Course prep

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

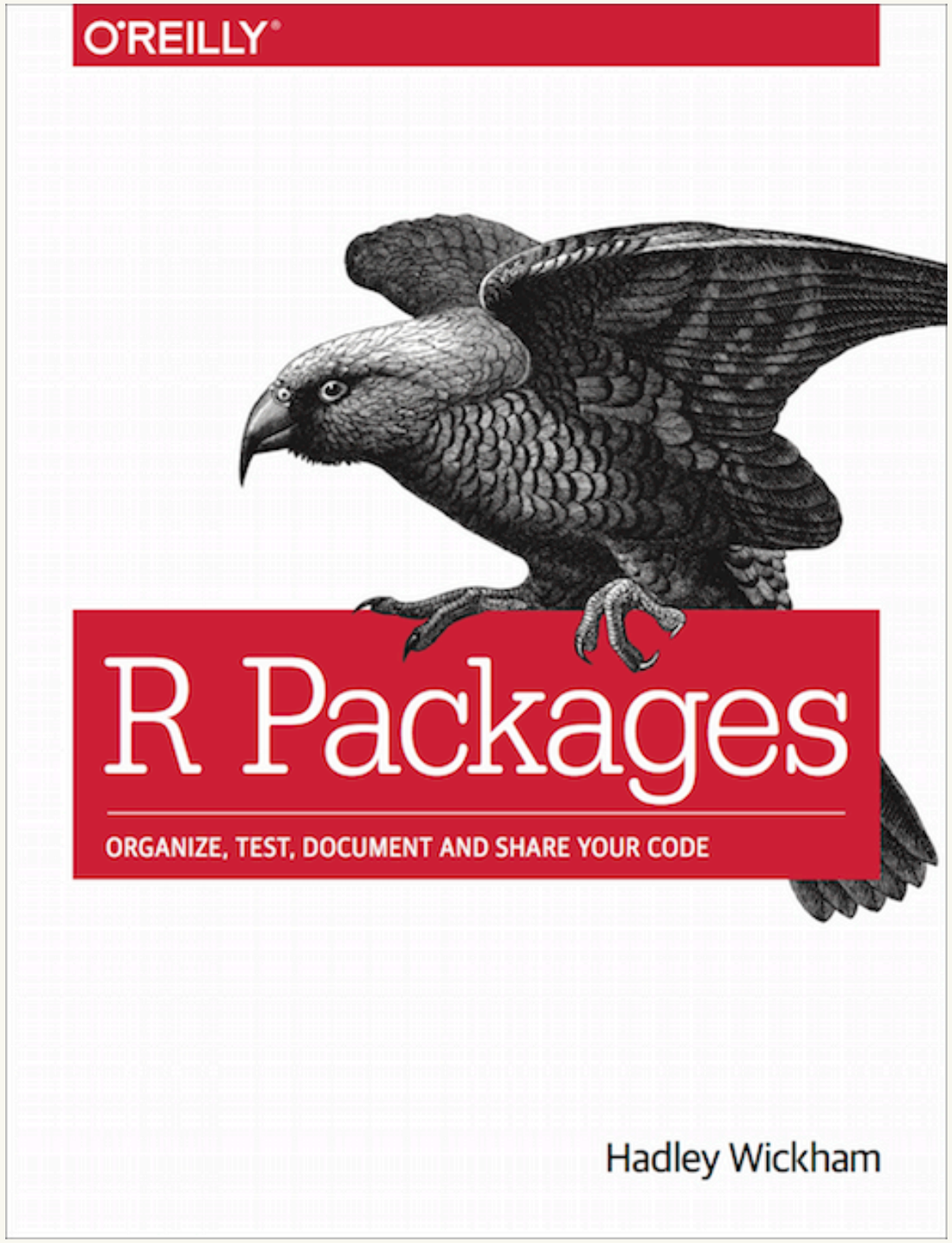
Intro & basic package workflow

August 2019

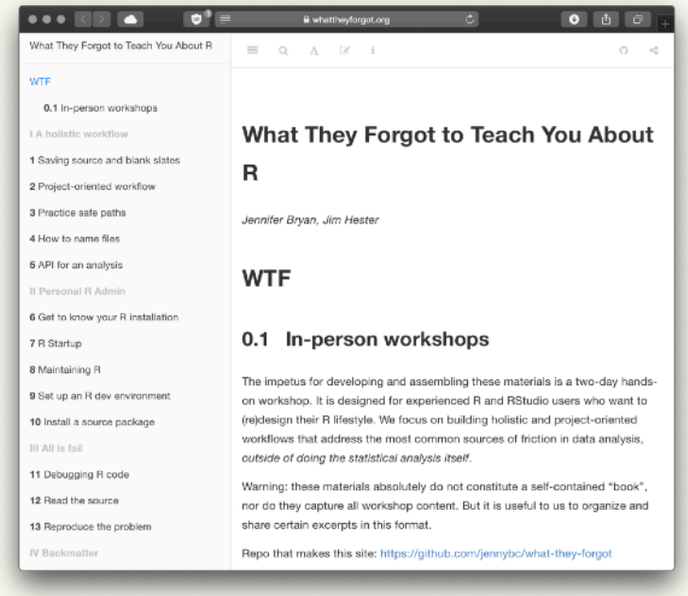
Hadley Wickham
@hadleywickham

Overview

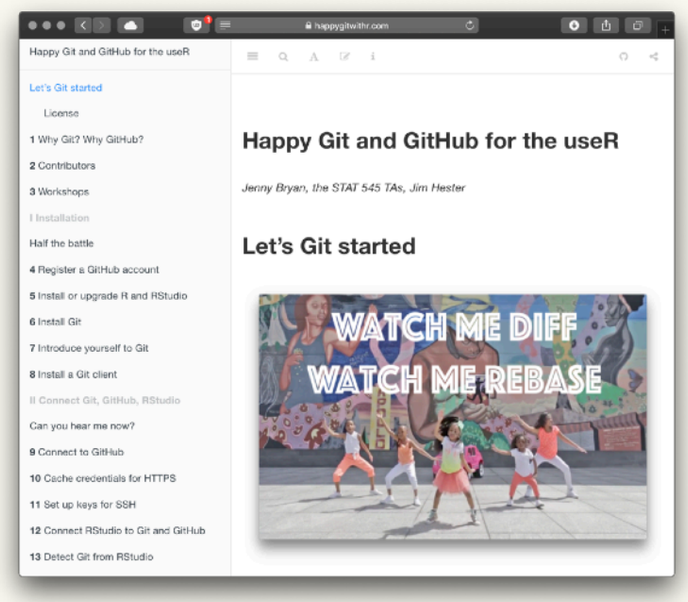
1. Warmups
2. "The whole game"
3. Testing
4. Documentation
5. Sharing



https://r-pkgs.org/



