

# Unit testing

**September 2020**

Charlotte Wickham & Sara Altman

Adapted from *Tidy Tools* by Hadley Wickham



# Getting help

**slido:** Ask a question in the Q&A at anytime. Vote on questions.



## In Breakout Rooms:

1. Ask your roommates
2. If your room is stuck, change your status to "Send Help" in the Google Doc

**Zoom chat:** Reserved for urgent technical matters (e.g., "we can't hear you")

# Overview

1. Motivation
2. Testing workflow
3. Test coverage
4. Test-driven development

# Motivation

Goal: Write a function that adds one data frame to another at specified position.

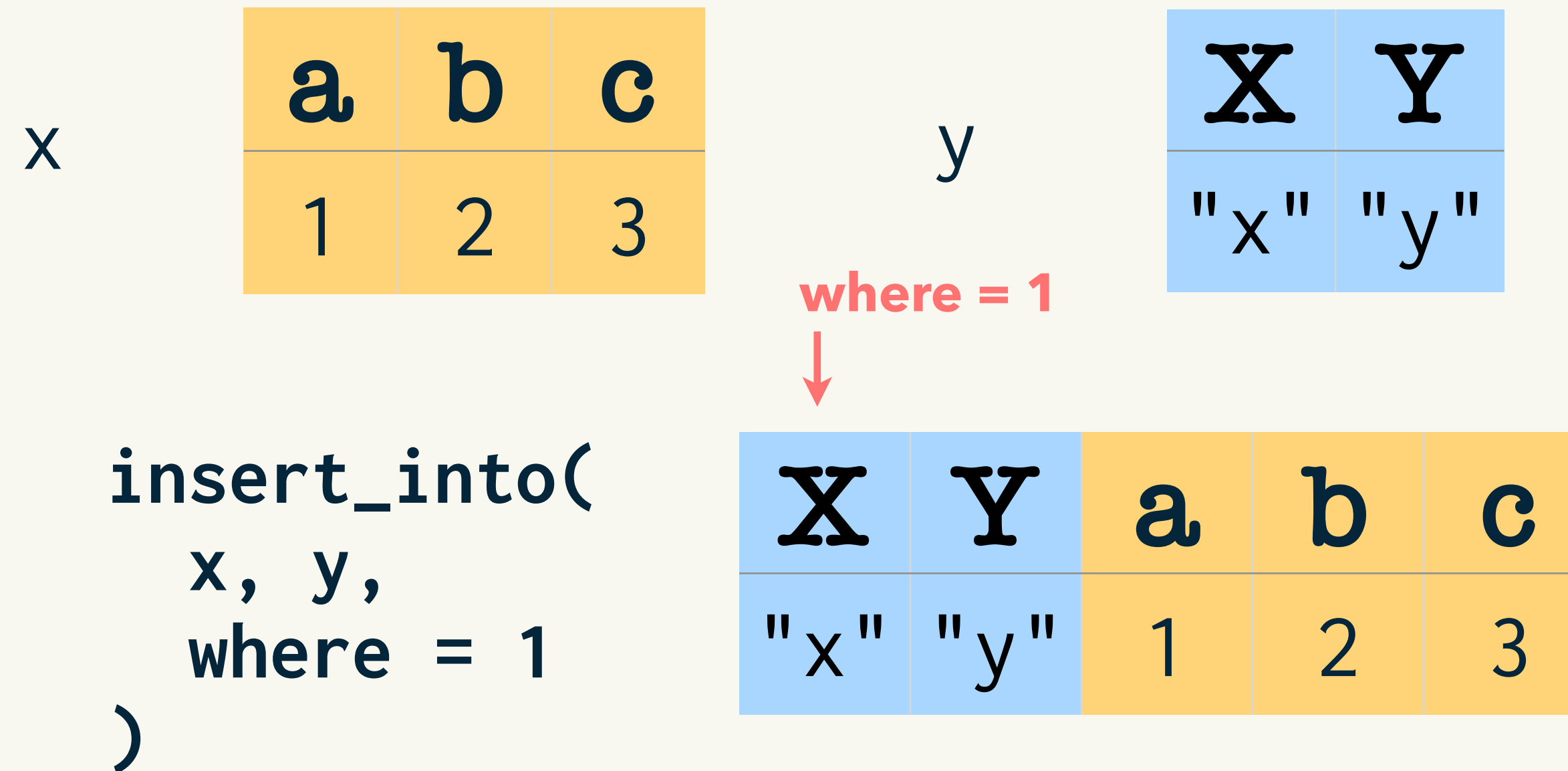
# insert\_into()

Add the columns of **y** to **x** at position **where**

x	a	b	c	y	X	Y
	1	2	3		"x"	"y"

# insert\_into()

Add the columns of **y** to **x** at position **where**



# insert\_into()

Add the columns of **y** to **x** at position **where**

x	a	b	c
	1	2	3

y	X	Y
	"x"	"y"

where = 1  
↓

```
insert_into(  
  x, y,  
  where = 1  
)
```

X	Y	a	b	c
"x"	"y"	1	2	3

```
insert_into(  
  x, y,  
  where = 2  
)
```

a	X	Y	b	c
1	"x"	"y"	2	3

↑  
where = 2



# Your turn: Breakout rooms

Follow the steps on the **following slides** to:

1. Finish writing `insert_into()`.
2. Let me know when you're done.

7 minutes

Take a note of the  
slide number

**Slides:** <http://bit.ly/build-tt>

# Your turn: 1. Finish insert\_into()

```
# Copy the code into RStudio and finish the function
# Add the columns of y to x at position where
# Hint: cbind() will be useful
insert_into <- function(x, y, where = 1) {
  if (where == 1) { # first col
    ...
  } else if (where > ncol(x)) { # last col
    ...
  } else {
    ...
  }
}
```

No need to make a new package yet—just write the function in a new R Script.

# Your turn: 2. Ready to move on?

- # Check in with your breakout room.
- # Is everyone ready to move on?
- # If not, help them out!
- # If yes, edit your status in the Google Doc:
- # [bit.ly/built-tt-breakout](http://bit.ly/built-tt-breakout)

- # While you wait:
- # Make sure your function behaves as you expect.
- # What's your process for doing so?

