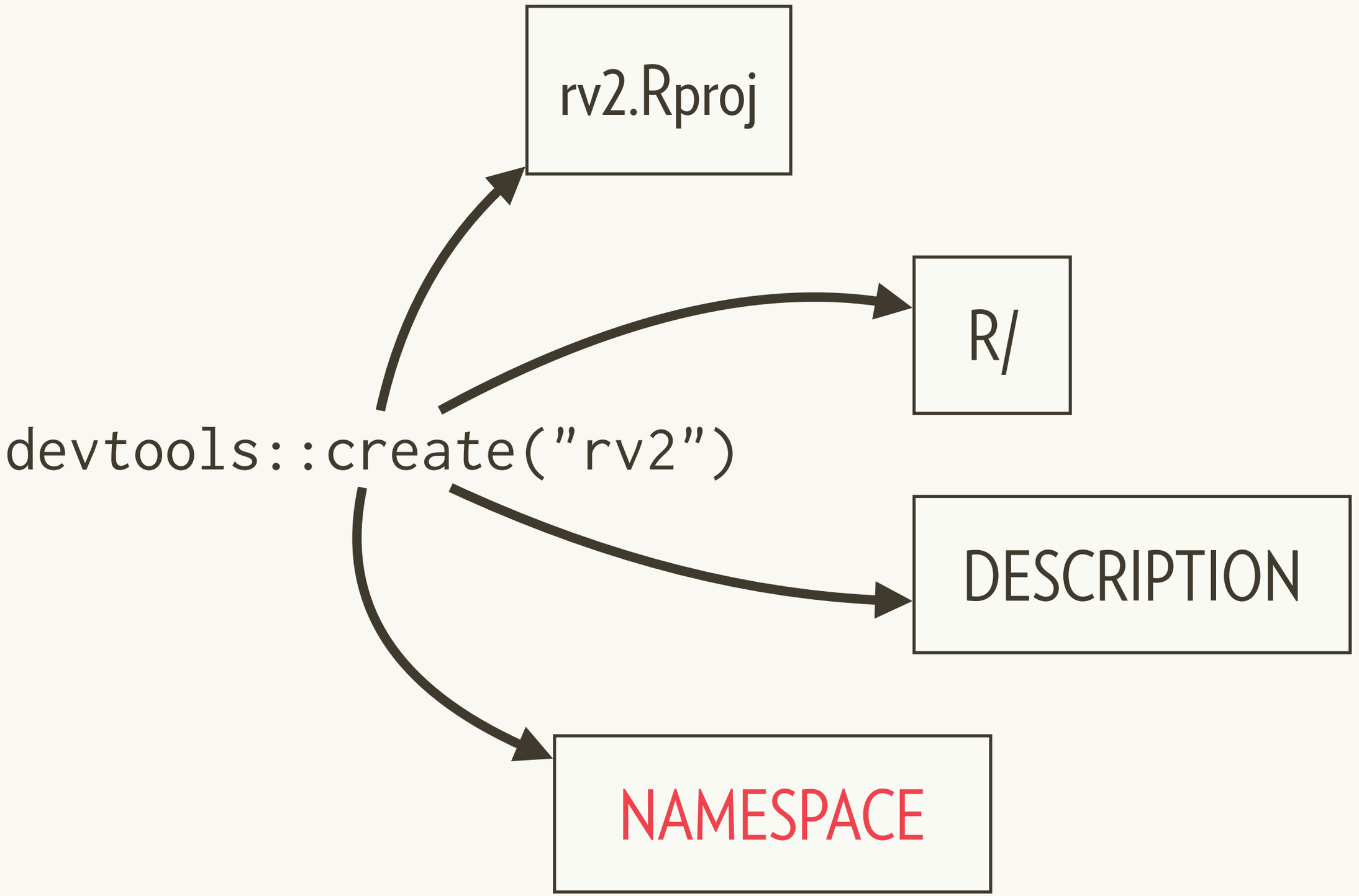
NAMESPACE

January 2017

Hadley Wickham
@hadleywickham
Chief Scientist, **RStudio**

What happens we run create?



Motivation

What happens if two packages use the same name?

```
library(plyr)
library(Hmisc)
is.discrete
```

```
library(Hmisc)
library(plyr)
is.discrete
```

```
Hmisc::is.discrete
plyr::is.discrete
```

User needs to specify which function to use. Can't solve automatically.

What happens if you override a function?

```
nrow
dim <- function(x) c(1, 1)
```

```
dim(mtcars)
nrow(mtcars)
```

Inside a function, a name always needs to point to the same place. Can solve automatically.

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 peoples code

The default NAMESPACE exports everything

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

Better to export function explicitly

```
# Generated by roxygen2
export(fun1)
export(fun3)
```

Most important if you're
planning on sharing with others

But that's tedious to manage by hand

```
#' @export
fun1 <- function(x, y) {
  ...
}

fun2 <- function(x, y) {
  ...
}

#' @export
fun3 <- function(x, y) {
  ...
}
```

@export generates the right NAMESPACE directive

Object type	Namespace directive
Function	export()
S3 method	S3method()
S4 class	exportClass()
S4 method	exportMethods()

