Unit Testing

September 2019

Angela Li
Samantha Toet

Workshop materials: bit.ly/cville_pkg

Why test?

Improve readability or performance without changing behavior.

https://unsplash.com/photos/gYBMinOIAUE

Coding is Iterative

We build new functions one bit at a time.

What if a new thing we add changes the existing functionality?

How can we check and be sure all the old functionality still works with New Fancy Feature?

Unit Tests!

Test Driven Development (aka package dev workflow)

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

Of course there's a `usethis::` for it

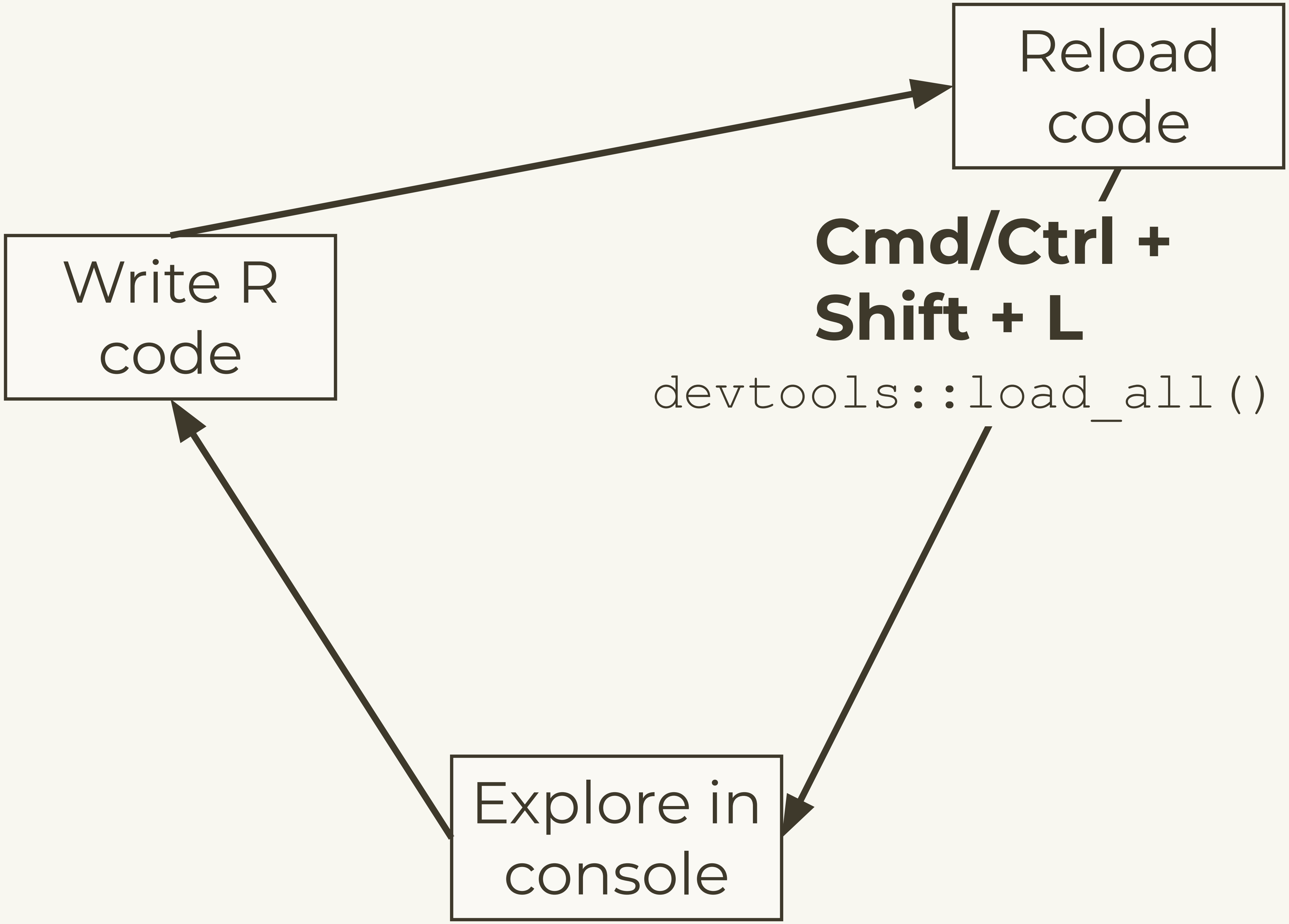
Sets up file structure

```
usethis::use_testthat()
✔ Adding 'testthat' to Suggests field
✔ Creating 'tests/testthat/'
✔ Writing 'tests/testthat.R'
✔ Writing 'tests/testthat/test-zooSounds.R'
● Modify 'tests/testthat/test-zooSounds.R'

devtools::test()
# Or Command + Shift + T
```

