Everything You Wanted to Know About Contributing to an R Package

But Were Too Afraid to Ask

Daniel D. Sjoberg

April 4, 2022

Memorial Sloan Kettering Cancer Center

@statistishdan

@ddsjoberg

Outline

1. Anatomy of an R Package
2. Preparing to Contribute
3. Fork + Clone + Branch
4. Write/Modify a Function
5. Document function/changes
6. Style your contribution
7. Add unit tests
8. R CMD Checks
9. Submit Pull Request

*We will cover how to contribute to an R package whose source code lives on **GitHub***

Other Resources

Anatomy of an R Package

```
## C:/Users/laveryj/Desktop/contributing-to-an-R-pkg/toy.pkg
## +-- DESCRIPTION
## +-- man
## |   \-- foo.Rd
## +-- NAMESPACE
## +-- R
## |   \-- foo.R
## \-- tests
##     \-- testthat
##         \-- test-foo.R
```

DESCRIPTION

- Every package must have a `DESCRIPTION`.
- The job of the `DESCRIPTION` file is to store important metadata about your package.
  - list dependencies
  - package title/description
  - package version
  - specify the package license
  - list package authors and contributors
- This file is setup by the package maintainer, and you will likely *NOT* need to modfiy this file.
- Read more here https://r-pkgs.org/description.html

```
Package: mypackage
Title: What The Package Does (one line)
Version: 0.1
Authors@R: person(
  "First", "Last",
  email = "first.last@example.com",
  role = c("aut", "cre"))
Description: What the package does
  (one paragraph)
Depends: R (>= 3.5)
Imports: dplyr
License: What license is it under?
LazyData: true
```

NAMESPACE

- You can see that the `NAMESPACE` file looks a bit like R code. Each line contains a directive: `S3method()`, `export()`, `importFrom()`, and so on.
- Each directive describes an R object, and says whether it's exported from this package to be used by others, or it's imported from another package to be used locally.
- {roxygen2} will generate `NAMESPACE` for you! Do *not* edit this file.

```
# Generated by roxygen2: do not edit by hand
S3method(add_n,tbl_summary)
export(tbl_regression)
export(tbl_summary)
export(tbl_uvregression)
importFrom(glue,glue)
importFrom(knitr,knit_print)
importFrom(magrittr,"%>%")
```

What are you going to contribute?

Please do not make a blind pull request into a package.

1. Before you begin working on a new function or feature, **submit an Issue on GitHub**.

2. **Work with the package maintainer** to ensure the new functionality fits with the existing package.

3. If a **GitHub Issue already exists** for the feature request, join the conversation on the Issue: let the maintainer know it's something you'd like to work on.

Getting Ready

Before you can contribute to a package, you need to **get your system setup** for development.

1. If it's been a while, take the time to **update R and RStudio**.
2. **Install and configure git** (if you haven't already)
3. **Configure your GitHub Personal Access Token (PAT)**. Remember that Enterprise GitHub and public GitHub require two different PATs. Details here https://happygitwithr.com/https-pat.html
4. **Install** the following packages

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

5. Using Windows? **Install RTools**. https://cran.r-project.org/bin/windows/Rtools/
6. Using macOS? **Install Xcode**.

```
xcode-select --install
```

Fork + Clone + Branch

.xlarge[

- Most R Package source code is kept on GitHub
- Navigate to the package's GitHub page
    - Fork the repository
    - Clone the forked repository from your personal GH page to your local machine
    - Create a *new* branch to work on where you will add your function
    - Review the training materials for details on forking + cloning + branching

Install Package Dependencies

- Each package has dependencies: packages that another package relies upon.
- Before you can contribute to a package, you must install that package *and* all it's dependencies.
- There are different kinds of dependencies, and the most common are `Imports` (packages required to install the package) and `Suggests` (packages required to build the package, including packages used in documentation, vignettes, and unit testing).
- Before you can contribute to a package, you need to install *all* of its dependencies.
- Open the package R project in RStudio, and run `renv::install()`; this will install all the dependencies.

