

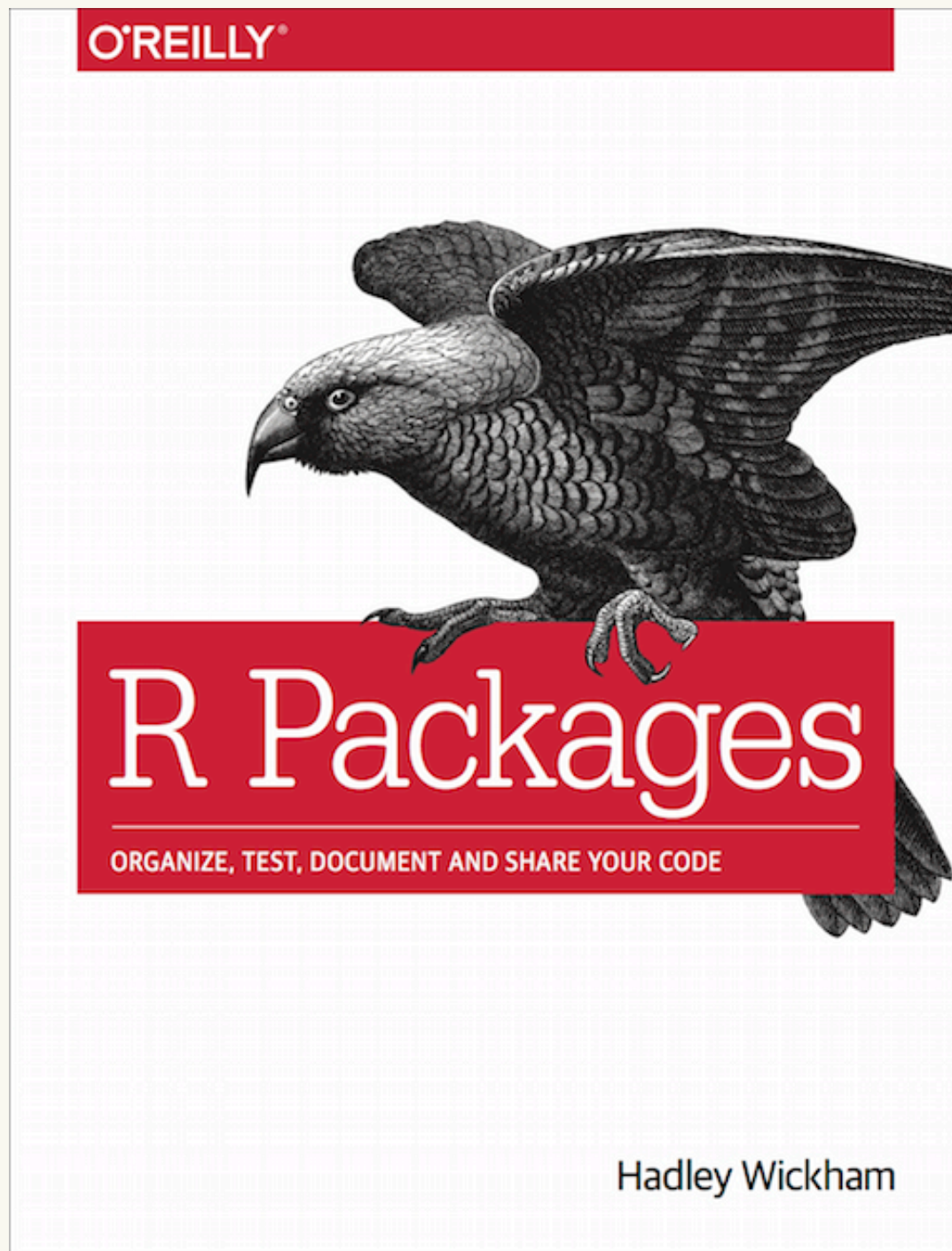
Course prep

```
install.packages(c(
  "devtools", "roxygen2",
  "testthat", "covr"
))
usethis::use_course(
  "http://bit.ly/30kL8QD"
)
# https://github.com/hadley/pkg-dev
```

