Programming with R
Day 4
Packages
Using and Writing
University of Potsdam

Detlef Groth

Last Lecture

- R OOP:
  - OOP terminology
  - S3 method(object)
  - proto object$method()
  - saveRDS/readRDS
- code documentation
  - R vs Rd files
  - roxygen2 documentation
- R base graphics:
- empty plot surface
- low level commands
- multifigure plots

Outline

Day 1 (basics)
- setup, install
  editor, simple programs
- variables, operators
- data structures, control flow

Day 2 (basics)
- functions
- file input/output
- terminal interaction
- command line arguments

Day 3 (advanced)
- object oriented progr.
- code documentation
- R base graphics system

Day 4 (advanced)
- **using packages**
- **writing packages**
- **package documentation**

Day 5 (advanced)
- graphical user interfaces
- tcltk (shiny)

1 R Packages

Outline

Outline

- installation
  - CRAN
  - Bioconductor
- usage
  - library
  - require
- writing
  - coding
  - documenting
  - testing
  - packaging
  - installing

1.1 Installing packages from the web

- there are many R packages in the web available
- repositories (tested and quality controlled):
  - `https://cran.r-project.org/web/packages`
  - `https://www.bioconductor.org`
- from github (less trustable)

CRAN

`https://cran.r-project.org/web/packages/`

- August 2009 - 1928 packages
- Juni 2011 - 3023 packages
- July 2012 - 3914 packages
- May 2013 - 4526 packages
- January 2015 - 6221 packages
- October 2016 - 9332 packages
- July 2017 - 10875 packages
- August 2018 - 12954 packages
- Juny 2019 - 14311 packages

CRAN September 2020

https://cran.r-project.org 150%

CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Contributed Packages

Available Packages

Currently, the CRAN package repository features 16322 available packages.

Table of available packages, sorted by date of publication

Table of available packages, sorted by name

Installation of Packages

Please type `help("INSTALL")` or `help("install.packages")` in R for information on how to install packages from this repository. The manual R Installation and Administration (also contained in the R base sources) explains the process in detail.

CRAN Task Views allow you to browse packages by topic and provide tools to automatically install all packages for special areas of interest. Currently, 41 views are available.

Package Check Results

All packages are tested regularly on machines running Debian GNU/Linux, Fedora, macOS (formerly OS X), Solaris and Windows.

$\Rightarrow$ more than 16.000 packages

41 CRAN Task Views - 1

https://cran.r-project.org 150%

- For general concerns regarding task views contact the ctv package maintainer.

CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

Topics

Bayesian	Bayesian Inference
ChemPhys	Chemometrics and Computational Physics
ClinicalTrials	Clinical Trial Design, Monitoring, and Analysis
Cluster	Cluster Analysis & Finite Mixture Models
Databases	Databases with R
DifferentialEquations	Differential Equations
Distributions	Probability Distributions
Econometrics	Econometrics
Environmetrics	Analysis of Ecological and Environmental Data
ExperimentalDesign	Design of Experiments (DoE) & Analysis of Experimental Data
ExtremeValue	Extreme Value Analysis
Finance	Empirical Finance
FunctionalData	Functional Data Analysis
Genetics	Statistical Genetics
Graphics	Graphic Displays & Dynamic Graphics & Graphic Devices & Visualization
HighPerformanceComputing	High-Performance and Parallel Computing with R
Hydrology	Hydrological Data and Modeling
MachineLearning	Machine Learning & Statistical Learning
MedicalImaging	Medical Image Analysis
MetaAnalysis	Meta-Analysis
MissingData	Missing Data

