Tinytest by example

Mark van der Loo

December 17, 2020 | Package version 1.2.4

Contents

Introduction

This document provides a number of real-life examples on how **tinytest** is used by other packages. The examples aim to illustrate the purpose of testing functions and serve as a complement to the technical documentation and the 'using `tinytest`' vignette. There is a section for each function. Each section starts with a short example that demonstrates the core purpose of the function. Next, one or more examples from packages that are published on CRAN are shown and explained.

Sometimes a few lines of code were modified or deleted for brevity. This is indicated with comment between square brackets, e.g.

```
## [this is an extra comment, only for this vignette]
```

This document is probably not interesting to read front-to-back. It is more aimed to browse once in a while to get an idea on how **tinytest** can be used in practice.

Package authors are invited to contribute new use cases so new users can learn from them. Please contact the author of this package either by email or via the github repository.

1 expect_equal

R objects are described by the data they contain and the attributes attached to them. For example, in the vector `c(x=1,y=2)`, the data consist of the numbers 1 and 2 (in that order) and there is a single attribute called `names`, consisting of the two strings `"x"` and `"y"` (in that order).

The `expect_equal` function tests whether both the data and the attributes of two objects are the same.

```
R>  expect_equal(1,1)
----- PASSED      : <-->
 call| expect_equal(1, 1)
R>  expect_equal(1, c(x=1))
----- FAILED[data]: <-->
 call| expect_equal(1, c(x = 1))
 diff| Expected 1, got 1
```

Numbers do not have to be exactly the same to be equal (by default).

```
R>  0.9-0.7-0.2
[1] 5.551115e-17
R>  expect_equal(0.9-0.7-0.2,0)
----- PASSED      : <-->
 call| expect_equal(0.9 - 0.7 - 0.2, 0)
R>  expect_equal(0.9-0.7-0.2,0, tolerance=0)
----- FAILED[data]: <-->
 call| expect_equal(0.9 - 0.7 - 0.2, 0, tolerance = 0)
 diff| Expected 0, got 5.55111512312578e-17
```

Here is an example from the **stringdist** package. This package implements various methods to determine how different two strings are. In this test, we check one aspect of the 'optimal string alignment' algorithm. In particular, we test if it correctly counts the switch of two adjacent characters as a single operation.

```
expect_equal(stringdist("ab", "ba", method="osa"), 1)
```

The **benchr** package is a package to time R code, and it uses `expect_equal` to extensively check the outputs. Here are a few examples.

```
b <- benchr::benchmark(1 + 1, 2 + 2)
m <- mean(b)
expect_equal(class(m), c("summaryBenchmark", "data.frame"))
expect_equal(dim(m), c(2L, 7L))
expect_equal(names(m), c("expr", "n.eval", "mean", "trimmed", "lw.ci", "up.ci", "relative"))
expect_equal(class(m$expr), "factor")
expect_equal(levels(m$expr), c("1 + 1", "2 + 2"))
expect_true(all(sapply(m[-1], is.numeric)))
```

2 expect_equivalent

This function ignores the attributes when comparing two R objects. Two objects are equivalent when their data are the same.

```
R>  expect_equivalent(1,1)
----- PASSED      : <-->
 call| expect_equivalent(1, 1)
R>  expect_equivalent(1, c(x=1))
----- PASSED      : <-->
 call| expect_equivalent(1, c(x = 1))
```

The **validate** package offers functions to define restrictions on data, and then confront the data with them. The function `values` extracts the boolean results in the form of a matrix with specific row- and column names. In the example below we are only interested in testing whether the *contents* of the matrix is computed correctly.

```
v <- validator(x > 0)
d <- data.frame(x=c(1,-1,NA))
expect_equivalent(values(confront(d,v)), matrix(c(TRUE,FALSE,NA)) )
```

The **anytime** package translates text data into data/time format (`Date` or `POSIXct`). Here, a test is performed to equivalence, to ignore the timezone label that is attached by anytime but not by `as.Date`.

```
refD <- as.Date("2016-01-01")+0:2
expect_equivalent(refD, anydate(20160101L + 0:2))
```

3 expect_identical

This is the most strict test for equality. The best way to think about this is that two objects must be byte-by-byte indistinguishable in order to be identical. The differences can be subtle, as shown below.

```
R>  La <- list(x=1);
R>  Lb <- list(x=1)
R>  expect_identical(La, Lb)
```

```
----- PASSED      : <-->
 call| expect_identical(La, Lb)
```

```
R>  a <- new.env()
R>  a$x <- 1
R>  b <- new.env()
R>  b$x <- 1
R>  expect_identical(a,b)
```

```
----- FAILED[attr]: <-->
 call| expect_identical(a, b)
 diff| Equal environment objects, but with different memory location
```

Here, `La` and `Lb` are indistinguishable from R's point of view. They only differ in their location in memory. The environments `a` and `b` *are* distinguishable since they contain an explicit identifier which make them unique.

```
R>  print(a)
```

```
<environment: 0x56526975d500>
```

```
R>  print(b)
```

```
<environment: 0x5652696e7010>
```

Another difference with `expect_equal` and `expect_equivalent` is that `expect_identical` does not allow any tolerance for numerical differences.

