MONASH University

MONASH BUSINESS SCHOOL

ETC4500/ETC5450 Advanced R programming

Week 3: R package development

`arp.numbat.space`

Outline

Outline

Conditions

```
message("This is what a message looks like")
#> This is what a message looks like

warning("This is what a warning looks like")
#> Warning: This is what a warning looks like

stop("This is what an error looks like")
#> Error in eval(expr, envir, enclos): This is what an error looks like
```

Conditions

```r
message("This is what a message looks like")
#> This is what a message looks like

warning("This is what a warning looks like")
#> Warning: This is what a warning looks like

stop("This is what an error looks like")
#> Error in eval(expr, envir, enclos): This is what an error looks like
```

- Ignore messages with `suppressMessages()`.
- Ignore warnings with `suppressWarnings()`.
- Ignore errors with `try()`.

try()

- Allows execution to continue even if an error occurs.
- Returns a special object that captures the error.

```r
f1 <- function(x) {
  log(x)
  10
}
f1("x")
```

```
Error in log(x): non-
numeric argument to mathematical function
```

```r
f2 <- function(x) {
  try(log(x))
  10
}
f2("a")
```

```
Error in log(x) : non-
numeric argument to mathematical function
[1] 10
```

Handling conditions

Allow you to specify what should happen when a condition occurs.

```r
tryCatch(
  error = function(cnd) {
    # code to run when error is thrown
  },
  code_to_run_while_handlers_are_active
)
withCallingHandlers(
  warning = function(cnd) {
    # code to run when warning is signalled
  },
  message = function(cnd) {
    # code to run when message is signalled
  },
  code_to_run_while_handlers_are_active
)
```

tryCatch()

```r
f3 <- function(x) {
  tryCatch(
    error = function(cnd) NA,
    log(x)
  )
}

f3("x")
```

```
[1] NA
```

withCallingHandlers()

```r
f4 <- function(x) {
  withCallingHandlers(
    warning = function(cnd) cat("How did this happen?\n"),
    log(x)
  )
}

f4(-1)
```

```
How did this happen?

[1] NaN
```

Exercise

Explain the results of running the following code

```r
show_condition <- function(code) {
  tryCatch(
    error = function(cnd) "error",
    warning = function(cnd) "warning",
    message = function(cnd) "message",
    {
      code
      5
    }
  )
}
show_condition(stop("!"))
show_condition(10)
show_condition(warning("?!"))
```

Outline

System setup

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

System setup

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

R build toolchain

- Windows:
  `https://cran.r-project.org/bin/windows/Rtools/`
- macOS: `xcode-select --install`
- Linux: `sudo apt install r-base-dev`

System setup

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

R build toolchain

- Windows: `https://cran.r-project.org/bin/windows/Rtools/`
- macOS: `xcode-select --install`
- Linux: `sudo apt install r-base-dev`

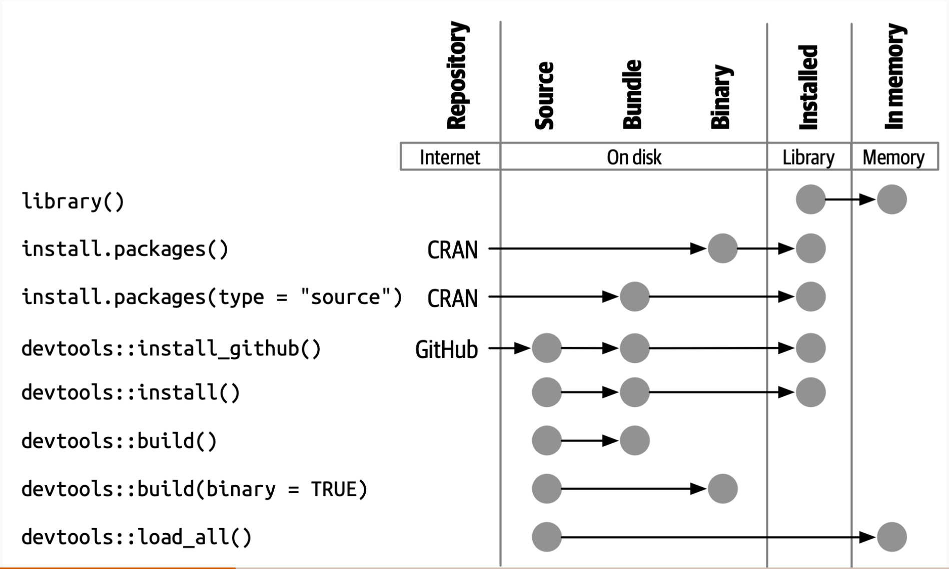
Verify

```r
library(devtools)
dev_sitrep()
```

