كلية طب وايل كورنيل في قطر
Weill Cornell Medical College in Qatar

Member of Qatar Foundation

Testing R Code

Richie Cotton, June 2015

bitly.com/cottontestingr

Today's Itinerary

1. Introduction
2. Run-time testing with `assertive`

2.5 Tea and biscuits, maybe cake

3. Development-time testing with `testthat`
4. Writing maintainable and testable code (depending on timing)

The very short version

- Run-time testing is for checking that people using your code aren't doing stupid things.
- Development-time testing is for checking that you didn't do stupid things when writing your code.
- If you break your code down into small pieces, it will usually be easier to maintain and easier to test.

I HATE
EVERYTHING
memegenerator.net

Run-time testing with `assertive`

check that `counts` is a numeric vector of non-negative, whole numbers

```
counts <- c(1, 2, 3, 4.5)
```

check that `counts` is a numeric vector of non-negative, whole numbers

```
counts <- c(1, 2, 3, 4.5)
stopifnot(




)
```

check that `counts` is a **numeric vector** of non-negative, whole numbers

```
counts <- c(1, 2, 3, 4.5)
stopifnot(
  is.numeric(counts)


)
```

check that `counts` is a numeric vector of non-negative, whole numbers

```
counts <- c(1, 2, 3, 4.5)
stopifnot(
  is.numeric(counts),
  all(counts >= 0)

)
```

check that `counts` is a numeric vector of non-negative, whole numbers

```
counts <- c(1, 2, 3, 4.5)
stopifnot(
  is.numeric(counts),
  all(counts >= 0),
  isTRUE(all.equal(counts, round(counts)))
)
```

check that `counts` is a numeric vector of non-negative, whole numbers

```
counts <- c(1, 2, 3, 4.5)
stopifnot(
  is.numeric(counts),
  all(counts >= 0),
  isTRUE(all.equal(counts, round(counts)))
)
## Error: isTRUE(all.equal(counts,
##   round(counts))) is not TRUE
```

check that `counts` is a numeric vector of non-negative, whole numbers

```
counts <- c(1, 2, 3, 4.5)
assert_is_numeric(counts)
assert_all_are_non_negative(counts)
assert_all_are_whole_numbers(counts)
```

check that `counts` is a numeric vector of non-negative, whole numbers

```
counts <- c(1, 2, 3, 4.5)
assert_is_numeric(counts)
assert_all_are_non_negative(counts)
assert_all_are_whole_numbers(counts)
## Error: counts are not all whole numbers.
## There was 1 failure:
##   Position Value      Cause
## 1        4   4.5 fractional
```

```
is_numeric(1:6)
## [1] TRUE

is_numeric(letters)
## [1] FALSE
## attr(,"cause")
## [1] letters is not of type 'numeric';
## it has class 'character'.
```

```
is_numeric(1:6)
## [1] TRUE

is_numeric(letters)
## [1] FALSE
## attr(,"cause")
## [1] letters is not of type 'numeric';
## it has class 'character'.

assert_is_numeric(1:6)

assert_is_numeric(letters)
## Error: letters is not of type 'numeric';
## it has class 'character'.
```

```
is_non_negative(c(-1, 0, 1, NA))

##    -1     0     1  <NA>
## FALSE  TRUE  TRUE    NA
## attr(,"cause")
## [1] too low                 missing
```

```
is_non_negative(c(-1, 0, 1, NA))

##    -1     0     1  <NA>
## FALSE  TRUE  TRUE    NA
## attr(,"cause")
## [1] too low                 missing

assert_any_are_non_negative(c(-1, 0, 1, NA))

assert_all_are_non_negative(c(-1, 0, 1, NA))
## Error: c(-1, 0, 1, NA) are not all non-negative.
## There were 2 failures:
##   Position Value   Cause
## 1        1    -1 too low
## 2        4  <NA> missing
```

Testing Variable Types

Check variable types

`is_numeric, is_character, is_data.frame, is_qr and many more`

Check properties of variables

`is_s4, is_atomic, is_recursive, is_language`

Combine variable type check with `is_scalar`

`is_a_number, is_a_string, is_a_bool, etc.`

Testing Variable Sizes

Check variable has length one, or one element

`is_scalar`

Check length/n elements zero/not zero

`is_empty, is_non_empty`

Generalization

`is_of_length, has_elements`

Testing Missing Values

Check for NA, NaN, and NULL

`is_na, is_nan, is_null`

The opposite checks

`is_not_na, is_not_nan, is_not_null`

Testing Number Ranges

Check variable is in a range

`is_in_range`

Control edges

`is_in_open_range`, `is_in_closed_range`,
`is_in_left_open_range`, `is_in_right_open_range`

