Building R Packages with RStudio

Ren-Huai Huang
May 20, 2016
Monsanto R Building

Prepare Your System

Install the latest version of R: R version 3.2.5 (April, 2016)

Update the latest Rstudio

install.packages(c("devtools", "roxygen2", "testthat", "knitr"))

devtools::install_github("hadley/devtools")

devtools::has_devel()

Git

Rtools-CRAN for Windows

Xcode for Mac

r-base-dev for Linux

R Packages Are The Fundamental Unit of R

29 packages ship with all binary distributions of R

- 14 base packages : what you think of as "base R"

- recommended packages installed

CRAN + bioconductor : > 10 000 more packages

- install.packages(c("devtools", "roxygen2"))

And then there's Github ...

- devtools::install_github("hadley/devtools")

R User → Package Developer

Package is an efficient way to share code

Useful Even If You Never Share Your Code

- ○ Because packages come with standardized conventions
- ○ Organize a project with a standardised tools
- ○ Better to use the available R Package
- ○ Save your time

Philosophy of Package Development

By Hadley Wickham (http://r-pkgs.had.co.nz/intro.html)

- Anything that can be automated, should be automated.

- Do as little as possible by hand.

- Do as much as possible with functions.

- The philosophy is realized primarily through the devtools package.

Basic Structure of R Packages

DESCRIPTION   Describe and setup

R/   R scripts and Rxoygen2 instructions

src/   Source code: C++, C, Fortran, etc.

tests/   Testing to ensure

vignettes/   Long-form guide, R markdown

data/   Exported data object. need documentation

man/   .Rd documentation, the help pages

NAMESPACE   Organize package self-contained

R package is collections of **R** functions, data, and compiled code in a well-defined format

A Minimal R Package

Create a directory and file pkgA/DESCRIPTION

```
Package:        pkgA
    Type:       Package
Title:                  A First Test
Version:        0.0.1
    Date:       2016-5-17
Author:         Dirk Eddelbuettel
Maintainer:     Dirk Eddelbuettel <edd@debian.org>
Description:    pkgA is a minimal package.
License:        GPL (>= 2)
```

```
$ R CMD build ../pkgA
$ R CMD check ./pkgA_0.0.1.tar.gz
$ R CMD INSTALL ./pkgA_0.0.tar.gz
```

Build & Check With OKs

bash-3.2$ R CMD build ../pkgA
* checking for file '../pkgA/DESCRIPTION' ... OK
* preparing 'pkgA':
* checking DESCRIPTION meta-information ... OK
* checking for LF line-endings in source and make files
* checking for empty or unneeded directories
* creating default NAMESPACE file
* building 'pkgA_0.0.1.tar.gz'

Success = NO NOTE
NO WARNING
NO ERROR

bash-3.2$ R CMD check pkgA_0.0.1.tar.gz
* using log directory '/Users/huangrh/Git/pkgA/pkgA.Rcheck'
* using R version 3.2.4 (2016-03-10)
* using platform: x86_64-apple-darwin13.4.0 (64-bit)
* using session charset: UTF-8
* checking for file 'pkgA/DESCRIPTION' ... OK
* checking extension type ... Package
* this is package 'pkgA' version '0.0.1'
* 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 'pkgA' can be installed ... OK
* checking installed package size ... OK
* checking package directory ... 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 whether the package can be loaded ... OK
* checking whether the package can be loaded with stated dependencie
* checking whether the package can be unloaded cleanly ... OK
* checking whether the namespace can be loaded with stated dependen
OK
* checking whether the namespace can be unloaded cleanly ... OK
* checking loading without being on the library search path ... OK
* checking examples ... NONE
* checking PDF version of manual ... OK
* DONE

Status: OK

DESCRIPTION : Package Metadata

- Describes the package

- Setup up the package dependencies.

```
1  Package: mypackage
2  Type: Package
3  Title: What the Package Does (Title Case)
4  Version: 0.1.0
5  Authors@R: person("First", "Last", email =
6  "first.last@example.com", role = c("aut", "cre"))
7  Description: More about what it does
8            (maybe more than one line)
9  License: What license is it under?
10 LazyData: TRUE
11
12 Depends:  R (>= 3.1.0)
13 Imports:  Rcpp, dplyr (>= 0.4.0)
14 Suggests: Knitr (>= 0.1.0)
15 LinkingTo: Rcpp
16 Enhances: sp
```

DESCRIPTION : Package Metadata

- Describes the package

- Setup up the package dependencies.

```
1  Package: mypackage
2  Type: Package
3  Title: What the Pack
4  Version: 0.1.0
5  Authors@R: person("F
6  "first.last@example.
7  Description: More ab
8            (maybe m
9  License: What licens
10 LazyData: TRUE
11
12 Depends:  R (>= 3.1.
13 Imports:  Rcpp, dply
14 Suggests: Knitr (>=
15 LinkingTo: Rcpp
16 Enhances: sp
```

CRAN - Package ggplot2

ggplot2: An Implementation of the Grammar of Graphics

An implementation of the grammar of graphics in R. It combines the advantages of both base and lattice graphics: conditioning and shared axes are handled automatically, and you can still build up a plot step by step from multiple data sources. It also implements a sophisticated multidimensional conditioning system and a consistent interface to map data to aesthetic attributes. See http://ggplot2.org for more information, documentation and examples.

Version:	2.1.0
Depends:	R (≥ 3.1)
Imports:	digest, grid, gtable (≥ 0.1.1), MASS, plyr (≥ 1.7.1), reshape2, scales (≥ 0.3.0), stats
Suggests:	covr, ggplot2movies, hexbin, Hmisc, lattice, mapproj, maps, maptools, mgcv, multcomp, nlme, testthat (≥ 0.11.0), quantreg, knitr, rpart, rmarkdown, svglite
Enhances:	sp
Published:	2016-03-01
Author:	Hadley Wickham [aut, cre], Winston Chang [aut], RStudio [cph]
Maintainer:	Hadley Wickham <hadley at rstudio.com>

DESCRIPTION : Package Metadata

- Describes the package

- Setup up the package dependencies.

```
1  Package: mypackage
2  Type: Package
3  Title: What the Package Does (Ti
4  Version: 0.1.0
5  Authors@R: person("First", "Last
6  "first.last@example.com", role =
7  Description: More about what it
8             (maybe more than one
9  License: What license is it unde
10 LazyData: TRUE
11
12 Depends:  R (>= 3.1.0)
13 Imports:  Rcpp, dplyr (>= 0.4.0)
14 Suggests: Knitr (>= 0.1.0)
15 LinkingTo: Rcpp
16 Enhances: sp
```

Work With Other Package:

- Depends: if the other packages is expect to be attached and the functions can be accessed directly
- Imports: packages must have. R will install them when it installs your pkg.
- Suggests: packages are not essential. Users can install them manually, or not, as they like
- LinkingTo: the header files from another package, e.g., Rcpp

Helpers For Creating Package

package.skeleton() main worker, has warts.