Write a Function

1. If you're adding a new function to a package, the code for the function will generally live in its own `.R` file by the same name. Run `usethis::use_r("foo")` and a blank script file will be added to the R folder.

```
usethis::use_r("foo")
#> ✓ Setting active project to 'C:/Users/SjobergD/GitHub/contributing-to-an-R-pkg'
#> ✓ Creating 'R/'
#> * Modify 'R/foo.R'
#> * Call `use_test()` to create a matching test file
```

2. Write your function!

```
foo <- function(x) {
# check input is numeric
stopifnot(is.numeric(x))

# return the mean
mean(x)
}
```

3. When writing function that utilize {tidyverse} functions, be sure to use the `.data` and `.env` prefixes. Review these slides for details https://mskcc-epi-bio.github.io/Writing-Function-with-the-tidyverse/

Document Your Function

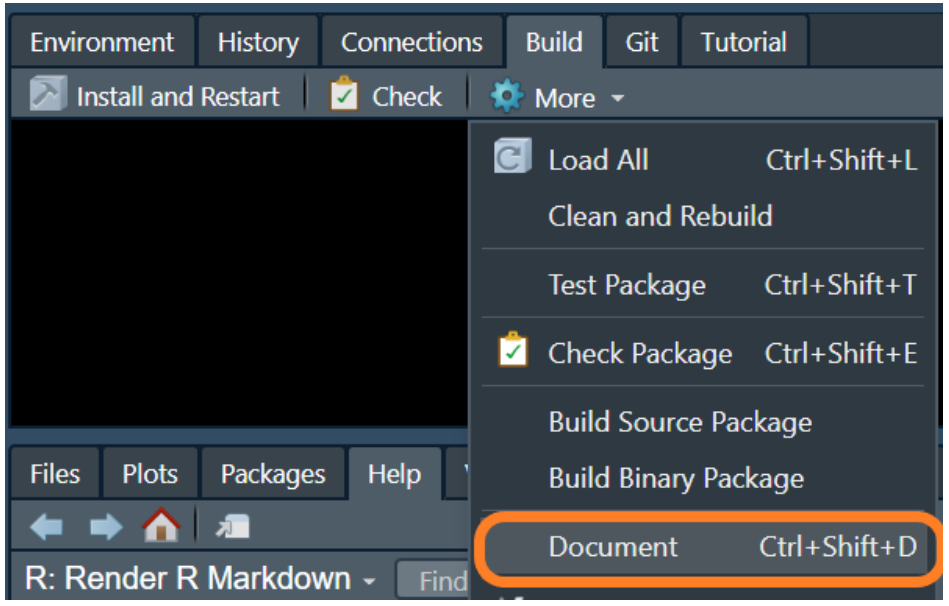
- After you've written your function, you need to document each of the arguments, include examples, and any other pertinent information.
    - Use the {roxygen2} comments package to document your function.
    - The {roxygen2} comments begin with `#'` and use tags like `@param`, `@export`, and `@examples` to generate the help file code.

```
#' Title
#'
#' @param x
#'
#' @return
#' @export
#'
#' @examples
foo <- function(x) {
  # check input is numeric
  stopifnot(is.numeric(x))

  # return the mean
  mean(x)
}
```

Document Your Function

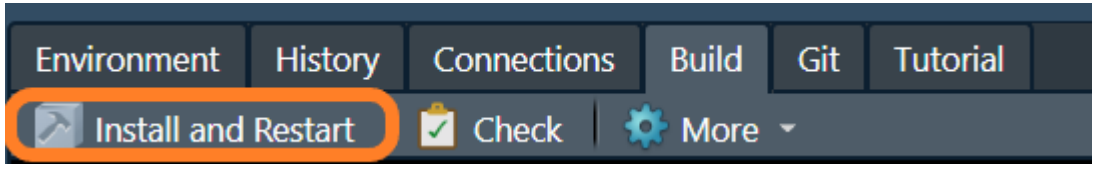
```
#' Calculate the Mean
#'
#' @param x a numeric vector
#'
#' @return numeric scaler
#' @export
#'
#' @examples
#' foo(mtcars$mpg)
foo <- function(x) {
  # check input is numeric
  stopifnot(is.numeric(x))

  # return the mean
  mean(x)
}
```

- Each time you update the roxygen comments, you must re-document the package (Ctrl+Shift+D in RStudio)