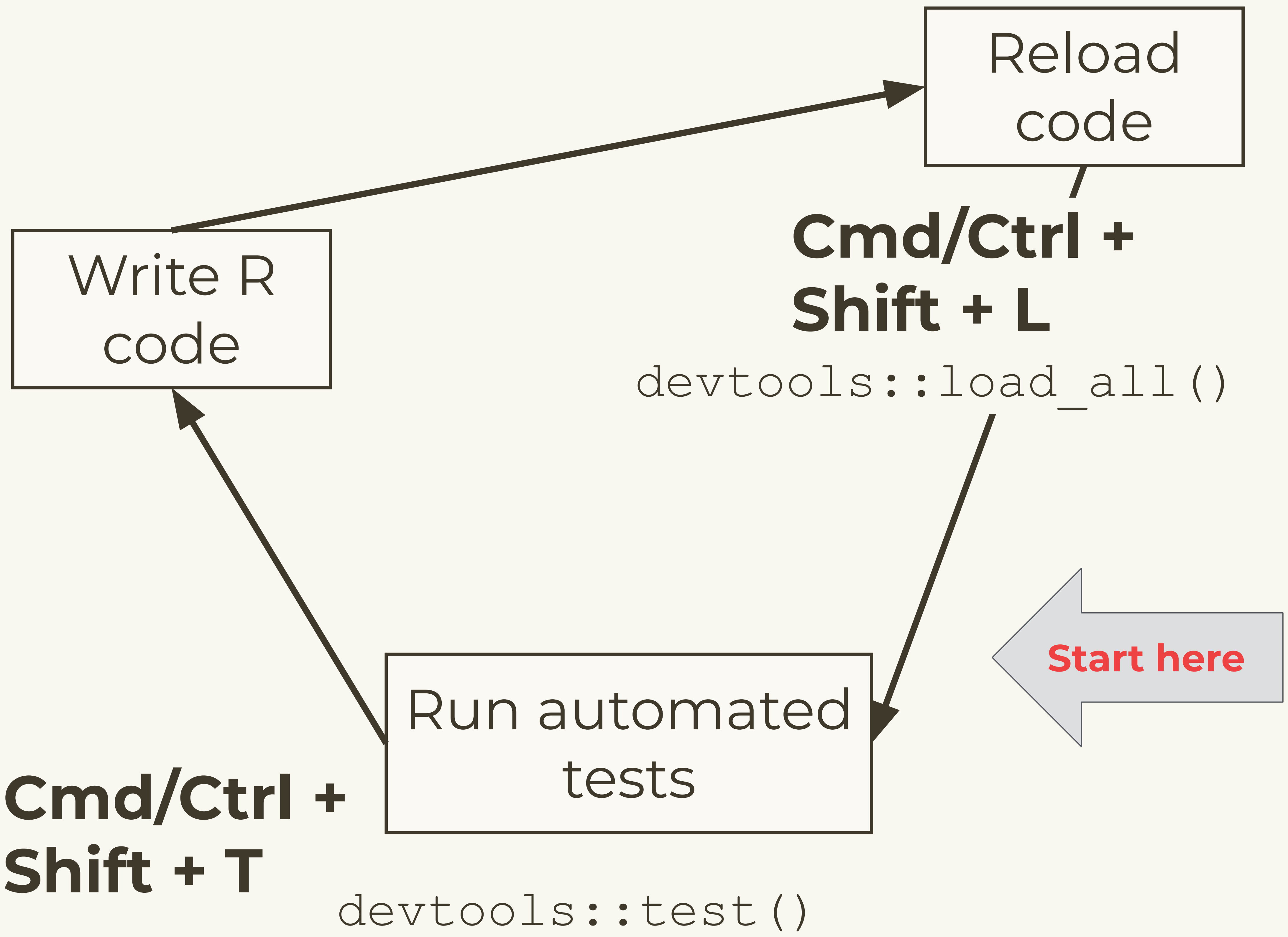
Helps you write tests for each file

So far we've done this:



```
devtools::load_all()
```

Test driven development is a new workflow



Cmd/Ctrl + Shift + L

`devtools::load_all()`

Cmd/Ctrl + Shift + T

`devtools::test()`

Four expectations cover 90% of cases

```
expect_equal(object, expected)

expect_error(object, regex)

expect_warning(object, regex)

expect_known_output(code)
```

Our Example Function

`zooSounds.R`

```
goToTheZoo <- function(animal, sound){
assertthat::assert_that(
  assertthat::is.string(animal),
  assertthat::is.string(sound))
  glue::glue("The ", animal, " goes ", sound,"!",
sep = " ")
}
```

