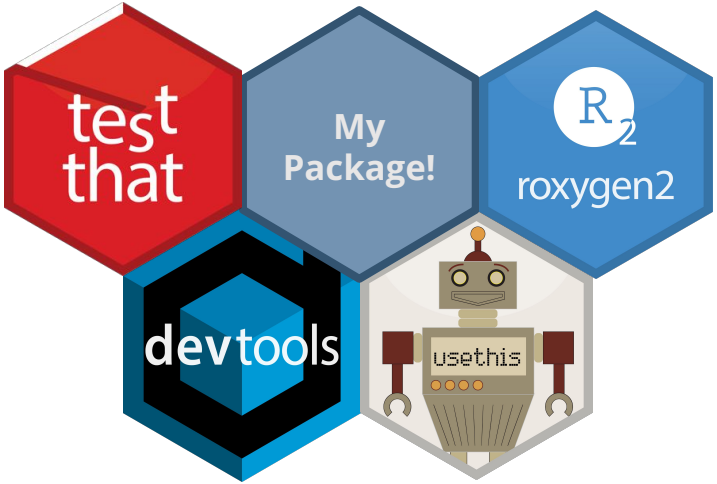
posit™

Create and Share R Packages

Ryan Johnson



Ryan Johnson

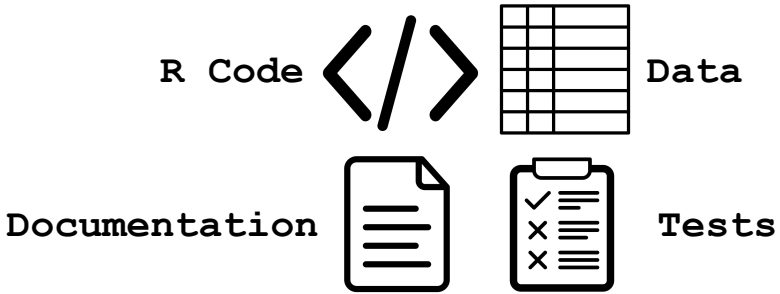
ryan@posit.co

Data Science Advisor

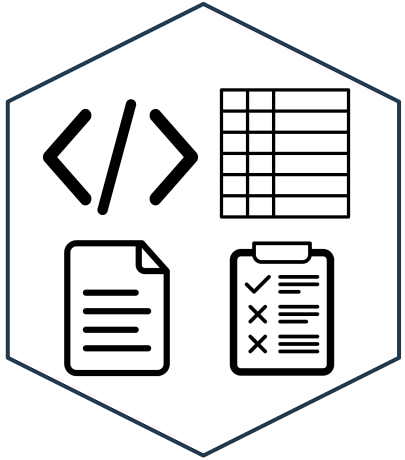
@ryjohnson09

1. **Questions?** Pop them in the chat (Zoom / MS Teams).
2. Keep it **fun**, keep it **casual**, keep it **safe**!
3. Slides, material, and recording will be made available after.

In R, the fundamental unit of shareable code is the package.



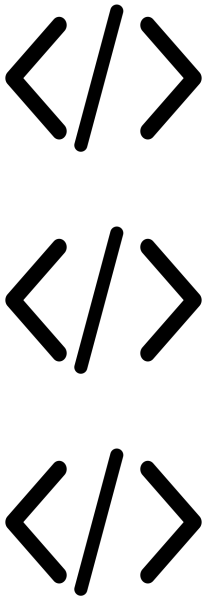
In R, the fundamental unit of shareable code is the package.



rmarkdown
rmarkdown.rstudio.com
Shiny
www.rstudio.com
reticulate
rstudio.com
devtools
www.rstudio.com
plumber
rplumber.io
pins
www.rstudio.com
www.rstudio.com
usethis
rstudio.com
ggplot2
www.rstudio.com
dplyr
dplyr.tidyverse.org

posit™

Why create a package?

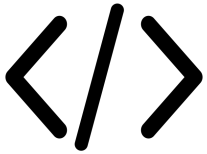


“Consider writing a function whenever you’ve copied and pasted a block of code more than twice (i.e. you now have three copies of the same code).”

- R4DS (2e)

Why create a package?

my_function <- function(){



}

If you:

- Want to **reuse my_function** later in a project or another project
- **Share my_function** with others

Why create a package?

