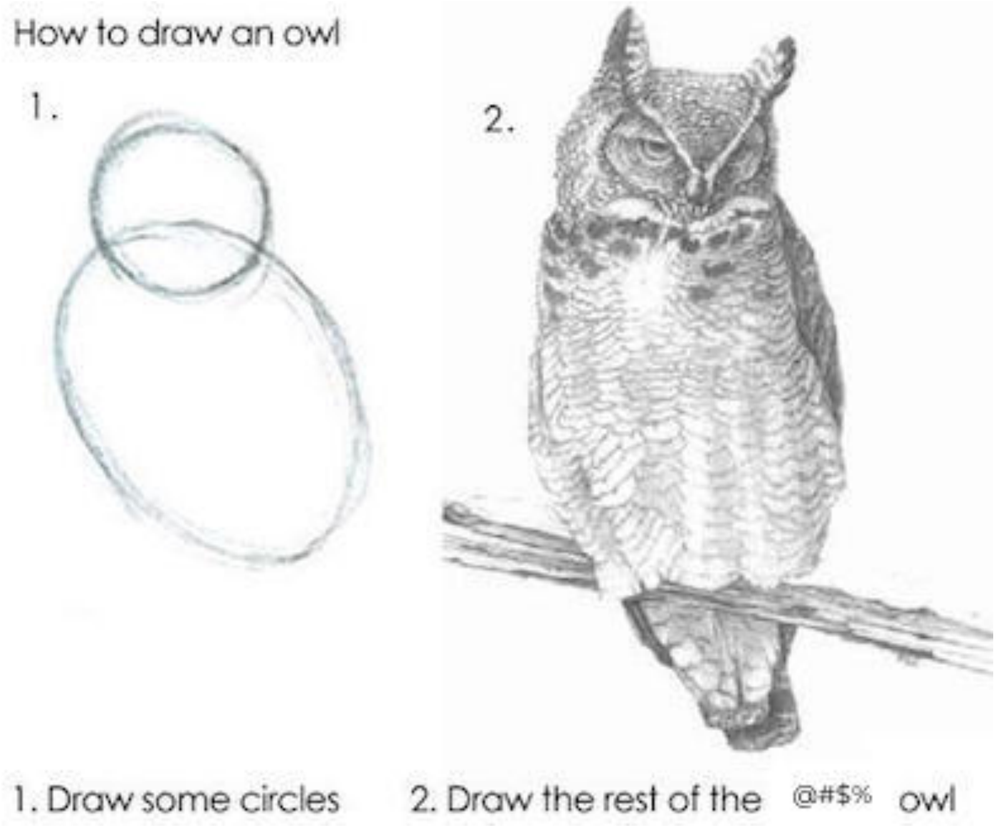
Package Development:

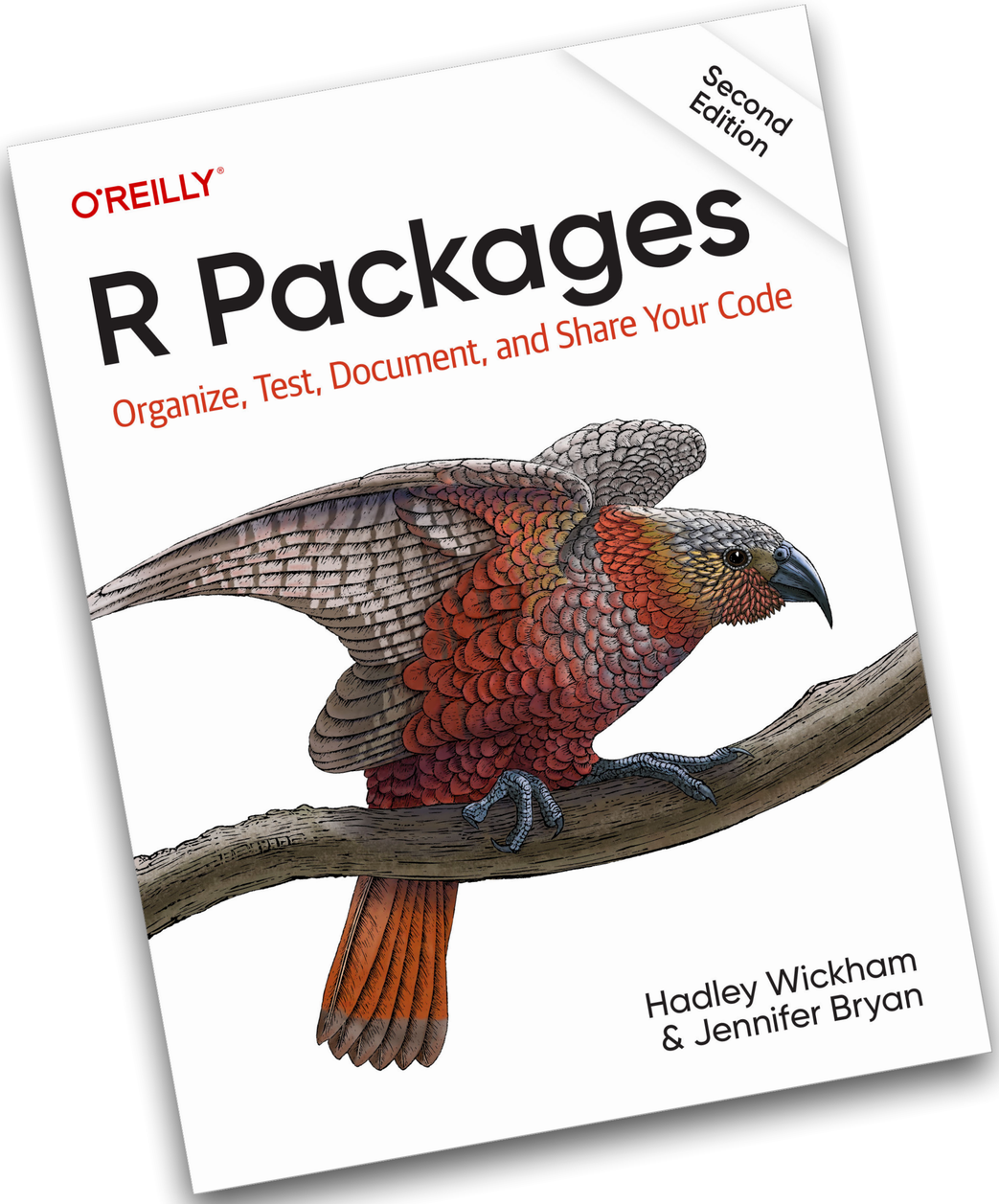
The Rest of the Owl

GitHub repo: pos.it/pkg_dev_24

Wifi: Posit Conf 2024 | conf2024

Discord: #workshop-pkg-dev





https://r-pkgs.org/

Introductions

- Jenny Bryan, instructor
- Tomasz Kalinowski, TA
- Kevin Ushey, TA

- Introduce yourself to your neighbors.
  - Did you bring a package with you today? If so, what are you most hoping to learn today?
  - Are you just starting out? What do you feel the most confused or curious about?

Outline
Kick the tires of your setup Core workflows Let me GitHub search that for you
Testing
Documentation
GitHub Actions Beautiful UI with cli Topic by popular request?

Kicking the tires

Prework

```
install.packages(c("devtools", "knitr"))
# devtools also installs other important packages, such as
# usethis
# roxygen2
# testthat
# pkgdown

install.packages("pak")
# extremely useful for package installation, as a developer
# pak is the successor to remotes
```

```
library(devtools)
#> Loading required package: usethis
```

From this point on, I assume you're attaching devtools, and therefore usethis, in today's interactive work.

Don't expect `devtools::this()` or `usethis::that()`.

Prework

```
# you might want this in .Rprofile
# to make devtools always available in all interactive sessions
if (interactive()) {
  suppressMessages(require(devtools))
}


# helps you get ^^^ the code above ^^^ into your .Rprofile
use_devtools()
# won't take affect until you restart R
```

Prework

```
options(
    usethis.description = list(
      "Authors@R" = utils::person(
        "Jennifer", "Bryan",
        email = "jenny@posit.co",
        role = c("aut", "cre"),
        comment = c(ORCID = "0000-0002-6983-2759")
      ),
      License = "MIT + file LICENSE"
    ),
    usethis.destdir = "~/rrr",
    usethis.overwrite = TRUE
)
```

Customize your own defaults for DESCRIPTION.

Tell usethis where to put new projects.

Rely on Git to help you review (and undo?) unwanted changes.

Prework

```
# package development "situation report"
dev_sitrep()
```

```
# git/github "situation report"
git_sitrep()
```

(Quickly) get a source package

- Recommended workflow:
  - `create_from_github("jennybc/regexcite")`
  - `create_from_github("jennybc/regexcite", destdir = "some/folder")`
  - Does "fork and clone" of a Git repo.

- Alternative workflow:
  - `use_course("jennybc/regexcite")`
  - Does a one-time download and unpacking of a ZIP archive.

