Share!

April 2019

Hadley Wickham

@hadleywickham

Survey of important topics

1. Git + GitHub
2. License
3. Dependencies
4. Namespace: imports
5. Namespace: exports
6. R CMD check
7. Travis
8. CRAN

Git + GitHub

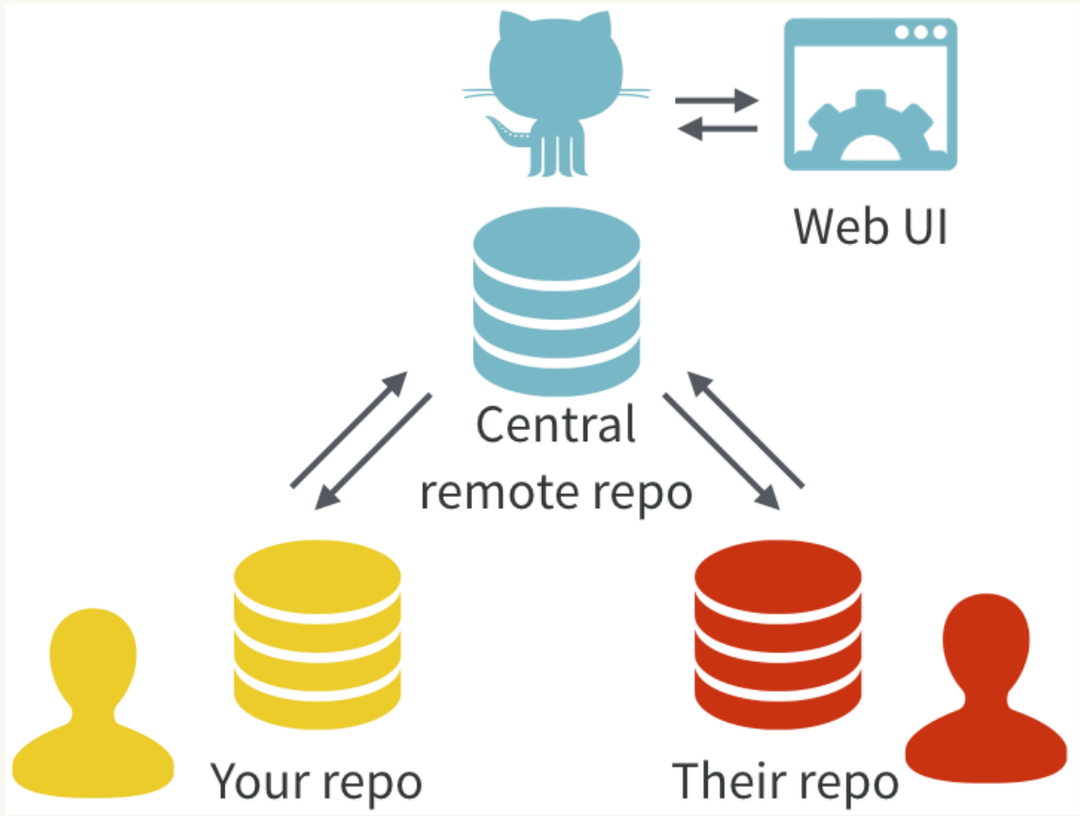
Use both!

The 😡💩😱 will pay off

happygitwithr.com

Excuse me, do you have a moment to talk about version control?

https://doi.org/10.7287/peerj.preprints.3159v2



Easy to get started

```
# Set up git
usethis::use_git()

# Publish with github
usethis::use_github()

# Tools for working with PRs
usethis::pr_init()
usethis::pr_fetch()
usethis::pr_push()
```

License

There are three main open source licenses

CC0

"public domain", best for data packages

MIT

Free for anyone to do anything with

GPL

Changes and bundles must also be GPL

These are gross simplifications!

Use helper to set up

```
usethis::use_cc0_license()
usethis::use_mit_license()
usethis::use_gpl_license()
```

You can also make clear that your package isn't open source

```
DESCRIPTION:
License: file LICENSE

LICENSE:
Proprietary: do not distribute outside of
Widgets Incorporated.
```

Dependencies

~~`library(xyz)`~~

~~`require(xyz)`~~

Needed by user

```
optional version spec
Imports:
    stringr (>= 1.0.0),
    lubridate
Suggests:
```

Needed by developer

There are three types of dependency

Imports = required. Installed automatically.

Suggests = optional. Development only; used in vignette or example. Not installed automatically.