Advantages

- Makes your code more reproducible, organized, and efficient
- Prevents errors when duplicating code
- Gives you the opportunity to document your functions

Disadvantages

- None

All Packages start with a **function**

Example Function

```
cat_function <- function(love=TRUE){
    if(love==TRUE){
        print("I love cats!")
    } else {
        print("I am not a cool person.")
    }
}
```

Example function

```
> cat_function(love = TRUE)
[1] "I love cats!"
>
> cat_function(love = FALSE)
[1] "I am not a cool person :("
```

Goals for Today

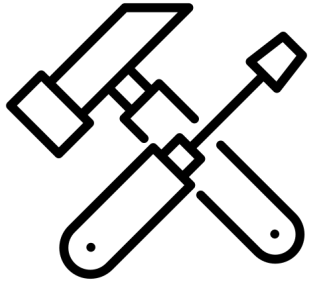
- Walk through (**together**) the entire package development process!
- Keep it high level
  - Go more in-depth here!
- Discuss options for sharing your R package

https://posit.cloud/

posit™ Cloud

1. R Package Development Space
2. Open the **mypackage** project

Step 1: Load your package development tools



```
library(devtools)
#> Loading required package: usethis
```

Step 2: Initialize a new package

Don't run this command - I've created a Posit Cloud environment for you!

```
usethis::create_package("~/mypackage")
```

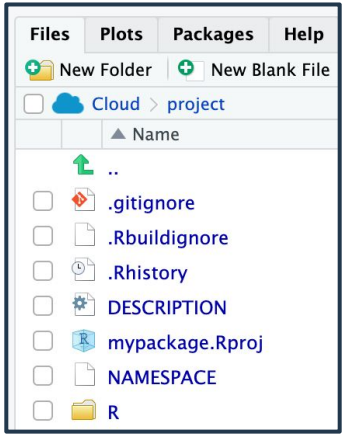
Make a deliberate choice about **where** to create this package on your system:

- **Good**:
  - Within your home directory
  - Next to your other R projects
- **Bad**:
  - Nested within another RStudio project, R package, R package library or GitHub Repository

Step 2: Initialize a new package

Don't run this command - I've created a Posit Cloud environment for you!

```
usethis::create_package("~/mypackage")
```



Step 3: Write the first function

Where do we place our functions?

```
usethis::use_r("cat_function")
#> • Edit 'R/cat_function.R'
#> • Call `use_test()` to create
a matching test file
```

To start, place all user facing functions in their own .R script within the R directory

Step 4: Write the first function

Where do we place our functions?

cat_function.R

```r
cat_function <- function(love){
  if (love == TRUE){
    print("I love cats!")
  } else {
    print("I am not a cool person :(")
  }
}

```

RStudio pro-tip: type “fun” and hit enter to insert a function() skeleton

Step 5: Test drive **cat_function()**

```
devtools::load_all()
#> ℹ Loading mypackage
```

Note: `load_all()` has made the `cat_function()` function available, although it does not exist in the global environment.

`load_all() != library()`

Step 6: Check that all is well with your package

`devtools::check()`

Can also run `R CMD check` in the terminal

```
── R CMD check results ──────────────────────────────────────────
mypackage 0.0.0.9000 ────
Duration: 3.2s

❯ checking DESCRIPTION meta-information ... WARNING
  Non-standard license specification:
    `use_mit_license()`, `use_gpl3_license()` or friends to pick a
    license
  Standardizable: FALSE

0 errors ✔ | 1 warning ✖ | 0 notes ✔
```

Step 7: Add a License

Q: Which license should I use?

A: [Pick a license...any license!]

```
usethis::use_mit_license()
#> ✔ Setting License field in
DESCRIPTION to 'MIT + file LICENSE'
#> ✔ Writing 'LICENSE'
#> ✔ Writing 'LICENSE.md'
#> ✔ Adding '^LICENSE\\.md$' to
'.Rbuildignore'
```

Step 8: Edit DESCRIPTION

DESCRIPTION

```
Package: mypackage
Title: What the Package Does (One Line, Title Case)
Version: 0.0.0.9000
Authors@R:
    person("First", "Last", , "first.last@example.com", role = c("aut", "cre"))
Description: What the package does (one paragraph).
License: `use_mit_license()`, `use_gpl3_license()` or friends to pick a
    license
Encoding: UTF-8
Roxygen: list(markdown = TRUE)
RoxygenNote: 7.2.3
```