https://whattheyforgot.org



https://happygitwithr.com

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.

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

This means that you have to work!

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 ≈ 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

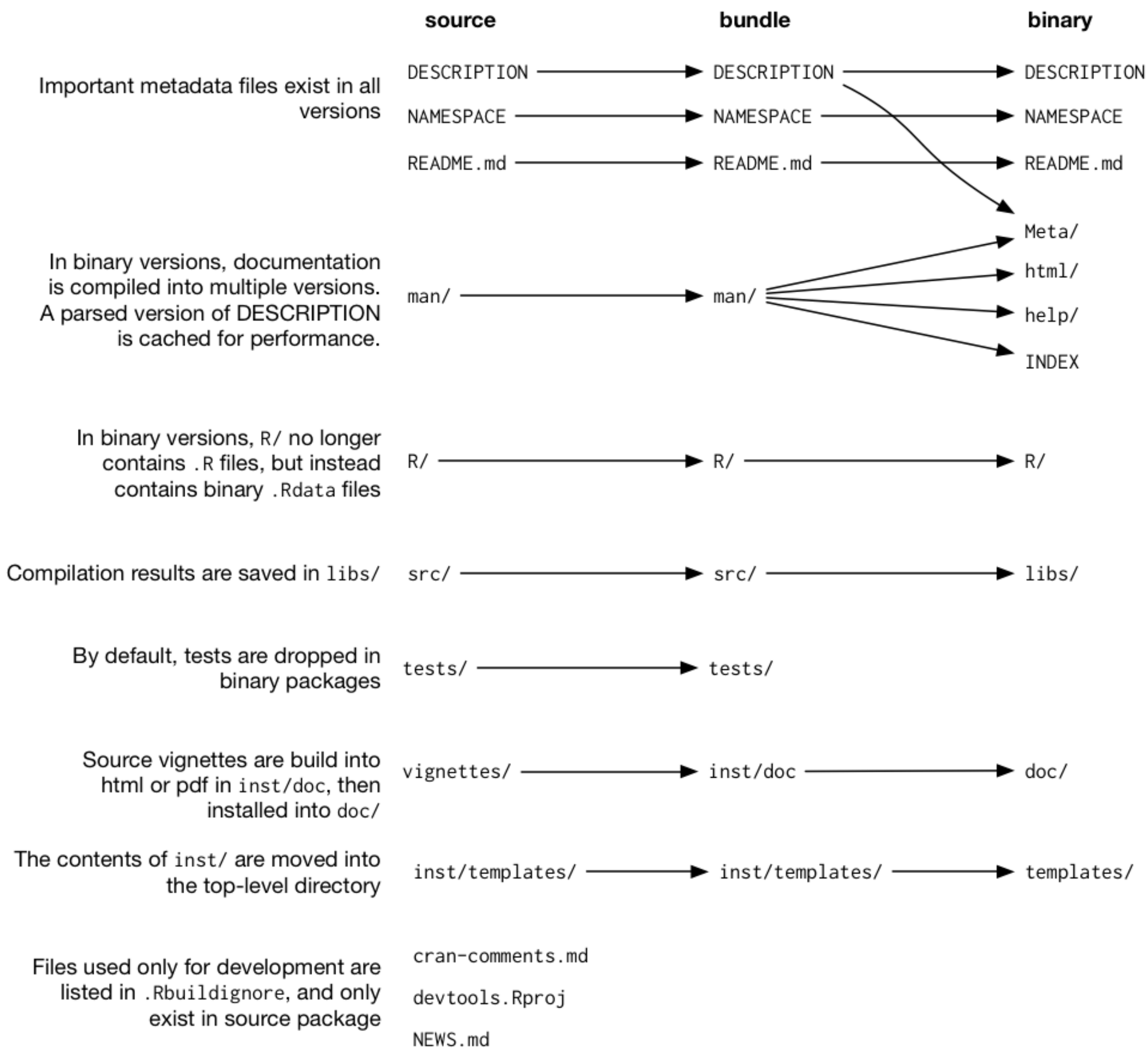
install, then attach:

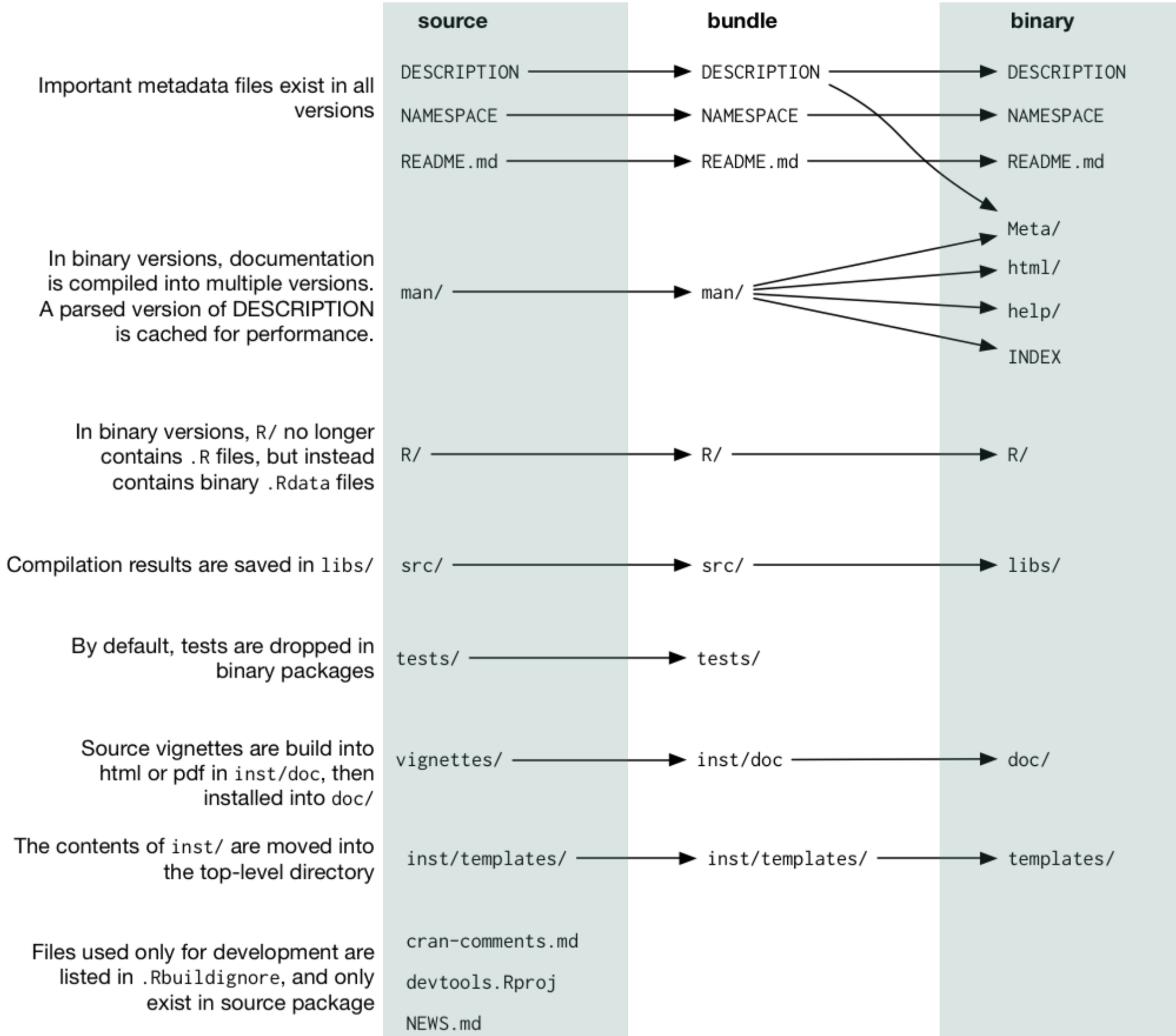
```
install.packages("devtools")
library(devtools)
```

And then there's GitHub ...

install via devtools, then attach:

