

# R packages & Debugging



Berry Boessenkool, [uni-potsdam.de](http://uni-potsdam.de) &  
Marvin Reich, [gfz-potsdam.de](http://gfz-potsdam.de)  
May 2017

[berry-b@gmx.de](mailto:berry-b@gmx.de)  
[github.com/brry](https://github.com/brry)

[swc-bb.github.io/2017-05-17-r-workshop](http://swc-bb.github.io/2017-05-17-r-workshop)

*Presentation template generated with `berryFunctions::createPres`*

## Why R packages

Marvins Stuff here...

Good introductions linked here: [github.com/brry/misc](https://github.com/brry/misc)

## Devtools for package structure creation

```
dir.create("subfolder") # optional
setwd("subfolder")
library(devtools)

create("packageName")
setup() # if repos is initialized on github, see Appendix

check()
```

### Exercise 1: create the package structure

- ▶ Create a package for the linear storage cascade functions we provide.
- ▶ BONUS: Use git from the beginning (see Appendix)
- ▶ Check the package
- ▶ Now add the functions in [bit.ly/swc\\_rfun](https://bit.ly/swc_rfun) to the R folder

## Solution to Exercise 1: create the package structure

```
devtools::create("lsc")  
devtools::check()
```

## Roxygen for Documentation

Rstudio - Code - Insert Roxygen Skeleton (CTRL + ALT + SHIFT + R)

```
devtools::document() # calls roxygen2::roxygenise()
```

```
check() # will also update documentation
```

```
install() # install the package locally
```

### Exercise 2: Document functions

- ▶ Document the nse function (we'll do this one together)
- ▶ Create datasets within the package (following instructions in data.R)
- ▶ Document the rmse function
- ▶ Check and install the package

## Debugging: useful functions

<code>source("projectFuns.R")</code>	execute complete file
<code>traceback()</code>	find error source in sequence of function calls
<code>options(warn=2)</code>	warnings to error. default 0
<code>browser()</code>	go into function environment: <code>n</code> , <code>s</code> , <code>f</code> , <code>c</code> , <code>Q</code>
<code>options(error=recover)</code>	open interactive session where error occurred
<code>debug(func)</code>	toggle linewise function execution
<code>undebug(func)</code>	after calling and fixing func

```
if(length(input)>1) stop("length must be 1, not ", length(input))
```

`stop`: Interrupts function execution and gives error

`warning`: continues but gives warning

`message`: to inform instead of worry the user

R. Peng (2002): Interactive Debugging Tools in R

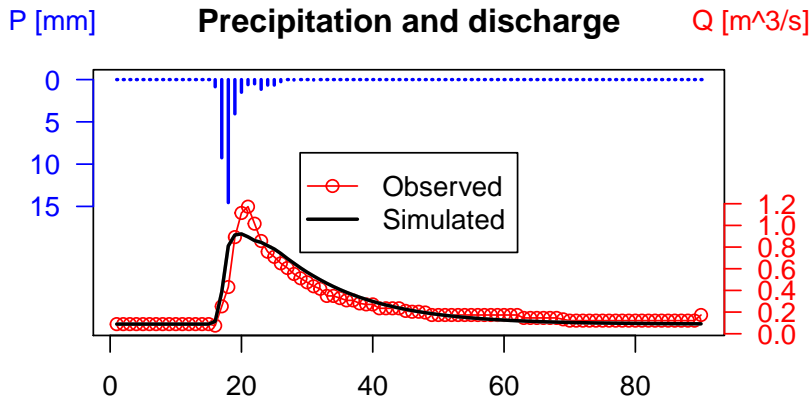
D. Murdoch (2010): Debugging in R

H. Wickham (2015): Advanced R: debugging

Example: Pete Werner Blog Post (2013)

## Exercise 3: Debugging

- ▶ Load your package and the datasets. Correct the functions until `lsc(calib$P, calib$Q, area=1.6)` returns the result below.
- ▶ BONUS: commit each change to git.



## Solutions to Exercise 3: Debugging

- ▶ **stupid error you can easily remove** - traceback - find location of error - lsc#75 - just comment it out
- ▶ **harder to find but still stupid** - traceback - nse#22 - ditto
- ▶ **Error in plot: need finite 'ylim' value** - debug/browser/options(error=recover) - lsc#107 - NAs in Q - range(Q, na.rm=TRUE) - also in other applicable locations
- ▶ **There were 50 or more warnings** - come from rmse being called in optimization - add argument quietNA (or similar) to lsc that is passed to rmse in lsc#81



## Version control with github

- ▶ Connect git to Rstudio ([kbroman.org/github\\_tutorial](http://kbroman.org/github_tutorial))
- ▶ github.com, log in, on "+" in top-right select "New repository", initialize with readme!
- ▶ clone or download: copy link
- ▶ Rstudio - file - new project - version control - git: paste url and create
- ▶ `devtools::setup()`, choose option number to Overwrite 'mhmVis.Rproj'
- ▶ Work. Commit. Push. Repeat.

## More things