41 CRAN Task Views - 2

https://cran.r-project.org 150%

CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

Hydrology	Hydrological Data and Modeling
MachineLearning	Machine Learning & Statistical Learning
MedicalImaging	Medical Image Analysis
MetaAnalysis	Meta-Analysis
MissingData	Missing Data
ModelDeployment	Model Deployment with R
Multivariate	Multivariate Statistics
NaturalLanguageProcessing	Natural Language Processing
NumericalMathematics	Numerical Mathematics
OfficialStatistics	Official Statistics & Survey Methodology
Optimization	Optimization and Mathematical Programming
Pharmacokinetics	Analysis of Pharmacokinetic Data
Phylogenetics	Phylogenetics, Especially Comparative Methods
Psychometrics	Psychometric Models and Methods
ReproducibleResearch	Reproducible Research
Robust	Robust Statistical Methods
SocialSciences	Statistics for the Social Sciences
Spatial	Analysis of Spatial Data
SpatioTemporal	Handling and Analyzing Spatio-Temporal Data
Survival	Survival Analysis
TeachingStatistics	Teaching Statistics
TimeSeries	Time Series Analysis
Tracking	Processing and Analysis of Tracking Data
WebTechnologies	Web Technologies and Services
gR	gRaphical Models in R

Installation from CRAN

On Unix systems with package managers, try first the package manager, this ensures automatic updates.
Linux Fedora:

```
[groth@bariuke build]$ sudo dnf search R-devtools
[sudo] password for groth:
Last metadata expiration check: 9:59:20 ago on Sun
   20 Sep 2020 07:33:48 PM CEST.
======= Name Exactly Matched: R-devtools ======
R-devtools.noarch : Tools to Make Developing R
    Packages Easier

[groth@bariuke build]$ sudo dnf install R-devtools
...
```

Fedora how many packages?

```
[groth@bariuke build]$ sudo dnf search R-* | head
Last metadata expiration check: 10:00:36 ago on
    Sun 20 Sep 2020 07:33:48 PM CEST.
======= Name & Summary Matched: R-* =============
R-littler-examples.x86_64 : R-littler Examples
======== Name Matched: R-* =======================
R-ALL.noarch : Data of T- and B-cell Acute Lymphocytic
    Leukemia
R-AUC.noarch : Threshold independent performance measures
    for probabilistic classifiers
R-AnnotationDbi.noarch : Annotation Database Interface
R-AsioHeaders-devel.noarch : Asio C++ Header Files
R-BH-devel.noarch : Boost C++ Header Files for R
R-BSgenome.noarch : Infrastructure for Biostrings-based
                    genome data packages
R-BSgenome.Celegans.UCSC.ce2.noarch : Caenorhabditis
            elegans genome (UCSC Release ce2)
```

```
[groth@bariuke]$ sudo dnf search R-* | grep -E '^R-' |
    grep -v i686 | wc -l
Last metadata expiration check: 10:01:12 ago on
    Sun 20 Sep 2020 07:33:48 PM CEST.
414
```

Information about a package

```
[groth@bariuke build]$ sudo dnf info R-brio
Available Packages
Name         : R-brio
Version      : 1.1.0
Release      : 1.fc31
Architecture : x86_64
Size         : 42 k
Source       : R-brio-1.1.0-1.fc31.src.rpm
Repository   : updates
Summary      : Basic R Input Output
URL          : https://CRAN.R-project.org/package=brio
License      : MIT
Description  : Functions to handle basic input output,
             : these functions always read and write
             : UTF-8 (8-bit Unicode Transf. Format)
             : files and provide more explicit
             : control over line endings
```

Installation within R

```
install.packages('pkgname')
```

Other commands:

