Fundamentals of Package Development

The Whole Game

Let’s make a package together

We will:

- Create a simple package
- Use git to track our changes
- Push the code to a repository on GitHub
- Create tests for our functions
- Create documentation for our functions
- Create a documentation website (if we have time)

Explore our end goal

On GitHub

- This is the package in “Source” form:
  - Package code in R/
  - Function documentation files in `man/`
  - Package vignettes in `vignettes/`
  - `DESCRIPTION`
  - `NAMESPACE`

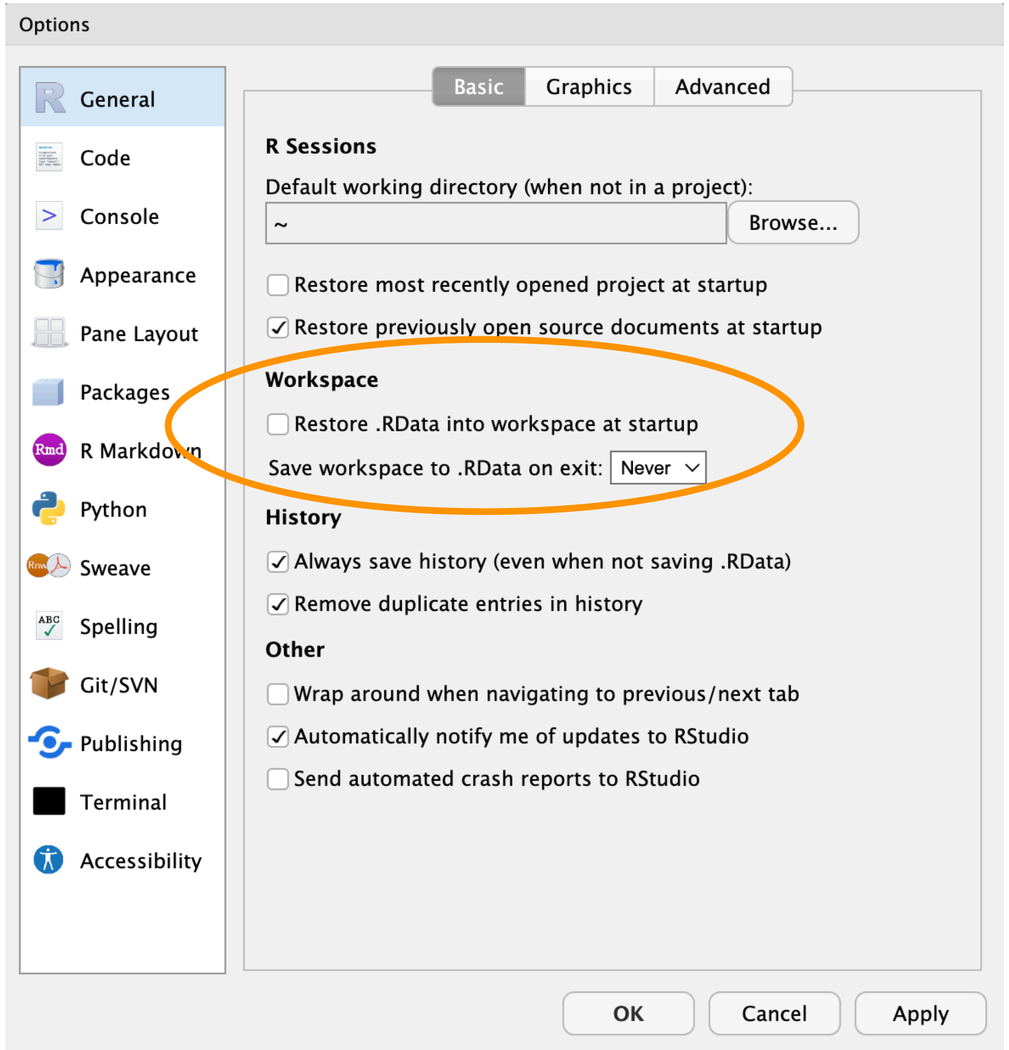
Tools 🧰

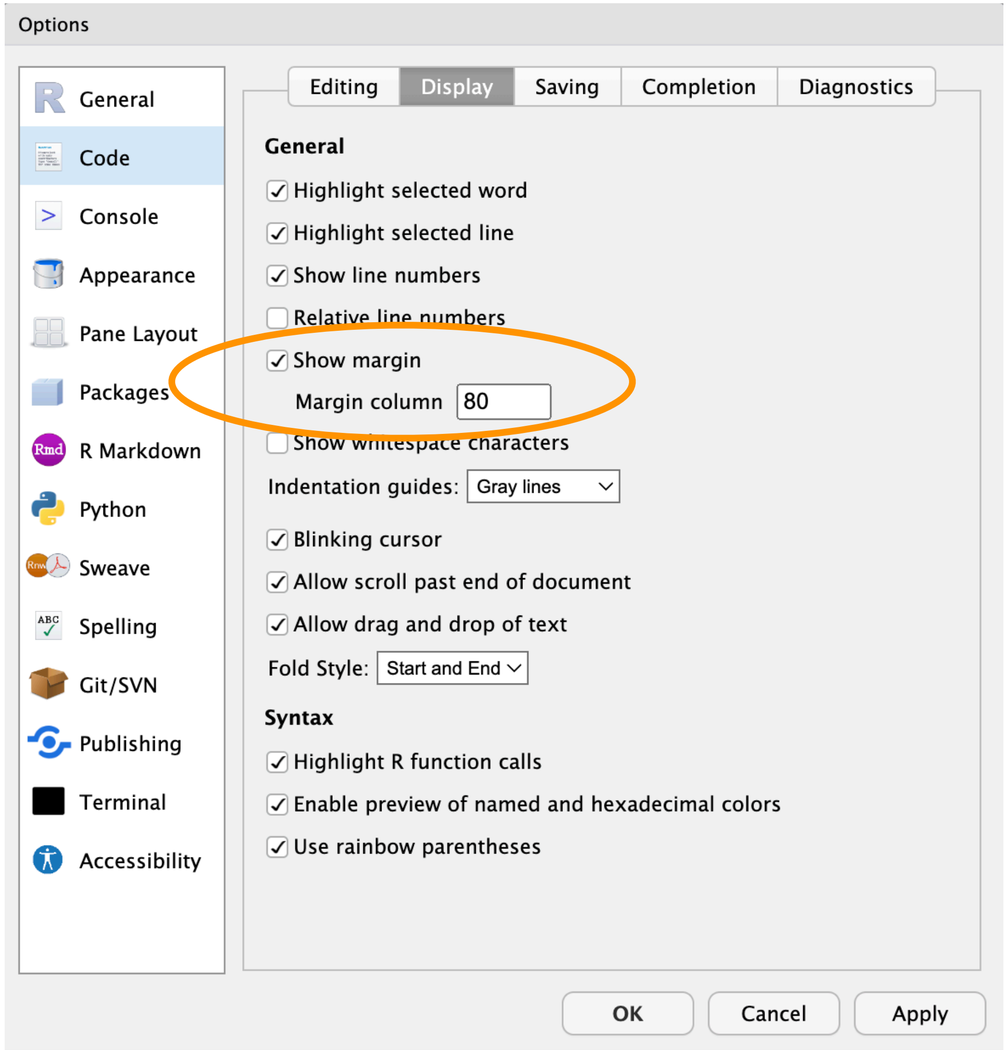
- R >= 4.3.0
- RStudio (https://posit.co/download/rstudio-desktop/)
- Packages:

```
install.packages(
  c("devtools", "roxygen2", "testthat", "knitr")
)
```

