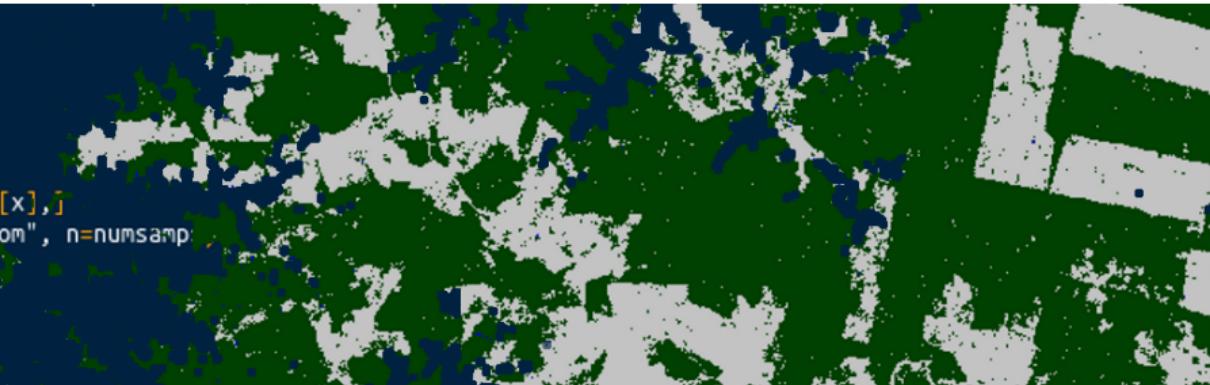




```
allAtt <- slot(vec, "data")
tabAtt <- table(allAtt[[attName]])
uniqueAtt <- as.numeric(names(tabAtt))

for (x in 1:length(uniqueAtt)) {
  class_data <- vec[vec[[attName]]==uniqueAtt[x],]
  classpts <- spsample(class_data, type="random", n=numsamp)
  if (x == 1) {
    xy <- classpts
  } else {
    xy <- rbind(xy, classpts)
  }
}
```



January 21, 2019
package development from A to Z



- good speed
- change detection overview and practice
- good explanations
- enough time for tasks
- good individual help
- code tipps



- good speed
- change detection overview and practice
- good explanations
- enough time for tasks
- good individual help
- code tipps

- too fast
- more time for tasks
- not useful to create training data in the course
- more explanations about the data is needed



building an actual R package



What kind of package do we want?

We like to develop a package that

- can deal with time-series stacks (provided by `getSpatialData` or other sources as stacks)



What kind of package do we want?

We like to develop a package that

- can deal with time-series stacks (provided by `getSpatialData` or other sources as stacks)
- improve time-series stacks (interpolation, handling missing values, ...)



What kind of package do we want?

We like to develop a package that

- can deal with time-series stacks (provided by `getSpatialData` or other sources as stacks)
- improve time-series stacks (interpolation, handling missing values, ...)
- computes temporal metrics (mean, max, start of season, end of season, amplitude, ...)



What kind of package do we want?

We like to develop a package that

- can deal with time-series stacks (provided by `getSpatialData` or other sources as stacks)
- improve time-series stacks (interpolation, handling missing values, ...)
- computes temporal metrics (mean, max, start of season, end of season, amplitude, ...)
- queries the time stack behind vector objects (point/polygon)



What kind of package do we want?

We like to develop a package that

- can deal with time-series stacks (provided by `getSpatialData` or other sources as stacks)
- improve time-series stacks (interpolation, handling missing values, ...)
- computes temporal metrics (mean, max, start of season, end of season, amplitude, ...)
- queries the time stack behind vector objects (point/polygon)
- visualizes/animations temporal dynamics



What exactly do we need to achieve that?

- sound understanding of user needs
- good structure how all functions interact
- clear definition of needed internal (invisible) and user (visible) functions
- providing enough flexibility for the user (e.g. maps with graphs flexible arrangement)



Brainstorming



Brainstorming

- ➊ Preparation: What should the package do? What user functions do we want to develop?



Brainstorming

- ① Preparation: What should the package do? What user functions do we want to develop?
- ② Function design: What are the inputs and outputs of the functions, we want to implement?