A sample test

Tests for R/zooSounds.R

```r
# In tests/testthat/test-zooSounds.R

library(testthat)


test_that("goToTheZoo produces expected strings", {
  allSounds <- as.character(goToTheZoo("giraffe",
"moo"))
  expect_equal(allSounds, "The giraffe goes moo!")
})
```

Your turn

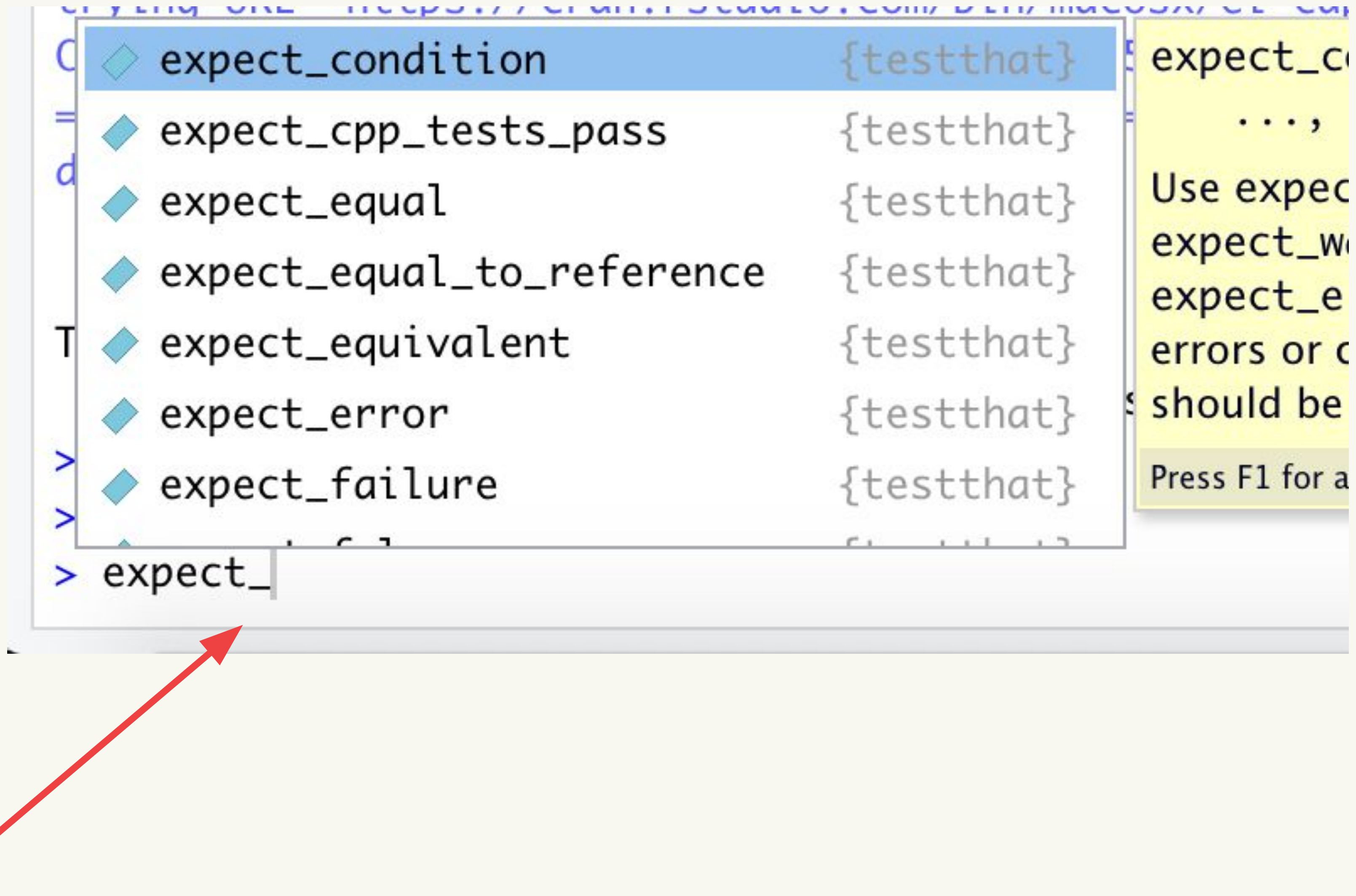
Write a new unit test for `goToTheZoo()` using `expect_error()`.

