

Unit testing

(With a dash of API design)

January 2018

Hadley Wickham

@hadleywickham

Chief Scientist, RStudio

Motivation

Let's add a column to a data frame

```
# Write a function that allows us to add a  
# new column to a data frame at a specified  
# position.
```

```
add_col(df, "name", value, where = 1)  
add_col(df, "name", value, where = -1)
```

```
# Start simple and try out as we go
```

Your turn

```
# A useful building block is add_cols() -  
# works like cbind() but can insert anywhere
```

```
add_cols <- function(x, y, where = 1) {  
  if (where == 1) { # first col  
    ...  
  } else if (where > ncol(x)) { # last col  
    ...  
  } else {  
    ...  
  }  
}
```

My first attempt

```
add_cols <- function(x, y, where = 1) {  
  if (where == 1) {  
    cbind(x, y)  
  } else if (where > ncol(x)) {  
    cbind(y, x)  
  } else {  
    cbind(x[1:where], y, x[where:nrow(x)])  
  }  
}
```

Actually correct

```
add_cols <- function(x, y, where = 1) {  
  if (where == 1) {  
    cbind(y, x)  
  } else if (where > ncol(x)) {  
    cbind(x, y)  
  } else {  
    lhs <- 1:(where - 1)  
    cbind(x[lhs], y, x[-lhs])  
  }  
}
```

How did I write that code?

```
# Some simple inputs
```

```
df1 <- data.frame(a = 3, b = 4, c = 5)
```

```
df2 <- data.frame(X = 1, Y = 2)
```

```
# Then each time I tweaked it, I re-ran
```

```
# these cases
```

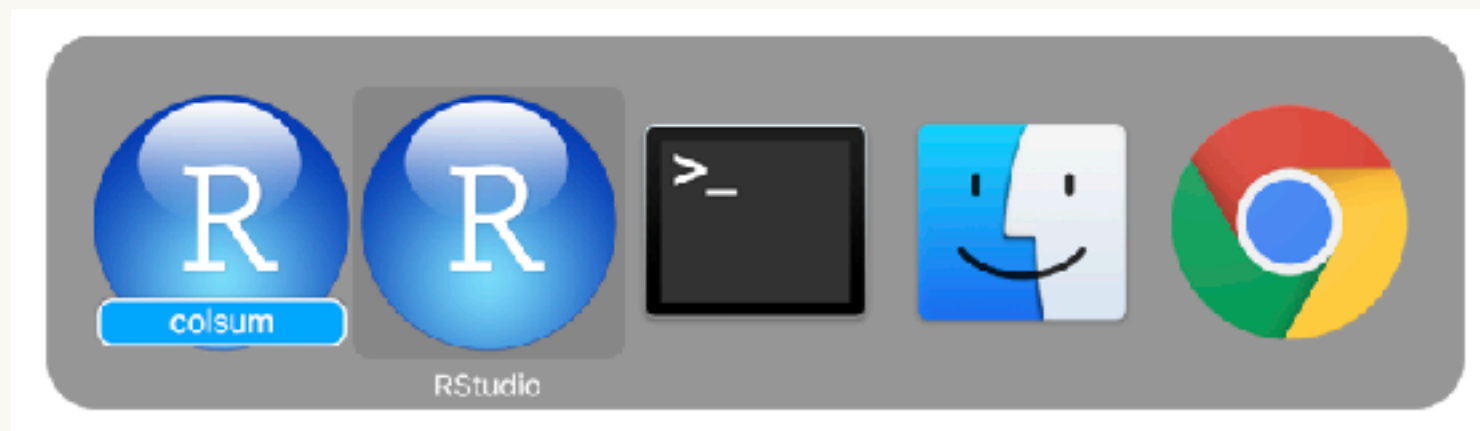
```
add_cols(df1, df2, where = 1)
```

```
add_cols(df1, df2, where = 2)
```