Depends = basically deprecated for packages.(Correct uses exist, but beyond the scope of this class)

Use :: to access functions in imported packages

```
# In DESCRIPTION
Imports: foo
```

```
# In bar.R
new_function <- function(x, y, z) {
  foo::bar(x, y) + z
}
```

Should check if suggested package available

```
# In DESCRIPTION
Suggests: foo

# In bar.R
new_function <- function(x, y, z) {
  if (!requireNamespace("foo", quietly = TRUE)) {
    stop("Need foo! Use install.packages('foo').")
  }
  foo::bar(x, y) + z
}
```

Of course, usethis provides helpers

```
# use_package() will modify the DESCRIPTION
# and remind you how to use the function.
usethis::use_package("ggplot2")
usethis::use_package("ggplot2", "suggests")
```

Namespace: imports

You might get tired of using :: all the time

```
# Or you might want to use an infix function
`%>%` <- magittr::`%>%`

col_summary <- function(df, fun) {
  stopifnot(is.data.frame(df))
  df %>%
    purrr::keep(is.numeric) %>%
    purrr::modify(fun)
}
```

You can **import** functions into the package

```
#' @importFrom purrr keep modify
#' @importFrom magrittr %>%
col_summary <- function(df, fun) {
  stopifnot(is.data.frame(df))

  df %>%
    keep(is.numeric) %>%
    modify(fun)
}
```

Alternatively, create R/imports.R

```
# Imports belong to the package, not to
# individual functions, so you might want
# to recognise this by storing in a central
# location

#' @importFrom purrr keep map
#' @importFrom magrittr %>%
NULL
```

Importing everything from a package seems easy

```r
#' @import purrr
col_summary <- function(df, fun) {
  stopifnot(is.data.frame(df))

  df %>%
    keep(is.numeric) %>%
    map_dfc(fun)
}
```

But is dangerous...

```
#' @import foo
#' @import bar
fun <- function(x) {
  fun1(x) + fun2(x)
}

# Works today
# But next year, bar package adds fun1 function
# Recent ggplot2 release broke ~150 packages
```

Imports

Description	NAMESPACE
Makes package available	Makes function available
Mandatory	Optional (can use :: instead)
use_package()	#' @importFrom

Namespace: exports

A namespace splits functions into two classes

Internal	External
Only for use within package	For use by others
Documentation optional	Must be documented
Easily changed	Changing will break other people’s code

The default NAMESPACE exports everything

```
# Generated by roxygen2: fake comment so
# roxygen2 overwrites silently.
exportPattern("^[^\\.]")
```

Better to export function explicitly

```r
#' @export
fun1 <- function(...) {}

#' @export
fun2 <- function(...) {}
```

Most important if you're planning on sharing with others

Export functions that people should use

```r
# Don't export internal helpers


# Defaults for NULL values
`%||%` <- function(a, b) if (is.null(a)) b else a


# Remove NULLs from a list
compact <- function(x) {
  x[!vapply(x, is.null, logical(1))]
}
```

R CMD check

Automated checking

Runs automated checks for common problems in R packages.

Useful for local packages, even with some false positives.

If you want to submit to CRAN, you must pass R CMD check cleanly.