Configure RStudio

Tools > Global Options





Load devtools 📦

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

packageVersion("devtools")
#> [1] '2.4.5'
```

- Update if necessary!
- Provides a suite of functions to aid package development
- Loads **usethis**, the source of most functions we will be using

create_package() 📦

```
create_package("~/Desktop/mypackage")
#> ✔ Creating '/Users/jane/Desktop/mypackage/'
#> ✔ Setting active project to '/Users/jane/Desktop/mypackage'
#> ✔ Creating 'R/'
#> ✔ Writing 'DESCRIPTION'
#> Package: mypackage
#> Title: What the Package Does (One Line, Title Case)
#> Version: 0.0.0.9000
#> Authors@R (parsed):
#>     * First Last <first.last@example.com> [aut, cre] (YOUR-ORCID-ID)
#> Description: What the package does (one paragraph).
#> License: `use_mit_license()`, `use_gpl3_license()` or friends to pick a
license
#> Encoding: UTF-8
#> Roxygen: list(markdown = TRUE)
#> RoxygenNote: 7.2.3
#> ✔ Writing 'NAMESPACE'
#> ✔ Writing 'mypackage.Rproj'
#> ✔ Adding '^mypackage\\.Rproj$' to '.Rbuildignore'
#> ✔ Adding '.Rproj.user' to '.gitignore'
#> ✔ Adding '^\\.Rproj\\.user$' to '.Rbuildignore'
#> ✔ Setting active project to '<no active project>'
```

create_package() 📦

```
create_package("~/Desktop/mypackage")

├── .Rbuildignore
├── .Rproj.user
├── .gitignore
├── DESCRIPTION
├── NAMESPACE
├── R
└── mypackage.Rproj
```

- Creates directory
- Sets up basic package skeleton
- Opens a new RStudio project
- Activates "build" pane in RStudio

use_git()

- `use_git_config(`
  `user.name = "Jane Doe",`
  `user.email = "jane@example.org"`
  `)`

- `use_git()`

- Turns package directory into a git repository
- Commits your files (with a prompt)
- Restarts RStudio (with a prompt)
- Activates "git" pane in RStudio

```
use_git()

#> ✔ Setting active project to
#>       '/Users/Jane/rrr/mypackage'
#> ✔ Adding '.Rhistory', '.Rdata',
#>     '.httr-oauth', '.DS_Store',
#>     '.quarto' to '.gitignore'
#> There are 5 uncommitted files:
#> * '.gitignore'
#> * '.Rbuildignore'
#> * 'DESCRIPTION'
#> * 'metrify.Rproj'
#> * 'NAMESPACE'
#> Is it ok to commit them?
#>
#> 1: Absolutely not
#> 2: Not now
#> 3: Yeah
```

`use_github()`

Put your package code on GitHub

- Prerequisites:
  - GitHub account
  - `create_github_token()` - follow instructions
  - `gitcreds::gitcreds_set()` - paste PAT
  - `git_sitrep()` - verify

`use_github()` - push content to new repository on GitHub

use_r()

Write your first function

- R code goes in R/
- Name the file after the function it defines

```
use_r("my-fun")

#> ✔ Setting active project to '/Users/jane/rrr/mypackage'
#> • Edit 'R/my-fun.R'
```

- Put the definition of your function (and only the definition!) in this file

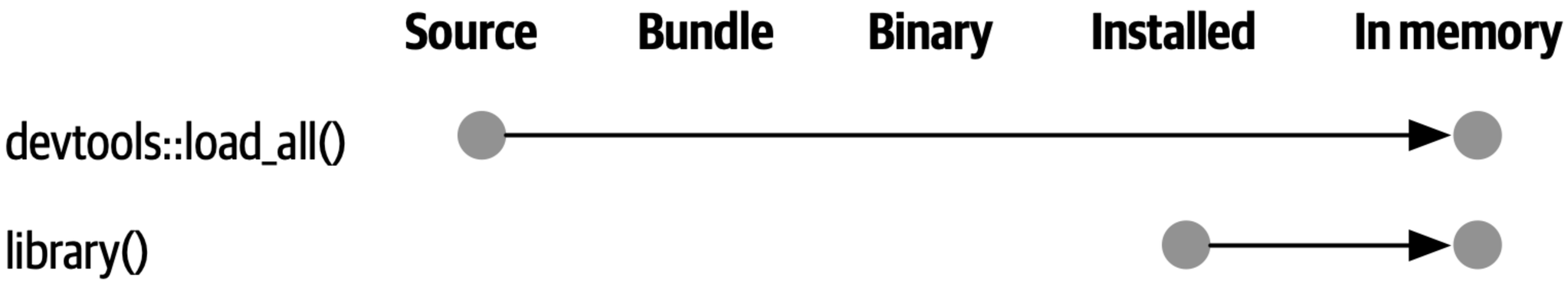
Test your function in the new package

But how?

- ~~source("R/my-fun.R")~~
- ~~Send function to console using RStudio (Ctrl/CMD+Return)~~
- `devtools::load_all()`

`load_all()`

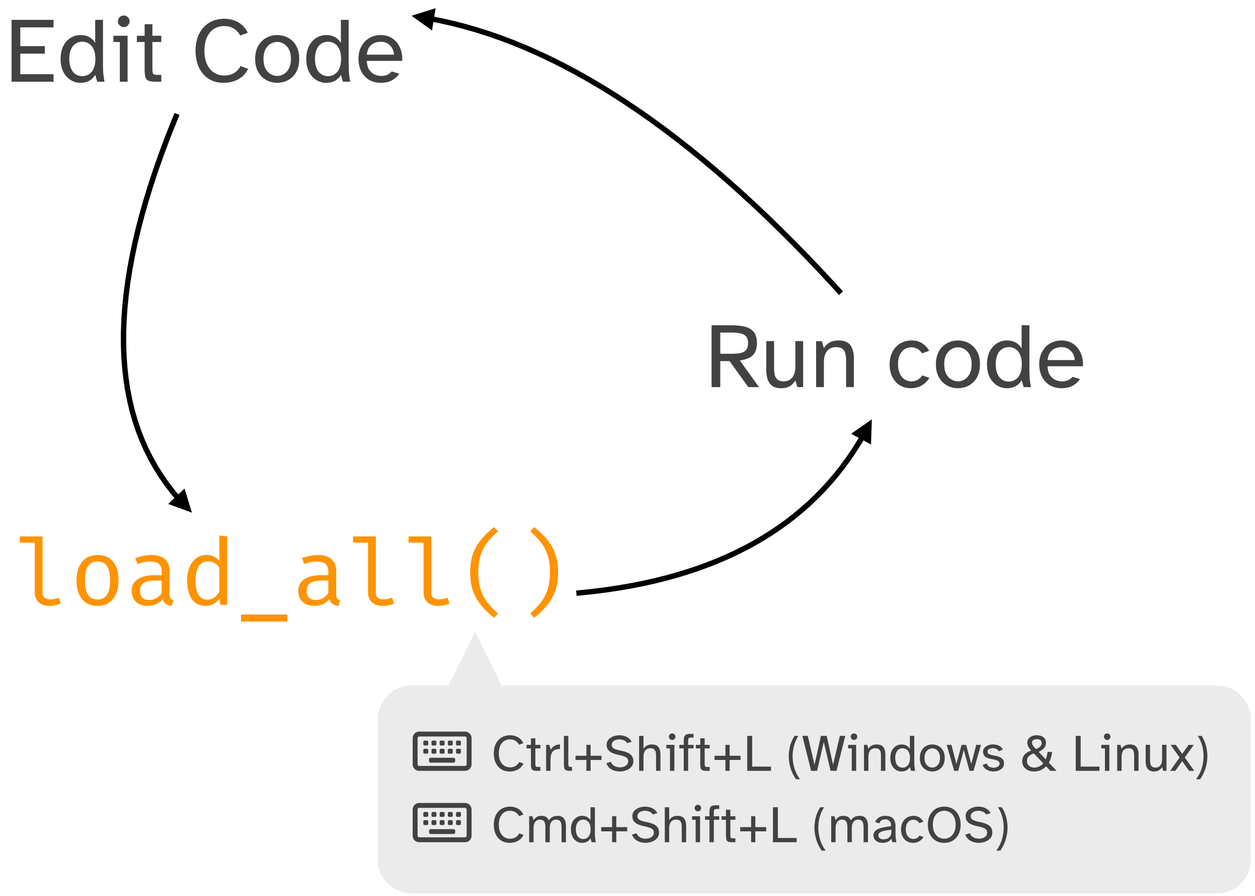
Simulates building, installing, and attaching your package



- Makes all of the functions from your package immediately available to use
- Allows fast iteration of editing and test-driving your functions
- Good reflection of how users will interact with your package*

*With `load_all()`, unexported functions are also made available, which they are not via install and attach

Workflow


Edit Code
Run code
load_all()
Ctrl+Shift+L (Windows & Linux)
Cmd+Shift+L (macOS)

Try it out, and commit your changes

check()

Run R CMD check from within R