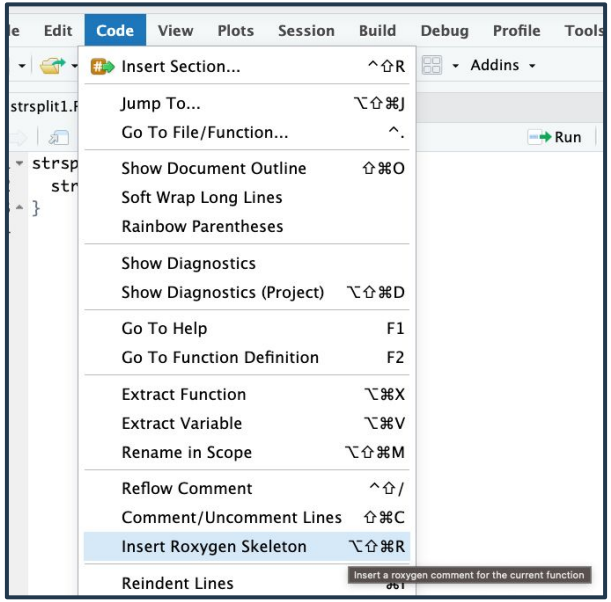
Step 8: Edit DESCRIPTION

```
Package: mypackage
Title: Cat love
Version: 0.0.0.9000
Authors@R:
    person("Jane", "Doe", , "jane@example.com", role = c("aut",
"cre"))
Description: Function to determine if you love cats or not.
License: MIT + file LICENSE
Encoding: UTF-8
Roxygen: list(markdown = TRUE)
RoxygenNote: 7.2.3
```

Step 9: Document

Let users know what your function does and how to use it!

1. Open R/cat_function.R
2. Put the cursor somewhere in the cat_function() function definition
3. Code → Insert roxygen skeleton



Step 9: Document

Let users know what your function does and how to use it!

cat_function.R*

```r
#' Title
#'
#' @param love
#'
#' @return
#' @export
#'
#' @examples
cat_function <- function(love){
  if (love == TRUE){
    print("I love cats!")
  } else {
    print("I am not a cool person :(")
  }
}
```

roxygen2

Step 9: Document

Let users know what your function does and how to use it!

```
#' Cat love function
#'
#' @param love Enter TRUE if you love cats!
#'
#' @return A message
#' @export
#'
#' @examples
#' cat_function(love = TRUE)
cat_function <- function(love=TRUE){
    if(love==TRUE){
        print("I love cats!")
    } else {
        print("I am not a cool person.")
    }
}
```

Step 9: Document

Let users know what your function does and how to use it!

```
devtools::document()
#> ℹ Updating mypackage documentation
#> Setting `RoxygenNote` to "7.2.3"
#> ℹ Loading mypackage
#> Writing 'NAMESPACE'
#> Writing 'cat_function.Rd'
```

Check...again!

```
devtools::check()
```

Can also run `R CMD check` in the terminal

```
── R CMD check results ──────────────────
mypackage 0.0.0.9000 ────
Duration: 9.7s

0 errors ✔ | 0 warnings ✔ | 0 notes ✔
```

Step 10: Install and load your package

RESTART YOUR R SESSION!

```
devtools::install()
```

```
> devtools::install()
── R CMD build ─────────────────────────────────────────────────────────────
✔  checking for file '/cloud/project/DESCRIPTION' ...
─  preparing 'mypackage':
✔  checking DESCRIPTION meta-information ...
─  checking for LF line-endings in source and make files and shell scripts
─  checking for empty or unneeded directories
─  building 'mypackage_0.0.0.9000.tar.gz'
   Warning: invalid uid value replaced by that for user 'nobody'

Running /opt/R/4.2.2/lib/R/bin/R CMD INSTALL \
  /tmp/RtmpIFIn45/mypackage_0.0.0.9000.tar.gz --install-tests
* installing to library '/cloud/lib/x86_64-pc-linux-gnu-library/4.2'
* installing *source* package 'mypackage' ...
** using staged installation
** R
** byte-compile and prepare package for lazy loading
** help
*** installing help indices
** building package indices
** testing if installed package can be loaded from temporary location
** testing if installed package can be loaded from final location
** testing if installed package keeps a record of temporary installation path
* DONE (mypackage)
```