The `stringdistmatrix` function of **stringdist** computes a matrix of string dissimilarity measures between all elements of a character vector. Below, it is tested whether the argument `useNames="none"` and the legacy (deprecated) argument `useName=FALSE`.

```
    a <- c(k1 = "aap",k2="noot")
    expect_identical(stringdistmatrix(a,useNames="none")
                   , stringdistmatrix(a,useNames=FALSE))
```

The **wand** package can retrieve MIME types for files and directories. This means there are many cases to test. In this particular package this is done by creating two lists, one with input and one with expected results. The tests are then performed as follows:

```
  list(
  ## [long list of results removed for brevity]
  ) -> results
  fils <- list.files(system.file("extdat", package="wand"), full.names=TRUE)
  tst <- lapply(fils, get_content_type)
  names(tst) <- basename(fils)
  for(n in names(tst)) expect_identical(results[[n]], tst[[n]])
```

4 expect_null

The result of an operation should be NULL.

```
R>  expect_null(iris$hihi)
----- PASSED      : <-->
 call| expect_null(iris$hihi)
R>  expect_null(iris$Species)
----- FAILED[data]: <-->
 call| expect_null(iris$Species)
 diff| Expected NULL, got 'factor'
```

This function is new in version 0.9.7 and not used in any depending packages yet.

5 expect_true, expect_false

The result of an operation should be precisely `TRUE` or `FALSE`.

```
R>  expect_true(1 == 1)
----- PASSED      : <-->
 call| expect_true(1 == 1)
R>  expect_false(1 == 2)
----- PASSED      : <-->
 call| expect_false(1 == 2)
```

The **anytime** package converts many types of strings to date/time objects (`POSIXct` or `Date`). Here is a part of it's **tinytest** test suite.

```
## Datetime: factor and ordered (#44)
refD <- as.Date("2016-09-01")
expect_true(refD == anydate(as.factor("2016-09-01")))
expect_true(refD == anydate(as.ordered("2016-09-01")))
expect_true(refD == utcdate(as.factor("2016-09-01")))
expect_true(refD == utcdate(as.ordered("2016-09-01")))
```

Note that == used here has subtly different behavior from `all.equal` used by `expect_equal`. In the above case, == does not compare time zone data, which is not added by `as.Date` but is added by `anytime`. This means that for example

```
expect_equal(anydate(as.factor("2016-09-01")), refD)
```

would fail.

The **ulid** package uses `expect_true` to verify the type of a result.

```
x <- ULIDgenerate(20)
expect_true(is.character(x))
```

6 expect_message

Expect that a message is emitted. Optionally you can specify a regular expression that the message must match.

```
R>  expect_message(message("hihi"))
----- PASSED      : <-->
 call| expect_message(message("hihi"))
R>  expect_message(message("hihi"), pattern = "hi")
----- PASSED      : <-->
 call| expect_message(message("hihi"), pattern = "hi")
R>  expect_message(message("hihi"), pattern= "ha")
----- FAILED[xcpt]: <-->
 call| expect_message(message("hihi"), pattern = "ha")
 diff| Found 1 message(s) of class 'message', but not matching 'ha'.
 diff| Showing up to three messages:
 diff| Message 1 of class <simpleMessage, message, condition>:
 diff| 'hihi'
R>  expect_message(print("hihi"))
[1] "hihi"
----- FAILED[xcpt]: <-->
 call| expect_message(print("hihi"))
 diff| No message was emitted
```

7 expect_warning

Expect that a warning is emitted. Optionally you can specify a regular expression that the warning must match.

```
R>  expect_warning(warning("hihi"))
----- PASSED      : <-->
 call| expect_warning(warning("hihi"))
R>  expect_warning(warning("hihi"), pattern = "hi")
----- PASSED      : <-->
 call| expect_warning(warning("hihi"), pattern = "hi")
R>  expect_warning(warning("hihi"), pattern= "ha")
----- FAILED[xcpt]: <-->
 call| expect_warning(warning("hihi"), pattern = "ha")
 diff| Found 1 warnings(s) of class 'warning', but not matching 'ha'.
 diff| Showing up to three warnings:
 diff| Warning 1 of class <simpleWarning, warning, condition>:
 diff| 'hihi'
R>  expect_warning(1+1)
----- FAILED[xcpt]: <-->
 call| expect_warning(1 + 1)
 diff| No warning was emitted
```