- available.packages
- update.packages
- remove.packages

Package dependencies

If you have the choice between different packages which solves your task prefer packages with few dependencies over packages with many and prefere the MIT or BSD license over GPL licenses. GPL license requires that you always publish your source code if you add such packages to your application bundle.

```
> options(continue=' ')
> options(width=55)
> package.deps <- function(x,mode='all')  {
     if (!interactive()) {
          r <- getOption("repos");
          r["CRAN"] <- "https://lib.ugent.be/CRAN/"
          options(repos=r)
     }
```

```
    require(tools)
    deps=package_dependencies(x,recursive=TRUE)[[1]]
    if (mode == 'install') {
        idx = which(
        !(deps %in% rownames(installed.packages())))
        return(deps[idx])
    } else if (mode == 'nonbase') {
        ipacks=installed.packages()
        bpacks=ipacks[ipacks[,'Priority'] %in%
            c('base','recommended'),]
        rnms=setdiff(rownames(ipacks),rownames(bpacks))
        return(intersect(deps,rnms))
    } else if (mode == 'all') {
        return(deps)
    } else {
        stop('mode is either install or nonbase')
    }

}
```

```
> package.deps('igraph',mode='all')
 [1] "methods"   "graphics"  "grDevices" "magrittr"
 [5] "Matrix"    "pkgconfig" "stats"     "utils"
 [9] "grid"      "lattice"
> package.deps('igraph',mode='nonbase')
[1] "magrittr"  "pkgconfig"
> package.deps('argparse',mode='nonbase')
[1] "R6"       "jsonlite"
> package.deps('argparser',mode='nonbase')
character(0)
```

packageDescription

```
> packageDescription('argparser')
Package: argparser
Type: Package
Title: Command-Line Argument Parser
Version: 0.4
Date: 2016-04-03
Author: David J. H. Shih
Maintainer: David J. H. Shih
     <djh.shih@gmail.com>
Description: Cross-platform command-line
     argument parser written purely in R with no
     external dependencies. It is useful with
     the Rscript front-end and facilitates
     turning an R script into an executable
     script.
URL: https://bitbucket.org/djhshih/argparser
```

```
BugReports:
     https://bitbucket.org/djhshih/argparser/issues
Depends: methods
License: GPL (>= 3)
RoxygenNote: 5.0.1
NeedsCompilation: no
Packaged: 2016-04-04 00:02:25 UTC; davids
Repository: CRAN
Date/Publication: 2016-04-04 08:37:01
Built: R 3.6.1; ; 2019-11-03 13:22:01 UTC; unix

-- File: /usr/share/R/library/argparser/Meta/package.rds
> packageDescription('argparser')$License
[1] "GPL (>= 3)"
```

But not showing non-installed packages:

```
> packageDescription('brio')
[1] NA
```

url.show?

```
url.show("https://cran.r-project.org/web/packages/brio/")
```

⇒ But display to the terminal as html, unreadable

url2txt

```
> url2txt <- function (url,file=tempfile(),quiet=TRUE) {
    download.file(url, destfile = file,mode = "w",
        quiet=quiet)
    fin=fin = file(file,'r')
    res=''
    flag=FALSE
    while(length((line=readLines(fin,n=1L)))>0) {
        if (grepl('<body>',line)) {
            flag=TRUE
        } else if (flag) {
            line=gsub('</?.+?>','',line)
            res=paste(res,'\n',line)
        }
    }
    close(fin)
    return(res)
 }
```

```
> cat(substr(
  url2txt("https://cran.r-project.org/web/packages/brio")
  1,250))
 brio: Basic R Input Output
 Functions to handle basic input output, these functions
   read and write UTF-8 (8-bit Unicode Transformation For
   more explicit control over line endings.


 Version:
 1.1.0
```

Bioconductor

www.bioconductor.org

Bioconductor
OPEN SOURCE SOFTWARE FOR BIOINFORMATICS

Search:

Home | Install | Help | Developers | About

About *Bioconductor*

Bioconductor provides tools for the analysis and comprehension of high-throughput genomic data. *Bioconductor* uses the R statistical programming language, and is open source and open development. It has two releases each year, and an active user community. *Bioconductor* is also available as an AMI (Amazon Machine Image) and Docker images.

News

- *Bioconductor* 3.12 release schedule is announced.
- BiocAsia virtual conference registration is open (free registration!). October 15-18, 2020.
- BiocEurope virtual conference registration and abstract submission open Decemeber 14-18, 2020.
- See our google calendar for events, conferences, meetings, forums, etc. Add your event with email to events at

Install »

- Discover 1903 software packages available in *Bioconductor* release 3.11.