```
check()

#> ── R CMD check results ──────────────────────────────────────────────
#> Duration: 3.1s
#>
#> ❯ checking DESCRIPTION meta-information ... WARNING
#>   Invalid license file pointers: LICENSE
#>
#> 0 errors ✔ | 1 warning ✖ | 0 notes ✔
```

- check() early and often
- Reduce future pain by catching problems early*

*If it hurts, do it more often, by Martin Fowler:
https://martinfowler.com/bliki/FrequencyReducesDifficulty.html

R CMD check

3 types of messages

- **ERRORs:** Severe problems - always fix.
- **WARNINGs:** Problems that you should fix, and must fix if you're planning to submit to CRAN.
- **NOTEs:** Mild problems or, in a few cases, just an observation.
  - When submitting to CRAN, try to eliminate all NOTEs.

The DESCRIPTION file

Package metadata

- Make yourself the author
  - Name & Email
  - ORCID (optional)
- Write descriptive
  - Title:
  - Description:

Ctrl+.

start typing DESCRIPTION

```
Package: mypackage
Title: What the Package Does (One Line, Title Case)
Version: 0.0.0.9000
Authors@R: person(
     "First", "Last", ,
     "first.last@example.com",
     role = c("aut", "cre"),
     comment = c(ORCID = “YOUR-ORCID-ID")
   )
Description: What the package does (one paragraph).
License: `use_mit_license()`, `use_gpl3_license()` or
   friends to pick a license
Encoding: UTF-8
Roxygen: list(markdown = TRUE)
RoxygenNote: 7.2.3
```

Licenses

`use_*_license()`

- Permissive:
  - **MIT:** simple and permissive.
  - **Apache 2.0:** MIT + provides patent protection.
- Copyleft:
  - Requires sharing of improvements.
  - **GPL (v2 or v3)**
  - **AGPL, LGPL** (v2.1 or v3)
- Creative commons licenses:
  - Appropriate for data packages.
  - **CC0:** dedicated to public domain.
  - **CC-BY:** Free to share and adapt, must give appropriate credit.

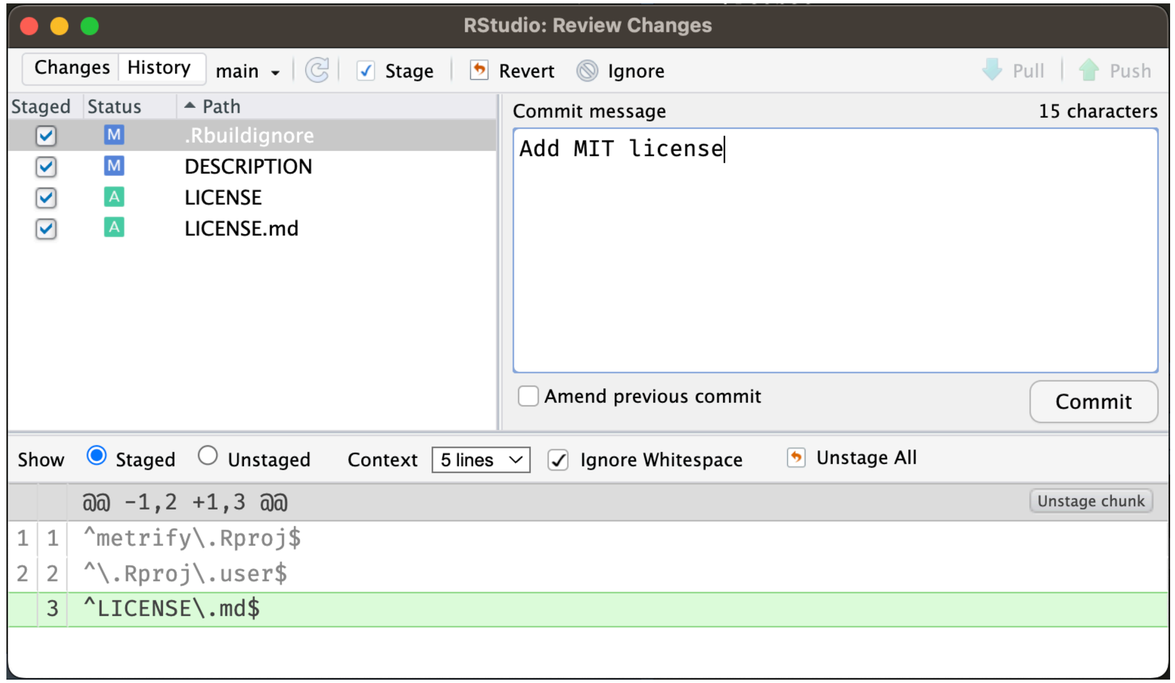
use_mit_license()

```
✔ Adding 'MIT + file LICENSE' to License
✔ Writing 'LICENSE'
✔ Writing 'LICENSE.md'
✔ Adding '^LICENSE\\.md$' to '.Rbuildignore'
```

Commit changes to git

```
$ git add DESCRIPTION \
          LICENSE \
          LICENSE.md \
          .Rbuildignore