8 expect_error

Expect that an error is emitted. Optionally you can specify a regular expression that the error must match.

```
R>  expect_error(stop("hihi"))
----- PASSED      : <-->
 call| expect_error(stop("hihi"))
R>  expect_error(stop("hihi"), pattern = "hi")
----- PASSED      : <-->
 call| expect_error(stop("hihi"), pattern = "hi")
R>  expect_error(stop("hihi"), pattern= "ha")
----- FAILED[xcpt]: <-->
 call| expect_error(stop("hihi"), pattern = "ha")
 diff| The error message:
 diff| 'hihi'
 diff| does not match pattern 'ha'
R>  expect_error(print("hoho"))
[1] "hoho"
----- FAILED[xcpt]: <-->
 call| expect_error(print("hoho"))
 diff| No error
```

The **ChemoSpec2D** package implements exploratory methods for 2D-spectrometry data. Scaled data has negative values, so one cannot take the logarithm. The function `centscaleSpectra2D` must eject an error in such cases and this is tested as follows.

```
  # Check that log and centering cannot be combined
  expect_error(
    centscaleSpectra2D(tiny, center = TRUE, scale = "log"),
    "Cannot take log of centered data")
```

9 expect_silent

Sometimes a test is only run to check that the code does not crash. This function tests that no warnings or errors are emitted when evaluating it's argument.

```
R>  expect_silent(print(10))
```

```
----- PASSED      : <-->
 call| expect_silent(print(10))
```

```
R>  expect_silent(stop("haha"))
```

```
----- FAILED[xcpt]: <-->
 call| expect_silent(stop("haha"))
 diff| Execution was not silent. An error was thrown with message
 diff| 'haha'
```

The **validate** package defines an object called a `validation`, which is the result of confronting a dataset with one or more data quality restrictions in the form of rules. A **validation** object can be plotted, but this would crash with an error in a certain edge case. Here is a test that was added in response to a reported issue.

```
data <- data.frame(A = 1)
rule <- validator(A > 0)
cf <- confront(data, rule)
expect_silent(plot(rule))
expect_silent(plot(cf))
```

The **lumberjack** package creats log files that track changes in data. In one test it is first tested whether a file has been generated, next it is tested whether it can be read properly. This is also an example of programming over test results, since the file is deleted if it exists.

```
run("runs/multiple_loggers.R")
simple_ok <- expect_true(file.exists("runs/simple_log.csv"))
expect_silent(read.csv("runs/simple_log.csv"))
if (simple_ok) unlink("runs/simple_log.csv")
```

10 ignore

Ignore allows you to not record the result of a test. It is not used very often. Its use is probably almost exclusive to **tinytest** where it is used while testing the expectation functions.

The following result is not recorded (note placement of brackets!)

```
ignore(expect_equal)(1+1, 2)
```

The **digest** package computes hashes of R objects. It uses `ignore` in one of it's files.

```
mantissa <- gsub(" [0-9]*$", "", x.hex)
ignore(expect_true)(all(
    sapply(
        head(seq_along(mantissa), -1),
        function(i){
            all(
                grepl(
                    paste0("^", mantissa[i], ".*"),
                    tail(mantissa, -i)
                )
            )
        }
    )
))
```

References

[1] anytime D. Eddelbuettel (2019) *Anything to 'POSIXct' or 'Date' Converter*. R package version 0.3.3.5

[2] benchr Arttem Klevtsov (2019) *High Precise Measurement of R Expressions Execution Time*. R package version 0.2.3-1.

[3] ChemoSpec2D B.A. Hanson (2019) *Exploratory Chemometrics for 2D Spectroscopy* R package version 0.3.166

[4] digest D. Eddelbuettel (2019) *Create Compact Hash Digests of R Objects* R package version 0.6.20

[5] stringdist M. van der Loo (2014). *The stringdist package for approximate string matching*. The R Journal 6(1) 111-122

[6] ulid B. Rudis (2019) *Generate Universally Unique Lexicographically Sortable Identifiers*. R package version 0.3.0

[7] validate M. van der Loo, E. de Jonge and P. Hsieh (2019) *Data Validation Infrastructure for R*. R package version 0.2.7

[8] wand B. Rudis (2019) *Retrieve 'Magic' Attributes from Files and Directories* R package version 0.5.0