Fail with R CMD check

Not recommend

pkgKitten::kitten() corrects issues with package.skeleton()
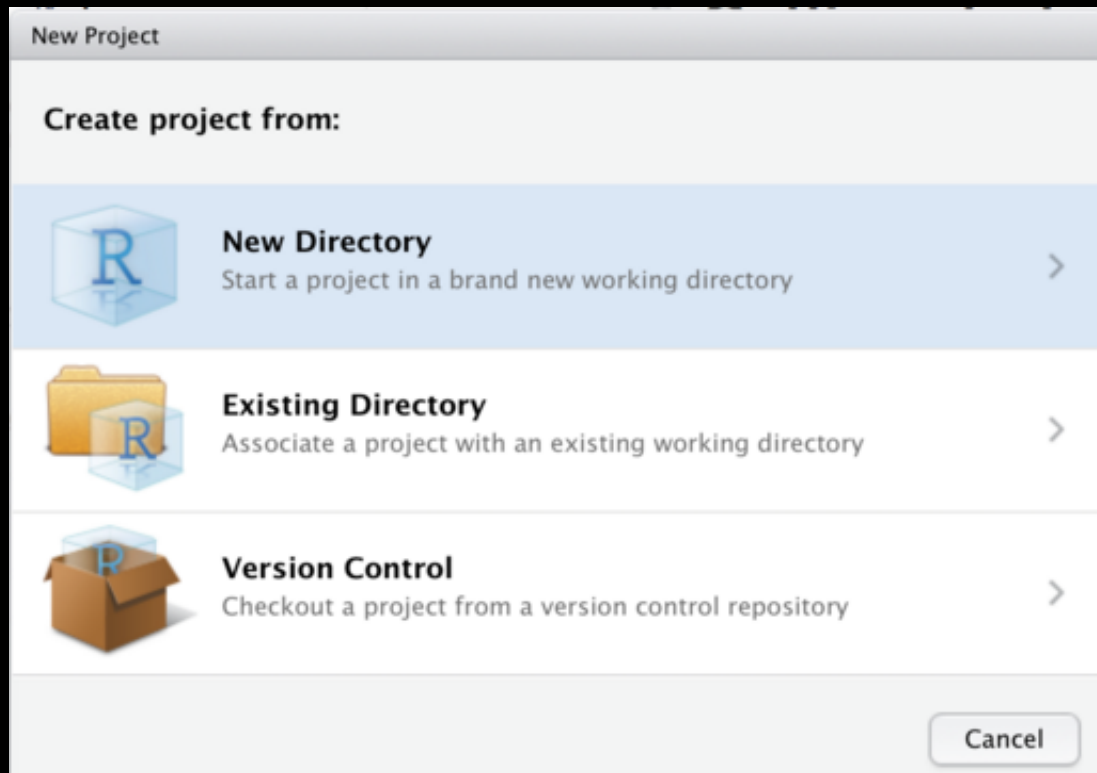
Success with R CMD check

devtools::create() In active development.