$ git commit -m "Add MIT license"
```



edit_r_profile()

Set default DESCRIPTION values

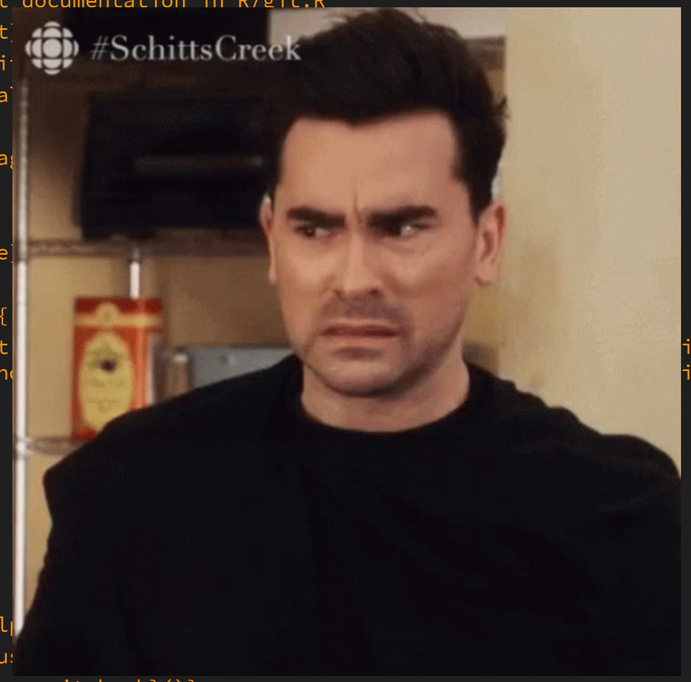
```r
if (interactive()) {
  # Load package dev packages:
  suppressMessages(require("devtools"))
  suppressMessages(require("testthat"))

  # Set usethis options:
  options(
    "Authors@R" = utils::person(
      "Jane", "Doe",
      email = "jane@example.com",
      role = c("aut", "cre"),
      comment = c(ORCID = "0000-1111-2222-3333")
    ),
    License = "MIT + file LICENSE"
  )
}
```

Documentation

man/*.Rd

```
% Generated by roxygen2: do not edit by hand
% Please edit documentation in R/git.R
\name{use_git
\alias{use_gi
\title{Initia
\usage{
use_git(messa
}
\arguments{
\item{message
}
\description{
\code{use_git                                   iles to
\code{.gitign                                   it.
}
\examples{
\dontrun{
use_git()
}
}
\seealso{
Other git hel
\code{\link{u
\code{\link{use_git_hook}()},
\code{\link{use_git_ignore}()}
}
\concept{git helpers}
```



Function documentation

```
> ?use_git

use_git                package:usethis                R Documentation

Initialise a git repository

Description:

     'use_git()' initialises a Git repository and adds important files
     to '.gitignore'. If user consents, it also makes an initial
     commit.

Usage:

     use_git(message = "Initial commit")

Arguments:

 message: Message to use for first commit.

See Also:

     Other git helpers: 'use_git_config()', 'use_git_hook()',
     'use_git_ignore()'

Examples:

     ## Not run:

     use_git()
     ## End(Not run)
```

roxygen2

- RStudio: *Code* > *Insert Roxygen Skeleton*
- Special comments (`#'`) above function definition in `R/*.R`
  - Title
  - Parameter descriptions (`@param`)
  - Return value (`@return`)
  - Export tag (`@export`)
  - Example usage (`@examples`)
  - ...
- Markdown-like syntax
- Keep documentation with code!

⌨ Cmd/Ctrl+Alt+Shift+R

With cursor in function definition

```
#' Title
#'
#' @param x
#' @param y
#'
#' @return
#' @export
#'
#' @examples
add <- function(x, y) {
  x+y
}
```

document()

R/use-git.R

```r
#' Initialise a git repository
#'
#' `use_git()` initialises a Git
#' repository and adds important
#' files to `.gitignore`. If user
#' consents, it also makes an
#' initial commit.
#'
#' @param message Message to use
#'    for first commit.
#' @export
#' @examples
#' \dontrun{
#'   use_git()
#' }
use_git <- function(message = "Initial
commit") {
  . . .
}
```

Cmd/Ctrl
+Shift+D

document()

man/*.Rd

