

R Packages

Lecture 05

Dr. Colin Rundel

What are R packages?

R packages are just a collection of files - R code, compiled code (C, C++, etc.), data, documentation, and others that live in your library path.

```
1 .libPaths()
```

```
[1] "/Users/rundel/Library/R/arm64/4.5/library"
```

```
[2] "/Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/library"
```

```
1 dir(.libPaths())
```

```
[1] "_backup"
```

```
[2] "_build"
```

```
[3] "_cache"
```

```
[4] "abind"
```

```
[5] "and"
```

```
[6] "anytime"
```

```
[7] "ape"
```

```
[8] "archive"
```

```
[9] "arrayhelpers"
```

```
[10] "arrow"
```

```
[11] "AsioHeaders"
```

```
[12] "askpass"
```

```
[13] "assertthat"
```

```
[14] "astsa"
```

```
[15] "available"
```

```
[16] "babelwhale"
```

```
[17] "backports"
```

```
[18] "BART"  
[19] "base"  
[20] "base64enc"  
[21] "BayesFactor"  
[22] "bayesplot"
```

Search path

When you run `library(pkg)` the functions (and objects) in the package's namespace are attached to the global search path.

```
1 search()
```

```
[1] ".GlobalEnv"      "package:stats"  
[3] "package:graphics" "package:grDevices"  
[5] "package:utils"    "package:datasets"  
[7] "package:methods"  "Autoloads"  
[9] "package:base"
```

```
1 library(diffmatchpatch)
```

```
1 search()
```

```
[1] ".GlobalEnv"  
[2] "package:diffmatchpatch"  
[3] "package:stats"  
[4] "package:graphics"  
[5] "package:grDevices"  
[6] "package:utils"  
[7] "package:datasets"  
[8] "package:methods"  
[9] "Autoloads"  
[10] "package:base"
```

Loading vs attaching

If you do not want to attach a package you can directly *load* the package with `requireNamespace()`.

```
1 loadedNamespaces()
```

```
[1] "compiler"      "fastmap"  
[3] "cli"           "graphics"  
[5] "diffmatchpatch" "tools"  
[7] "htmltools"     "utils"  
[9] "yaml"          "grDevices"  
[11] "Rcpp"          "stats"  
[13] "datasets"      "rmarkdown"  
[15] "knitr"         "methods"  
[17] "jsonlite"      "xfun"  
[19] "digest"        "rlang"  
[21] "base"          "evaluate"
```

```
1 requireNamespace("forcats")
```

Loading required namespace: forcats

```
1 loadedNamespaces()
```

```
[1] "digest"        "methods"  
[3] "diffmatchpatch" "fastmap"  
[5] "xfun"          "magrittr"  
[7] "glue"          "knitr"  
[9] "htmltools"     "rmarkdown"  
[11] "lifecycle"     "utils"  
[13] "cli"           "graphics"  
[15] "grDevices"     "stats"  
[17] "compiler"      "forcats"  
[19] "base"          "tools"  
[21] "evaluate"      "Rcpp"  
[23] "yaml"          "rlang"  
[25] "jsonlite"      "datasets"
```

```
1 search()
```

```
[1] ".GlobalEnv"  
[2] "package:diffmatchpatch"  
[3] "package:stats"  
[4] "package:graphics"  
[5] "package:grDevices"  
[6] "package:utils"  
[7] "package:datasets"  
[8] "package:methods"  
[9] "Autoloads"  
[10] "package:base"
```

`requireNamespace()` also returns `TRUE` or `FALSE` depending on if it succeeds - it can be used to test if a package is

