Unit testing

September 2020

Charlotte Wickham and Sara Altman

Adapted from *Tidy Tools* by Hadley Wickham

Motivation

Goal: Write a function that allows us to add one data frame to another, at a position we choose.

insert_into()

Add the columns of **df2** to **df1** 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: insert_into()

```r
# 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 {
    ...
  }
}
```

Take a note of the slide number

A first attempt

```
insert_into <- 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:ncol(x)])
  }
}
```

A first attempt

```
insert_into <- 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:ncol(x)])
  }
}
```

Actually correct

Possible workflow

```r
# Some simple inputs
df1 <- data.frame(a = 1, b = 2, c = 3)
df2 <- data.frame(X = "x", Y = "y")

# Then each time I tweaked it, I re-ran
# these cases
insert_into(df1, df2, where = 1)
insert_into(df1, df2, where = 2)
insert_into(df1, df2, where = 3)
```

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

Your turn: Recall

1. Create a package called addcol
2. Add the code for 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])
  }
}
```

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

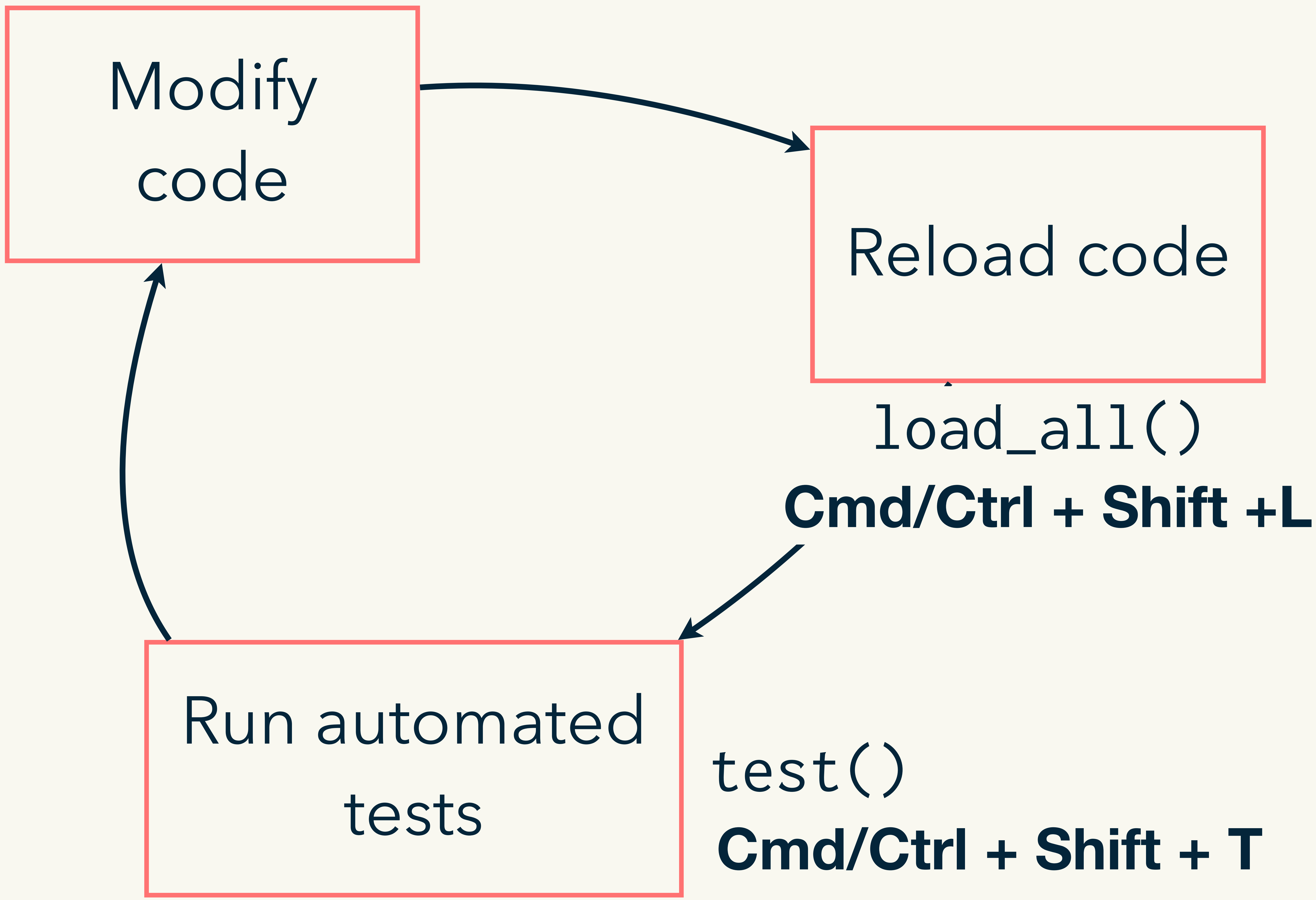
3. Run tests