Core workflows

Let's run a little code from regexcite's README, as if you were going to do some development on it.

```
(x ← "alfa,bravo,charlie,delta")
#> [1] "alfa,bravo,charlie,delta"

str_split_one(x, pattern = ",")
#> Error in str_split_one(x, pattern = ","): could not find function "str_split_one"
```

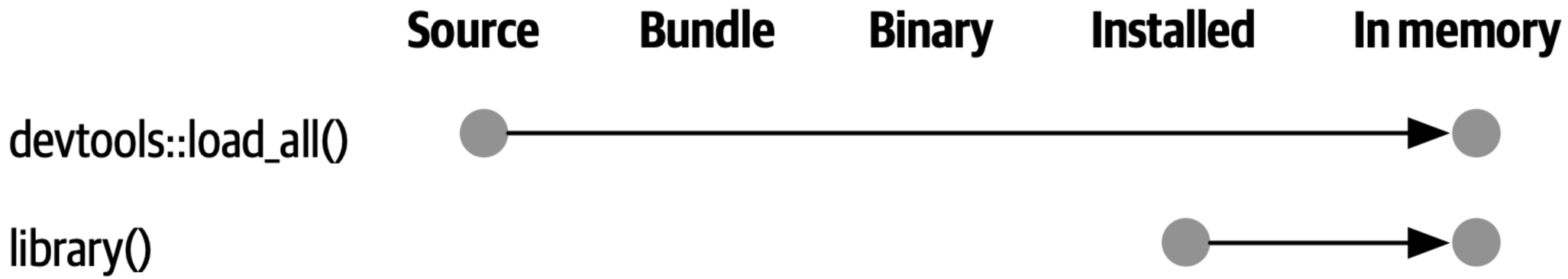
I assume you have NOT installed regexcite.

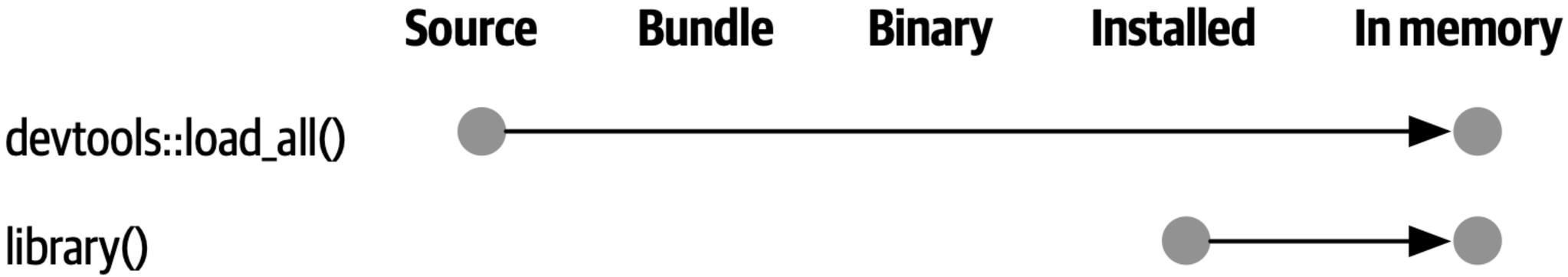
load_all()

Let's run a little code from regexcite's README, as if you were going to do some development on it.

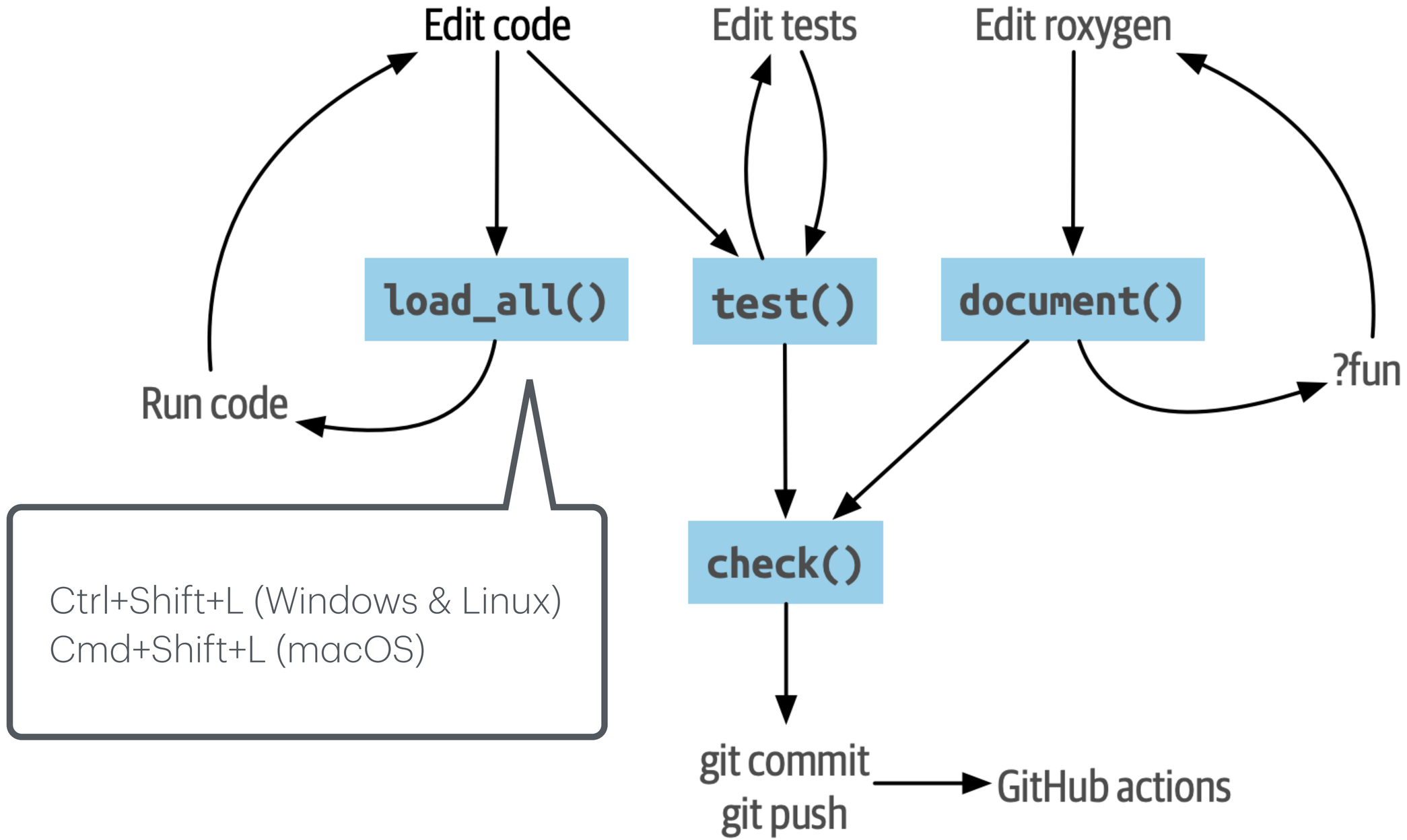
```
load_all()
#> ℹ Loading regexcite

(x ← "alfa,bravo,charlie,delta")
#> [1] "alfa,bravo,charlie,delta"

str_split_one(x, pattern = ",")
#> [1] "alfa"    "bravo"   "charlie" "delta"
```

I assume you still have NOT installed regexcite.


Source
Bundle
Binary
Installed
In memory
devtools::load_all()
library()

