Contents

[general rules 1](#_Toc54629710)

[file structure 3](#_Toc54629711)

[data 3](#_Toc54629712)

[plots 3](#_Toc54629713)

[scripts 3](#_Toc54629714)

[.Rproject 3](#_Toc54629715)

[README.txt 3](#_Toc54629716)

[code & comments 3](#_Toc54629717)

[commenting, when, where and how? 4](#_Toc54629718)

[names 4](#_Toc54629719)

[*Make it easy to modify* 4](#_Toc54629720)

[further reading 5](#_Toc54629721)

[Questions to SH 5](#_Toc54629722)

[*The roxygen format* 5](#_Toc54629723)

## general rules

Writing an R script is one thing. Organizing a whole project another. Questions will arise like: Where to store the data, how to name files, how to load files in your scripts independent of the workstation you are on, and how to produce, collect and report results. It seems like a good idea, to have some profound answers!  
  
There exist a zillion different answers in the net on how to have the most efficient workflow. But all these workflows have the same general answers in common.

* **Transparency:** Objects of the project are organized logically and clearly, so that an observer has an easy way to understand: Where do I find things in the project? What does the file do, I am looking at? Informative file names, a README.txt and a logical file structure are a good approach.
* **Maintainability:** It is easy to modify and adapt the project. Standardized script names, informative commenting and outsourcing style decisions into variables are key here.
* **Modularity:** Encapsulate discrete tasks into separate components (e.g scripts), so that it is always clear where modifications need to be made and components like functions are re-usable for other projects!
* **Portability:** make it easy to move the project to another system or another person. That can be achieved by relative file names and a version control system like GitHub.
* **Reproducibility:** make it easy for you and others, to reproduce and understand the way to your results.

In the following, you will see one way, which I would recommend, to structure and code your Rprojects. But feel free to keep what suits you and modify what does not.

## file structure

When someone opens the project, they will normally have a specific question in mind. The goal should be, to answer that question as fast as possible. Therefore, it is necessary, that it is easy to find out, where to go.

### 1. data

It is always a good idea to store all datafiles in one place. This folder is this place. You’ll find two folder here

#### 1.1 raw

Distinguish between the raw files, which you should never edit…

#### 1.2 processed

…and the files you created in your analysis process.

### 2. plots

When you generate plots in your analysis, save them! In scripts >> helpers is such a function. That way, you do not have to run the whole script anew, to look up one plot.

### 3. scripts

Do not have all your analysis in one script! Try to write scripts for specific purposes like: getting to know the data, RSA Analysis on IQ vs other variables and so on.

#### 3.1 helpers

Always outsource code you use more than two times into functions. Store each of them in an individual R Script. That keeps your code clearly arranged and saves work if you want to change that chunk of code for all application cases.

### nameofproject.Rproject

Always use an Rproject file. That is the basis for relative file names and sets your working directory to where the .RProject file is!

### README.txt

Never write a readme as .docx. Only as .txt! That way everybody can open and read it without formatting issues. A README should:  
- briefly explain what the project is about.  
- contain a table of contents / overview of the files. That makes finding the things you search for even easier.

## code & comments

Each connected block of **code should not exceed a single screen** (about 50 lines of code). This allows to be read and understood without having to scroll. Exceeding the limit is a good sign, that some of the code should be outsourced in a function.