**Slides:** <http://bit.ly/build-tt>

# Problems with this approach

**Tedious**

**Error-prone**

**Frustrating**

**Let your computer do what it's good at.**

# A better workflow

Put code in R/ and use devtools::**load\_all()**

Write unit tests and use devtools::**test\_file()**

# Testing workflow

<https://r-pkgs.org/tests.html>

# 1. Create a package

```
usethis::create_package("~/Desktop/addcol")
```

```
usethis::use_r("insert_into")
```

```
insert_into <- 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])  
  }  
}
```

Code goes in  
insert\_into.R



## 2. Set up testing infrastructure

Key infrastructure

`usethis::use_test()`

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

Creates test file  
matching script file

### 3. Run tests

`devtools::test()`

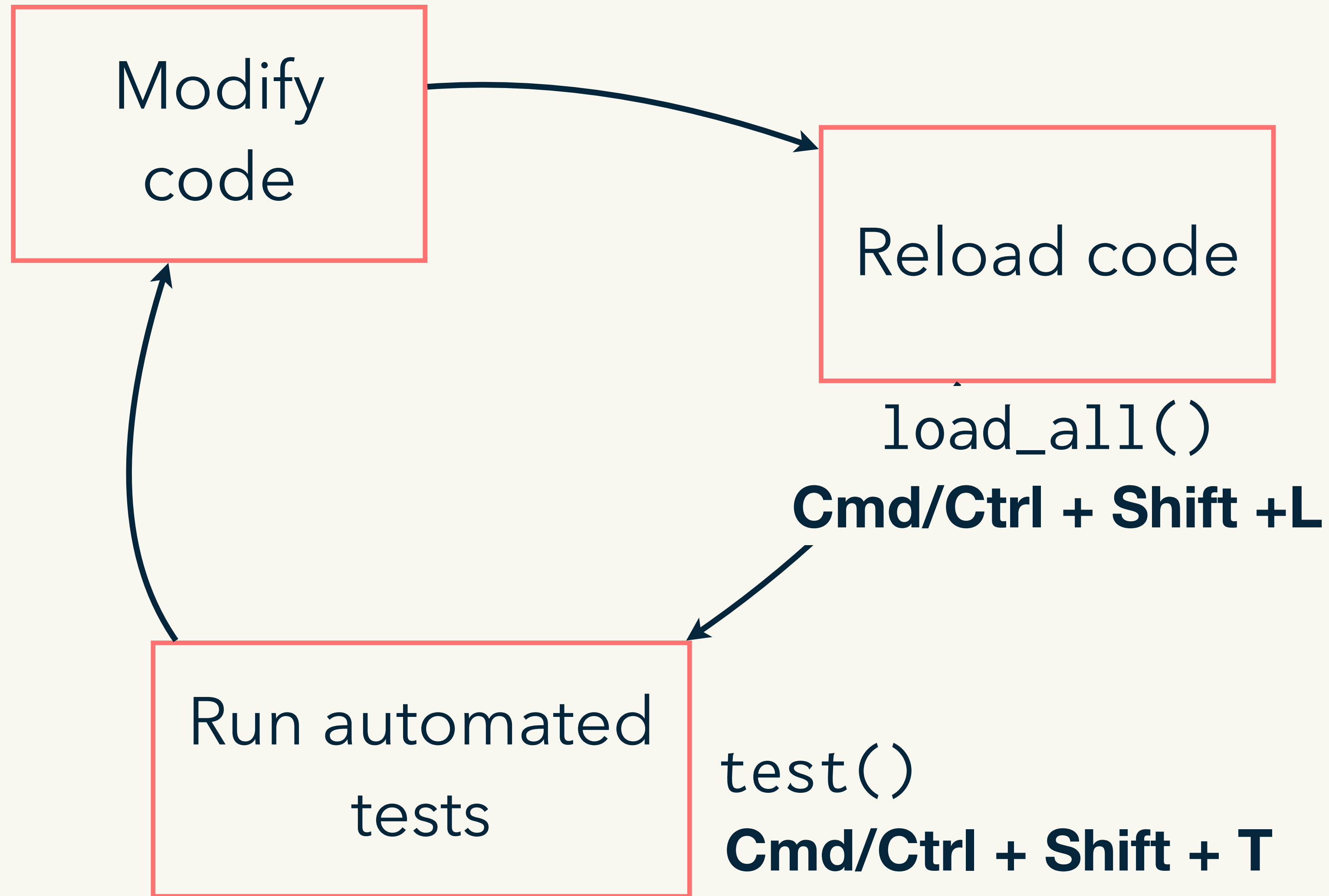
Runs all tests

# or:

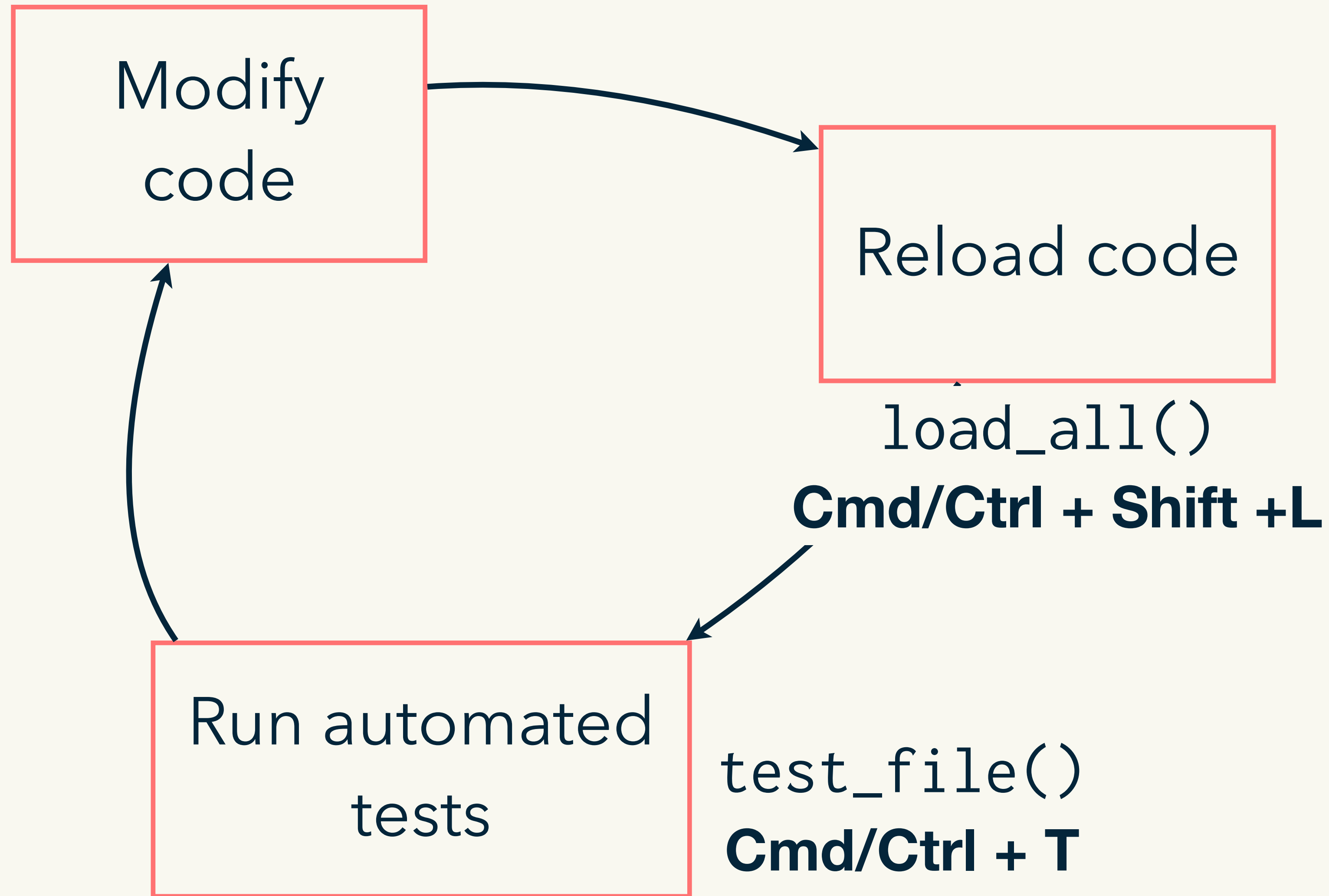
`devtools::test_file()`

Runs tests for  
current file

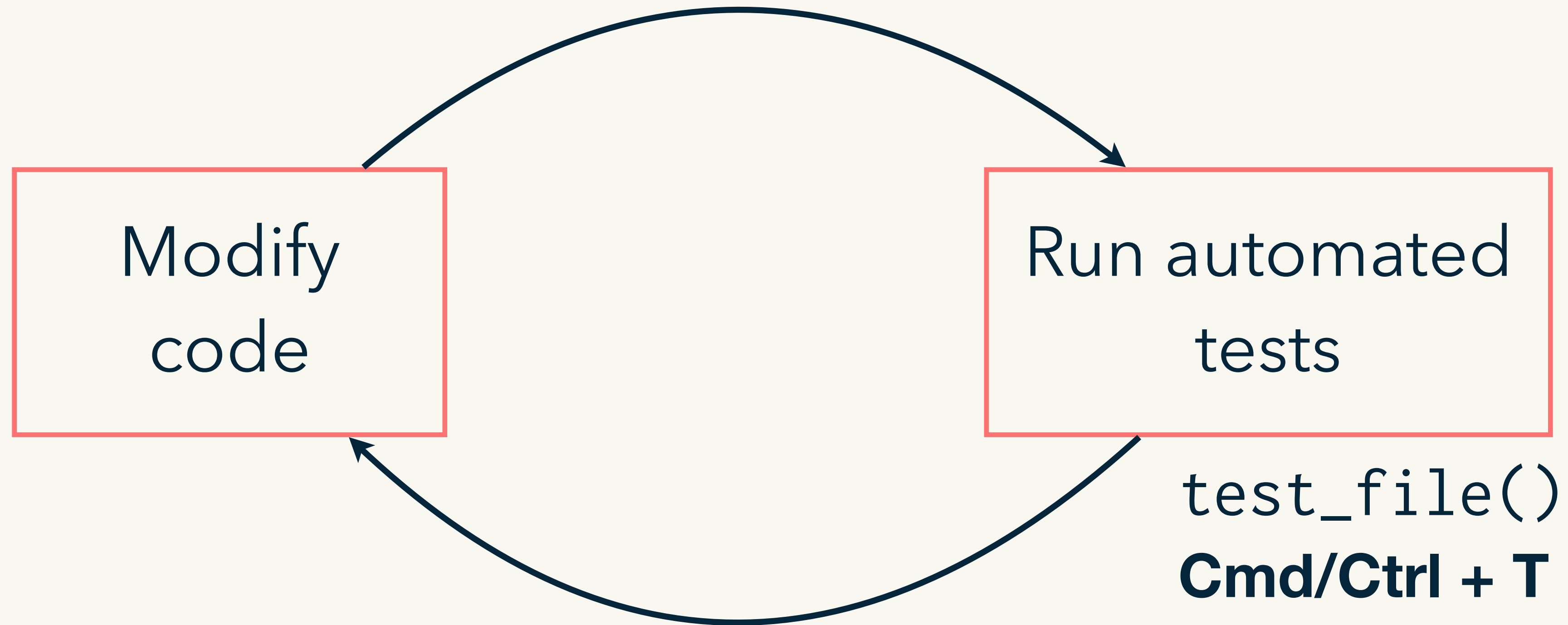
# New workflow with testthat



# New workflow with testthat



# But why reload the code?



# Your turn: Breakout Rooms

Follow the steps on the **following slides** to:

1. Create a package called addcol.
2. Add the function insert\_into().
3. Setup your keyboard shortcuts.
4. Practice the testing workflow.

10 min

**Slides:** <http://bit.ly/build-tt>

Your turn: 1. Create a package

Recall: Create a package called addcol.

# Your turn: 2. Add insert\_into()

Create an R file for `insert_into()`, then paste in the code on the right.

Run `devtools::load_all()`, then confirm that `insert_into()` works.

```
insert_into <- 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])  
  }  
}
```



# Your turn: 3. Setup keyboard shortcuts.

*Tools > Modify Keyboard Shortcuts*

Install Packages...  
Check for Package Updates...  
Version Control  
Shell...  
Terminal  
Jobs  
Addins  
Keyboard Shortcuts Help  
**Modify Keyboard Shortcuts...**  
Project Options...  
Global Options...