Building Pkg with Your Function

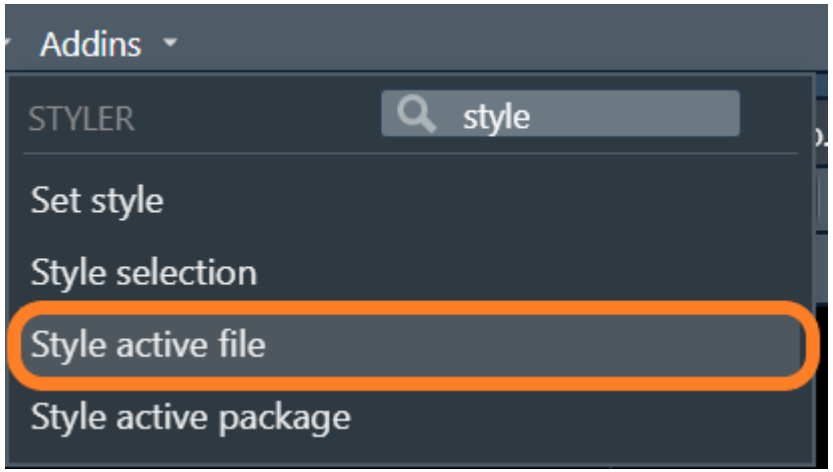
- After you've written a function (or half of a function) and documented it, you'll want to do some ad-hoc testing.
- Build + Install the package by clicking the "Build and Restart" button in the "Build" tab.



- You can also load the package (including exported and non-exported objects) with `devtools::load_all()` or Ctrl+Shift+L in RStudio

Make it CUTE!

- Most open-source projects follow a style guidelines.
- Even if you're not familiar with the styles in the guide, you can easily conform your code using the {styler} package.



Add Unit Tests

- Testing is a vital part of package development.
- It ensures that your code does what you want it to do.
- Testing adds an additional step to your development workflow.
- We will use the {testthat} package for all our unit testing.

Add Unit Tests

- Run `usethis::use_test("foo")` to add a unit testing file

```
usethis::use_test("foo")
#> ✓ Creating 'tests/testthat/'
#> ✓ Writing 'tests/testthat.R'
#> ✓ Writing 'tests/testthat/test-foo.R'
#> * Modify 'tests/testthat/test-foo.R'
```

- The {testthat} package has MANY functions for testing your function. Here are the ones I use most often:
  - `expect_error()` can test for the presence (or lack) of an error
  - `expect_message()` tests whether the function prints a message (can also test the text of the message)
  - `expect_equal()` test your function's return result equals a specified value
  - `expect_true()` checks expression evaluates to `TRUE`
- Each test should have an informative name and cover a single unit of functionality. The idea is that when a test fails, you'll know what's wrong and where in your code to look for the problem.

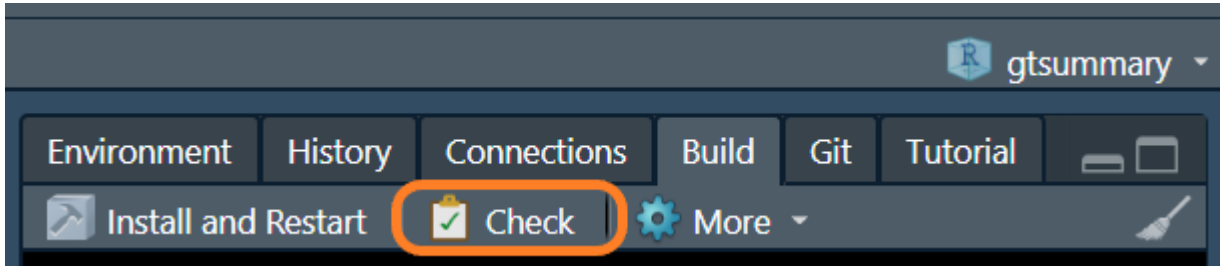
Code Coverage

After writing your unit tests, calculate the coverage for your new function.

```
withr::with_envvar(new = c("NOT_CRAN" = "true"), covr::report())
```

Aim for 100% coverage of any addition you've made, and add unit tests as needed until you're there!

