

Unit testing

Back at 11am

October 2019

Hadley Wickham
@hadleywickham

Motivation

Let's add a column to a data frame

Goal:

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 = 2)
```

Start simple and try out as we go

where =

1	2	3	4
x	y	z	
3.4	1.2	6.7	
1.9	6.1	3.1	
10.0	2.7	7.7	

Start with insert_into()

Works like cbind() but can insert anywhere

df1	a	b	c
	3	4	5

df2	X	Y
	1	2

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

X	Y	a	b	c
1	2	3	4	5

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

a	X	Y	b	c
3	1	2	4	5

Add the columns of df2 to df1 at position where

What goes in ...?

```
insert_into <- function(x, y, where = 1) {  
  if (where == 1) { # first col  
    ...  
  } else if (where > ncol(x)) { # last col  
    ...  
  } else {  
    ...  
  }  
}  
  
# Hint: cbind() will be useful  
# Add the columns of df2 to df1 at position where
```

My 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

```
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])  
  }  
}
```


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
```

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

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

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

Two challenges

Cmd + Enter is error prone

**Looking at the outputs of
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 of
each run is tedious**

Write unit tests and use devtools::**test_file()**

Testing workflow

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

First, create a package

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

```
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])  
  }  
}
```

copy + paste
this code into
insert_into.R

Then, set up testing infrastructure

Key infrastructure

`usethis::use_test()`

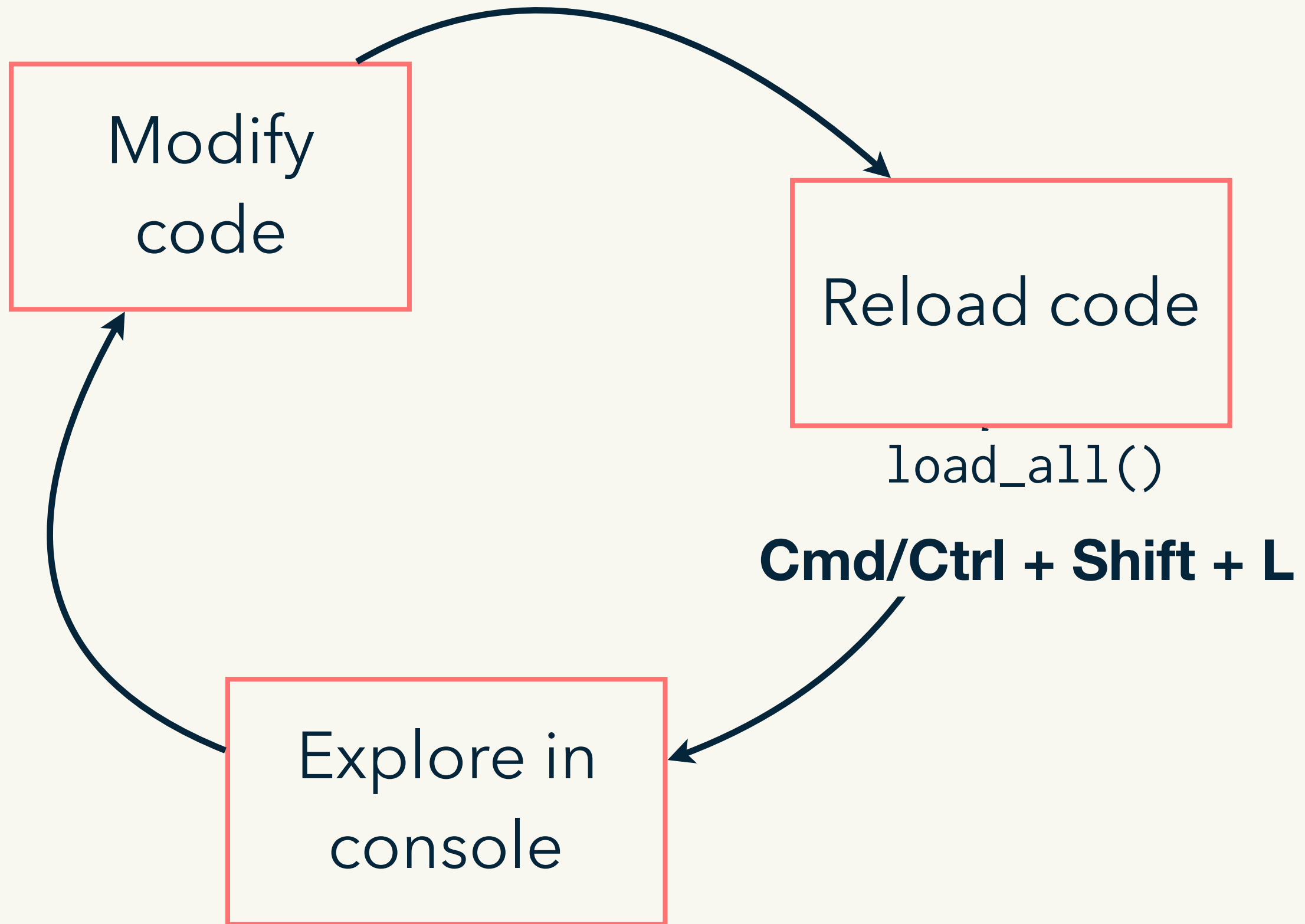
- ✓ Adding 'testthat' to Suggests field
- ✓ Creating 'tests/testthat/'
- ✓ Writing 'tests/testthat.R'
- ✓ Writing 'tests/testthat/test-insert_into.R'
- Modify 'tests/testthat/test-insert_into.R'

