Test tooling and design

Drawing from newly updated
https://r-pkgs.org/tests.html

July 2022

Jenny Bryan
@jennybryan twitter, @jennybc github
Software Engineer, RStudio

testthat setup: once per package*

```
usethis::use_testthat(3)
```

*ok, technically, once per package, per testthat edition

create or open a test file

```
usethis::use_test("test-whatever")


# in RStudio, with a R/.R file focused,
# target test file can be inferred
usethis::use_test()


# use_test() is half of a matched pair:
# usethis::use_r()
```

*ok, technically, once per package, per testthat edition

Workflow: micro-iteration, interactive experimentation

```
# tweak the foofy() function and re-load it
devtools::load_all()

# interactively explore and refine expectations
# and tests
expect_equal(foofy( ... ), EXPECTED_FOOFY_OUTPUT)

testthat("foofy does good things", { ... })
```

`load_all()`

- testthat's workflow is designed around `load_all()`
- makes entire package namespace available
- attaches testthat
- sources `tests/testthat/helper.R`

Workflow: mezzo-iteration, whole test file

```
testthat::test_file("tests/testthat/test-foofy.R")


# in RStudio, with test file focused, "Run Tests"


# in RStudio, with test file focused
devtools::test_active_file()
# pro tip: bind this to Ctrl/Cmd + T
# also works when matching R/.R file is focused
```

Workflow: macro-iteration, whole test suite

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

```
devtools::check()
```

High-level principles

1. A test should be self-sufficient and self-contained.
2. The interactive workflow is important.
3. Obvious >>> DRY
4. Don't let a nonstandard workflow "leak".

Test smell: top-level code that's outside test_that()

```
dat ← data.frame(x = c("a", "b", "c"), y = c(1, 2, 3))

skip_if(today_is_a_monday())

test_that("foofy() does this", {
  expect_equal(foofy(dat), ... )
})

dat2 ← data.frame(x = c("x", "y", "z"), y = c(4, 5, 6))

skip_on_os("windows")

test_that("foofy2() does that", {
  expect_snapshot(foofy2(dat, dat2)
})
```

Deodorizing the previous example

```
test_that("foofy() does this", {
  skip_if(today_is_a_monday())

  dat ← data.frame(x = c("a", "b", "c"), y = c(1, 2, 3))

  expect_equal(foofy(dat), ...)
})

test_that("foofy() does that", {
  skip_if(today_is_a_monday())
  skip_on_os("windows")

  dat ← data.frame(x = c("a", "b", "c"), y = c(1, 2, 3))
  dat2 ← data.frame(x = c("x", "y", "z"), y = c(4, 5, 6))

  expect_snapshot(foofy(dat, dat2)
})
```

Move file-scope logic to a narrower scope (as done here) or a broader scope (coming soon).

Test code doesn't have to be super DRY.

Leave the world the way you found it

```
test_that("side-by-side diffs work", {
  withr::local_options(width = 20)
  expect_snapshot(
    waldo::compare(c("X", letters), c(letters, "X"))
  )
})
```

withr's `local_*()` functions
are super useful for this.