By Hadley Wickham
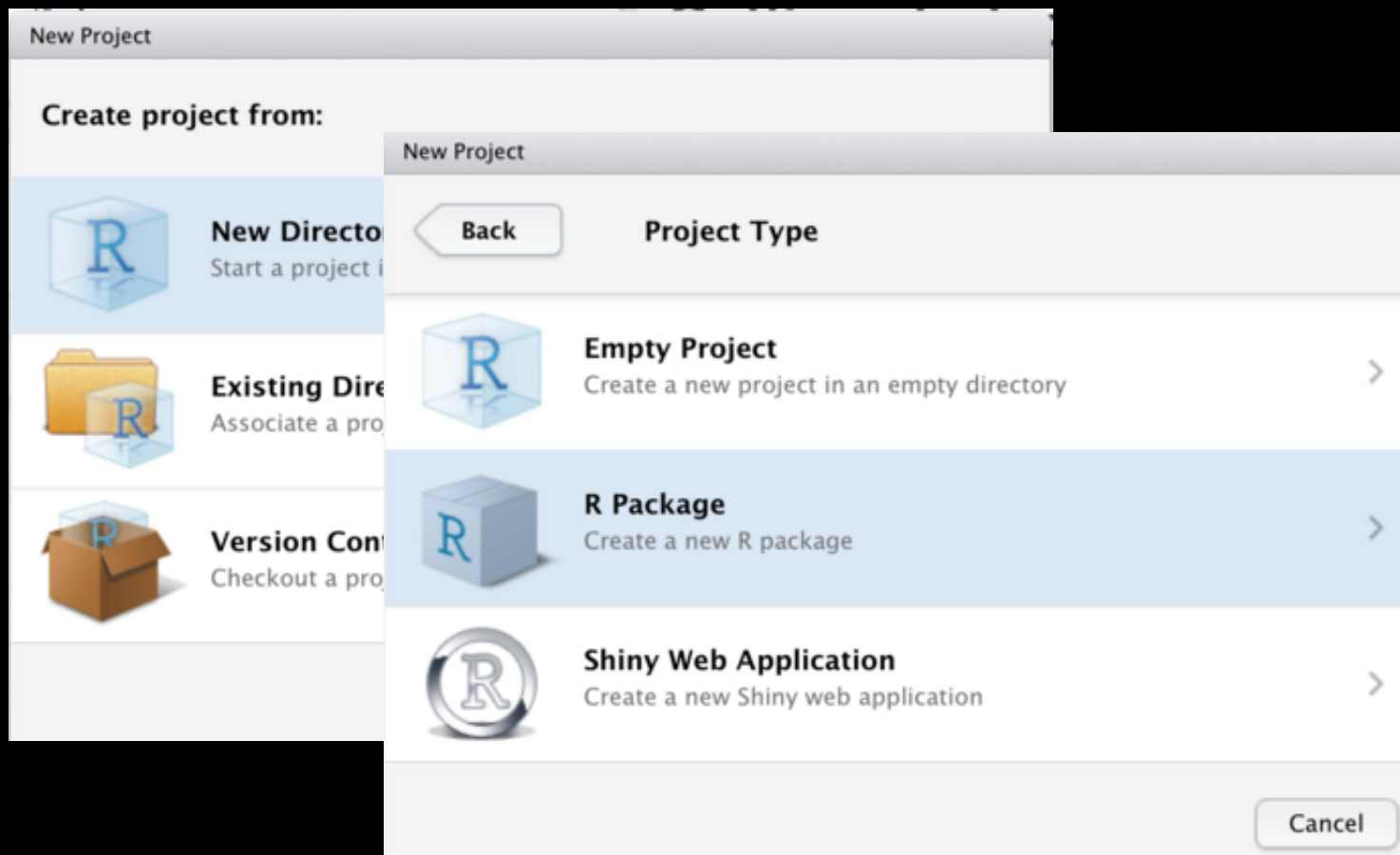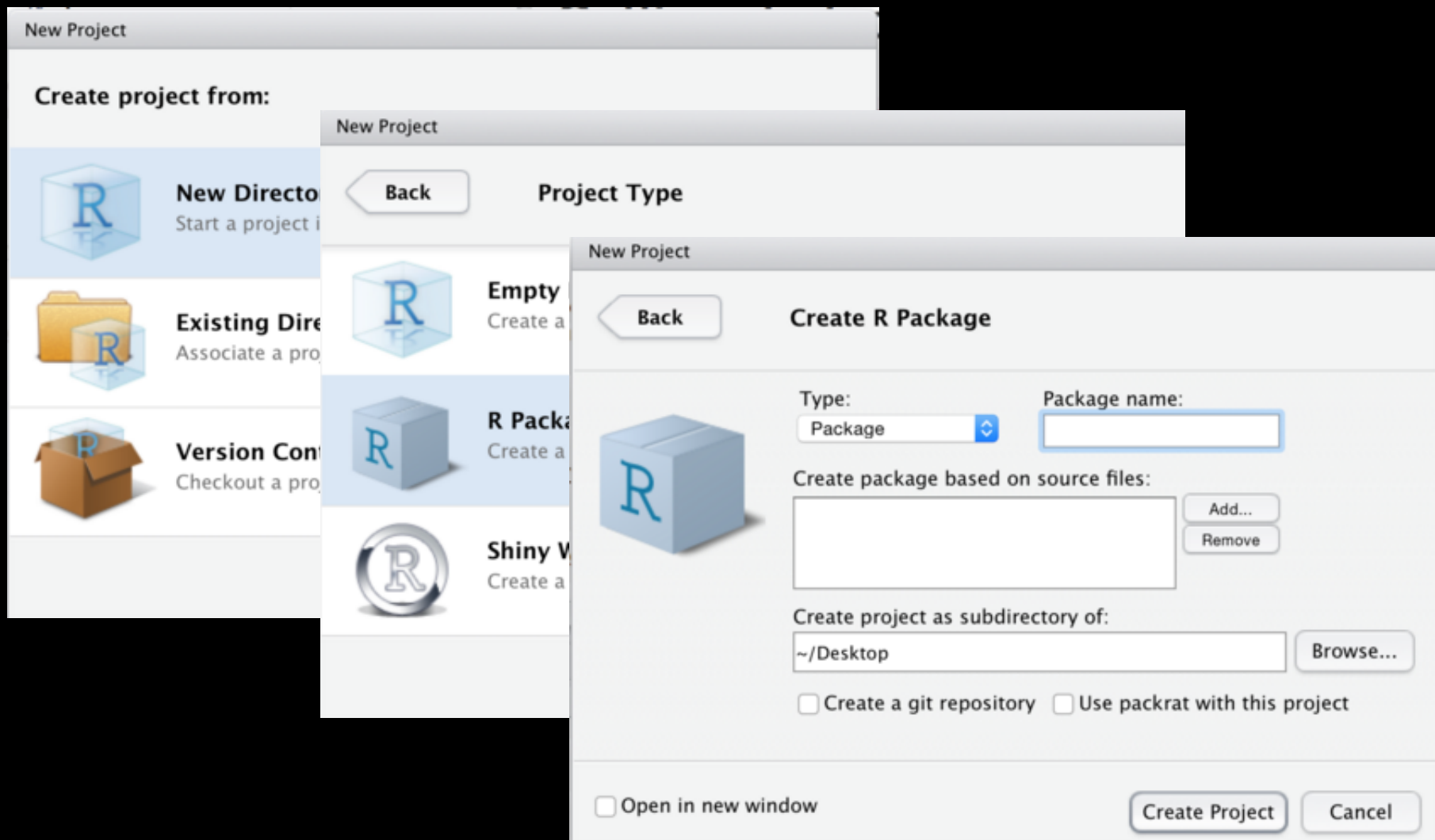
Integrated friendly with Rstudio

Create A Source Package

Rstudio: Click File | New Project.

Create A Source Package

Rstudio: Click File | New Project.

Create A Source Package

Rstudio: Click File | New Project.

Create A Source Package

Rstudio: Click File | New Project.

```
$ tree pkgA

pkgA
├── DESCRIPTION
├── NAMESPACE
├── R
│   └── hello.R
├── man
│   └── hello.Rd
└── pkgA.Rproj

2 directories
5 files
```

Create A Source Package

Rstudio: Click File | New Project.

$ tree pkgA

```
pkgA
├── DESCRIPTION
├── NAMESPACE
├── R
│   └── hello.R
├── man
│   └── hello.Rd
└── pkgA.Rproj

2 directories
5 files
```

devtools::create("pkgB")

$ tree pkgA

```
pkgB
├── DESCRIPTION
├── NAMESPACE
├── R
└── pkgB.Rproj

1 directory
3 files
```