Each line of code should not exceed **80 characters per line**. This fits on a printed page and therefore can be easily processed by a human.   
Alternative: Write a long comment in 1 line, that way your code is easy to read, and press Strg+Shift+/ to break and read it.

loading & saving data

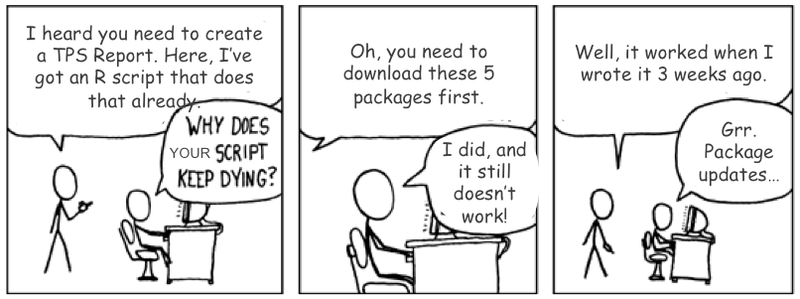
Use an .Rproject file and relative file paths, like in this example:  


Save all (intermediate) results. Be it plots, edited data, or result tables. Use the save() or saveplot() functions.

### Random data

Results you cannot easily reproduce are values created by rng functions. Well its rng right? But if you use set.seed( *insert a number of your choice*), you make sure to reproduce and save the same random data every time.

### Packages



Use the checkpoint() package. This package downloads the currently used versions of your packages and stores them in your file. This way everyone has automatically the same version of the packages you used.

commenting, when, where and how?  
  
There are two parts of commenting:  
First comments to structure your file into understandable chunks of code and second commenting to help understand what happens in your code. You will find a pre-structured file in scripts//. Therefore, we will focus on the second part.  
  
Document everything, with roxygen2 or a consistent self-annotation! Comment whenever something is not self-explanatory. Always comment above or on the same line, of what is to be commented.  
When commenting describe how and why stuff happens. What happens should be clear by reading the headers and the filenames.

#### The roxygen format

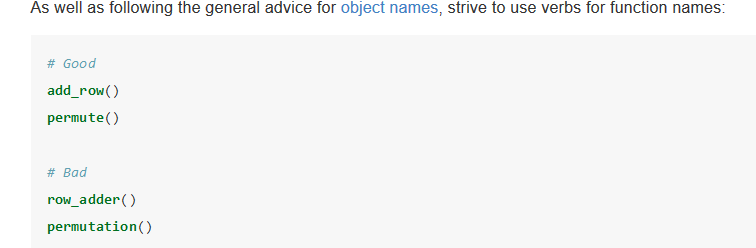
This format is a commenting style designed for function headers. You find an example at script/helpers. Use #’ to call the roxygen commenting style. The following commands are available

* @title: short sentence of what the code does
* @description: extended description of the code, if necessary
* @details: alternative to @description
* @param: which datatype is the parameter? What is it for?
* @return: describes the output of the code
* @examples: Example code that shows the code working.

### names

Choose intuitive names for functions and data.

* A function should always be named after what it does. Use verbs for function names!



* A variable should be named after what it contains.

### Make it easy to modify

Let us say you have a script which plots 20 plots. Now you want to change the background of all plots to white. Comes in handy you placed a variable called *backgroundcolor\_plots* added to each plot. Now you just must change only the value of that variable.   
That goes with other style decisions too. If possible**, make it a variable or a function!**