Run tests

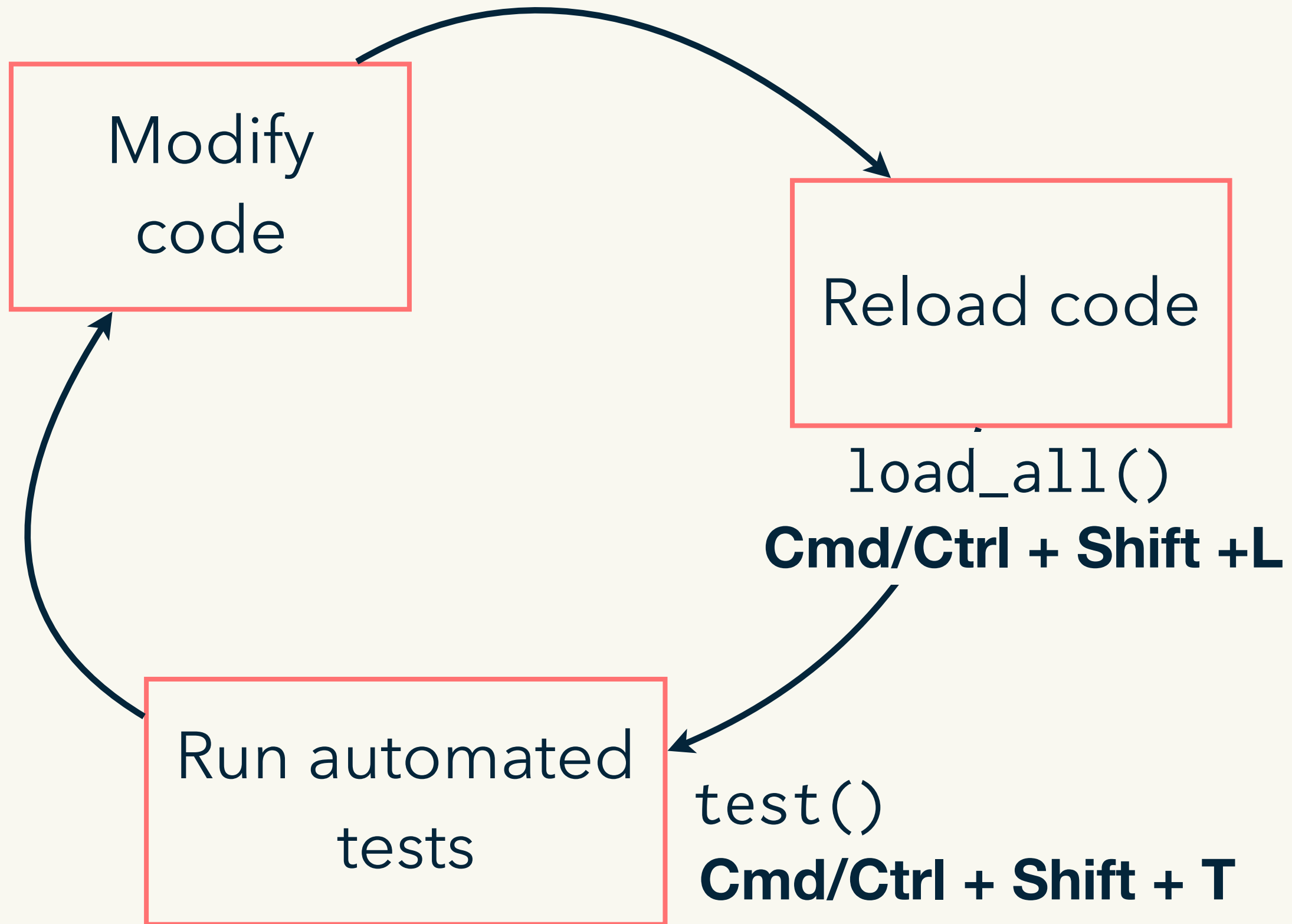
Creates test file
matching script file