`http://r-pkgs.had.co.nz/check.html`


To avoid frustration run
early and run often

```
==> R CMD build rv2

* checking for file ‘rv2/DESCRIPTION’ ... OK
* preparing ‘rv2’:
* checking DESCRIPTION meta-information ... OK
* installing the package to build vignettes
* creating vignettes ... OK
* checking for LF line-endings in source and make files
* checking for empty or unneeded directories
* building ‘rv2_0.1.tar.gz’

==> R CMD check rv2_0.1.tar.gz

* using log directory ‘/Users/hadley/Documents/courses/13-devtools/rv2.Rcheck’
* using R version 3.0.2 (2013-09-25)
* using platform: x86_64-apple-darwin10.8.0 (64-bit)
* using session charset: UTF-8
* checking for file ‘rv2/DESCRIPTION’ ... OK
* this is package ‘rv2’ version ‘0.1’
* checking package namespace information ... OK
* checking package dependencies ... OK
* checking if this is a source package ... OK
* checking if there is a namespace ... OK
* checking for executable files ... OK
* checking for hidden files and directories ... OK
* checking for portable file names ... OK
```

```
* checking for sufficient/correct file permissions ... OK
* checking whether package ‘rv2’ can be installed ... OK* checking installed package size ... OK
* checking package directory ... OK
* checking DESCRIPTION meta-information ... OK
* checking top-level files ... OK
* checking for left-over files ... OK
* checking index information ... OK
* checking package subdirectories ... OK
* checking R files for non-ASCII characters ... OK
* checking R files for syntax errors ... OK
* checking whether the package can be loaded ... OK
* checking whether the package can be loaded with stated dependencies ... OK
* checking whether the package can be unloaded cleanly ... OK
* checking whether the namespace can be loaded with stated dependencies ... OK
* checking whether the namespace can be unloaded cleanly ... OK
