Package ‘unittest’

January 10, 2024

Encoding UTF-8

Type Package

Title TAP-Compliant Unit Testing

Version 1.6-1

Date 2024-01-10

Description Concise TAP <`http://testanything.org/`> compliant unit testing package. Authored tests can be run using CMD check with minimal implementation overhead.

License GPL (>= 3)

Depends R (>= 3.6.0)

Imports methods

Suggests knitr, rmarkdown

VignetteBuilder knitr

BugReports `https://github.com/ravingmantis/unittest/issues`

NeedsCompilation no

Author Jamie Lentin [aut, cre],
Anthony Hennessey [aut]

Maintainer Jamie Lentin <`lentinj@shuttlethread.com`>

Repository CRAN

Date/Publication 2024-01-10 12:40:02 UTC

R topics documented:

`unittest-package` *TAP-compliant Unit Testing*

Description

Concise TAP-compliant unit testing package. Authored unit tests can be run using `R CMD check` with minimal implementation overhead.

Details

Given a simple function you'd like to test in the file `myfunction.R`:

```
biggest <- function(x,y) { max(c(x,y)) }
```

A test script for this function `test_myfunction.R` would be:

```
library(unittest)

source('myfunction.R')  # Or library(mypackage) if part of a package

ok(biggest(3,4) == 4, "two numbers")
ok(biggest(c(5,3),c(3,4)) == 5, "two vectors")
```

You can then run this test in several ways:

1. `source('test_myfunction.R')` from R
2. `Rscript --vanilla test_myfunction.R` from the command prompt
3. `R CMD check`, if `test_myfunction.R` is inside the `tests` directory of `mypackage` being tested. 'unittest' doesn't require any further setup in your package

If writing tests as part of a package, see `vignette("testing_packages", package='unittest')`.

The workhorse of the 'unittest' package is the `ok` function which prints "ok" when the expression provided evaluates to `TRUE` and "not ok" if the expression evaluates to anything else or results in an error. There are several `ut_cmp_*` helpers designed to work with `ok`:

1. `ok(ut_cmp_equal( biggest(1/3, 2/6), 2/6), "two floating point numbers")`: Uses `all.equal` to compare within a tolerance
2. `ok(ut_cmp_identical( biggest("c", "d") ), "two strings")`: Uses `identical` to make sure outputs are identical
3. `ok(ut_cmp_error(biggest(3), '"y".*missing'), "single argument is an error")`: Make sure the code produces an error matching the regular expression

In all cases you get detailed, colourised output on what the difference is if the test fails.

Author(s)

Maintainer: Jamie Lentin <lentinj@shuttlethread.com>, Anthony Hennessey <anthony.hennessey@protonmail.com>.

References

Inspired by Perl's Test::Simple (`https://metacpan.org/pod/Test::Simple`).

See Also

testthat, RUnit, svUnit.

`ok` *The unittest package's workhorse function*

Description

Report the test of an expression in TAP format.

Usage

```
ok(test, description)
```

Arguments

`test`	Expression to be tested. Evaluating to `TRUE` is treated as success, anything else as failure.
`description`	Character string describing the test. If a description is not given a character representation of the test expression will be used.

Details

See `unittest` package documentation.

The `unittest.output` option tells unittest where output should be sent. This is most useful for vignettes, where sending output to `stderr` separates the unittest output from the vignette itself.

Value

`ok()` returns whatever was returned when `test` is evaluated. More importantly it has the side effect of printing the result of the test in `TAP` format.

Examples

```
ok(1==1, "1 equals 1")

ok(1==1)

ok(1==2, "1 equals 2")

ok(all.equal(c(1,2),c(1,2)), "compare vectors")

fn <- function () stop("oops")
ok(fn(), "something with a coding error")
ok(c("Some diagnostic", "messages"), "A failure with diagnostic messages")

## Send unittest output to stderr()
options(unittest.output = stderr())
ok(ut_cmp_equal(4, 5), "4 == 5? Probably not")

## Reset unittest output to default (stdout())
options(unittest.output = NULL)
ok(ut_cmp_equal(4, 5), "4 == 5? Probably not")
```

ok_group *Group associated unit tests*

Description

Group associated unit tests with TAP compliant comments separating the output.

Usage

```
ok_group(message, tests)
```

Arguments

`message`	Character vector describing this group. Will be printed as a comment before the tests are ran.
`tests`	A code block full of tests.

Details

Used to group a selection of tests together, for instance you may group the tests relating to a function together.

Value

Returns NULL.