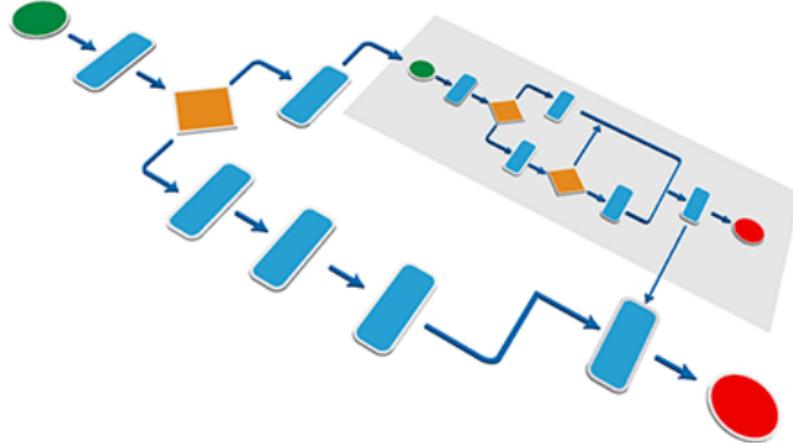
Brainstorming

- ① Preparation: What should the package do? What user functions do we want to develop?
- ② Function design: What are the inputs and outputs of the functions, we want to implement?
- ③ Internal functions: Which tasks will be relevant for multiple of the user functions?



Brainstorming

- ① Preparation: What should the package do? What user functions do we want to develop?
- ② Function design: What are the inputs and outputs of the functions, we want to implement?
- ③ Internal functions: Which tasks will be relevant for multiple of the user functions?
- ④ Task distribution: Who develops which function? Teams or solo depending on the function



Workflow



Workflow

- ➊ Fork the package skeleton from <https://github.com/16EAGLE/rabbiTS>



Workflow

- ① Fork the package skeleton from <https://github.com/16EAGLE/rabbiTS>
- ② Create a local git repository from your github fork (recommended using RStudio)



Workflow

- ① Fork the package skeleton from <https://github.com/16EAGLE/rabbiTS>
- ② Create a local git repository from your github fork (recommended using RStudio)
- ③ Work on your local git repository (your RStudio project)



Workflow

- ① Fork the package skeleton from <https://github.com/16EAGLE/rabbiTS>
- ② Create a local git repository from your github fork (recommended using RStudio)
- ③ Work on your local git repository (your RStudio project)
- ④ Add, commit, push your local changes to your github fork (e.g. using RStudio)



Workflow

- ① Fork the package skeleton from <https://github.com/16EAGLE/rabbiTS>
- ② Create a local git repository from your github fork (recommended using RStudio)
- ③ Work on your local git repository (your RStudio project)
- ④ Add, commit, push your local changes to your github fork (e.g. using RStudio)
- ⑤ Merge your changes back to the main respository using a Pull Request on GitHub



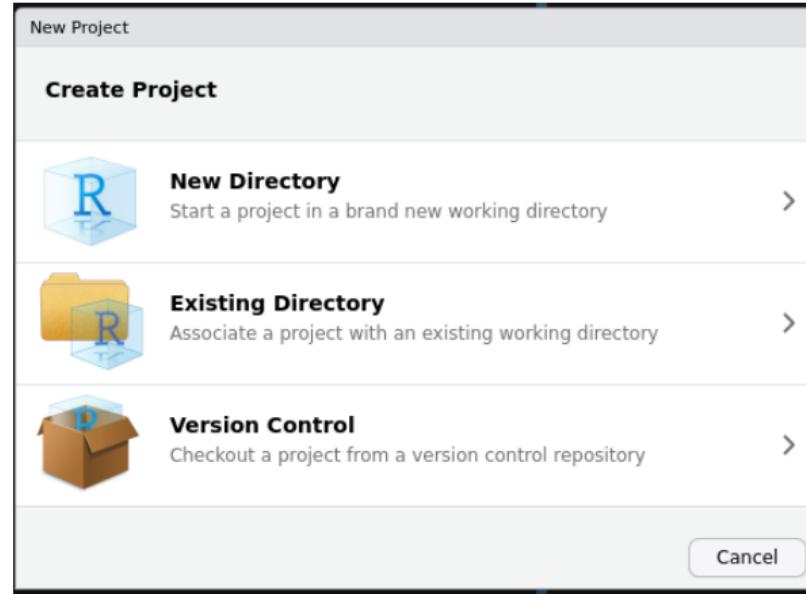
Getting Started: To do, before we continue:

Task:

- ① Login to your GitHub account
- ② Go to <https://github.com/16EAGLE/rabbiTS>
- ③ Create a forked repository of rabbiTS by clicking fork
- ④ Check out your fork on <https://github.com/username/rabbiTS>

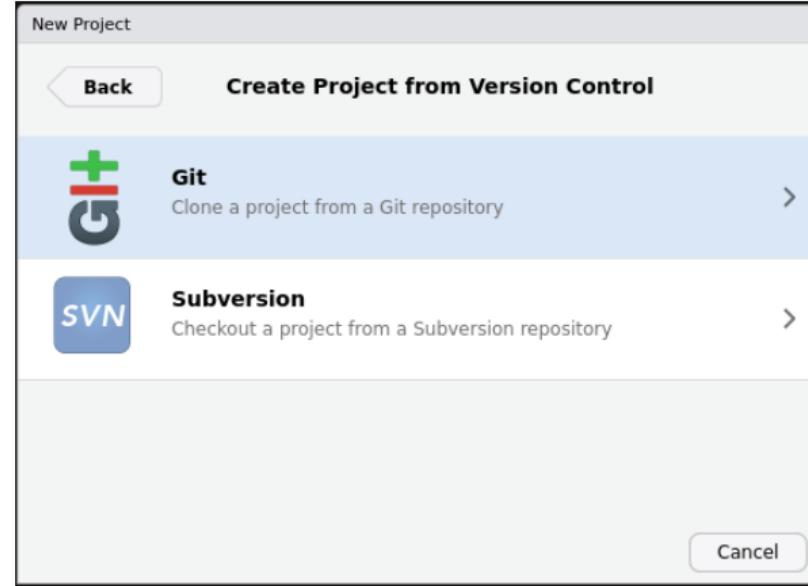


Getting started - Creating a git-linked project in RStudio





Getting started - Creating a git-linked project in RStudio





Getting started - Creating a git-linked project in RStudio

New Project

[Back](#)

Clone Git Repository



Repository URL:
`https://github.com/<your_username>/rabitTS`

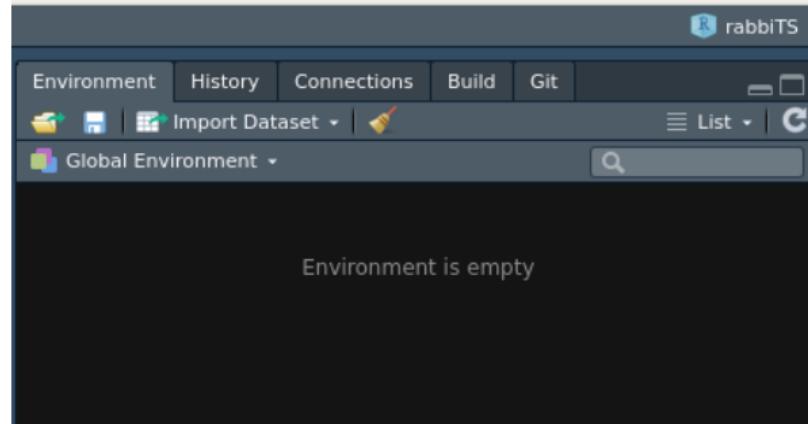
Project directory name:
`rabitTS`

Create project as subdirectory of:
`~/Documents/dev` [Browse...](#)

Open in new session [Create Project](#) [Cancel](#)

