Bernhard Meindl
(bernhard.meindl@statistik.gv.at)
Statistics Austria (QM)

Modern Packaging in R

Best practices (CI, testing, linting and more)

uRos 2019 - Bukarest
May 20th, 2019

Bernhard Meindl
(bernhard.meindl@statistik.gv.at)
Statistics Austria (QM)

uRos 2019 - Bukarest
May 20th, 2019

Modern Packaging in R

Motivation

- Why
- How? $\rightarrow$ This is what we are now doing
- But: in a nice and easy way
  - using CI
  - producing nicely formatted code
  - automated tested code
- Good ideas:
  - find a good, short name (Package `available`)
  - think about
    - why the package exist and
    - what functionality should it contain?
  - define a namespace

Bernhard Meindl
(bernhard.meindl@statistik.gv.at)
Statistics Austria (QM)

uRos 2019 - Bukarest
May 20th, 2019

Modern Packaging in R

Create a Package

Traditionally

- Traditionally
  - `utils::package.skeleton()`
  - manually writing `man`-files and managing `NAMESPACE`, ...
- Idea:
  - some functions and exports where defined
  - a `README` file with instructions was created
- Problem:
  - quite inconvenient

An improved and simpler way (1)

- Assumptions for this tutorial:
  - Rstudio is installed and also `git` is available
  - packages `devtools`, `usethis`, `roxygen2`, `testthat`, `covr` and `lintr` are installed
- Using package `usethis` to create a package skeleton (and a corresponding Rstudio project)
  - `?usethis::create_package()`
- Note: take care to use an absolute path here if you're already in an rstudio project
- Then we can add additional parts for a good package "selectively"

An improved and simpler way (2)

- These codes create a new package `urosconfpkg` and a Rstudio project

```
usethis::create_package(path = "~/urosconfpkg")
```

- if Rstudio is available, it switches to the new project
- Note: one may change defaults of the `DESCRIPTION` file

```
usethis::create_package(path = "~/urosconfpkg", fields = list(
  `Authors@R` = 'person(
    given = "Bernhard", family = "Meindl",
    email = "bernhard.meindl@statistik.at", role = c("aut", "cre"))',
  License = "GPL-3",
  Version = "0.1"
))
```

An improved and simpler way (3)

- content of the `DESCRIPTION` file (after some manual updates)

```
Package: urosconfpkg
Title: A Demo Package for Uros19 in Bukarest
Version: 0.1
Authors@R:
  person(given = "Bernhard",
         family = "Meindl",
         role = c("aut", "cre"),
         email = "bernhard.meindl@statistik.at")
Description: A boilerplate for a CI-capable modern R package
License: GPL-3
Encoding: UTF-8
LazyData: true
RoxygenNote: 6.1.1
```

Bernhard Meindl
(bernhard.meindl@statistik.gv.at)
Statistics Austria (QM)