Keyboard Shortcuts

Show: ☒ All ☐ Customized

Name	Shortcut	Scope
Calculate package test coverage	Ctrl+Shift+C	Addin
Compare test results for Shiny application		Workbench
Record a test for Shiny		Workbench
Report test coverage for a file	Cmd+R	Addin
Report test coverage for a package	Shift+Cmd+R	Addin
Run Test		Workbench
Run Tests		Workbench
Run a test file	Cmd+T	Addin
Run tests for Shiny application		Workbench
Test Package	Shift+Cmd+T	Package Development
View Latest Run		Addin

Reset...  
Apply  
Cancel

# Your turn: 4. Practice the workflow

```
# Setup testing infrastructure
usethis::use_test()
# No need to edit the test file yet!

# Run tests
devtools::test_file()
# If the checks Pass, you'll see GREEN.
```

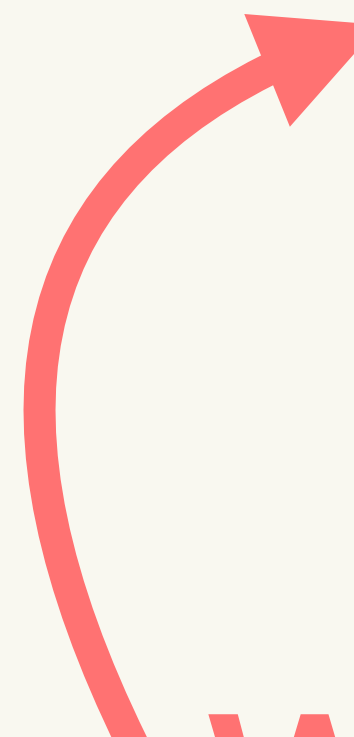
# Your turn: Ready to move on?

- # Check in with your breakout room.
- # Is everyone ready to move on?
- # If not, help them out!
- # If yes, edit your status in the Google Doc:
- # [bit.ly/built-tt-breakout](https://bit.ly/built-tt-breakout)
  
- # While you wait:
- # Can you make the test fail?

Questions?

# Our workflow so far

1. `create_package()`
2. `use_r("file_name")`
3. `use_test()`
4. `test_file()` (Cmd/Ctrl + T)



**Write tests!**

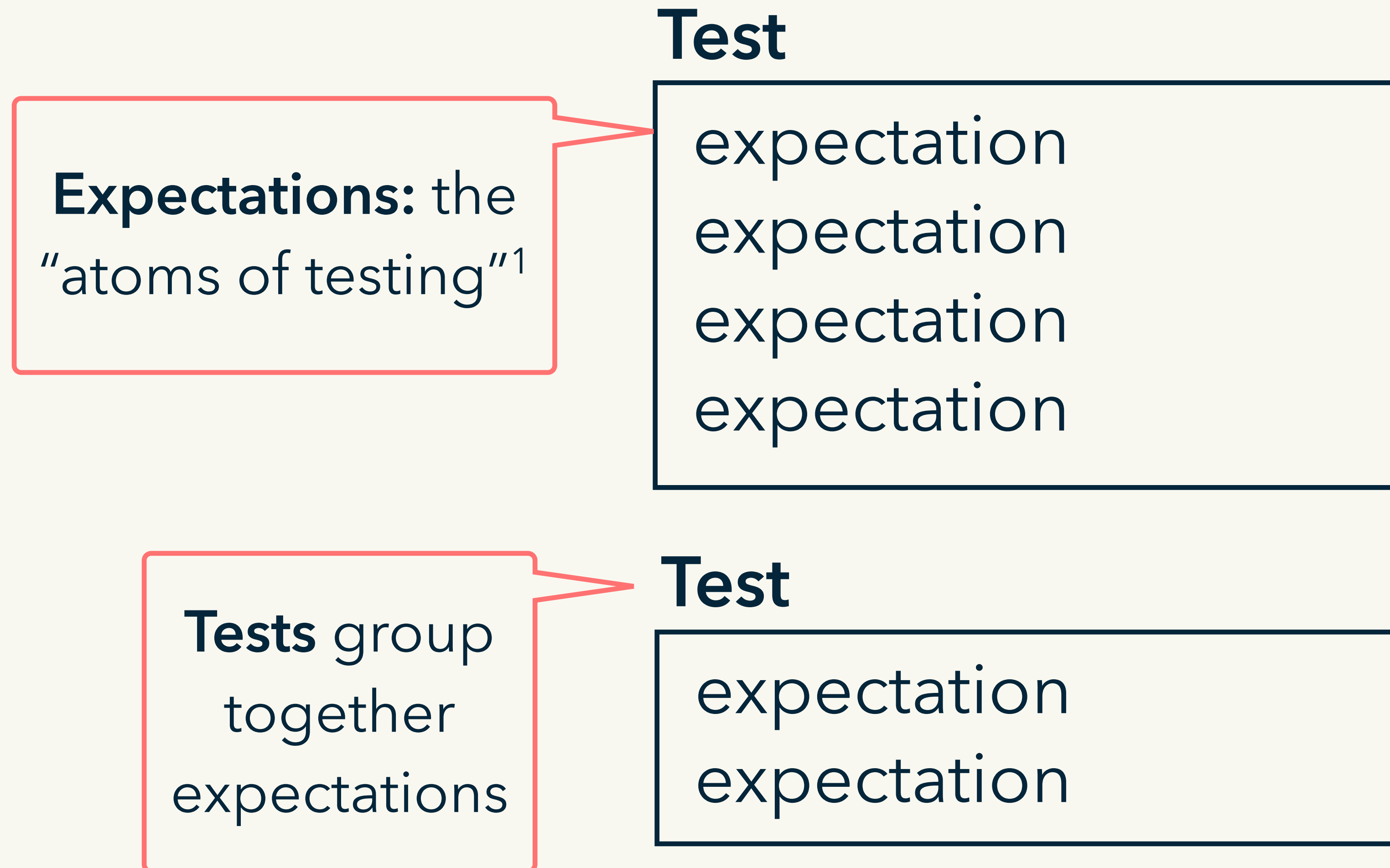
**Computers need humans to set  
expectations.**

