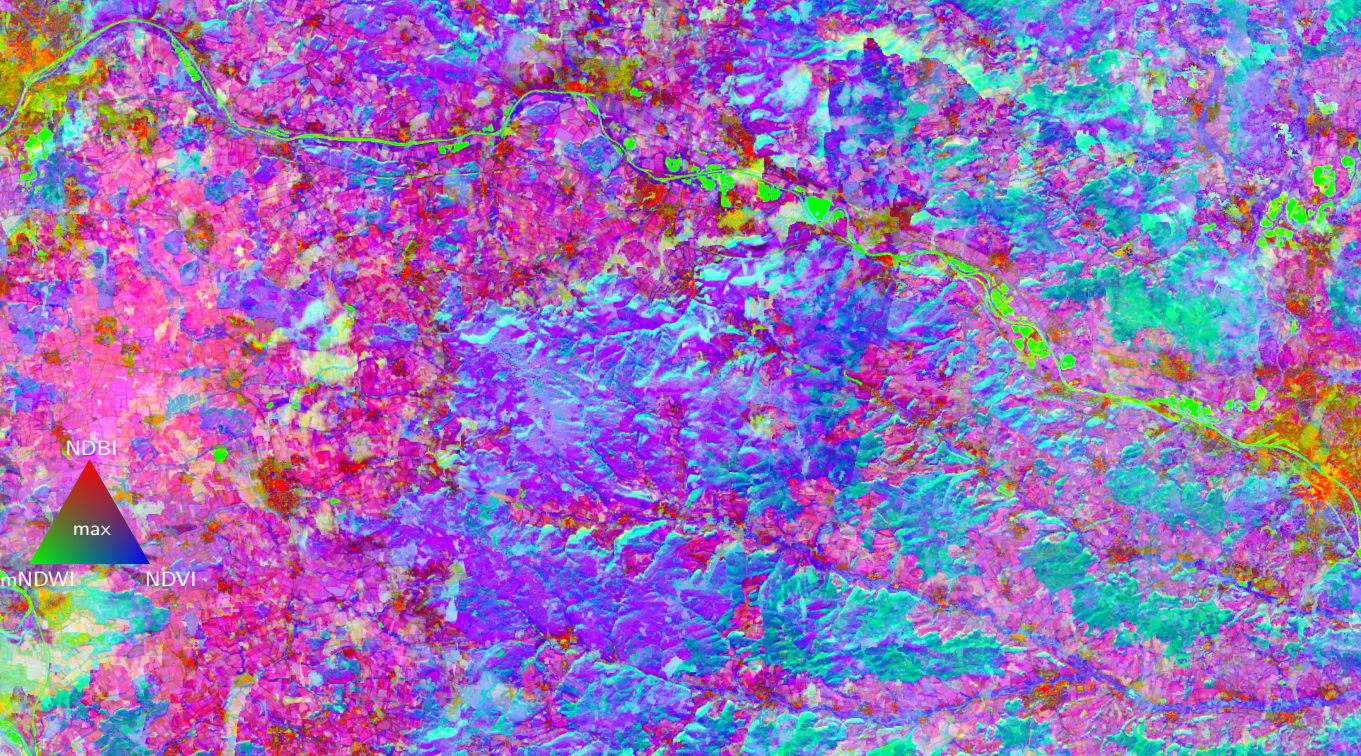
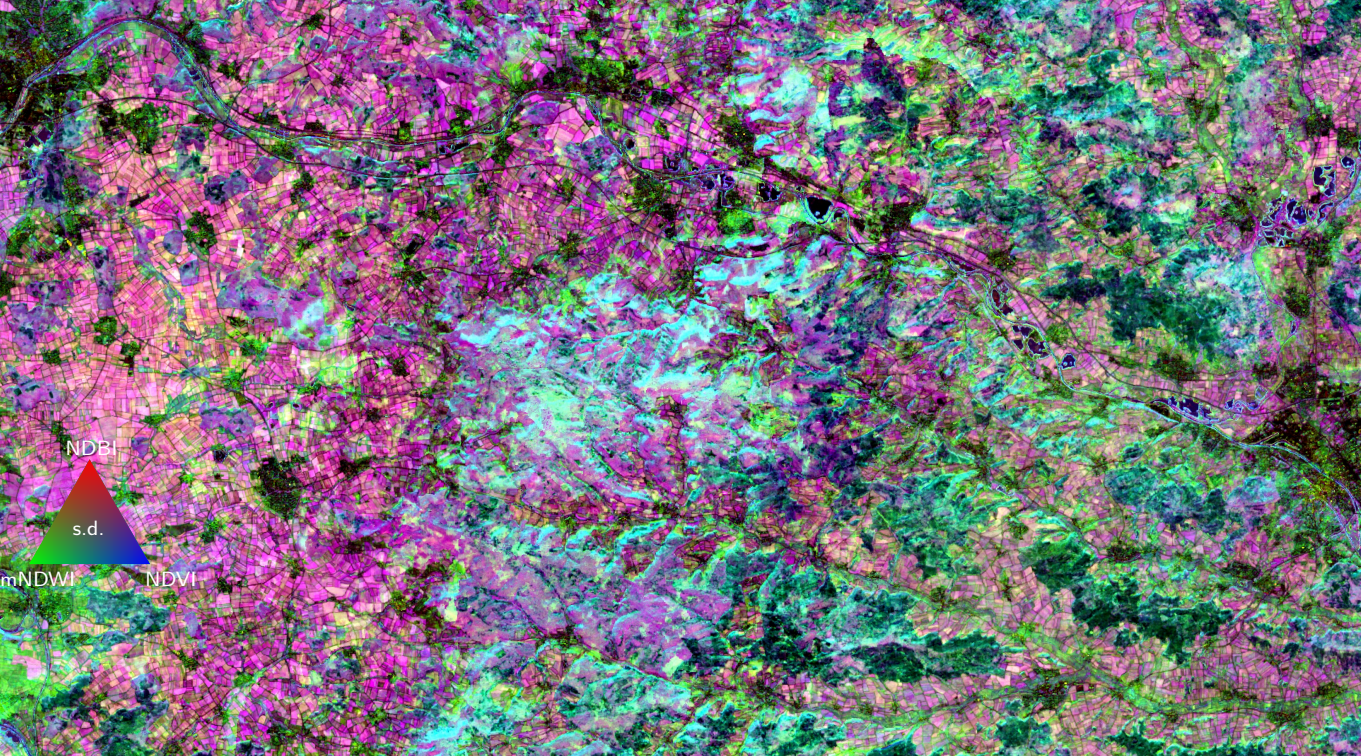