Package Development Stages

- In your working directory

  - source   - a directory with subdirectories (as above)
  - bundle   - a single compressed file (.tar.gz)
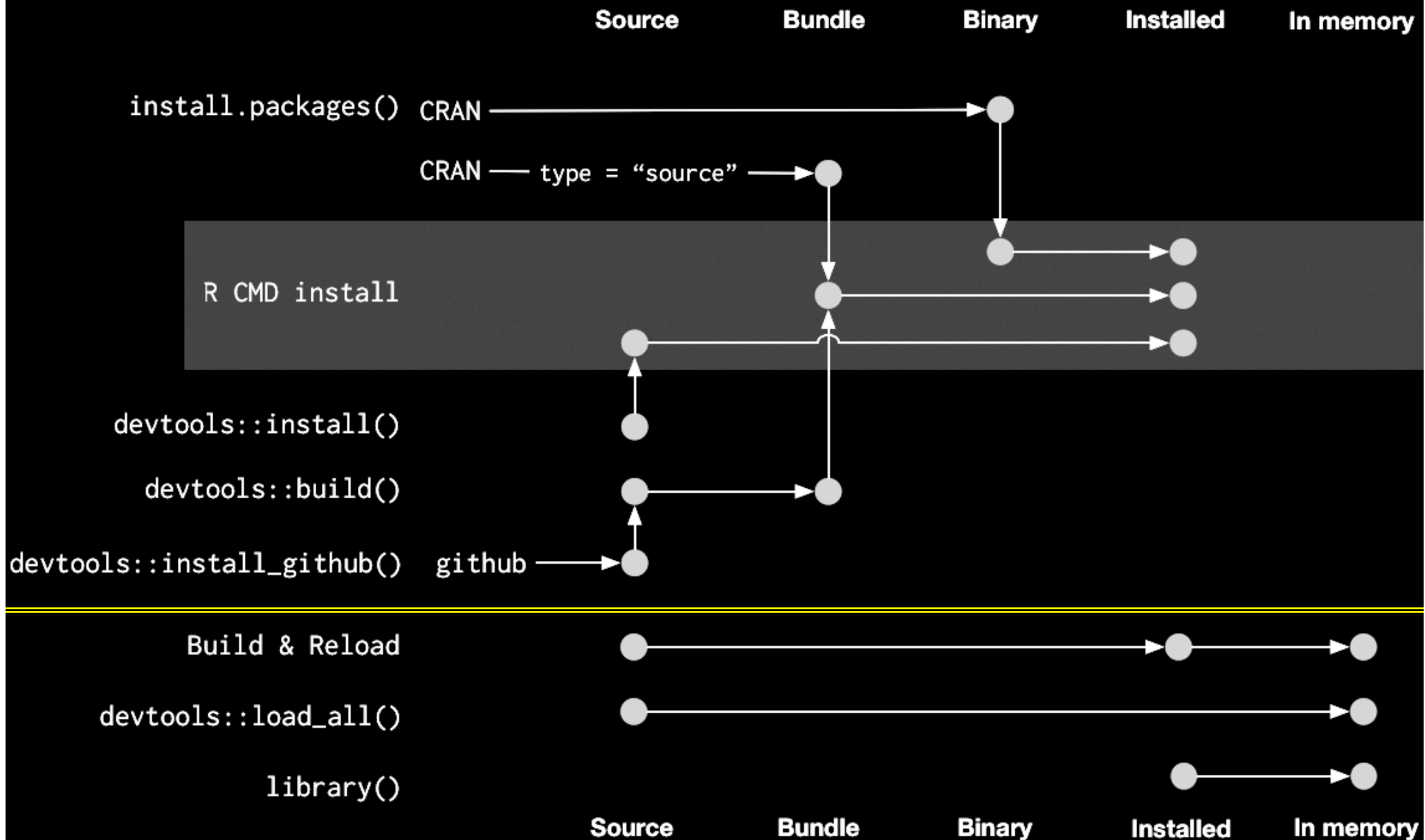 - binary   - a single compressed file for a specific OS

- In library

```
> .libPaths()
[1] "/Library/Frameworks/R.framework/Versions/3.2/Resources/library"
> library()
```

- In memory

  - Loading: requireNamespace(x)   loadNamespace("x")

  - Attaching: require(x)   library(x)

Package Development Stages

- - - From Hadley

R Coding Workflow

1. Edit an R code in R file

2. devtools::load_all()

3. Test the function in the console

4. Rinse and repeat

Workflow Example

```
 1  #' Hello, world!
 2  #'
 3  #' This is an example function named
 4  #' 'hello' which prints 'Hello, world!'.
 5  #'
 6  #' You can learn more about package
 7  #' authoring with RStudio at:
 8  #'
 9  #' http://r-pkgs.had.co.nz/
10  #'
11  #' Some useful keyboard shortcuts for
12  #' package authoring:
13  #'
14  #' Build and Reload Package:'Cmd+Shift+B'
15  #' Check Package:           'Cmd+Shift+E'
16  #' Test Package:            'Cmd+Shift+T'
17  hello <- function() {
18    print("Hello, world!")
19  }
20
```

```
> devtools::load_all()
Loading testPackage
> hello
function() {
  print("Hello, world!")
}
<environment: namespace:testPackage>
>
```

Rule : Preserve R Landscape

Avoid changing the global settings

- library(), require()

- source()

- setwd()

- options()

Suggest to use

- pkg::function()

- requireNamespace(), loadNamespace()

- NAMESPACE

Roxygen2 Documentation

- Advantages over writing .Rd files:

  ○ Code and documentation are together

  ○ It dynamically inspects the objects that it documents

  ○ It abstracts over the differences in documenting different types of objects.

- Three main components/categories:

  ○ Generate .Rd files

  ○ Manage NAMESPACE

  ○ Control the collation order

Common Roxygen Tags:

● Function

@param
@return
@examples
@inheritsParams

● Basic

@title
@describtion
@detail
@section

● NAMESPACE

@export
@import
@importFrom
@importClassesFrom
@importMethodsFrom
@useDynLib

● Navigation

@family
@seealso

● help.search

@aliases, ?
@concept
@keywords

● Data

@format
@source

● S4

@include
@slot

● RC

@field

● Together

@describeIn
@rdname

Roxygen Workflow:

```
> file.create("R/adder.R")
[1] TRUE
```

```
 1  #' Add Two Numbers.
 2  #'
 3  #' This is the description.
 4  #'
 5  #' @section A Custom Section:
 6  #' Text accompanying the custom section.
 7  #' @param x,y A number.
 8  #' @return The sum of \code{x} and \code{y}
 9  #' @examples
10  #' add_numbers(1, 2) ## returns 3
11  add <- function(x, y) {
12      x + y
13  }
```

Roxygen Workflow:

```
> file.create("R/adder.R")
[1] TRUE

> devtools::document()
Updating mypackage documentation
Loading mypackage
First time using roxygen2 4.0. Upg
y...
```

```
 3  \name{add}
 4  \alias{add}
 5  \title{Add Two Numbers.}
 6  \usage{
 7  add(x, y)
 8  }
 9  \arguments{
10  \item{x, y}{A number.}
11  }
12  \value{
13  The sum of \code{x} and \code{y}
14  }
15  \description{
16  This is the description.
17  }
18  \section{A Custom Section}{
19
20  Text accompanying the custom section.
21  }
22  \examples{
23  add_numbers(1, 2) ## returns 3
24  }
```

Roxygen Workflow:

```
> file.create("R/adder.R")
[1] TRUE

> devtools::document()
Updating mypackage documentation
Loading mypackage
First time using roxygen2 4.0. Upgrading a
y...
> library(pkgA)
> ?add
```

Add Two Numbers.