uRos 2019 - Bukarest
May 20th, 2019

Modern Packaging in R

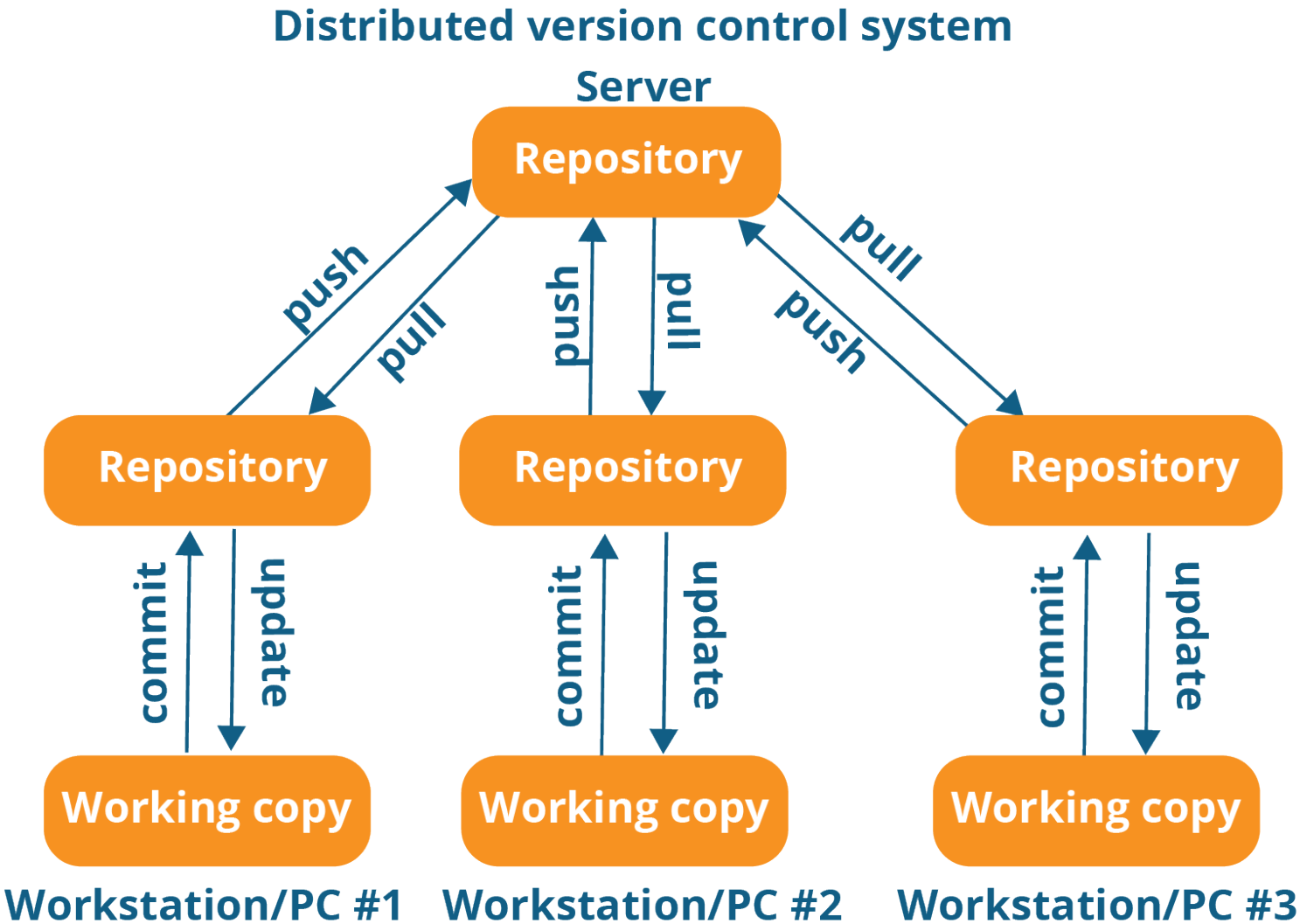
Using Git

What is git? (1)

Git is an open source, distributed version control system.

- **control system**: can be used to store content (mostly code)
- **versioned**: keeps track of multiple versions and the entire history
- **distributed**: git has a central repository and a number of local repos

What is git? (2)


Distributed version control system
Server
Repository
push
pull
push
pull
pull
push
Repository
Repository
Repository
commit
update
commit
update
commit
update
Working copy
Working copy
Working copy
Workstation/PC #1
Workstation/PC #2
Workstation/PC #3

Using git and github (1)

- code should always be versioned → we want to use `git`
  - 1: `usethis::use_git()` initiates a local git repo
  - 2: `usethis::use_github()` links the local repo to github
- it is required to have `github.com` account
- we need to set up authentification, either
  - `usethis::browse_github_pat()` using an access token or
  - using public/private key authorization, see `?usethis::use_github`
- Good documentation can be found **here**

Using git and github (2)

STATISTICS AUSTRIA
The Information Manager

Creating a local git repository

```
usethis::use_git()
```

- this function initiates the local repo
- it asks to also commits all existing files (you should says `yes` here)
- also say yes, if you are asked whether Rstudio should be restarted

Using git and github (3)

- Authorization to github (via token)

```
usethis::browse_github_pat() # create a access token/pattern
usethis::edit_r_environ() # add it to the .Renviron file
```

- The token is like a password (and should be kept as such)
- The `.Renviron` file should have a line in the following format:

```
GITHUB_PAT=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

- Next step: create remote repo on github and push our code

```
usethis::use_github()
```

- modifies `DESCRIPTION` adding `URL` and `BugReports` fields
- Note: this linkage may also be done manually, `usethis` is not required

Bernhard Meindl
(bernhard.meindl@statistik.gv.at)
Statistics Austria (QM)

uRos 2019 - Bukarest
May 20th, 2019

Modern Packaging in R

Adding Features

Adding features (1)

- Add a `NEWS.md` file in markdown format document track changes in the package

```
usethis::use_news_md()
```

```
# urosconfpkg 0.1

* Added a `NEWS.md` file to track changes to the package.
```

Adding features (2)

- Add a `README.(r)md` in (R)markdown format
- already filled with a nice template that can be easily adjusted
- The `README.md` generates the content at the bottom of a GitHub repository, such as how to install or use the package.
- You can also put your badges for number of downloads, version on CRAN (if applicable), continuous integration status, and code coverage.