Automatic loading

Using a package function via `::` will automatically *load* the package (and its dependencies) but not *attach* it to the search path.

```
1 loadedNamespaces()
```

```
[1] "digest"      "methods"
[3] "diffmatchpatch" "fastmap"
[5] "xfun"        "magrittr"
[7] "glue"        "knitr"
[9] "htmltools"   "rmarkdown"
[11] "lifecycle"   "utils"
[13] "cli"         "graphics"
[15] "grDevices"   "stats"
[17] "compiler"    "forcats"
[19] "base"        "tools"
[21] "evaluate"    "Rcpp"
[23] "yaml"        "rlang"
[25] "jsonlite"    "datasets"
```

```
1 dplyr::lag(1:3)
```

```
[1] NA  1  2
```

```
1 loadedNamespaces()
```

```
[1] "digest"      "methods"
[3] "R6"          "diffmatchpatch"
[5] "fastmap"     "tidyselect"
[7] "xfun"        "magrittr"
[9] "glue"        "tibble"
[11] "knitr"       "pkgconfig"
[13] "htmltools"   "generics"
[15] "dplyr"       "rmarkdown"
[17] "lifecycle"   "utils"
[19] "cli"         "vctrs"
[21] "graphics"    "grDevices"
[23] "stats"       "compiler"
[25] "forcats"     "base"
[27] "tools"       "pillar"
[29] "evaluate"    "Rcpp"
[31] "yaml"        "rlang"
[33] "jsonlite"    "datasets"
```

```
1 search()
```

```
[1] ".GlobalEnv"
[2] "package:diffmatchpatch"
[3] "package:stats"
[4] "package:graphics"
[5] "package:grDevices"
[6] "package:utils"
[7] "package:datasets"
[8] "package:methods"
[9] "Autoloads"
[10] "package:base"
```

Where do R packages come from?

Generally from somewhere on the internet, most commonly from CRAN or GitHub, the methods for installing from these locations are slightly different.

CRAN:

```
1 install.packages("diffmatchpatch")
```

GitHub:

```
1 remotes::install_github("rundel/diffmatchpatch")
```

there is one other method that comes up (particularly around package development), which is to install a package from local file(s).

Local install:

From the terminal,

```
1 R CMD install diffmatchpatch_0.1.0.tar.gz
```

or from R,

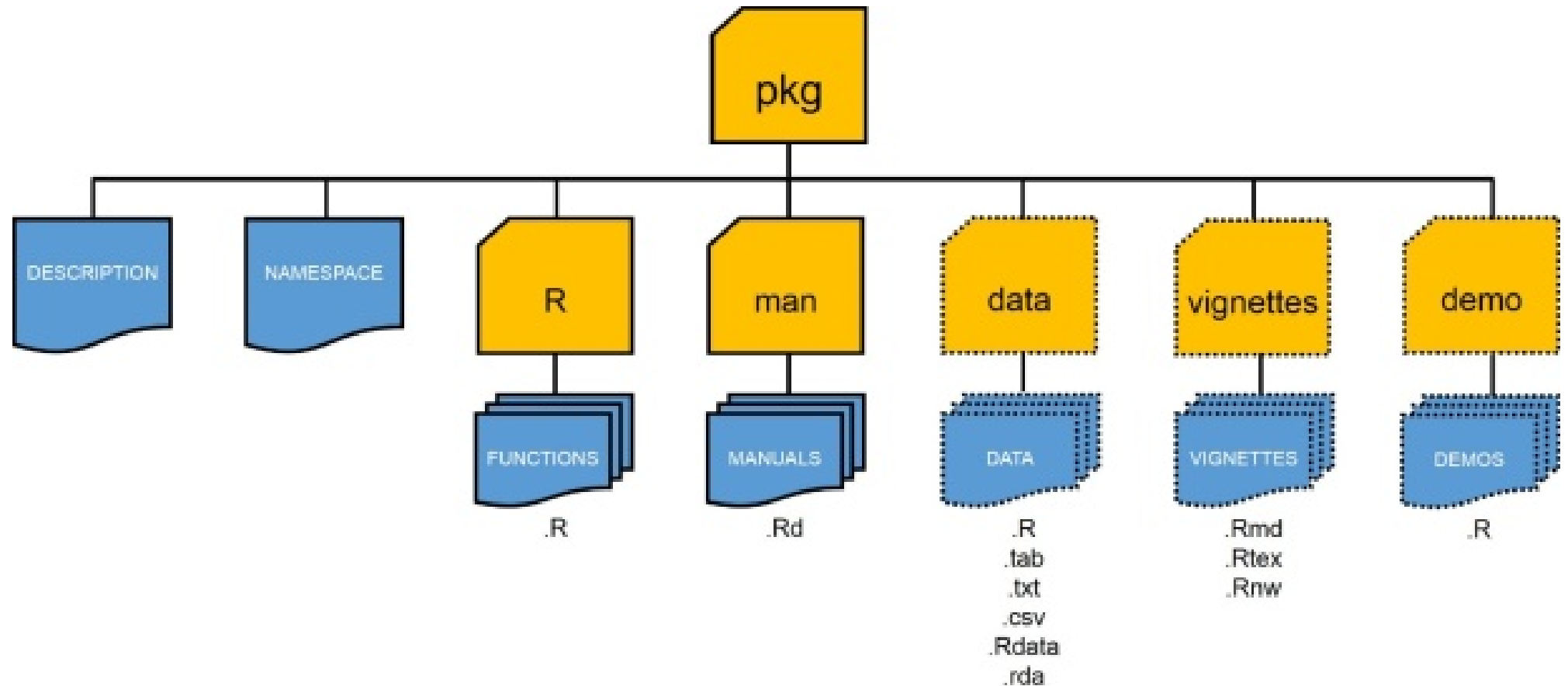
```
1 devtools::install("diffmatchpatch_0.1.0.tar.gz")
```

