Frequently Asked Questions

unittest v1.3-0 (2017-11-01)

Contents

1 General

1.1 How do I test error conditions?

Define a helper function `test_for_error` that uses `tryCatch`. If your test results in an error that does not match, then the test fails and the actual error will be included in your test results.

```
test_for_error <- function(code, expected_regexp) {
    tryCatch({
            code
            return("No error returned")
        }, error = function(e) {
            if(grepl(expected_regexp, e$message)) return(TRUE)
            return(c(e$message, "Did not match:-", expected_regexp))
        }
    )
}
```

For example, here is a function that will throw an error for a bad argument

```
add_four <- function(x) {
    if( ! is.numeric(x) ) stop("x must be numeric")
    return( x+4 )
}
```

Then we can test the argument check like this

```
ok(test_for_error(add_four("a"), "must be numeric"), "add_four() argument not numeric throws error")
```

```
ok - add_four() argument not numeric throws error
```

1.2 How do I test multivalue results?

Use `all.equal(...)`

```
a <- c(1,2,3)
b <- 1:3
ok(all.equal(a,b), "a and b are equal")
```

```
ok - a and b are equal
```

Alternatively, the following helper function will give coloured output showing what's different

```
cmp <- function(a, b) {
    if(identical(all.equal(a,b), TRUE)) return(TRUE)

    if (file.exists(Sys.which('git'))) {
        totmp <- function(x) {
            f <- tempfile(pattern = "str.")
            capture.output(str(x,
                vec.len = 1000,
                digits.d = 5,
                nchar.max = 1000), file = f)
            return(f)
        }

        return(suppressWarnings(system2(
            Sys.which('git'),
            c("diff", "--no-index", "--color-words", totmp(a), totmp(b)),
            input = "",
            stdout = TRUE, stderr = TRUE)))
    }

    return(c(
        capture.output(str(a)),
        "... does not equal...",
        capture.output(str(b))
    ))
}
```

To see the coloured output, try the following:

```
ok(cmp(1:3, 1:8))
```

1.3 Grouping tests

When dealing with many unit tests in one file it can be useful to group related unit tests.
The `ok_group()` function is used like this

```
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")
})
```

which produces the following output

```
# Test addition
ok - Can add 1
ok - Can add 3

# Test subtraction
ok - Can subtract 1
ok - Can subtract 3
```

1.4 I am sure I do not need to test my code. Is this true?

No. Sit down and have a cup of tea. Hopefully the feeling will go away.

2 Working with packages

2.1 I'm writing a package, how do I put tests in it?

Add the following line to the package `DESCRIPTION` file.

```
Suggests: unittest
```

Create a directory called `tests` in your package source, alongside your `R` directory.
Place your tests in a file with the extension `.R` and add the following lines to the top of the file (replacing `mypackage` with the name of your package).

```
library(mypackage)
library(unittest, quietly = TRUE)
```

That's it; `R CMD check` will run the tests and fail if any of the tests fail.

Any `.R` file in the `tests` directory will be run by `R CMD check`.
When you use the `unittest` package the package "knows" that it is being run by `CMD check` and at the end of the tests it produces a summary of the results. The package will also throw an error if any tests fail; throwing an error will in turn cause `CMD check` to report the error and fail the `check`.

Here is a very simple example:
Assuming your package contains (and `exports`) the function `biggest()`

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

then the `tests/my_tests_for_biggest.R` file could contain something like

```
library(mypackage)
library(unittest, quietly = TRUE)

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

2.2 How do I test un-exported package functions?

If you have some unit tests which require access to un-exported functions, or un-exported S3 methods, you can use `local`.

```
local({
    ok(internal_function() == 3)
    ok(another_internal_function() == 4)
    ok(final_internal_function() == 5)
}, asNamespace('mypackage'))
```