Description

This is the description.

Usage

```
add(x, y)
```

Arguments

x, y A number.

Value

The sum of x and y

A Custom Section

Text accompanying the custom section.

Examples

```
add_numbers(1, 2) ## returns 3
```

Roxygen Workflow:

```
> file.create("R/adder.R")
[1] TRUE
```

```
> devtools::document()
Updating mypackage documentation
Loading mypackage
First time using roxygen2 4.0. Upgrading automaticall
y...
```
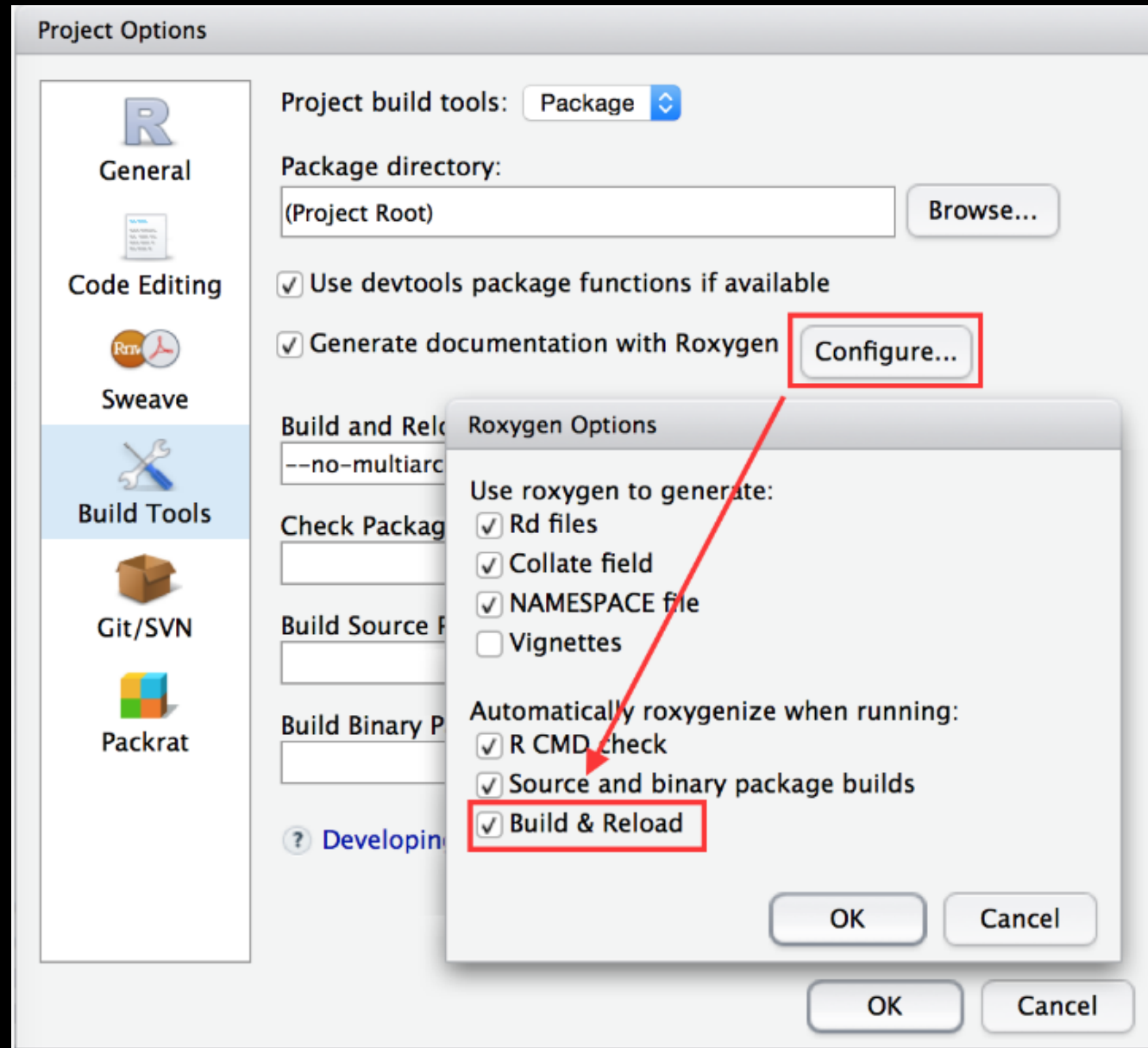
```
> library(pkgA)
> ?add
```

```
> example("add")

add> add(1, 1)
[1] 2

add> add(10, 1)
[1] 11
```

Alternative Workflow

Alternative Workflow

Formal Test

```
> require(testthat)
Loading required package: testthat
> devtools::use_testthat()
* Adding testthat to Suggests
* Creating `tests/testthat`.
* Creating `tests/testthat.R` from template.
`
> devtools::use_test("math")
* Creating `tests/testthat/test-math.R` from template.
* Modify `tests/testthat/test-math.R`.
```

Formal Test

```
> require(testthat)
Loading required package: testthat
> devtools::use_testthat()
* Adding testthat to Suggests
* Creating `tests/testthat`.
* Creating `tests/testthat.R` from template.
> devtools::use_test("math")
* Creating `tests/testthat/test-math.R` from template.
* Modify `tests/testthat/test-math.R`.
```

```r
1  context("math")
2
3  ## TODO: Rename context
4  ## TODO: Add more tests
5
6  test_that("multiplication works", {
7    expect_equal(2 * 2, 4)
8  })
```

Formal Test

```
> devtools::test()
Loading mypackage
Testing mypackage
math: .

DONE ======================================================================
```

```
1  context("math")
2
3  ## TODO: Rename context
4  ## TODO: Add more tests
5
6  test_that("multiplication works", {
7      expect_equal(2 * 2, 4)
8  })
```

Formal Test

```
> devtools::test()
Loading mypackage
Testing mypackage
math2: .1

Failed ------------------------------------------------
1. Failure: 2. multiplication works (@test-math.R#8) --
2 * 2 not equal to 5.
1/1 mismatches
[1] 4 - 5 == -1


DONE ==================================================
```

```
1  context("math2")
2
3  test_that("1. multiplication works", {
4    expect_equal(2 * 2, 4)
5  })
6
7  test_that("2. multiplication works", {
8    expect_equal(2 * 2, 5)
9  })
```

Formal Test

```
> devtools::test()
Loading mypackage
Testing mypackage
math2: 1.

Failed -------------------------------------------------------
1. Failure: 1. multiplication works (@test-math.R#4) --
2 * 2 not equal to 5.
1/1 mismatches
[1] 4 - 5 == -1



DONE ===========================================================
```

```
1  context("math2")
2
3  test_that("1. multiplication works", {
4      expect_equal(2 * 2, 5)
5  })
6
7  test_that("2. multiplication works", {
8      expect_equal(2 * 2, 4)
9  })
```

Common Test Functions

expect_equal()	is equal within small numerical tolerance?
expect_identical()	is exactly equal?
expect_match()	match specified string/regular expression?
expect_output()	prints specified output?
expect_message()	displays specified message?
expect_warning()	displays specified warning?
expect_error()	throws specified error?
expect_is()	output inherits from certain class?
expect_false()	return FALSE?
expect_true()	return TRUE?