```rd
% Generated by roxygen2: do not edit by hand
% Please edit documentation in R/git.R
\name{use_git}
\alias{use_git}
\title{Initialise a git repository}
\usage{
use_git(message = "Initial commit")
}
\arguments{
\item{message}{Message to use for first commit.}
}
\description{
\code{use_git()} initialises a Git repository and adds
important files to \code{.gitignore}. If user consents,
it also makes an initial commit.
}
\examples{
\dontrun{
use_git()
}
}
```

Create roxygen comments

- Cursor in function definition
- Insert roxygen skeleton

⌨ Cmd/Ctrl+Alt+Shift+R

- Complete the roxygen fields
- `document()`

⌨ Cmd/Ctrl+Shift+D

- `?myfunction`
- 🎉

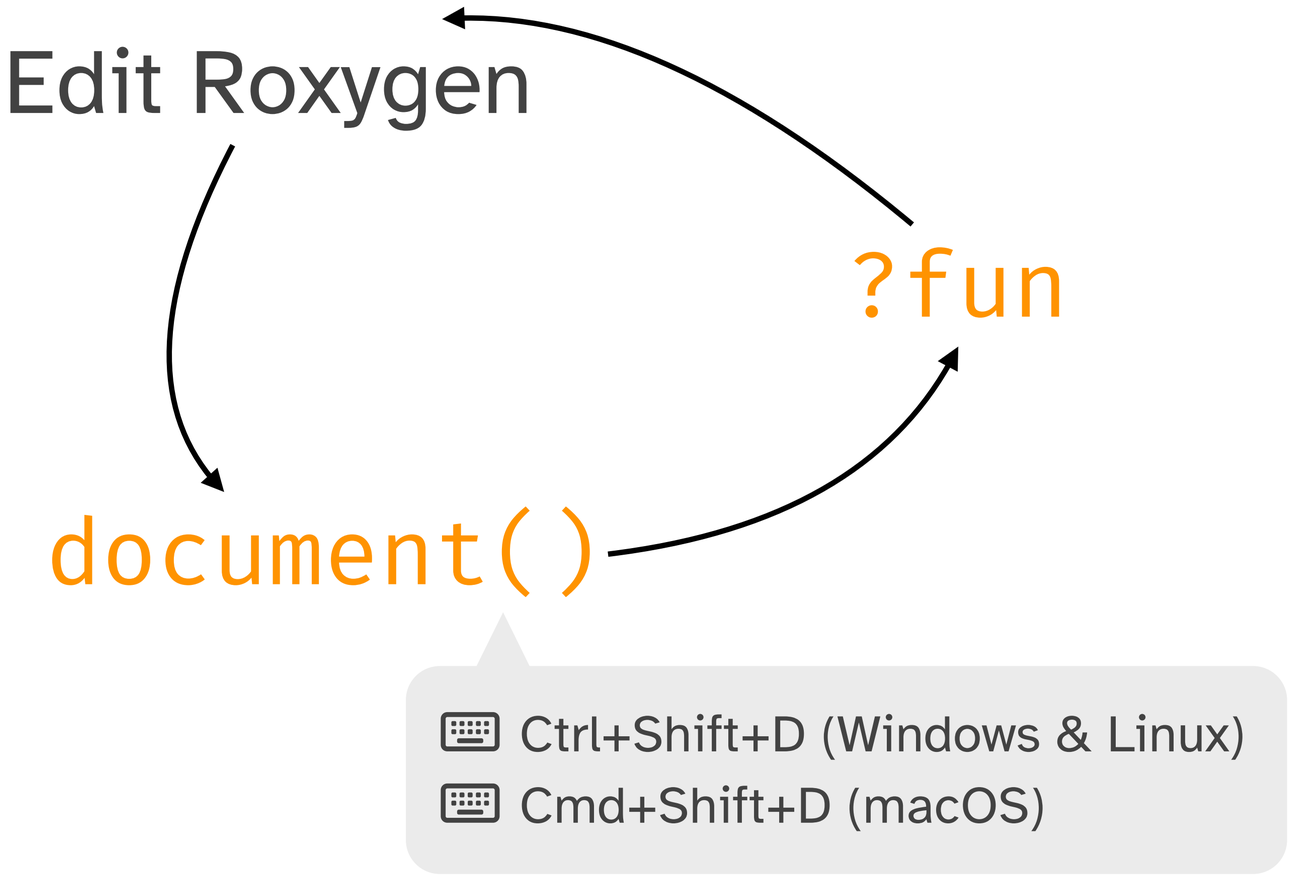
Commit your changes
Push to Github

NAMESPACE

An introduction

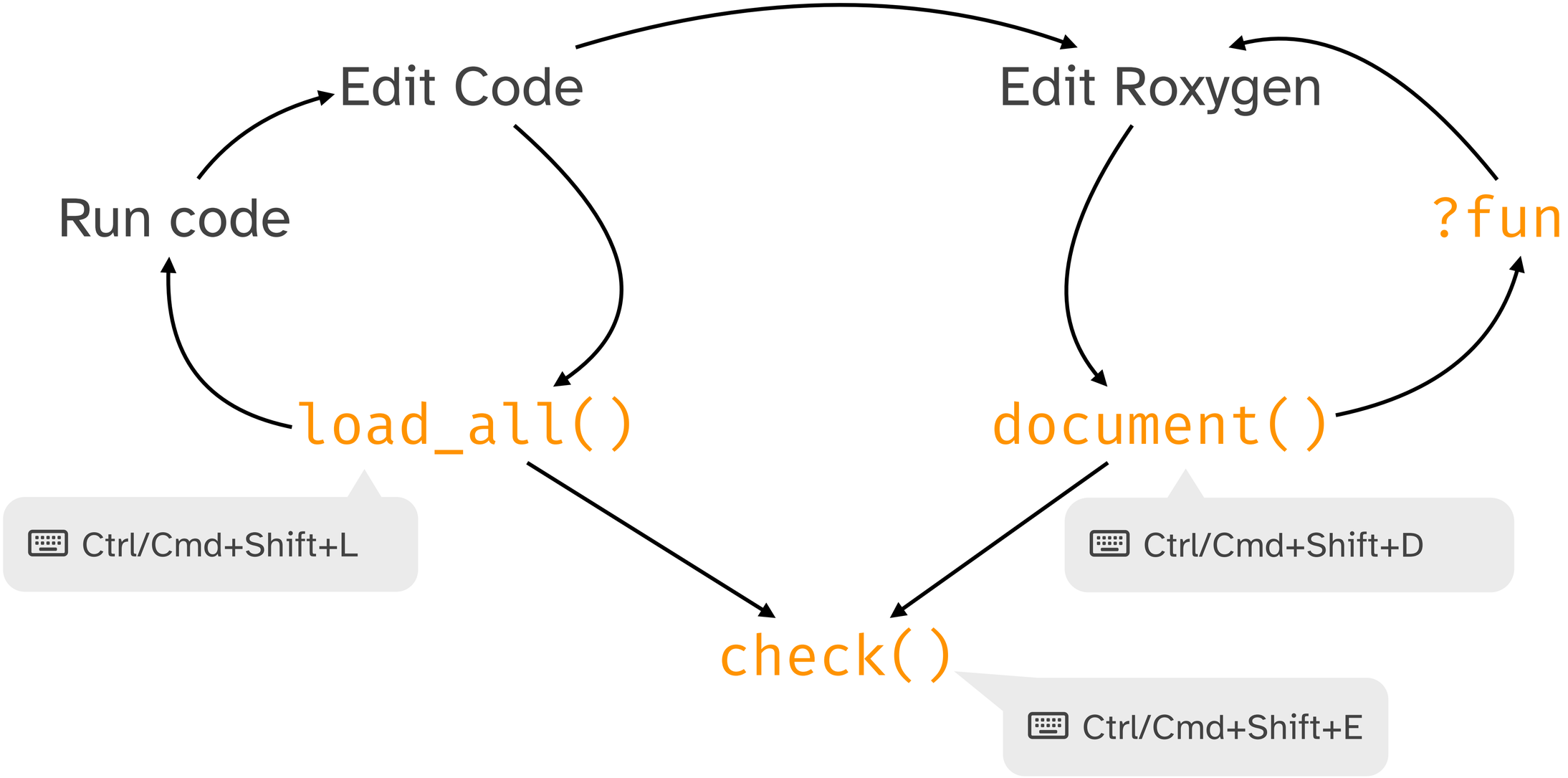
- Lists R objects that are:
  - **Exported** from your package to be used by package users
    - `export()`, `S3method()`, …
  - **Imported** from another package to be used internally by your package
    - `import()`, `importFrom()`, …
- `document()` updates the `NAMESPACE` file with directives from Roxygen comments

Documentation workflow



Workflow

Code + documentation + check



check() again

```
check()