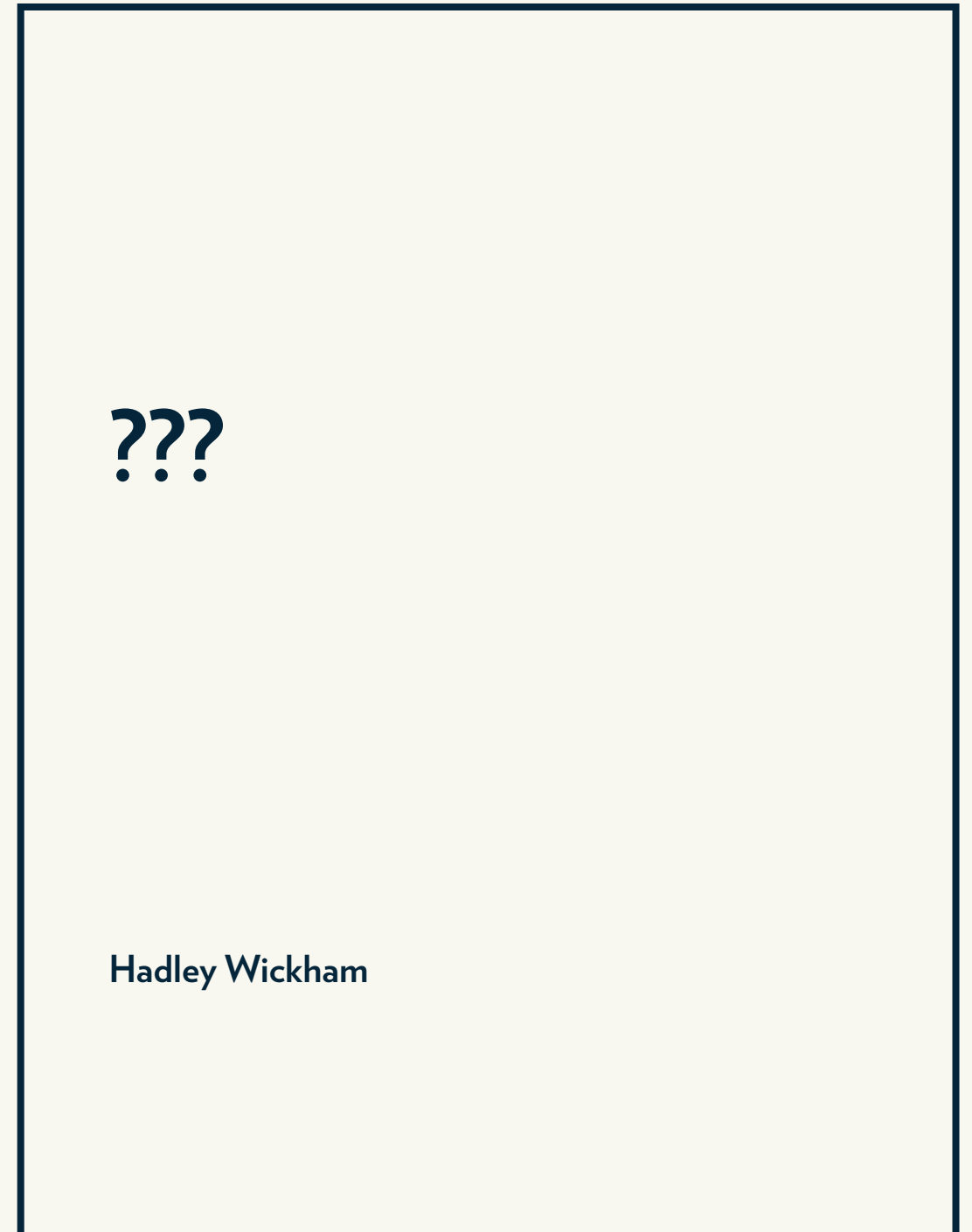
Intro & basic package workflow

August 2019

Hadley Wickham
@hadleywickham



<https://r-pkgs.org/>



<https://principles.tidyverse.org>

<https://github.com/hadley/pkg-dev>

Your turn

This course is hands-on and, while we're here to help, the best resource may be the person sitting next to you.

This means that you have to work!

Introduce yourself to your neighbours. Who are you and what are you using R for?

<https://github.com/hadley/pkg-dev>

Warmup

Get to know your R installation!

Your turn

```
.Library
```

```
.libPaths()
```

```
installed.packages()
```

```
# What is a library?
```

```
# Where's your default library?
```

```
# Which libraries are searched for packages?
```

```
# How many packages do you have installed?
```

```
# Make a frequency table of package Priority.
```

Package = natural unit for distributing R code

base R \approx 14 base + 15 recommended packages

ship with all binary distributions of R

can use right out of the box:

```
library(lattice)
```

Package = natural unit for distributing R code

CRAN has ~14K additional packages

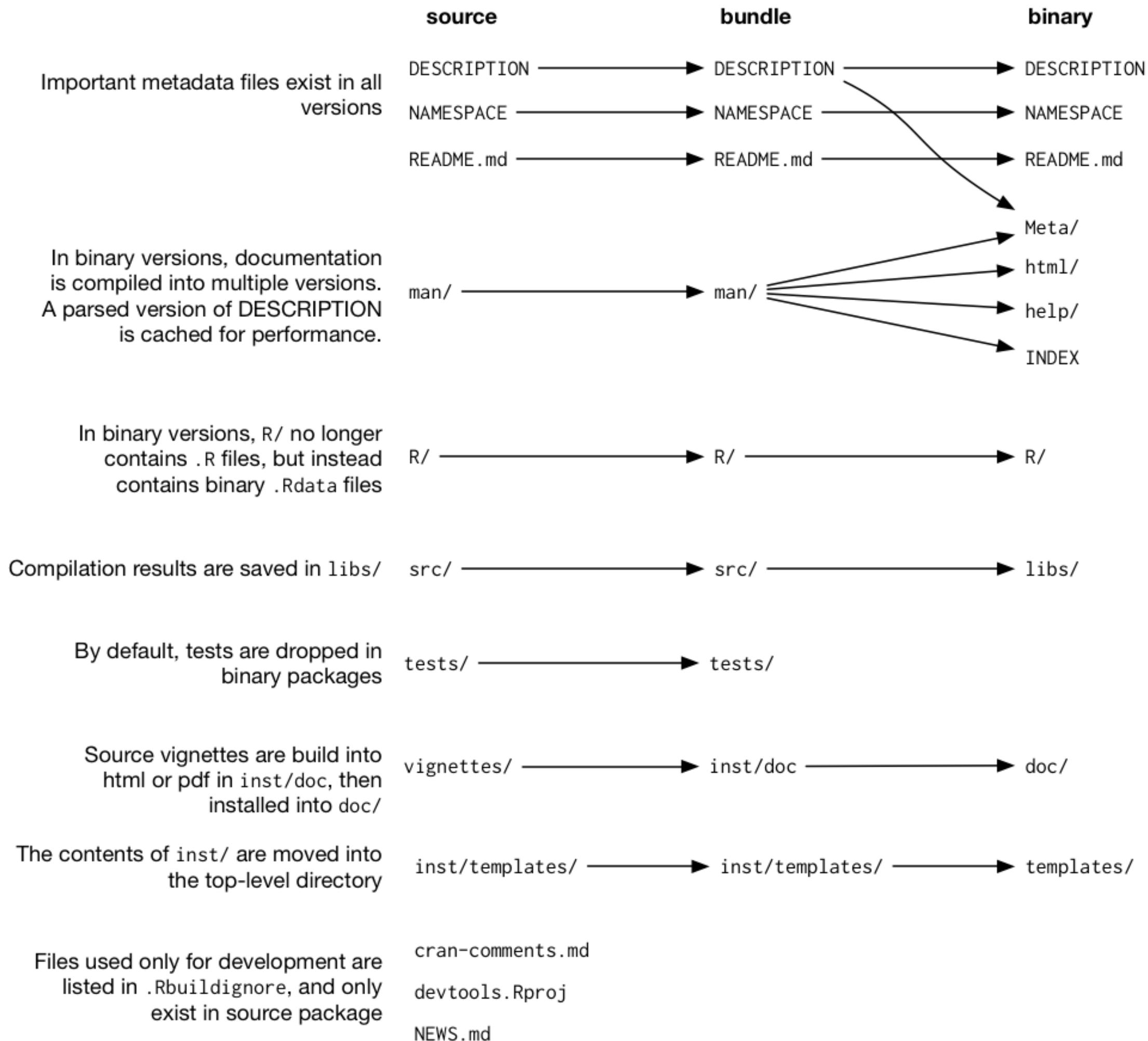
install, then attach:

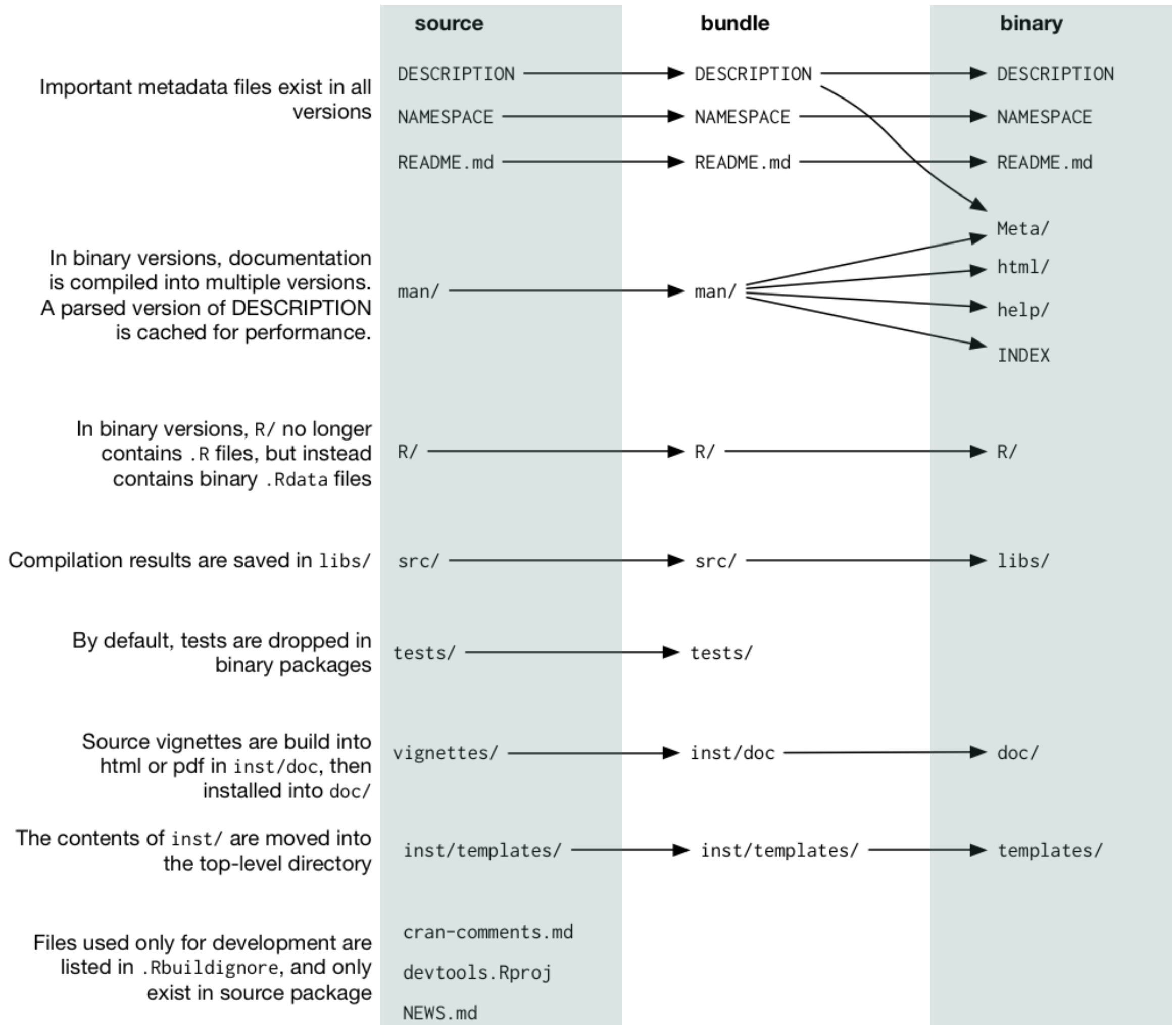
```
install.packages("devtools")  
library(devtools)
```

And then there's GitHub ...

install via devtools, then attach:

```
devtools::install_github("jimhester/lookup")  
library(lookup)
```



**A package is a set of
conventions that
(with the right tools)
makes your life easier**

foofactors

```
usethis::create_package()
```

Your turn

Create a package with:

```
usethis::create_package("~/Desktop/foofactors")
```

Substitute your preferred location.