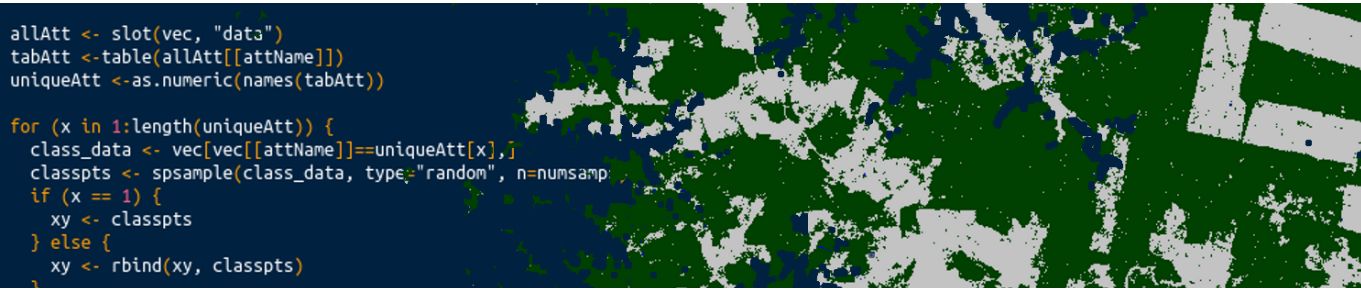

