

Writing R Packages

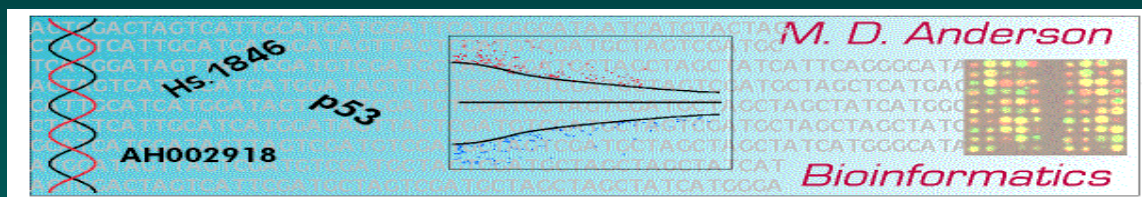
Keith A. Baggerly

Bioinformatics and Computational Biology

UT M. D. Anderson Cancer Center

kabagg@mdanderson.org

SISBID, July 17, 2017



This is Not a Lecture

well, ok, it is a bit.

Since we're hoping to teach you to use a tool, a big part of this will involve live demo, and letting you follow along.

Thus, these notes are at least as much for **reference documentation** as for presentation.

We'll also point out the very nice [Cheatsheet from RStudio](#)

R Packages are *Cool*

They let us easily share code, data, and documentation

They can be shared at common sites such as

CRAN and
Bioconductor

We're going to show you how you can
assemble a minimal R package in under
15 minutes.

Writing R Packages Can be Painful

Why was assembling packages hard?

They have specific structure and **documentation requirements**.

Most of the documentation (.Rd) files use a syntax based on \LaTeX , which can take practice in itself

Getting the interconnections right can be annoying and time-consuming, not to mention counterintuitive

The final reference, the manual on “**Writing R Extensions**” is not ideally welcoming to newcomers

It's Less Painful Today

Writing packages is less painful today primarily because of *other R packages* from **Hadley Wickham** and **Yihui Xie**.

These packages shift bookkeeping from frustrated mortals to laconic computers:

devtools

roxygen2

knitr

rmarkdown

These lower the barriers to entry!

I'm assuming these have been loaded going forward.

In the Beginning...

there was **creation**.

Let's shift to the folder will work in, and invoke

```
> devtools::create("toyPackage")
```

This creates a new folder (toyPackage) populated with some of the files every package needs.

What's There to Start With?

(using the latest versions of everything)

```
.gitignore  
.Rbuildignore  
DESCRIPTION  
NAMESPACE  
R/    (initially empty)  
toyPackage.Rproj
```

Not a heck of a lot, and fortunately
we'll leave most of these alone.

What we will edit (first) is the DESCRIPTION.

What do we need to DESCRIBE a package?

Who wrote it, and how to contact them

Who maintains it, and how to contact them

What other packages it builds on

What license it uses

This is outlined in the DESCRIPTION file Rstudio pre-assembled.

What needs changing

```
Package: toyPackage
Title: What the Package Does (one line,
      title case)
Version: 0.0.0.9000
Authors@R: person("First", "Last",
                  email="first.last@example.com",
                  role=c("aut", "cre"))
Description: What the package does (one
            paragraph) .
Depends: R (>= 3.4.0)
License: What license is it under?
Encoding: UTF-8
LazyData: true
```

What needs changing (changed)

Package: toyPackage

Title: Illustrates a Minimal Package

Version: 0.1.0

Authors@R: person("Keith", "Baggerly",
email="kabagger@mdanderson.org",
role=c("aut", "cre"))

Description: Trying to get something working
quickly and easily

Depends: R (>= 3.4.0)

License: CC0

Encoding: UTF-8

LazyData: true

Only the “License” above really needs expansion

Licenses, Tigers and Bears

Your work is your property, by default.

Specifying the license tells others what they have your permission to do with your work. Fuller descriptions are here:

[R licenses](#)

Per the Rstudio cheatsheet,

CC0 (Creative Commons) is extremely broad; other commonly used ones are

MIT and

GPL (please see cheatsheet).

When we Change Things...

```
document() ## updates NAMESPACE, creates man/,  
           ## edits DESCRIPTION  
  
build()  
install()  
check()  
.rs.restartR()  
library(toyPackage)
```

Guess what, we're there.

It's useless, but **we're there**.

Mimicking External Commands

The commands within R are actually testing out commands generally used on the command line, specifically

R CMD BUILD myPackage

R CMD INSTALL myPackage

R CMD CHECK myPackage

This last is typically the most stringent, and applies many specific tests CRAN uses before accepting packages for distribution.

If you've already built the package, you can **CHECK myPackage_version.tar.gz**.

Documenting a Package

The first thing we want to add is more text documenting what we want the package to do.

To do this, we're going to create a new R script in R/, "toyPackage.R", which will consist almost entirely of comments for roxygen2.

Almost every script needs to state

What it is (the **TITLE**)

What it's for (the **DESCRIPTION**)

Let's take a look at such a script...

Rd files

Now, invoking

```
document ()
```

generates an R documentation (.Rd) file in man/.

The syntax of Rd files is similar to that of \LaTeX , but **we're not going to write Rd files by hand.**

We'll let the packages do that so everything lines up correctly.

Cap'n, there be text here!

Once we cycle through build/install/check and
rs.restartR/library, invoking

```
?toyPackage
```

will now render the Rd file to show the documentation
(and a link to the package Index).