Getting started - Creating a git-linked project in RStudio

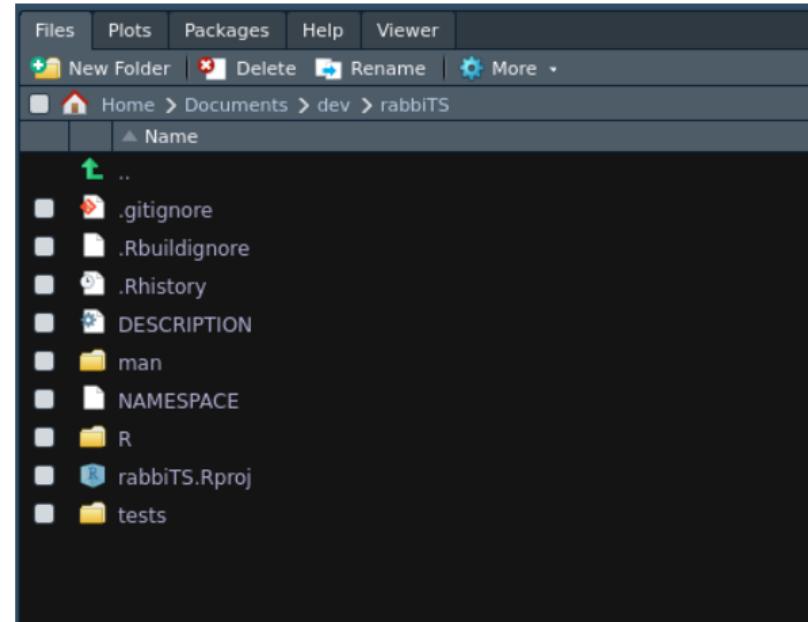
- If the system requirements have been installed correctly before today's session, you should now see five tabs, including "build" and "git":



- If this is not the case, you still can manually download your forked repository and later manually upload all files you created or changed
- This is not recommended, since you and your project participants have no track about changes made to the project.

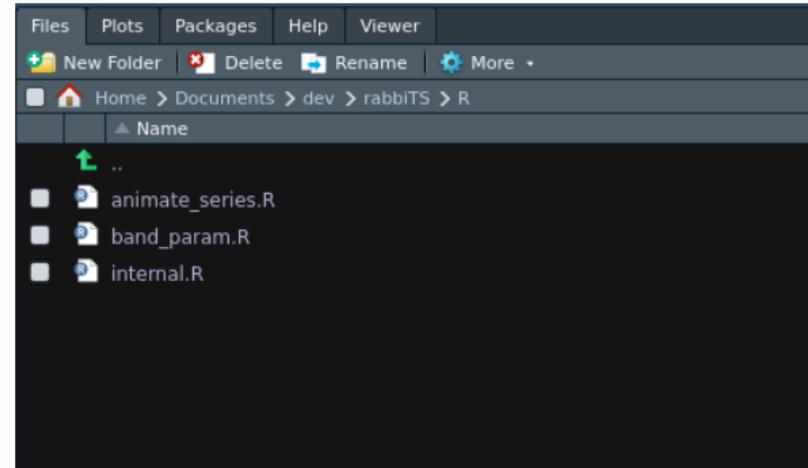
Getting started - Files

In this session, only the DESCRIPTION file and the files inside the “R” directory should be manipulated manually, everything else will be managed by devtools and roxygen for us.





Getting started - Files





Getting started - Files

A screenshot of an RStudio interface showing a file named 'dummy.R'. The code in the file is a template for creating a new R function. It includes documentation blocks (#'), parameter descriptions (@param), imports (@importFrom), and a function definition. The code is numbered from 1 to 33.

```
dummy.R x
1 #' Dummy function
2 #
3 #' A short description of your function, for example: \code{dummy} is a dummy function,
4 #' serving as a template for you to create new functions. Just copy this file into the
5 #' R directory of this package, rename it to the name of your function and start changing
6 #' it, including this text.
7 #
8 #' @param x numeric, a number of your choice.
9 #' @param y numeric, a number to be multiplied.
10 #' @param additive logical, if \code{TRUE}, x and y are added, else they are subtracted..
11 #
12 #' @return Describe, what is returned by your function, for example: a numeric value \code{z}.
13 #
14 #' @importFrom raster raster extract projectRaster
15 #' @importFrom rasterVis levelplot
16 #' @importFrom ggplot2 ggplot aes geom_line geom_area theme_bw theme scale_y_continuous scale_x_continuous
17 #
18 #' @export
19
20 dummy <- function(x, y, additive){
21
22     # here you start coding, for example, start with sum checks:
23
24     if(!inherits(x, "numeric")) cat("Argument x should be a 'numeric' object.\n")
25     if(!inherits(y, "numeric")) cat("Argument y should be a 'numeric' object.\n")
26     if(!inherits(additive, "logical")) cat("Argument additive should be a 'logical' object.\n")
27
28     # after some user input checks, you could continue, e.g.:
29     if(isTRUE(additive)) z <- x+y else z <- x-y
30
31     # when finished, return the result
32     return(z)
33 }
```



