**Writing an R package from scratch**

Posted on [**April 29, 2014**](https://hilaryparker.com/2014/04/29/writing-an-r-package-from-scratch/)

As I have worked on various projects at Etsy, I have accumulated a suite of functions that help me quickly produce tables and charts that I find useful. Because of the nature of iterative development, it often happens that I reuse the functions many times, mostly through the shameful method of copying the functions into the project directory. I have been a fan of the idea of [personal R packages](https://hilaryparker.com/2013/04/03/personal-r-packages/) for a while, but it always seemed like A Project That I Should Do Someday and someday never came. Until…

Etsy has an amazing week called “[hack week](https://twitter.com/Etsy/status/460840965411205120)” where we all get the opportunity to work on fun projects instead of our regular jobs. I sat down yesterday as part of Etsy’s hack week and decided “I am finally going to make that package I keep saying I am going to make.” It took me such little time that I was hit with that familiar feeling of the joy of optimization combined with the regret of past inefficiencies (joygret?). I wish I could go back in time and create the package the first moment I thought about it, and then use all the saved time to watch cat videos because that really would have been more productive.

This tutorial is not about making a beautiful, perfect R package. This tutorial is about creating a bare-minimum R package so that you don’t have to keep thinking to yourself, “I really should just make an R package with these functions so I don’t have to keep copy/pasting them like a goddamn luddite.” Seriously, it doesn’t have to be about sharing your code (although that is an added benefit!). It is about saving yourself time. (n.b. this is my attitude about all reproducibility.)

(For more details, I recommend [this chapter](http://adv-r.had.co.nz/Package-basics.html) in Hadley Wickham’s *Advanced R Programming*book.)

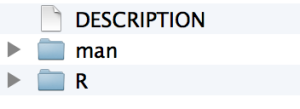
**Step 0: Packages you will need**  
The packages you will need to create a package are devtools and roxygen2. I am having you download the development version of the roxygen2 package.

|  |  |
| --- | --- |
| 1  2  3  4 | install.packages("devtools")  library("devtools")  devtools::install\_github("klutometis/roxygen")  library(roxygen2) |

**Step 1: Create your package directory**  
You are going to create a directory with the bare minimum folders of R packages. I am going to make a cat-themed package as an illustration.

|  |  |
| --- | --- |
| 1  2 | setwd("parent\_directory")  create("cats") |

If you look in your parent directory, you will now have a folder called cats, and in it you will have two folders and one file called DESCRIPTION.

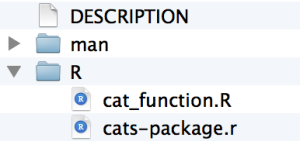
[](https://hilaryparker.files.wordpress.com/2014/04/screen-shot-2014-04-29-at-4-26-23-pm.png)

You should edit the DESCRIPTION file to include all of your contact information, etc.

**Step 2: Add functions**  
If you’re reading this, you probably have functions that you’ve been meaning to create a package for. Copy those into your R folder. If you don’t, may I suggest something along the lines of:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | cat\_function <- function(love=TRUE){      if(love==TRUE){          print("I love cats!")      }      else {          print("I am not a cool person.")      }  } |

Save this as a cat\_function.R to your R directory.

[](https://hilaryparker.files.wordpress.com/2014/04/screen-shot-2014-04-29-at-4-28-01-pm.png)

(cats-package.r is auto-generated when you create the package.)

**Step 3: Add documentation**  
This always seemed like the most intimidating step to me. I’m here to tell you — it’s super quick. The package roxygen2 that makes everything amazing and simple. The way it works is that you add special comments to the beginning of each function, that will later be compiled into the correct format for package documentation. The details can be found in the [roxygen2documentation](https://github.com/klutometis/roxygen#roxygen2) — I will just provide an example for our cat function.

The comments you need to add at the beginning of the cat function are, for example, as follows:

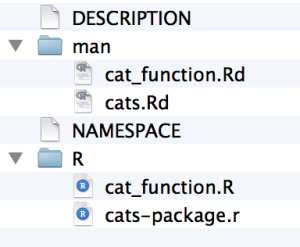
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | #' A Cat Function  #'  #' This function allows you to express your love of cats.  #' @param love Do you love cats? Defaults to TRUE.  #' @keywords cats  #' @export  #' @examples  #' cat\_function()    cat\_function <- function(love=TRUE){      if(love==TRUE){          print("I love cats!")      }      else {          print("I am not a cool person.")      }  } |

I’m personally a fan of creating a new file for each function, but if you’d rather you can simply create new functions sequentially in one file — just make sure to add the documentation comments before each function.

**Step 4: Process your documentation**  
Now you need to create the documentation from your annotations earlier. You’ve already done the “hard” work in Step 3. Step 4 is as easy doing this:

|  |  |
| --- | --- |
| 1  2 | setwd("./cats")  document() |

This automatically adds in the .Rd files to the man directory, and adds a NAMESPACE file to the main directory. You can [read up more](http://adv-r.had.co.nz/Package-basics.html) about these, but in terms of steps you need to take, you really don’t have to do anything further.

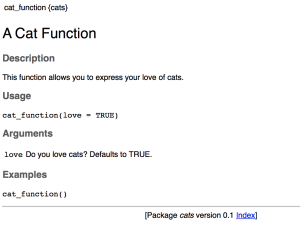
[](https://hilaryparker.files.wordpress.com/2014/04/screen-shot-2014-04-29-at-4-33-58-pm.png)

(Yes I know my icons are inconsistent. Yes I tried to fix that.)

**Step 5: Install!**  
Now it is as simple as installing the package! You need to run this from the parent working directory that contains the cats folder.

|  |  |
| --- | --- |
| 1  2 | setwd("..")  install("cats") |

Now you have a real, live, functioning R package. For example, try typing ?cat\_function. You should see the standard help page pop up!

[](https://hilaryparker.files.wordpress.com/2014/04/screen-shot-2014-04-29-at-5-04-55-pm.png)

**(Bonus) Step 6: Make the package a GitHub repo**  
This isn’t a post about learning to use git and GitHub — for that I recommend Karl Broman’s[Git/GitHub Guide](http://kbroman.github.io/github_tutorial/). The benefit, however, to putting your package onto GitHub is that you can use the devtools install\_github() function to install your new package directly from the GitHub page.

|  |  |
| --- | --- |
| 1 | install\_github('cats','github\_username') |

**Step 7-infinity: Iterate**  
This is where the benefit of having the package pulled together really helps. You can flesh out the documentation as you use and share the package. You can add new functions the moment you write them, rather than waiting to see if you’ll reuse them. You can divide up the functions into new packages. The possibilities are endless!

Additional pontifications: If I have learned anything from my (amazing and eye-opening) first year at Etsy, it’s that the best products are built in small steps, not by waiting for a perfect final product to be created. This concept is called the [minimum viable product](http://en.wikipedia.org/wiki/Minimum_viable_product) — it’s best to get a project started and improve it through iteration. R packages can seem like a big, intimidating feat, and they really shouldn’t be. The minimum viable R package is a package with just one function!

Additional side-notes: I learned basically all of these tricks at the [rOpenSci hackathon](http://ropensci.github.io/hackathon/). My academic sister [Alyssa](http://alyssafrazee.com/) wrote a [blog post](http://simplystatistics.org/2014/04/10/the-ropensci-hackathon-ropenhack/) describing how great it was. Hadley Wickham gets full credit for [envisioning that R packages should be the easiest way to share code](http://adv-r.had.co.nz/Philosophy.html), and making functions/resources that make it so easy to do so.