# Tests are organized in three layers

**Expectations:** the  
"atoms of testing"<sup>1</sup>

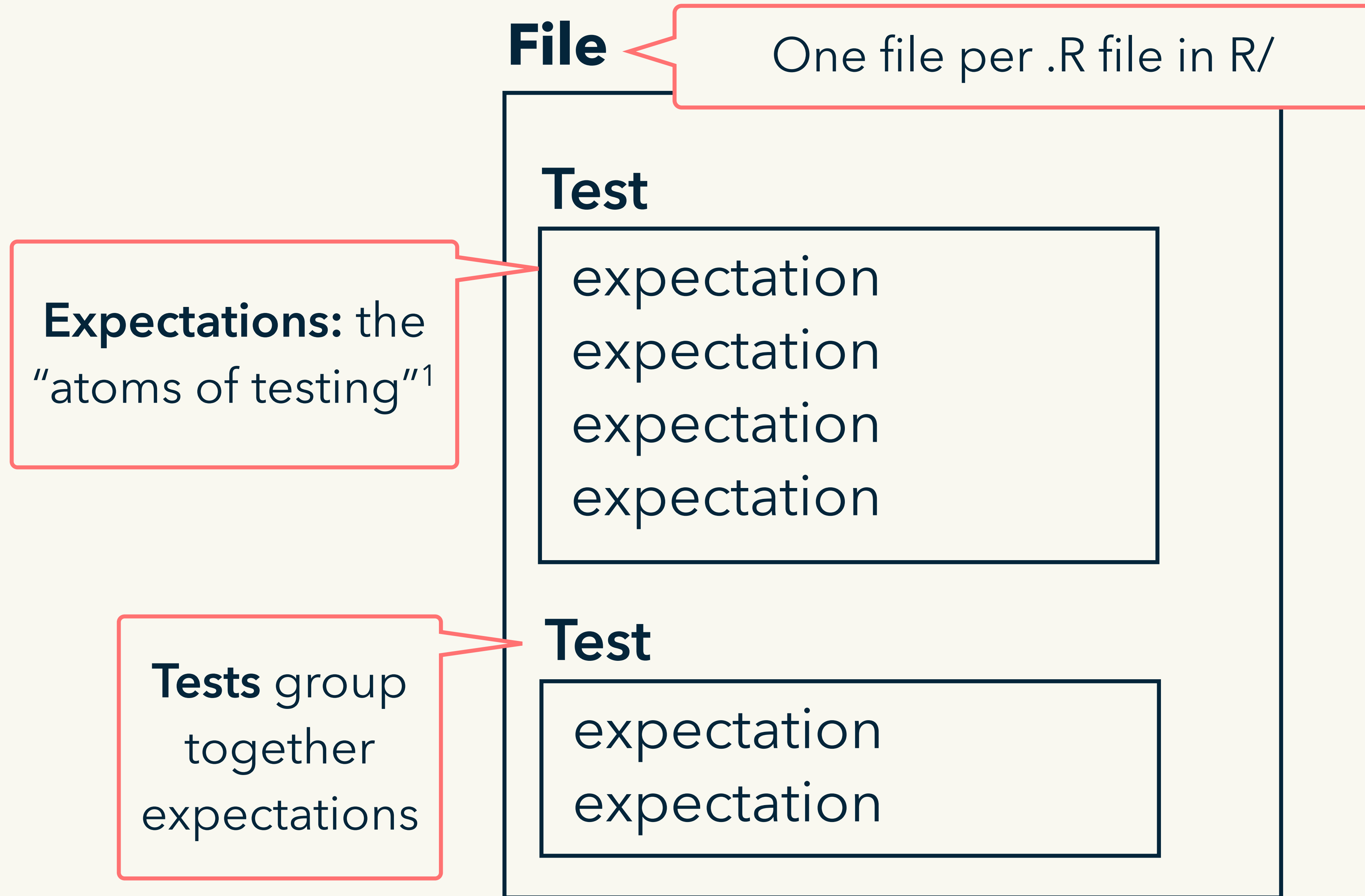
expectation  
expectation  
expectation  
expectation

# Tests are organized in three layers





# Tests are organized in three layers



# Expectations

**expect\_named**(OBJECT, EXPECTATION)

Describes an expected  
property of the output

# Expectations

Object (output of  
function)

Expected vector  
of column names

```
expect_named(insert_into(df1, df2, where = 1), c("X", "Y", "a", "b", "c"))
```

Describes an expected  
property of the output

# Automate!

Helper function to  
reduce duplication

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

Expected vector  
of column names

```
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"))
```