What files and directories are created?

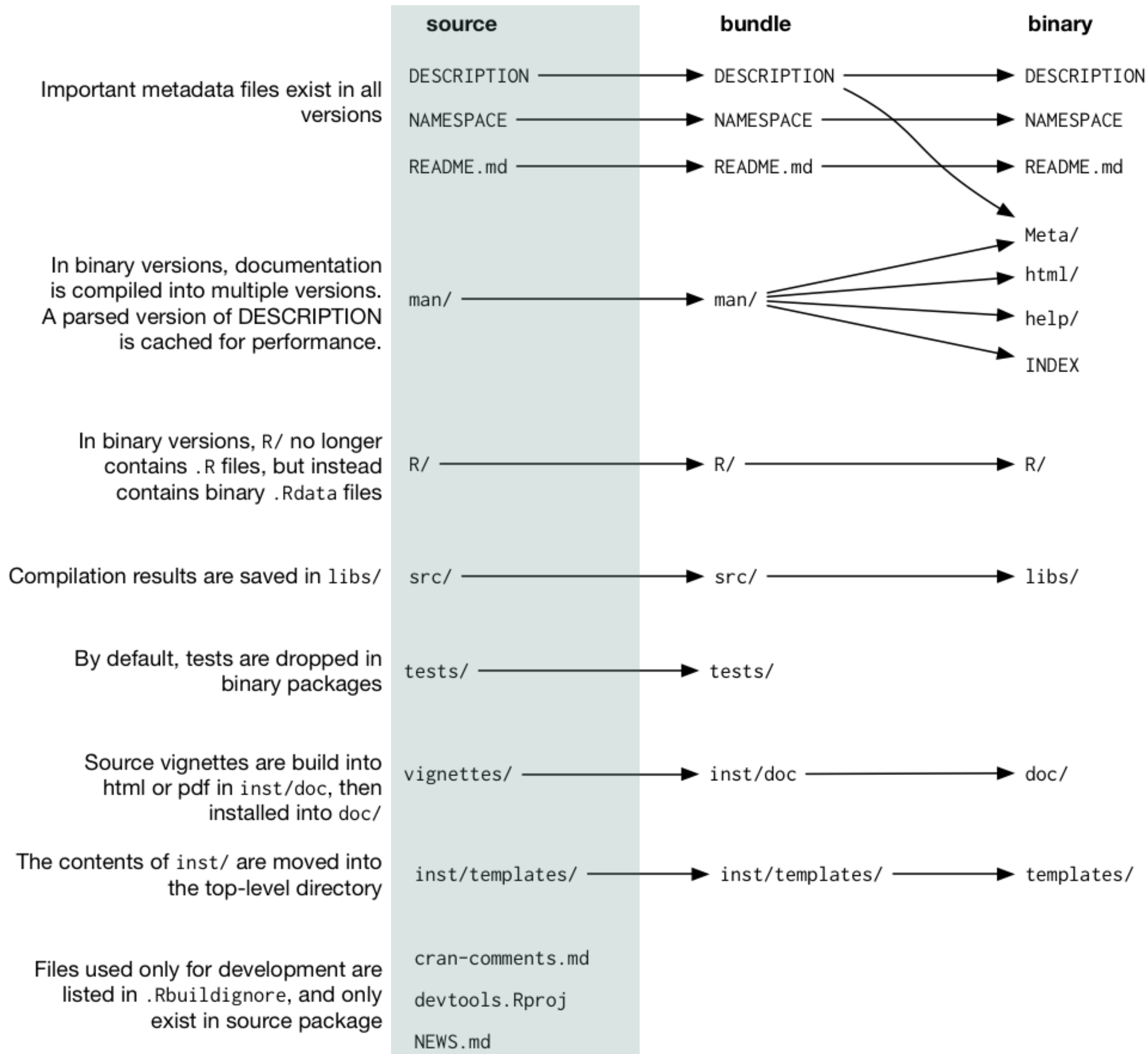
Compare to source package diagram.

FYI, you can also create new project using

RStudio but it has some slight differences.

What does `create_package()` do?

- ✓ Creating `'/Users/jenny/tmp/foofactors2/'`
 - ✓ Setting active project to `'/Users/jenny/tmp/foofactors2'`
 - ✓ Creating `'R/'`
 - ✓ Writing `'DESCRIPTION'`
- Package: `foofactors2`
- Title: What the Package Does (One Line, Title Case)
- Version: `0.0.0.9000`
- Authors@R (parsed):
- * Jennifer Bryan <jenny@rstudio.com> [aut, cre]
- Description: What the package does (one paragraph).
- License: MIT + file LICENSE
- Encoding: UTF-8
- LazyData: true
- ✓ Writing `'NAMESPACE'`
 - ✓ Writing `'foofactors2.Rproj'`
 - ✓ Adding `'Rproj.user'` to `'gitignore'`
 - ✓ Adding `'^foofactors2\\.Rproj$'`, `'^\\.Rproj\\.user$'` to `'Rbuildignore'`
 - ✓ Opening `'/Users/jenny/tmp/foofactors2/'` in new RStudio session
 - ✓ Setting active project to `'<no active project>'`



The whole game

What follows is adapted from

The Whole Game

chapter in the revised version of R Packages.