```
# plain markdown
usethis::use_readme_md()

# rmarkdown style (you can use code-chunks)
usethis::use_readme_rmd()
```

- if using Rmarkdown, one has to regularily regenerate the `md` file!

Bernhard Meindl
(bernhard.meindl@statistik.gv.at)
Statistics Austria (QM)

uRos 2019 - Bukarest
May 20th, 2019

Modern Packaging in R

Add functionality

Adding R code (1)

- the `R` folder is by default empty
- we write a simple function in `R/greets.R`
- note, we're using a function from another package here!

```
greets <- function(name = "everybody") {
  s <- "hi {name}, the uros 2019 in bukarest is great!"
  message(glue::glue(s, name = shQuote(name)))
}
```

- it is required to add the `glue` package to the `Imports` section of `DESCRIPTION`
  - easy: `usethis::use_package("glue")`
- if we use `glue::fun()`, this is all we need to do

Adding R code (2)

- the `DESCRIPTION` file now is

```
Package: urosconfpkg
Title: A Demo Package for Uros19 in Bukarest
Version: 0.1
Authors@R:
    person(given = "Bernhard",
           family = "Meindl",
           role = c("aut", "cre"),
           email = "bernhard.meindl@statistik.at")
Description: A boilerplate for a CI-capable modern R package
License: GPL-3
Encoding: UTF-8
LazyData: true
Imports:
    glue
RoxygenNote: 6.1.1
```

Bernhard Meindl
(bernhard.meindl@statistik.gv.at)
Statistics Austria (QM)

uRos 2019 - Bukarest
May 20th, 2019

Modern Packaging in R

Adding data

Adding data to the package (1)

- Two possible ways:
  - 1: directly use an existing data set with `usethis::use_data()`
  - 2: use a script to generate data for improved reproducability
- We give an example for the second approach

```
usethis::use_data_raw()
```

- This create a folder `data-raw` in which we can put scripts that generate data

Adding data to the package (2)

- `data-raw/urosdata.R` is a script to generate data (note the `set.seed` call)

```
set.seed(1)
N <- 50
urosdata <- data.frame(
  x = sample(letters, N, replace = TRUE),
  v = rnorm(N)
)
usethis::use_data(urosdata, overwrite = TRUE)
```

- we just need to source this file
- `usethis::use_data()` copies the dataset to the `data` folder and makes it available

Bernhard Meindl
(bernhard.meindl@statistik.gv.at)
Statistics Austria (QM)

uRos 2019 - Bukarest
May 20th, 2019

Modern Packaging in R

Documentation

Use roxygen (1)

- do not repeat yourself and document consistently
- make use of markdown to even further simplify the documentation
- More information here
- `usethis::use_roxygen_md()` sets up everything
- **Note**: If you try to document before setting this up, you probably need to install `roxygen2md` to convert existing documentation to markdown format in advance

```
remotes::install_github("r-lib/roxygen2md")
usethis::use_roxygen_md()
```

Use roxygen (2)

- we need to document `greets()` and the dataset
- in Rstudio: `Code` → `Insert Roxygen Skeleton`
- modify `R/greets.R` (note the Markdown tags)

```
#' A nice hello from Bukrarest
#'
#' [greets()] gives a **warm** greeting from the Uros19.
#'
#' @param name specify who we want to greet.
#' @return `NULL`
#' @export
#' @examples
#' greets()
greets <- function(name = "everybody") {
  s <- "hi {name}, the uros 2019 in bukarest is great!"
  message(glue::glue(s, name = shQuote(name)))
}
```

Use roxygen (3)

- documenting data is done via `R/data.R`

```
#' urosdata
#'
#' Some testdata for `urosconfpkg` with this two variables:
#' - `v`: a factor of lower case letter
#' - `x` random values from a standard normal distribution
#' @usage data(urosdata)
#' @name urosdata
NULL
```

- update the documentation and the `NAMESPACE`

```
devtools::document() # or in the Rstudio "Build"-tab
```

Bernhard Meindl
(bernhard.meindl@statistik.gv.at)
Statistics Austria (QM)

uRos 2019 - Bukarest
May 20th, 2019

Modern Packaging in R

Vignette

Adding a package vignette (1)

- Package vignettes are a great way to demonstrate the functionality of your package
- Setting everything up can be done with:

```
usethis::use_vignette("intro")
```

- updates `NAMESPACE` with required `Imports` and `Suggests`
- updates `DESCRIPTION` to specify the vignette builder (markdown)
- adding files that should be ignored in `git`
- creates `vignettes/intro.Rmd` with a default template

Adding a package vignette (2)

- we modify `vignettes/intro.Rmd`

```
---
title: "A short introduction"
author: "Bernhard Meindl"
date: "`r Sys.Date()`"
output: rmarkdown::html_vignette
vignette: >
  %\VignetteIndexEntry{Vignette Title}
  %\VignetteEngine{knitr::rmarkdown}
  %\VignetteEncoding{UTF-8}
---

## Motivation
We write this to facilitate the useage of this package.
```

- build with `devtools::build_vignettes()`
- will be updated whenever we build the package.

Bernhard Meindl
(bernhard.meindl@statistik.gv.at)
Statistics Austria (QM)

uRos 2019 - Bukarest
May 20th, 2019

Modern Packaging in R

Use Rcpp

Using Rcpp (1)

- Setup the package to use Rcpp is easy

```
usethis::use_rcpp()
```

- modifies the DESCRIPTION file
  - adds Imports and LinkingTo fields
- creates a src folder
- updates .gitignore and .Rbuildignore to exclude compiled files

Using Rcpp (2)

- We get instructions how to update our `NAMESPACE` via roxygen2
- we update a `R/urosconfpkg-package.R` adding at the top

```
#' @useDynLib urosconfpkg, .registration = TRUE
#' @importFrom Rcpp sourceCpp
NULL
```

- The updated `DESCRIPTION` file now is

```
Package: urosconfpkg
Title: A Demo Package for Uros19 in Bukarest
Version: 0.1
Authors@R:
  person(
  given = "Bernhard",
  family = "Meindl", role = c("aut", "cre"),
  email = "bernhard.meindl@statistik.at")
Description: A boilerplate for a CI-capable modern R package
License: GPL-3
Encoding: UTF-8
LazyData: true
LinkingTo: Rcpp
Imports: Rcpp, glue
Suggests: knitr, rmarkdown, testthat
RoxygenNote: 6.1.1
```

Using Rcpp (4)

- finishing `Rcpp` requires to add a simple C++ function
- content of `src/info.cpp`

```
#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
CharacterVector uros_info() {
  CharacterVector v;
  v = "the uros 2020 will take place in vienna :)";
  return(v);
}
```

- run `devtools::load_all()` and `devtools::document()`

Using Rcpp (5)

- We can then either
  - write a wrapper function and export those for the cpp function
  - call it within an existing `R` function like `greets()`

```
#' A nice hello from Bukrarest
#'
#' [greets()] gives a **warm** greeting from the Uros19.
#'
#' @param name specify who we want to greet.
#' @return `NULL`
#' @export
#' @examples
#' greets()
greets <- function(name = "everybody") {
  s <- "hi {name}, the uros 2019 in bukarest is great!"
  message(glue::glue(s, name = shQuote(name)))
  message(uros_info())
}
```

Bernhard Meindl
(bernhard.meindl@statistik.gv.at)
Statistics Austria (QM)

uRos 2019 - Bukarest
May 20th, 2019

Modern Packaging in R

Using CI

Using CI (1)

- Continous integration: allows to automatically build, check, test and deploy our code
- runs whenever the underlying code base (typically a connected git repo) changes
- we are using Travis CI: (`travis-ci.org`)
  - allows to build, test and check `R` packages
  - account is required, but you can you your github.com account
  - required to authorize travis to be able to access your github repos
  - after successful authorizing and syncing the accounts, the new `urosconfpkg` repo is shown

Using CI (2)

- telling R that we want to use travis

```
usethis::use_travis()
```

- it adds a basic `.travis.yml` file
- adds this file to `.Rbuildignore`
- useful settings in `.travis.yml`
  - `warnings_are_errors: true`
  - `r_check_args: "--as-cran"`

Using CI (3)

- additional options