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


source
bundle
binary
Important metadata files exist in all versions
DESCRIPTION
NAMESPACE
README.md
DESCRIPTION
NAMESPACE
README.md
DESCRIPTION
NAMESPACE
README.md
Meta/
html/
help/
INDEX
In binary versions, documentation is compiled into multiple versions. A parsed version of DESCRIPTION is cached for performance.
man/
man/
In binary versions, R/ no longer contains .R files, but instead contains binary .Rdata files
R/
R/
R/
Compilation results are saved in libs/
src/
src/
libs/
By default, tests are dropped in binary packages
tests/
tests/
Source vignettes are build into html or pdf in inst/doc, then installed into doc/
vignettes/
inst/doc
doc/
The contents of inst/ are moved into the top-level directory
inst/templates/
inst/templates/
templates/
Files used only for development are listed in .Rbuildignore, and only exist in source package
cran-comments.md
devtools.Rproj
NEWS.md


source
bundle
binary
Important metadata files exist in all versions
DESCRIPTION
NAMESPACE
README.md
DESCRIPTION
NAMESPACE
README.md
DESCRIPTION
NAMESPACE
README.md
Meta/
html/
help/
INDEX
In binary versions, documentation is compiled into multiple versions. A parsed version of DESCRIPTION is cached for performance.
man/
man/
In binary versions, R/ no longer contains .R files, but instead contains binary .Rdata files
R/
R/
R/
Compilation results are saved in libs/
src/
src/
libs/
By default, tests are dropped in binary packages
tests/
tests/
Source vignettes are build into html or pdf in inst/doc, then installed into doc/
vignettes/
inst/doc
doc/
The contents of inst/ are moved into the top-level directory
inst/templates/
inst/templates/
templates/
Files used only for development are listed in .Rbuildignore, and only exist in source package
cran-comments.md
devtools.Rproj
NEWS.md

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

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

usethis::create_package()

Your turn

```
# Create a package with:
usethis::create_package("~/Desktop/foofactors")
```

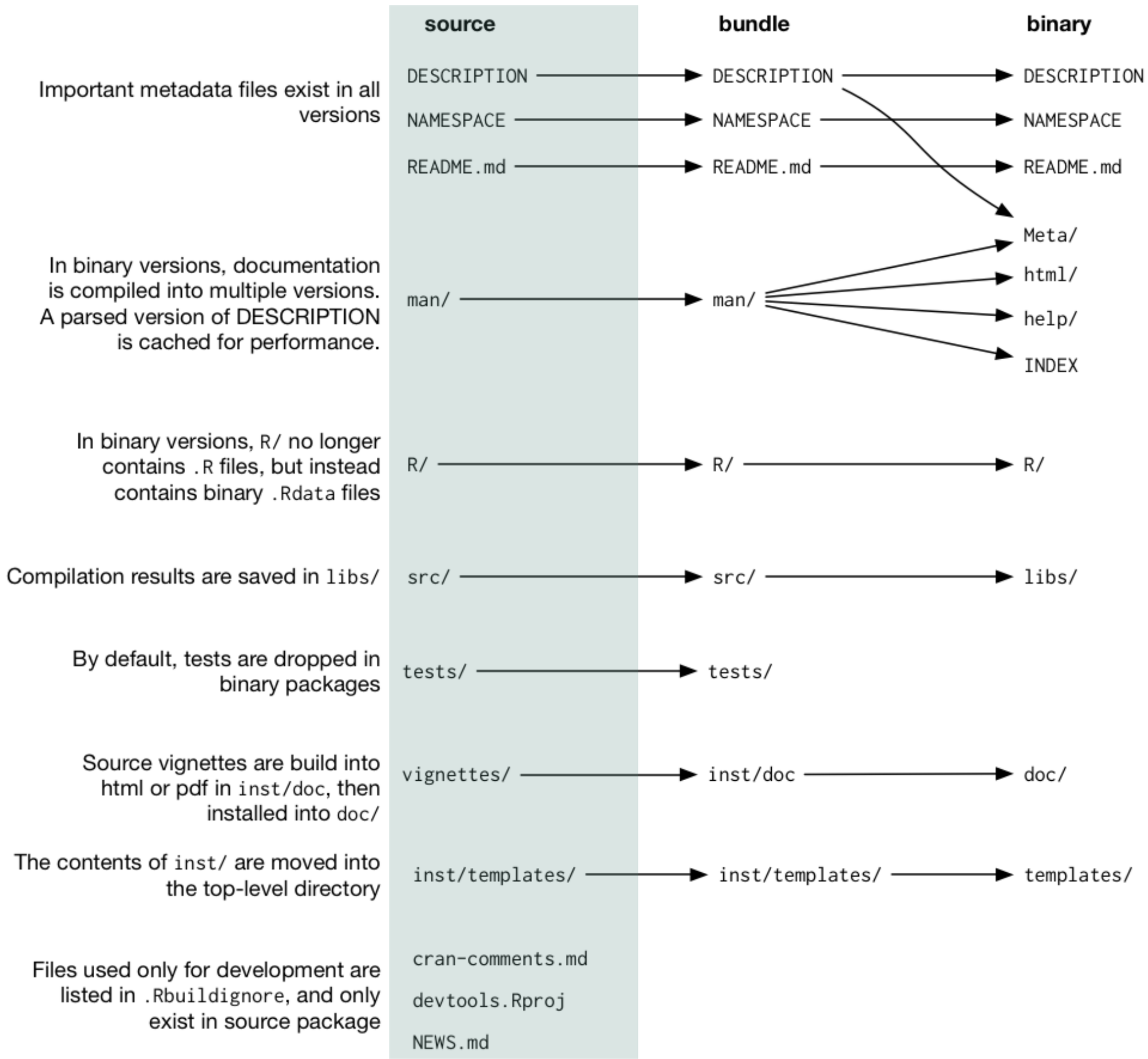
Substitute your preferred location.

```
# 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>'
```