* checking loading without being on the library search path ... OK
* checking dependencies in R code ... OK
* checking S3 generic/method consistency ... OK
* checking replacement functions ... OK
* checking foreign function calls ... OK
* checking R code for possible problems ... OK
* checking Rd files ... OK
* checking Rd metadata ... OK
* checking Rd cross-references ... OK
* checking for missing documentation entries ... OK
* checking for code/documentation mismatches ... OK
```

```
* checking Rd \usage sections ... OK
* checking Rd contents ... OK
* checking for unstated dependencies in examples ... OK
* checking installed files from ‘inst/doc’ ... OK
* checking files in ‘vignettes’ ... OK
* checking examples ... OK
* checking for unstated dependencies in tests ... OK
* checking tests ...
  Running ‘testthat.R’ OK
* checking for unstated dependencies in vignettes ...
 OK
* checking package vignettes in ‘inst/doc’ ... OK
* checking running R code from vignettes ...
   ‘clt.Rmd’ ... OK
 OK
* checking re-building of vignette outputs ... OK
* checking PDF version of manual ... OK

R CMD check succeeded
```

Types of problem

ERROR

Must fix!

WARNING

Fix if submitting to CRAN

NOTE

Fix if submitting to CRAN

It is possible to submit with a NOTE, but it's best avoided

	Local	CRAN
ERROR	✓	✓
WARNING		✓
NOTE		✓

```
# Cmd/Ctrl + Shift + E
devtools::check()

# If you don't understand an error,
# google it!
```

Travis

What is travis?

Travis is a continuous integration service = automatically runs code every time you push your changes to GitHub.

Really important for package developers.

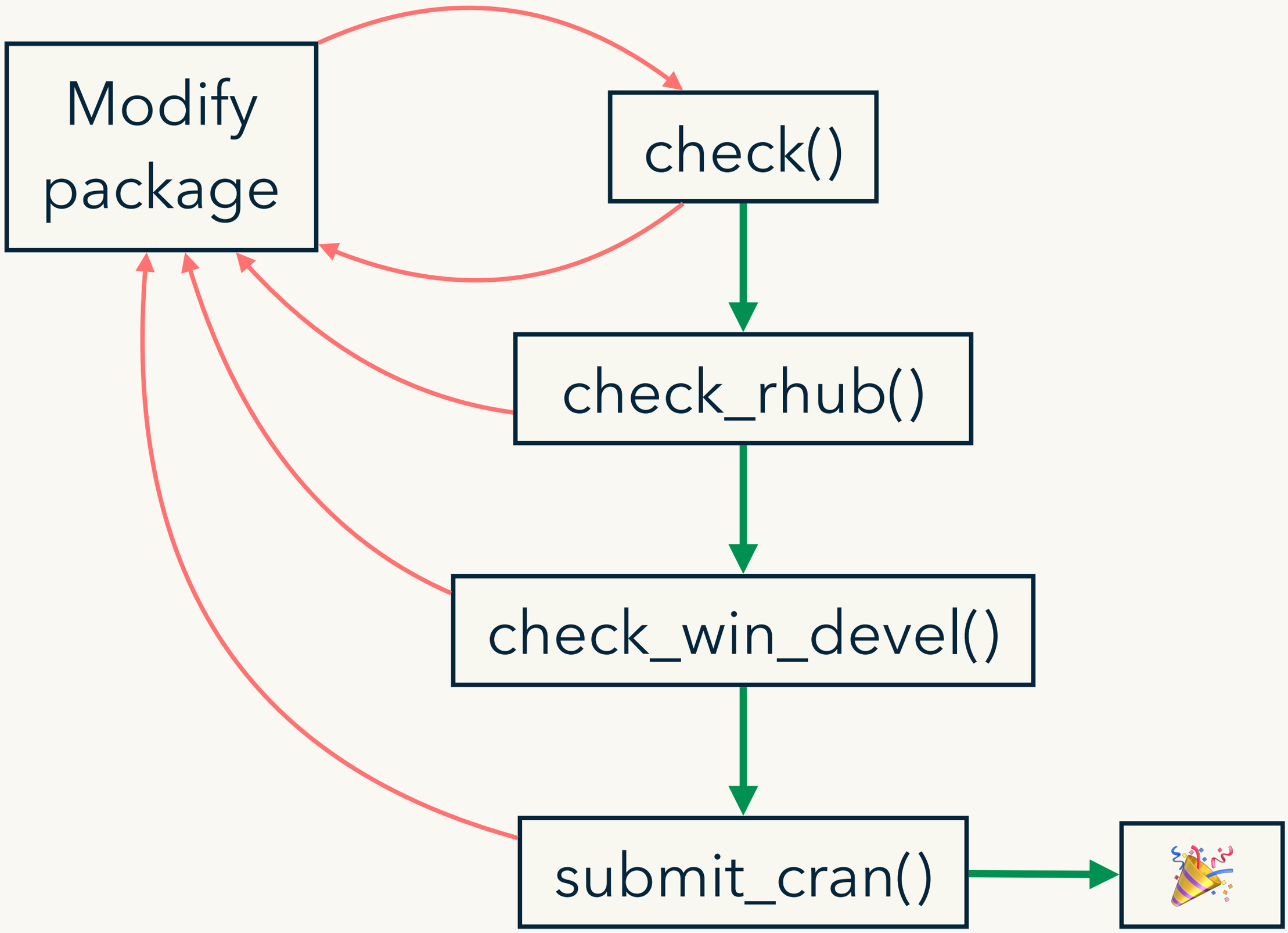
If you use git and GitHub

```
# Run R CMD check every time your code changes
usethis::use_travis()

# Automatically compute test coverage
usethis::use_codecov()

# Automatically rebuild package website
usethis::use_pkgdown_travis()
```