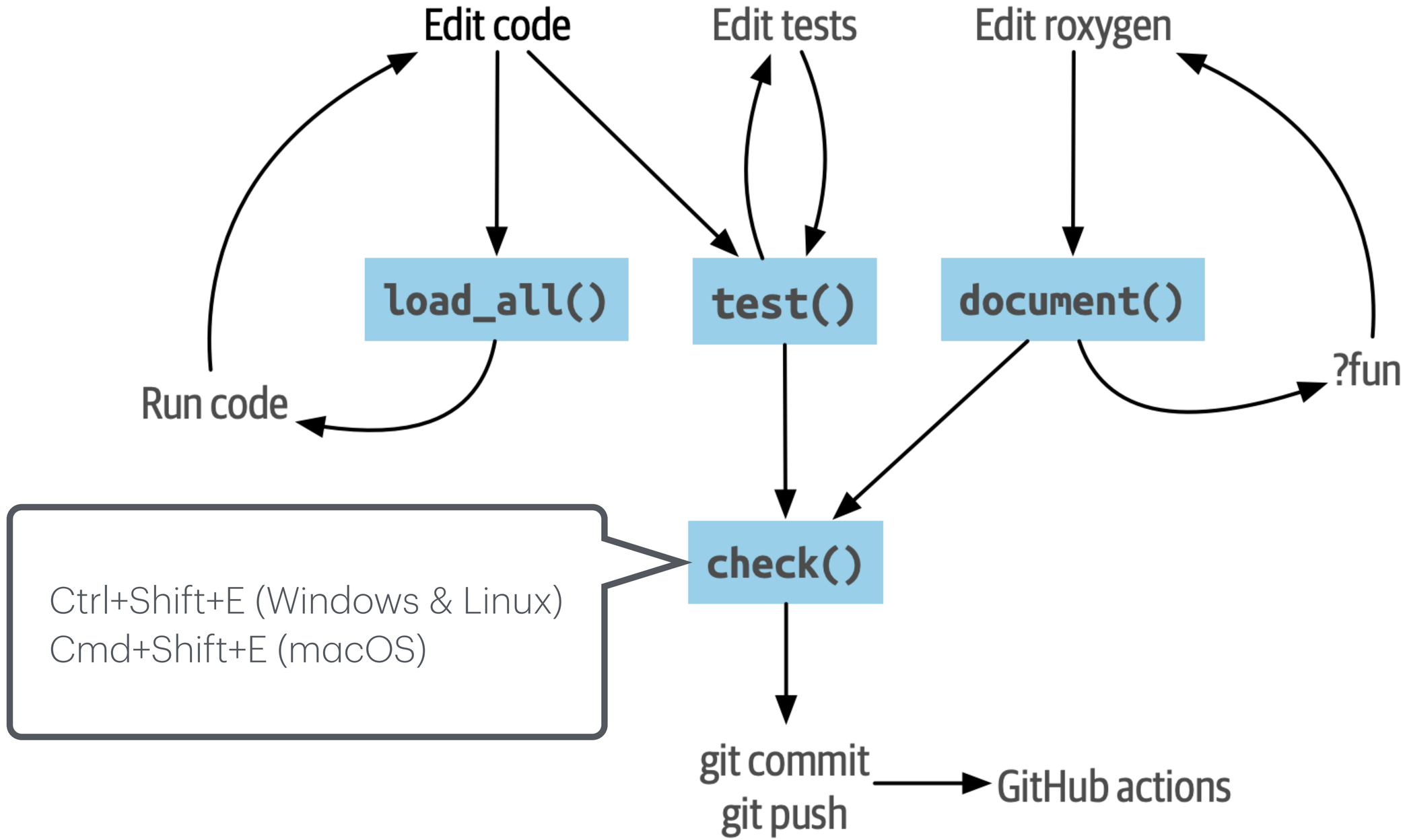


- Simulates building, installing, and attaching your package
- Makes all of the functions in your package available to use
- Makes anything you've imported available to use
- Allows fast iteration of editing and test-driving your functions

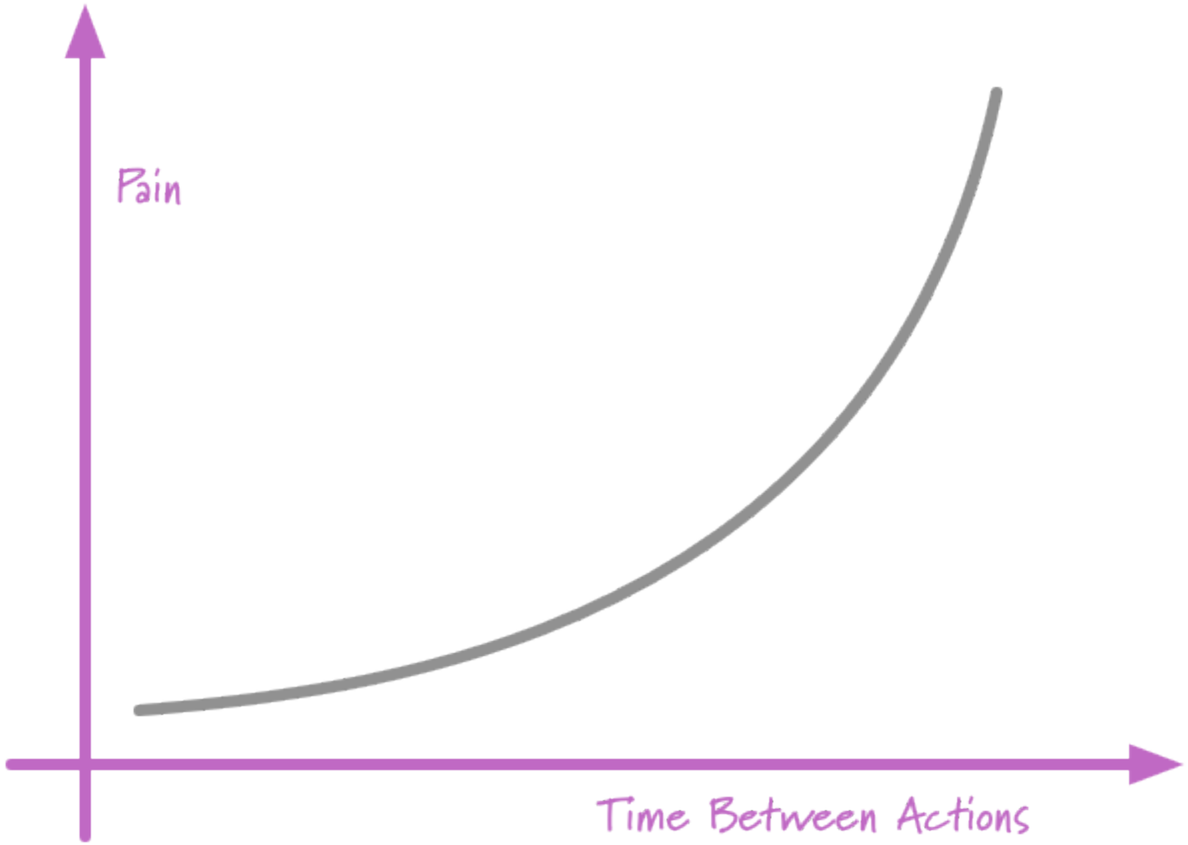

Edit code
Edit tests
Edit roxygen
load_all()
test()
document()
Run code
?fun
check()
git commit
git push
GitHub actions
Ctrl+Shift+L (Windows & Linux)
Cmd+Shift+L (macOS)

check()

```
check()
#> ══ Documenting ═══════════════════════════════════════════════════════════════
#> ℹ Installed roxygen2 version (7.3.2) doesn't match required (7.1.2)
#> ✖ `check()` will not re-document this package
#> ══ Building ══════════════════════════════════════════════════════════════════
 ...
#> ── R CMD build ────────────────────────────────────────────────────────────────
 ...
#>   −  building ‘regexcite_0.0.0.9000.tar.gz’
#>
#> ══ Checking ══════════════════════════════════════════════════════════════════
 ...
#> ── R CMD check ────────────────────────────────────────────────────────────────
 ...
#> ── R CMD check results ──────────────────────────── regexcite 0.0.0.9000 ─────
#> Duration: 11.2s
#>
#> 0 errors ✔ | 0 warnings ✔ | 0 notes ✔
```


Edit code
Edit tests
Edit roxygen
load_all()
test()
document()
Run code
?fun
check()
Ctrl+Shift+E (Windows & Linux)
Cmd+Shift+E (macOS)
git commit
git push
GitHub actions

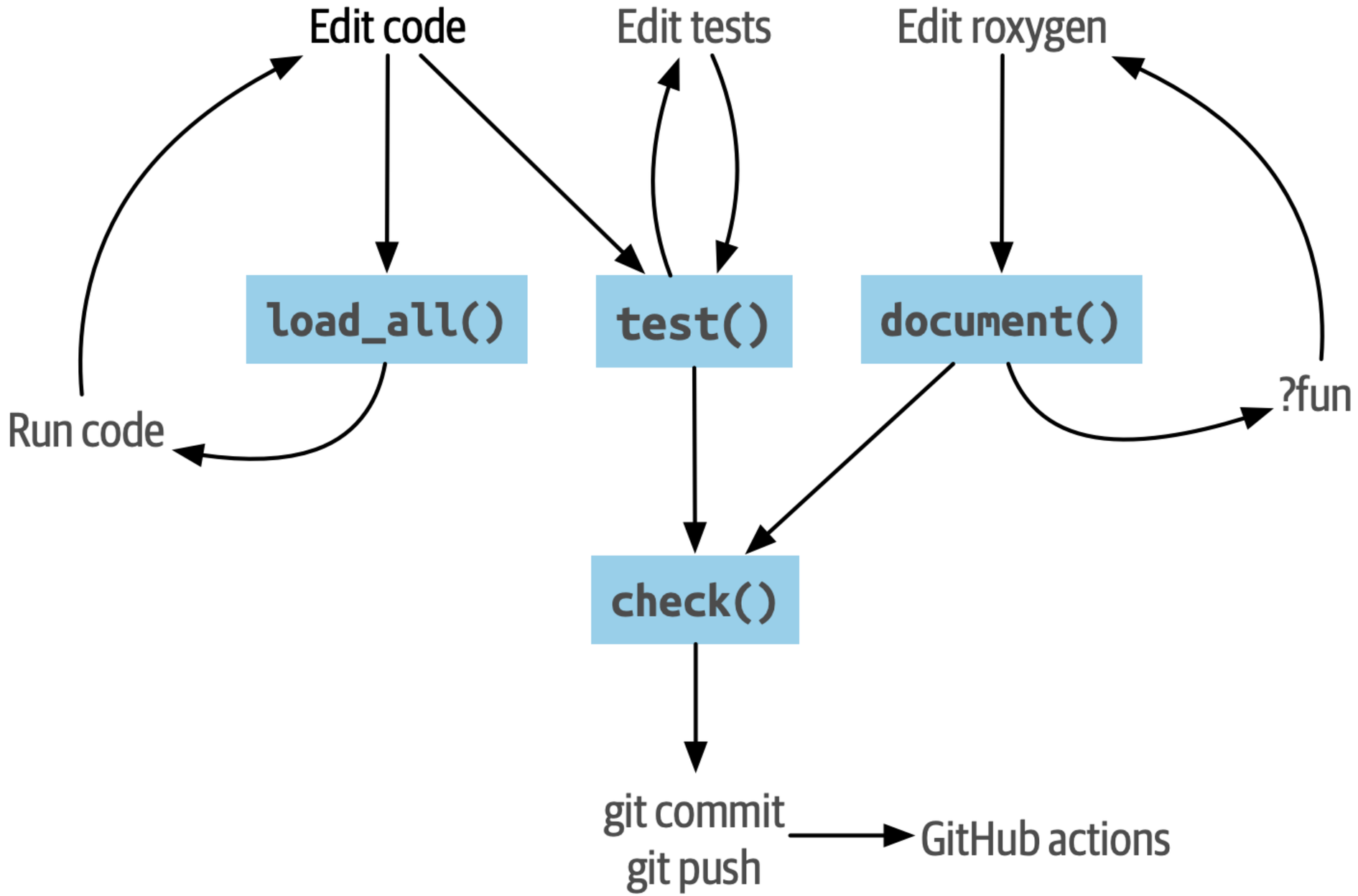
"If it hurts, do it more often."