What is CRAN

The Comprehensive R Archive Network is the central repository of R packages.

- Maintained by the R Foundation and run by a team of volunteers, ~23k packages
- Contains all *current* versions of released packages as well as previous releases (including archived packages)
- Similar in spirit to Perl's CPAN, TeX's CTAN, and Python's PyPI
- Some important features:
 - All submissions are reviewed by humans + automated checks
 - Strictly enforced submission policies and package requirements
 - All packages must be actively maintained and support upstream and downstream changes

Structure of an R Package



Core components

- **DESCRIPTION** - file containing package metadata (e.g. package name, description, version, license, and author details). Also specifies package dependencies.
- **NAMESPACE** - details which functions and objects are exported by your package.
- **R/** - contains all R script files (**.R**) implementing package
- **man/** - contains all R documentation files (**.Rd**)

Optional components

The following components are optional, but quite common:

- `tests/` - contains unit tests (`.R` scripts)
- `src/` - contains code to be compiled (usually C / C++)
- `data/` - contains example data sets (`.rds` or `.rda`)
- `inst/` - contains files that will be copied to the package's top-level directory when it is installed (e.g. C/C++ headers, examples or data files that don't belong in `data/`)
- `vignettes/` - contains long form documentation, can be static (`.pdf` or `.html`) or literate documents (e.g. `.qmd`, `.Rmd` or `.Rnw`)