Specific common ranges

`is_positive`, `is_negative`,
`is_non_positive`, `is_non_negative`,
`is_proportion` and `is_percentage`

Testing Number Properties

Check infiniteness

`is_finite, is_infinite, is_positive_infinity, is_negative_infinity`

Check oddness

`is_odd, is_even, is_divisible_by, is_whole_number`

Check complexity

`is_real, is_imaginary`

Testing Attributes

Check dimensions and their names

`has_rows, has_cols, has_dims, has_rownames, has_colnames, has_dimnames, has_names`

Check duplication

`has_duplicates, has_no_duplicates`

Check attributes

`has_attributes, has_any_attributes`

Testing Files and Connections

Check file and dir existence

`is_existing_file, is_dir`

Check file permissions (dubious under Windows)

`is_executable_file, is_readable_file,`
`is_writable_file`

Check attributes

`is_connection, is_file_connection,`
`is_fifo_connection, is_open_connection,`
`is_writable_connection, is_stdin, etc.`

Testing Times

Check string formatted correctly
`is_date_string`

Check time relative to now
`is_in_future, is_in_past`

Testing Sets

Check same elements, whatever order

`is_set_equal`

Check sets contained in one another

`is_subset, is_superset`

Testing Complex Data Types

Check misc types

`is_email_address, is_credit_card_number, is_honorific, is_ip_address, is_hex_color, is_cas_number, is_isbn_code`

Check UK types

`is_uk_car_licence, is_uk_postcode, is_uk_national_insurance_number, is_uk_telephone_number`

Check US types

`is_us_telephone_number, is_us_zip_code`

Testing The Setup

Check operating system

`is_windows, is_linux, is_mac,`
`is_solaris, is_bsd, is_unix`

Check R version

`is_r, is_r_devel, is_r_stable, is_r_alpha,`
`is_r_patched, is_current_r, etc.`

Check R's capabilities

`r_has_png_capability, r_has_tcltk_capability`

More Testing The Setup

Check decimal point convention

`is_comma_for_decimal_point,`
`is_period_for_decimal_point`

Check how R is run

`is_slave_r, is_interactive, is_batch_mode,`
`is_64_bit, is_rstudio, is_revo_r, is_architect`

Check system tool availability

`r_can_compile_code, r_can_build_translations`

Testing Code

Check code properties

```
is_debugged, is_valid_variable_name,
is_error_free, is_valid_r_code
```

```
geomean <- function(x, na.rm = FALSE)
{
  exp(mean(log(x), na.rm = na.rm))
}

geomean("a")
## Error in log(x): non-numeric argument
## to mathematical function
```

```
geomean2 <- function(x, na.rm = FALSE)
{
  assert_is_numeric(x)
  exp(mean(log(x), na.rm = na.rm))
}

geomean2("a")
## Error: x is not of type 'numeric'; it has
## class 'character'.
```