source
bundle
binary
Important metadata files exist in all versions
DESCRIPTION
NAMESPACE
README.md
DESCRIPTION
NAMESPACE
README.md
DESCRIPTION
NAMESPACE
README.md
Meta/
html/
help/
INDEX
In binary versions, documentation is compiled into multiple versions. A parsed version of DESCRIPTION is cached for performance.
man/
man/
In binary versions, R/ no longer contains .R files, but instead contains binary .Rdata files
R/
R/
R/
Compilation results are saved in libs/
src/
src/
libs/
By default, tests are dropped in binary packages
tests/
tests/
Source vignettes are build into html or pdf in inst/doc, then installed into doc/
vignettes/
inst/doc
doc/
The contents of inst/ are moved into the top-level directory
inst/templates/
inst/templates/
templates/
Files used only for development are listed in .Rbuildignore, and only exist in source package
cran-comments.md
devtools.Rproj
NEWS.md

foofactors::fbind()

Factors can be vexing

```
(a <- factor(c("character", "in", "the", "streets")))
#> [1] character in        the       streets
#> Levels: character in streets the
(b <- factor(c("integer", "in", "the", "sheets")))
#> [1] integer in      the     sheets
#> Levels: in integer sheets the

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

Factors can be vexing

```
factor(c(as.character(a), as.character(b)))
#> [1] character in        the       streets   integer   in
#> [7] the       sheets
#> Levels: character in integer sheets streets the
```

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?

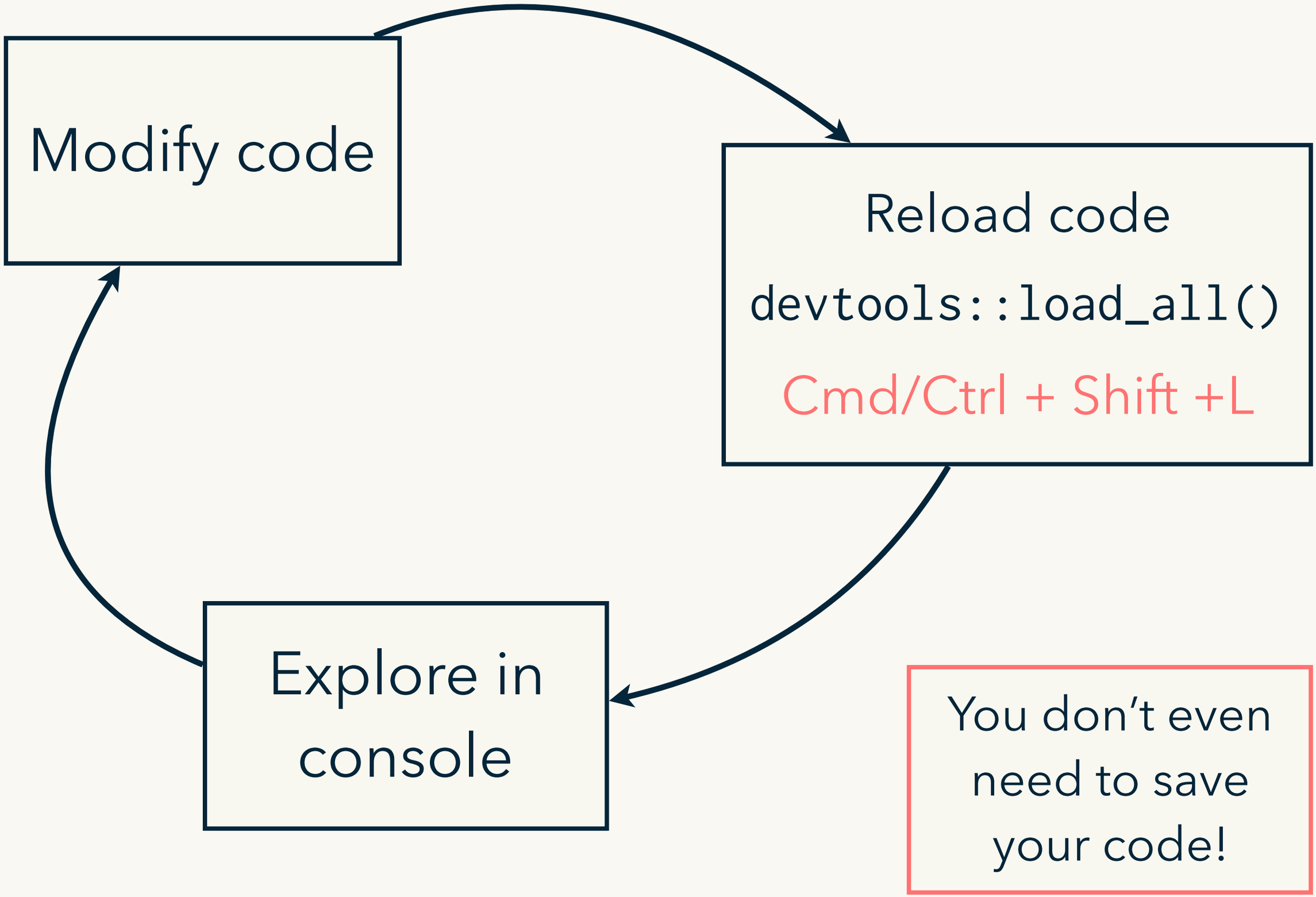
~~`source("R/fbind.R")`~~

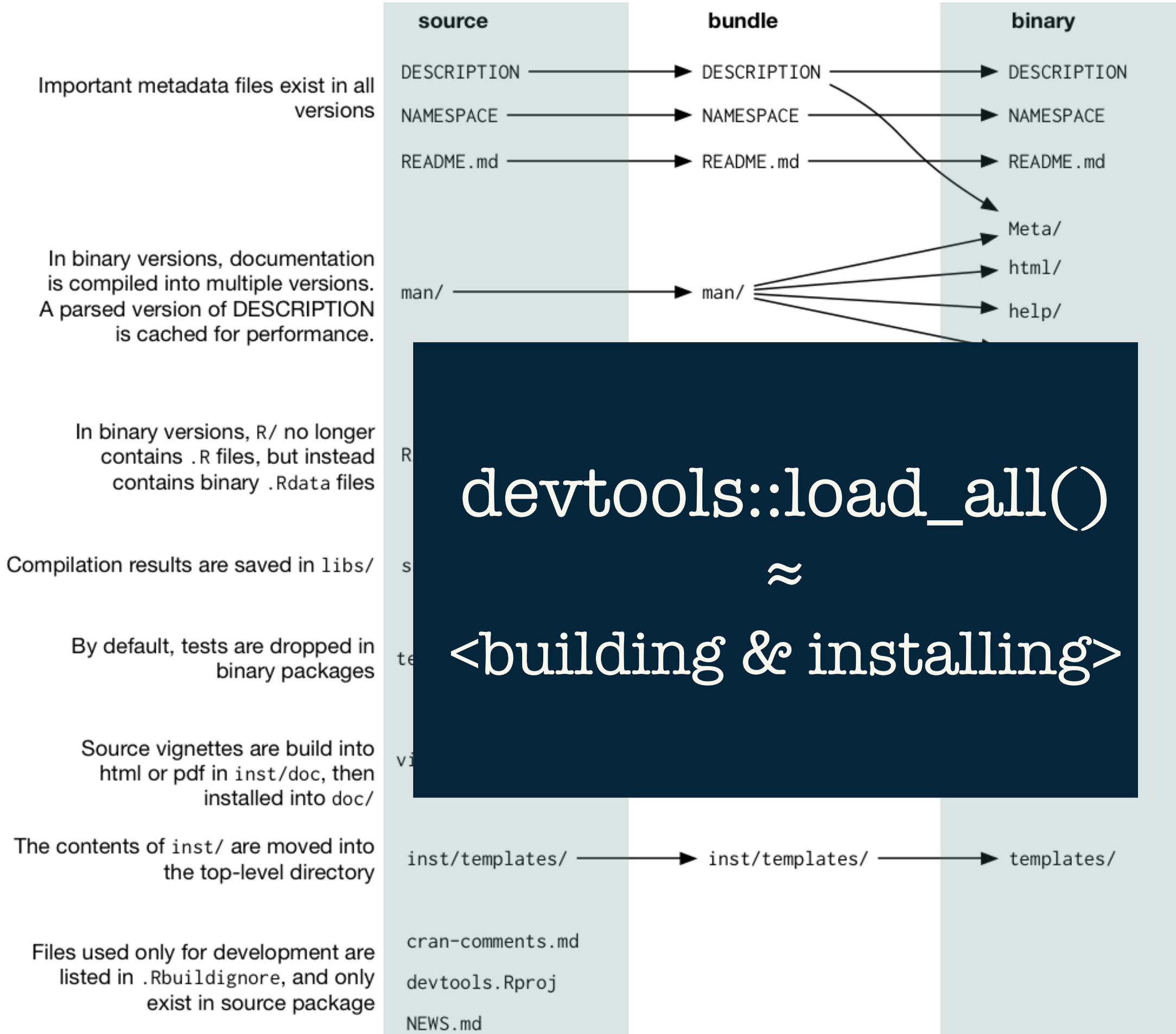
~~Use IDE tricks to send definition of fbind() to the R Console~~