`devtools::test_file()`

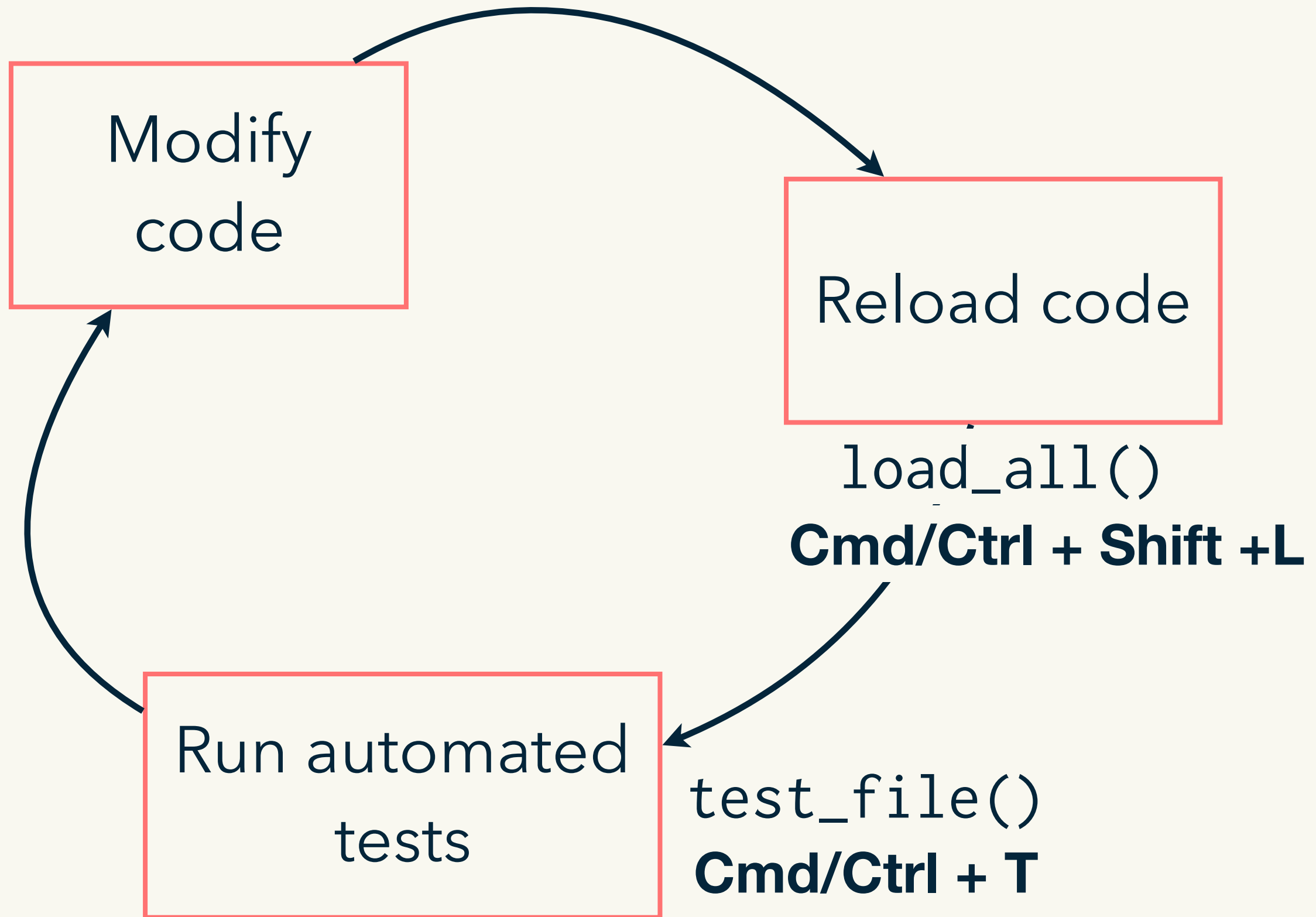
So far we've done this:



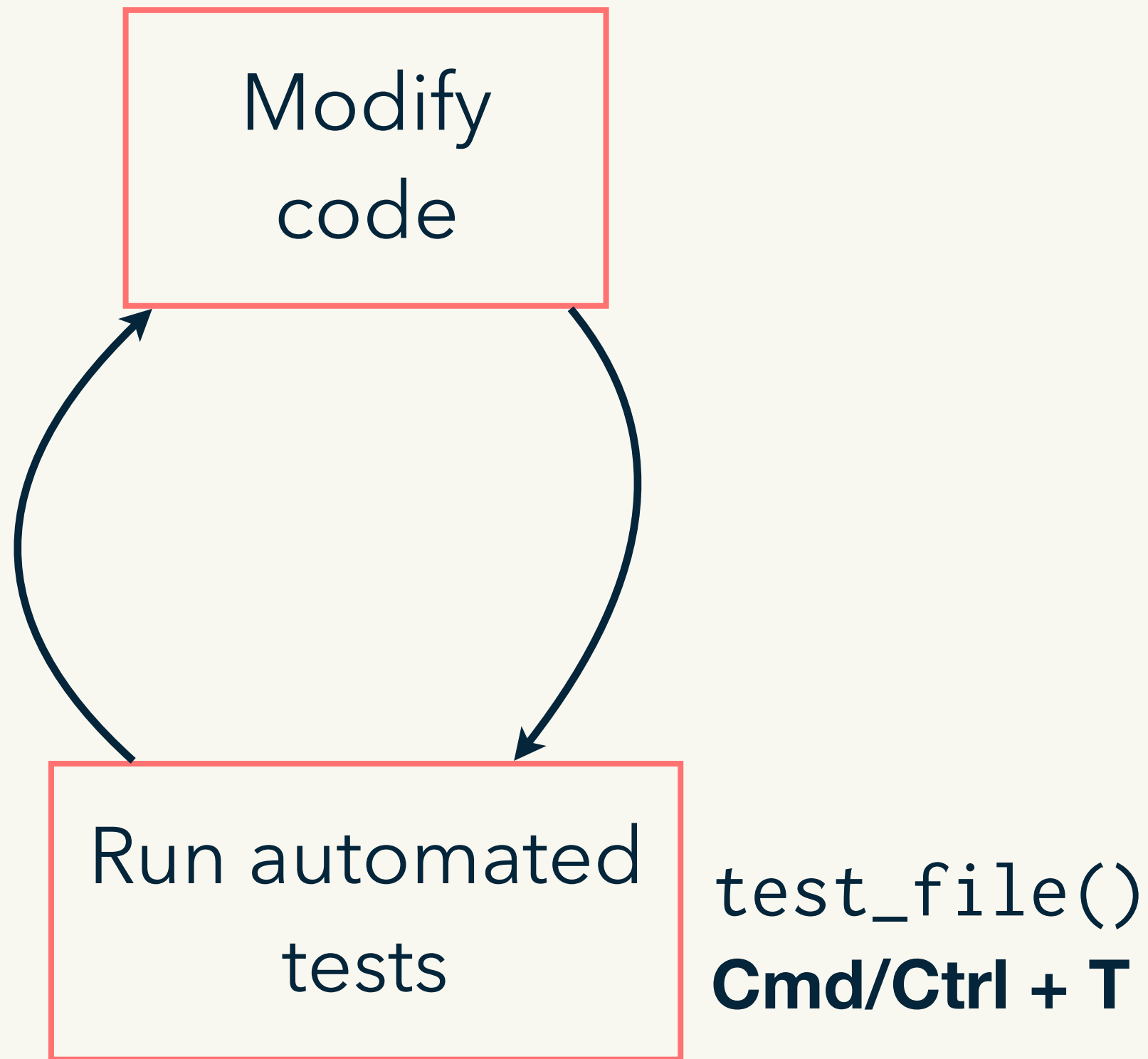
Testthat gives a new workflow



Testthat gives a new workflow



But why reload the code?



Key idea of unit testing is to automate!