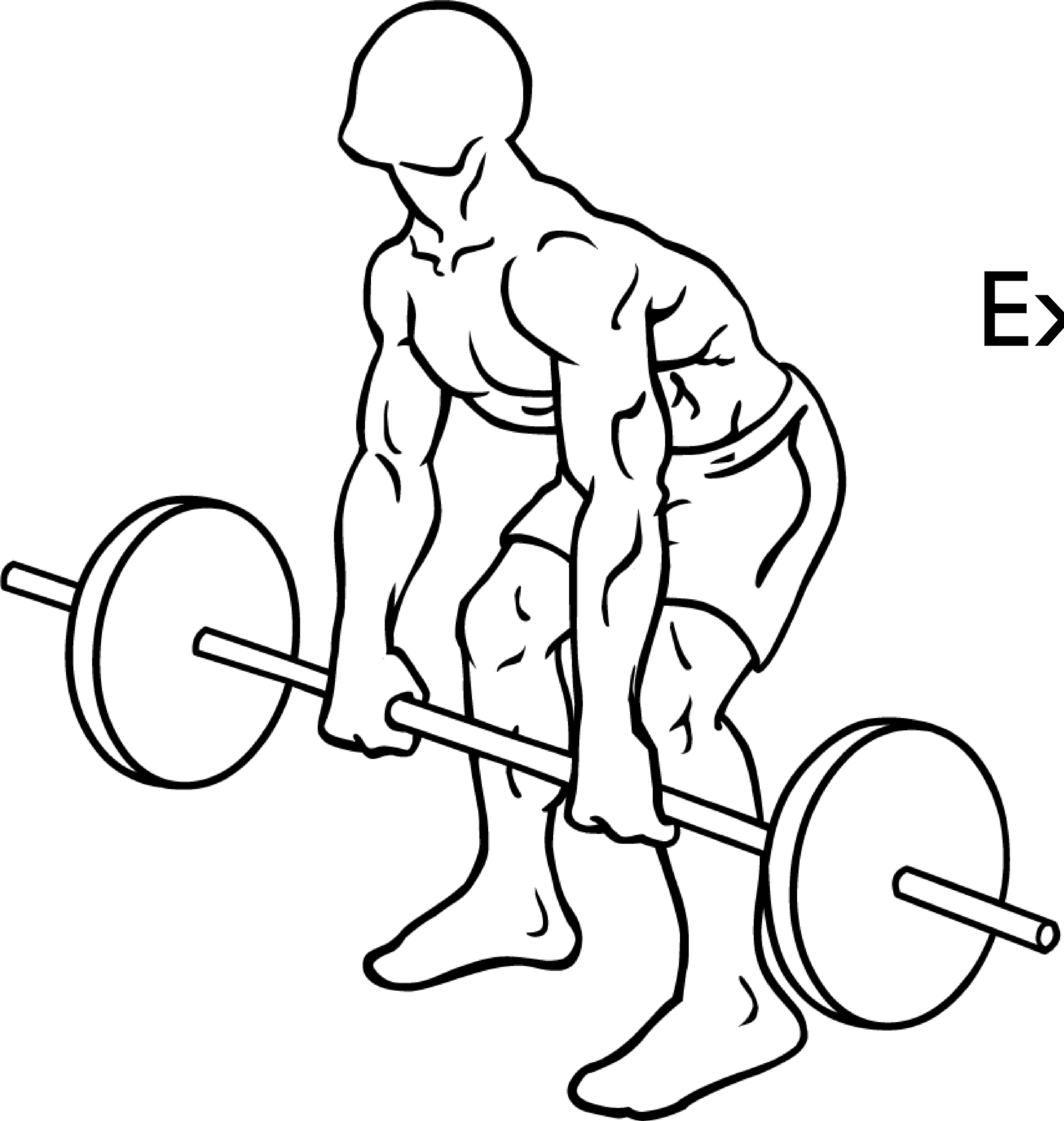
```
geomean2 <- function(x, na.rm = FALSE)
{
  assert_is_numeric(x)
  exp(mean(log(x), na.rm = na.rm))
}

geomean2(rnorm(20))
## Warning in log(x): NaNs produced
## [1] NaN
```

```
geomean3 <- function(x, na.rm = FALSE)
{
  assert_is_numeric(x)
  if(any(is_negative(x), na.rm = TRUE))
  {
    warning("x contains negative values, so
the geometric mean makes no sense.")
    return(NaN)
  }
  exp(mean(log(x), na.rm = na.rm))
}

geomean3(rnorm(20))
## Warning in geomean3(rnorm(20)): x contains
## negative values, so the geometric mean
## makes no sense.
## [1] NaN
```

```
x <- rlnorm(20)
x[sample(20, 5)] <- NA

geomean(x, c(1.5, 0))
## Warning in if (na.rm) x <- x[!is.na(x)]:
## the condition has length > 1 and only the
## first element will be used
## [1] 0.990337
```

```
use_first(c(1.5, 0))
## [1] 1.5
## Warning message:
## Only the first value of ‘c(1.5, 0)’ will
## be used.

coerce_to(c(1.5, 0), "logical")
[1]  TRUE FALSE
Warning message:
Coercing c(1.5, 0) to class ‘logical’.
```

```
geomean4 <- function(x, na.rm = FALSE)
{
  assert_is_numeric(x)
  if(any(is_negative(x), na.rm = TRUE))
  {
    warning("x contains negative values, so the
geometric mean makes no sense.")
    return(NaN)
  }
  na.rm <- coerce_to(use_first(na.rm), "logical")
  exp(mean(log(x), na.rm = na.rm))
}

geomean4(x, c(1.5, 0))
## Warning: Only the first value of 'na.rm' will
be used.
## Warning: Coercing use_first(na.rm) to class
'logical'.
## [1] 0.990337
```



Exercises

Development-time testing with `testthat`

```
hypotenuse <- function(x, y)
{
  sqrt(x ^ 2 + y ^ 2)
}
```

```r
test_that(
  "hypotenuse, with inputs x = 3 and y = 4, returns 5",
  {
    expected <- 5
    actual <- hypotenuse(3, 4)
    expect_equal(actual, expected)
  }
)
```

```
test_that(
  "hypotenuse, with no inputs, throws an 
error",
  {
    expect_error(
      hypotenuse(),
      'argument "x" is missing, with no 
default'
    )
  }
)
```

Other common variants of expectations are:

- `expect_true`, `expect_false` and `expect_null`, which are shortcuts for checking those common return types.
- `expect_warning`, `expect_message` and `expect_output`, for testing feedback, which work like `expect_error`.