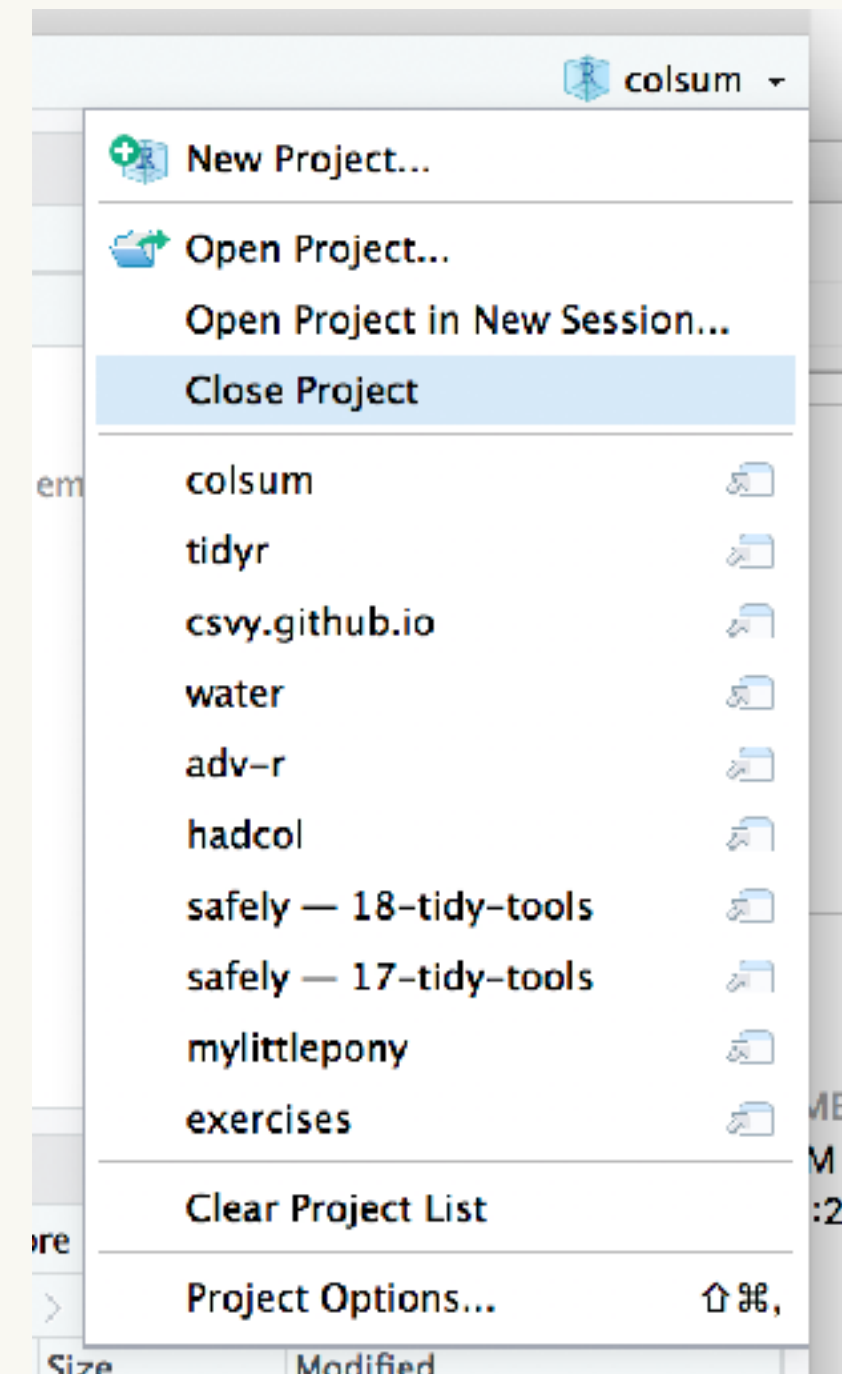
```
add_cols(df1, df2, where = 3)
```

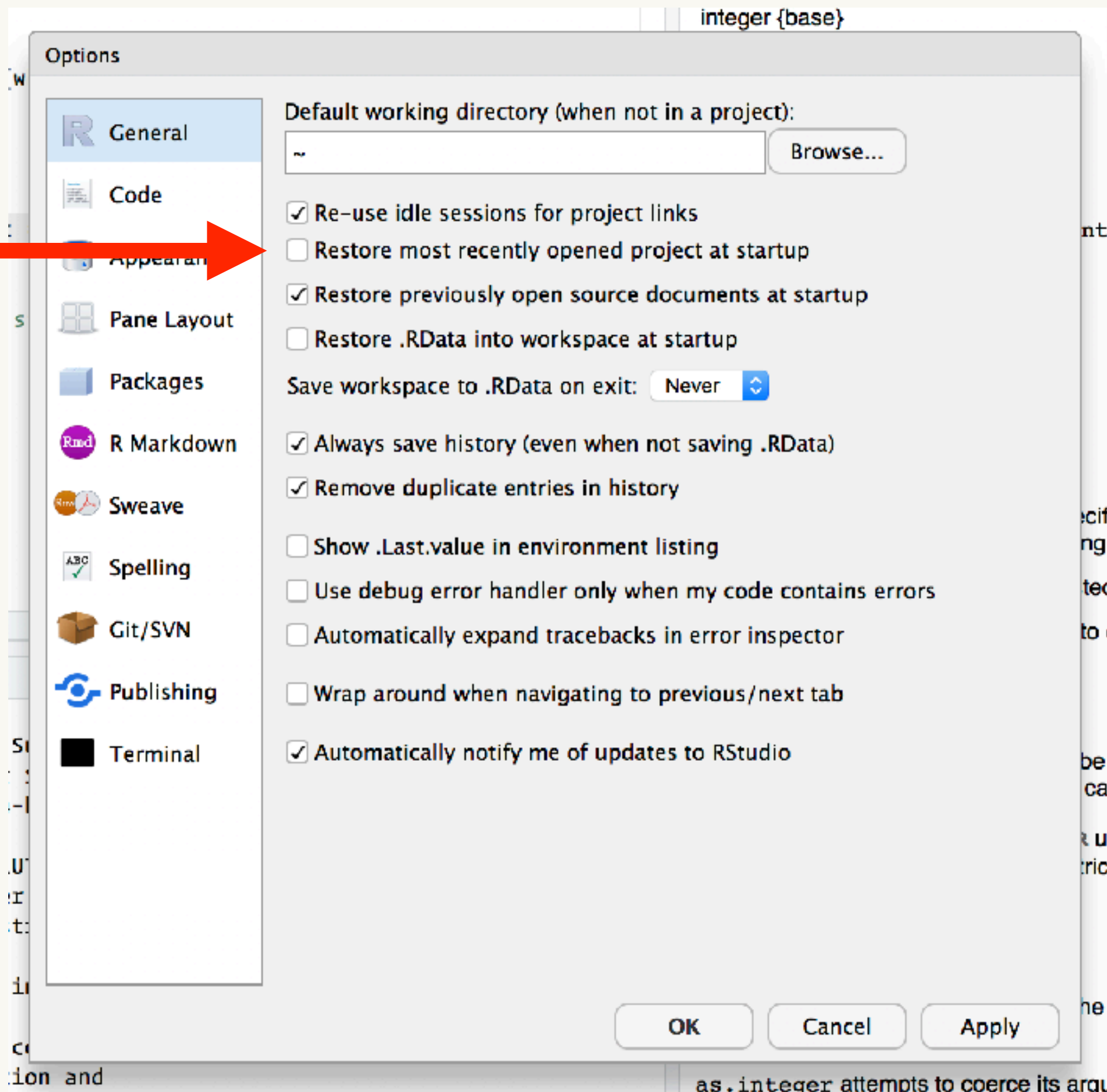
```
add_cols(df1, df2, where = 4)
```

Where did I write that code?



As well as RStudios associated with a project, you also get one associated with no project





Two challenges

Cmd + Enter is error prone

Looking at the outputs
each run is tedious

We need a new workflow!

Cmd + Enter is error prone

Put code in R/ and use `devtools::load_all()`

Looking at the outputs
each run is tedious

Write unit tests and use `devtools::test()`

Testing workflow

<http://r-pkgs.had.co.nz/tests.html>

We know how to create a package

```