#> ══ Documenting ══════════════════════════════════════════════════
…
#> ══ Building ═══════════════════════════════════════════════════
…
#> ══ Checking ═══════════════════════════════════════════════════
…
#> ── R CMD check results ─────────────────────────────────────
#> Duration: 3.1s
#>
#> 0 errors ✔ | 0 warnings ✔ | 0 notes ✔
```

`install()`

Install package to your library

- `R CMD INSTALL`

  Ctrl+Shift+B (Windows & Linux)
  Cmd+Shift+B (macOS)

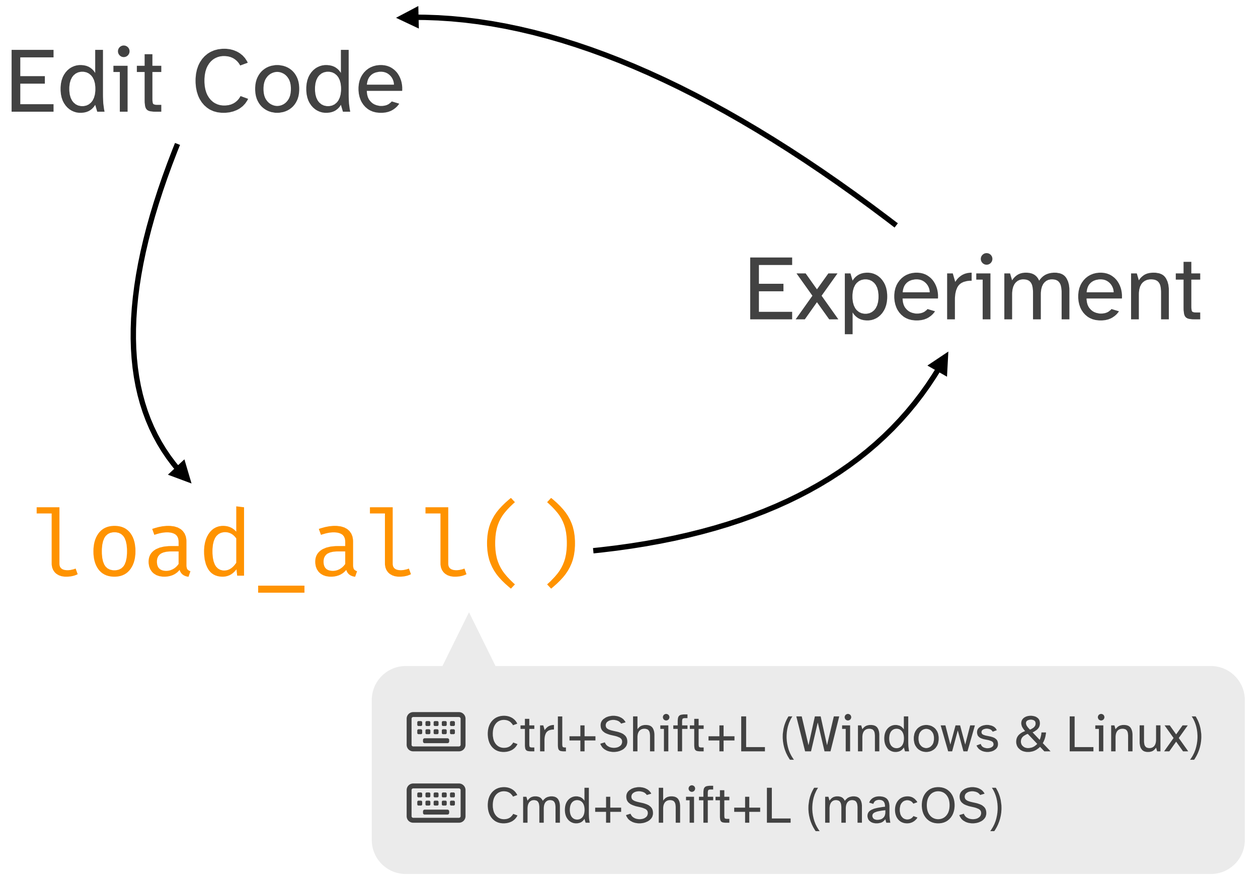
- Restart R

  Ctrl+Shift+F10 (Windows & Linux)
  Cmd+Shift+0 (macOS)

- Load package with `library()` like any other package

Testing

Current workflow



use_testthat()

Set up formal testing of your package*

```
use_testthat()

#> ✔ Adding 'testthat' to Suggests field in DESCRIPTION
#> ✔ Adding '3' to Config/testthat/edition
#> ✔ Creating 'tests/testthat/'
#> ✔ Writing 'tests/testthat.R'
#> • Call `use_test()` to initialize a basic test file and
open it for editing.
```

*Sorry, you still have to write the tests

use_test()

```
use_test('my-fun.R')*

#> ✔ Writing 'tests/testthat/test-my-fun.R'
#> • Edit 'tests/testthat/test-my-fun.R'
```

*Omit file name when `'R/my-fun.R'` is active file

Edit test

- Example test: Delete and replace with your own

```
testthat("description of what you're testing", {
  expect_equal([function output], [expected output])
})
```

test()

- Runs all tests in your test suite

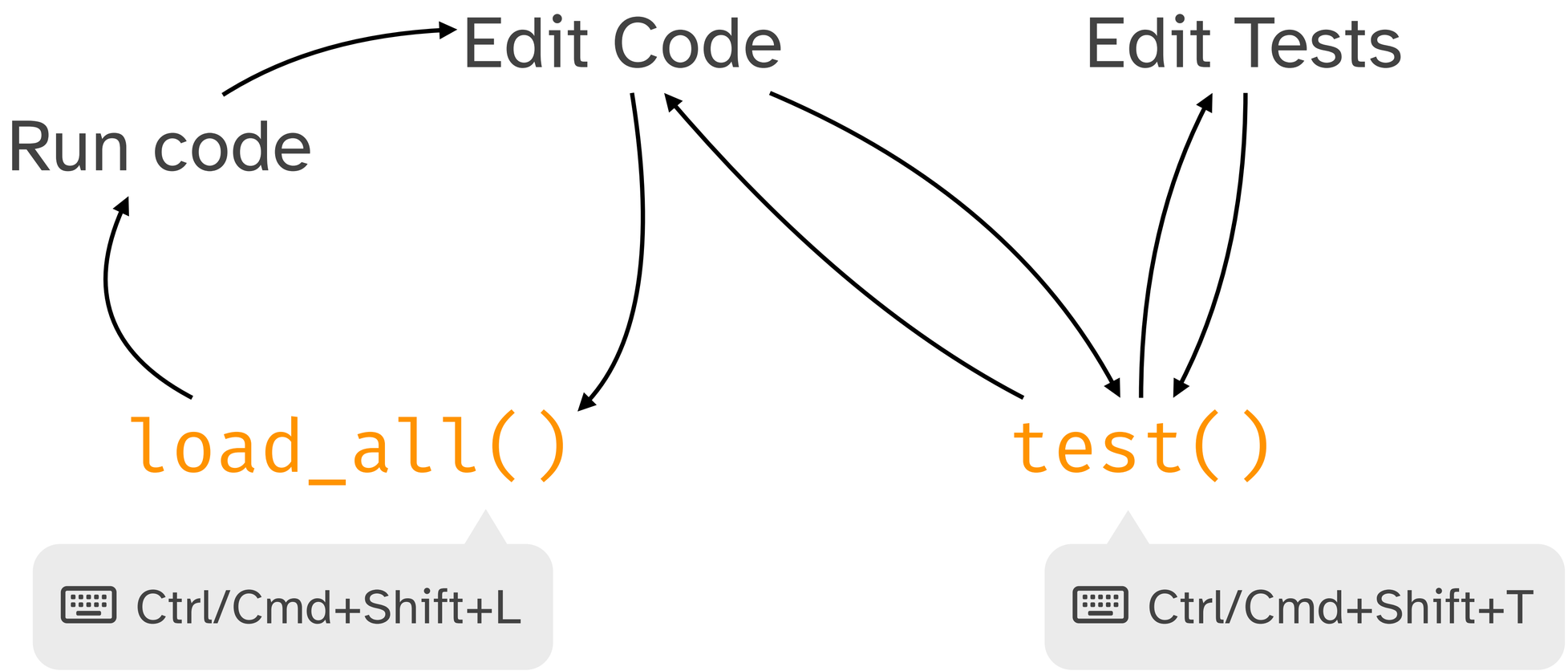
```
test()