Get started with *Bioconductor*

- Install *Bioconductor*
- Get support
- Latest newsletter
- Follow us on twitter
- Install R

Learn »

Master *Bioconductor* tools

- Courses
- Support site
- Package vignettes
- Literature citations
- Common work flows
- FAQ
- Community resources
- Videos

Use »

Create bioinformatic solutions with *Bioconductor*

- Software, Annotation, and Experiment packages
- Docker and Amazon machine images
- Latest release announcement
- Use *Bioconductor* in the AnVIL. See our project updates.

Develop »

Contribute to *Bioconductor*

- Developer resources
- Use Bioc 'devel'
- 'Devel' packages
- Package guidelines
- New package submission
- Git source control
- Build reports

Aim

Bioconductor provides tools for the analysis and comprehension of high-throughput genomic data. Bioconductor uses the R statistical programming language, and is open source and open development. It has two releases each year, and an active user community.

...

Discover 1903 software packages available in Bioconductor release 3.11.

`https://bioconductor.org/`

Installation of Bioconductor packages

Example: package impute - Imputation for microarray data (currently KNN only)

```
if (!requireNamespace("BiocManager",
      quietly = TRUE)) {
     # install from CRAN
    install.packages("BiocManager")
}
# install from Bioconductor
BiocManager::install("impute")
```

CRAN vs Bioconductor

Bioconductor is more restrictive:

- should use S4 as OOP system
- S3 is not allowed
- must have vignette
- no line in code wider than 80 characters
- no dots in function or method names
- ...

See:

`http://bioconductor.org/developers/package-guidelines/`

Github

Sometimes a package might be available only on github or you might (need) to try out the newest package version.

Example:

```
library(devtools)
install_github("hadley/dplyr")
```

⇒ **Do this if you must!** No real integrity check for the code.

1.2 Using packages

- library - returns error if not installed
- require - returns false if not installed

```
> print(library(argparser)) # show which pkg is loaded
[1] "argparser" "tools"     "stats"     "graphics"
[5] "grDevices" "utils"     "datasets"  "methods"
[9] "base"
> print(require(argparse))
[1] FALSE
> print(require(argparser))
[1] TRUE
> print(require(argparsers))
[1] FALSE
```

Use require for install on request

```
if (!require('argparser')) {
    # in scripts you need to set the repo
    if (!interactive()) {
         r <- getOption("repos");
         r["CRAN"] <- "https://lib.ugent.be/CRAN/"
         options(repos=r)
    }
    install.packages('argparser')
    library('argparser')
}
```

The install.packages lib argument

R has normally two standard library folders where it puts its R packages, it is recommend to place non-standard packages not into the main package folder (Windows: 'C:/Program Files/...').
So I usually use if I install as root the second standard folder, on Windows this is mostly somewhere in C:/Users.
To find out which are the folders where your R-libraries are places use the .libPaths() function:

```
> print(.libPaths())
[1] "/home/groth/workspace/delfgroth/myr/rlibs"
[2] "/home/groth/R/x86_64-redhat-linux-gnu-library/3.6"
[3] "/usr/lib64/R/library"
[4] "/usr/share/R/library"
```