Calls to avoid below `tests/`

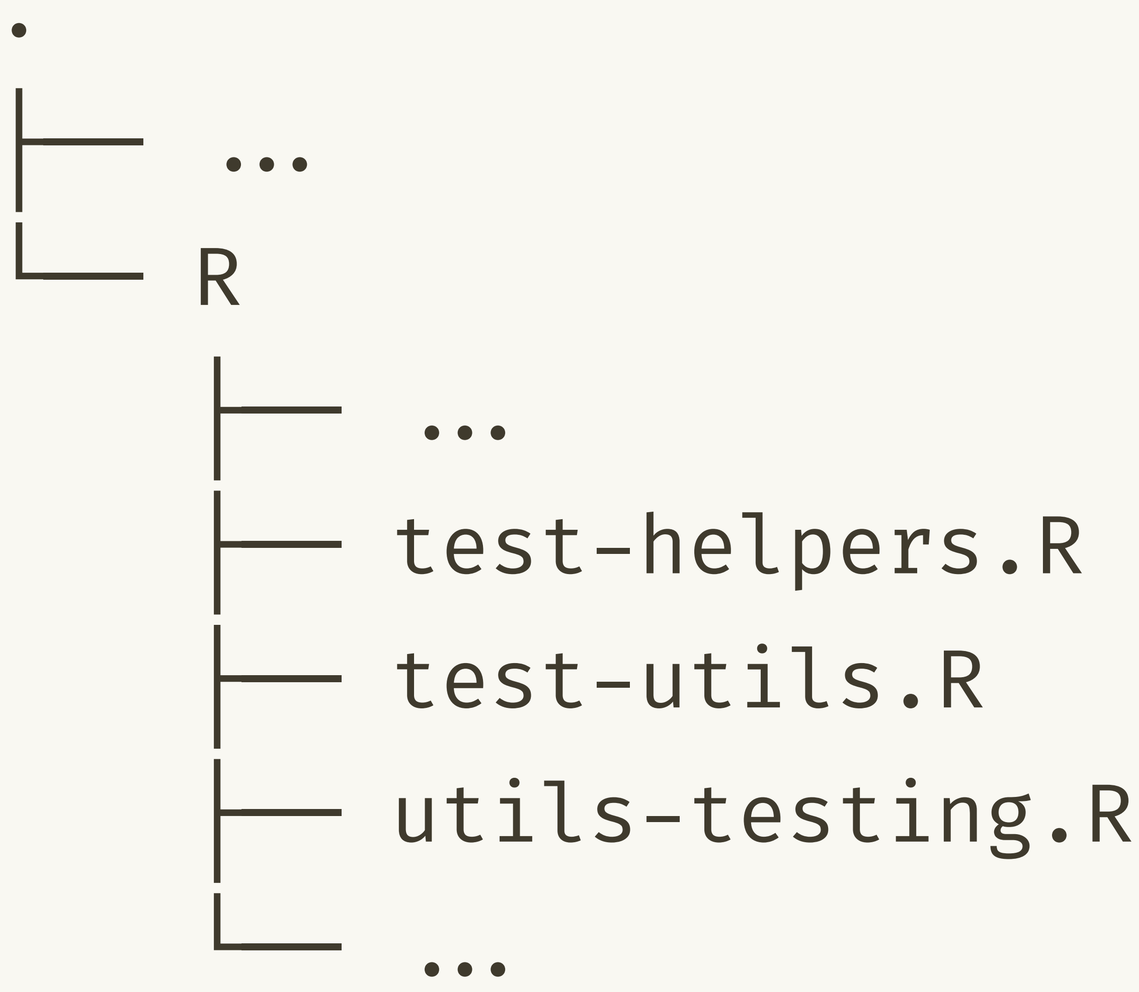
```
library(somedependency)
```

Access functions from your dependencies, in your tests, exactly as you do below `R/`.

```
source("random-stuff.R")
```

`R/*.R`,`tests/testthat/helper.R`, and `tests/testthat/setup.R` are all better locations for whatever's in `random-stuff.R`.

Files relevant to testing: `R/*.R`



test helpers can be internal functions in your package.

