages. Useful to override default (deparsed expected expression) when doing tests in a loop. For short cut form, object label. When NULL, computed from deparsed object.
- **expected.label**: Equivalent of label for shortcut form.
expect_equal_to_reference

See Also

Other expectations: equivalence, expect_equal, expect_equivalent, expect_identical; expect_equal_to_reference, expect_error, expect_match, expect_message, expect_output, expect_warning, matching_expectations; expect_false, expect_true; expect_is; expect_named; expect_null; takes_less_than

Examples

```r
a <- 9
expect_less_than(a, 10)

## Not run:
expect_less_than(11, 10)

## End(Not run)

a <- 11
expect_more_than(a, 10)

## Not run:
expect_more_than(9, 10)

## End(Not run)
```

---

**expect_equal_to_reference**

*Expectation: is the object equal to a reference value stored in a file?*

---

Description

This expectation is equivalent to **expect_equal**, except that the expected value is stored in an RDS file instead of being specified literally. This can be helpful when the value is necessarily complex. If the file does not exist then it will be created using the value of the specified object, and subsequent tests will check for consistency against that generated value. The test can be reset by deleting the RDS file.

Usage

```r
expect_equal_to_reference(object, file, ..., info = NULL, label = NULL, expected.label = NULL)
```

Arguments

- **object**: object to test
- **file**: The file name used to store the object. Should have an "rds" extension.
- **...**: other values passed to **expect_equal**
- **info**: extra information to be included in the message (useful when writing tests in loops).
For the full form, a label for the expected object, which is used in error messages. Useful to override the default (which is based on the file name), when doing tests in a loop. For the short-cut form, the object label, which is computed from the deparsed object by default.

expected.label Equivalent of label for shortcut form.

**Details**

It is important to initialize the reference RDS file within the source package, most likely in the tests/testthat/ directory. Testing spawned by devtools::test(), for example, will accomplish this. But note that testing spawned by R CMD check and devtools::check() will NOT. In the latter cases, the package source is copied to an external location before tests are run. The resulting RDS file will not make its way back into the package source and will not be available for subsequent comparisons.

**See Also**

Other expectations: equivalence, expect_equal, expect_equivalent, expect_identical; expect_compare, expect_less_than, expect_more_than; expect_error, expect_match, expect_message, expect_output, expect_warning, matching_expectations; expect_false, expect_true; expect_is; expect_named;
expect_null; takes_less_than

**Examples**

```r
## Not run:
expect_equal_to_reference(1, "one.rds")
## End(Not run)
```

---

**Description**

Tests whether or not an object inherits from any of a list of classes.

**Usage**

```r
expect_is(object, class, info = NULL, label = NULL)
```

**Arguments**

- `object`: object to test
- `class`: character vector of class names
- `info`: extra information to be included in the message (useful when writing tests in loops).
- `label`: object label. When NULL, computed from deparsed object.
expect_named

See Also

inherits

Other expectations: equivalence, expect_equal, expect_equivalent, expect_identical; expect_compare, expect_less_than, expect_more_than; expect_equal_to_reference; expect_error, expect_match, expect_message, expect_output, expect_warning, matching_expectations; expect_false, expect_true; expect_named; expect_null; takes_less_than

Examples

```r
expect_is(1L, "numeric")
a <- matrix(1:10L, nrow = 5)
expect_is(a, "matrix")

expect_is(mtcars, "data.frame")
# alternatively for classes that have an is method
expect_true(is.data.frame(mtcars))
```

---

**expect_named**  
**Expectation: does object have names?**

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

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

Arguments

- **object**  
  object to test

- **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 (useful when writing tests in loops).

- **label**  
  object label. When NULL, computed from deparsed object.