https://martinfowler.com/bliki/FrequencyReducesDifficulty.html

"If it hurts, do it more often."

- If `check()` goes from passing to failing after you've tinkered with one function or touched 10 lines of code, it's clear what to troubleshoot.
- If you've added 15 functions that all call each other and touched 100s of lines of code, bless your heart.
- Better to learn of a regrettable choice before you've built upon it for days or weeks.


Edit code
Edit tests
Edit roxygen
load_all()
test()
document()
Run code
?fun
check()
git commit
git push
GitHub actions

Your turn

- Grab the source of some other package and take it out for a test drive.
  - `create_from_github("OWNER/REPO")` or `use_course("OWNER/REPO")`
  - Does it pass R CMD check locally for you? `pak::local_install_dev_deps()` is very useful!
- Ideas
  - r-lib/usethis
  - r-lib/rematch2
  - r-lib/sessioninfo
  - r-lib/desc
  - r-lib/withr

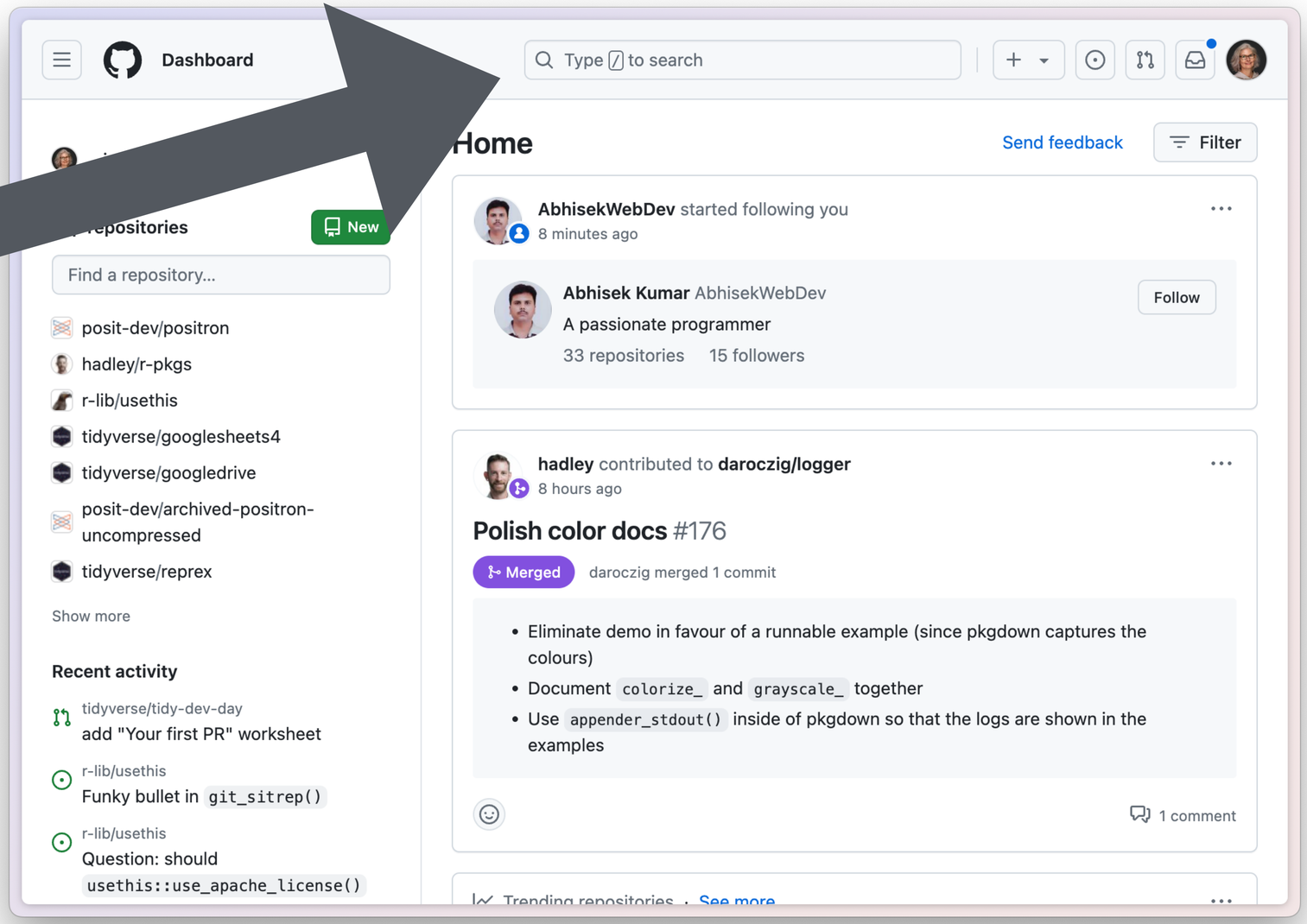
GitHub Code Search

Let Me Google That For You...

Google Search I'm Feeling Lucky

Type a question, click a button.

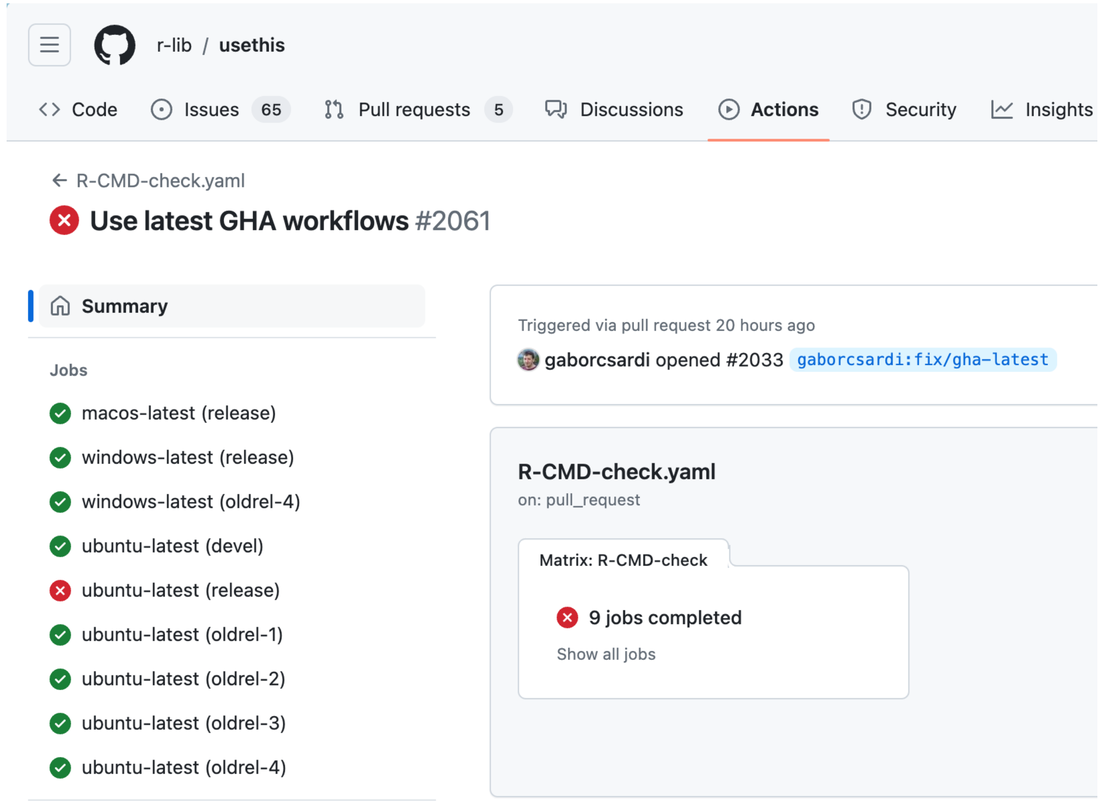
Let me GitHub search that for you



I merged a pull request (PR) to usethis that updated its GitHub Actions (GHA) workflows.