CRAN

First submission to CRAN

```
usethis::use_release_issue()

# Particularly important for larger packages
# where release process might take weeks.
# But also useful for smaller packages,
# and you should feel free to tweak for your
# needs
```


Modify package
check()
check_rhub()
check_win_devel()
submit_cran()

cran-comments.md

Goal is to document your process

```
## Test environments
* local OS X install (R-release)
* win-builder (R-release, R-devel)

## R CMD check results

0 errors | 0 warnings | 1 note

* This is a new release.
```

There's always one note for a new submission

If your submission fails

Do not despair! It happens to everyone, even R-core members.

If it's from the CRAN robot, just fix the problem & resubmit.

If it's from a human, **do not respond** to the email and **do not argue**. Instead update cran-comments.md & resubmit.

For resubmission:

This is a resubmission. Compared to the last submission, I have:

* First change.
* Second change.
* Third change

Test environments
* local OS X install, R 3.2.2
* win-builder (devel and release)

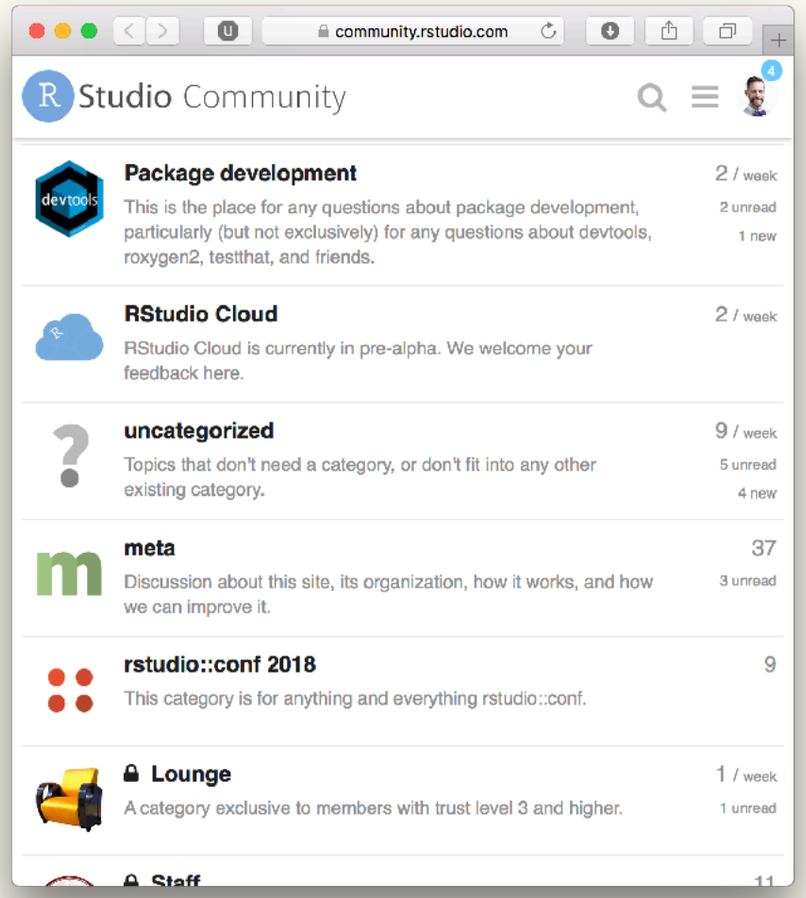
R CMD check results

...

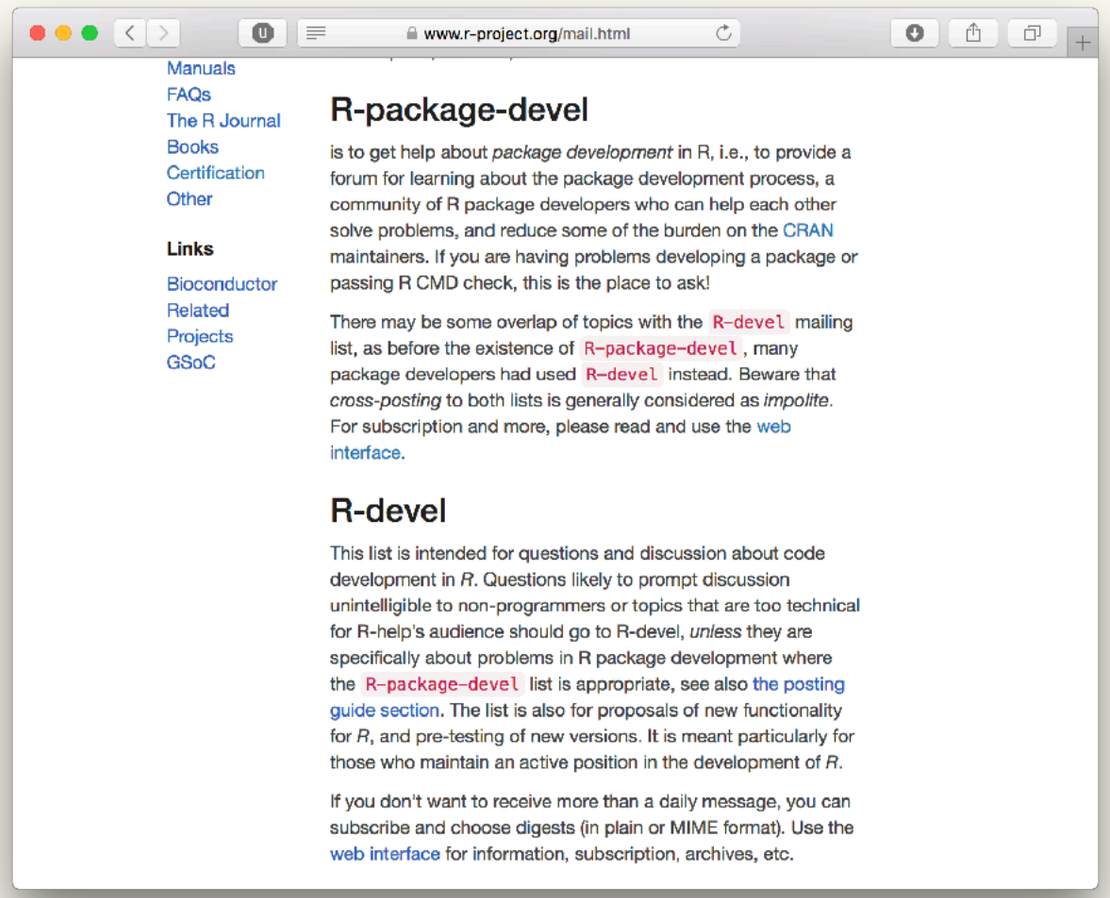
Subsequent submissions to CRAN

```
# Proceed as before. If you have reverse
dependencies
# you need to also run R CMD check on them, and
# notify CRAN if you have deliberately broken them.
# Fortunately the revdepcheck package makes this
# fairly easy

install_github("r-lib/revdepcheck")
usethis::use_revdep_check()

library(revdepcheck)
revdep_check(num_workers = 4)
revdep_report_cran()
```