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))]
}
```

Change working directory/project to:

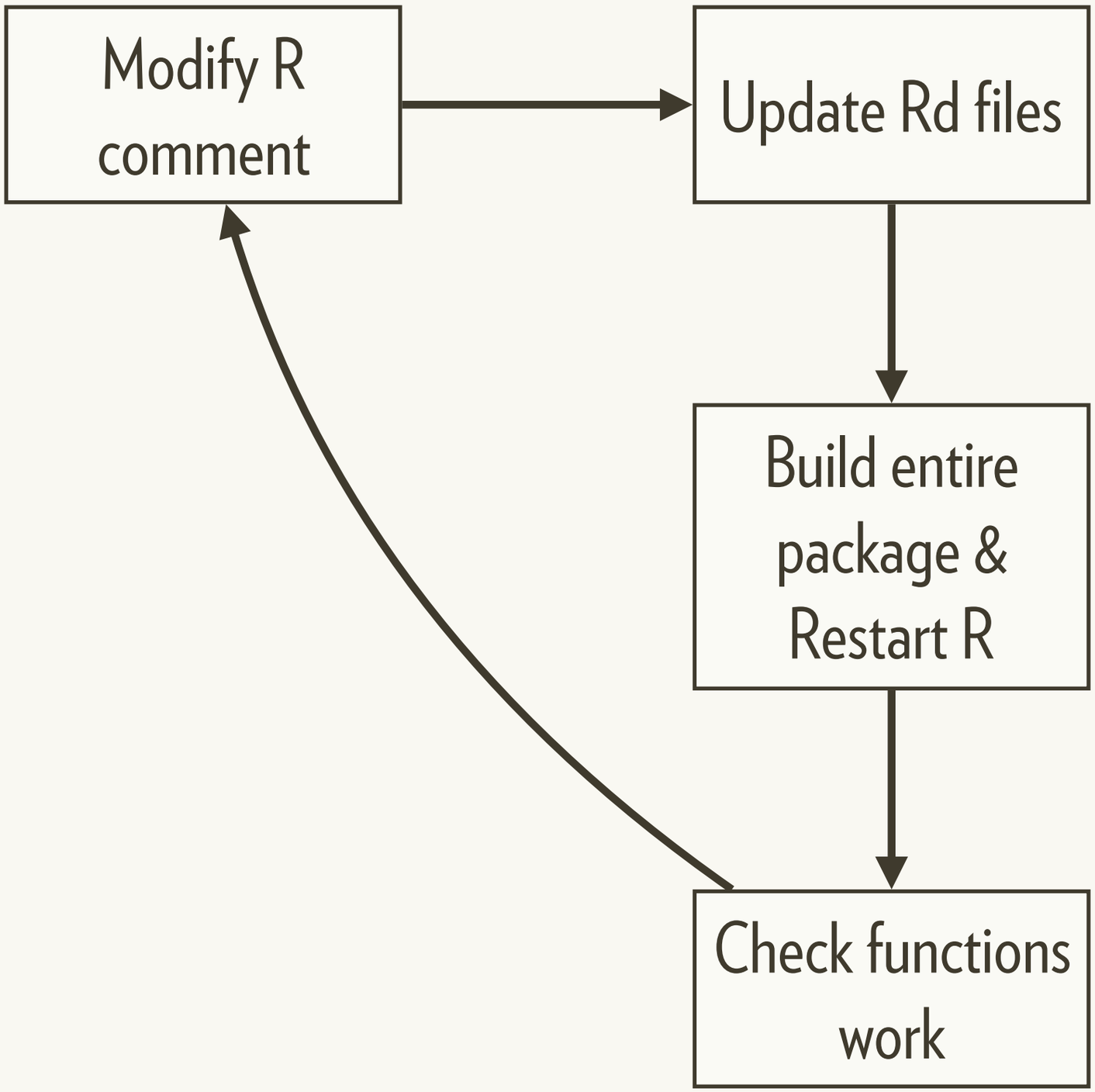
[document-me]

Diagnosing an export failure

```
# I forgot to export two functions.
# Everything looks ok if I use load_all()
devtools::load_all()
rsim(rv(1:6), 100)
Z(rv(1:6))

# But if I build & reload:
library(rv2)
rsim(rv(1:6), 100)
Z(rv(1:6))
```

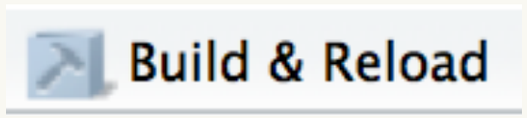
Documentation workflow 2



Cmd/Ctrl + Shift + D

`devtools::document()`

Cmd/Ctrl + Shift + B



We'll learn a robust workflow later

Your turn

Export the two functions I forgot to.

Document, then build & install, and check that this code works:

```
library(rv2)
rsim(rv(1:6), 100)
Z(rv(1:6))
```

Imports

You might get tired of using :: all the time

```r
# Plus you can't call infix functions like %>%
col_summary <- function(df, fun) {
  stopifnot(is.data.frame(df))
  df %>%
    purrr::keep(is.numeric) %>%
    purrr::map(fun) %>%
    as.data.frame()
}
```

You can *import* functions into the package

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

  df %>%
    keep(is.numeric) %>%
    map(fun) %>%
    as.data.frame()
}
```

Alternatively, create R/imports.R

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

Importing everything is easy, but dangerous

```r
#' @import purrr
col_summary <- function(df, fun) {
  stopifnot(is.data.frame(df))
  df %>%
    keep(is.numeric) %>%
    map(fun) %>%
    as.data.frame()
}
```

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

# Works today
# But next year, bar package adds fun1 function
```

Description	NAMESPACE
Makes sure **package** is installed	Makes sure **function** is available to your code
Mandatory	Optional (can use :: instead)
use_package()	#' @importFrom