<https://r-pkgs.org/whole-game.html>

A proper package for the care and feeding of factors:

forcats

<https://forcats.tidyverse.org>

foofactors::fbind()

Factors can be vexing

```
(a <- factor(c("character", "hits", "your", "eyeballs")))  
#> [1] character hits          your          eyeballs  
#> Levels: character eyeballs hits your
```

```
(b <- factor(c("but", "integer", "where it", "counts")))  
#> [1] but          integer    where it counts  
#> Levels: but counts integer where it
```

```
c(a, b)  
#> [1] 1 3 4 2 1 3 4 2
```

Factors can be vexing

```
factor(c(as.character(a), as.character(b)))  
#> [1] character hits      your      eyeballs but      integer  where it  
#> [8] counts  
#> Levels: but character counts eyeballs hits integer where it your
```

Let's turn this into our first function:

```
fbind()
```

Where do we define functions?

Beautiful pairing:
`use_r()` & `use_test()`

There's a `usethis` helper for that too!

```
usethis::use_r("file-name")
```

Organise files so that related code

lives together. If you can give a file

a concise and informative name, it's

probably about right

Your turn

Use `usethis::use_r("fbind")` to create a new file

In it, define a function named "fbind" that combines its inputs (presumably factors) like so:

- coerce each input to character
- combine inputs
- make output a factor

```
factor(c(as.character(a), as.character(b)))
```

Now what?

```
source("R/fbind.R")
```

Use IDE tricks to send definition of
fbind() to the R Console

Now what?

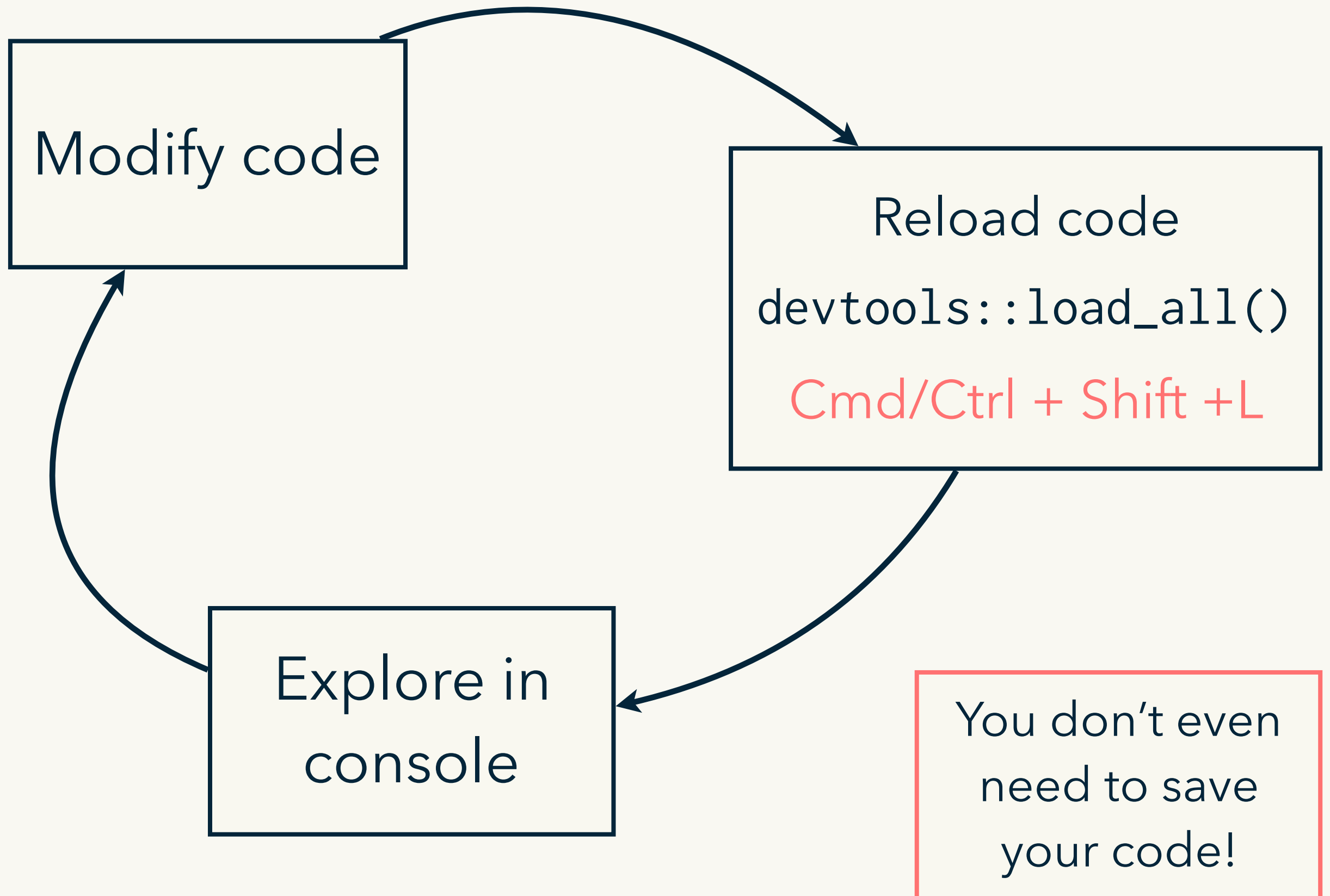
~~source("R/fbind.R")~~

~~Use IDE tricks to send definition of
fbind() to the R Console~~

devtools::load_all()

devtools::load_all()

Why do we love devtools? Workflow!