Learning more

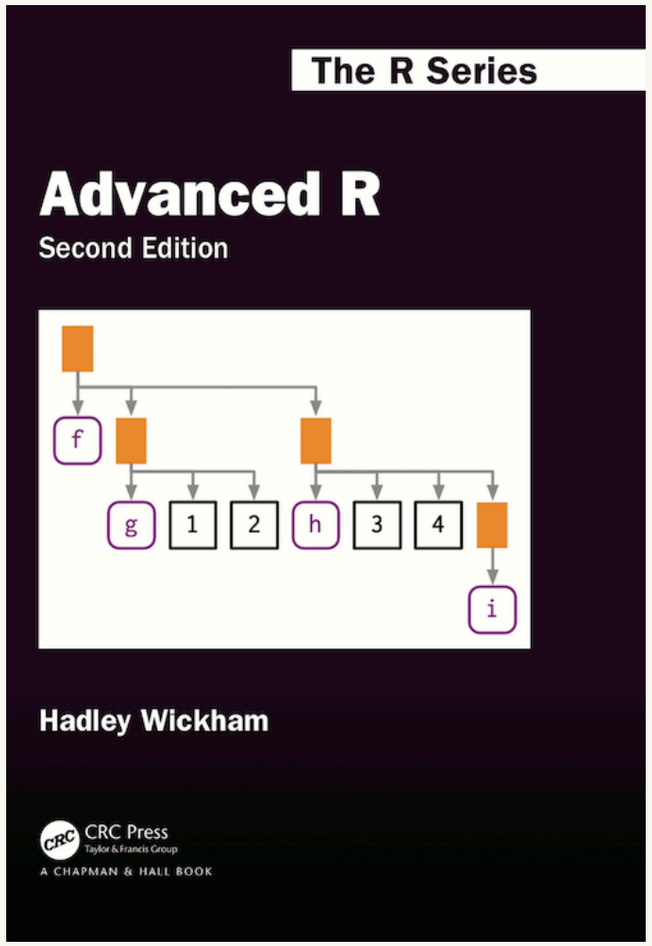


community.rstudio.com

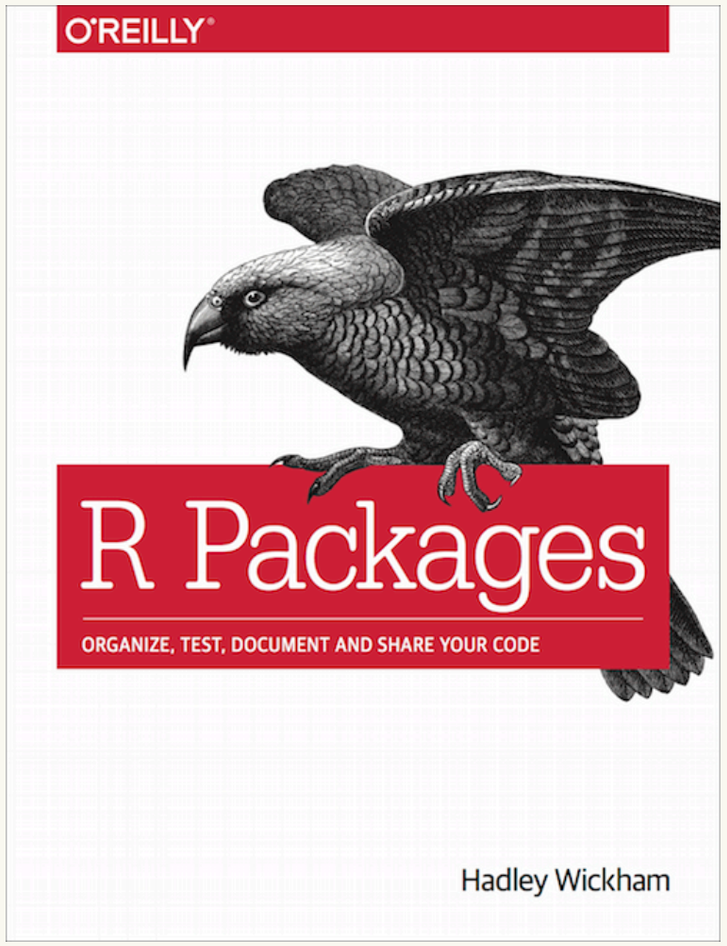


R-package-devel mailing list