- **...**  
  Other arguments passed onto has_names.
**See Also**

Other expectations: equivalence, expect_equal, expect_equivalent, expect_identical; expect_compare, expect_less_than, expect_more_than; expect_equal_to_reference; expect_error, expect_match, expect_message, expect_output, expect_warning, matching_expectations; expect_false, expect_true; expect_is; expect_null; takes_less_than

**Examples**

```r
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**

*Expectation: is the object NULL?*

**Description**

Expectation: is the object NULL?

**Usage**

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

**Arguments**

- **object**: object to test
- **info**: extra information to be included in the message (useful when writing tests in loops).
- **label**: object label. When NULL, computed from deparsed object.

**See Also**

Other expectations: equivalence, expect_equal, expect_equivalent, expect_identical; expect_compare, expect_less_than, expect_more_than; expect_equal_to_reference; expect_error, expect_match, expect_message, expect_output, expect_warning, matching_expectations; expect_false, expect_true; expect_is; expect_named; takes_less_than

**Examples**

```r
expect_null(NULL)
```
expect_true

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
info extra information to be included in the message (useful when writing tests in loops).
label object label. When NULL, computed from deparsed object.

Details
Attributes are ignored.

See Also

is_false for complement

Other expectations: equivalence, expect_equal, expect_equivalent, expect_identical; expect_compare, expect_less_than, expect_more_than; expect_equal_to_reference; expect_error, expect_match, expect_message, expect_output, expect_warning, matching_expectations; expect_is, expect_named; expect_null; takes_less_than

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)
fail

A default expectation that always fails.

Description

The fail function forces a test to fail. This is useful if you want to test a pre-condition.

Usage

fail(message = "Failure has been forced")

Arguments

message a string to display.

Examples

```r
## Not run:
test_that("this test fails", fail())

## End(Not run)
```

ListReporter-class

List reporter: gather all test results along with elapsed time and file information.

Description

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

Arguments

... Arguments used to initialise class
make_expectation

Make an equality test.

Description
This a convenience function to make a expectation that checks that input stays the same.

Usage
make_expectation(x, expectation = "equals")

Arguments
x a vector of values
expectation the type of equality you want to test for (equals, is_equivalent_to, is_identical_to)

Examples
x <- 1:10
make_expectation(x)

make_expectation(mtcars$mpg)

df <- data.frame(x = 2)
make_expectation(df)

matching.expectations

Expectation: does string/output/warning/error match a regular expression?

Description
Expectation: does string/output/warning/error match a regular expression?

Usage
expect_match(object, regexp, ..., all = TRUE, info = NULL, label = NULL)
expect_output(object, regexp, ..., info = NULL, label = NULL)
expect_error(object, regexp = NULL, ..., info = NULL, label = NULL)
expect_warning(object, regexp = NULL, ..., all = FALSE, info = NULL, label = NULL)
expect_message(object, regexp = NULL, ..., all = FALSE, info = NULL, label = NULL)
Arguments

object  object to test
regexp  regular expression to test against. If omitted, just asserts that code produces
some output, message, warning or error.
...  Additional arguments passed on to `grepl`, e.g. `ignore.case` or `fixed`.
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 (useful when writing tests in
loops).
lable  object label. When NULL, computed from deparsed object.

See Also

Other expectations: `equivalence`, `expect_equal`, `expect_equivalent`, `expect_identical`; `expect_compare`,
`expect_less_than`, `expect_more_than`; `expect_equal_to_reference`; `expect_false`, `expect_true`;
`expect_is`; `expect_named`; `expect_null`; `takes_less_than`;

Examples

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

```r
# Output
str(mtcars)
extpect_output(str(mtcars), "32 obs")
extpect_output(str(mtcars), "11 variables")
```

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

```r
# Messages
f <- function(x) {
  if (x < 0) message("*x* is already negative")
  -x
}  # Not run: expect_message(f(1))
```  