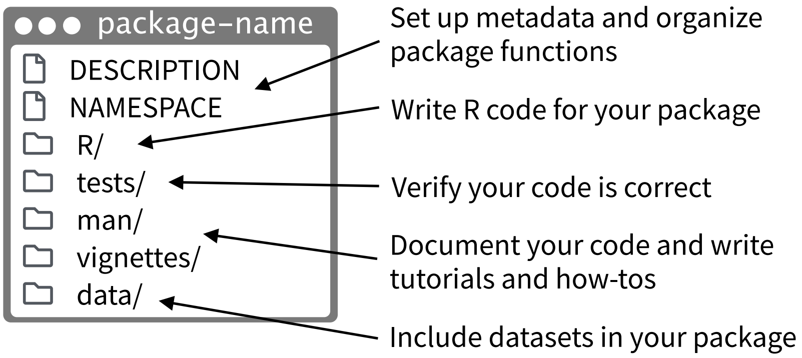
Package states

- **source**: the original files
- **bundled**: some processing, and compressed to a single `.tar.gz` file (e.g., to upload to CRAN)
- **binary**: what you usually download from CRAN
- **installed**: decompressed binary file stored in package library
- **in-memory**: loaded into R session using `library()`

Package states



Package structure


package-name
DESCRIPTION
NAMESPACE
R/
tests/
man/
vignettes/
data/
Set up metadata and organize package functions
Write R code for your package
Verify your code is correct
Document your code and write tutorials and how-tos
Include datasets in your package

There are multiple packages useful to package development, including **usethis** which handily automates many of the more repetitive tasks. Install and load **devtools,** which wraps together several of these packages to access everything in one step.

Package name

- Only letters, numbers and periods.
- Must start with a letter.
- It cannot end with a period.
- No hyphens or underscores.
- Use the `available::available()` function to try ideas.

Package code is different

- The `DESCRIPTION` file is the principal way to declare dependencies; we don’t do this via `library(somepackage)`.
- Be explicit about which functions are user-facing and which are internal helpers. By default, functions are not exported.

Exercise: Start on your package

1. Because we have already set up github repos for everyone, we need to first clone the repo to your local machine. Then we can start working on the package.

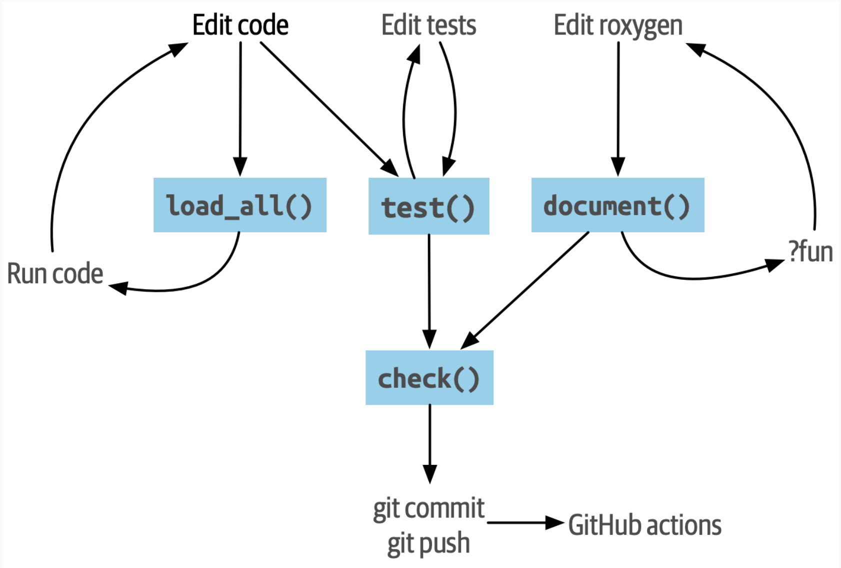
```
create_from_github("numbats/assignment-1-package-creation-robjhyndman")
```