```
# usethis::use_test()

devtools::test_file()
```

Runs tests for current file

New workflow with testthat



Modify code

Reload code

`load_all()`

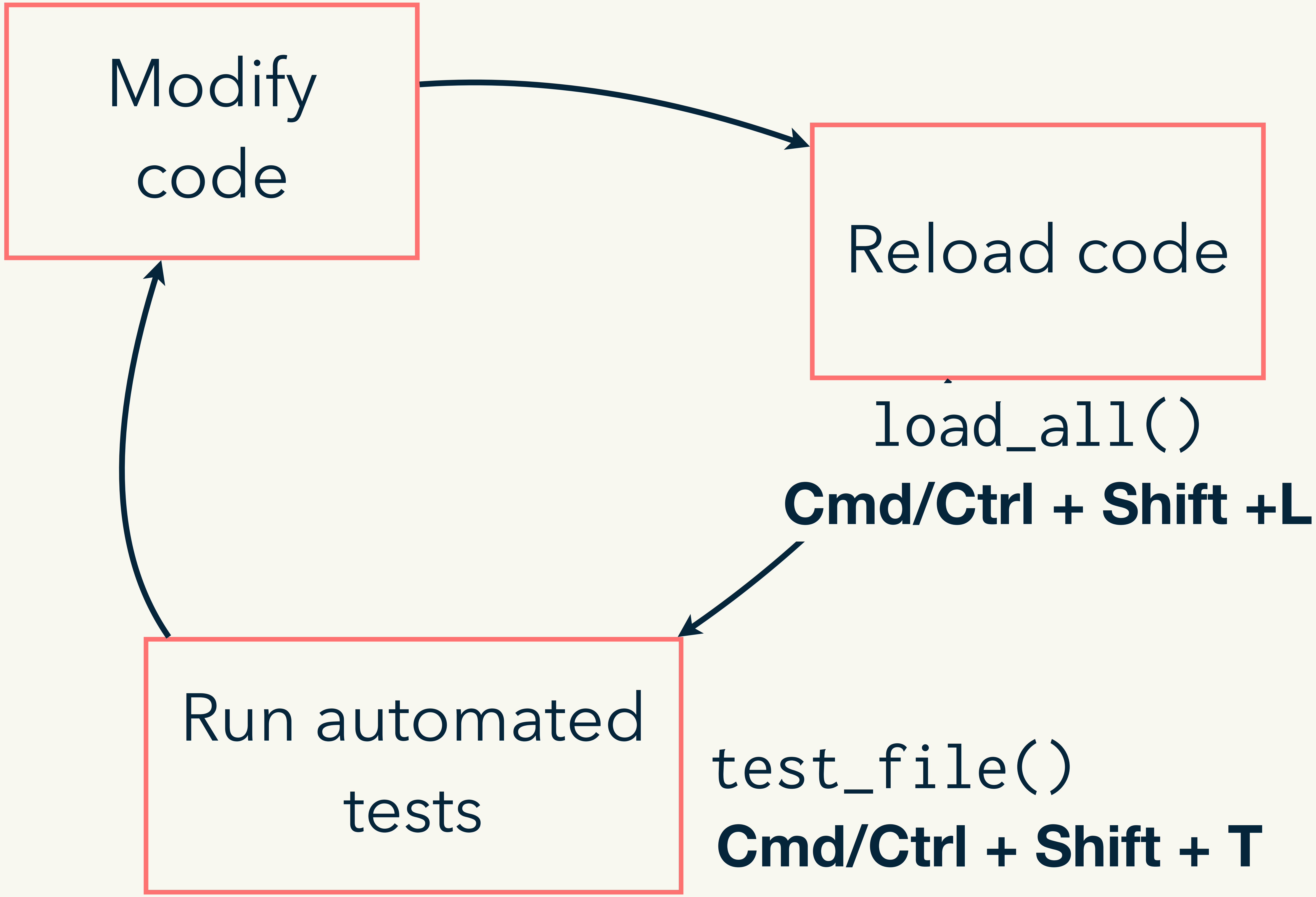
Cmd/Ctrl + Shift +L

Run automated tests

`test()`

Cmd/Ctrl + Shift + T