```r
# 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)
```  

```r
# Warnings
f <- function(x) {
  if (x < 0) warning("*x* is already negative")
  -x
}  # Not run: expect_message(f(1))
```
expect_warning(f(-1))  # Not run: expect_warning(f(1))

# You can use the arguments of grepl to control the matching
expect_warning(f(-1), "xx", 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)

---

**MinimalReporter-class**  
**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.

**Arguments**

...  
Arguments used to initialise class

---

**MultiReporter-class**  
**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.

**Arguments**

...  
Arguments used to initialise class
not

Negate an expectation

Description
This negates an expectation, making it possible to express that you want the opposite of a standard expectation.

Usage
not(f)

Arguments

f an existing expectation function

Examples

x <- 1
expect_that(x, equals(1))
expect_that(x, not(equals(2)))
## Not run:
expect_that(x, equals(2))
expect_that(x, not(equals(1)))

## End(Not run)

Reporter-class
Stub object for managing a reporter of tests.

Description
Do not clone directly from this object - children should implement all methods.

Arguments

... Arguments used to initialise class

RstudioReporter-class
Test reporter: RStudio

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

Arguments

... Arguments used to initialise class
**setup_test_dir**

Take care or finding the test files and sourcing the helpers.

**Description**

Take care or finding the test files and sourcing the helpers.

**Usage**

setup_test_dir(path, filter, env)

**Arguments**

- **path**  path to tests
- **filter** If not NULL, only tests with file names matching this regular expression will be executed. Matching will take on the file name after it has been stripped of "test-" and ".r".
- **env** environment in which to source the helpers

**Value**

the test file paths

---

**SilentReporter-class**  Test reporter: gather all errors silently.

**Description**

This reporter quietly runs all tests, simply gathering the results for later use. This is helpful for programmatically inspecting errors after a test run.

**Arguments**

... Arguments used to initialise class
skip

*Skip a test.*

**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**

```r
skip(message)

skip_if_not_installed(pkg)

skip_on_cran()

skip_on_travis()
```

**Arguments**

- `message`: A message describing why the test was skipped.
- `pkg`: Name of package to check for

**Helpers**

- `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_if_not_installed()` skips tests if a package is not installed (useful for suggested packages).

**Examples**

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

---

**StopReporter-class**

*Test reporter: stop on error.*

**Description**

The default reporter, executed when `expect_that` is run interactively, or when the test files are executed by R CMD check. It responds by `stop()`ing on failures and doing nothing otherwise. This will ensure that a failing test will raise an error.
succeed

**Arguments**

... Arguments used to initialise class

**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.

---

*succeed*  
*A default expectation that always succeeds.*

**Description**

A default expectation that always succeeds.

**Usage**

succeed(message = "Success has been forced")

**Arguments**

message a string to display.

**Examples**

```r
# Not run:
test_that("this test fails", fail())
```

---

*SummaryReporter-class*  
*Test reporter: summary of errors.*

**Description**

This is the most useful reporting reporter as it lets you know both which tests have run successfully, as well as fully reporting information about failures and errors. It is the default reporting reporter used by *test_dir* and *test_file*.

**Arguments**

... Arguments used to initialise class
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.

---

TapReporter-class

Test reporter: TAP format.

---

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

Arguments

... Arguments used to initialise class

---

TeamcityReporter-class

Test reporter: Teamcity format.

---

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

Arguments

... Arguments used to initialise class

---

testthat

R package to make testing fun!

---

Description

Try the example below. Have a look at the references and learn more from function documentation such as `expect_that`.

Details

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

https://github.com/hadley/testthat
http://adv-r.had.co.nz/Testing.html

Examples

library(testthat)
a <- 9
expect_that(a, is_less_than(10))
expect_less_than(a, 10)

---

**testthat_results**

Create a `testthat_results` object from the test results as stored in the ListReporter results field.

Description

Create a `testthat_results` object from the test results as stored in the ListReporter results field.

Usage

`testthat_results(results)`

Arguments

- `results` a list as stored in ListReporter

Value

its list argument as a `testthat_results` object

See Also

ListReporter
test_dir

Run all of the tests in a directory.

Description

Test files start with test and are executed in alphabetical order (but they shouldn’t have dependencies). Helper files start with helper and loaded before any tests are run.

Usage

test_dir(path, filter = NULL, reporter = "summary", env = test_env(), ...)

Arguments

path path to tests
filter If not NULL, only tests with file names matching this regular expression will be executed. Matching will take on the file name after it has been stripped of "test-" and ".r".
reporter reporter to use
env environment in which to execute test suite.
... Additional arguments passed to grep1 to control filtering.

Value

the results as a "testthat_results" (list)

test_examples

Test package examples

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 = "..//..//man")
test_example(path)

Arguments

path For test_examples, path to directory containing Rd files. For test_example, path to a single Rd file. Remember the working directory for tests is tests/testthat.
**test_file**

*Run all tests in specified file.*

**Description**

Run all tests in specified file.

**Usage**