Modifying the .libPaths() folders

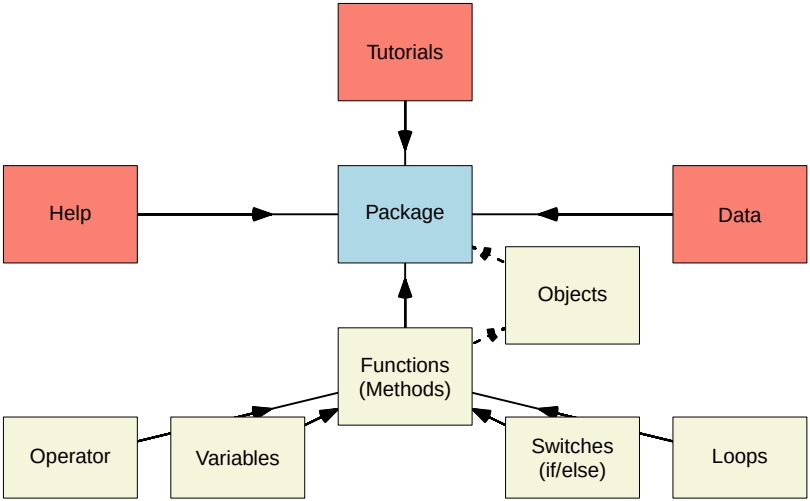
You can add manually an additional folder the .libPaths() folders.

```
.libPaths(c('/new/path/',.libPaths()))
# now install it into the first folder of libPaths()
install.packages('pkgname',lib=.libPaths()[1])
```

To use this new folder you have then every time to start your scripts with:

```
.libPaths(c('/new/path/',.libPaths()))
```

1.3 Writing packages



THE References

R Core team Writing R extensions (199 pages!):
`https://cran.r-project.org/doc/manuals/r-release/R-exts.pdf`
Wickham - R packages (198 pages, Book):
`https://r-pkgs.org/` (HTML)

Take this as references but we try to make it shorter here (KISS).

When to write a package?

When to start writing an R package?
As soon as you have 2 functions.
Why 2? After you have more than one function it starts to get easy to lose track of what your functions do, it starts to be tempting to name your functions foo or tempfunction or some other such nonsense. You are also tempted to put all of the functions in one file and just source it. That was what I did with my first project, which ended up being an epically comical set of about 3,000 lines of code in one R file. Ask my advisor about it sometime, he probably is still laughing about it.

`https://github.com/jtleek/rpackages`

Benefits:

- more structured work
- you don't have to hunt for the latest version of a certain function
- you can publish it
- someone else can use it more easily then your arbitary collections of code
- you can give him/her your tar.gz file of your package
- BTW: Where was this `package.deps` function or was it named `package.dependencies` ?

Hints

- Write your first package if you start your first real project using R
- You might not have the intension to publish it at all, that is not the point, the point is to structure your work!
- Have one personal collection of functions in your own package (my one is `dgtools`) and one project specific package (I have `mcgraph, asg, swiss`, ...)
- use git to track your code changes
- we cover this in 1st semester course "Databases and Practical Programming (Using Python 3)"
- Good: 2nd semester (Machine Learning course)
- Latest: project thesis 3rd semester if done with R

Steps

- setup a minimal R-package one function, one data set
- have on folder for all your developer versions of the package *rlibs-dev*
- run R CMD check *pgkname*
- run R CMD BUILD *pkgname*
- run R CMD INSTALL *pkgname.VERSION.tar.gz*
- have one folder for all your own installed packages *rlibs*