usethis::create_package("~/desktop/hadcol")  
usethis::use_r("add_col")
```

```
add_cols <- function(x, y, where = 1) {  
  if (where == 1) {  
    cbind(y, x)  
  } else if (where > ncol(x)) {  
    cbind(x, y)  
  } else {  
    lhs <- 1:(where - 1)  
    cbind(x[lhs], y, x[-lhs])  
  }  
}
```

Even more convenient with some conventions

```
usethis::use_test()
```

Set up testthat infrastructure

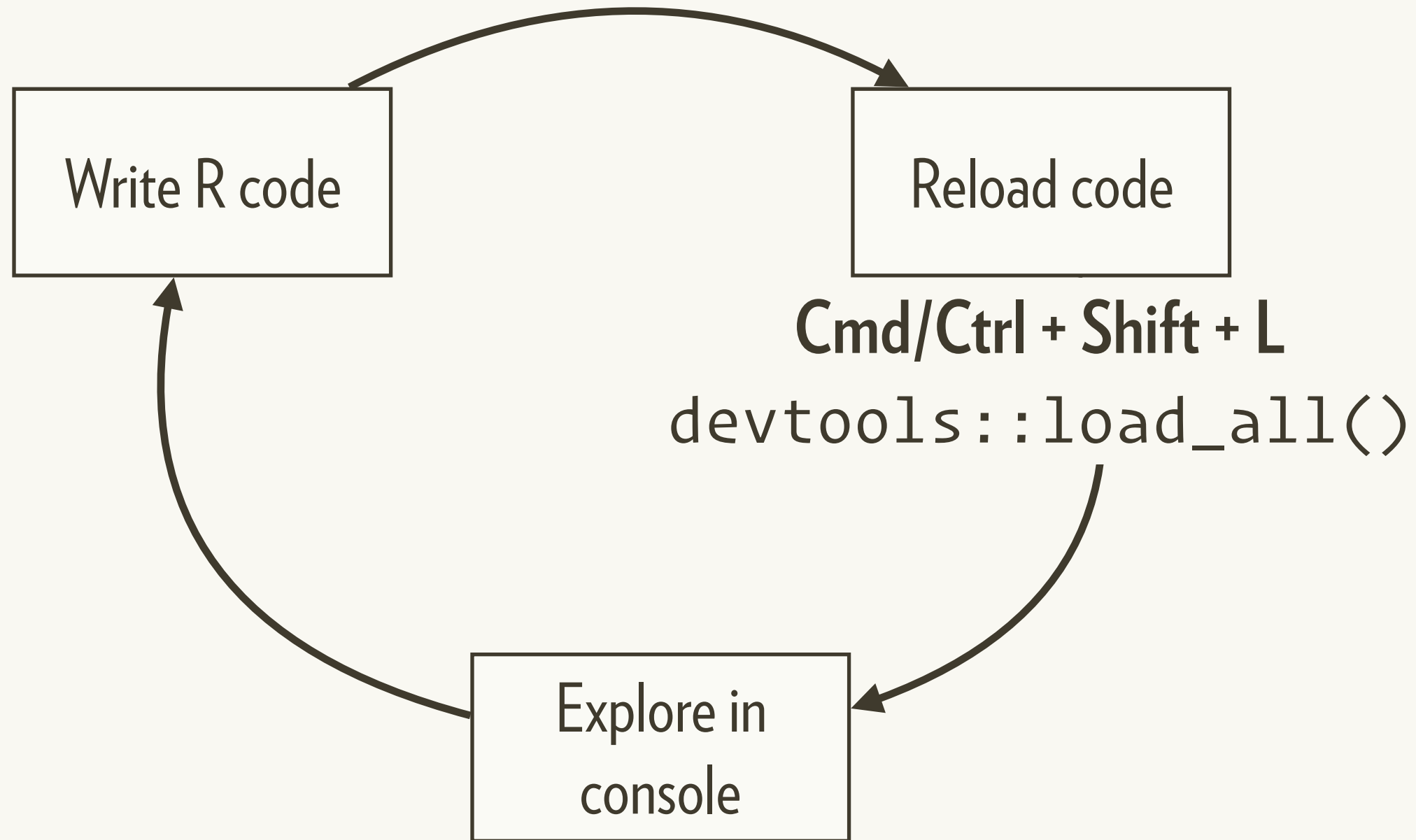
- ✓ Adding 'testthat' to Suggests field
- ✓ Creating 'tests/testthat/'
- ✓ Writing 'tests/testthat.R'
- ✓ Writing 'tests/testthat/test-add_cols.R'
- Modify 'tests/testthat/test-add_cols.R'

```
devtools::test()
```