```r
test_file(path, reporter = "summary", env = test_env(), start_end_reporter = TRUE)
```

**Arguments**

- **path**: path to file
- **reporter**: reporter to use
- **env**: environment in which to execute the tests
- **start_end_reporter**: whether to start and end the reporter

**Value**

the results as a "testthat_results" (list)

---

**test_package**

*Run all tests in an installed package*

**Description**

Test are run in an environment that inherits from the package’s namespace environment, so that tests can access non-exported functions and variables. Tests should be placed in either inst/tests, or (better) tests/testthat.

**Usage**

```r
test_package(package, filter = NULL, reporter = "summary", ...)
test_check(package, filter = NULL, reporter = "check", ...)
```

**Arguments**

- **package**: package name
- **filter**: If not NULL, only tests with file names matching this regular expression will be executed. Matching will take on the file name after it has been stripped of "test-" and ".r".
- **reporter**: reporter to use
- ... Additional arguments passed to `grep1` to control filtering.
Value
the results as a "testthat_results" (list)

R CMD check
Use test_package to test an installed package, or in tests/test-all.R if you're using the older inst/tests convention.
If your tests live in tests/testthat (preferred) use test_check in tests/testthat.R. You still use test_package when testing the installed package.

Examples

## Not run: test_package("testthat")

---

test_that Create a test.

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 expectation generated by expect_that.

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

## End(Not run)

---

### watch

**Watch a directory for changes (additions, deletions & modifications).**

#### Description

This is used to power the `auto_test` and `auto_test_package` functions which are used to rerun tests whenever source code changes.

#### Usage

```r
watch(path, callback, pattern = NULL, hash = TRUE)
```

#### Arguments

- **path**: character vector of paths to watch. Omit trailing backslash.
- **callback**: function called everytime a change occurs. It should have three parameters: added, deleted, modified, and should return TRUE to keep watching, or FALSE to stop.
- **pattern**: file pattern passed to `dir`.
- **hash**: hashes are more accurate at detecting changes, but are slower for large files. When FALSE, uses modification time stamps.

#### Details

Use Ctrl + break (windows), Esc (mac gui) or Ctrl + C (command line) to stop the watcher.

---

### with_mock

**Mock functions in a package.**

#### Description

Executes code after temporarily substituting implementations of package functions. This is useful for testing code that relies on functions that are slow, have unintended side effects or access resources that may not be available when testing.

#### Usage

```r
with_mock(..., .env = topenv())
```
with_mock

Arguments

... named parameters redefine mocked functions, unnamed parameters will be evaluated after mocking the functions

.env the environment in which to patch the functions, defaults to the top-level environment. A character is interpreted as package name.

Details

This works by using some C code to temporarily modify the mocked function in place. On exit (regular or error), all functions are restored to their previous state. This is somewhat abusive of R’s internals, and is still experimental, so use with care.

Value

The result of the last unnamed parameter

References

Suraj Gupta (2012): How R Searches And Finds Stuff

Examples