I noticed that recent changes in r-lib/actions actually broke 2 tests in usethis.

But why were they only failing on ubuntu-latest (release)?



ubuntu-latest (release)

```
══ Failed tests ════════════════════════════════════════════════════════════════
── Failure ('test-github-actions.R:106:3'): use_github_action() accepts a name ──
yml$name (`actual`) not identical to "R-CMD-check" (`expected`).

`actual`:   "R-CMD-check.yaml"
`expected`: "R-CMD-check"

── Failure ('test-github-actions.R:132:3'): use_tidy_github_actions() configures the full check and pr commands ──
yml$name (`actual`) not identical to "R-CMD-check" (`expected`).

`actual`:   "R-CMD-check.yaml"
`expected`: "R-CMD-check"

[ FAIL 2 | WARN 0 | SKIP 0 | PASS 936 ]
```

macOS-latest (release)

```
> test_check("usethis")
  Starting 2 test processes
  [ FAIL 0 | WARN 0 | SKIP 31 | PASS 838 ]

  ── Skipped tests (31) ──────────────────────────────────────────────────────
  • No Git user configured (31): 'test-github-actions.R:2:3',
    'test-github-actions.R:29:3', 'test-github-actions.R:45:3',
    'test-github-actions.R:60:3', 'test-github-actions.R:90:3',
    'test-github-actions.R:117:3', ..., 'test-utils-github.R:122:3'
```

Why were they only failing on ubuntu-latest (release)?

Because these tests were only **running** on ubuntu-latest (release).

But why is that?

```
test_that("use_github_action() accepts a name", {
  skip_if_no_git_user()
   ...
}
```

Ah, it must be because no Git user is configured in those GHA jobs.

But I usually set that up! I must have goofed up and allowed those lines to get deleted.*

Hmmm, what's the usual way to configure the Git user in a GHA workflow?

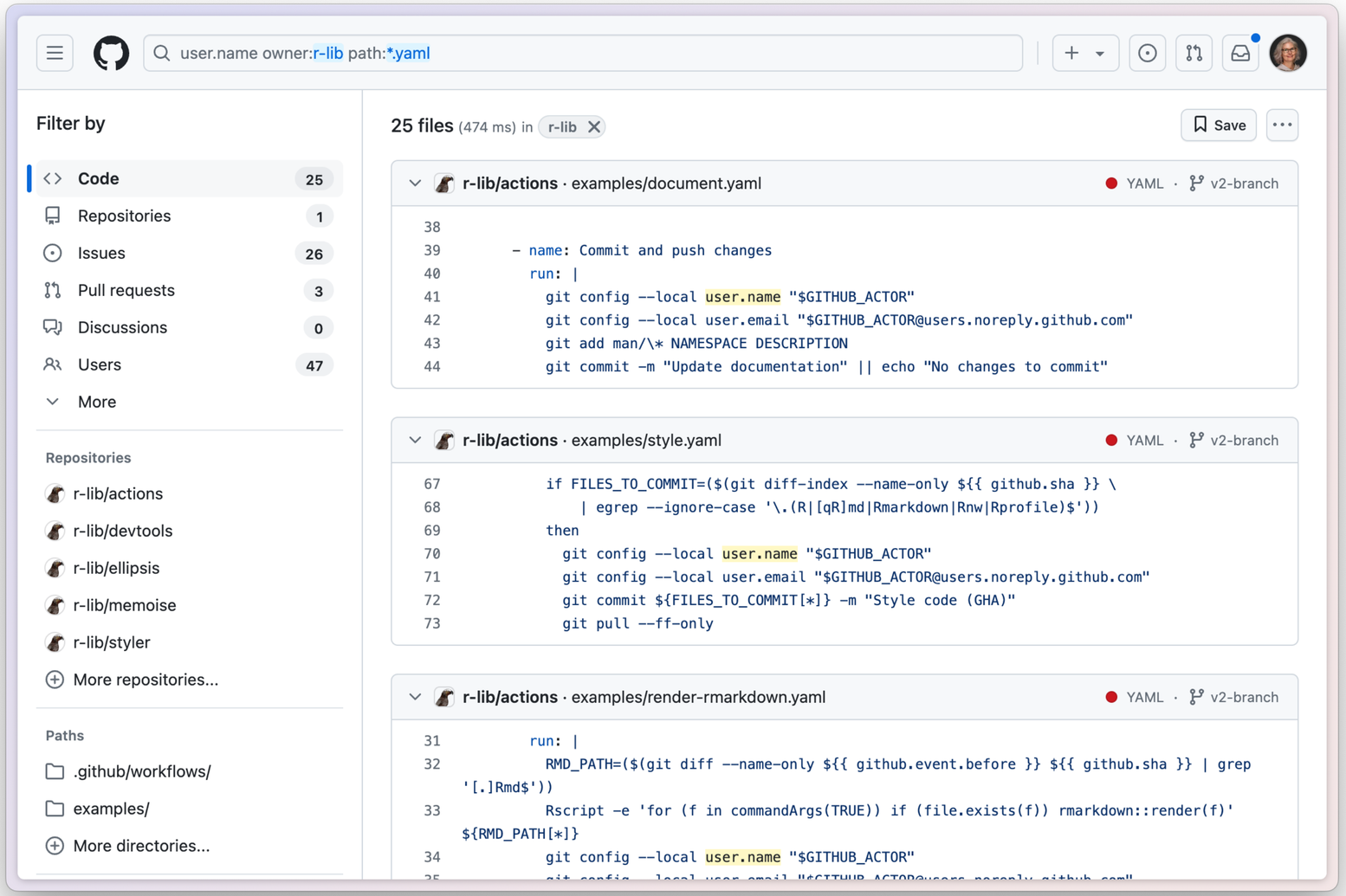
* This was, in fact, not true. But I'm just taking you on my journey.

What's the usual way to configure the Git user in a GHA workflow?

- Google it?
- Ask ChatGPT or Co-Pilot?
- Read documentation for GitHub Actions?
- Read a git man page?
- Post on social media with #rstats hashtag?
- Ask in Slack?
- Ask my local expert?

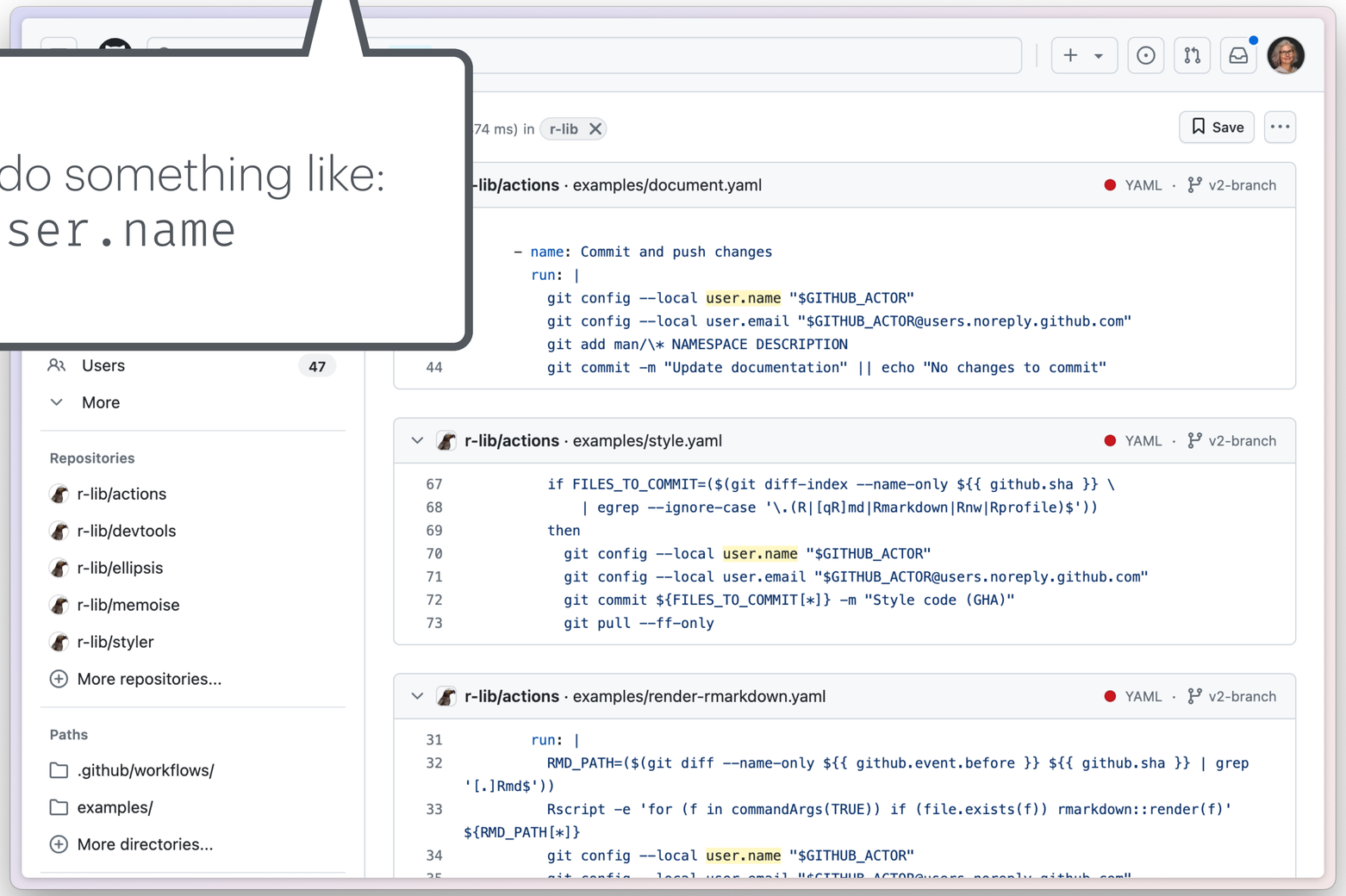
- **Search GitHub to find working examples of exactly the thing I want to do.**