Examples

```
ok_group("Test addition", {
    ok(1 + 1 == 2, "Can add 1")
    ok(1 + 3 == 4, "Can add 3")
})

ok_group("Test subtraction", {
    ok(1 - 1 == 0, "Can subtract 1")
    ok(1 - 3 == -2, "Can subtract 3")
})

# Multiline group message
ok_group(c("Test multiplication", "but not division"),{
    ok(1 * 1 == 1, "Can multiply by 1")
    ok(2 * 3 == 6, "Can multiply by 3")
})
```

ut_cmp *Compare variables with verbose error output*

Description

A wrapper for `all.equal` and `identical` that provides more useful diagnostics when used in a unittest `ok` function.

Usage

```
ut_cmp_equal(a, b, filter = NULL, deparse_frame = -1, ...)
ut_cmp_identical(a, b, filter = NULL, deparse_frame = -1)
```

Arguments

`a`	First item to compare, usually the result of whatever you are testing
`b`	Second item to compare, usually the expected output of whatever you are testing
`filter`	An optional filter function, that turns either a or b into text, and prints this out
`deparse_frame`	Tell `sys.call` which frame to deparse to get original expressions. Set to `-2` when making a helper function, see examples.
`...`	Other arguments passed directly to `all.equal`

Details

For both functions, `a` and `b` are first passed to `all.equal` (for `ut_cmp_equal()`) or `identical` (for `ut_cmp_identical()`). If they match, then the function returns TRUE and your test passes.

If this fails, then we turn both `a` and `b` into text, and then use `git diff` to compare the 2 outputs. If you do not have git installed, then the 2 outputs will be shown side-by-side.

When using `git diff`, we turn colored output on when outputting to a terminal. You can force this on or off using `options("cli.num_colors" = 1)` or the `NO_COLOR` or `R_CLI_NUM_COLORS` environment variable.

The step of turning into text is done with the filter function. There are several of these built-in, and it will choose the one that produces the simplest output. This may mean that the output will be from the `print` function if the differences are obvious, or `str` with many decimal places if there are subtle differences between the 2.

You can also provide your own filter function if there's a particular way you would like to see the data when comparing, for example you can use `write.table` if your data is easiest to understand in tabular output.

Value

Returns TRUE if a & b are `all.equal` (for `ut_cmp_equal()`) or `identical` (for `ut_cmp_identical()`). Otherwise, returns an `invisible()` character vector of diagnostic strings helping you find where the difference is.

If called directly in an interactive R session, this output will be printed to the console.

Examples

```
## A function to test:
fn <- function(x) { seq(x) }

## Get it right, and test passes:
ok(ut_cmp_equal(fn(3), c(1,2,3)))

## Get it wrong, and we get told where in the output things are different:
ok(ut_cmp_equal(fn(3), c(1,4,3)))

## Using a custom filter, we can format the output with write.table:
ok(ut_cmp_equal(fn(3), c(1,4,3), filter = write.table))

## With ut_cmp_equal, an integer 1 is the same as a numeric 1
ok(ut_cmp_equal(as.numeric(1), as.integer(1)))

## With ut_cmp_identical, they're not
ok(ut_cmp_identical(as.numeric(1), as.integer(1)))

## all.equal() takes a tolerance parameter, for example:
all.equal(0.01, 0.02, tolerance = 0.1)

## ...we can also give this to to ut_cmp_equal if we want a very
## approximate comparison
ok(ut_cmp_equal(0.01, 0.02, tolerance = 0.1))

## We can make a comparison function of our own, and use
## deparse_frame to show the right expression in diff output
cmp_noorder <- function (a, b) {
    sortlist <- function (x) if (length(x) > 0) x[order(names(x))] else x
    ut_cmp_identical(sortlist(a), sortlist(b), deparse_frame = -2)
}
ok(cmp_noorder(list(a=1, b=2), list(b=2, a=3)))
```

`ut_cmp_error` *Test for and compare errors generated by code*

Description

A helper to catch expected errors and ensure they match what is expected

Usage

```
ut_cmp_error(code, expected_regexp, ignore.case = FALSE, perl = FALSE, fixed = FALSE)
```

Arguments

`code`	Code expression to test, should generate an error
`expected_regexp`	Regular expression the error should match
`ignore.case`	Passed to `grepl`
`perl`	Passed to `grepl`
`fixed`	Passed to `grepl`

Value

Returns `TRUE` if `exp` generates an error and matches `expected_regexp`. Returns a string with expected and actual error if `exp` generates an error but does not match. Returns `"No error returned"` if `exp` does not generate an error.

Examples

```
ok(ut_cmp_error({
  stop("Hammer time")
}, "hammer", ignore.case = TRUE), "Returned a hammer-based error")
```

Index