

WRITING R PACKAGES

PART II

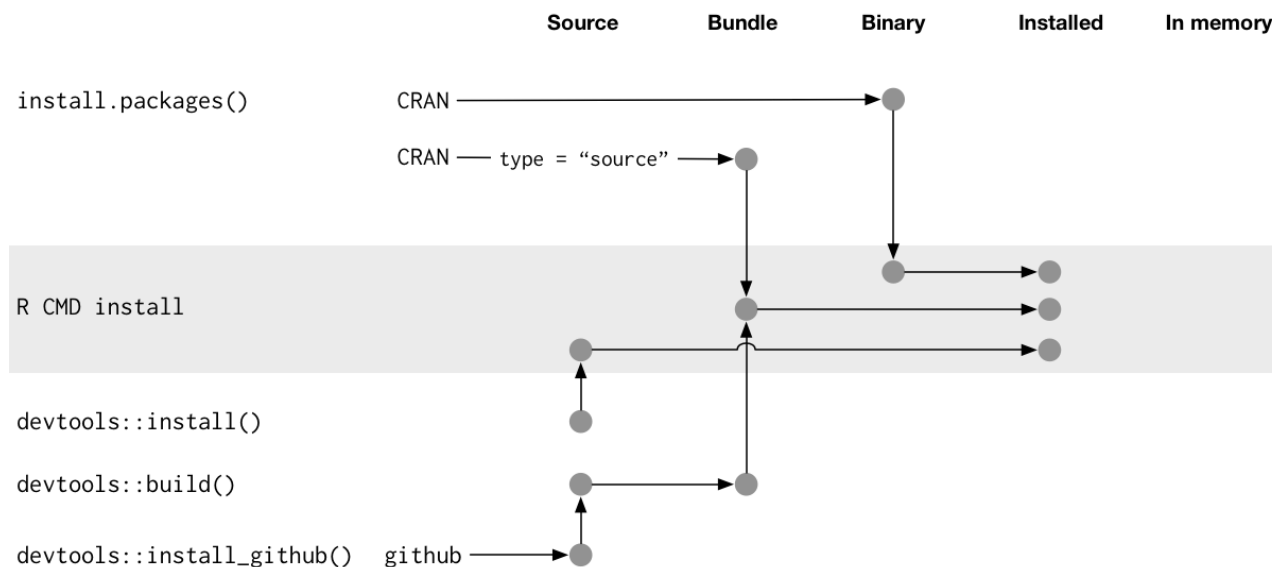
Jeff Goldsmith, PhD
Department of Biostatistics

What is a package?

- Collection of related functions, with helpful documentation, written to stand alone
- Can also include data, vignettes, non-R code (e.g. C++), tests, ...

What is a package?

- Package can exist as:
 - Source (what we've written)
 - Bundle (modified and zipped version of what we've written)
 - Binary (system-specific format used by CRAN)
- Which version you share depends on how it will be installed



Install vs attach

- “Installation” takes a source / bundle / binary and embeds in it your package library, making it available to R
 - The path to your package library can be found using `.libPaths()`
 - Various functions can be used for installation
- “Attaching” takes an installed package from your library and places it in R’s **search path** so that you can find the functions it includes
 - The somewhat-unfortunately-named `library()` function attaches packages
- This distinction is why we “Install and Restart” when building packages to include edits changes in the package attached by `library()`

Search path

- When you attach a package using `library()` you add it to the **search path**
- The search path is where R looks for functions
 - First stop: global environment
 - Next up: attached packages
- If a function doesn't exist in the global environment or attached packages, you can't run it directly
- You can avoid attaching a package by using `package::function()`, because this tells R specifically where to look
 - This is preferred in cases where there might be **name conflicts**
- Search paths are important for using R in general; they matter even more when you're developing a package

NAMESPACE

- Your package's search path is controlled by the NAMESPACE file in your documentation
 - Packages imported by the NAMESPACE are in your package's search path
 - Code in your package can access functions in imported packages directly, just as you would in usual code after attaching a package
- This file is edited by roxygen, and reflects the packages you import using @import
- This also reflects the functions you export using @export; that is, which functions are visible to users when your package is attached

Imports and Exports

- Why not import everything?
 - Packages often have functions with the same name
 - Importing everything increases the chance of a **name conflict**, which makes function calls ambiguous
 - This is also why you should only attach packages you need, and use `package::function()` in your code and packages
- Why not export everything?
 - To help prevent name conflicts!

Checks

- There's a lot to keep track of when writing a package
- You will make mistakes
 - In code
 - In documentation
 - In namespace
 - In examples
- `check()` automatically runs a wide range of checks on your package
- When it finds issues (and it will), fix them!
 - Improves code consistency
 - Updates documentation
 - Keeps package self-contained

Tests

- Checks are the same for every package
 - These focus on common issues
- Tests are package-specific
 - These make sure that the functions produce expected output
- Writing tests formalizes the ad hoc code you try on your package as you develop it
 - The testthat package facilitates this process
 - Basically, you write code with expected results and test that the results match your expectations



Vignettes

- Function-level documentation is good for quick references
 - Check argument names and allowed values
 - Understand output
 - Find easy examples
- This is helpful if users already mostly know what's going on
- You need longer documentation to give a package overview
 - What problem you solve
 - How functions work together
 - Non-trivial examples
- Vignettes are this long-form documentation
 - ... written in R Markdown!
 - You don't even have to learn anything new ...

Other topics

- There's a lot of stuff we won't touch on, but matters for development
 - Including data
 - Documenting data
 - Including compiled code (e.g. C / C++)
 - Continuous integration / automated checking
 - Deployment on CRAN
 - Backward compatibility