user.name owner:r-lib path:*.yaml

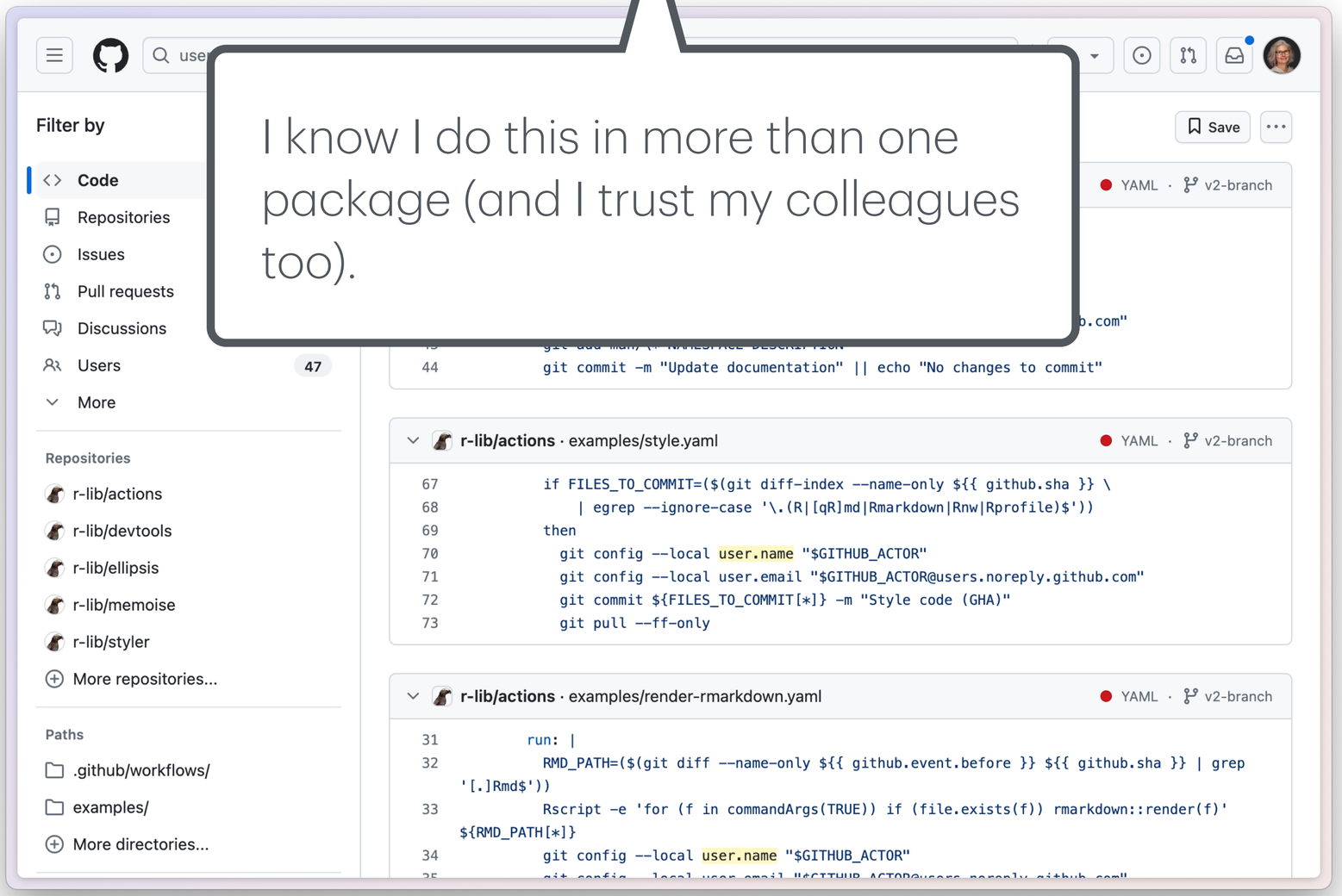


`user.name owner:r-lib path:*.yaml`

I know that you do something like:
`git config user.name`



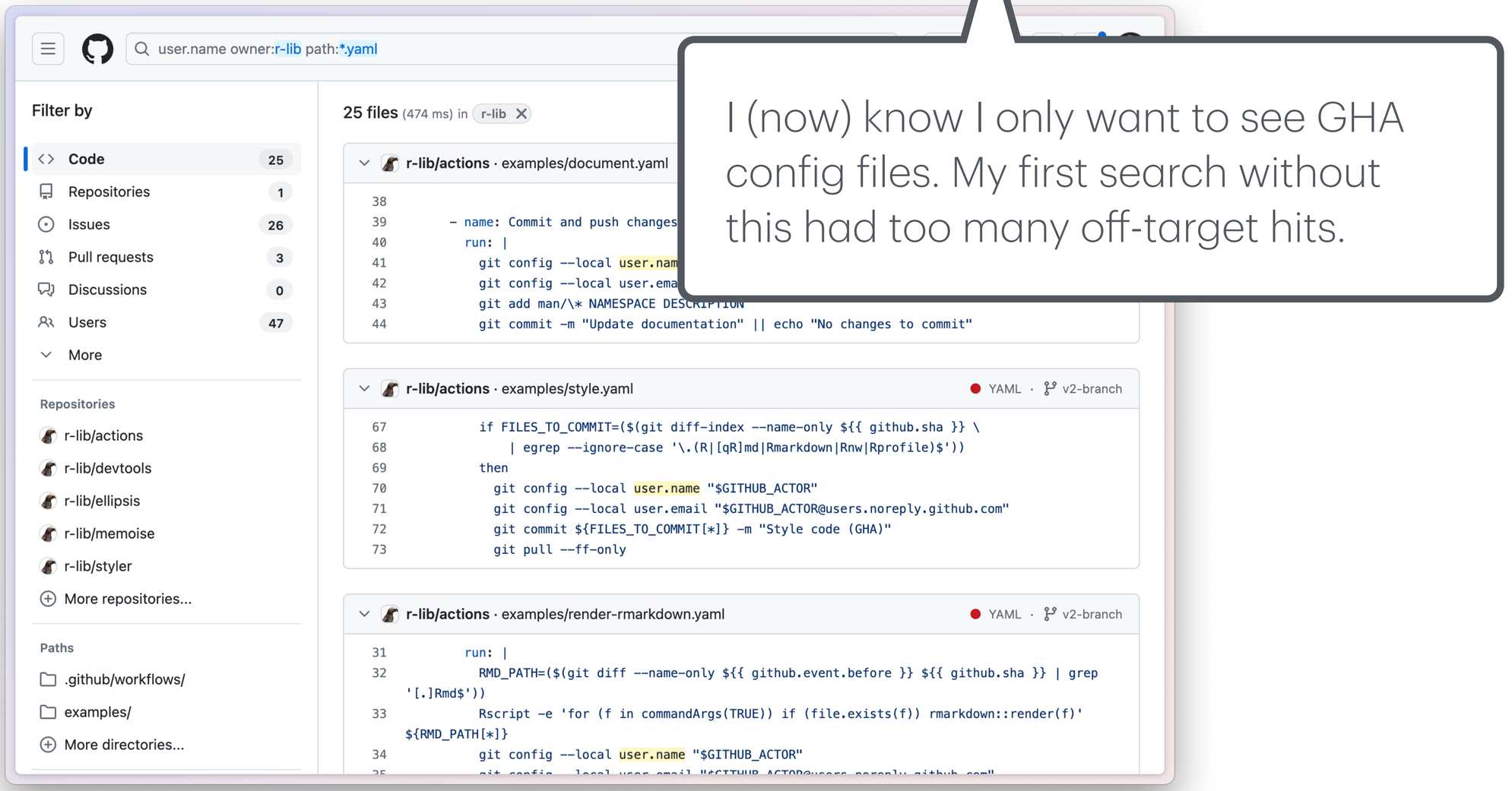
`user.name owner:r-lib path:*.yaml`



I know I do this in more than one package (and I trust my colleagues too).