NDBI
max
mNDWI
NDVI


NDBI
s.d.
mNDWI
NDVI



January 21, 2019

package development from A to Z

MB2 - Programming and Geostatistics

feed-back

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

feed-back

- 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/animates 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

1. Preparation: What should the package do? What user functions do we want to develop?

Brainstorming

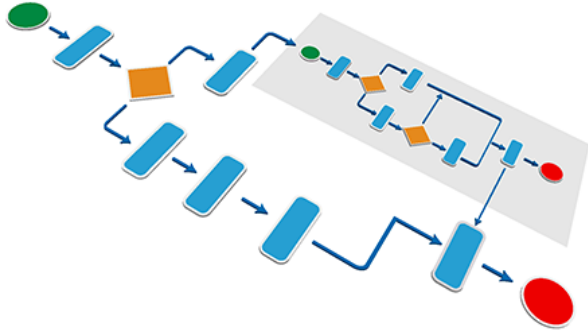
1. Preparation: What should the package do? What user functions do we want to develop?
2. Function design: What are the inputs and outputs of the functions, we want to implement?

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

1. 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)

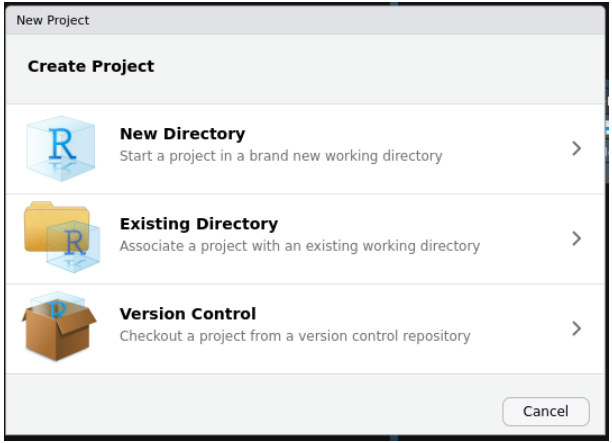
Workflow

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

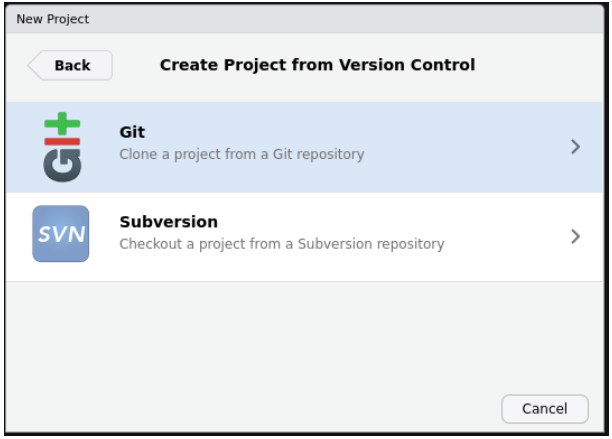
Task:

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

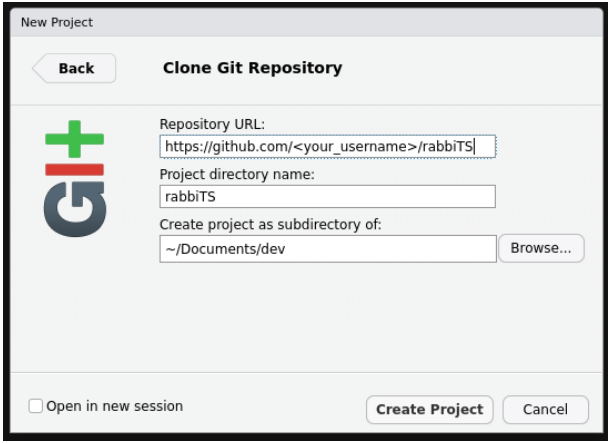
Getting started - Creating a git-linked project in RStudio



Getting started - Creating a git-linked project in RStudio

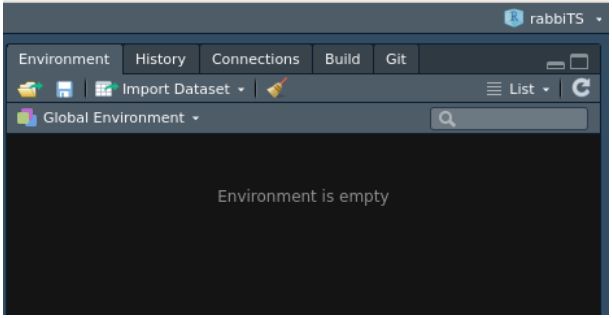


Getting started - Creating a git-linked project in RStudio



Getting started - Creating a git-linked project in RStudio

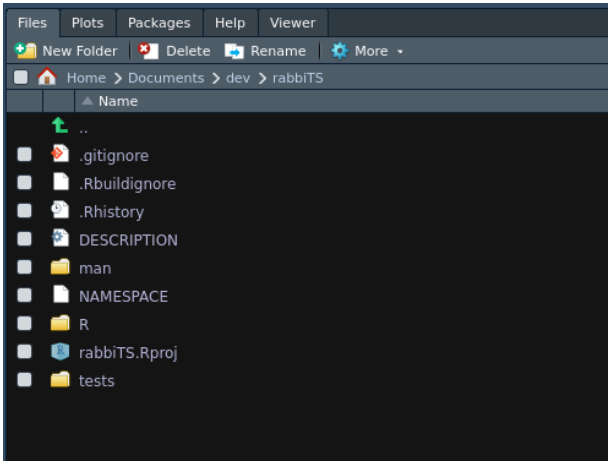
- If the system requirements have been installed correctly before today's session, you should now see five tabs, including "build" and "git":