You may also occasionally come across these rare expectations:

- `expect_less_than` and `expect_greater_than`, for numeric inequalities.
- `expect_identical`, a stricter check than `expect_equal`.
- `match`, for matching strings using regular expressions.
- `is`, for checking the class of variables.

```r
test_that(
  "min, with a zero-length input, returns infinity with a warning",
  {
    expected <- Inf
    expect_warning(
      actual <- min(numeric()),
      "no non-missing arguments to min; returning Inf"
    )
    expect_equal(actual, expected)
  }
)
```

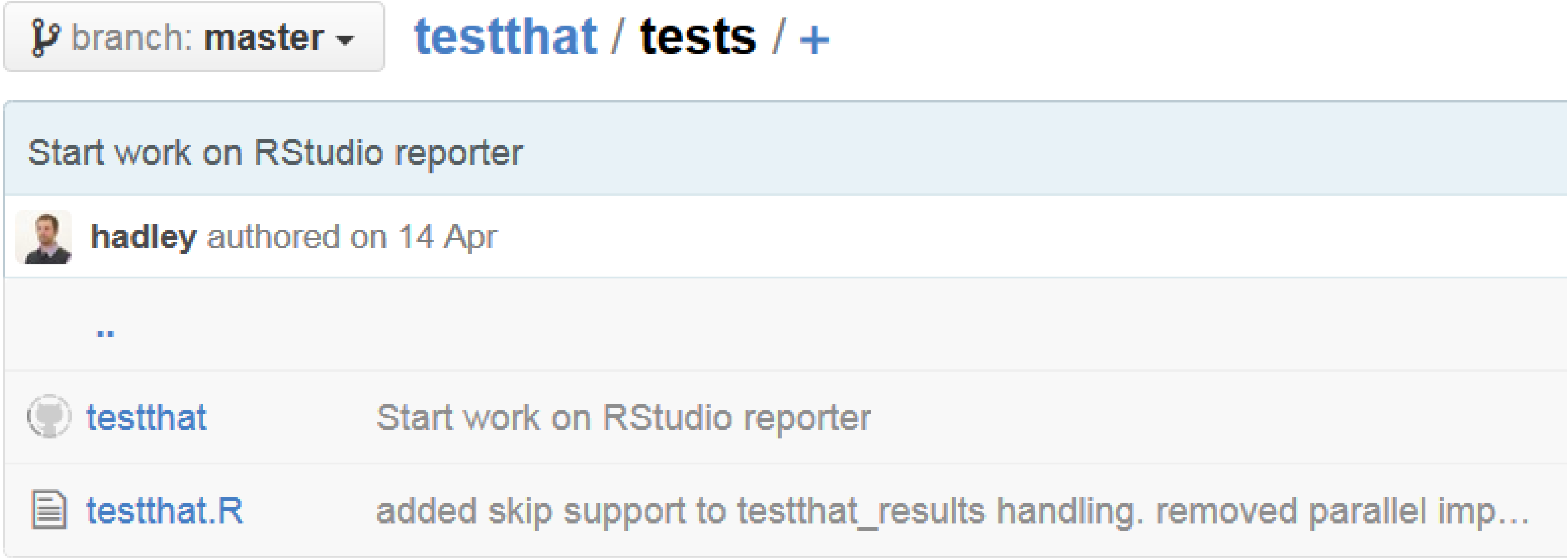
```
test_that(
  "hypotenuse, with a NULL input, returns NULL",
  {
    expect_null(hypotenuse(3, NULL))
  }
)
```

```
test_that(
  "hypotenuse, with a NULL input, returns NULL",
  {
    expect_null(hypotenuse(3, NULL))
  }
)
```

```
## Error: Test failed: 'hypotenuse, with a NULL
## input, returns NULL'
## Not expected: hypotenuse(3, NULL) isn't null.
```

```r
test_that(
  "hypotenuse, with a NULL input, returns NULL",
  {
    actual <- hypotenuse(3, NULL)
    label <- paste(
      "hypotenuse(3, NULL) =",
      deparse(actual)
    )
    expect_null(actual, label = label)
  }
)
```

```
## Error: Test failed: 'hypotenuse, with a NULL
## input, returns NULL'
## Not expected: hypotenuse(3, NULL) = numeric(0)
## isn't null.
```