```
language: r
sudo: required

cache:
  - packages: true

r_packages:
 - devtools

r_github_packages:
  - r-lib/usethis

notifications:
  email:
    on_success: always
    on_failure: always
```

Using CI (4)

- further documentation: `travis for R`
- additional options:
  - `after_success:` additional steps that should be done after the package was successfully built, checked and tested can be listed
  - `deploy:` it is possible to copy results, e.g a `pkgdown site`

Bernhard Meindl
(bernhard.meindl@statistik.gv.at)
Statistics Austria (QM)

uRos 2019 - Bukarest
May 20th, 2019

Modern Packaging in R

Testing the Package

Adding Tests (1)

- Testing the package is important
- package `testthat` can be used for this
- setting up the testing infrastucture is as easy as

```
usethis::use_testthat()
```

- adds `testthat` to field `Suggests` in `DESCRIPTION`
- creates a folder `tests` in which the unit-tests should be placed

Adding Tests (2)

- add as many files and tests folder `tests/testthat` as you like
- we add `files/tests/test_hello.R` (must start with `test*`)

```
## files/tests
## +-- testthat
## |   \-- test_greets.R
## \-- testthat.R
```

- content of `tests/testthat/test_hello.R`

```
# testing greets()
expect_null(greets())
expect_message(greets())
NULL
```

- finally, all tests can be run using `devtools::test()`

Monitoring test coverage (1)

- Code Coverage: measures the percentage of code that is tested through unit tests
- we want the percentage as high as possible
- we are using package `covr`
- interactive: `covr::report()` returns an clickable html-widget
- provides backends to two online code coverage tools
  - coveralls.io
  - codecov.io

Monitoring test coverage (2)

- we are now using codecov.io
- adding the coverage step to the package can be done with:

```
usethis::use_coverage("codecov")
```

- this function
  - creates `codecov.yml`
  - updates `.Rbuildignore`
  - updates `.travis.yml` adding
- after the package is successfully built, the code coverage is computed and uploaded

```
after_success:
  - Rscript -e 'covr::codecov()'
```

Link **codecov** and **travis**

- Login to **codecov.io**
  - `Add new repository` → Select your package
  - copy `Upload Token`
- Login to **travis-ci**
  - Go to your package
  - `More options` → `Settings`
  - Under `Environment variables` add:
    - Name: "CODECOV_TOKEN"
    - Value: der Token

Bernhard Meindl
(bernhard.meindl@statistik.gv.at)
Statistics Austria (QM)

uRos 2019 - Bukarest
May 20th, 2019

Modern Packaging in R

Code style

Setup automated code styling (1)

- Readability of a package is improves a lot, when
  - coding style is consistent within the package
  - e.g. `snake_case` vs. `camelCase` or indenting
- one (of many) styleguides: tidyverse styleguide
- Package styler allows automated code formatting
- Package lintr allows to automate checks

Setup automated code styling (2)

- Linting rules can be defined either
  - directly within `lintr::lintr_package()` or
  - in a file `.lintr` in the root of the package
- Adding code-style checks to CI is easy by adding a line `.travis.yml` to the `after_success` step

```
after_success:
   - # ...
   - Rscript -e 'lintr::lint_package()'
```

- Alternative: create `tests/testthat/test_style.R` with

```
test_that("style is nice", {
  lintr::expect_lint_free()
})
```

Bernhard Meindl
(bernhard.meindl@statistik.gv.at)
Statistics Austria (QM)

uRos 2019 - Bukarest
May 20th, 2019

Modern Packaging in R

Even more

Improve the package (1)

- Add badges to your `README.md`

```
# travis
![Build](https://travis-ci.org/{user/repo}.svg?branch=master)]
(https://travis-ci.org/{user/repo})

# codecov.io
[![Coverage](https://codecov.io/gh/{user/repo}/branch/master/graph/badge.s
(https://codecov.io/gh/{user/repo}/branch/master)
```

- many other badges available
- don't forget the badge with the cool sunglasses from the awesome official statistics software

Improve the package (2)

- pkgdown is a nice way to build a static webpage for a package
  - github has github pages (free hosting for static project pages)
  - `usethis::use_pkgdown()`
  - automatic deployment via travis → documentation
- `devtools::spell_check()` allows to detect typos
- run `goodpractice::gp()` to get additional hints to improve the package
- use package analytics, e.g. using the cranlogs package
- think of good examples to help users

Finishing (1)

- commit all changes to the local git repo
- push the changes to github and let the magic happen
  - the package is now built
  - checked
  - tested
  - vignettes are built
- if successful:
  - code coverage is computed and synced
  - code style is checked

Finishing (2)

- Once everything works, it is easy to reproduce for other packages
- `devtools::release()` ? :)
- **Downloads**
  - sample package
  - slides