```
devtools::load_all()
```

`devtools::load_all()`

Why do we love devtools? Workflow!




source
bundle
binary
Important metadata files exist in all versions
DESCRIPTION
NAMESPACE
README.md
In binary versions, documentation is compiled into multiple versions. A parsed version of DESCRIPTION is cached for performance.
man/
Meta/
html/
help/
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
inst/templates/
templates/
Files used only for development are listed in .Rbuildignore, and only exist in source package
cran-comments.md
devtools.Rproj
NEWS.md
devtools::load_all()
≈
<building & installing>

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()

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

devtools makes usethis available too!

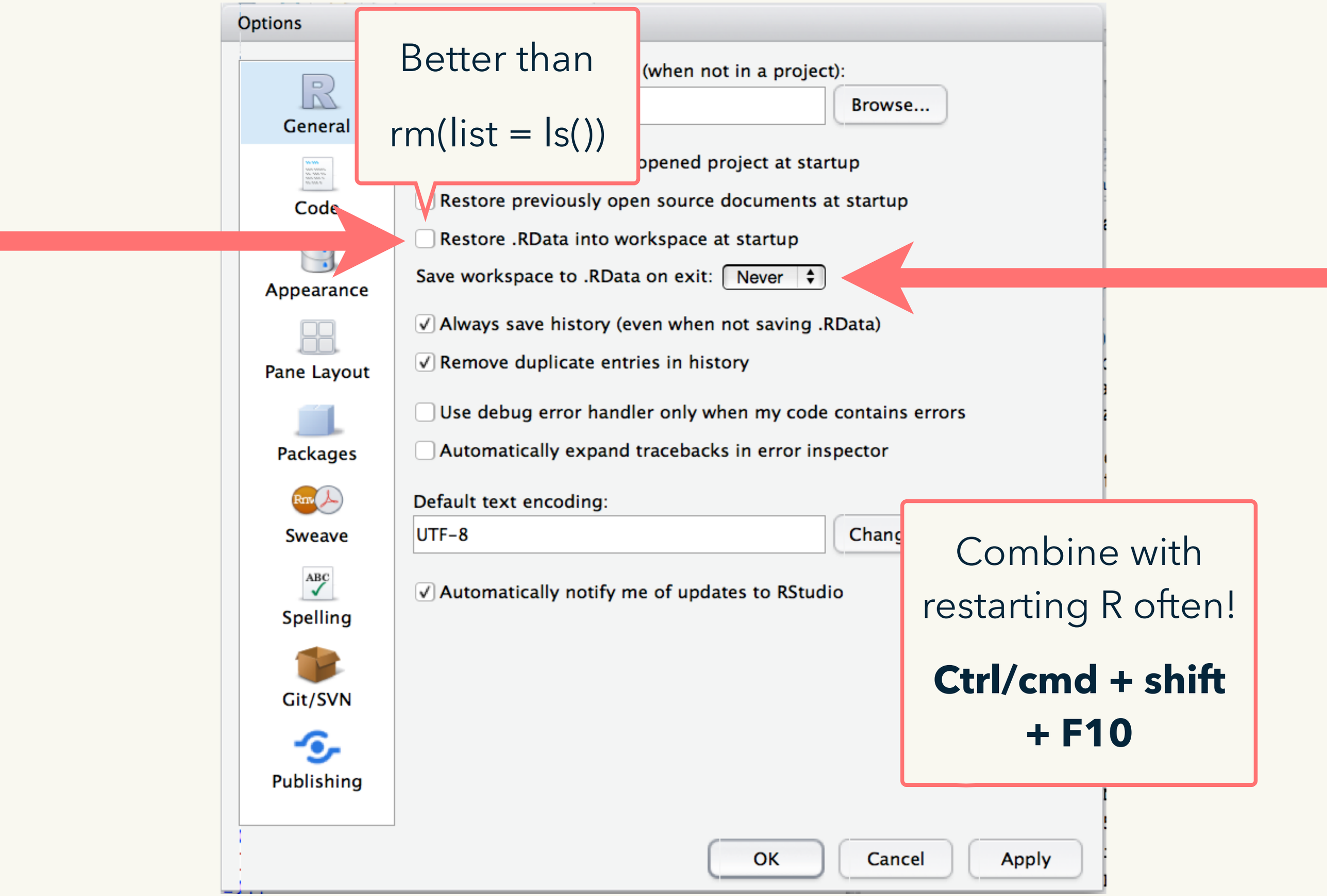
Never include analysis packages here