testthat / tests / +

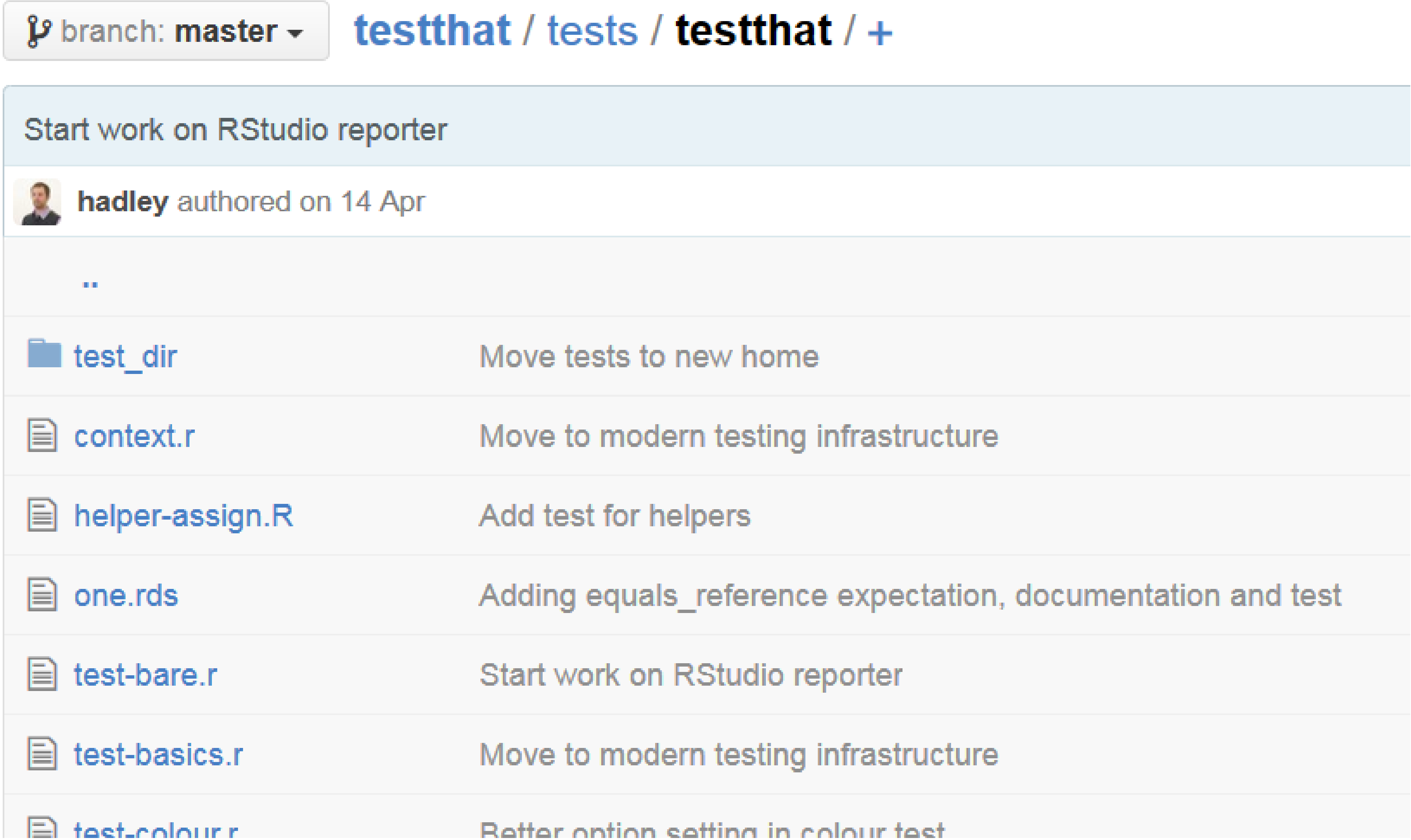
Start work on RStudio reporter

hadley authored on 14 Apr

..

testthat	Start work on RStudio reporter
testthat.R	added skip support to testthat_results handling. removed parallel imp…

```
library(testthat)
library(devtools)
library(yourpackage)

with_envvar(
  c(LANG = "en_US"),
  test_package("yourpackage")
)
```