Step 10: Install and load your package

```
library(mypackage)
```

1. Open the help page for cat_function (`?cat_function`)
2. Try adding TRUE and FALSE to the love argument

Bonus Steps

- Use version control (git)
- Make sure functions are working properly (**testthat**)
- Disclose package dependencies
- Create a README

Step 11: Use git for version control

```
usethis::use_git()
#> ✔ Initialising Git repo
#> ✔ Adding '.Rhistory', '.Rdata',
'.httr-oauth', '.DS_Store' to
'.gitignore'
```

Note: You'll likely hit an **error** here...that's OK! Learn more about setting up git in your environment here.

Step 12: Make sure functions work properly

Create unit tests

```
usethis::use_testthat()
#> ✔ Adding 'testthat' to Suggests field in
DESCRIPTION
#> ✔ Setting Config/testthat/edition field in
DESCRIPTION to '3'
#> ✔ Creating 'tests/testthat/'
#> ✔ Writing 'tests/testthat.R'
#> • Call `use_test()` to initialize a basic test
file and open it for editing.
```

Step 12: Make sure functions work properly

Create unit tests

```
usethis::use_test("cat_function")
#> ✔ Writing
'tests/testthat/test-cat_function.R'
#> • Edit 'tests/testthat/test-cat_function.R'
```

Note: You can delete the placeholder unit test

Step 12: Make sure functions work properly

Create unit tests

test-cat_function.R

Run Tests

```r
test_that("Cat function returns correct text if love = TRUE", {
  expect_equal(cat_function(love = TRUE), "I love cats!")
})

```

Step 12: Make sure functions work properly

Create unit tests

```
Console   Terminal ×   Background Jobs ×
R 4.2.2 · /cloud/project/
> devtools::test()
ℹ Testing mypackage

Attaching package: 'testthat'

The following object is masked from 'package:devtools':

    test_file

✔ | F W S  OK | Context
⠿ |         0 | cat_function
[1] "I love cats!"
✔ |         1 | cat_function

══ Results ═══════════════════════════════════════════
═══════════════════════════════════════════════════════
Duration: 0.1 s

[ FAIL 0 | WARN 0 | SKIP 0 | PASS 1 ]
>
```

Step 13: Disclose package dependencies

Often packages depend on other packages

What if you used the **purrr** in our package?

```
usethis::use_package("purrr")
#> ✔ Adding 'purrr' to Imports field in
DESCRIPTION
#> • Refer to functions with `purrr::fun()`
```

Step 14: Create a README

Your packages “home page” or “welcome mat”