Important metadata files exist in all versions

In binary versions, documentation is compiled into multiple versions. A parsed version of DESCRIPTION is cached for performance.

In binary versions, R/ no longer contains .R files, but instead contains binary .Rdata files

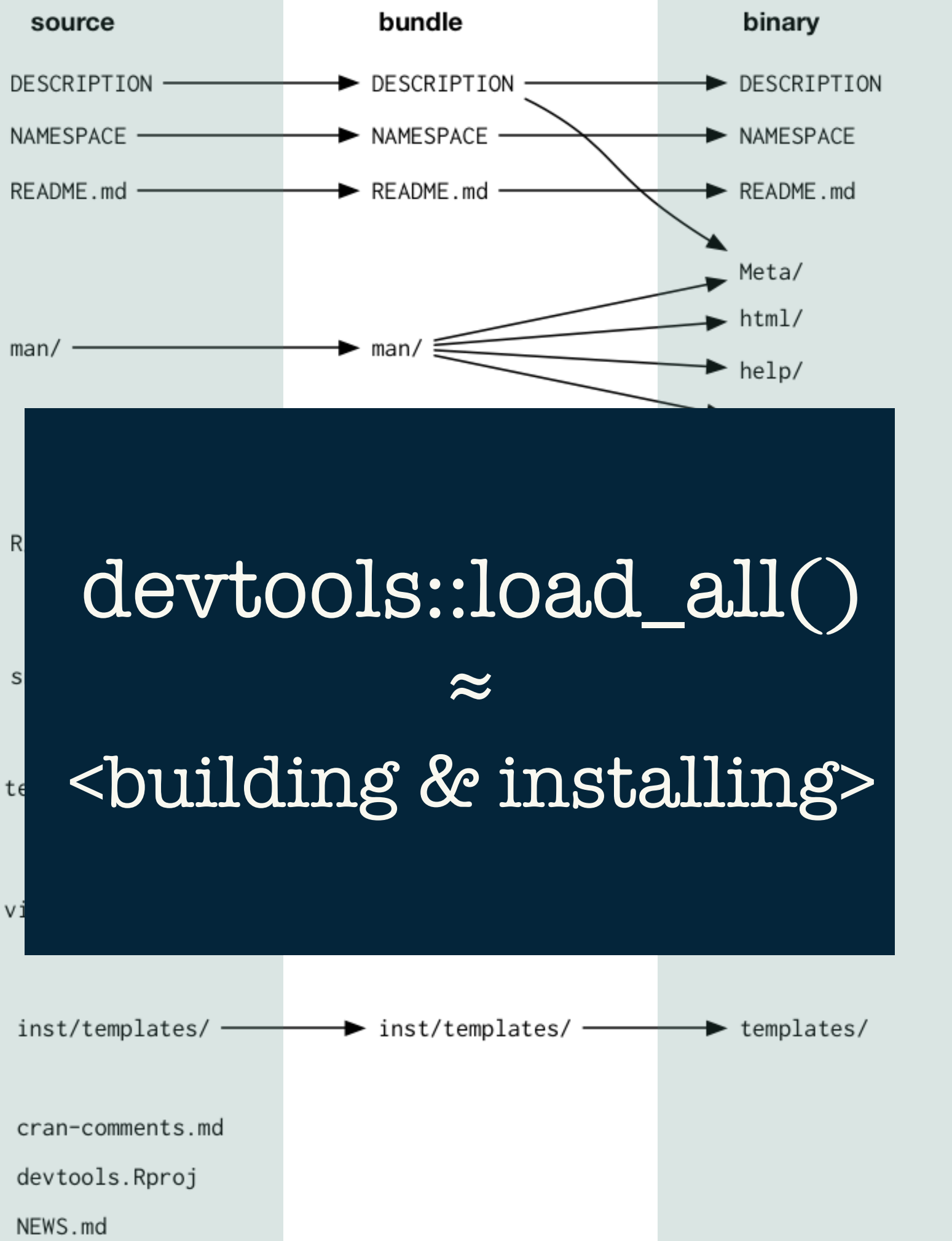
Compilation results are saved in libs/

By default, tests are dropped in binary packages

Source vignettes are build into html or pdf in inst/doc, then installed into doc/

The contents of inst/ are moved into the top-level directory

Files used only for development are listed in .Rbuildignore, and only exist in source package



R/RStudio setup

Workflow setup: your .Rprofile

```
# Setup code that is run at R startup:
```

```
# usethis::edit_r_profile()
```

```
# Helper to add devtools specifically:
```


```
# usethis::use_devtools()
```

devtools makes
usethis available too!

```
if (interactive()) {  
  suppressMessages(require(devtools))  
  suppressMessages(require(testthat))  
}
```

Never include analysis packages here

```
if (interactive()) {  
  suppressMessages(require(ggplot2))  
  suppressMessages(require(dplyr))  
}
```