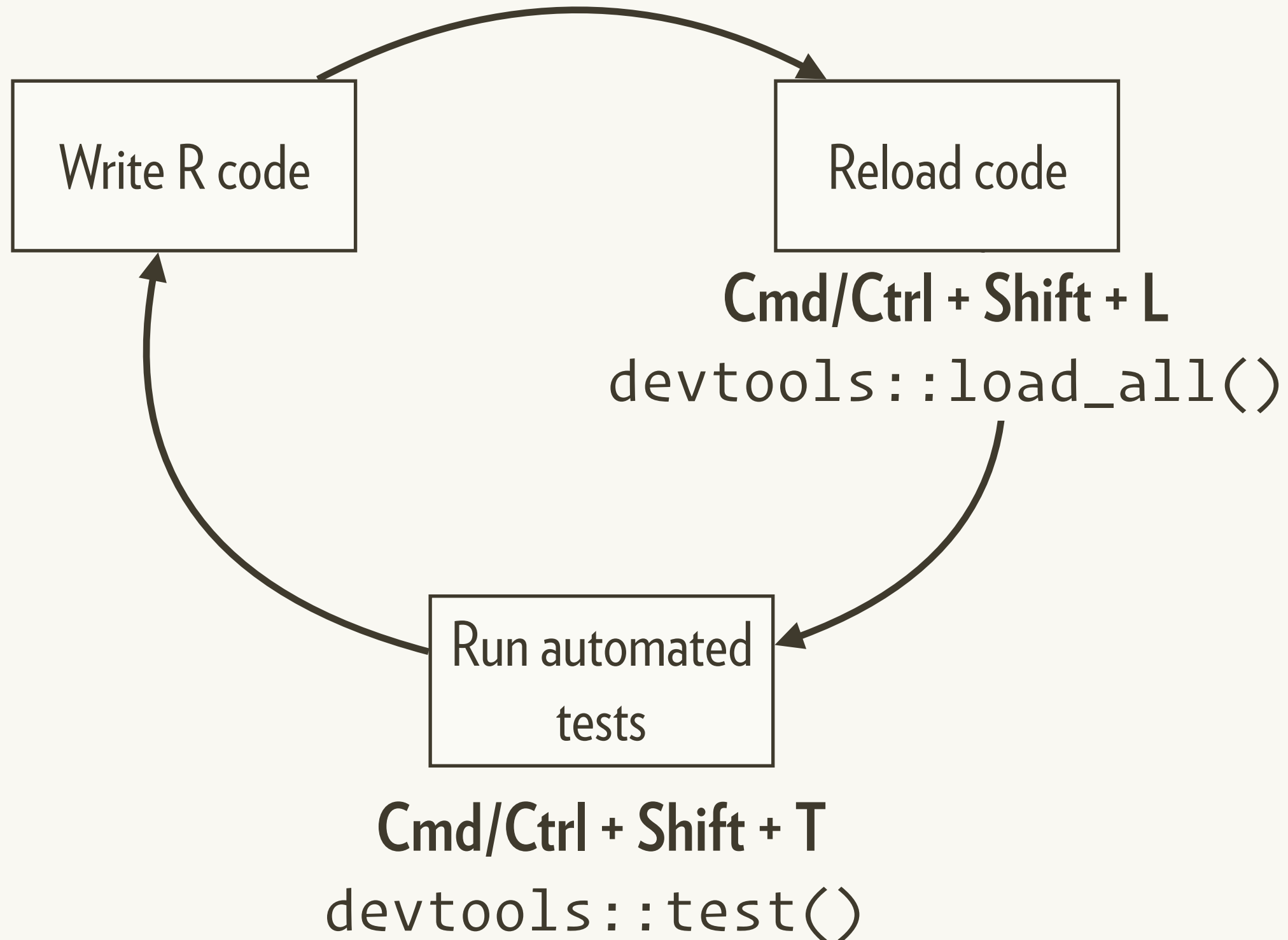
Create test file matching script

```
# Or Command + Shift + T
```