Similarly, we can add functions that do things...

Other Stuff to Add

What we've got is indeed a package.

That said, there are a few other basics you should (and know how to) include.

README.md

vignettes

data

Division of labor

```
use_readme_rmd()
```

```
use_vignettes()
```

```
use_data()
```

README.md

Adding a README file to any package is a good idea.

This should say what the package is designed to do, and what the motivation was in building it.

I write READMEs as .Rmd files, but **GitHub** will automatically render straight markdown README.md contents in html.

Fortunately, this usage has been anticipated...

README Templates

```
use_readme_rmd()
```

Creates a template README.Rmd file and automatically adds it to .Rbuildignore.

Once you've included the relevant content, simply invoking

```
knit("README.Rmd")
```

produces the README.md file desired.

Adding a vignette

We've documented the package at a high level, but **people will not use a package they can't figure out.**

We need to give details of how to apply the package to tasks they have.

These lengthier descriptions should be given as **vignettes**, which we can write as .Rmd reports.

These should describe, in both words and code, how to use the package to do something real.

Vignette setup

```
use_vignette("vignetteName")
```

will automatically add a template vignetteName.Rmd file to vignettes/ (and create the latter if it doesn't already exist).

After adding your content, calling

```
document()  
build_vignettes()  
build()
```

will build the vignette.

Adding Data

Every package should include the features discussed above.

Data is more tricky, but including some can be quite helpful, especially if the files you're analyzing have quirks you want to highlight.

Data can be included in either raw (e.g., .csv, .bam) or processed (.RData) forms, but **keep it small** ($< 1Mb$) if this is for wide sharing.

Process Raw Data

Raw data, in whatever form, should be put in

```
/inst/extdata
```

and one of the first uses of a vignette should be to show how to map the raw data to its final processed form.

Opinion: Don't include raw data without including the processed version as well.

We can find the data with `system.file`, e.g.

```
system.file("extdata", "myRawData.bam",  
            package="toyPackage")
```

Including Processed Data

Once you have object(s) in R,

```
use_data(myObject, pkg="toyPackage")
```

will create .rda file(s) and store it in /data

Each data object should be **documented** (in R/) as (e.g.)
myObject.R

That Looked Familiar...

As with **package** documentation, we use NULL to anchor the text for **data** documentation.

Much of the description of what a dataset is and how to use it should also be repeated in an associated vignette.

A Word of Caution

In general, packages shouldn't be huge with respect to processing requirements, memory requirements (e.g., 20Mb figures), or final page length (vignettes).

A similar point applies to the use examples supplied for individual functions.

Ideally, examples should run in a few seconds.

Keep packages small and nimble.

Walking Through Another Package

There's a good basic package from Alyssa Frazee here

[Alyssa Frazee's RSkittleBrewer](#)

and there's an associated [BLOG POST](#) with more details

DESCRIPTION, NAMESPACE, functions, documentation...

does all of this look familiar?

Some Thoughts on Coding

The code should work - sanity checks

The code should be readable - style, naming, formatting

The code should be reusable - generalizing beyond the immediate case at hand

The code should be reproducible - will you get the same results 5 minutes from now?

Think before you code - hard when people want it NOW.

Learn from others' code (leveraging GitHub!)

Don't Assume

Please avoid referring to other objects in your workspace. Pass what's needed as an argument.

Name your arguments and functions. Don't be cute. "tmp1, tmp2, x1, x2, x3"? Having a few words can clarify things.

Text autocompletion can be your friend at times!

Name your constants / avoid magic numbers

```
foo(12)  ## bad  
nMonths <- 12  
foo(nMonths)  ## good
```


Legibility

INDENT YOUR CODE.

There are **RStudio shortcuts** for many common tasks; for indentation it's CTRL-I or CMD-I depending on your platform

Use white space. This makes it easier for human beings, not the computer.

Avoid long lines (wrap them).

The **formatR** package can help here

General Habits

When you're writing functions, start including roxygen2 comments **right then**.

Everything packages need is pretty basic, and forcing yourself to write this out will help others (and you) a great deal.

For one thing, you'll be clearer about your initial motivations in writing it!

A similar heuristic applies to writing a README when you first write a package.

Encapsulating Analysis Projects as Packages

Even for those of us who work with big data, we still generate reports from some datasets which might be 20K in size.

In those cases,

include both the raw and processed data, and

include both your processing and analysis as vignettes.

Then, the entire analysis can be distributed as a package,
and you can know

if the package builds, your analysis can be done.

Packages Can Automate Other Tasks

Is there a common report structure you like to use?

You can include `template` reports and the like using

`inst/rmarkdown/templates/my_template_folder`

and including, in each such folder

`template.yaml`

`skeleton/skeleton.Rmd`

Start with examples from other packages

Invoking Automation

New RMarkdown Document - from template!

Positioning Folders

Anchoring everything - one purpose of the .Rproj file!

The rprojroot package helps with this

Things We've Glossed Over/Ignored

S3, S4, RC classes

Source functions from other languages (e.g., Rcpp) and compiled code

Use tests (a good idea! see the testthat package)

NAMESPACE explanations

Getting it ready for CRAN

Version Numbering

Most packages just use functions, text and data.

Where to find other examples

Aside from just poking around GitHub, here are a few:

Hadley Wickham's [book on R Packages](#) is on github, but you can also [buy it](#)

The documentation vignettes for [roxygen2](#)

[Hadley's GitHub Page](#)

Karl Broman's [Tools for RR course](#)

Onwards!