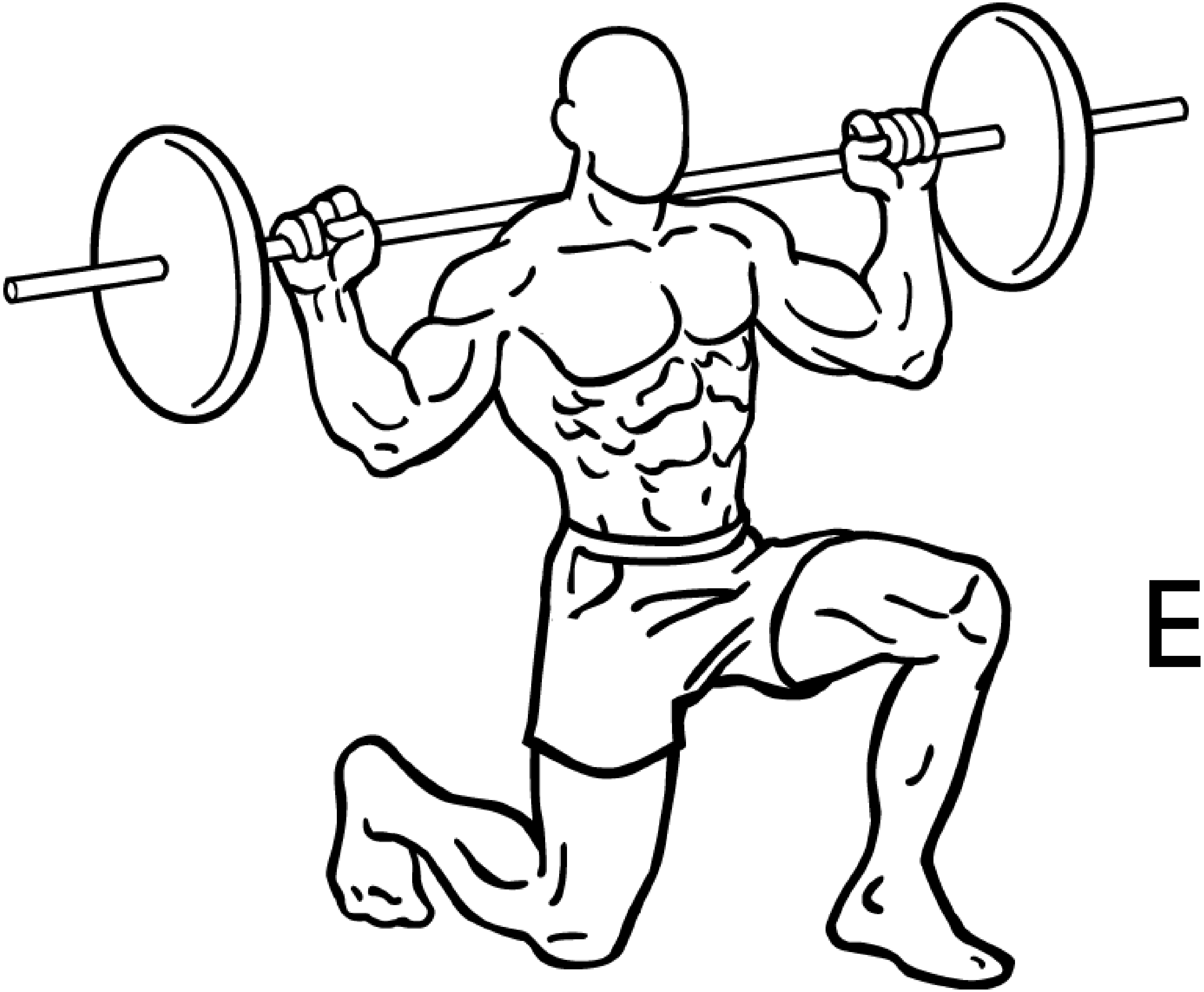
testthat / tests / testthat / +

Start work on RStudio reporter

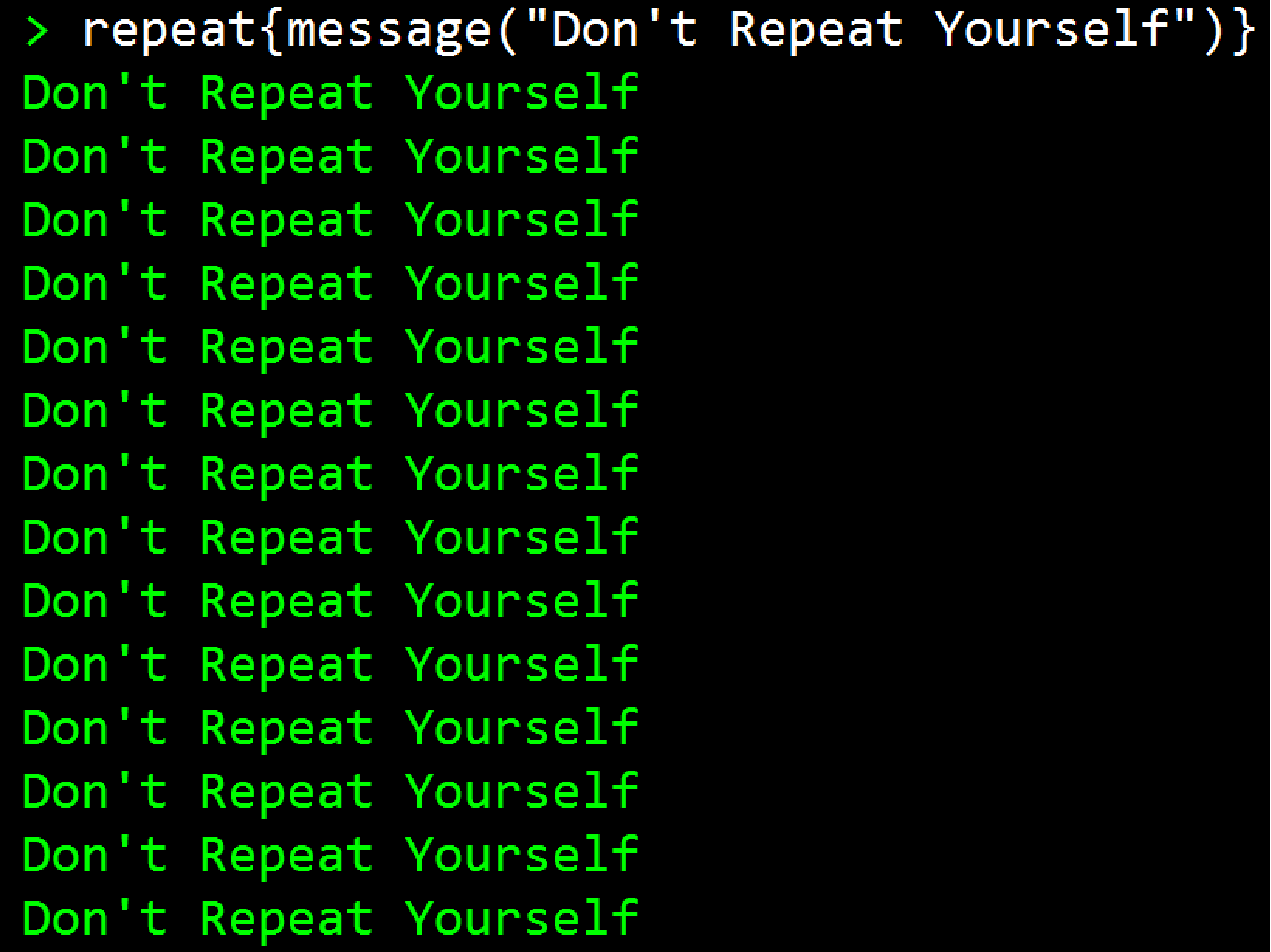
hadley authored on 14 Apr

..

test_dir	Move tests to new home
context.r	Move to modern testing infrastructure
helper-assign.R	Add test for helpers
one.rds	Adding equals_reference expectation, documentation and test
test-bare.r	Start work on RStudio reporter
test-basics.r	Move to modern testing infrastructure
test-colour.r	Better option setting in colour test