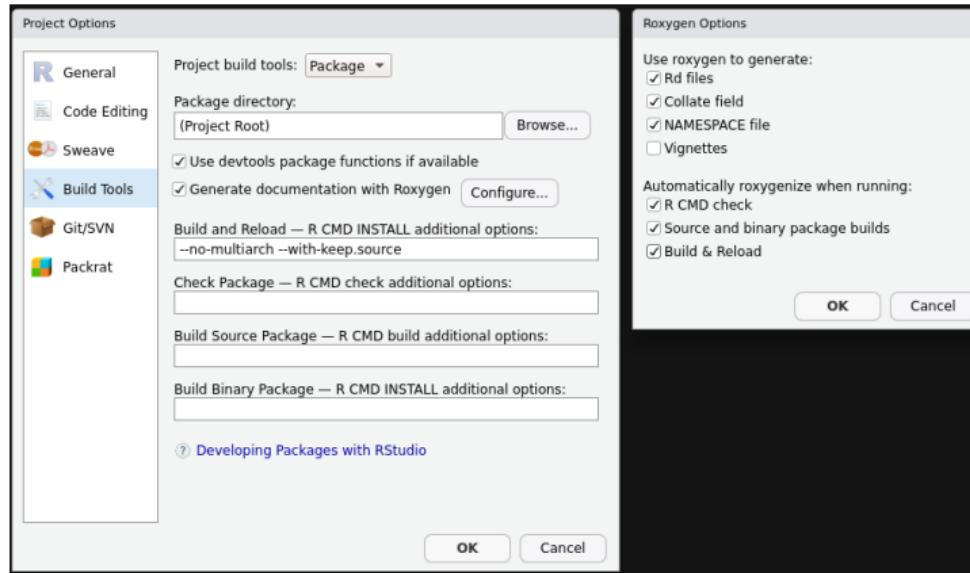
Building the package

Packages we need for building rabbiTS:

```
1 # necessary
2 install.packages(c("devtools", "usethis", "roxygen2"))
3
4 # nice to have, if we want to deploy a website for our package, when finished
5 install.packages("pkgdown")
```

Building the package

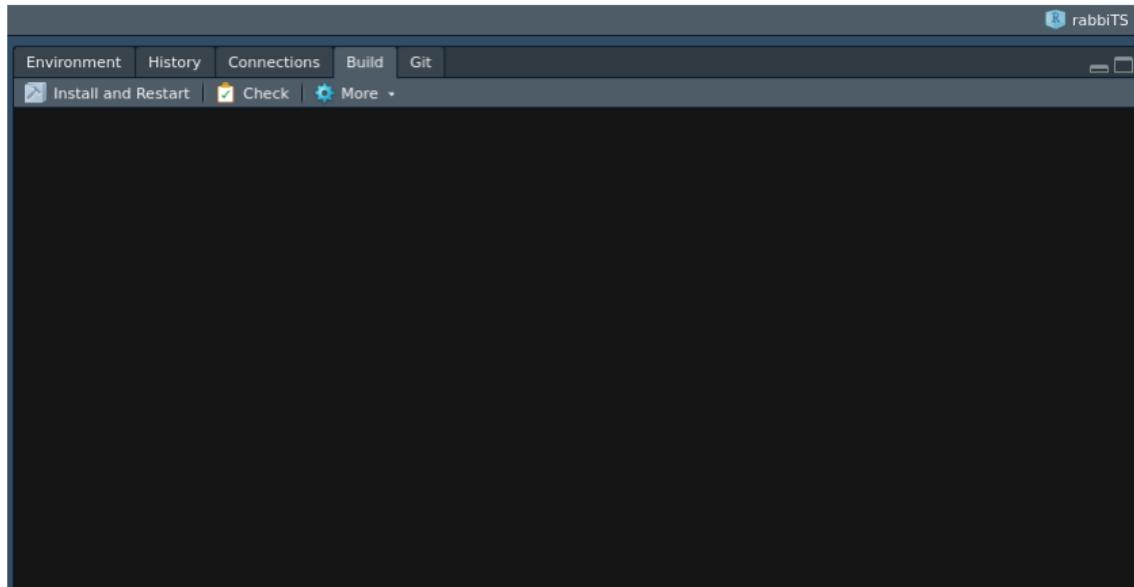
- In RStudio, go to Tools >Project Options >Build Tools and mark “Generate documentation with Roxygen”
- Click "Configure" and select "Build & Reload"