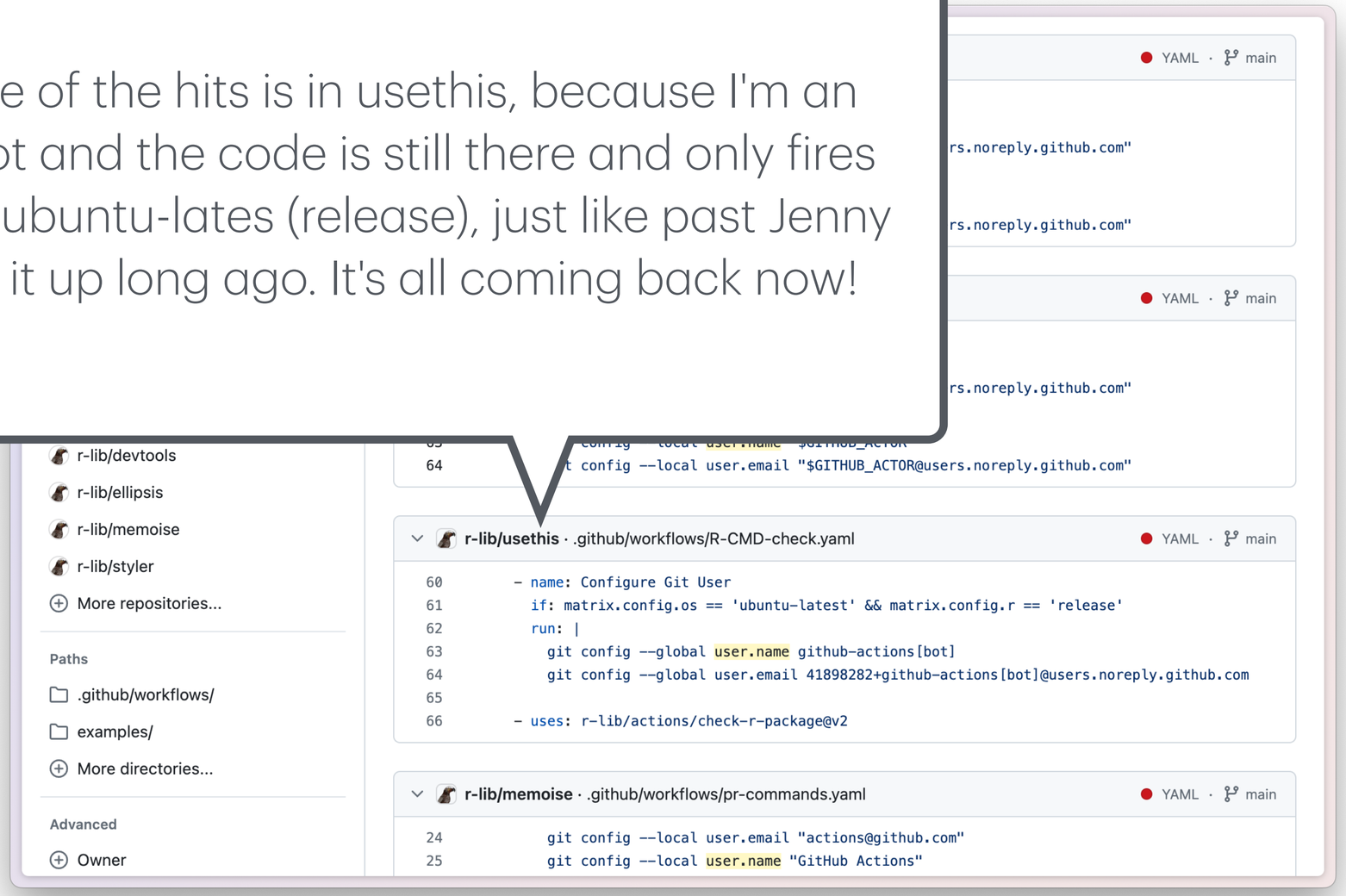
* `owner` seems to be an alias for the ?more correct? `user`

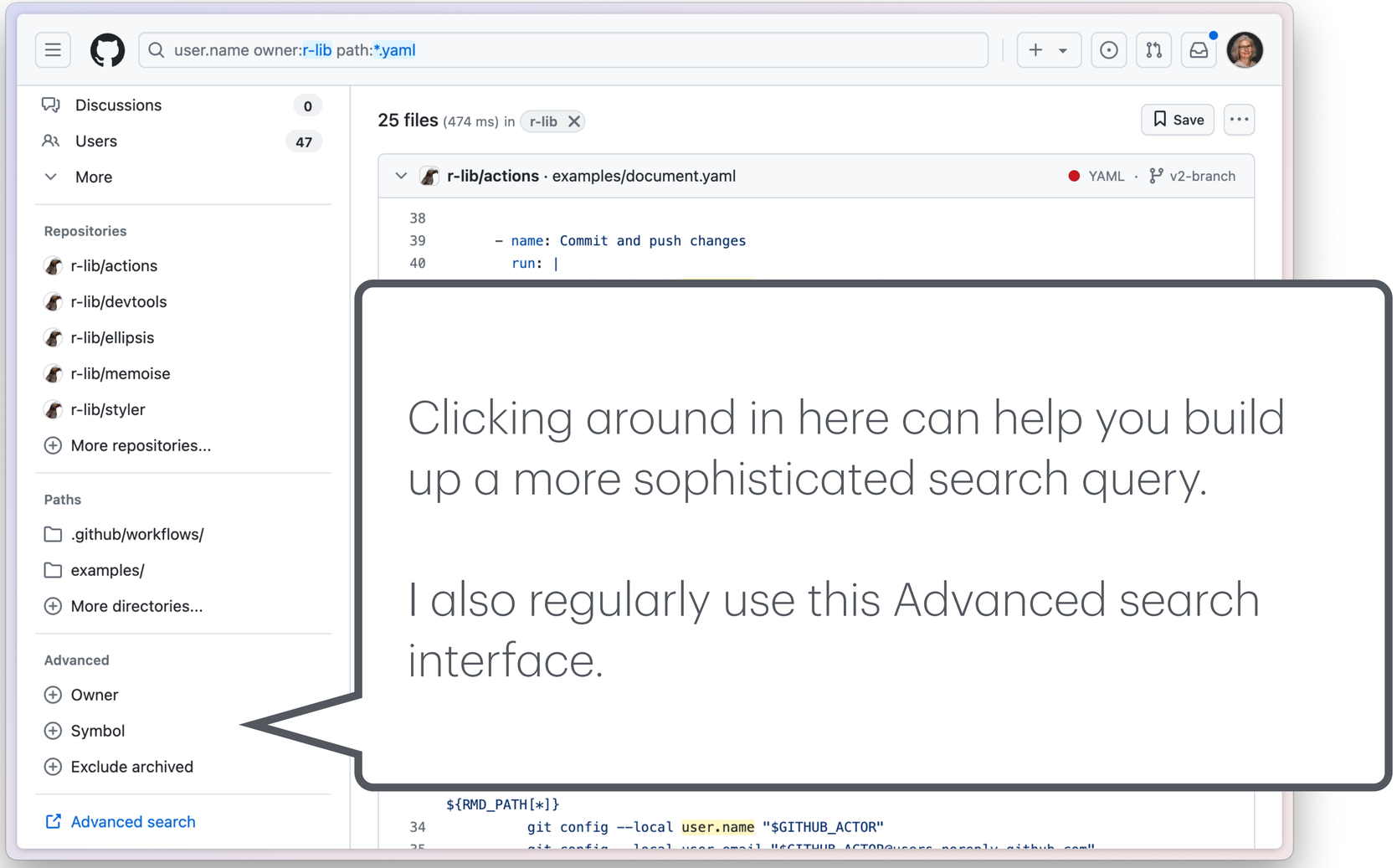
`user.name owner:r-lib path:*.yaml`



I (now) know I only want to see GHA config files. My first search without this had too many off-target hits.

One of the hits is in usethis, because I'm an idiot and the code is still there and only fires on ubuntu-lates (release), just like past Jenny set it up long ago. It's all coming back now!