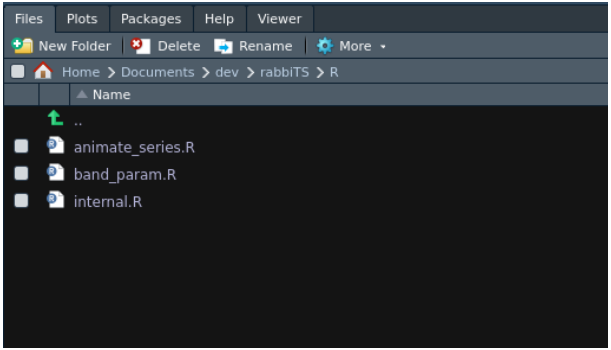
- If this is not the case, you still can manually download your forked repository and later manually upload all files you created or changed
- This is not recommended, since you and your project participants have no track about changes made to the project.

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

dummy.R

Source on Save

```r
#' Dummy function
#'
#' A short description of your function, for example: \code{dummy} is a dummy function,
#' serving as a template for you  to create new functions. Just copy this file into the
#' R directory of this package, rename it to the name of your function and start changing
#' it, including this text.
#'
#' @param x numeric, a number of your choice.
#' @param y numeric, a number to be multiplied-
#' @param additive logical, if \code{TRUE}, x and y are added, else they are substracted..
#'
#' @return Describe, what is returned by your function, for example: a numeric value \code{z}.
#'
#' @importFrom raster raster extract projectRaster
#' @importFrom rasterVis levelplot
#' @importFrom ggplot2 ggplot aes geom_line geom_area theme_bw theme scale_y_continuous scale_x_continuous
#'
#' @export

dummy <- function(x, y, additive){

  # here you start coding, for example, start with sum checks:

  if(!inherits(x, "numeric")) cat("Argument x should be a 'numeric' object.")
  if(!inherits(y, "numeric")) cat("Argument y should be a 'numeric' object.")
  if(!inherits(additive, "logical")) cat("Argument additive should be a 'logical' object.")

  # after some user input checks, you could continue, e.g.:
  if(isTRUE(additive)) z <- x+y else z <- x-y

  # when finished, return the result
  return(z)
}
```

Building the package

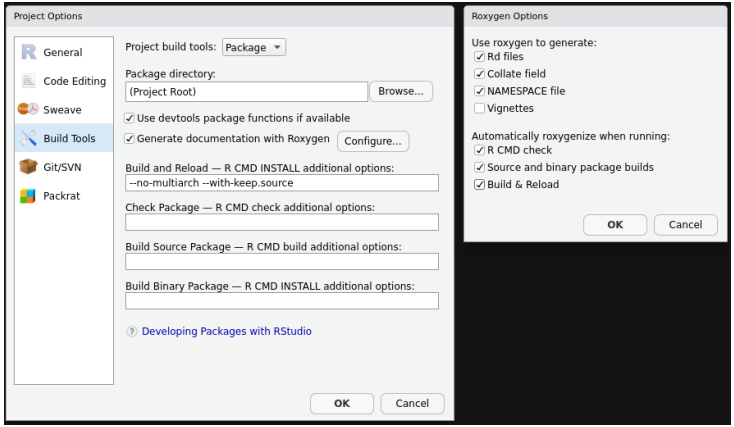
Packages we need for building rabbiTS:

```
# necessary
install.packages(c("devtools", "usethis", "roxygen2"))

# nice to have, if we want to deploy a website for our package, when finished
install.packages("pkgdown")
```