Building the package

After writing your function(s), build/rebuild the package to check, if it works by clicking "build and restart" or "install and restart":





Building the package

After writing your function(s), build/rebuild the package to check, if it works by clicking "build and restart" or "install and restart":

A screenshot of the RStudio interface. The top menu bar shows tabs for Environment, History, Connections, Build, and Git. The Build tab is active. Below the menu is a toolbar with icons for Install and Restart, Check, and More. The main pane displays the command-line output of the package build process.

```
==> devtools::document(rocllets=c('rd', 'collate', 'namespace'))  
  
Updating rabbiTS documentation  
Writing NAMESPACE  
Loading rabbiTS  
Writing NAMESPACE  
Documentation completed  
  
==> R CMD INSTALL --no-multiarch --with-keep.source rabbiTS  
  
* installing to library '/home/UNI-WUERZBURG.EU/jas24nx/R/x86_64-pc-linux-gnu-library/3.5'  
* installing *source* package 'rabbiTS' ...  
** R  
** byte-compile and prepare package for lazy loading  
** help  
*** installing help indices  
** building package indices  
** testing if installed package can be loaded  
* DONE (rabbiTS)
```



Add, commit and push

As a reminder:

- Add: locally stages the selected files for being committed
- Commit: locally commits the changes annotated with a message, so that the changes are now tracked within the local git repository. Changes made to files not added and committed are not tracked and will not be pushed to GitHub.
- Push: the local changes are pushed to the connected GitHub repository



Add, commit and push

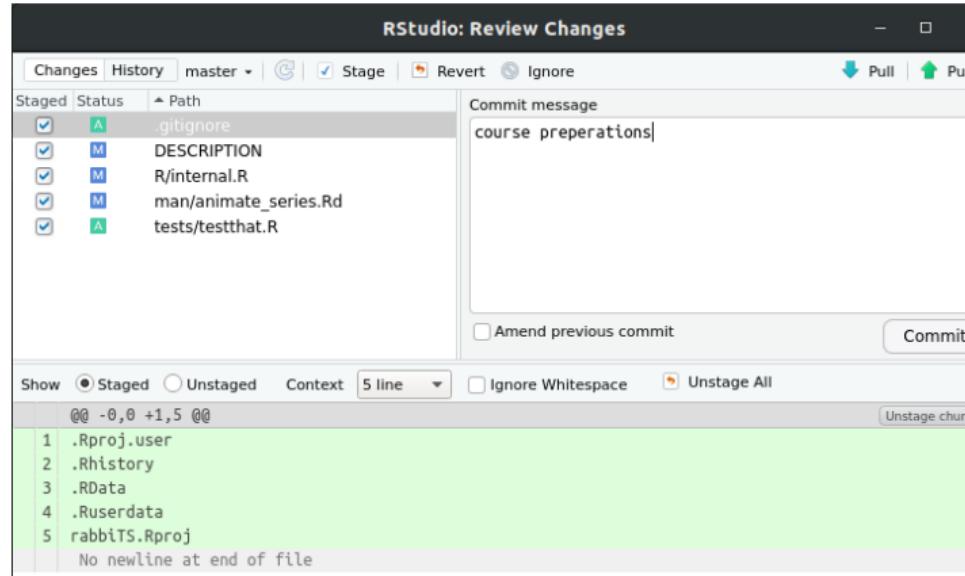
Adding (stage = add) files in RStudio (your local repository):

A screenshot of the RStudio interface, specifically the Git tab. The top navigation bar includes tabs for Environment, History, Connections, Build, and Git. Below the navigation bar is a toolbar with icons for Diff, Commit, Pull, Push, History, and More. The main area shows a table of staged files. The columns are Staged, Status, and Path. There are five files listed: .gitignore (Status A), DESCRIPTION (Status M), R/internal.R (Status M), man/animate_series.Rd (Status M), and tests/testthat.R (Status A).