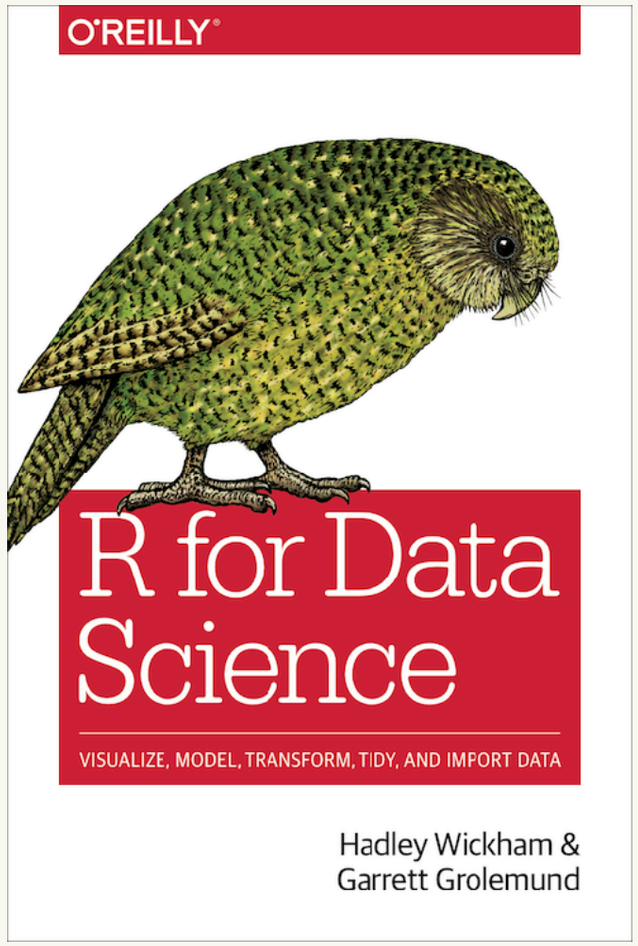
More details in books



http://adv-r.hadley.nz/

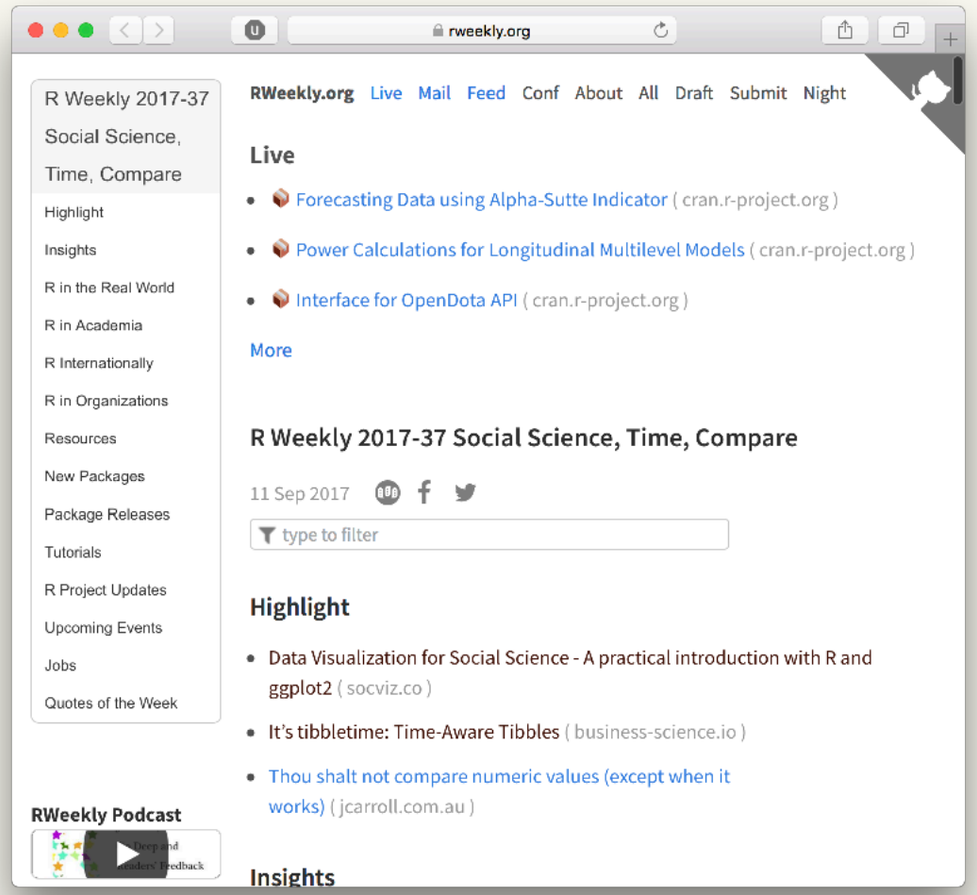


http://r-pkgs.had.co.nz