New workflow with testthat



Modify code

Reload code

load_all()

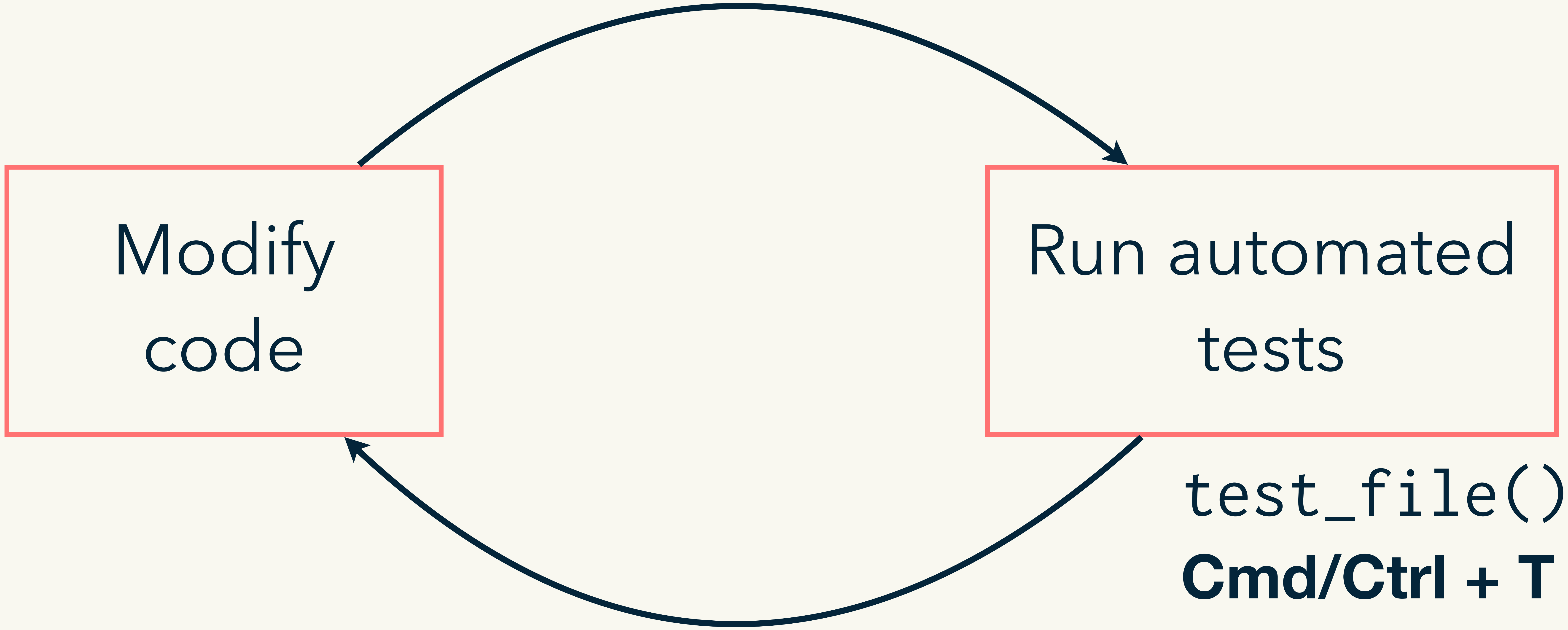
Cmd/Ctrl + Shift +L

Run automated tests

test_file()

Cmd/Ctrl + Shift + T

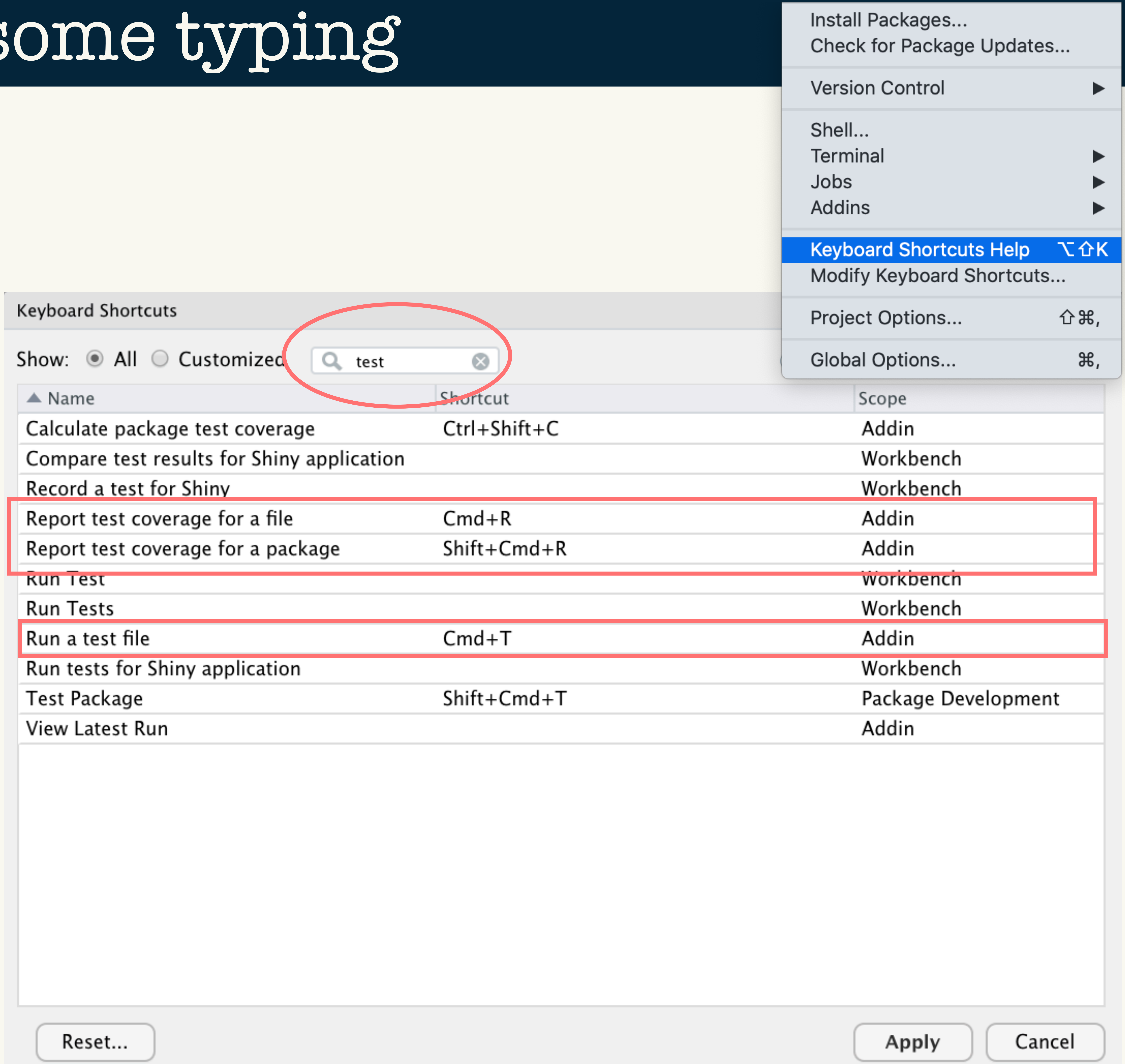
But why reload the code?



`test_file()`