```
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
)
```


Options
General
Code
Appearance
Pane Layout
Packages
Sweave
Spelling
Git/SVN
Publishing
(when not in a project):
Browse...
opened project at startup
Restore previously open source documents at startup
Restore .RData into workspace at startup
Save workspace to .RData on exit:
Never
Always save history (even when not saving .RData)
Remove duplicate entries in history
Use debug error handler only when my code contains errors
Automatically expand tracebacks in error inspector
Default text encoding:
UTF-8
Automatically notify me of updates to RStudio
OK
Cancel
Apply
Better than
rm(list = ls())
Combine with
restarting R often!
Ctrl/cmd + shift
+ F10

devtools::check()

`check()` ≈ `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

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

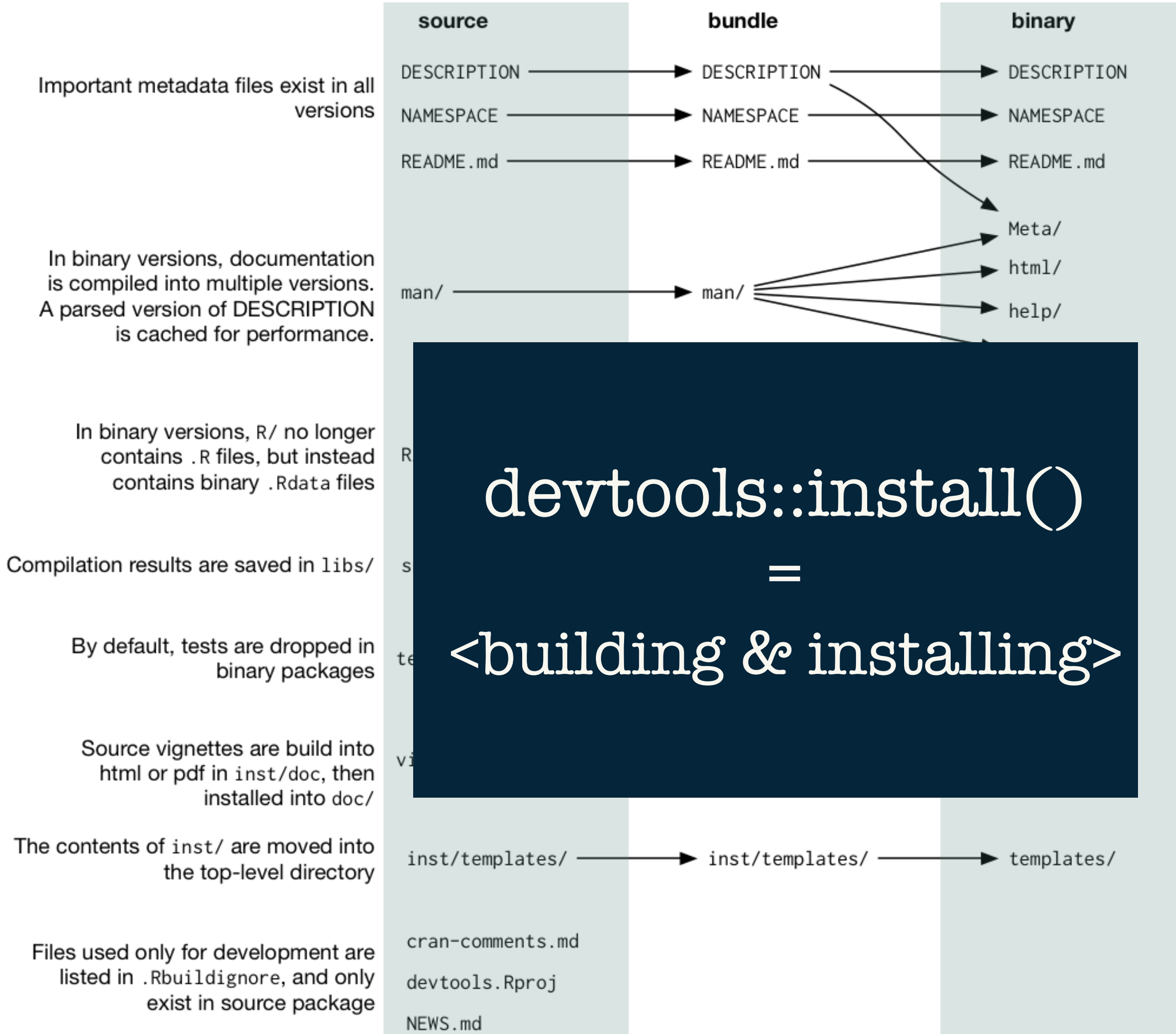
check() again and ... rejoice!

devtools::check()

devtools::install()

install() ≈ 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


source
bundle
binary
Important metadata files exist in all versions
DESCRIPTION
NAMESPACE
README.md
DESCRIPTION
NAMESPACE
README.md
DESCRIPTION
NAMESPACE
README.md
In binary versions, documentation is compiled into multiple versions. A parsed version of DESCRIPTION is cached for performance.
man/
man/
Meta/
html/
help/
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
inst/templates/
inst/templates/
templates/
Files used only for development are listed in .Rbuildignore, and only exist in source package
cran-comments.md
devtools.Rproj
NEWS.md
devtools::install()
=
<building & installing>

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()
```