Package contents

Source Package

```
1 fs::dir_tree("~/Desktop/Projects/diffmatchpatch/")
```

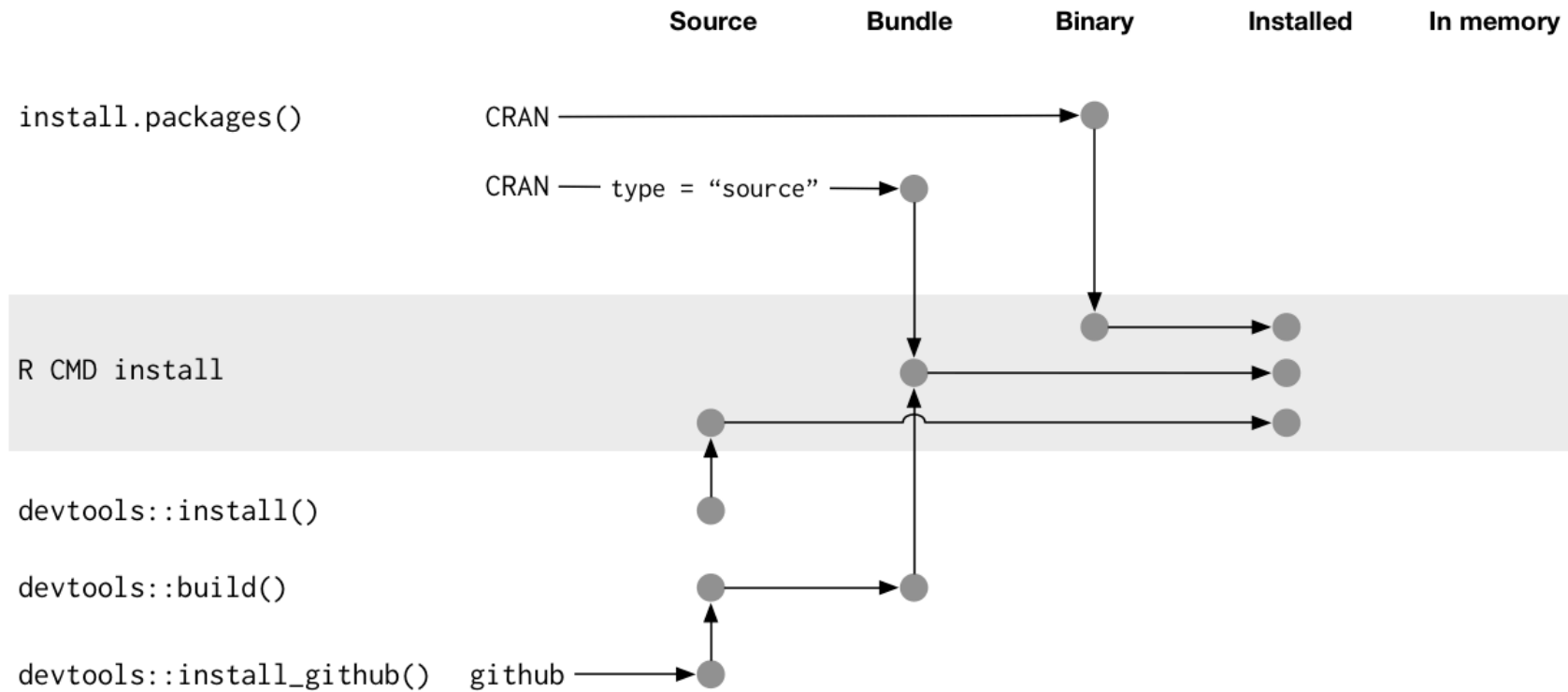
```
~/Desktop/Projects/diffmatchpatch/
├── DESCRIPTION
├── LICENSE.md
├── NAMESPACE
├── NEWS.md
├── R
│   ├── RcppExports.R
│   ├── diff.R
│   ├── diffmatchpatch-package.R
│   ├── match.R
│   ├── options.R
│   ├── patch.R
│   └── print.R
├── README.Rmd
├── README.md
├── cran-comments.md
├── diffmatchpatch.Rproj
├── inst
│   └── include
│       └── diff_match_patch.h
├── man
│   ├── diff.Rd
│   ├── dmp_options.Rd
│   ├── match.Rd
│   └── patch.Rd
└── src
    ├── Makevars
    ├── Makevars.win
    ├── RcppExports.cpp
    └── RcppExports.o
```