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"))
```

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"))
```

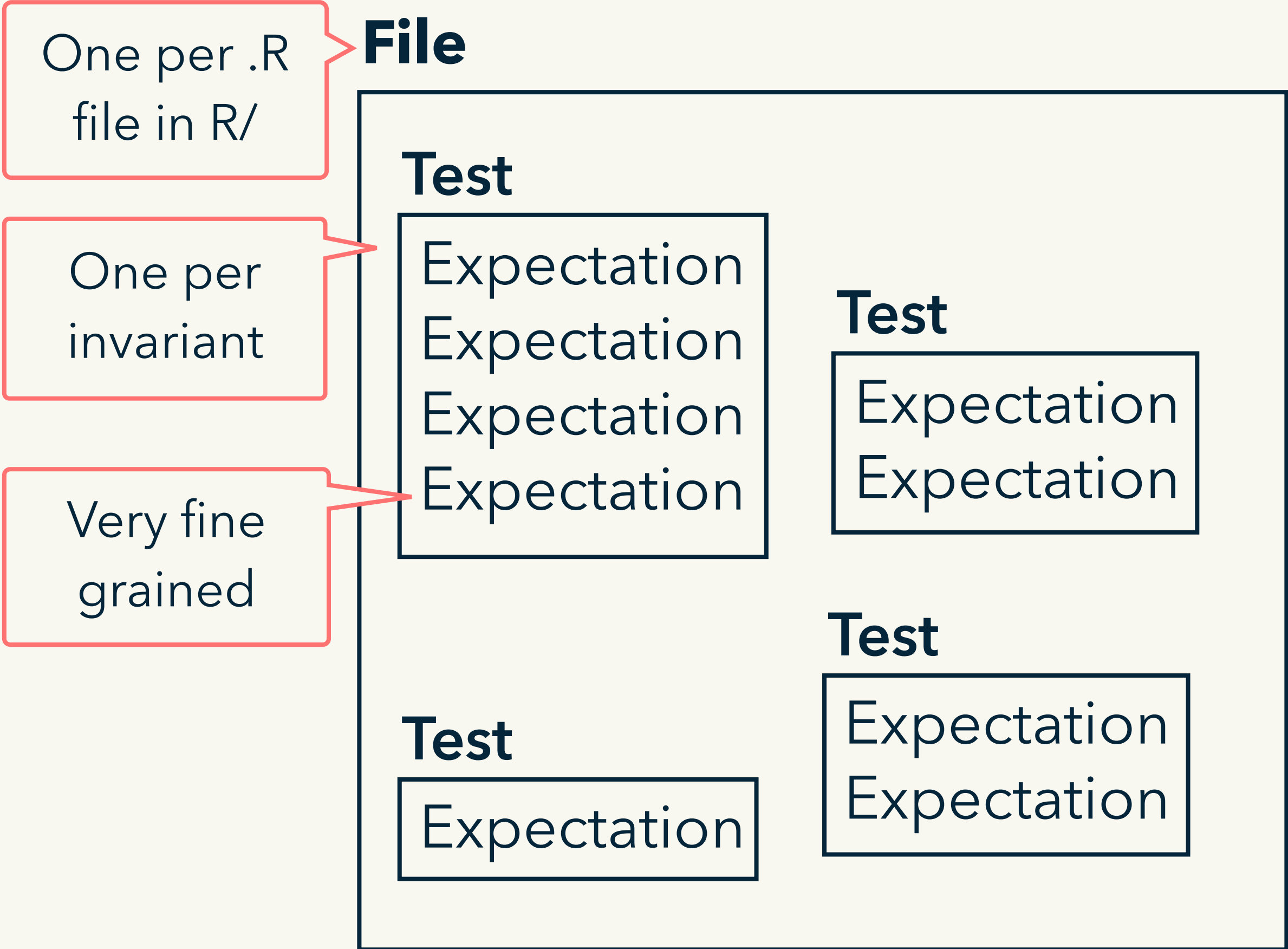
Easy to see the pattern

This automation must follow conventions

```
# In tests/testthat/test-insert_into.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) {
    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"))
})
```

Tests are organized in three layers



Setup keyboard shortcuts

Tools Window Help

Install Packages...
Check for Package Updates...

Version Control ▶

Shell...
Terminal ▶
Jobs ▶
Addins ▶

Keyboard Shortcuts Help ⌘⇧K
Modify Keyboard Shortcuts...