#> ℹ Testing
#> ✔ | F W S  OK | Context
#>
#> ⠏ |         0 |
#> ✔ |         1 |
#>
#> ══ Results ═══════════════════════════════════════════════
#> [ FAIL 0 | WARN 0 | SKIP 0 | PASS 1 ]
```

Ctrl+Shift+T (Windows & Linux)
Cmd+Shift+T (macOS)

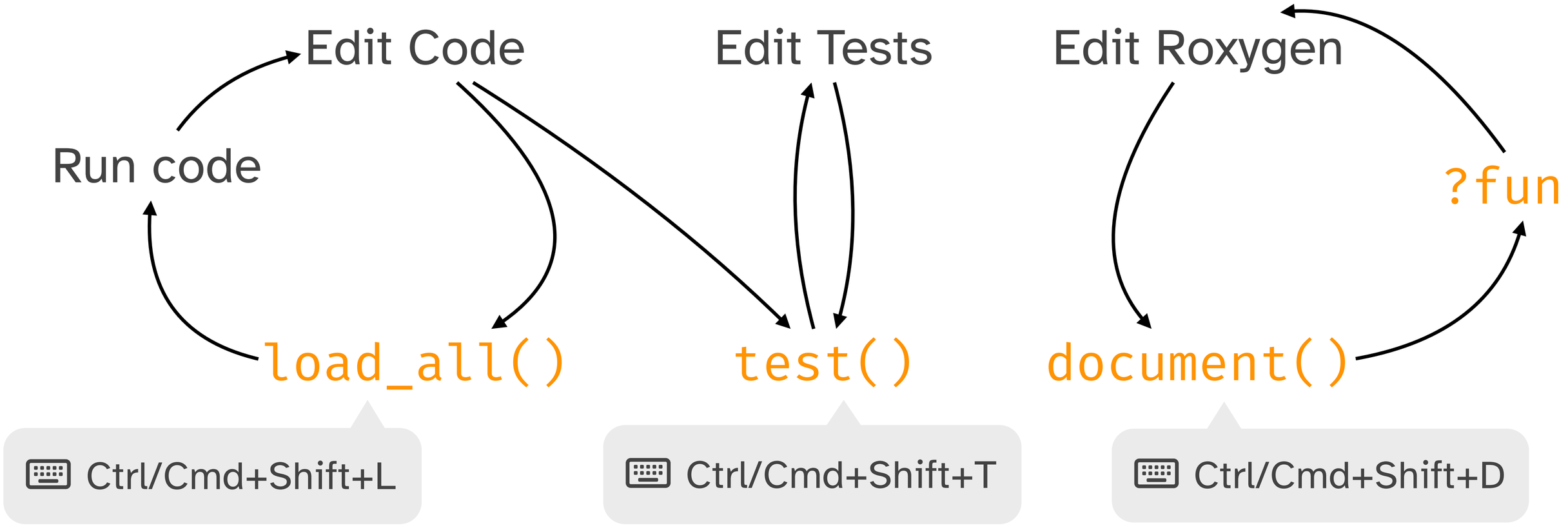
Updated workflow

Code + testing



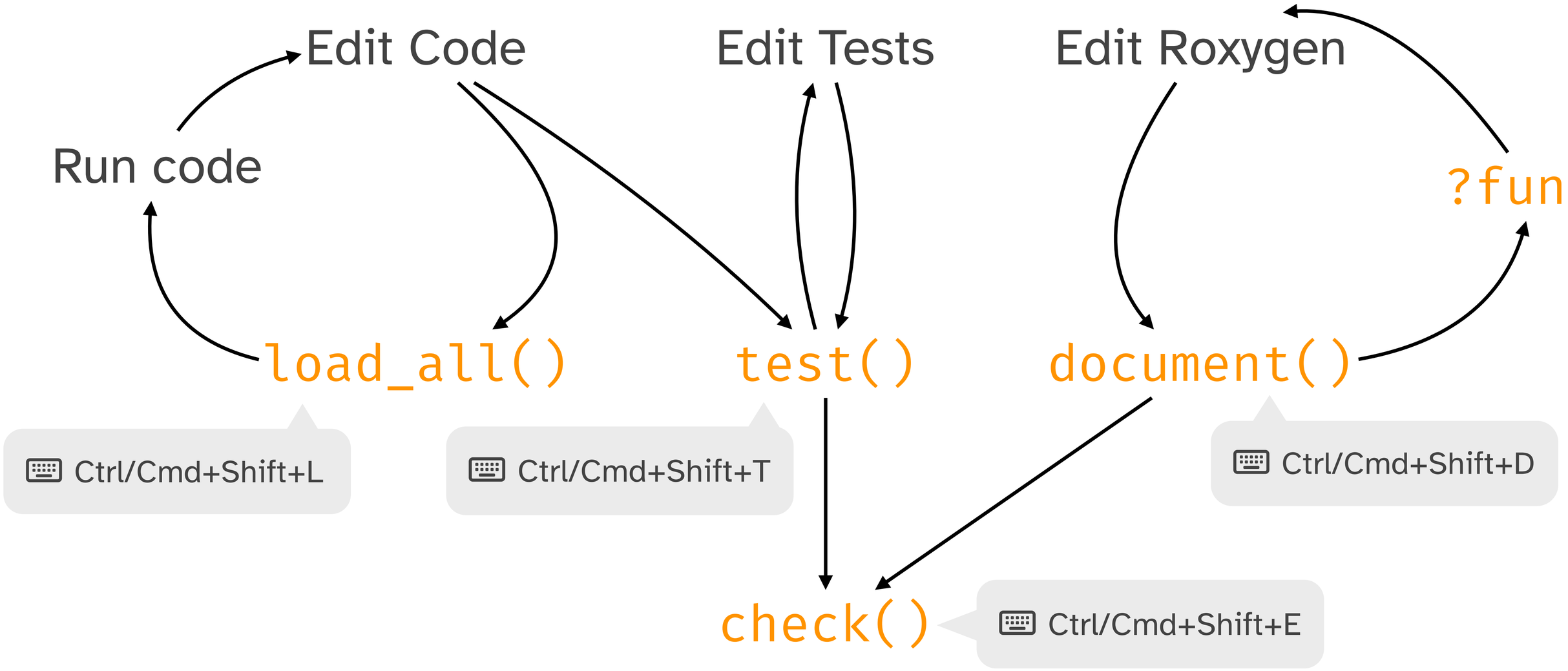
Workflow

Code + testing + documentation



Workflow

Code + testing + documentation + check



Commit your changes
Push to Github

use_package()

Add a dependency