Installed Package

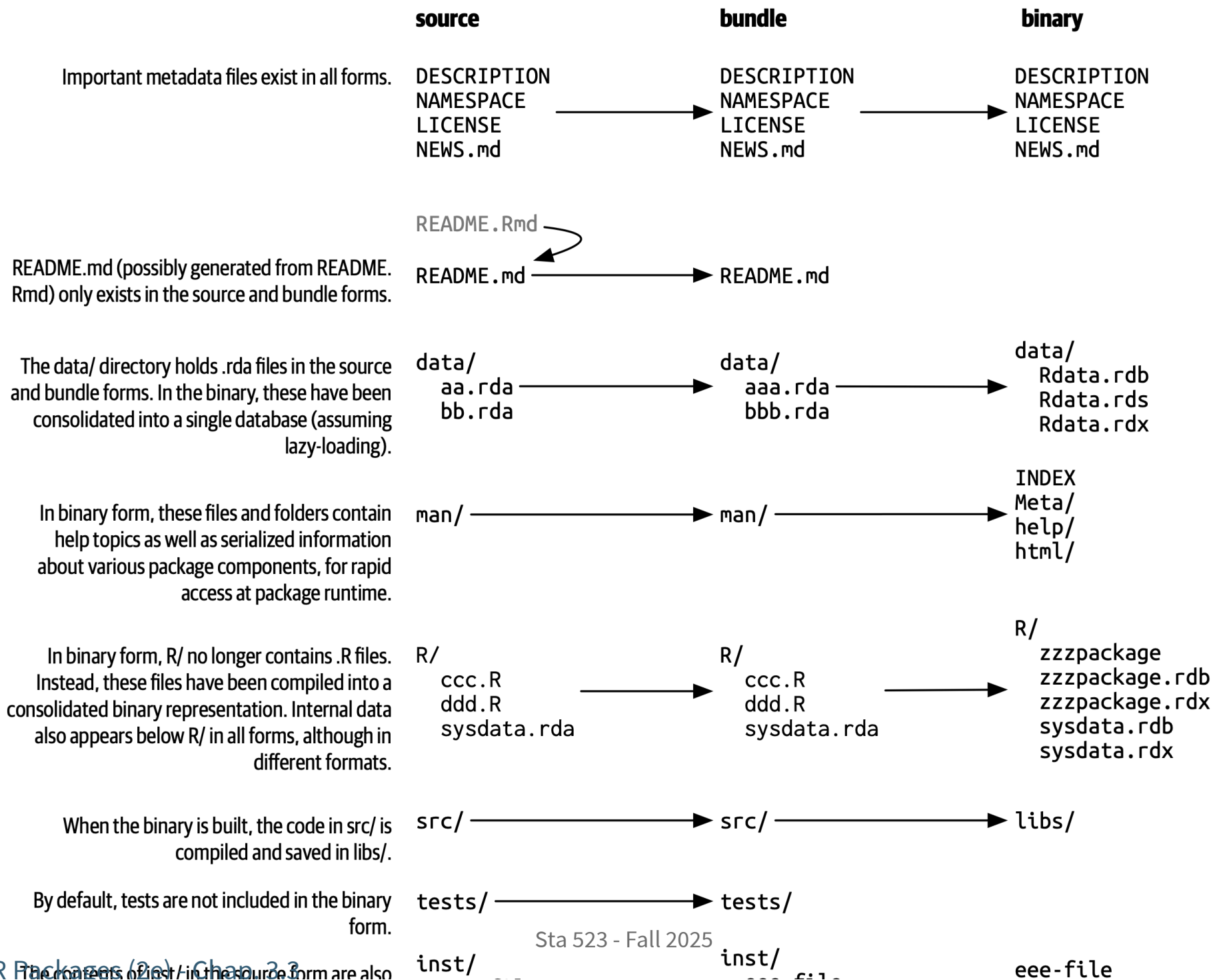
```
1 fs::dir_tree(system.file(package="diffmatchpatch"))
```

```
/Users/rundel/Library/R/arm64/4.5/library/diffmatchpatch
├── DESCRIPTION
├── INDEX
├── Meta
│   ├── Rd.rds
│   ├── features.rds
│   ├── hsearch.rds
│   ├── links.rds
│   ├── nsInfo.rds
│   └── package.rds
├── NAMESPACE
├── NEWS.md
├── R
│   ├── diffmatchpatch
│   ├── diffmatchpatch.rdb
│   └── diffmatchpatch.rdx
├── help
│   ├── AnIndex
│   ├── aliases.rds
│   ├── diffmatchpatch.rdb
│   ├── diffmatchpatch.rdx
│   └── paths.rds
├── html
│   ├── 00Index.html
│   └── R.css
├── include
│   └── diff_match_patch.h
└── libs
    └── diffmatchpatch.so
```

Package Installation



Package Installion - Files



the contents of `inst/` in the source form are also present in `inst/` in the bundle. In the binary, these files and folders move up to the top-level.

eee-file
fff-folder/
CITATION

eee-file
fff-folder/
CITATION

fff-folder/
CITATION

In the bundle, `inst/doc/` holds the results of rendering the vignettes and `build/vignette.rds` holds a table of vignette metadata.

vignettes/
articles/
ggg-article.Rmd
hhh-vignette.Rmd

build/
vignette.rds
inst/
doc/
hhh-vignette.R
hhh-vignette.Rmd
hhh-vignette.html

doc/
hhh-vignette.R
hhh-vignette.Rmd
hhh-vignette.html
index.html

The bundle also has the `vignettes/` directory, as in the source form, except articles are not included.

