Creating R Packages

W. Evan Johnson, Ph.D.
Professor, Division of Infectious Disease
Director, Center for Data Science
Rutgers University – New Jersey Medical School
w.evan.johnson@rutgers.edu

2026-02-09

Importance of developing software

"As a modern statistician, one of the most fundamental contributions you can make is to create and distribute software that implements the methods you develop. I have gone so far as to say if you write a paper without software, that paper doesn't exist." (Jeff Leek)

(Note: to give proper credit, I borrowed material shamelessly from Jeff Leek's R package tutorial on GitHub for these slides and code.)

Getting started

Before we get started, you will need to install the following R packages:

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

Creating your package

We will first create our package structure. Go to the directory where you want your package to be created. I am putting my package in the "~/github/" directory. Then use the `create` function to create a new R package structure:

```
setwd("~/github/")
devtools::create("mypackage")
```

Now go look at your new packge! Notice the `DESCRIPTION`, `LICENSE`, and `NAMESPACE` (more on this later) documents and `R` folder. The `R` folder is empty now, but this is were we will put our package functions.

Creating your package

You can automate your license selection with the **usethis** package in the tidyverse. We will assign an MIT open source license:

```
setwd("~/github/mypackage")
usethis::use_mit_license()
```

For more information on licenses, check out the following:
https://blog.codinghorror.com/pick-a-license-any-license/

Formatting functions

Also important before we begin: The **tidyverse** gives you some more advanced code formatting help to make your package functions look very clear! Check out the the **styler** package (check out the Add In for RStudio!). For example:

```
# Before
x=3; y<-10
if (x >3) {stop("this is an error")} else {
    c(there_are_fairly_long,
      1 / 33 *
          2 * long_long_variable_names)%>% k(

          ) }
```

Formatting functions

Go to **Addins→style active file**

```
# After
x <- 3
y <- 10
if (x > 3) {
  stop("this is an error")
} else {
  c(
    there_are_fairly_long,
    1 / 33 *
      2 * long_long_variable_names
  ) %>% k()
}
```

Adding functions

Okay! We are ready to start adding functions to our R package. In the `R/` directory, create a new file (RStudio→R Script) named **hello.R** and add the code:

```
hello <- function(){print("hello world!")}
```

Now, just this once, go to the `NAMESPACE` and add "export(hello)" to the file. This will make the `hello()` function available to users when they install the package. (Later, we will learn a better way!)

Installing packages

Now we can install the package using `devtools::install`. From the `mypackage` package directory, enter:

```
devtools::install()
```

Now check if it worked!

```
library(mypackage)
hello()
```

Installing packages

Now go and add two more functions to your package: **addition.R** and **subtraction.R**. And make sure to add them to the NAMESPACE (this is the last time you should ever do this in your career!)

Install the package and confirm that it works!

Documenting R packages

There are some AMAZING tools, such as **roxygen2**, that are available that make documenting your packages very easy! Add the following:

```
#' A one sentence description of what your function does
#'
#' A more detailed description of what the function is and how
#' it works. It may be a paragraph that should not be separated
#' by any spaces.
#'
#' @param inputParameter1 A description of the input parameter \code{inputParameter1}
#' @param inputParameter2 A description of the input parameter \code{inputParameter2}
#'
#' @return output A description of the object the function outputs
#'
#' @keywords keywords
#'
#' @export
#'
#' @examples
#' R code here showing how your function works

myfunction <- function(inputParameter1, inputParameter2){
    ## Awesome code!
    return(result)
}
```

Documenting R packages

Once you have added documentation to all of your functions, use **devtools::document** to compile your documented code.

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

Documenting R packages

Now check to see how this changes your **mypackage** directory. Then reinstall the package, and look at the help for your functions, for example:

```
?hello
?addition
?subtraction
```

More documentation: Vignettes

Documentation in the help files is important and is the primary way that people will figure out your functions if they get stuck. But it is equally (maybe more) critical that you help people get started. The way that you do that is to create a **vignette**. Vignettes can generate either HTML from R markdown, or pdf from latex.

Go check some of your favorite R packages. I bet many of them have vignettes!

Session Info

```
sessionInfo()
```

```
## R version 4.5.1 (2025-06-13)
## Platform: aarch64-apple-darwin20
## Running under: macOS Tahoe 26.2
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/lib/libRlapack.dylib;  LAPACK version 3.12.1
##
## 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/Denver
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## loaded via a namespace (and not attached):
##  [1] vctrs_0.6.5       cli_3.6.5         knitr_1.51        rlang_1.1.6
##  [5] xfun_0.55         stringi_1.8.7     otel_0.2.0        purrr_1.2.0
##  [9] pkgload_1.4.1     glue_1.8.0        rprojroot_2.1.1   htmltools_0.5.9
## [13] roxygen2_7.3.3    pkgbuild_1.4.8    rmarkdown_2.30    evaluate_1.0.5
## [17] tibble_3.3.0      ellipsis_0.3.2    fastmap_1.2.0     yaml_2.3.12
## [21] lifecycle_1.0.4   memoise_2.0.1     whisker_0.4.1     stringr_1.6.0
## [25] compiler_4.5.1    fs_1.6.6          sessioninfo_1.2.3 pkgconfig_2.0.3
```