Cmd/Ctrl + T

Save yourself some typing



Show: ● All ○ Customized test

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

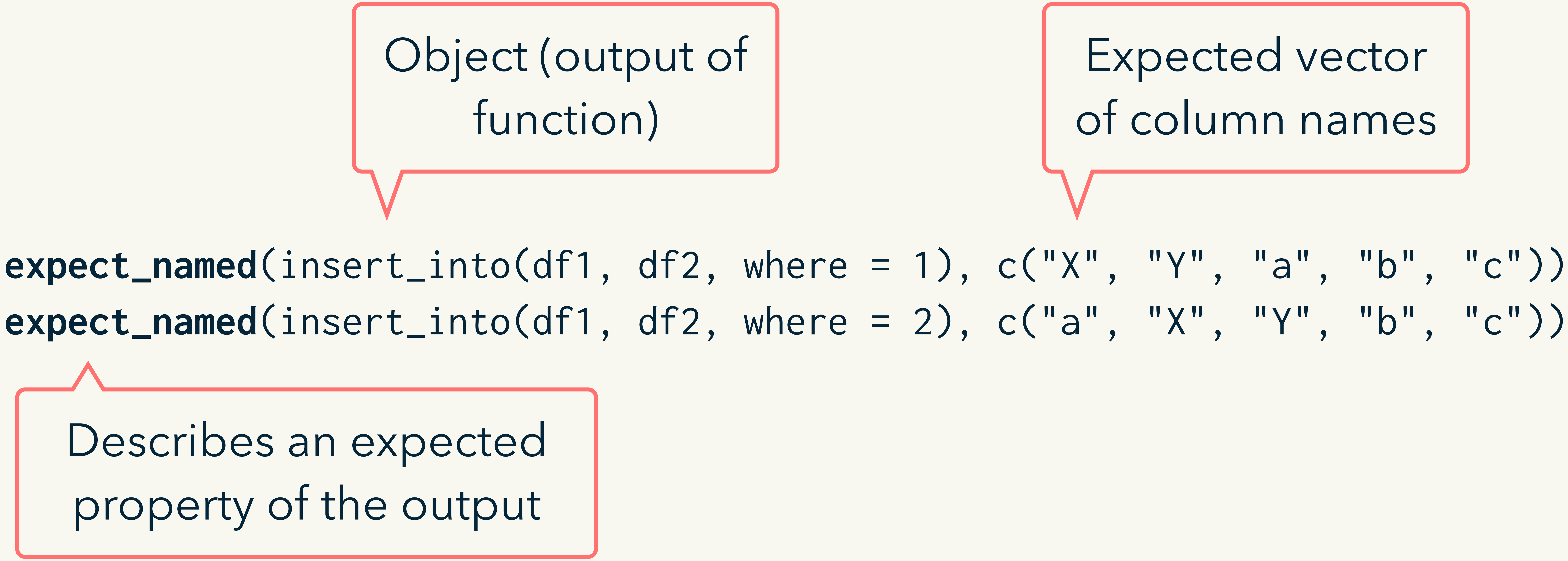
How do I write a unit test?

Computers need humans to set expectations.

Writing unit tests