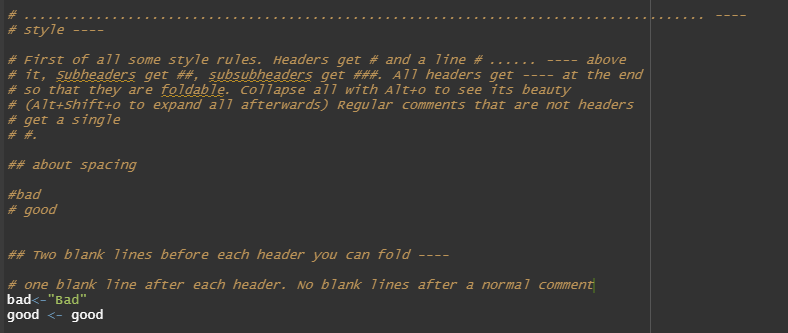
## R – Commands

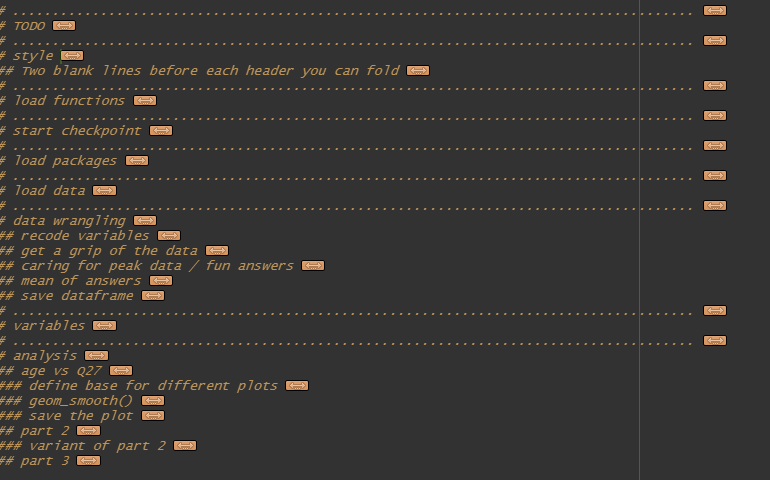
|  |  |
| --- | --- |
| Strg + Shift + C | Comment/uncomment all chosen lines at once |
| Alt + O | Collapse all folders |
| Alt + Shift + O | Unfold all folders |
| Strg + Shift + / | Break long line of comment |
|  |  |

### Style Guide

If you use Tools -> Global Options -> Appearances -> Editor Theme   
you can change the look of your R Studio. Do your eyes a favor and change it. The one below is called Idle Fingers

Übergangsdoku:





## 

## further reading

<https://www.r-statistics.com/2010/09/managing-a-statistical-analysis-project-guidelines-and-best-practices/>

<https://blog.revolutionanalytics.com/2010/10/a-workflow-for-r.html>

<https://chrisvoncsefalvay.com/2018/08/09/structuring-r-projects/>

[*https://style.tidyverse.org/*](https://style.tidyverse.org/)

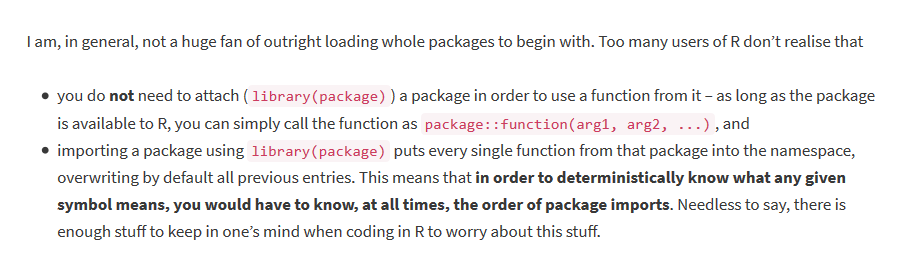
*https://www.r-bloggers.com/%F0%9F%96%8A-r-coding-style-guide/*

<https://rviews.rstudio.com/2016/12/02/writing-good-r-code-and-writing-well/>

https://www.r-bloggers.com/2014/10/introducing-the-reproducible-r-toolkit-and-the-checkpoint-package

### Questions to SH

SF@SH Als Praxis übernehmen? Ist hier bei dir schonmal was schief gelaufen?



SF@SH das hier noch dazu, oder lieber nicht?

### *The roxygen format*

This format is a commenting style designed for function headers. Use   
#’ to call it.  
the following commands are available

* @title: short sentence of what the code does
* @description: extended description of the code, if necessary
* @details: alternative to @description
* @param: which datatype is the parameter? What is it for?
* @return: describes the output of the code
* @examples: Example code that shows the code working.

# TODO? Ideen ab dem 29.10

Eventuell Leerzeichen und Formatierungsguide anfügen oder verlinken?  
x=x -> x= x