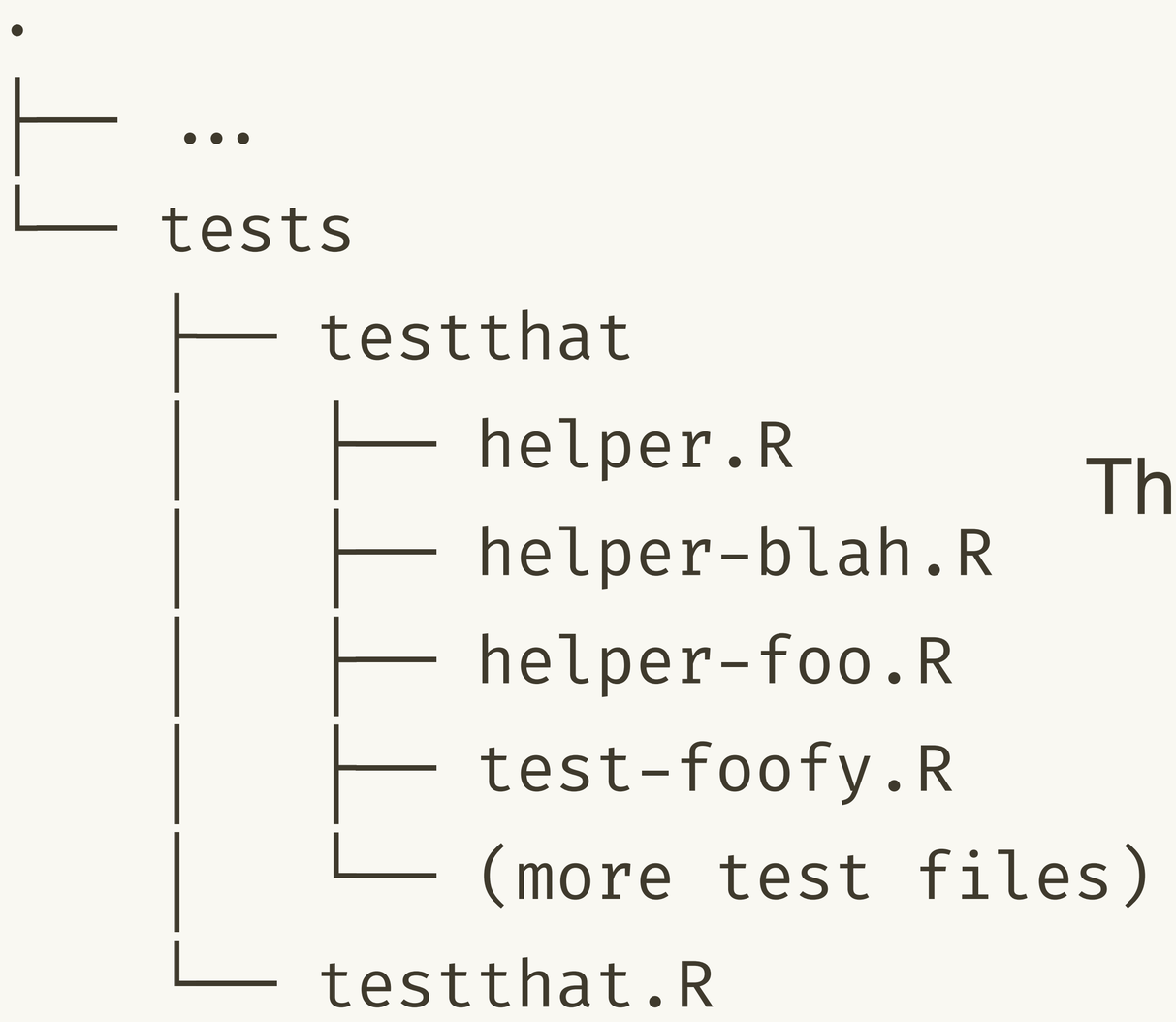
Files relevant to testing: `tests/testthat.R`

```
library(testthat)
library(abcde)

test_check("abcde")
```

DO NOT MESS WITH THIS FILE.
JUST DON'T.