```
expect_named(OBJECT, EXPECTATION)
```

Writing unit tests



Object (output of function)

Expected vector of column names

```
expect_named(insert_into(df1, df2, where = 1), c("X", "Y", "a", "b", "c"))
expect_named(insert_into(df1, df2, where = 2), c("a", "X", "Y", "b", "c"))
```

Describes an expected property of the output

Writing unit tests

Helper function to reduce duplication

```
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"))
```

Expected vector of column names

Describes an expected property of the output

Key idea of unit testing is to 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

Writing testthat tests

```
# In tests/testthat/test-insert_into.R


test_that("can add column at any position", {

  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"))
})
```

Complete the sentence: “Tests that the function…”

Your turn: Practice the workflow

```
# usethis::create_package(“~/Desktop/addcol”)
# usethis::use_r(“insert_into")
# Check all is ok with load_all()

usethis::use_test()
# Copy expectations from next next slide

devtools::test_file()
# Checks Pass: GREEN
# Run tests with keyboard shortcut (if you create it).
# Confirm that if you break insert_into() the
# tests fail.
```

Expectations

```r
# Create file with 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"))
})
```

Our workflow so far

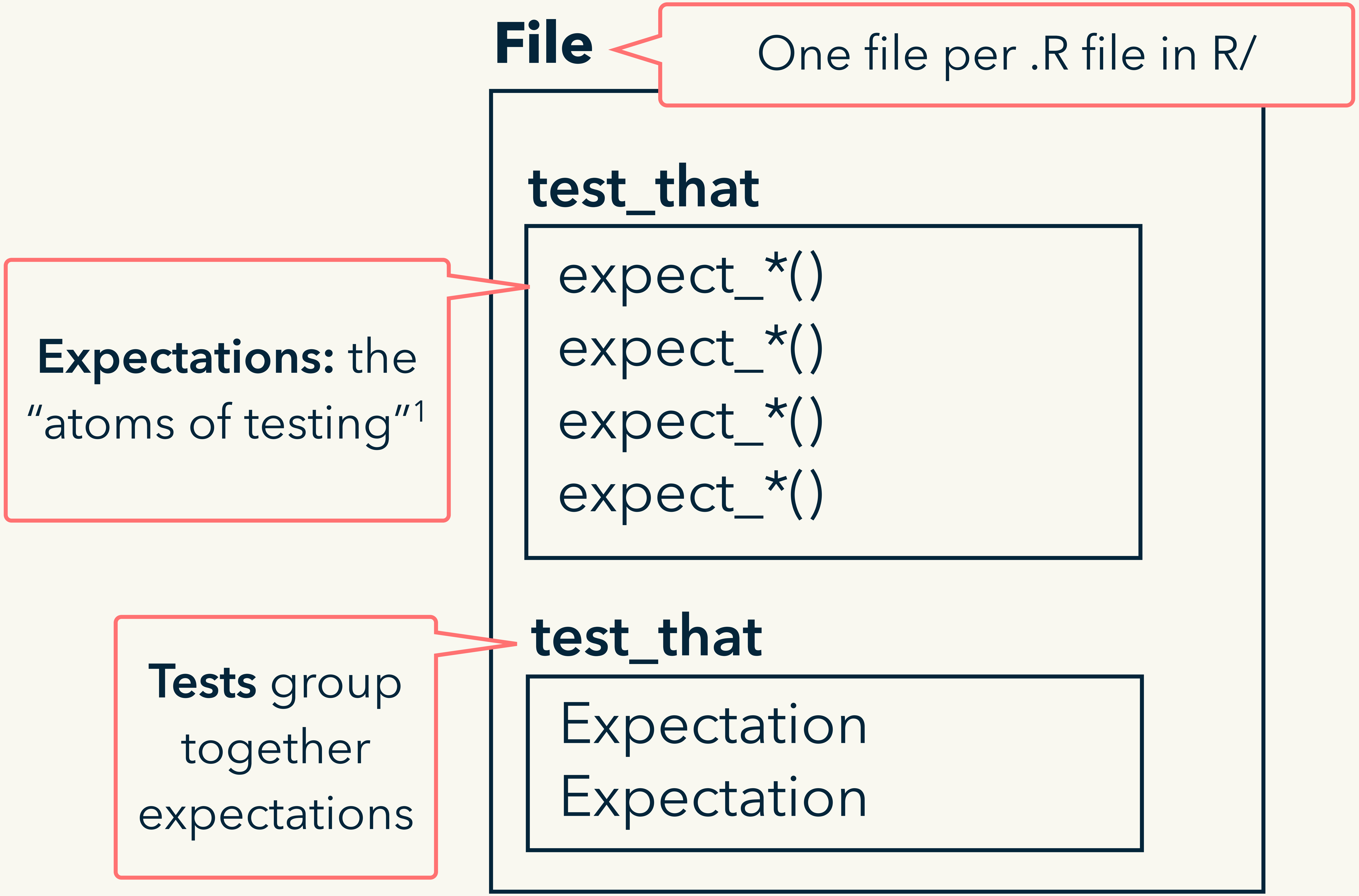
1. `create_package()`
2. `use_r("file_name")`
3. `use_test()`
4. `test() (Cmd/Ctrl + T)`

Our workflow so far

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

Write tests!

Tests are organized in three layers



File

One file per .R file in R/

test_that

expect_*()
expect_*()
expect_*()
expect_*()

Expectations: the "atoms of testing"[1]

test_that

Expectation
Expectation

Tests group together expectations

[1] https://r-pkgs.org/tests.html#test-structure

Your turn: Slido

expect_*() functions:

http://testthat.r-lib.org

Most important expectation

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

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

More at
http://testthat.r-lib.org

Your turn: insert_into()

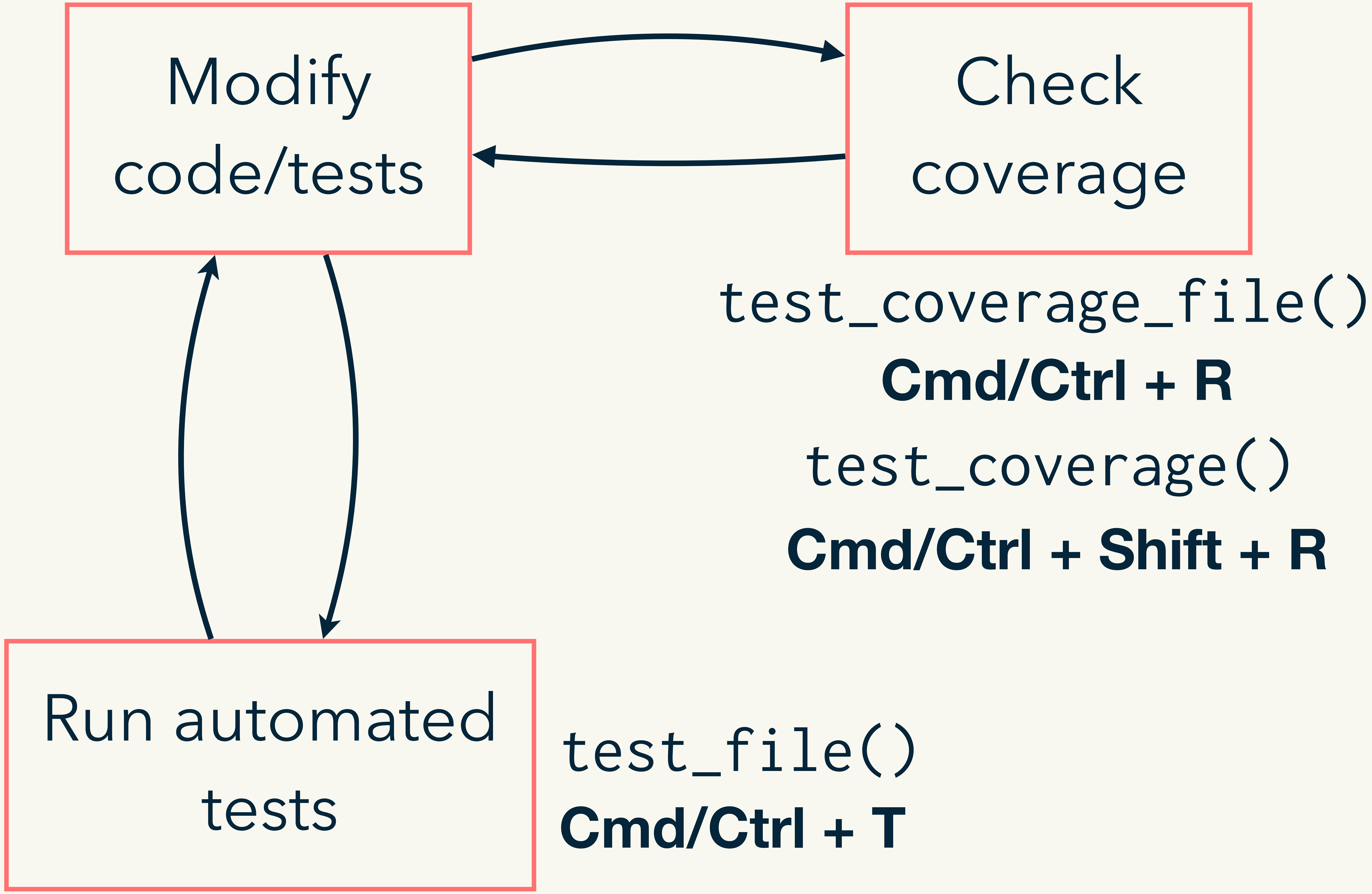
What does `insert_into()` do if `x` and `y` have different numbers of rows?

Decide if that's the behavior you want. Alter your code if needed, then write a test to check that the function behaves as expected.

Test coverage

https://covr.r-lib.org

Guide tests with coverage



Modify code/tests

Check coverage

Run automated tests

`test_coverage_file()`

Cmd/Ctrl + R

`test_coverage()`

Cmd/Ctrl + Shift + R

`test_file()`

Cmd/Ctrl + T

Your turn: Practice the workflow

```r
devtools::test_coverage_file()
# Are all the lines covered (green)?