```r
with_mock(
  all.equal = function(x, y, ...) TRUE,
  expect_equal(2 * 3, 4),
  .env = "base"
)
with_mock(
  `base::identical` = function(x, y, ...) TRUE,
  `base::all.equal` = function(x, y, ...) TRUE,
  expect_equal(x <- 3 * 3, 6),
  expect_identical(x + 4, 9)
)
throws_error(expect_equal(3, 5))
throws_error(expect_identical(3, 5))
```
Index

*Topic **debugging**
  auto_test, 3
  auto_test_package, 4
  CheckReporter-class, 4
  ListReporter-class, 16
  MinimalReporter-class, 19
  MultiReporter-class, 19
  RstudioReporter-class, 20
  SilentReporter-class, 21
  StopReporter-class, 22
  SummaryReporter-class, 23
  TapReporter-class, 24
  TeamcityReporter-class, 24

*Topic **internal.**
  Reporter-class, 20

all.equal, 5, 8, 10
auto_test, 3, 4, 24, 29
auto_test_package, 3, 4, 29

CheckReporter (CheckReporter-class), 4
CheckReporter-class, 4
compare, 5
ccontext, 6, 28
describe, 6
dir, 29
equivalence, 8, 11–15, 18
evaluate, 9
evaluate_promise, 9
expect-compare, 10
expect_equal, 11–15, 18
expect_equal (equivalence), 8
expect_equal_to_reference, 8, 11, 13–15, 18
expect_equivalent, 11–15, 18
expect_equivalent (equivalence), 8
expect_error, 8, 11–15
expect_error (matching-expectations), 17
expect_false, 8, 11–14, 18
expect_false (expect_true), 15
expect_identical, 11–15, 18
expect_identical (equivalence), 8
expect_is, 8, 11, 12, 14, 15, 18
expect_less_than, 8, 12–15, 18
expect_less_than (expect-compare), 10
expect_match, 8, 11–15
expect_match (matching-expectations), 17
expect_message, 8, 11–15
expect_message (matching-expectations), 17
expect_more_than, 8, 12–15, 18
expect_more_than (expect-compare), 10
expect_named, 8, 11–13, 14, 15, 18
expect_null, 8, 11–14, 15, 18
expect_output, 8, 11–15
expect_output (matching-expectations), 17
expect_that, 24, 28
expect_true, 8, 11–14, 15, 18
expect_warning, 8, 11–15
expect_warning (matching-expectations), 17
fail, 16
grepl, 18

identical, 8
inherits, 13
is_false, 15

ListReporter (ListReporter-class), 16
ListReporter-class, 16

make_expectation, 17
matching-expectations, 17
MinimalReporter
  (MinimalReporter-class), 19
MinimalReporter-class, 19
MultiReporter (MultiReporter-class), 19
MultiReporter-class, 19

not, 20

Reporter (Reporter-class), 20
Reporter-class, 20
RstudioReporter
  (RstudioReporter-class), 20
RstudioReporter-class, 20

setup_test_dir, 21
SilentReporter (SilentReporter-class), 21
SilentReporter-class, 21
skip, 22
skip_if_not_installed (skip), 22
skip_on_cran (skip), 22
skip_on_travis (skip), 22
stop, 22
StopReporter (StopReporter-class), 22
StopReporter-class, 22
succeed, 23
SummaryReporter
  (SummaryReporter-class), 23
SummaryReporter-class, 23

takes_less_than, 8, 11–15, 18
TapReporter (TapReporter-class), 24
TapReporter-class, 24
TeamcityReporter
  (TeamcityReporter-class), 24
TeamcityReporter-class, 24
test_check (test_package), 27
test_dir, 23, 26
test_example (test_examples), 26
test_examples, 26
test_file, 23, 27
test_package, 27
test_that, 7, 28
testthat, 24
testthat-package (testthat), 24
testthat_results, 25

watch, 29
with_mock, 29