2. Rename folder and Rproj file to your package name

3. Create package skeleton

```
create_package()
```

Workflow



Edit code
Edit tests
Edit roxygen
load_all()
test()
document()
Run code
?fun
check()
git commit
git push
GitHub actions

Workflow shortcuts

- `install()` : Ctrl-Shift-B
- `load_all()` : Ctrl-Shift-L
- `document()` : Ctrl-Shift-D
- `check()` : Ctrl-Shift-E
- `test()` : Ctrl-Shift-T

Outline

DESCRIPTION file

```
Package: doubler
Title: This package doubles numbers
Version: 0.0.0.9000
Authors@R:
    person("Rob", "Hyndman", , "Rob.Hyndman@monash.edu", role = c("aut", "cre"))
Description: Whether the input is real, complex or character, this will double it.
License: GPL (>= 3)
Encoding: UTF-8
Roxygen: list(markdown = TRUE)
RoxygenNote: 7.3.1
Suggests:
    testthat (>= 3.0.0)
Config/testthat/edition: 3
```

DESCRIPTION file

- **Title**: one line description. Plain text, title case, no more than 65 characters.
- **Description**: Several sentences, one paragraph. 80 characters per line, 4 space indentation. Don't include the package name in the Title or Description. Do not start with "This package does…"
- **Author**: Use `Authors@R` with `person()` for each author.
- **Version**. `Major.Minor.Patch.9000`. The `9000` is a placeholder for development versions.
- **License**: `GPL-3` or `MIT` are common.

DESCRIPTION file

- **Depends**: packages that are attached with your package. (Not needed for most packages.)
- **Imports**: packages that are used in your package. (Refer to functions using `pkg::fun()`.)
- **Suggests**: packages that are used in your package, but not required. (E.g., in tests or examples.)
- **LazyData**: `true` prevents users having to use `data()`.

DESCRIPTION file

Functions to help with the DESCRIPTION file:

- `use_github()` or `use_github_links()`: set the GitHub repository, URL and BugReports.
- `use_mit_license()`: set the license to MIT.
- `use_gpl3_license()`: set the license to GPL-3.
- `use_package()`: Add package to Imports or Suggests.
- `use_data()`: Add data to your package.
- `use_tidy_description()`: Clean up the DESCRIPTION file.

NAMESPACE file

- Generated by `roxygen2`, so don’t edit by hand.
- `export()`: export a function (including S3 and S4 generics).
- `S3method()`: export an S3 method.
- `importFrom()`: import selected object from another namespace (including S4 generics).
- `import()`: import all objects from another package’s namespace.
- `useDynLib()`: registers routines from a DLL (this is specific to packages with compiled code).

Outline

Documenting the package

```r
use_package_doc()
```

Documenting functions

- Add roxygen2 comments to your `.R` files
  - RStudio menu: `Code > Insert roxygen skeleton` (while cursor is within function)
  - Or use Github Copilot (in RStudio or VS-Code)
  - Or write them by hand
- Then use `document()` to generate the `Rd` files and the `NAMESPACE` file. (Or press `Ctrl+Shift+D` in RStudio.)
- Preview documentation with `?function`

Documenting functions

```
#' Title
#'
#' Description
#' More description
#'
#' @param x Description of x
#' @inheritParams fun
#' @returns Description of return value
#' @examples
#' @importFrom pkg fun
#' @import pkg
#' @rdname fun
#' @aliases fun
#' @seealso fun
#' @references Some reference
#' @author Your name
#' @export
```

Documenting data

- Put raw data in `data-raw/`
- Code to wrangle data and create objects in `data-raw/`
- `use_data(object)` to add rda to `data/`

```r
#' Title
#'
#' Description
#' More description
#'
#' @source Where did you get the data?
#' @format Class, dimensions, or other details
#' @keywords datasets
#' @examples
"object"
```

README.Rmd

1. Describe the high-level purpose of the package.
2. A simple example illustrating package.
3. Installation instructions
4. An overview of the main components of the package.