# If not add a test for the missing case

# Check you now cover all cases
```

Other advantages

Fewer bugs.

Improve readability and performance without changing behavior.

Easier for you.

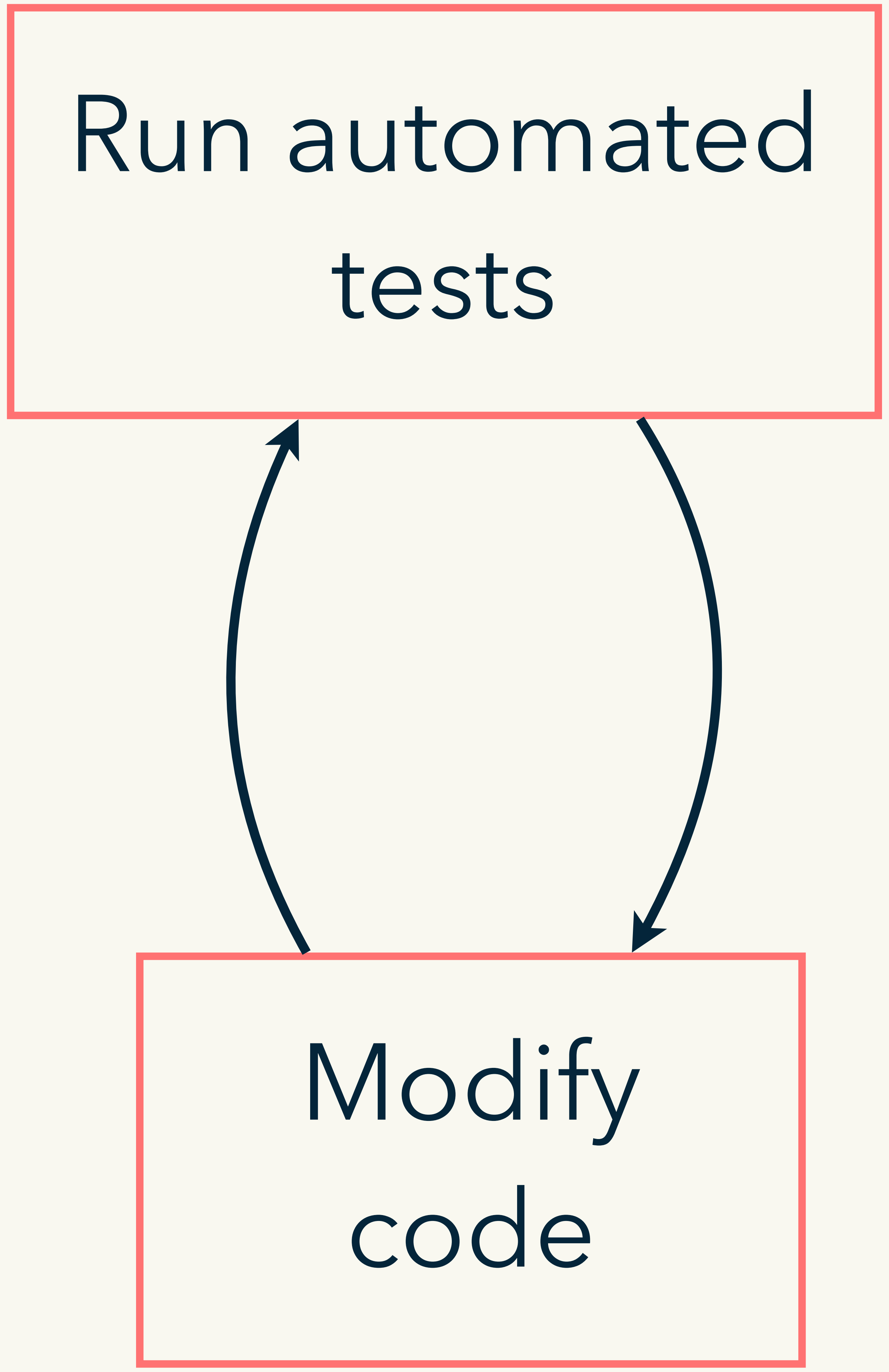
Leave a test failing.

Make the computer remember.

Stress less.

add_col

Or you might start with the tests



`test_file()`

Cmd/Ctrl + T

This is called test driven development (TDD)

Next challenge is to implement add_col()

Helper: `insert_into()`

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

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

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

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

Your turn: Make these tests pass

```
usethis::use_test("add_col")
# Copy this test:
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
```

My solution

Your turn: Make this test pass

```r
# 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: Make this test pass

```r
# 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)
  )
})
```

My solution

What about bad inputs?