Project Options... ⌘⌘,
Global Options... ⌘⌘,

Keyboard Shortcuts

Show: ☒ All ☐ Customized

▲ Name	Shortcut	
Calculate package test coverage	Ctrl+Shift+C	
Compare test results for Shiny application		
Record a test for Shiny		
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

Practice the workflow

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

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

```
# Check all is ok with load_all()
```

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

```
# Copy expectations from next next slide
```

```
# Run tests with keyboard shortcut (if you create it).
```

```
# Or devtools::test_file()
```

```
# Confirm that if you break insert_into() the
```

```
# tests fail.
```


Expectations

```
# 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 = 1, Y = 2)
  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>

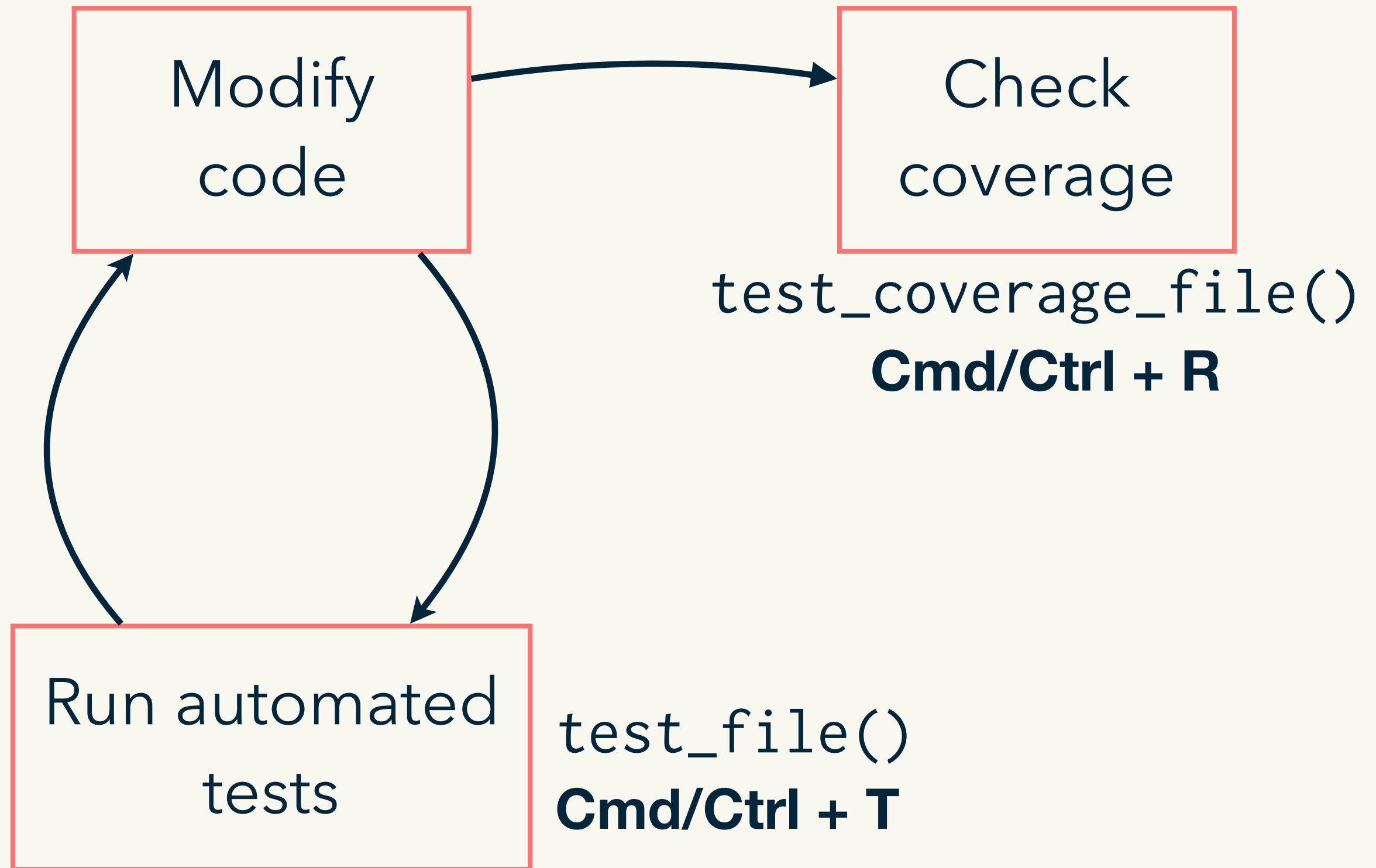
Test coverage shows you what you've tested