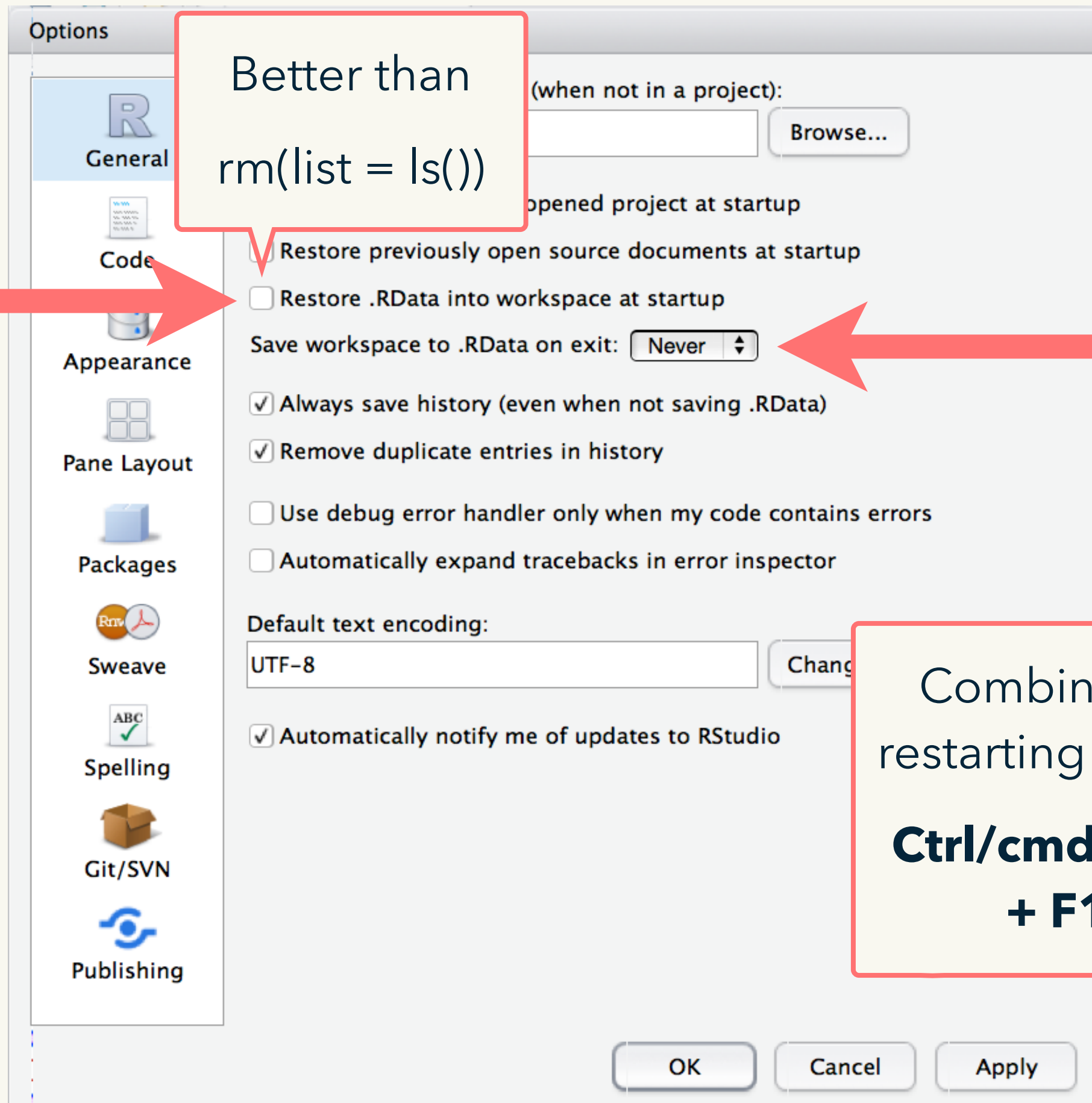


While you're in there, also add:

```
# Helper to add devtools specifically:
```

```
# usethis::use_partial_warnings()
```

```
options(  
  warnPartialMatchArgs = TRUE,  
  warnPartialMatchDollar = TRUE,  
  warnPartialMatchAttr = TRUE  
)
```

Better than
`rm(list = ls())`

Combine with
restarting R often!
**Ctrl/cmd + shift
+ F10**

devtools::check()



`check()` \approx R CMD check

Checks package for technical validity

Do from R (or RStudio Ctrl/cmd + shift + e)

`check()` early, `check()` often

Get it working, keep it working

Necessary (but not sufficient) for CRAN

Excellent way to run your tests (and more)

Your turn

Edit DESCRIPTION (optional)

- Make yourself the author
- Update Title and Description

Nice to do, but
skippable.

`use_mit_license("Your Name")`

Fixes 1 of our 2
warnings.

