Setting up an R development environment on github

RG Thomas

2024-12-24

Table of contents

0.1 Introduction

Its often the case that a data scientist needs to share an R function with a co-worker or a student. This post describes a step by step methodology for wrapping the function in a package and sharing it either via github or CRAN.

Figure 1: purrr

0.2 The Problem/Data

The end goal is to create a directory (or repository) that contains the package contents. The top level elements of

the package are the `DESCRIPTION` file, the `NAMESPACE` file, the `R` directory, the `tests` directory, and the `man` directory. Other files such as the `README.md`, the `LICENSE` file, and the `.gitignore` file are optional but recommended. The `DESCRIPTION` file contains metadata about the package such as the package name, the version number, the author, and the license. The `NAMESPACE` file contains the export and import declarations. The `R` directory contains the R functions. The `tests` directory contains the unit tests.

0.2.1 Step 1: Initial Repo Setup

Start by using the various helpful tools in the `devtools` and `usethis` packages to facilitate the repository building process.

Open `R` from the shell prompt in your development directory. and run the command `usethis::create_package("my_package")` to create the package directory and the `DESCRIPTION` and `NAMESPACE` files. Assuming the package will be named `my_package`.

```
install.packages("devtools")
install.packages("usethis")
library(usethis)
library(devtools)
usethis::create_package("my_package")
```

This creates the following directory structure.

```
my_package   tree --charset=ascii
.
|-- DESCRIPTION
|-- NAMESPACE
`-- R

my_package   more DESCRIPTION
Package: my_package
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: 7.3.2
```

```
usethis::use_git()
usethis::use_github()
use_gpl_license(version = 3, include_future = TRUE)
usethis::use_readme_md()
usethis::use_code_of_conduct("rgthomas@ucsd.edu")
usethis::use_tidy_contributing()
```

Next copy the R file containing the function to the `R` directory and add a `#'` roxygen comment block to the top of the file. Then call `devtools::document()` to generate the `man` directory containing the help page.

```
devtools::document()
```

At this point the directory structure looks like this.

```
julia dev/my_package   tree --charset=ascii
.
|-- DESCRIPTION   # package Metadata
|-- NAMESPACE     # Exports and imports declarations
|-- R             # R functions
|   `-- my_package.R
|-- man           # Documentation for the functions
    `-- my_package.Rd
```

The next step is to set up testing.

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

```
call inside R
usethis::use_test("my_package")
```

This open an editor. Enter the unit tests using the test_that function.

```
# Test: Empty dataframe error
test_that("t2f throws an error for empty dataframe", {
  empty_df <- data.frame()
  expect_error(my_package(empty_df, filename = "empty_table"), "`df` must not be empty")
})
```

Set up a new repository on github.

```
git init
git add .
git commit -m "Initial commit"
```

Add each dependency (e.g. `kableExtra`) (e.g. kableExtra) `(e.g. kableExtra)`

```
usethis::use_package("kableExtra", type = "Imports")
```

Finally do a full check using `devtools::check()`. This reflects the checks that CRAN will perform when you submit the package.

```
devtools::build()
devtools::install()
devtools::test()
devtools::check()
```

1 Debugging workflow

```
git checkout -b fix-bug
```

```
Debug locally and isolate the issue.
```

- ```
  Create a local branch (fix-bug) for the fix.
  ```
- git checkout -b fix-bug
- ```
  Make and test the changes.
  ```
- Run devtools::test() to confirm all tests pass.
- Use devtools::check() to validate the package.
- git add .
- git commit -m “Fix issue with my_package function”
- git push
- Merge the branch into the main branch and clean up.
- git checkout main
- git merge fix-bug
- git branch -d fix-bug
- git push
- ```
  Open a Pull Request.
  ```
- ```
  Update the version number
  ```
- usethis::use_version(“patch”)
- and push the final changes.
- git push

2 Debugging Workflow from chatGPT

Follow these steps to debug and fix issues in your R package:

1. **Debug Locally**
   - Isolate the issue using R debugging tools like `browser()`, `traceback()`, or `debug()`.
2. **Create a Local Git Branch**
   - Create a branch for the fix to isolate your changes:

```
git checkout -b fix-bug
```

3. **Make and Test Changes**
   - Modify your code to fix the issue and add or update unit tests as needed.
   - Run tests to confirm functionality:

```
devtools::test()  # Confirm all tests pass
devtools::check() # Validate the package complies with CRAN standards
```

4. **Commit Your Changes**
   - Stage and commit your changes:

```
git add .
git commit -m "Fix issue with my_package function"
```

5. **Push the Branch**
   - Push the branch to GitHub for collaboration or to prepare for merging:

```
git push origin fix-bug
```

6. **Open a Pull Request**
   - Open a Pull Request (PR) on GitHub to merge the fix into the main branch. Include a clear description of the changes.
7. **Merge and Clean Up**

- After review and approval, merge the branch into the main branch:

```
git checkout main
git merge fix-bug
```

- Delete the branch locally and remotely:

```
git branch -d fix-bug
git push origin --delete fix-bug
```

8. **Test the Main Branch**
   - Ensure the main branch passes all tests:

```
devtools::test()  # Confirm functionality
devtools::check() # Validate compliance
```

9. **Update the Version Number**
   - Increment the package version using `usethis::use_version()`:

```
usethis::use_version("patch")  # Use "patch", "minor", or "major"
```

   - Commit and push the version update:

```
git add DESCRIPTION
git commit -m "Bump version to 1.0.1"
git push
```

2.1 Reproducibility

```
# Print session info for reproducibility
sessionInfo()
```

```
R version 4.4.2 (2024-10-31)
Platform: aarch64-apple-darwin20
Running under: macOS Sequoia 15.2

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRblas.0.dylib
```

```
LAPACK: /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRlapack.dylib;  LA

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

time zone: America/Los_Angeles
tzcode source: internal

attached base packages:
[1] stats     graphics  grDevices datasets  utils     methods   base

other attached packages:
 [1] here_1.0.1         shiny_1.9.1        styler_1.10.3      quarto_1.4.4       pacman_0.5.1
[11] dplyr_1.1.4        purrr_1.0.2        readr_2.1.5        tidyr_1.3.1        tibble_3.2.1
[21] janitor_2.2.0      datapasta_3.1.0    ggthemes_5.1.0     conflicted_1.2.0   DT_0.33

loaded via a namespace (and not attached):
 [1] tidyselect_1.2.1  viridisLite_0.4.2 R.utils_2.12.3    fastmap_1.2.0     promises_1.3.2
[11] processx_3.8.4    magrittr_2.0.3    compiler_4.4.2     rlang_1.1.4       tools_4.4.2
[21] pkgload_1.4.0     miniUI_0.1.1.1    R.cache_0.16.0     withr_3.0.2       R.oo_1.27.0
[31] colorspace_2.1-1  scales_1.3.0      cli_3.6.3          generics_0.1.3    remotes_2.5.0
[41] vctrs_0.6.5       jsonlite_1.8.9    hms_1.1.3          visdat_0.6.0      systemfonts_1.1.0
[51] munsell_0.5.1     pillar_1.9.0      htmltools_0.5.8.1 R6_2.5.1          rprojroot_2.0.4
[61] Rcpp_1.0.13-1     svglite_2.1.3     xfun_0.49          fs_1.6.5          pkgconfig_2.0.3
```

2.2 Next Steps

- Suggest areas for further exploration
- Mention potential improvements
- Invite reader engagement

2.3 References

- Cite your sources
- Link to relevant documentation
- Credit other contributors

Tags: R, your-topic-tags Category: R