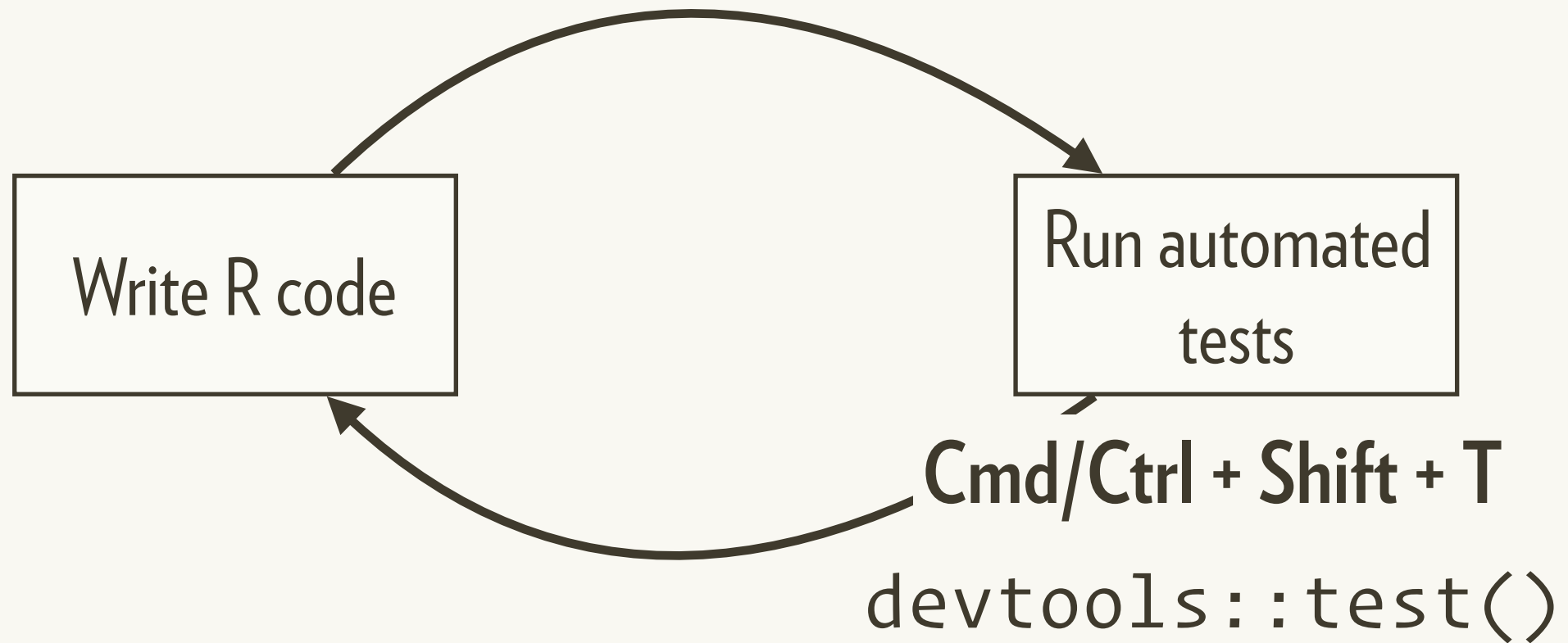
So far we've done this:



Testthat gives a new workflow



But why load the code?



Key idea of unit testing is to automate!

Helper function to reduce duplication

```
at_pos <- function(i) {  
  add_cols(df1, df2, where = i)  
}
```

```
expect_named(at_pos(1), c("X", "Y", "a", "b", "c"))  
expect_named(at_pos(2), c("a", "X", "Y", "b", "c"))  
expect_named(at_pos(3), c("a", "b", "X", "Y", "c"))  
expect_named(at_pos(4), c("a", "b", "c", "X", "Y"))
```