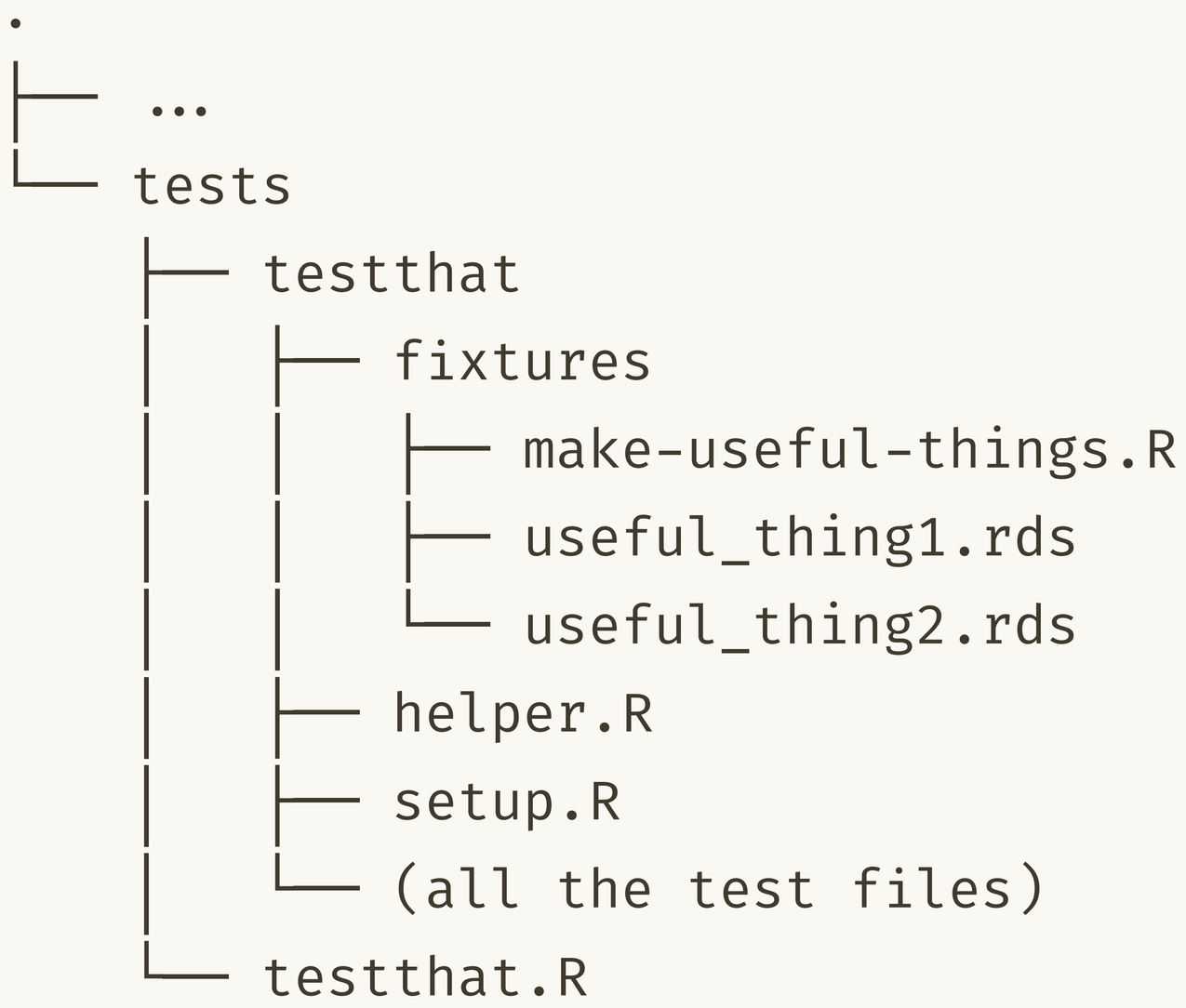
Files relevant to testing: `tests/testthat/helper.R`



Test helper files are executed by `load_all()` and at the start of automated testing.

This is what I meant by "relocate file-scope logic to a broader scope".

More re: files



Setup files are good for certain types of setup+teardown.

Sometimes fixtures are useful.

Only write to session temp dir.

Clean up after yourself.