```
devtools::test_coverage_file()
```

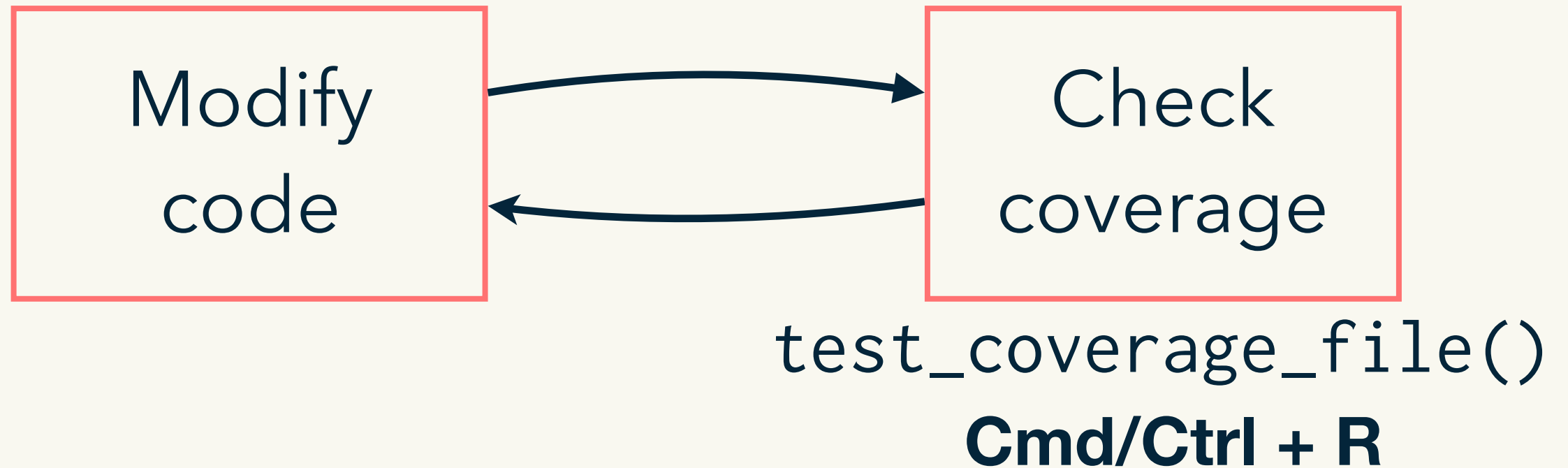
```
devtools::test_coverage()
```

```
usethis::use_coverage()
```

Guide tests with coverage



Guide tests with coverage



Practice the (new) workflow


```
devtools::test_coverage_file()
```

```
# Are all the lines covered (green)?
```

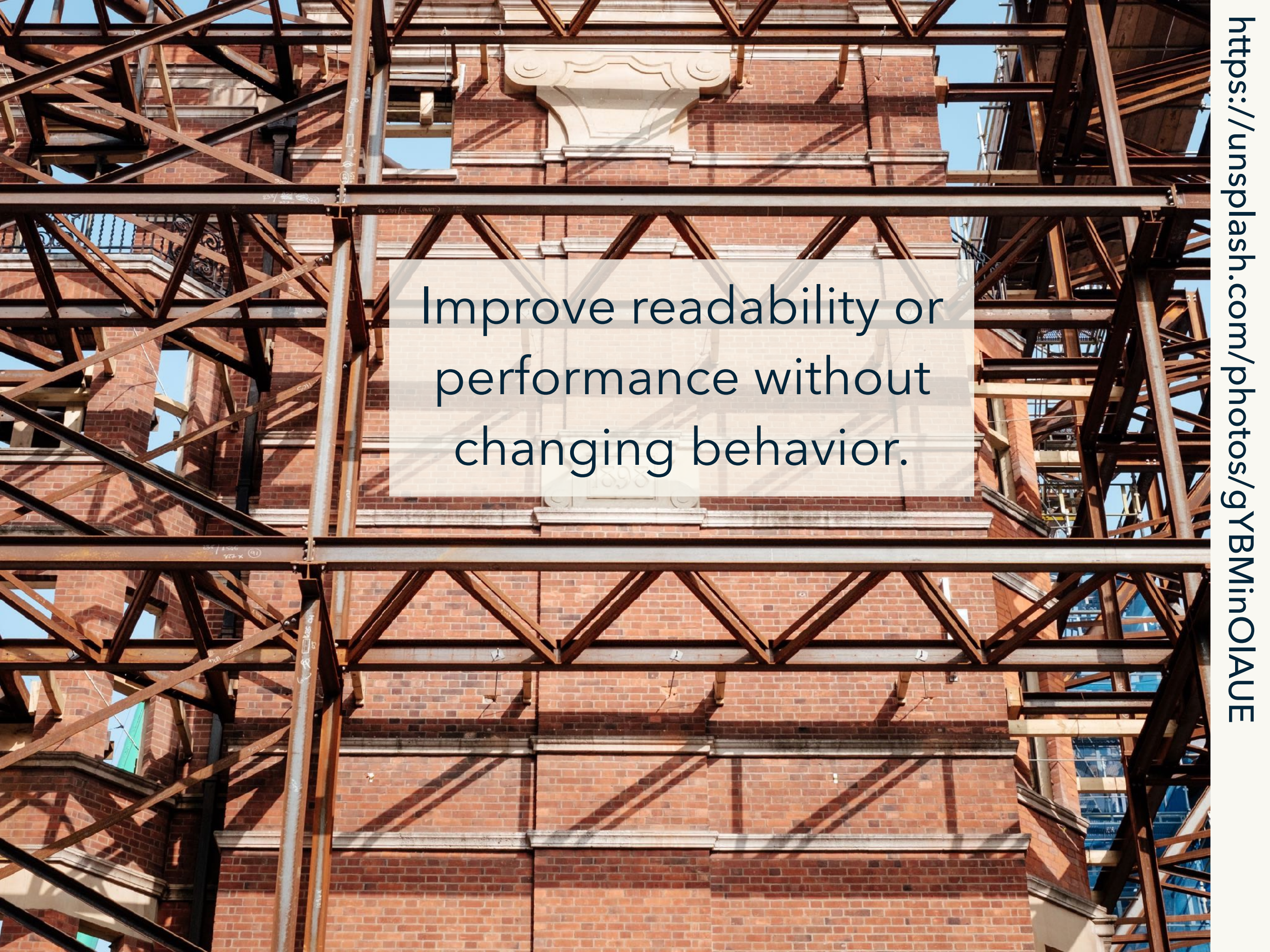
```
# If not add a test for the missing case
```