Interface To Compiled Languages : C++

●R is an extensible language

●Rcpp: add-on package which facilitates extending R with C++

●Rcpp offer s matching C++ classes for basic R data types

●The mapping data type works in both direction:

● It is straightforward to pass data from R to C++ and return data from C++ to R

●vignette(package="Rcpp")

Setup To Work With C++

```
> devtools::use_rcpp()
● Adding Rcpp to LinkingTo and Imports
● * Creating `src/`.
● * Ignoring generated binary files.
● Next, include the following roxygen tags somewhere in your
  package:

  #' @useDynLib mypackage
  #' @importFrom Rcpp sourceCpp
  NULL
```

● Set up a .gitignore file to ignore any compiled files

```
> devtools::use_package_doc()
* Creating `R/mypackage-package.r` from template.
* Modify `R/mypackage-package.r`.
1  #' mypackage.
2  #'
3  #' @name mypackage
4  #' @docType package
5  #' @useDynLib mypackage
6  #' @importFrom Rcpp sourceCpp
7  NULL
```
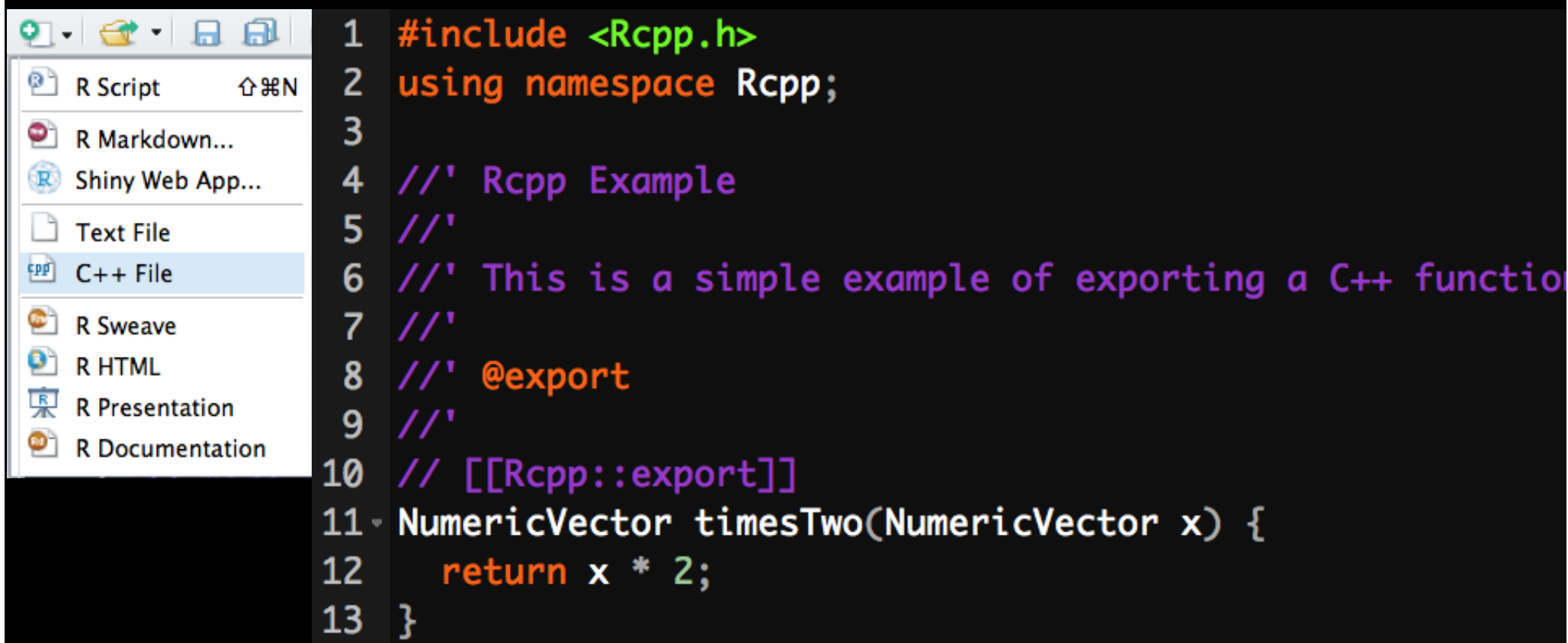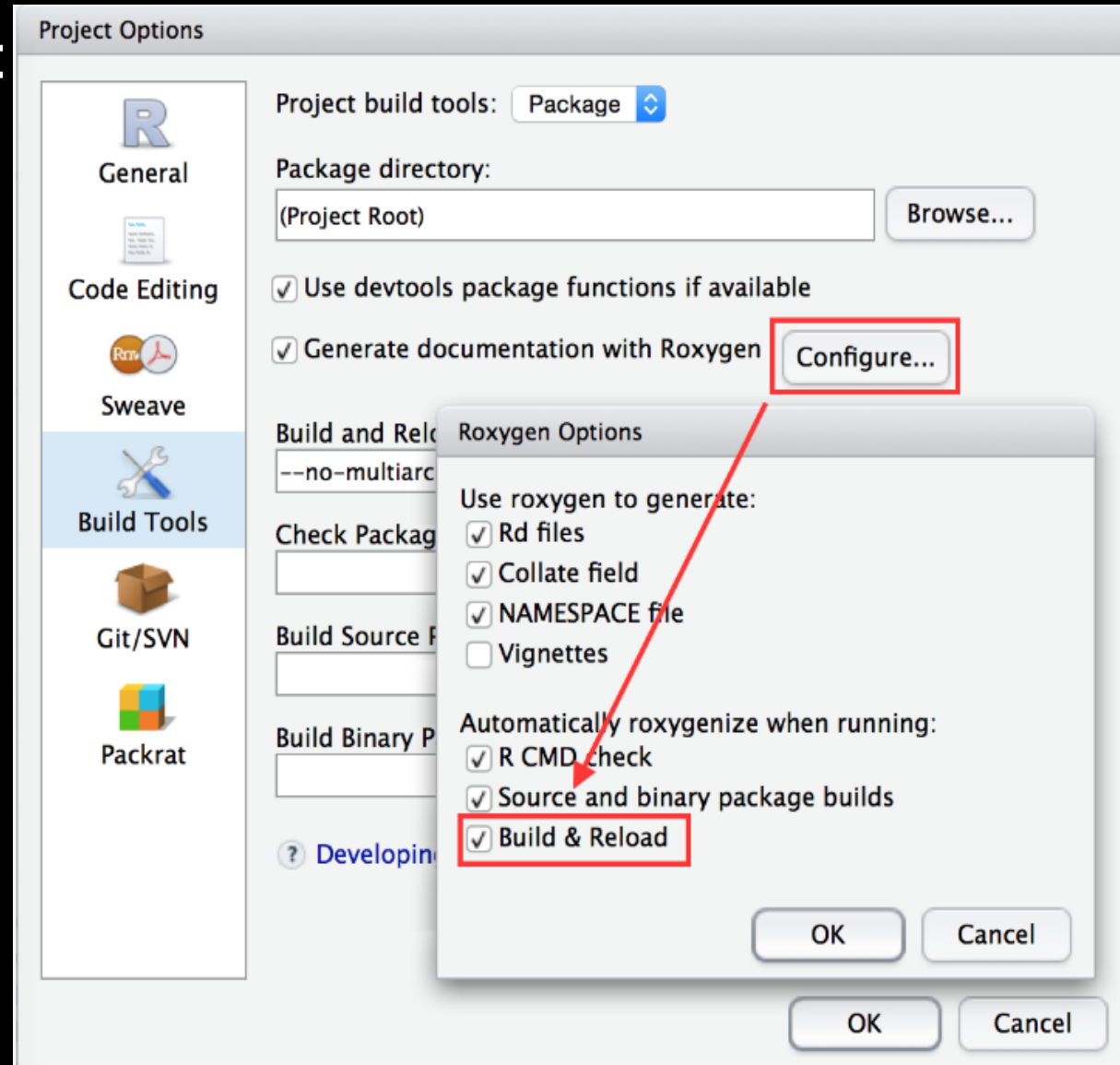
Rcpp Workflow