Describes an expected  
property of the output

# Automate!

```
at_pos <- function(i) {  
  insert_into(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"))
```

Easy to see the pattern

Most important expectation

**`expect_equal(obj, exp)`**

**`expect_equal(my_function(x, y), 1)`**

`expect_*()` functions:  
**<http://testthat.r-lib.org>**

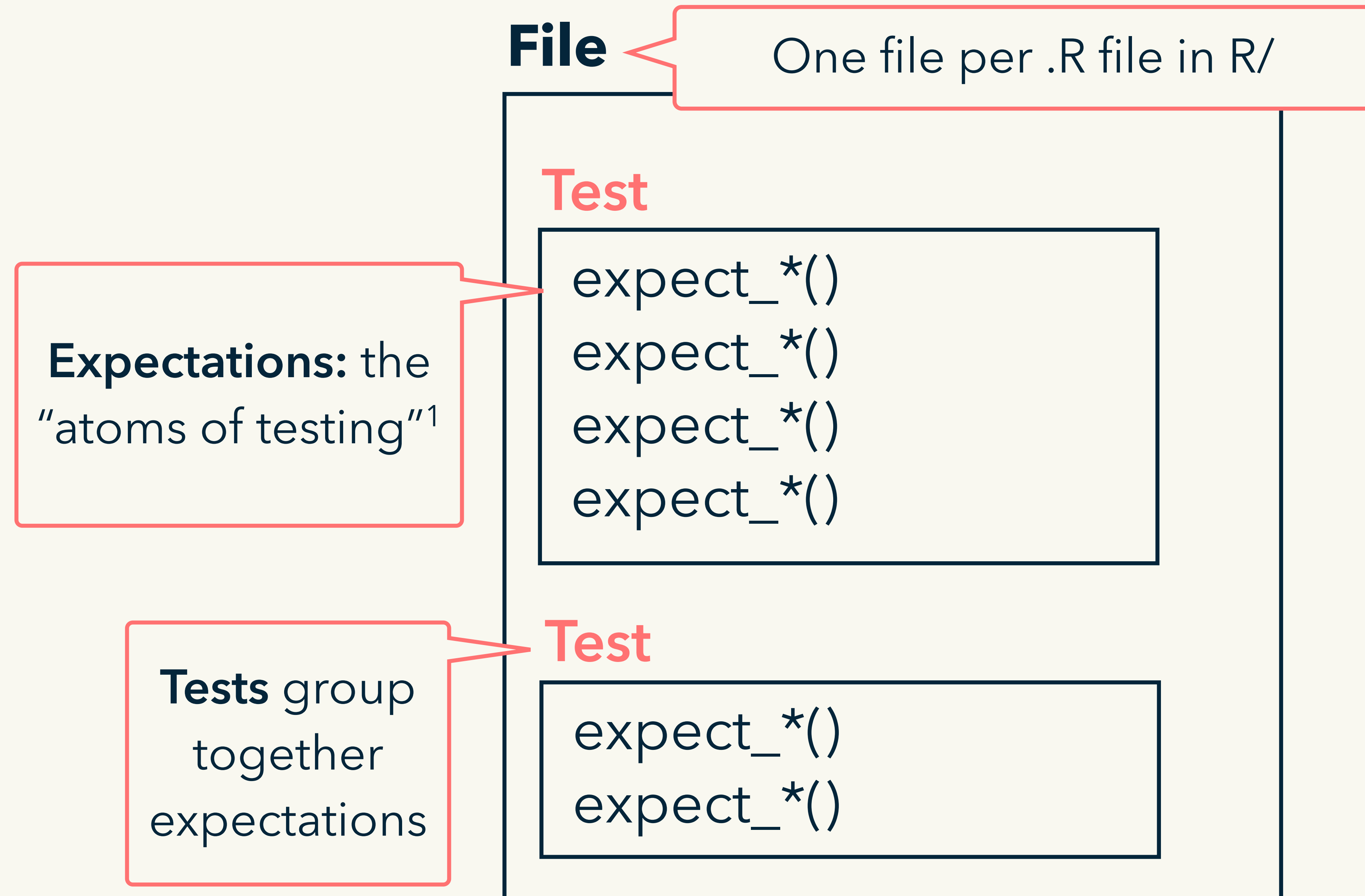


Join at  
**slido.com**  
**#80875**

🔑 Passcode: tidytools



# Tests are organized in three layers





# testthat tests

Complete the  
sentence: "Tests that  
the function..."

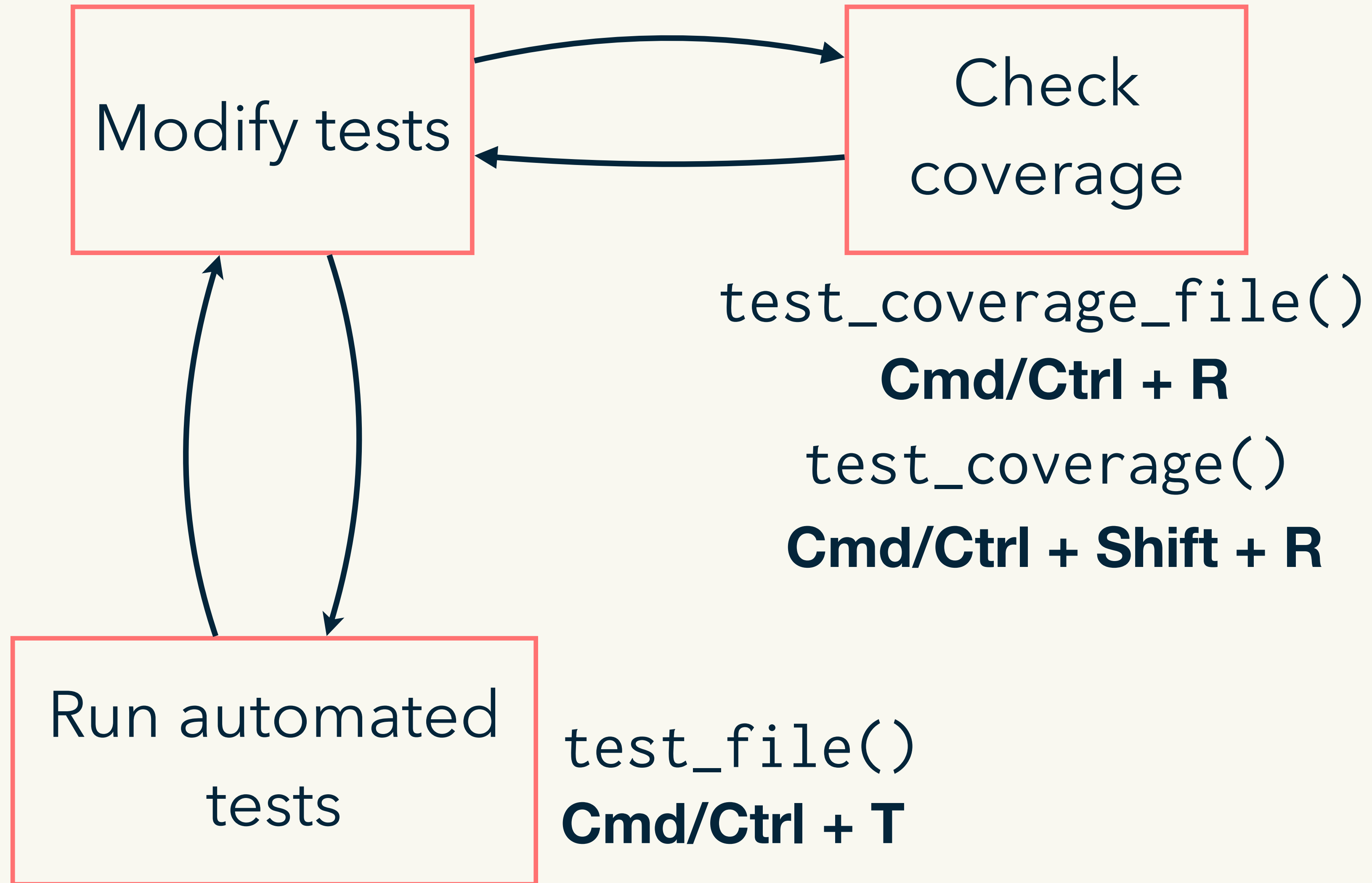
```
test_that("can insert columns at any position", {  
  at_pos <- function(i) {  
    insert_into(df1, df2, where = i)  
  }  
})
```

```
  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"))  
})
```