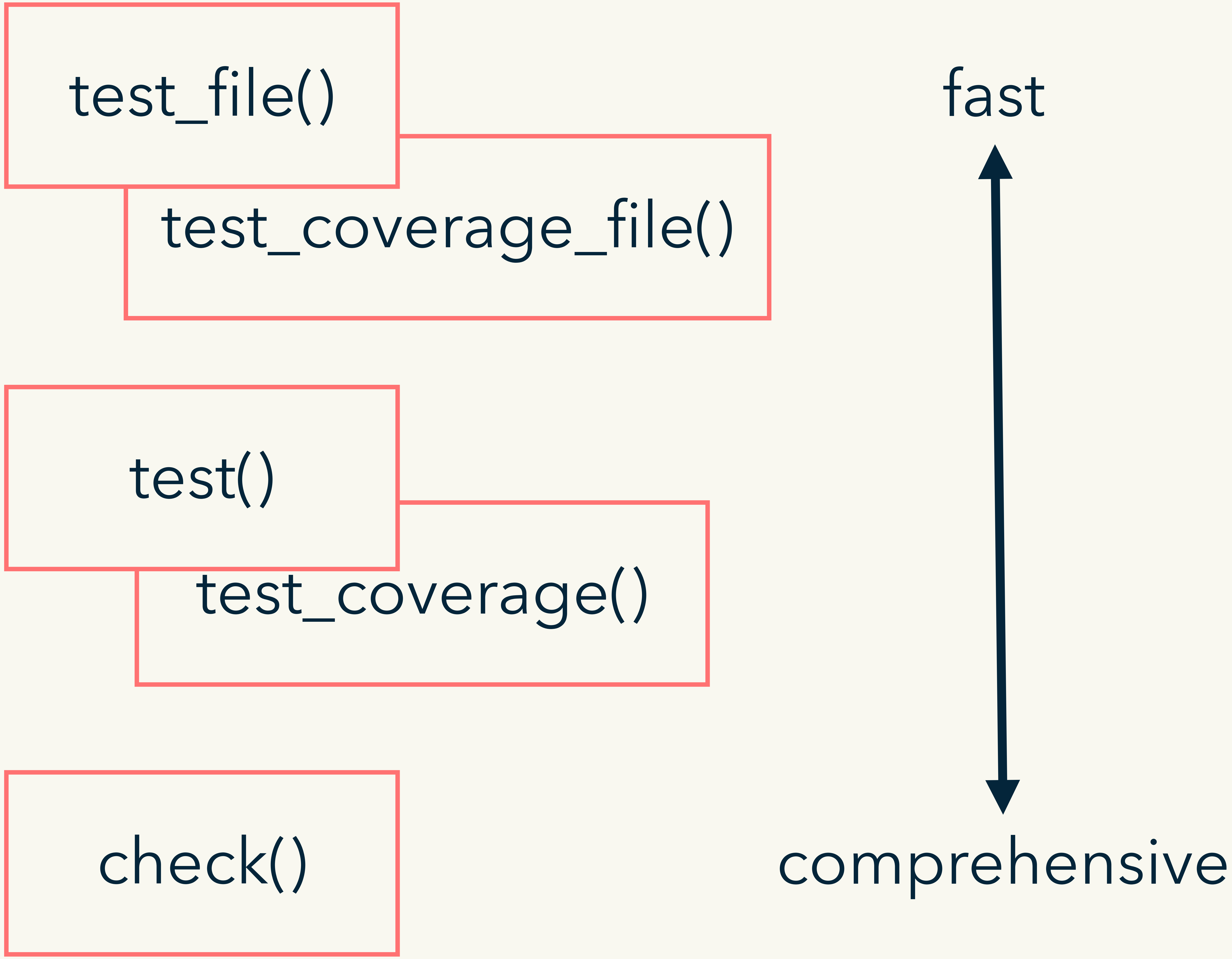
```r
# We need to test for errors too
df1 <- data.frame(a = 3, b = 4, c = 5)
df2 <- data.frame(X = 1, Y = 2)

insert_into(df1, df2, where = 0)
insert_into(df1, df2, where = NA)
insert_into(df1, df2, where = 1:10)
insert_into(df1, df2, where = "a")
```

Your turn: Deal with bad inputs

```
# We need to test for errors too
df1 <- data.frame(a = 3, b = 4, c = 5)
df2 <- data.frame(X = 1, Y = 2)

insert_into(df1, df2, where = 0)
insert_into(df1, df2, where = NA)
insert_into(df1, df2, where = 1:10)
insert_into(df1, df2, where = "a")
```



test_file()

test_coverage_file()

test()

test_coverage()

check()

fast

comprehensive