1. Create a new C++ file:

Rstudio | File | New File | C++ File


```cpp
1  #include <Rcpp.h>
2  using namespace Rcpp;
3
4  //' Rcpp Example
5  //'
6  //' This is a simple example of exporting a C++ function
7  //'
8  //' @export
9  //'
10 // [[Rcpp::export]]
11 NumericVector timesTwo(NumericVector x) {
12    return x * 2;
13 }
```

Rcpp Workflow

1. Create a new C++ file:

2. Click Build & Reload
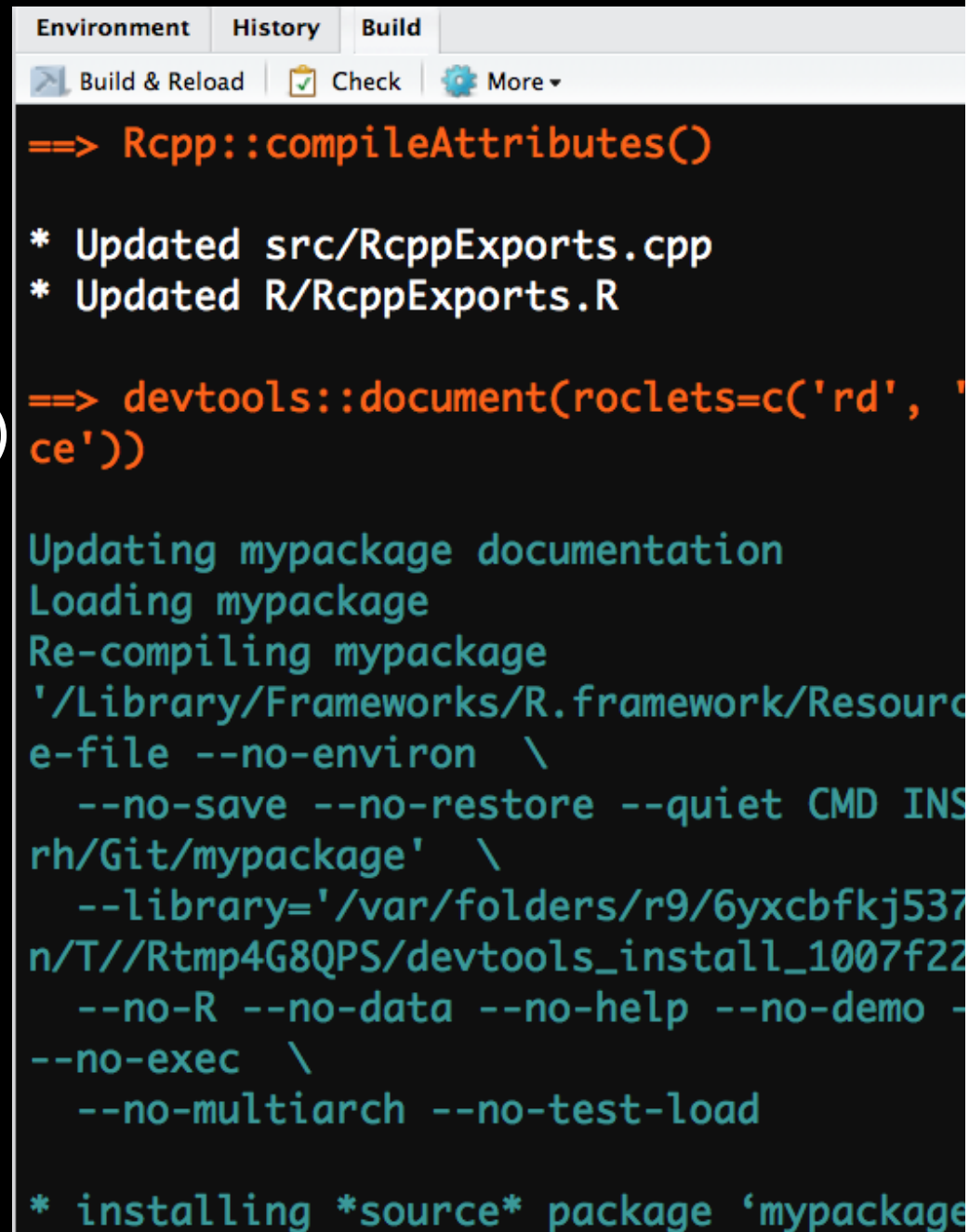
Rcpp Workflow

1. Create a new C++ file:

2. Click Build & Reload:

    a. Set up the environments

    b. Rcpp::compileAttributes()