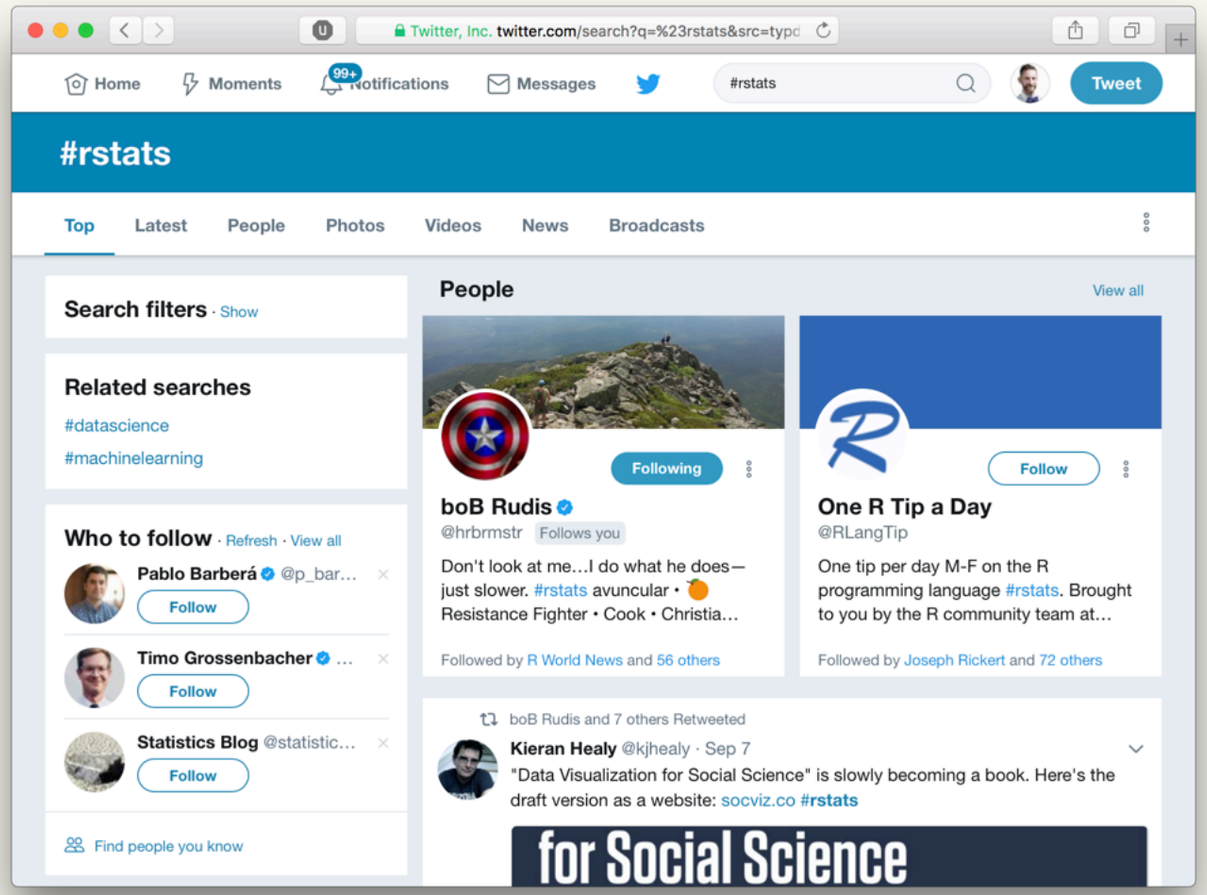


http://r4ds.had.co.nz

rweekly.org



#rstats



stackoverflow

[r] score:5 is:question closed:no

Thank you!