vignettes/
hhh-vignette.Rmd

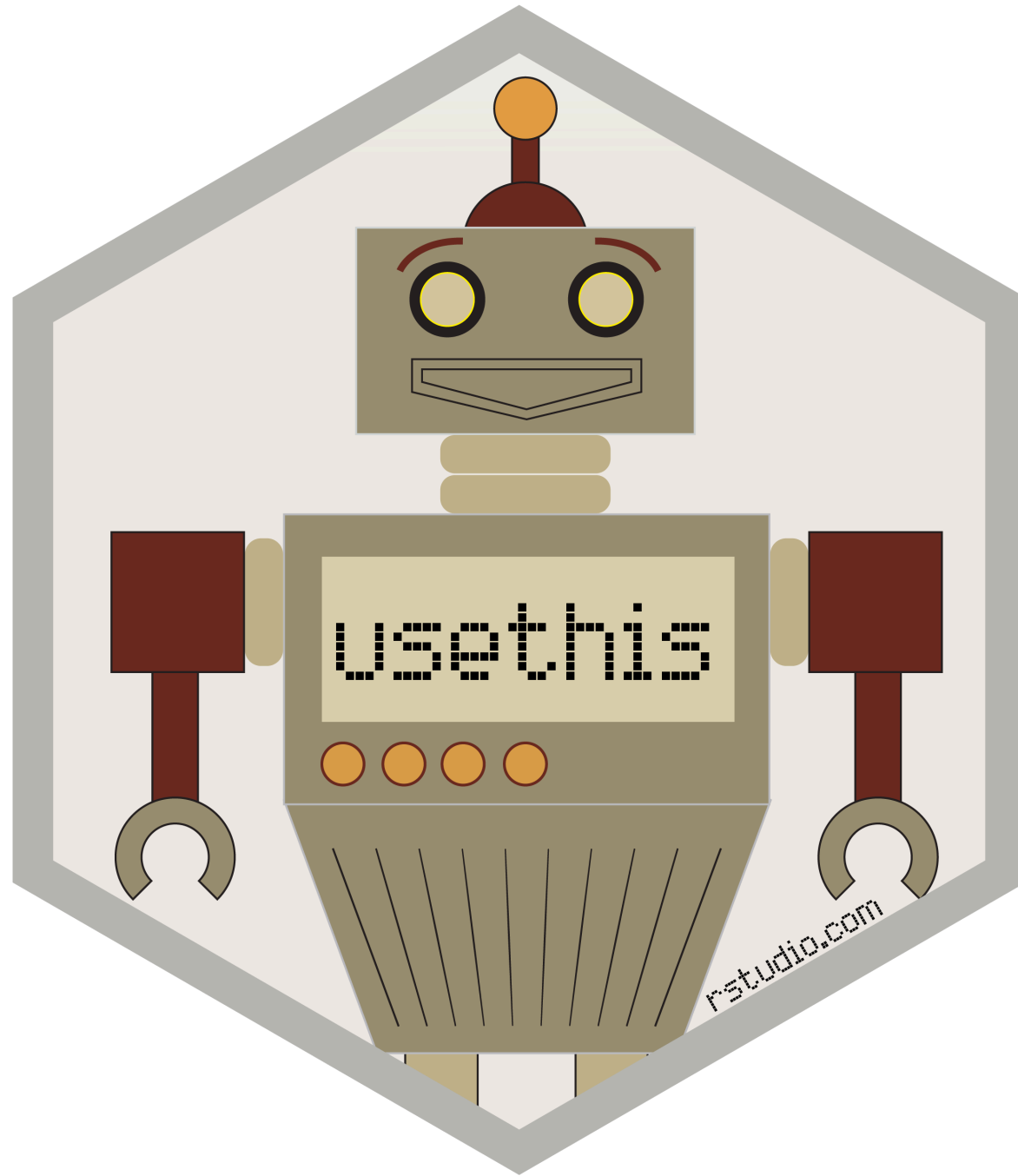
Files used only for development or for the pkgdown site are listed in `.Rbuildignore` and only exist in the source form.

_pkgdown.yml
.github/
.gitignore
.Rbuildignore
codecov.yml
cran-comments.md
data-raw/
LICENSE.md
pkgdown/
revdep/
zzzpackage.Rproj

Package development

What follows is an *opinionated* introduction to package development,

- this is not the only way to do thing (none of the following tools are required)
- We strongly recommend using:
 - RStudio
 - RStudio projects
 - GitHub
 - usethis
 - roxygen2
- Read and follow along with R Packages (2e) - [Chapter 1 - “The Whole Game”](#)



usethis

This is an immensely useful package for automating all kinds of routine (and tedious) tasks within R

- Tools for managing git and GitHub configuration
- Tools for managing collaboration on GitHub via pull requests (see `pr_*`)
- Tools for creating and configuring packages
- Tools for configuring your R environment (e.g. `.Rprofile` and `.Renv`)
- and much much more

Live demo

Building a Package

Start your package

Rather than having to remember all of the necessary pieces and their format, `usethis` can help you bootstrap your package development process.