    c. ....


```
Environment    History    Build

  Build & Reload    Check    More

==> Rcpp::compileAttributes()

* Updated src/RcppExports.cpp
* Updated R/RcppExports.R

==> devtools::document(roclets=c('rd', '
ce'))

Updating mypackage documentation
Loading mypackage
Re-compiling mypackage
'/Library/Frameworks/R.framework/Resourc
e-file --no-environ  \
  --no-save --no-restore --quiet CMD INS
rh/Git/mypackage'  \
  --library='/var/folders/r9/6yxcbfkj537
n/T//Rtmp4G8QPS/devtools_install_1007f22
  --no-R --no-data --no-help --no-demo -
--no-exec  \
  --no-multiarch --no-test-load

* installing *source* package 'mypackage
```

Rcpp Workflow

1. Create a new C++ file:

2. Click Build & Reload:

   a. Set up the environments

   b. Rcpp::compileAttributes()

   c.…….

```
> mypackage:::timesTwo
function(x) {
    .Call('mypackage_timesTwo', PACKAGE = 'mypackage', x)
}
<environment: namespace:mypackage>
> mypackage:::timesTwo(10)
[1] 20
```

Rcpp Roxygen2 Comments

```
 1  #include <Rcpp.h>
 2  using namespace Rcpp;
 3
 4  //' Rcpp Example
 5  //'
 6  //' This is a simple example of exporting a C++ function to R.
 7  //'
 8  //' @export
 9  //'
10  // [[Rcpp::export]]
11  NumericVector timesTwo(NumericVector x) {
12    return x * 2;
13  }
```

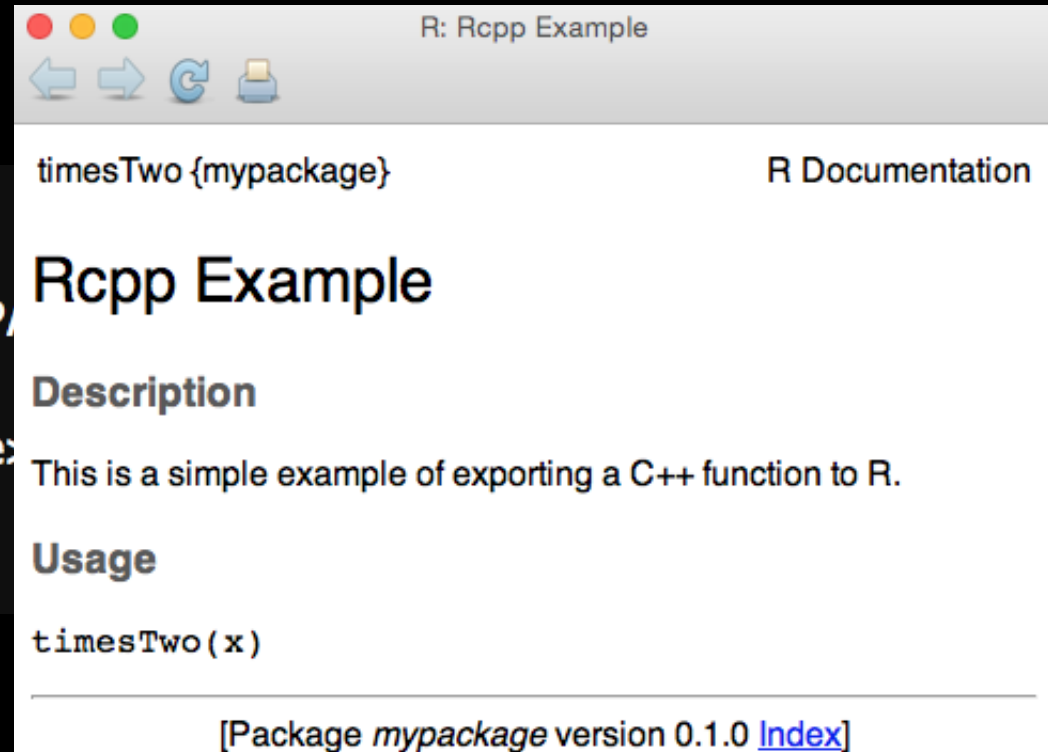# Rcpp Roxygen2 Comments

1. Create a new C++ file:

2. Click Build & Reload:

   a. Set up the environments

   b. Rcpp::compileAttributes()

   c. …….

```
> mypackage:::timesTwo
function(x) {
    .Call('mypackage_timesTwo', PA
}
<environment: namespace:mypackage
> mypackage:::timesTwo(10)
[1] 20

> ?timesTwo
```


timesTwo {mypackage}   R Documentation

Rcpp Example

Description

This is a simple example of exporting a C++ function to R.

Usage

```
timesTwo(x)
```

[Package *mypackage* version 0.1.0 Index]

Summary

devtools::create()

devtools::load_all()

devtools::document()

devtools::use_package_doc()

devtools::use_rcpp()

devtools::use_testthat()

devtools::use_test()

devtools::test()

devtools::use_package()

Topic Not Covered

```
> devtools::use_vignette("pkga-overview")
* Creating `vignettes`.
* Adding `inst/doc` to ./.gitignore
> devtools::use_data(mtcars)
Saving mtcars as mtcars.rda to /Users/huangrh/MyGit/pkgc/data
> devtools::use_data(mtcars,internal=TRUE)
Saving mtcars as sysdata.rda to /Users/huangrh/MyGit/pkgc/R
> devtools::use_data_raw()
* Creating `data-raw`.
* Adding `data-raw` to `.Rbuildignore`.
Next:
* Add data creation scripts in data-raw
* Use devtools::use_data() to add data to package
> devtools::use_git()
* Initialising repo
* Adding `.Rproj.user`, `.Rhistory`, `.RData` to ./.gitignore
* Adding files and committing
> devtools::check()
Updating pkgc documentation
Loading pkgc
> devtools::release()
> devtools::dev_mode()
Dev mode: ON
d>
```

References & Further Resources

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

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

"Writing R Extensions" https://cran.r-project.org/

Software For Data Analysis: Programming With R, John M. Chambers.

www.rcpp.org: Seemless R and C++ Intergration Using Rcpp

https://github.com/hadley/devtools

vignette("roxygen2", package="roxygen2")

https://github.com/jtleek/rpackages/ easy to follow.

Thank You!