R CMD Checks



What is checked? Here's a very abbreviated list

- package structure
  - hidden files/folders
  - portable file names
  - executable files
  - package subdirectories
  - left-over files
- DESCRIPTION/NAMESPACE file
  - package dependencies
  - files exist
  - NAMESPACE parses properly
- R code
  - non-ASCII characters
  - syntax errors
  - dependencies in R code
  - S3 generic/method consistency
- documentation
  - Rd/help files
  - Rd file metadata
  - examples
  - undocumented function arguments

Checks

Review results and check there are zero errors, warnings, and notes.

```
-- R CMD check results --------------------------- starter 0.1.8.9000 ----
Duration: 50.8s

0 errors v | 0 warnings v | 0 notes v

R CMD check succeeded
```

A common note is about **undefined global variables**. This most often occurs when using {dplyr} verbs without the `.data` prefix.

```
# bad syntax
mtcars |> mutate(mpg10 = mpg * 10)
```

```
# good syntax
mtcars |> mutate(mpg10 = .data$mpg * 10)
```

```
> checking R code for possible problems ... NOTE
  writing_files_folders: no visible binding for global variable
    'file_abbrv'
  Undefined global functions or variables:
    file_abbrv

0 errors v | 0 warnings v | 1 note x
```

Spell Check

One last check...the spell check!

```
usethis::use_spell_check()
```

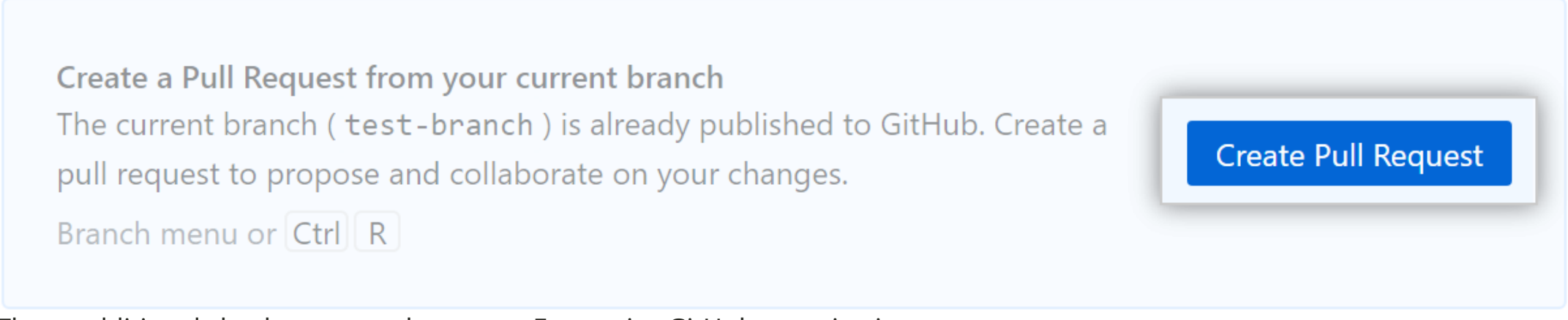
Submit Pull Request

You're almost there! Time to submit your change for acceptance into the main package repository.

When you submit a Pull Request

- The R CMD Checks will be run on Linux, Windows, and macOS
- The R CMD Checks will be run on multiple versions of R

  Click the Create Pull Request button in GitHub Desktop



These additional checks may not be run on Enterprise GitHub repositories.

Pull Request Checklist

Most package repositories for the Epi/Biostat Dept have a pull request checklist. Be sure to review each item of the checklist before asking the package maintainer to review your pull request.

Example from {gtsummary}

- [ ] Ensure all package dependencies are installed by running `renv::install()`
- [ ] PR branch has pulled the most recent updates from master branch. Ensure the pull request branch and your local version match and both have the latest updates from the master branch.
- [ ] If an update was made to `tbl_summary()`, was the same change implemented for `tbl_svysummary()`?
- [ ] If a new function was added, function included in `_pkgdown.yml`
- [ ] If a bug was fixed, a unit test was added for the bug check
- [ ] Run `pkgdown::build_site()`. Check the R console for errors, and review the rendered website.
- [ ] Code coverage is suitable for any new functions/features. Review coverage with `withr::with_envvar(new = c("NOT_CRAN" = "true"), covr::report())`. Begin in a fresh R session without any packages loaded.
- [ ] R CMD Check runs without errors, warnings, and notes
- [ ] `usethis::use_spell_check()` runs with no spelling errors in documentation

Let's Practice

We'll now walk through an example by submitting a pull request to the {tidycmprsk} package. https://github.com/MSKCC-Epi-Bio/tidycmprsk

Any questions before we begin?