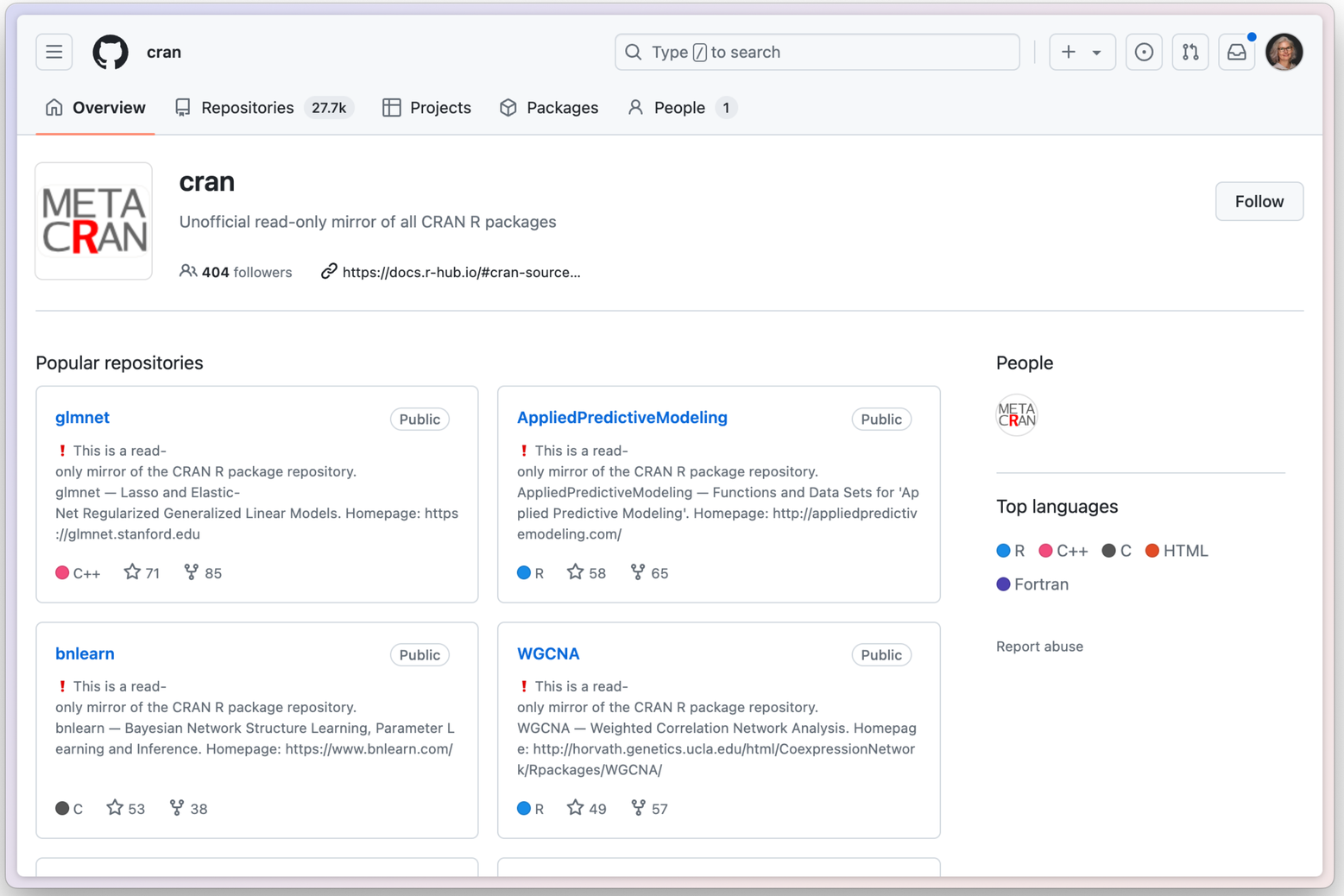



user.name owner:r-lib path:*.yaml
Discussions 0
Users 47
More
Repositories
r-lib/actions
r-lib/devtools
r-lib/ellipsis
r-lib/memoise
r-lib/styler
More repositories...
Paths
.github/workflows/
examples/
More directories...
Advanced
Owner
Symbol
Exclude archived
Advanced search
25 files (474 ms) in r-lib
Save
r-lib/actions · examples/document.yaml
YAML · v2-branch
38
39 - name: Commit and push changes
40 run: |
Clicking around in here can help you build up a more sophisticated search query.
I also regularly use this Advanced search interface.
${RMD_PATH[*]}
34 git config --local user.name "$GITHUB_ACTOR"

GitHub code search pro tips

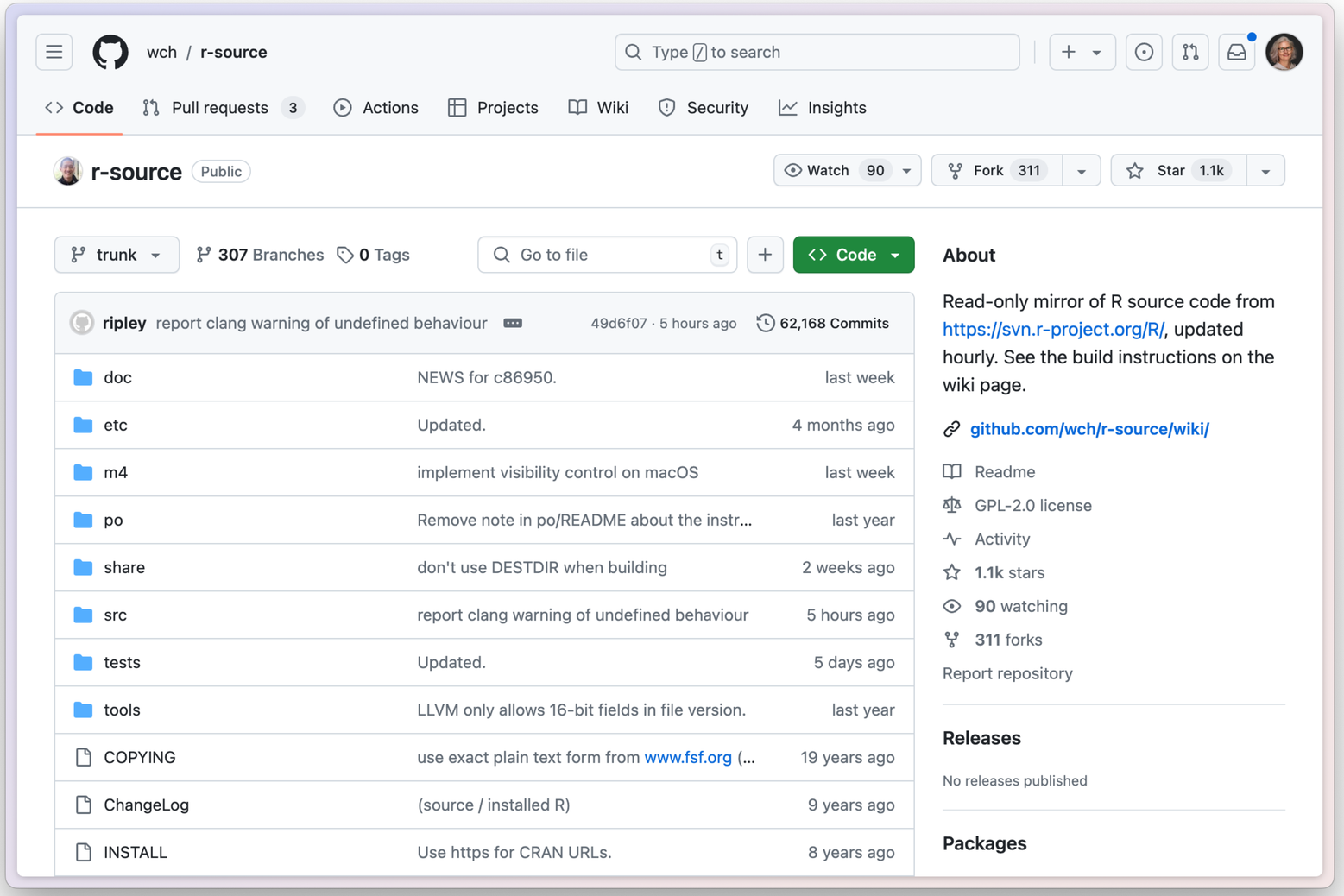
- `user:cran` is super powerful for searching across the source code of every package on CRAN
- I am biased, but I often use `user:tidyverse` or `user:r-lib` to focus on solutions by folks who generally approach R and package development the way I do.
- GitHub search is a tool where consulting the docs regularly really pays off:
  - Understanding GitHub Code Search syntax
- `repo:wch/r-source` narrows the search to a GitHub mirror of the R source
- Other qualifiers I use a lot:
  - `path`
  - `language`
- A qualifier that is newly relevant for R: `symbol`

https://github.com/cran `user:cran`



https://github.com/wch/r-source

`repo:wch/r-source`



Your turn

You can find some GitHub code search challenges in the workshop repo:

https://github.com/posit-conf-2024/pkg-dev

Feel free to follow your heart and come up with your own, based on questions you have.

Big Idea	Featured implementation
Become fluid at package development and installation	devtools pak
Eliminate repetitive tasks	Curate your `.Rprofile`
Regular health checks	`dev_sitrep()` `git_sitrep()`
Make experimentation cheap	`create_from_github()` `create_package()`
Test drive your package often	`load_all()`
Check your package often	`check()`
Study how other people do things	GitHub code search