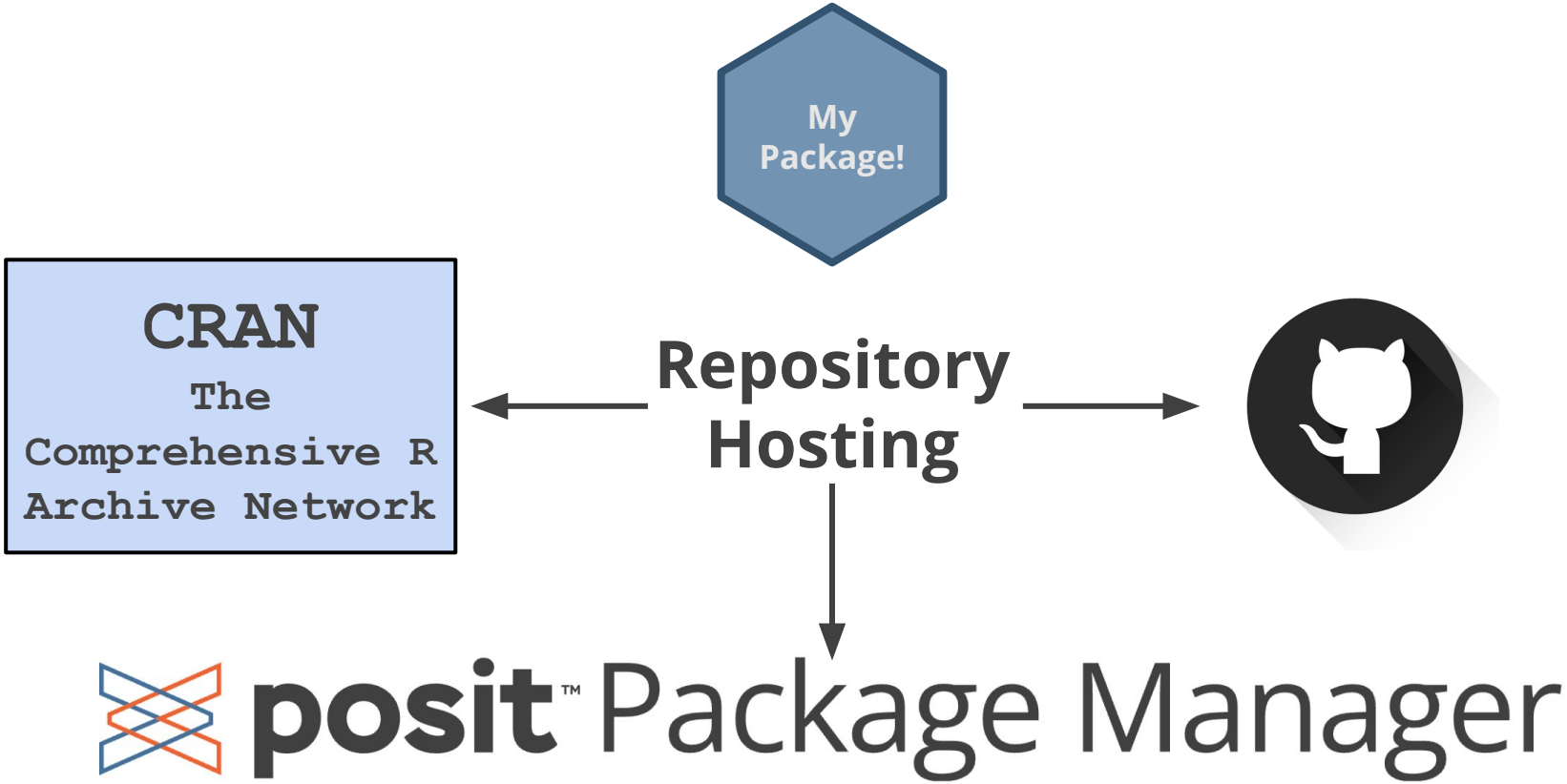
```
usethis::use_readme_rmd()
#> ✔ Writing 'README.Rmd'
#> ✔ Adding '^README\\.Rmd$' to
'.Rbuildignore'
#> • Update 'README.Rmd' to include
installation instructions.
#> ✔ Writing '.git/hooks/pre-commit'
```

check() and **install()** your package

Now What?

Time to share it!

How to share your package



posit™ Package Manager

posit Package Manager

R Repos

- all
- biocran-3.16
- cran
- curated
- dev
- internal
- published
- validated

Bioconductor Repos

- bioconductor

internal | Packages | Activity | Setup | What's New Bring Your Own Python Packages

DISTRIBUTION
Ubuntu 20.04 (Focal)

R Packages

Type to search

archer 0.4.0.0.0.0.1633386145
RStudio automatic architecture diagram creator

bikeHelpR 0.2.4.0.0.0.1653415303
Package of Helper Functions for the Bike Prediction Asset

connectapi 0.1.3.9000.0.0.0.1676307756
Utilities for Interacting with the 'Posit Connect' Server API

connectApiUtils 0.1.0.0.0.0.1548181775
Utilities for Interacting with RStudio Connect's API

archer

RStudio automatic architecture diagram creator

Overview | Readme

Provides a mechanism for quickly creating basic architecture diagrams for RStudio commercial products, based on some user configurable parameters.

INSTALL CODE

Copy

```
install.packages('archer')
```

VERSION
0.4.0.0.0.0.1633386145

19 IMPORTS

1 DEPENDS

1 SUGGESTS

posit™ Package Manager

- Pull in packages directly from **GitHub**
  - Documentation
- Users can upload locally built packages directly
  - Documentation
  - Example workflow
- Users can **serve previous versions of custom packages** by **freezing** repositories to specific dates, so code that used to work always will

Want to learn more?

- R Packages (2e)
  - By Hadley Wickham and Jennifer Bryan
  - Chapter 1 - "The Whole Game"
- Writing an R package from Scratch
  - 2014 blog post from Dr. Hilary Parker
- Posit Package Manager