`check()` again, if you wish

devtools::document()

roxygen2 turns comments into help

```
#' Bind two factors
#
# Create a new factor from two existing factors, where the new
# factor's levels are the union of the levels of the input
# factors.
#
#' @param a factor
#' @param b factor
#
#' @return factor
#' @export
#' @examples
#' fbind(factor(letters[1:3]), factor(letters[26:24]))
fbind <- function(a, b) {
  factor(c(as.character(a), as.character(b)))
}
```

RStudio helper:
Code > Insert roxygen skeleton

RStudio helper:


Code > Insert roxygen skeleton

Your turn

Add docs for `fbind()` as a roxygen comment

- RStudio helper: *Code > Insert roxygen skeleton*
- Lines MUST start with `#'`

`document()`



Makes an `.Rd` file
from the comment

`check()` again and ... rejoice!

devtools::check()



devtools::install()



`install()` \approx R CMD install

- Makes an *installed* pkg from your *source* pkg
- Do from R (or RStudio *Install and Restart*)
- `install()` less often than you `load_all()` or `check()`
- Marks transition from developing your package to using your package

Important metadata files exist in all versions

In binary versions, documentation is compiled into multiple versions. A parsed version of DESCRIPTION is cached for performance.

In binary versions, R/ no longer contains .R files, but instead contains binary .Rdata files

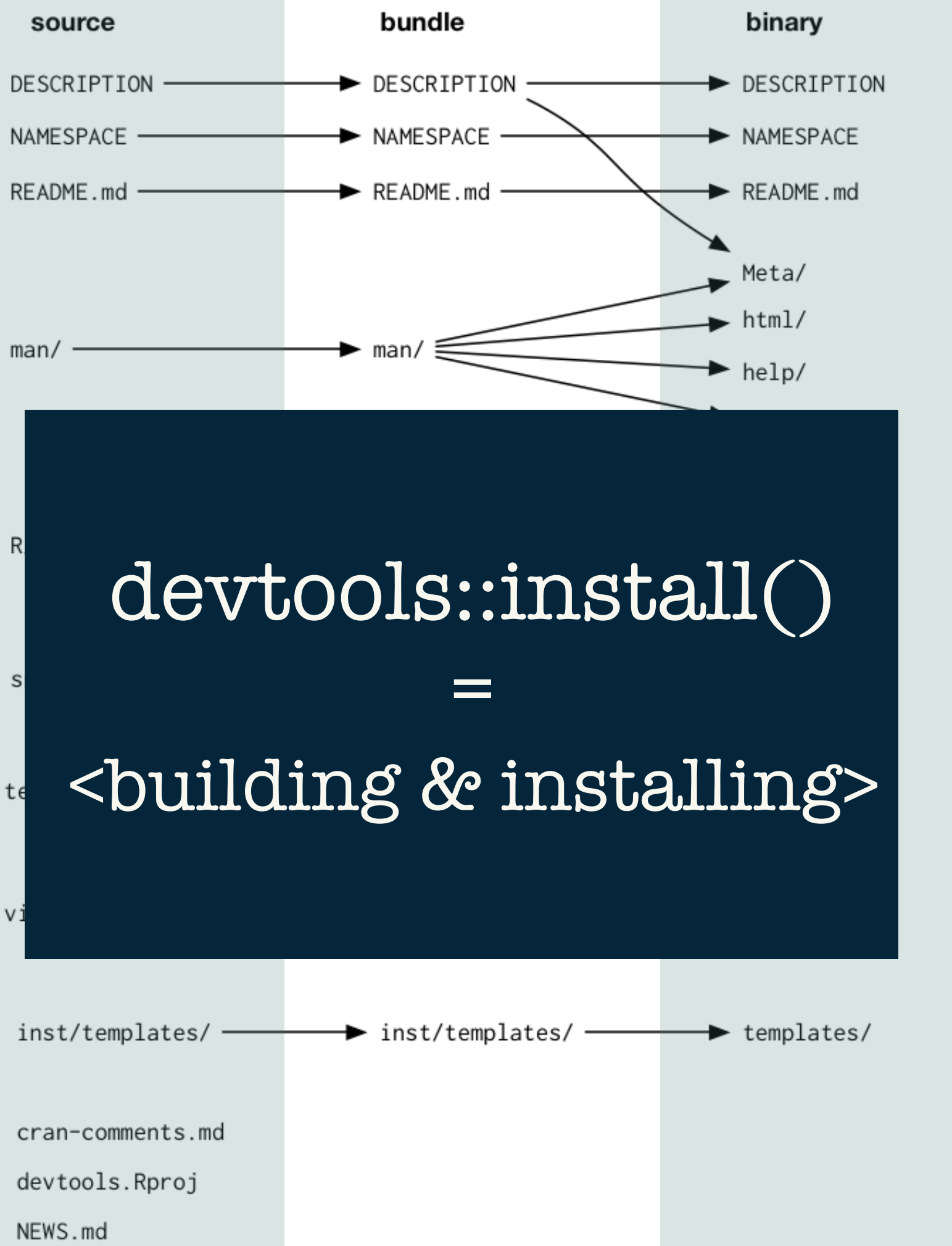
Compilation results are saved in libs/

By default, tests are dropped in binary packages

Source vignettes are build into html or pdf in inst/doc, then installed into doc/

The contents of inst/ are moved into the top-level directory

Files used only for development are listed in .Rbuildignore, and only exist in source package



Your turn

Install your foofactors package

- Call `install()` in R
- RStudio *Build & Restart*
- Shell: R CMD build + R CMD install

Restart R

Attach like a "regular" package with `library()`

Call `fbind()`

Revel in your success!

Other goodies if time allows

- `use_package_doc()`
- `use_git() + use_github()`

usethis::create_package()

devtools::load_all()

devtools::check()

devtools::document()

devtools::install()

This work is licensed as
Creative Commons
Attribution-ShareAlike 4.0
International

To view a copy of this license, visit
[https://creativecommons.org/
licenses/by-sa/4.0/](https://creativecommons.org/licenses/by-sa/4.0/)