Staged	Status	Path
<input checked="" type="checkbox"/>	A	.gitignore
<input checked="" type="checkbox"/>	M	DESCRIPTION
<input checked="" type="checkbox"/>	M	R/internal.R
<input checked="" type="checkbox"/>	M	man/animate_series.Rd
<input checked="" type="checkbox"/>	A	tests/testthat.R

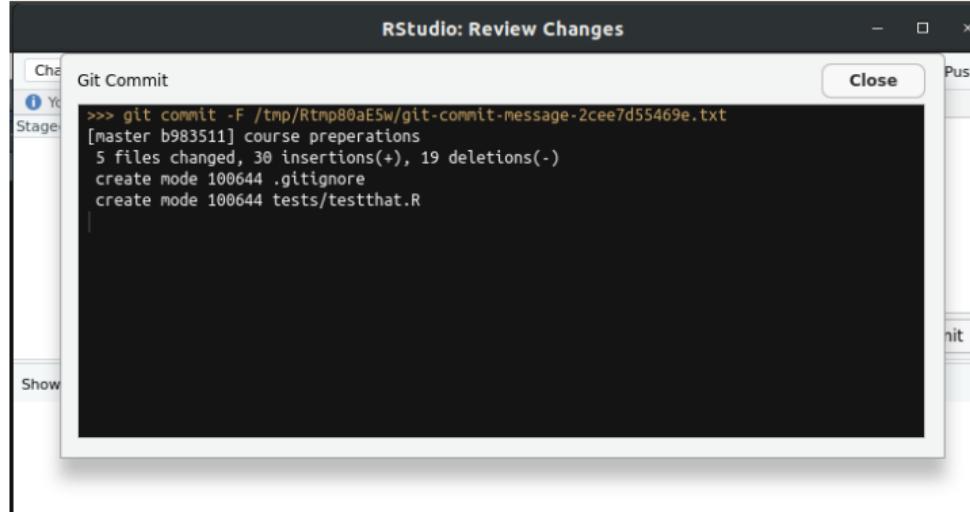
Add, commit and push

Committing changes in RStudio (your local repository):