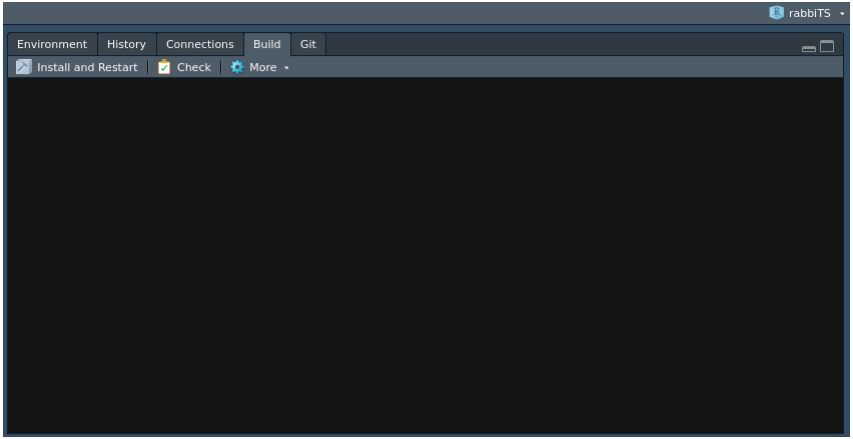
Building the package

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



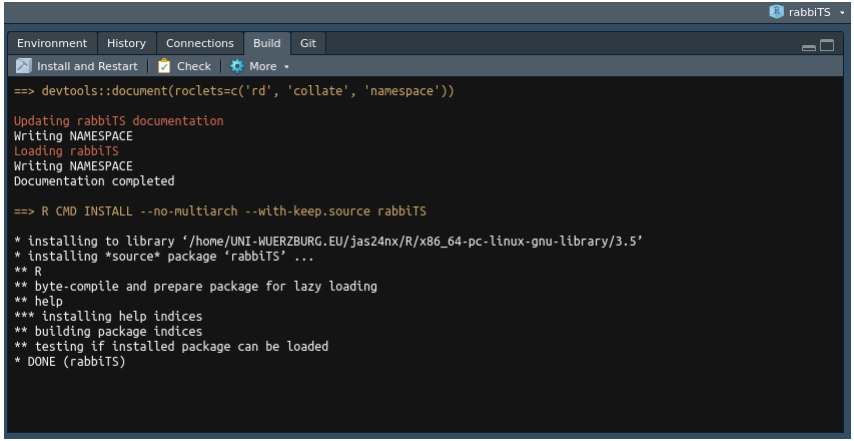
Building the package

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":



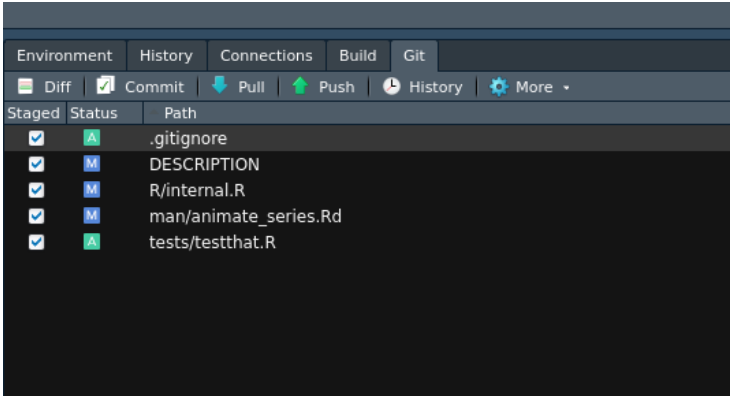
Add, commit and push

As a reminder:

- Add: locally stages the selected files for being commited
- 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 commited are not tracked and will not be pushed to GitHub.
- Push: the local changes are pushed to the connected GitHub repository