Describes an expected property of the output

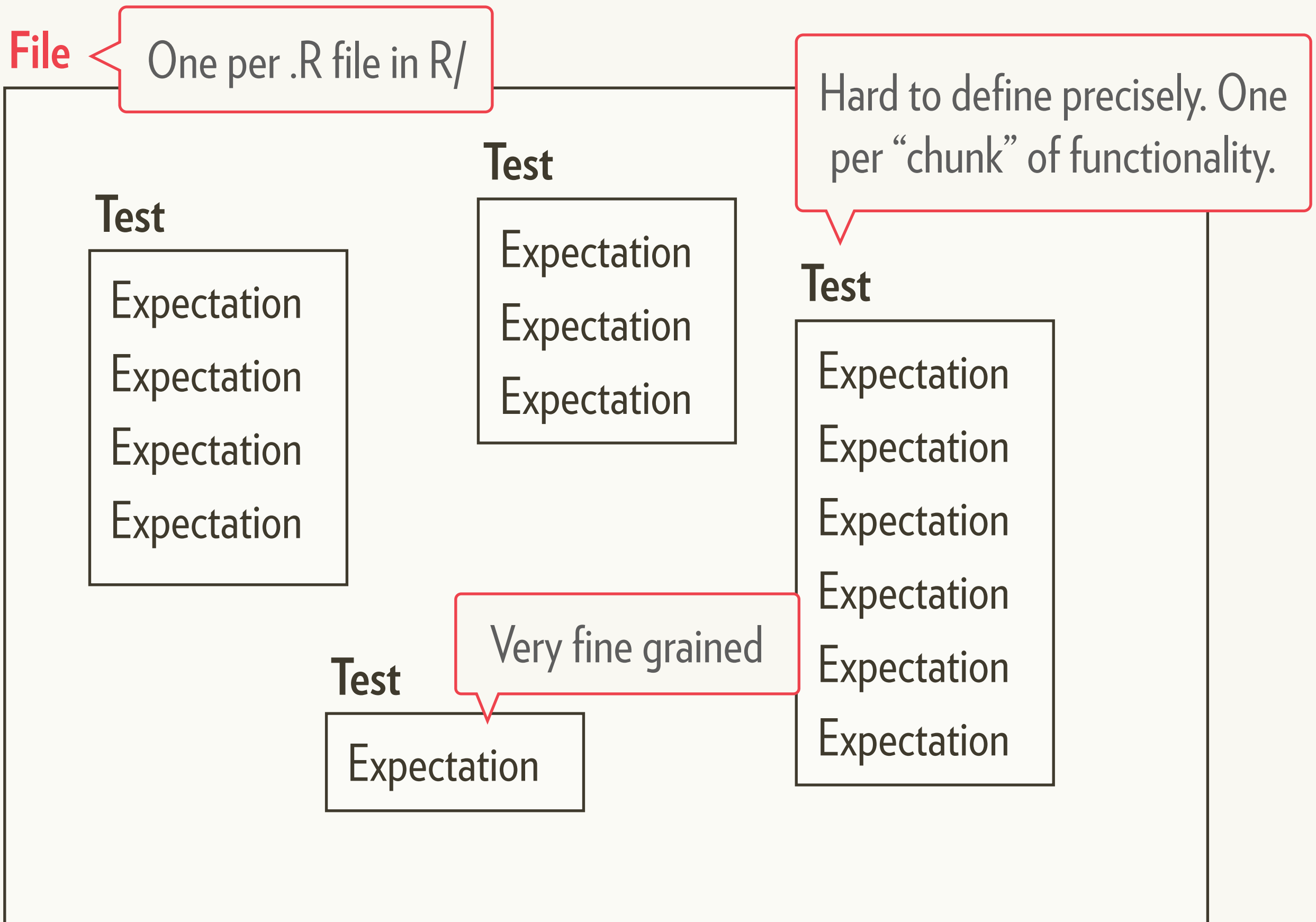
And this automation n⁶ ¹¹ventions

Tests for R/add_cols.R

```
# In tests/testthat/test-add_cols.R
test_that("can add column at any position", {
  df1 <- data.frame(a = 3, b = 4, c = 5)
  df2 <- data.frame(X = 1, Y = 2)
  at_pos <- function(i) {
    add_cols(df1, df2, where = i)
  }

  expect_named(at_pos(1), c("X", "Y", "a", "b", "c"))
  expect_named(at_pos(2), c("a", "X", "Y", "b", "c"))
  expect_named(at_pos(3), c("a", "b", "X", "Y", "c"))
  expect_named(at_pos(4), c("a", "b", "c", "X", "Y"))
})
```

Tests are organised in three layers



Practice the workflow

```
usethis::create_package("~/desktop/hadcol")  
usethis::use_r("add_col")  
# Copy add_cols() from slides  
usethis::use_test()  
# Copy expectations from slides  
# Run tests with keyboard shortcut  
# Break add_cols. Do the tests fail?
```

You should now be in freshly created

`[hadcol]`

(Download also has more complete hadcol if you get stuck)