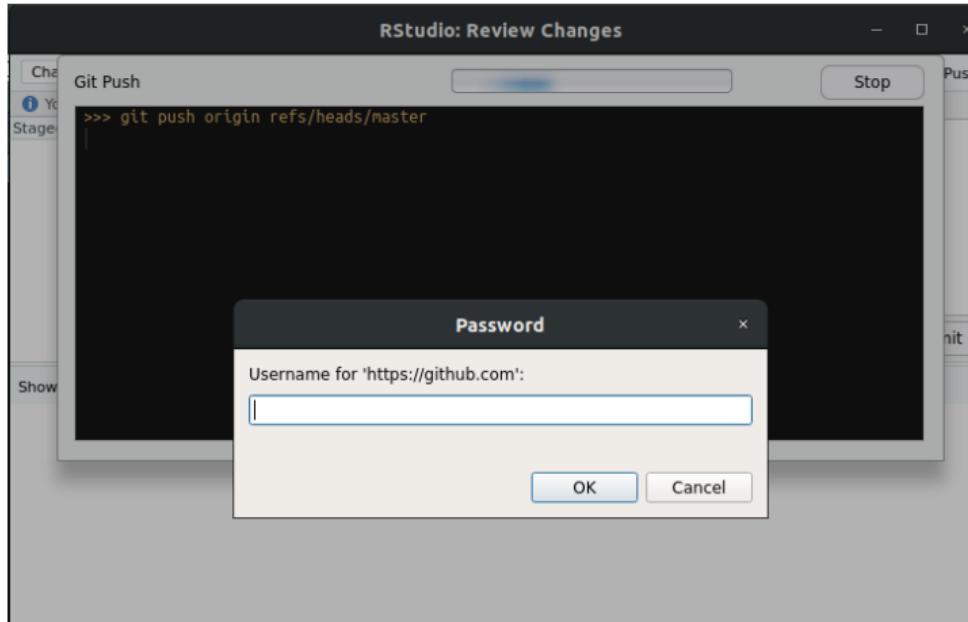
Add, commit and push

Committing changes in RStudio (your local repository):



Add, commit and push

Pushing commits in RStudio (your local repository) to GitHub:





Finally, the pull request

“Pull request” on GitHub to propose your changes to the maintainer of the original repository for merging:

A screenshot of a GitHub repository page for '16EAGLE / rabbits'. The top navigation bar shows 'Code', 'Issues 0', 'Pull requests 0' (which is the active tab), 'Projects 0', and 'Insights'. To the right are buttons for 'Watch 0', 'Star 0', 'Fork 1', and a green 'New pull request' button. Below the navigation is a search bar with the query 'is:pr is:open', a 'Labels' button, and a 'Milestones' button. The main content area is titled 'Welcome to Pull Requests!' and contains the text: 'Pull requests help you collaborate on code with other people. As pull requests are created, they'll appear here in a searchable and filterable list. To get started, you should [create a pull request](#).'

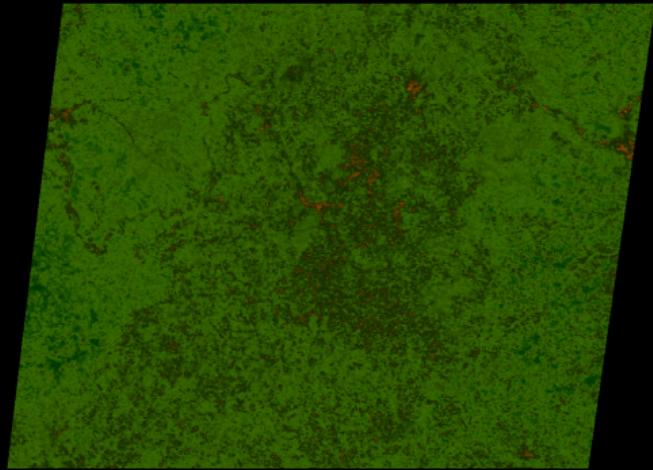
16EAGLE / rabbits

Code Issues 0 Pull requests 0 Projects 0 Insights

is:pr is:open Labels Milestones New pull request

Welcome to Pull Requests!

Pull requests help you collaborate on code with other people. As pull requests are created, they'll appear here in a searchable and filterable list. To get started, you should [create a pull request](#).



Data for today's session



Data for today's session

- Session data: MODIS NDVI time series (2014-2016) for the Wuerzburg region
- Download link: https://github.com/16EAGLE/AUX_data/raw/master/data.zip
- Writing code can be much easier when an examplatory dataset can be used to test what the code does instead of doing it fully theoretical
- For example, you could start writing your code as a simple script to make it easier to test, if it works with the data and later integrate it into a function



Questions, before we start?

```
83 #extract temporal boundaries per year
84 tb = list()
85   if(sYear == eYear) {tb[[1]] = c(sDoy,eDoy)}
86   if(sYear != eYear) {
87     for(y in 1:nYears){
88       if(y != nYears) {
89         if((sYear+(y-1) %% 4) > 1) {nDays = 366} else {nDays = 365}    #number of days in the year
90         if(y == 1) {tb[[1]] = c(sDoy,nDays)}
91         if(y > 1) {tb[[length(tb)+1]] == c(0,nDays)}
92         }
93         if(y == nYears) {tb[[length(tb)+1]] = c(0,as.numeric(eDoy))}
94       }
95     }
```

Time to code

A close-up photograph of an elephant's face, focusing on the left side. The skin is dark grey and textured with wrinkles. A small, dark fly is perched on the elephant's eye. The background is blurred green foliage.

lunch time