Creating R Packages

A very short introduction

Manuel Koller, 2 February 2022

Why Create Your Own R Packages?

Share to increase efficiency

- Share commonly used code between projects (of yours)
- Improve reproducibility
- Make sure code keeps working
- Add documentation to help yourself
- Control what gets used when

What is in an R Package?

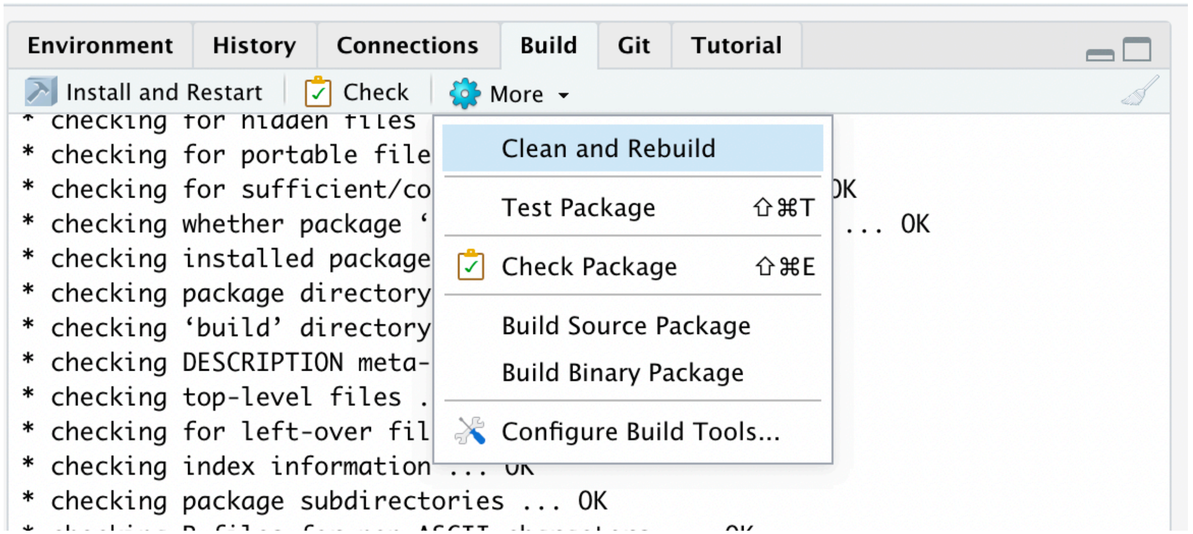
More than just R functions

- R functions
- Datasets
- Documentation: help, examples and vignettes
- Tests
- Namespace information (import & export)
- To be compiled code (C, C++, Fortran)
- Author information

Convert R Scripts to R Package

Let R do it for you

- The old / default way: `package.skeleton` gets you started and automatically converts functions and creates help files.
- Or: Use the `devtools` R package (Cheatsheet: https://devtools.r-lib.org)
- R Studio has lots of tools to help



Documentation

Keep code & help close

- Pure R package structures keeps R code in .R files and documentation in .Rd files.
- Keeping things in sync is a hassle to do “by hand”.
- Roxygen2 lets you add documentation in your R files and generates lots automatically.
- Also deals with imports and exports.

```
401  ##' Use \code{compare} to quickly compare the estaimated parameters of
402  ##' the fits of multiple lmerMod or rlmerMod objects.
403  ##'
404  ##' @title Create comparison charts for multiple fits
405  ##' @param ... objects to compare, or, for the \code{\link{xtable}}
406  ##' functions: passed to the respective \code{\link{xtable}} function.
407  ##' @param digits number of digits to show in output
408  ##' @param dnames names of objects given as arguments (optional)
409  ##' @param show.rho.functions whether to show rho functions in output.
410  ##' @keywords models utilities
411  ##' @examples
412  ##' \dontrun{
413  ##'   fm1 <- lmer(Yield ~ (1|Batch), Dyestuff)
414  ##'   fm2 <- rlmer(Yield ~ (1|Batch), Dyestuff)
415  ##'   compare(fm1, fm2)
416  ##'   require(xtable)
417  ##'   xtable(compare(fm1, fm2))
418  ##'   str(getInfo(fm1))
419  ##' }
420  ##' @export
421  compare <- function(..., digits = 3, dnames = NULL,
422                      show.rho.functions = TRUE) {
423      linfos <- list(...)
424      if (!missing(dnames) && !is.null(dnames)) names(linfos) <- dnames
425      linfos <- lapply(linfos, getInfo)
```

Share with Others

CRAN is not the only way

- `R CMD build` creates an .tar.gz archive you can install elsewhere
- `R CMD INSTALL --build <your-package>.tar.gz` if you need to compile code for someone on Windows (produces a .zip archive that everybody can install).
- Put your package on github, etc, and install via `devtools::install_github("kollerma/robustlmm")`
- But CRAN is definitely the easiest way to share your package

Publish your Package on CRAN

The CRAN police will be watching you

- If you follow the rules, it's easy. Make sure you know them: https://cran.r-project.org/web/packages/policies.html
- Check your package:
- Run `R CMD check --as-cran <your-package>.tar.gz`
- with several versions of R (previous, current and development)
- Submit via web-form: https://xmpalantir.wu.ac.at/cransubmit/

Tests

What isn't tested will break

- Write examples for every function you export / expose to the user.
- All examples are run when running `R CMD check`
- Add test scripts to tests/ subdirectory.
- Simplest way: accompany .R files in tests/ with .Rout files. (Hint: Copy them over from previous `R CMD check` runs.) Any changes will be highlighted.
- Write unit tests via `testthat` R package (https://testthat.r-lib.org).

R Documentation

The R Packages “Bible”

- Writing R Extensions
  https://cran.r-project.org/doc/manuals/r-release/R-exts.html

- Whatever your problem, this usually has the solution.