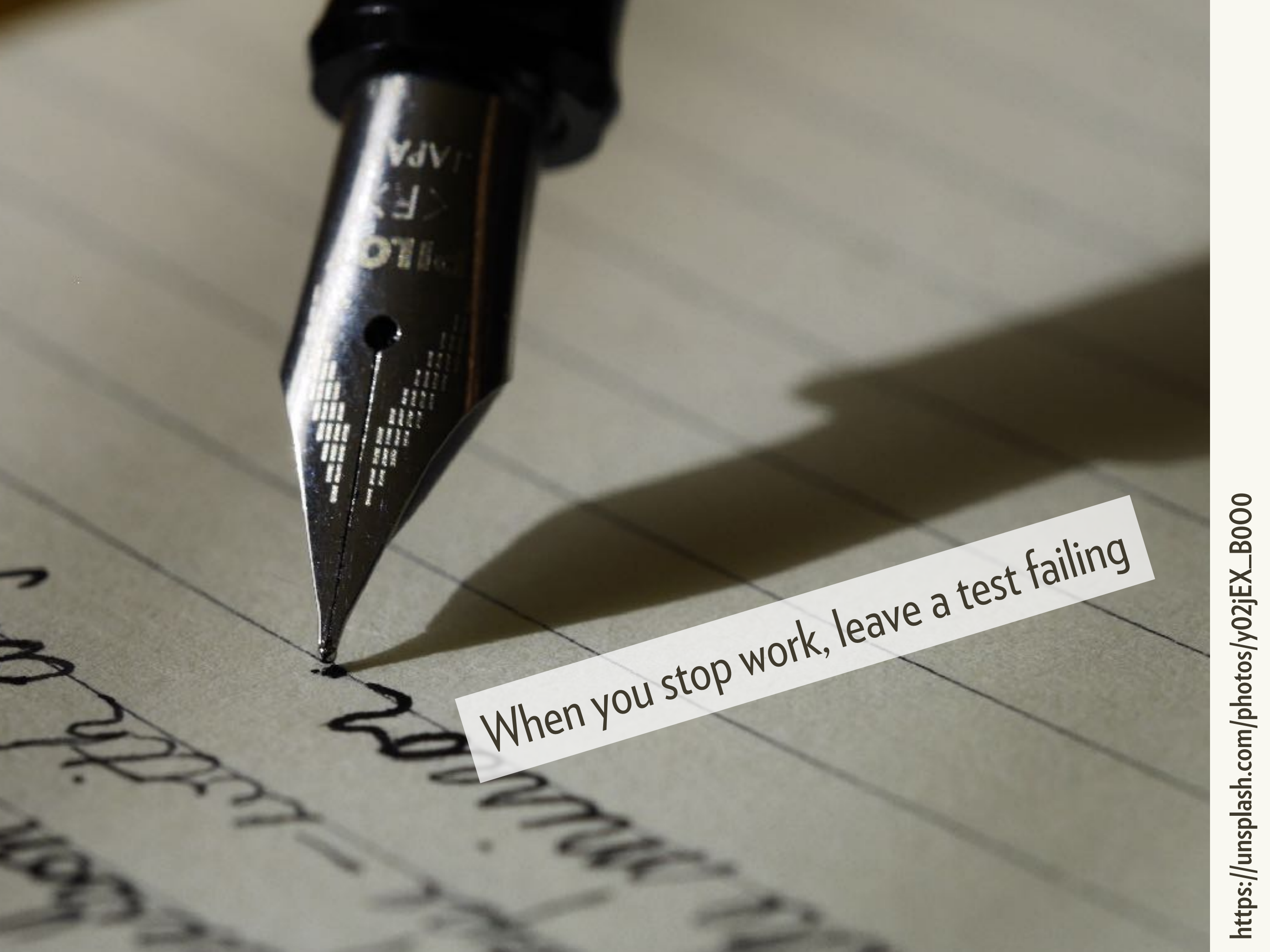
Other advantages



Writing tests improves your API



Improve readability or performance
without changing behaviour.



When you stop work, leave a test failing

add_col

Next challenge is to implement add_col

```
df <- data.frame(x = 1)
```

```
add_col(df, "y", 2, where = 1)
```

```
add_col(df, "y", 2, where = 2)
```

```
add_col(df, "x", 2)
```

Two expectations cover 80% of cases

`expect_equal(obj, exp)`

`expect_error(code, regexp)`

You'll learn others throughout the course.

Complete list at

<http://testthat.r-lib.org/reference>

Make these tests pass

Back in 20 mins

```
# use_test("add_col")
test_that("where controls position", {
  df <- data.frame(x = 1)

  expect_equal(
    add_col(df, "y", 2, where = 1),
    data.frame(y = 2, x = 1)
  )
  expect_equal(
    add_col(df, "y", 2, where = 2),
    data.frame(x = 1, y = 2)
  )
})
# Some hints on next slide
```

Hints

```
# Start by establishing basic form of the  
# function and setting up the test case.  
add_col <- function(x, name, value, where) {  
  
}
```

```
# Make sure that you can Cmd + Shift + T  
# and get two test failures before you  
# continue
```

```
# More hints on the next slide
```

More hints

```
# You'll need to use add_cols
```

```
# add_cols() takes two data frames and
```

```
# you have a data frame and a vector
```

```
# setNames() lets you change the names of
```

```
# data frame
```


My solution

```
add_col <- function(x, name, value, where) {  
  df <- setNames(data.frame(value), name)  
  add_cols(x, df, where = where)  
}
```

Make this test pass

```
test_that("can replace columns", {  
  df <- data.frame(x = 1)  
  
  expect_equal(  
    add_col(df, "x", 2, where = 2),  
    data.frame(x = 2)  
  )  
})
```

My solution

```
add_col <- function(x, name, value, where) {  
  if (name %in% names(x)) {  
    x[[name]] <- value  
    x  
  } else {  
    df <- setNames(data.frame(value), name)  
    add_cols(x, df, where = where)  
  }  
}
```