- Like a short vignette
- Displayed on the Github repository and the front page of the pkgdown site.
- Create with `usethis::use_readme_rmd()`
- Build with `devtools::build_readme()`

Vignettes

- A long-form guide to your package, or an extended example.
    - `usethis::use_vignette("my-vignette")`
    - Creates a `vignettes/` directory.
    - Adds the necessary dependencies to `DESCRIPTION`
    - Drafts a vignette, `vignettes/my-vignette.Rmd`.
    - Adds some patterns to `.gitignore`

Vignettes YAML

```yaml
---
title: "Vignette Title"
author: Your name
output: rmarkdown::html_vignette
vignette: >
  %\VignetteIndexEntry{Vignette Title}
  %\VignetteEngine{knitr::rmarkdown}
  %\VignetteEncoding{UTF-8}
---
```

Vignettes initial code chunks

~~~
```{r, include = FALSE}
knitr::opts_chunk$set(
  collapse = TRUE,
  comment = "#>"
)
```
~~~

~~~
```{r setup}
library(yourpackage)
```
~~~

- Any package used in a vignette must be included in Suggests if not already in Imports.

NEWS

- List changes in each release that users might care about.
- Use `usethis::use_news_md()` to create a `NEWS.md` file.

```
# foofy (development version)

* Better error message when grooving an invalid grobble (#206).

# foofy 1.0.0

## Major changes

* Can now work with all grooveable grobbles!

## Minor improvements and bug fixes

* Printing scrobbles no longer errors (@githubusername, #100).
```

Outline

testthat v3

- `usethis::use_testthat()`
  - Create a `tests/testthat/` directory.
  - Add `testthat` to the Suggests field in DESCRIPTION and specify `testthat 3e` in the `Config/testthat/edition` field.
  - Create a file `tests/testthat.R` that runs all your tests when `check()` runs.
- Every exported function should have tests.
- `usethis::use_test("some_tests.R")` creates a test file for a function or group of functions.
- Each R file should match a test file.

testthat v3

- Test files live in `tests/testthat/` and are named `test-*.R`.
- Each test file should test one function or a small group of related functions.
- Useful testing functions:
  - `expect_equal()`, `expect_identical()`, `expect_true()`, `expect_false()`
  - `expect_error()`, `expect_warning()`, `expect_message()`
- `test()` runs all tests.

What to test

- Focus on testing the exported functions.
- Strive to test each behaviour in one and only one test.
- Avoid testing simple code that you're confident will work.
- Always write a test when you discover a bug.
- The test-first philosophy: always start by writing the tests, and then write the code that makes them pass.
- Use `devtools::test_coverage()` to see which parts of your package are tested.

Outline

pkgdown websites

- `usethis::use_pkgdown()`
  - Creates `_pkgdown.yml` to configure site.
  - Updates `.Rbuildignore`
  - Adds `docs` to `.gitignore`
- `pkgdown::build_site()` to build the site.
- `usethis::use_pkgdown_github_pages()` to publish the site via GitHub Actions and GitHub Pages.
- Make a hex sticker with the `hexSticker` package.
- Add it using `usethis::use_logo()`.

Outline

Github Actions

- Some development tasks can be executed automatically on Github with a trigger (e.g., a push)
- Run `R CMD check`:usethis::use_github_action("check_standard")'
- Compute test coverage: `usethis::use_github_action("test-coverage")`
- Build and deploy pkgdown site: `usethis::use_github_action("pkgdown")`
- The `.github/workflows/` directory contains action files.
- See `https://github.com/r-lib/actions/` for more examples.