Run the tests with `Cmd + Shift + T`

Other side effects

There are many other variables you can test for

In your console, type in `expect_` and then `Tab` to scroll through the options



```
  expect_condition               {testthat}   expect_c
  expect_cpp_tests_pass          {testthat}       ...,
  expect_equal                   {testthat}   Use expec
  expect_equal_to_reference      {testthat}   expect_w
  expect_equivalent              {testthat}   expect_e
  expect_error                   {testthat}   errors or c
  expect_failure                 {testthat}   should be
                                              Press F1 for a
> expect_
```

Organizing Tests

Think about the overall functionality, or “end to end” tests

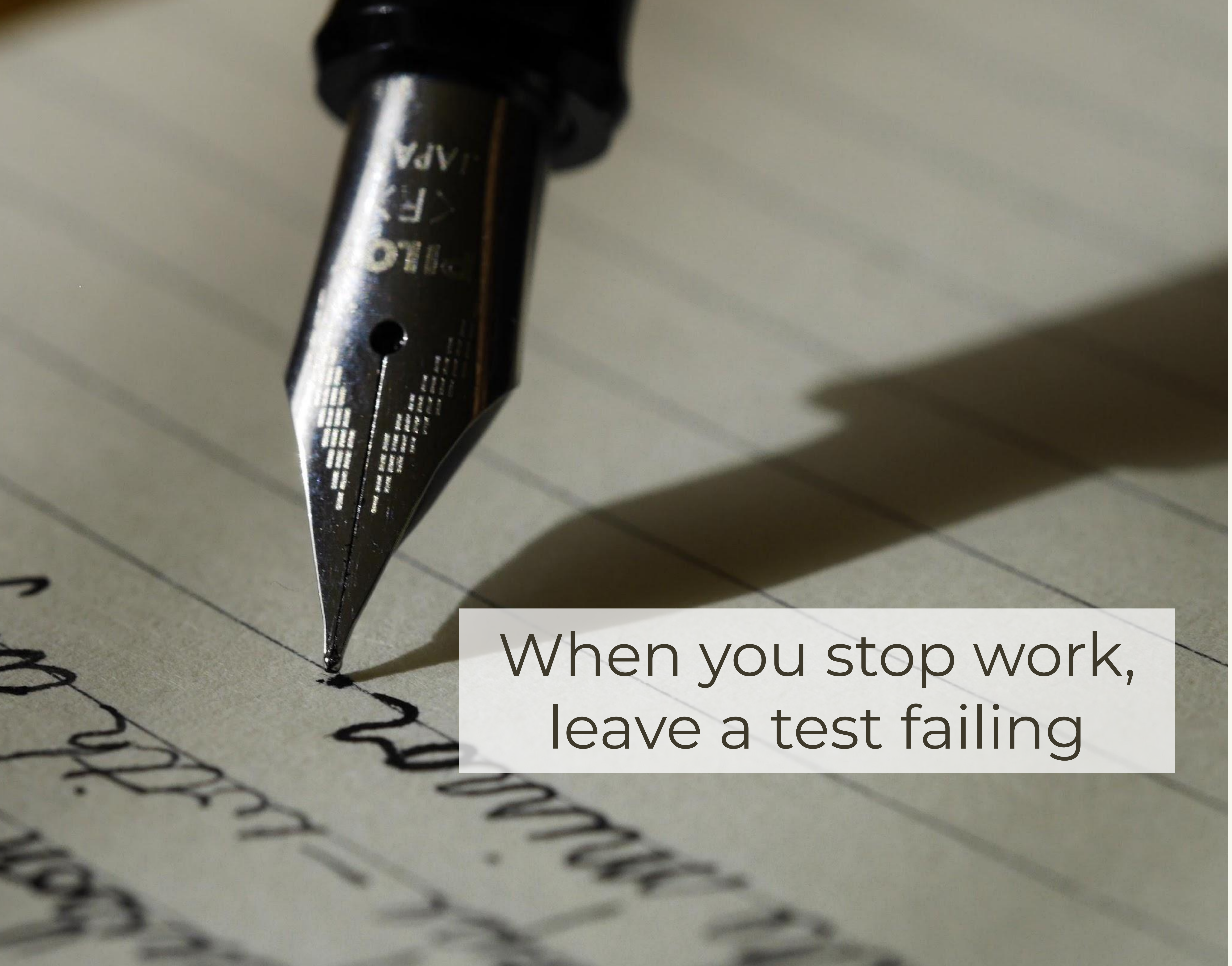
Test every individual task the function completes separately

Check both for successful situations and for expected failure situations

Hints

What conditions should cause the function to error?

What operations is the function supposed to do?



When you stop work, leave a test failing

https://unsplash.com/photos/y02jEX_B0O0

If you're bored in this class, write tests!

https://unsplash.com/photos/AVRxWIsK9Jk

Test Coverage

Useful to know which lines have been tested

```r
# Powered by the covr package
devtools::test_coverage()
```

Your turn

Test the coverage of your package, and verify that every line has been tested.

Have we missed anything? Can you add a test that checks it?

You can also automate

- GitHub = publish and manage your code online
- Travis or Jenkins = Continuous Integration; run code (like your tests) every time your code changes
  - https://travis-ci.org/
  - https://jenkins.io/
- Codecov = display which functions are tested
  - https://github.com/codecov/example-r