Making a package from base R files

John C. Nash

11/06/2021

Background

This article tries to explain an approach to developing alternative versions of functions which are in the distributed base of R. Our interest was in developing improvements to the nls() function and related features in R as part of a Google Summer of Code project for which Arkajyoti Bhattacharjee is the funded student. However, nls() has many tentacles involving a number of files and functions that may or may not be called as `nls()` is executed.

Part of the difficulty in carrying out such development of alternative versions is that one needs to be able to execute the new variants in parallel with the existing ones. A heavy-effort approach would be to have separate full sets of R code and build each system and run them separately.

Attempts to provide functions to switch between sets of functions were not promising, and not pursued.

Duncan Murdoch suggested, and generously provide an almost-complete prototype of, a package of interlocked functions that provide the set of `nls()` capabilities. This package `nlspkg` has been cloned to `nlsalt` which is being modified to provide the new capabilities or new expression of the code. Moreover, these capabilities, such as a function `Anyfn()` can be tested side by side by use of the double colon syntax, namely,

```
nlspkg::Anyfn() # existing functionality or code
# and
nlsalt::Anyfn() # New functionality of code

# Options


# Packaging from base R files

Duncan Murdoch described his process of developing the prototype of
`nlspkg`:
```

Identify the main R functions I wanted in the package, and copy that code into ./R.

Identify the C/Fortran functions that are referenced, and copy the source for that into ./src.

Try to build, and get lots of errors about missing functions. Find those and copy as above.

There was one case (the `%||%` operator) where I thought the code used an unexported object from utils; the base-internals.R function just used ::: to get it, but a better solution would be to copy the source (which is pretty trivial), and it turns out, is what stats did (the duplicate source is in AIC.R).

Some files contained functions or methods that weren't relevant to us (e.g. confint.lm). I commented those out so I didn't need to worry about the functions they used, and imported the generic (i.e. confint) from stats.

In the C code, we were lucky because the nls code only made use of one C function that's not in the API: R_NewEnv. (That was called from numeric_deriv.) There's an R function to do that, so I called it and passed the result in.

Finally, once it would build, I tried running R CMD check. This identified all the missing help pages, which I copied over. They didn't need any modifications.

Because nls isn't really base functionality, all of this was a lot easier than if I had tried to move something that works more with the internals. That's probably true for most functions in stats, but would not be true for other base packages (except datasets). "'

Developing an alternative set of functions and alternative package