# Test coverage

<https://covr.r-lib.org>

# Guide tests with coverage



# Your turn: Breakout Rooms

Follow the steps on the **following slides** to:

1. Add a test for `insert_into()`.
2. Check the test coverage.
3. Add additional tests.

10 min

**Slides:** <http://bit.ly/build-tt>

# Your turn: 1. Add a test for insert\_into()

```
# Copy the following tests into the file created by use_test()
test_that("can add column at any position", {
  df1 <- data.frame(a = 3, b = 4, c = 5)
  df2 <- data.frame(X = "x", Y = "y")
  at_pos <- function(i) {
    insert_into(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"))
})

# Run test file (Cmd/Ctrl + T)
devtools::test_file()

# Do the tests pass?
```

# Your turn: 2. Check test coverage

```
devtools::test_coverage_file()  
# Cmd/Ctrl + R  
# Are all the lines covered (green)?  
  
# If not add a test for the missing case  
  
# Check you now cover all cases
```

## Your turn: 3. Another test

What does `insert_into()` do if `x` and `y` have different numbers of rows?

Is that the behavior you want?

If not, alter your code.

Either way, write a test to check that the function behaves as expected.

# Your turn: Ready to move on?

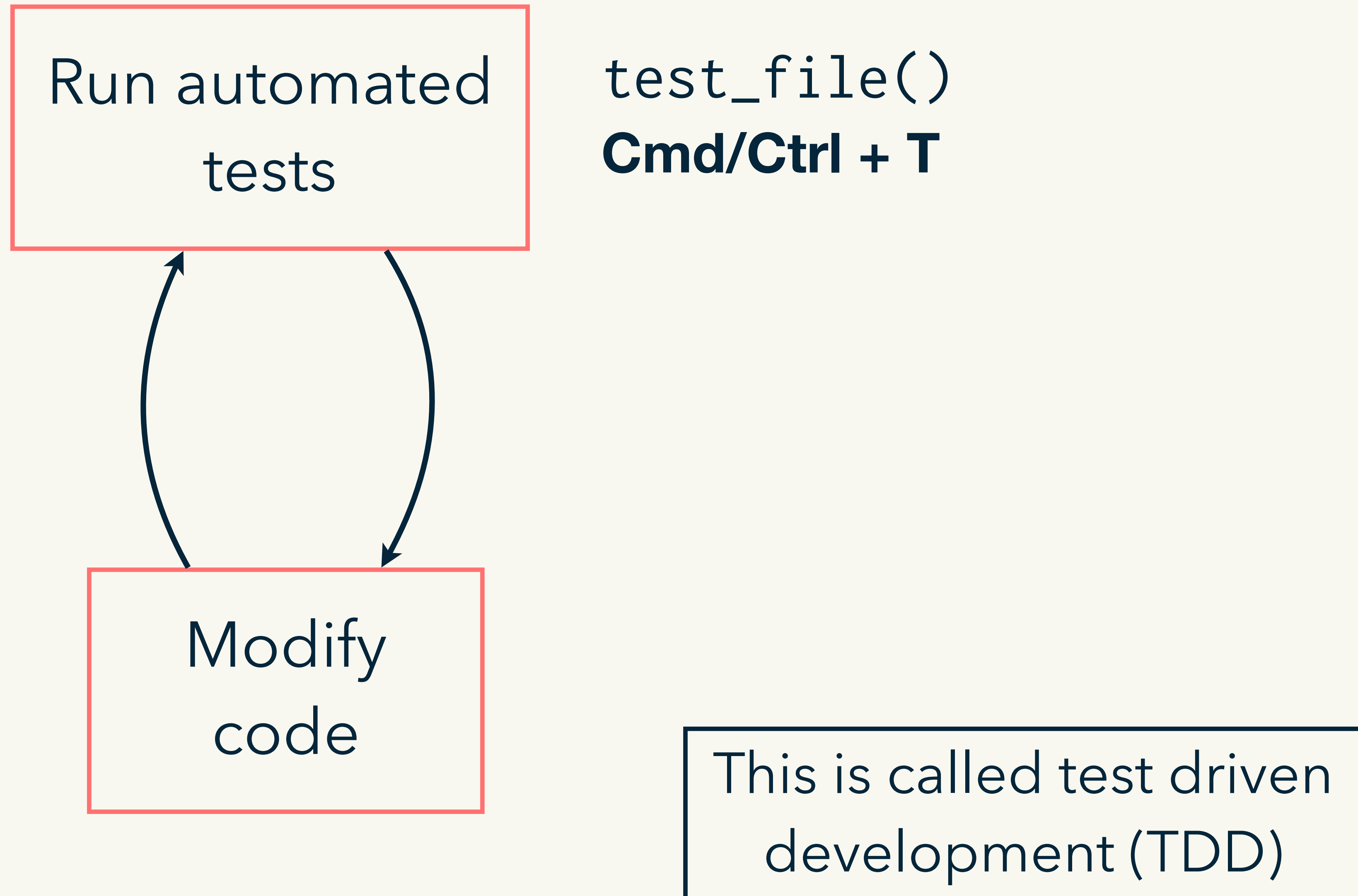
```
# Check in with your breakout room.  
# Is everyone ready to move on?  
# If not, help them out!  
# If yes, edit your status in the Google Doc:  
# bit.ly/built-tt-breakout  
  
# While you wait:  
# Write additional tests for insert_into().
```



Questions?