```
# Check you now cover all cases
```


Other advantages



Writing tests
improves your
interface



Improve readability or
performance without
changing behavior.



When you stop work,
leave a test failing

tidyverse/dplyr: dplyr: A grammar of data manipulation

GitHub, Inc. [US] | https://github.com/tidyverse/dplyr

README.md


dplyr

CRAN 0.8.1 build passing build passing codecov 83%

Overview

dplyr is a grammar of data manipulation, providing a consistent interface for data manipulation challenges:

Builds confidence
Guides contributors



ata

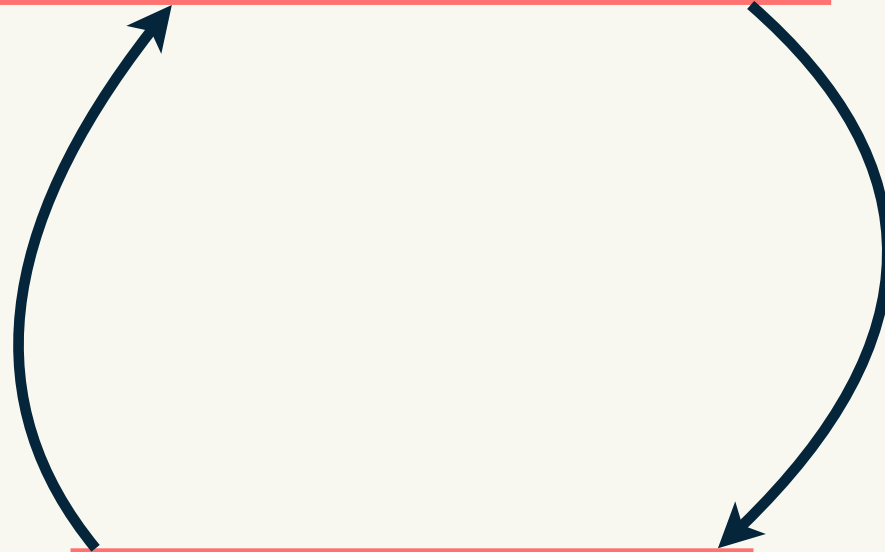
https://dplyr.tidyverse.org

add_col

Or you might start with the tests

Run automated
tests

`test_file()`
Cmd/Ctrl + T



Modify
code

This is called test driven
development (TDD)

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)
```

Most important expectation

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

More at

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

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)
  )
})
# Run tests with keyboard shortcut
# Some hints on next slide
```


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

```
# Lives in R/add_col.R
```

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

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)
  )
})
```

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

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)
  )
})
```

1	2	3	4
x	y	z	
3.4	1.2	6.7	
1.9	6.1	3.1	
10.0	2.7	7.7	

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)
  }
}
```


What about 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

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/)