```
use_package("fs")
#> ✔ Adding 'fs' to Imports field in DESCRIPTION
#> • Refer to functions with `fs::fun()`
```

- Use functions from another package inside your package
- Dependencies must be declared
  - Even from included packages (`stats::sd()`, `tools::file_ext()` etc.)
- Never call `library(pkg)` in code below `R/`!

Listing dependencies in `DESCRIPTION`

Three options

- `Depends:`
  - Ensures the package is installed with your package
  - *Attaches the package when yours is attached*
  - Rarely needed or recommended
- `Imports:`
  - Ensures the package is installed with your package
  - Most common location for dependencies
- `Suggests:`
  - Does not ensure installation automatically
  - Packages required for development (running tests, building vignettes, etc).
  - Rarely used functionality (especially if the dependency is difficult to install)

Imports: `DESCRIPTION` vs `NAMESPACE`

- Listing a package in `Imports` in `DESCRIPTION` does not “import” that package.
- A package listed in `Imports` in `DESCRIPTION` may, but does not have to, appear in `NAMESPACE`
- Every package listed in `NAMESPACE` must have an entry in `Imports` (or `Depends`) in `DESCRIPTION`

3 ways to use functions from another package

1. package::fun()
2. Import just the functions you want to use via @importFrom roxygen tag:

   ```
   #' @importFrom pkg fun1 fun2
   ```

   Adds to NAMESPACE:

   ```
   importFrom(pkg, fun1)
   importFrom(pkg, fun2)
   ```

3. Import the entire package with @import:

   ```
   #' @import pkg
   ```

   Adds to NAMESPACE:

   ```
   import(pkg)
   ```

Use your new dependency

Write a function using a function from the dependent package

- use_r("new-fun.R")

OR

- rename_files("my-fun.R", "new-fun.R")
  - Keeps R/ and test file names in sync.
- Write/edit function using dependency: pkg::fn()
- Add (or update) tests
- document()
  - Writes man/*.Rd files & regenerates NAMESPACE

`use_readme_rmd()`

Generates `README.md`, your package's home page on GitHub

```
use_readme_rmd()

#> ✔ Writing 'README.Rmd'
#> ✔ Adding '^README\\.Rmd$' to '.Rbuildignore'
#> • Update 'README.Rmd' to include installation instructions.
#> ✔ Writing '.git/hooks/pre-commit'
```

- The purpose of the package
- Installation instructions
- Example usage

Final `check()` and `install()`

You did it!

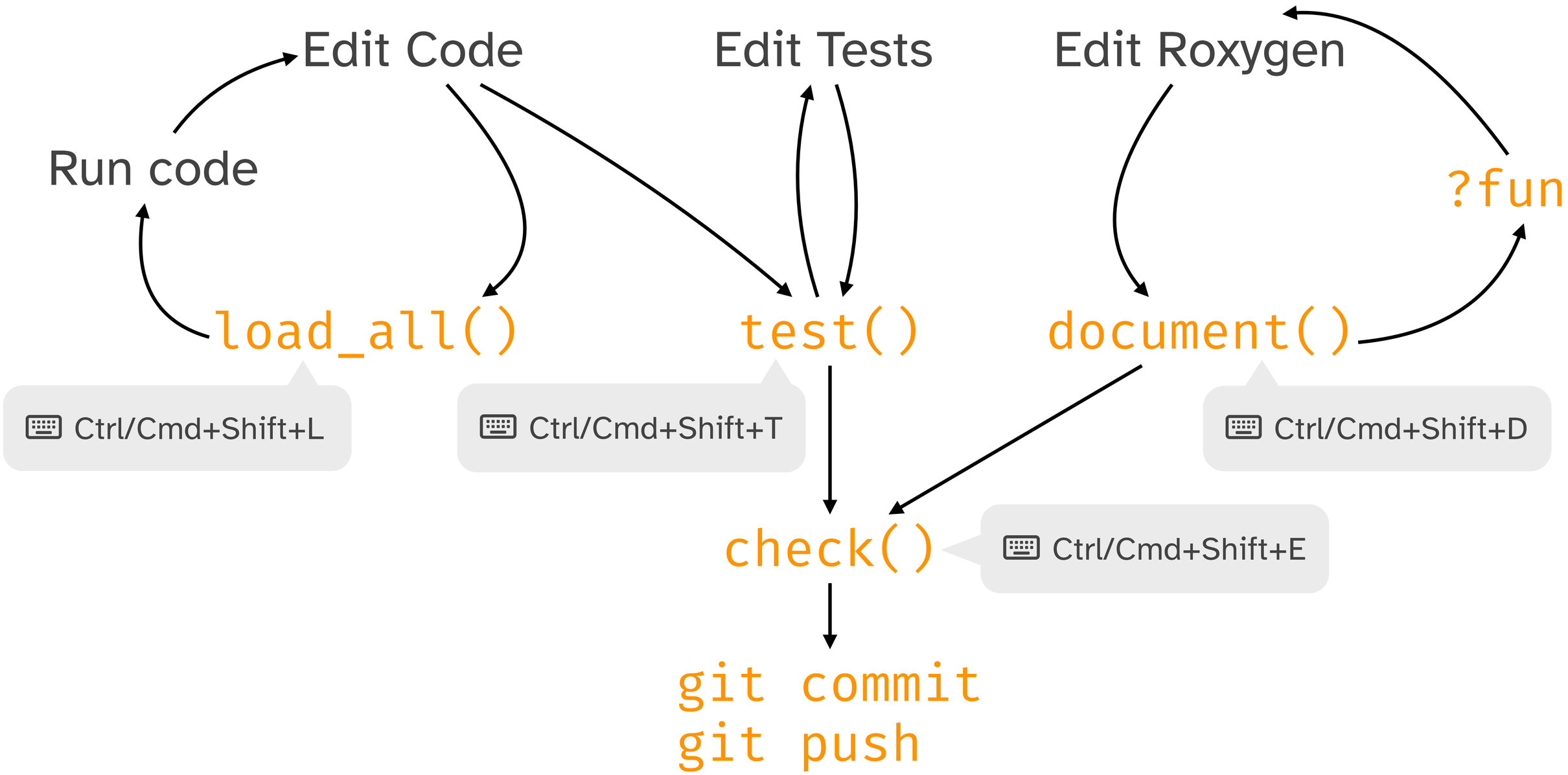
```
check()

#> ── R CMD check results ─────────
#> Duration: 3.1s
#>
#> 0 errors ✔ | 0 warnings ✔ | 0 notes
```

```
install()

#> ── R CMD build ─────────────────────
#> checking for file '/Users/jane/rrr/mypackage/DESCRIPTION' ...  ✔
#> preparing ‘mypackage’:
#> checking DESCRIPTION meta-information ...  ✔
#> checking for LF line-endings in source and make files and shell
scripts
#> checking for empty or unneeded directories
#> building ‘mypackage_0.0.0.9000.tar.gz’
#> Running /usr/local/bin/R CMD INSTALL \
#>   /tmp/RtmpK6WnOX/mypackage_0.0.0.9000.tar.gz --install-tests
#> * installing to library '/Users/jane/Library/R/arm64/4.3/library'
#> * installing *source* package 'mypackage' ...
#> ** using staged installation
#> ** help
#> *** installing help indices
#> ** building package indices
#> ** testing if installed package can be loaded from temporary
location
#> ** testing if installed package can be loaded from final location
#> ** testing if installed package keeps a record of temporary
installation path
#> * DONE (mypackage)
```

Review: Workflow



Commit your changes
Push to Github

Review: functions

Run once

- `create_package()`
- `use_git()`
- `use_github()`
- `use_mit_license()`
- `use_testthat()`
- `use_readme_rmd()`

Run periodically

- `use_r()`
- `use_test()`
- `use_package()`
- `rename_files()`

Run frequently

- `load_all()`
  Ctrl/Cmd+Shift+L
- `document()`
  Ctrl/Cmd+Shift+D
- `test()`
  Ctrl/Cmd+Shift+T
- `check()`
  Ctrl/Cmd+Shift+E