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

Choosing a license

An important early step in developing a package is choosing a license - this is not trivial but is important to do early on, particularly if collaborating with others.

There are many resources available to help you choose a license, including:

<https://choosealicense.com/>

Documentation

All R packages are expected to have documentation for all *exported* functions and data sets (this is a CRAN requirement). This documentation is stored as `.Rd` files in the `man/` directory.

- The Rd format is a markup language that is loosely based on LaTeX
- Rd files are processed into LaTeX, HTML, and plain text when building the package
- All packages need Rd files, that doesn't mean you need to write Rd

Roxygen2

The premise of roxygen2 is simple: describe your functions in comments next to their definitions and roxygen2 will process your source code and comments to automatically generate `.Rd` files in `man/`, `NAMESPACE`, and, if needed, the `Collate` field in `DESCRIPTION`.

- roxygen uses special comment lines prefixed with `#'`
- roxygen specific command have the format `@cmd` and mostly match `Rd` commands
- `devtools::document()` or menu `Build > Document` with reprocess all source files and rebuild all `Rds`
- `usethis::create_package()` with `roxygen = TRUE` will initialize your package to use roxygen (default behavior)

Package vignette(s)

Vignette

Long form documentation for your package that live in `vignette/`, use `browseVignette(pkg)` to see a package's vignettes.

- Not required, but adds a lot of value to a package
- Generally these are literate documents (`.Rmd`, `.Rnw`) that are compiled to `.html` or `.pdf` when the package is built.
- Built packages retain the rendered document, the source document, and all source code
 - `vignette("colwise", package = "dplyr")` opens rendered version
 - `edit(vignette("colwise", package = "dplyr"))` opens code chunks
- Use `usethis::use_vignette()` to create a RMarkdown vignette template

Articles

These are an un-official extension to vignettes where package authors wish to include additional long form documentation that is included in their `pkgdown` site but not in the package (usually for space reasons).

- Use `usethis::use_article()` to create
- Files are added to `vignette/articles/` which is added to `.Rbuildignore`

Package data

Exported data

Many packages contain sample data (e.g. `nycflights13`, `babynames`, etc.)

Generally these files are made available by saving a single data object as an `.Rdata` file (using `save()`) into the `data/` directory of your package.

- An easy option is to use `usethis::use_data(obj)` to create the necessary file(s)
- Data is usually compressed, for large data sets it may be worth trying different options (there is a 5 Mb package size limit on CRAN)
- Exported data must be documented (possible via roxygen)

Lazy data

By default when attaching a package all of that packages data is loaded - however if `LazyData: true` is set in the packages' `DESCRIPTION` then data is only loaded when used.

```
1 pryr::mem_used()
```

55 MB

```
1 library(nycflights13)
2 pryr::mem_used()
```

55.2 MB

```
1 invisible(flights)
2 pryr::mem_used()
```

95.9 MB

If you use `usethis::use_data()` this option will be set in `DESCRIPTION` automatically.

Raw data

When published a package should generally only contain the final data set, but it is important that the process to generate the data is documented as well as any necessary preliminary data.

- These can live anywhere but the general suggestion is to create a `data-raw/` directory which is included in `.Rbuildignore`
- `data-raw/` then contain scripts, data files, and anything else needed to generate the final object
- See examples `babynames` or `nycflights`
- Use `usethis::use_data_raw()` to create and ignore the `data-raw/` directory.

Internal data

If you have data that you want to have access to from within the package but not exported then it needs to live in a special Rdata object located at `R/sysdata.rda`.

- Can be created using `usethis::use_data(obj1, obj2, internal = TRUE)`
- Each call to the above will overwrite, so needs to include all objects
- Not necessary for small data frames and similar objects - just create in a script. Use when you want the object to be compressed.
- Example `nflplotR` which contains team logos and colors for NFL teams.

Raw data files

If you want to include raw data files (e.g. `.csv`, shapefiles, etc.) there are generally placed in `inst/` (or a nested folder) so that they are installed with the package.

- Accessed using `system.file("dir", package = "package")` after install
- Use folders to keep things organized, Hadley recommends and uses `inst/extdata/`
- Example `sf`