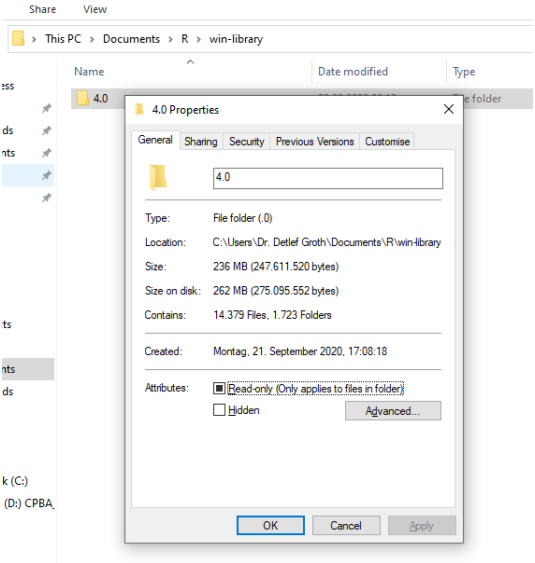
KISS with devtools?

```
> package.deps('devtools',mode='nonbase')
 [1] "usethis"     "callr"       "cli"
 [4] "desc"        "ellipsis"    "httr"
 [7] "jsonlite"    "memoise"     "pkgbuild"
[10] "pkgload"     "rcmdcheck"   "remotes"
[13] "rlang"       "roxygen2"    "rstudioapi"
[16] "sessioninfo" "testthat"    "withr"
[19] "processx"    "R6"          "assertthat"
[22] "crayon"      "glue"        "fansi"
[25] "digest"      "yaml"        "rprojroot"
[28] "htmltools"   "htmlwidgets" "magrittr"
[31] "crosstalk"   "promises"    "curl"
[34] "mime"        "openssl"     "prettyunits"
[37] "xopen"       "brew"        "commonmark"
[40] "knitr"       "purrr"       "Rcpp"
[43] "stringi"     "stringr"     "xml2"
```

```
[46] "evaluate"    "praise"      "clipr"
[49] "fs"          "gh"          "git2r"
[52] "whisker"     "lazyeval"    "ini"
[55] "base64enc"   "highr"       "markdown"
[58] "xfun"        "askpass"     "ps"
[61] "later"       "tibble"      "backports"
[64] "sys"         "BH"          "lifecycle"
[67] "pillar"      "pkgconfig"   "vctrs"
[70] "utf8"
> length(package.deps('devtools',mode='nonbase'))
[1] 70
```

⇒ not really KISS

250MB, 14.000 files for devtools ...



KISS with roxygen2? Only slightly better!

```
> package.deps('roxygen2',mode='nonbase')
 [1] "brew"         "commonmark"   "desc"
 [4] "digest"       "knitr"        "pkgload"
 [7] "purrr"        "R6"           "Rcpp"
[10] "rlang"        "stringi"      "stringr"
[13] "xml2"         "assertthat"   "crayon"
[16] "rprojroot"    "evaluate"     "highr"
[19] "markdown"     "yaml"         "xfun"
[22] "cli"          "pkgbuild"     "rstudioapi"
[25] "withr"        "magrittr"     "glue"
[28] "fansi"        "mime"         "callr"
[31] "prettyunits" "backports"    "processx"
[34] "ps"
> length(package.deps('roxygen2',mode='nonbase'))
[1] 34
```

Package folders and files

```
[groth@bariuke build]$ ls -R pwr2020
pwr2020:
data  DESCRIPTION  LICENSE  man  NAMESPACE  R

pwr2020/data:
tdata.rda

pwr2020/man:
add2.Rd  add.Rd  pwr2020-package.Rd  tdata.Rd

pwr2020/R:
add2.R  add.R  pwr2020-internal.R
```

Essentials

- Folder `R`: the R files with your code, multiple functions can be in the same file
- Folder `man`: The Rd manual files for your documentation, each function or dataset has its own file
- File `DESCRIPTION`: the meta information about your package (Name, Description, Author, Version, License, Dependencies (Imports, Depends))
- File `NAMESPACE`: the exported functions or data sets

Optionals

- Folder `data`: R data sets which can be loaded using the `data` command
- Folder `inst`: additional files required by your package like images, Tcl scripts etc.
- Folder `vignettes`: tutorials on how to use your package.
- File `LICENSE`: if you provide a special licsense just as the MIT license
- File `NEWS` or `ChangeLog`: a file with information on updates in your package

DESCRIPTION

```
> source('../scripts/file.head.R')
> file.head('pwr2020/DESCRIPTION',10)
Package: pwr2020
Type: Package
Title: Package for course Programming with R
Version: 0.1
Date: 2020-09-18
Author: Detlef Groth, University of Potsdam
Maintainer: Who to complain to <dgroth@uni-potsdam.de>
Description: Collection of utility functions for the cour
License: MIT + file LICENSE
```

NAMESPACE

```
> file.head('pwr2020/NAMESPACE',10)
exportPattern("^[[:alpha:]]+")
```