Exercises



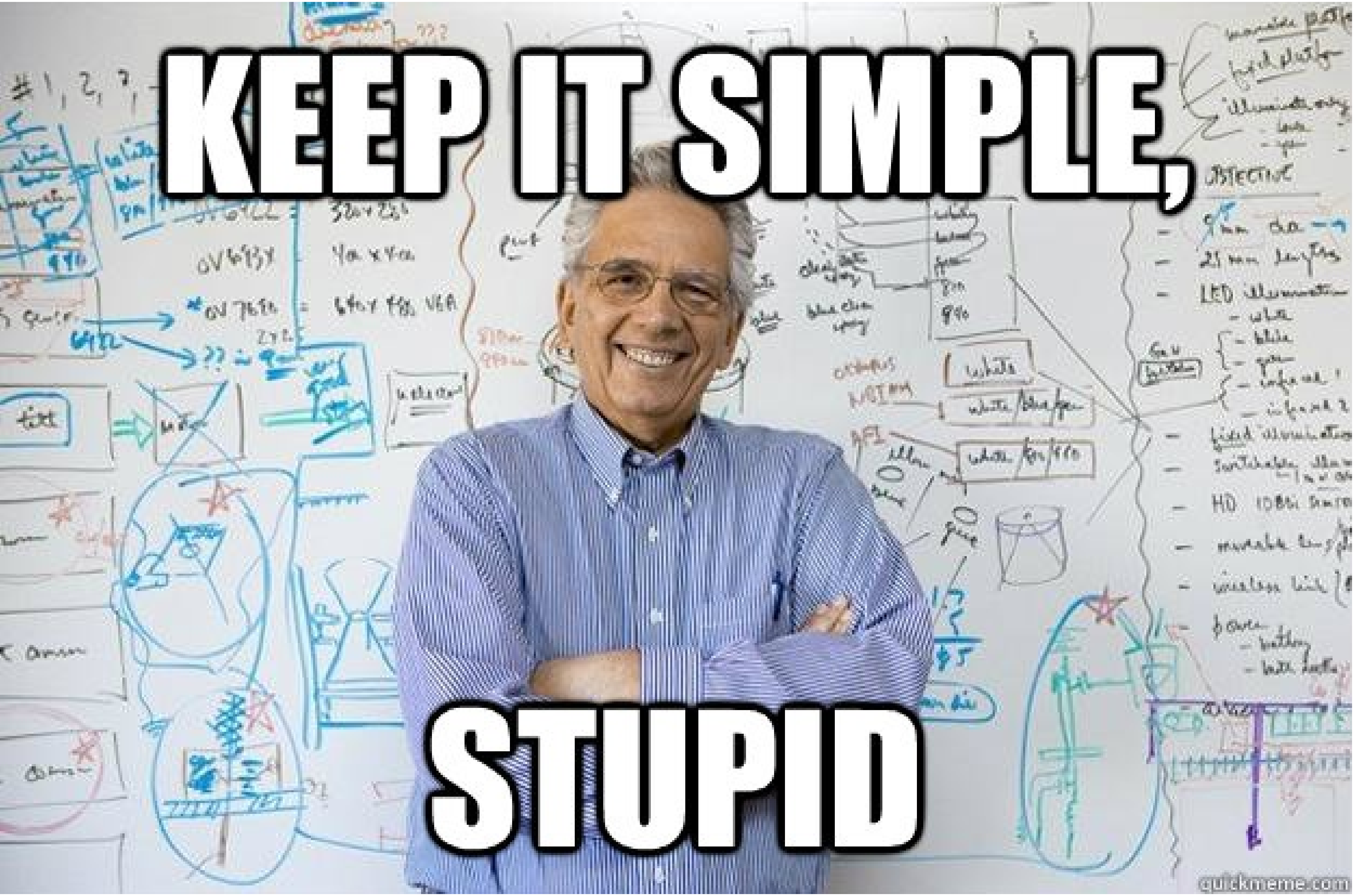
```
> repeat{message("Don't Repeat Yourself")}
Don't Repeat Yourself
Don't Repeat Yourself
Don't Repeat Yourself
Don't Repeat Yourself
Don't Repeat Yourself
Don't Repeat Yourself
Don't Repeat Yourself
Don't Repeat Yourself
Don't Repeat Yourself
Don't Repeat Yourself
Don't Repeat Yourself
Don't Repeat Yourself
Don't Repeat Yourself
Don't Repeat Yourself
```

Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.

`c2.com`

Duplicated code is bad code: anything that appears in two or more places in a program will eventually be wrong in at least one.

`softwarecarpentry.org`


KEEP IT SIMPLE,
STUPID
quickmeme.com

On a good day, you can keep 7 (give or take 2) things in your working memory

`Miller's law, paraphrased`

On a good day, you can keep 7 (give or take 2) things in your working memory

`Miller's law, paraphrased`

Most of your R functions should be seven lines or less

`Cotton's corollary`

Cyclomatic Complexity

Cyclomatic complexity measures the number of linearly independent paths through the method, which is determined by the number and complexity of conditional branches. A low cyclomatic complexity generally indicates a method that is easy to understand, test, and maintain.

`microsoft.com`

```
cyclo_single_path <- function()
{
  message("Hello World!")
}
```

Cyclomatic complexity = 1

```
cyclo_if <- function(condition)
{
  if(condition)
  {
    message("Hello World!")
  }
}
```

Cyclomatic complexity = 2 or 3

```
cyclo_switch <- function(date)
{
  switch(
    weekdays(date),
    Monday    = message("The day is Monday!"),
    Tuesday   = message("The day is Tuesday!"),
    Wednesday = message("The day is Wednesday!"),
    Thursday  = message("The day is Thursday!"),
    Friday    = message("The day is Friday!"),
    Saturday  = message("The day is Saturday!"),
    Sunday    = message("The day is Sunday!"),
    message("The input date is missing")
  )
}
```

Cyclomatic complexity = 8 or 9

Fail Early, Fail Often

Make each module fail fast – either it does the right thing or it stops.

Jim Gray

Exercises