Make this test pass

```
test_that("default where is far right", {  
  df <- data.frame(x = 1)  
  
  expect_equal(  
    add_col(df, "y", 2),  
    data.frame(x = 1, y = 2)  
  )  
})
```

My solution

```
add_col <- function(x, name, value,
                    where = ncol(x) + 1) {
  if (name %in% names(x)) {
    x[[name]] <- value
    x
  } else {
    df <- setNames(data.frame(value), name)
    add_cols(x, df, where = where)
  }
}
```

Can we use `add_col()` to **remove** columns?

```
df <- data.frame(x = 1, y = 2)
```

```
expect_equal(  
  add_col(df, "x", NULL)  
  data.frame(y = 2)  
)
```

```
# Should we? If not, what should add_col()  
# do when value is NULL? Would a separate  
# remove_col() be a good idea?
```

Can we use `add_col()` to **move** columns?

```
df <- data.frame(x = 1, y = 2)

expect_equal(
  add_col(df, "x", 1, where = 2)
  data.frame(y = 2, x = 2)
)
```

Should we?

Would `move_col()` be better?

How should we name this collection of functions?

Prefix?

add_col()

move_col()

remove_col()

Suffix?

col_add()

col_remove()

col_move()

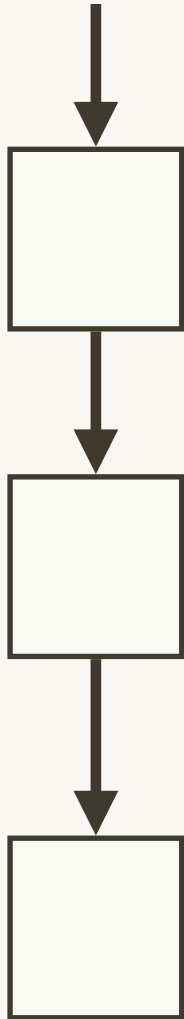
Fail fast

What about bad inputs?

```
# We need to test for errors too  
add_cols(df1, df2, where = 0)  
add_cols(df1, df2, where = NA)  
add_cols(df1, df2, where = 1:10)  
add_cols(df1, df2, where = "a")
```

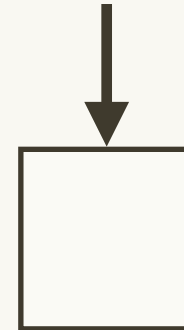
For robust code, fail early

Bad input



Uninformative error

Bad input



Useful error

We could add to `add_cols` directly

```
add_cols <- function(x, y, where = 1) {  
  if (!is.numeric(where) || length(where) != 1) {  
    stop("`where` is not a number", call. = FALSE)  
  } else if (where == 0 || is.na(where)) {  
    stop("`where` must not be 0 or NA", call. = FALSE)  
  } else if (where == 1 || where <= -ncol(x)) {  
    cbind(x, y)  
  } else if (where >= ncol(x) || where == -1) {  
    cbind(y, x)  
  } else {  
    if (where < 0) where <- nrow(x) + where  
    cbind(x[1:where], y, x[where:nrow(x)])  
  }  
}
```

But this confuses the intent of add_cols

```
# Better to have one function responsible  
# for checking for valid inputs
```

```
check_where <- function(where) {  
  ...  
}
```

```
# This also makes it easier to test because  
# it's independent of add_cols
```

Your turn

```
# Write down the error message that you think  
# each of these lines should generate
```

```
check_where(0)  
check_where(NA)  
check_where(1:10)  
check_where("a")
```

Error message structure

1. Problem statement
(use must or can't)
2. Error location
(where possible)
3. Hint
(if common)

Punctuation

- Always use `call. = FALSE`
- Surround variable names in ``...``, and strings in `'...'`
- Sentence case

Your turn

Write `check_where()`. It should return an integer or throw an error. I suggest you put in the same file as `add_cols()`.

My answer

```
check_where <- function(x) {  
  if (length(x) != 1 || !is.numeric(x)) {  
    stop("`where` must be a length one numeric vector.", call. = FALSE)  
  }  
  x <- as.integer(x)  
  
  if (x == 0 || is.na(x)) {  
    stop("`where` must not be zero or missing", call. = FALSE)  
  } else {  
    x  
  }  
}
```

Test coverage

Test coverage shows you what you've tested

```
devtools::test_coverage()
```

Use `expect_error()` to test for errors

```
expect_error(  
  check_where("a")  
)
```

```
expect_error(  
  check_where("a"),  
  "not a number"  
)
```



A regular expression

Your turn

Write tests to ensure that `check_where()` only allows valid inputs. (Where should the tests live?)

My tests

```
# check_where() lives in same file as add_cols()
# so tests should live in test-add_cols()

test_that("where must be valid value", {
  expect_error(check_where("a"), "length one numeric vector")
  expect_error(check_where(1:10), "length one numeric vector")

  expect_error(check_where(0), "not be zero or missing")
  expect_error(check_where(NA_real_), "not be zero or missing")
})
```


This work is licensed under the
Creative Commons Attribution-Noncommercial 3.0
United States License.

To view a copy of this license, visit
<http://creativecommons.org/licenses/by-nc/3.0/us/>