⇒ See: `https://www.regular-expressions.info/posixbrackets.html`

```
[:alpha:]   Alphabetic characters   [a-zA-Z]
# better export only lower case letters
[:lower:]   lowercase letters       [a-z]
```

R/add.R

```
> file.head('pwr2020/R/add.R',10)
add <-
function (x, y)
{
    x + y
}
```

man/add.Rd

```
> file.head('pwr2020/man/add.Rd',-1)
\name{add}
\title{
    add two numbers
}
\description{
    A intial starting function ...
}
\usage{
    add(x, y)
}
\arguments{
  \item{x}{
    numercial value
  }
  \item{y}{
```

```
    numerical value
  }
}
\details{
    Some more details ...
}
\value{
    return the sum of x and y
}
\author{
    Detlef Groth, University of Potsdam
}
\note{
    further notes ...
}

\seealso{
    See also % \code{\link{add2}}
}
```

```
\examples{
    add(2,3)
    x=2
    y=3
    add(x,y)
}

\keyword{ arith } % use one of  RShowDoc("KEYWORDS")
```

Minimal package: mini

Files:

- DESCRIPTION
- NAMESPACE
- LICENSE
- man/add.Rd
- R/add.R

⇒ That is a KISS approach !!

Steps in Installation

- build: `R CMD build pkgname`
- check: `R CMD check pkgname_VERSION.tar.gz`
- install: `R CMD INSTALL pkgname_VERSION.tar.gz`

Mini Build

```
#!/bin/bash
R CMD build mini |
    perl -pe 's/[^-:\*\/._A-Za-z0-9\s]+/"/g'
⇒ * checking for file "mini/DESCRIPTION" ... OK
⇒ * preparing "mini":
⇒ * checking DESCRIPTION meta-information ... OK
⇒ * checking for LF line-endings in source and make files
⇒ * checking for empty or unneeded directories
⇒ * building "mini_0.1.tar.gz"
```

Rbuild.sh

⇒ the Perl onliner just for replacing special characters which LaTeX can't compile ...

Mini Check

```
#!/bin/bash
R CMD check mini_0.1.tar.gz |
    perl -pe 's/[^-\*\/:._A-Za-z0-9\s]+/"/g' | tail
⇒ * checking Rd cross-references ... OK
⇒ * checking for missing documentation entries ... OK
⇒ * checking for code/documentation mismatches ... OK
⇒ * checking Rd "usage sections ... OK
⇒ * checking Rd contents ... OK
⇒ * checking for unstated dependencies in examples ... OK
⇒ * checking examples ... OK
⇒ * checking PDF version of manual ... OK
⇒ * DONE
⇒ Status: OK
```

Rcheck.sh

Mini INSTALL

```
#!/bin/bash
R CMD INSTALL --html mini_0.1.tar.gz \
    -l ../../../../myr/rlibs 2>&1 |
    perl -pe 's/[^-\*:\/._A-Za-z0-9\s]+/"/g'
⇒* installing *source* package "mini" ...
⇒** using staged installation
⇒** R
⇒** byte-compile and prepare package for lazy loading
⇒** help
⇒*** installing help indices
⇒  converting help for package "mini"
⇒    finding HTML links ... done
⇒    add                                    html
⇒** building package indices
⇒** testing if installed package can be loaded from temp
⇒** testing if installed package can be loaded from fina
⇒** testing if installed package keeps a record of tempo
```

⇒ `* DONE "mini"`

Rinstall.sh

⇒ redirect stderr to stdout: `2>&1`

Result

```
[groth@bariuke]$ ls ../../myr/rlibs/mini
DESCRIPTION  help  html  INDEX  LICENSE
Meta  NAMESPACE  R
```

Mini Test

```
> .libPaths(c('../../../../myr/',.libPaths()))
> library(mini)
> ls('package:mini')
[1] "add"
> mini::add(2,3)
[1] 5
> add(2,3)
[1] 5
> ?add
add                 package:mini        R Documentation

add two numbers

Description:
     A intial starting function ...
Usage:
     add(x, y)
...
```

Summary

- install.packages
- library
- require
- writing packages
- devtools/roxygen2
- minimal approach

1.4 Exercise 4 - packages

Goals:

- we will create a minimal R-package called pwr2020
- we will learn how to build and install the package
- we will add 1-2 addtional functions to the package such as `file.head(filename,n)` and `termcolor(color,txt)`
- On Windows we have to create batch files to aid in checking, building and installing.