# Test-driven development

# Or you might start with the tests



# add\_col()

Helper: insert\_into()

```
df1 <- data.frame(x = 1)
df2 <- data.frame(y = "a")
```

```
insert_into(
  x = df1, # data frame
  y = df2, # data frame
  where = 1
)
```

add\_col()

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

add_col(
  df, # data frame
  "y", # new column name
  "a", # vector of new column value(s)
  where = 1
)
```

# Your turn: Breakout Rooms

Follow the steps on the **following slides** to:

1. Make test #1 pass.
2. Make test #2 pass.
3. Make test #3 pass.

20 min

**Slides:** <http://bit.ly/build-tt>

# Your turn: 1. Make these tests pass

```
# Set up testing for add_col()
usethis::use_test("add_col")

# Add to test-add_col.R, then make the tests pass.
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)
  )
})
```

Hints on the next  
two slides

# Hint: getting started

```
usethis::use_r("add_col")
```

```
# In R/add_col.R
```

```
# 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 + T
```

```
# and get two test failures before you
```

```
# continue
```

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

# Hint: add\_col()

```
# You'll need to use insert_into()
```

```
# insert_into() takes two data frames and  
# you have a data frame and a vector
```

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



# Your turn: 2. Make this test pass

```
# Add me to test-add_col.R
test_that("can replace columns", {
  df <- data.frame(x = 1)

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

# Your turn: 3. Make this test pass

```
# Add me to test-add_col.R
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)
  )
})
```

# Your turn: Ready to move on?

```
# Check in with your breakout room.  
# Is everyone ready to move on?  
# If not, help them out!  
# If yes, edit your status in the Google Doc:  
# bit.ly/built-tt-breakout  
  
# While you wait:  
# Deal with bad inputs to add_col(). For example:  
# insert_into(df1, df2, where = 0)  
# insert_into(df1, df2, where = NA)  
# insert_into(df1, df2, where = 1:10)
```

# 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)
    insert_into(x, df, where = where)
  }
}
```

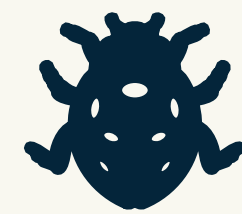
Questions?

# 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)
    insert_into(x, df, where = where)
  }
}
```

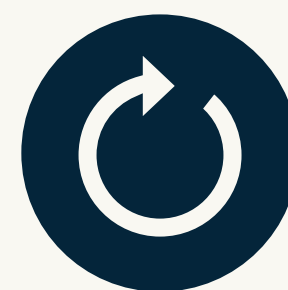
# Other advantages

Fewer bugs.





Improve readability and performance  
without changing behavior.



Try leaving a test  
failing when you end  
a work session.

Easier for you.



# Summary

## Setup

Create R file  
`use_r()`

Create test file  
`use_test()`

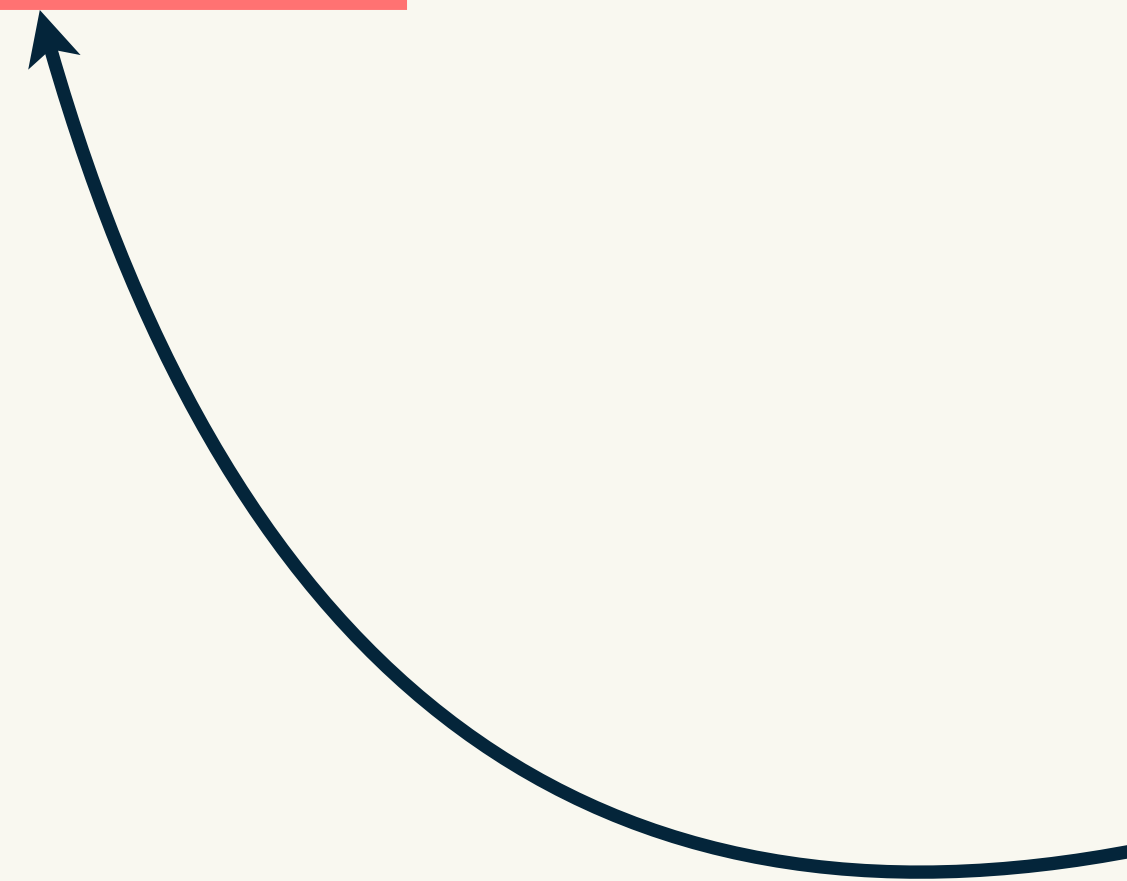


Modify  
code

Write tests

with `test_that()` and  
`expect_*()` functions

Run automated tests  
`test_file()`  
**Cmd/Ctrl + T**



# End-of-day survey on slido



Join at  
**slido.com**

**#80875**

---

🔑 Passcode:  
**tidytools**

This work is licensed as  
Creative Commons  
Attribution-ShareAlike 4.0  
International

To view a copy of this license, visit  
[https://creativecommons.org/  
licenses/by-sa/4.0/](https://creativecommons.org/licenses/by-sa/4.0/)