Task 1: Setup your working environment

- start Geany
- inside your `R-labs` directory creatr two other folder:
  - `rlibs` - therein will your packages installed
  - `rlibs-dev` - therein will your packages developed
- download the `mini.zip` file from Moodle (Download folder)
- unpack this `mini.zip` file into your `rlibs-dev` folder
- so you must have a folder `rlibs-dev/mini`
- Windows:
  - download the three Batch (.bat) files
  - place those three files into the `rlibs-dev` folder
  - so you should have a file like `rlibs-dev/Rcmdbuild.bat`

Task 2: Building, Checking, Installing the mini package

- UNIX:
  - open a terminal and switch into the directory `R-labs/rlibs-dev`
  - run the command: `R CMD build mini`
  - a file `mini_0.1.tar.gz` should be generated
  - run the command `R CMD check -no-manual mini_0.1.tar.gz`
  - check if this runs ok
  - if this is the case run: `R CMD INSTALL -l ../rlibs -html mini_0.1.tar.gz`
- Windows:
  - Open a console/powershell and switch into the `R-labs/rlibs-dev` directory using the `cd` command
  - do a directory listing `ls` to check if your batch files are here

- run the command: `.\Rcmdbuild.bat mini`
- a file `mini_0.1.tar.gz` should be generated
- run the command `.\Rcmdcheck.bat mini_0.1.tar.gz`
- check if this runs ok
- if this is the case run: `.\Rcmdinstall.bat mini_0.1.tar.gz`

Task 3: Test the mini package

- lets test first the mini package
- start R
- we have to extend the library path that R finds the mini package
- below is my R session on Windows which I used to test the package

```
> .libPaths(
 c("C:/Users/Dr. Detlef Groth/Documents/Rlabs/rlibs'
 .libPaths()))
> library(mini)
> mini::add(3,4)
7
```

⇒ Adapt the path and check if you can add two numbers mini package.
⇒ Be sure to use the tab key for folder completion, don't write the long file name by Hand !!

Task 4: Create a pwr2020 package

- pwr2020 because a pwr package exists already on CRAN
- Create a copy of the `mini` folder in the same directory and rename the new folder `pwr2020`.
- You should keep the `mini` package for future creations of new packages from scratch.
- So don't change anything in the `mini` package
- In Geany open the file browser sidebar left for easier navigation (Plugins->Filebrowser)
- Open the files `pwr2020/DESCRIPTION` and `pwr/LICENSE` in Geany and change the neccessary details, author, package name etc.
- Create a new file `pwr2020/R/file_head.R` and add the code from the lecture to this file.
- Rename the function to `file_head`.
- Create a new file `pwr2020/man/file_head.Rd`
- Copy the manual from add.Rd to this file and adapt this to your

needs

- In the examples section add the following code:

```
\examples {
file_head(file.path(system.file(package='pwr2020'),
    "man",'file_head.Rd'))
}
```

- After adding the two files do again a package build, check and install.
- Test the new function in an interactive R session.

Task 5: Add a termcolor function ...

to the package and use a few standard colors. Give a default color of green.

```
termcolor <- function (msg,col='GREEN') {
    if (col=='GREEN') {
        return(cat(codegreen,msg,resetcode))
    } else if (col == 'RED') {
```

```
        etc
    }
}
```

Task 6 - Homework

- add the FastaUtils.R file from day 3 exercise to the package
- comment first only the read.fasta function
- do a stupid example x=1 just for checking
- we will add a real fasta example later

References