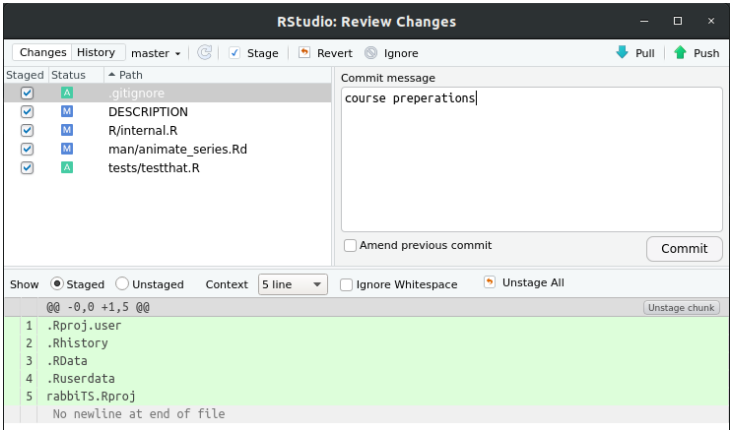
Add, commit and push

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



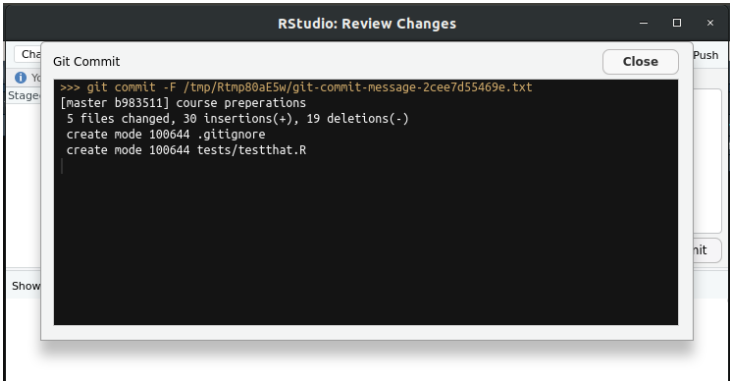
Add, commit and push

Commiting changes in RStudio (your local repository):



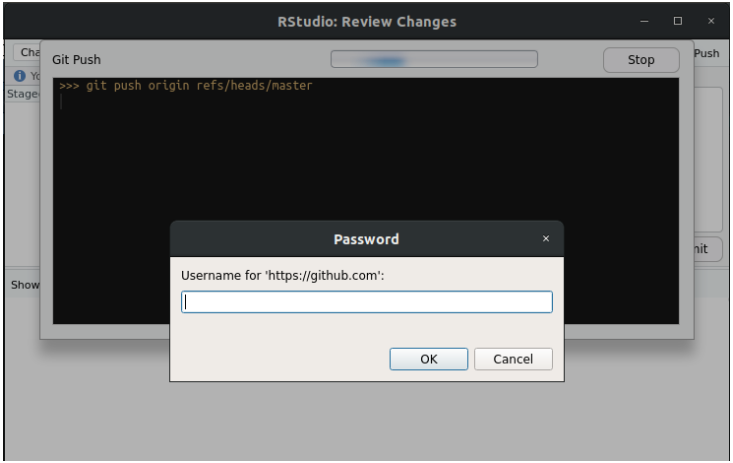
Add, commit and push

Commiting changes in RStudio (your local repository):



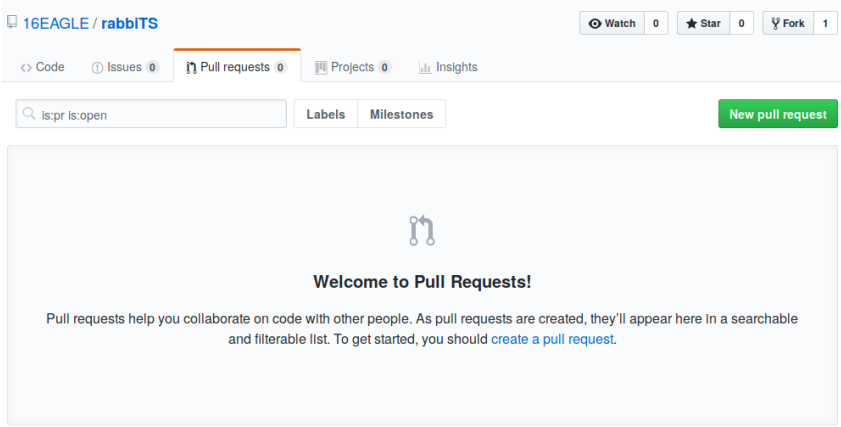
Add, commit and push

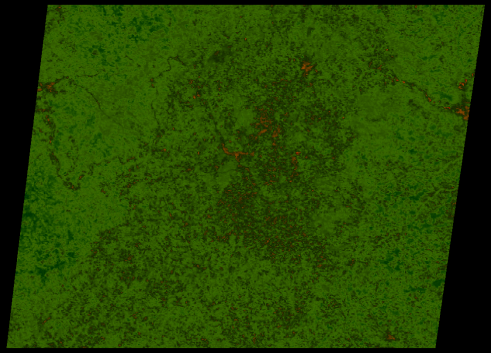
Pushing commits in RStudio (your local repository) to GitHub:



Finally, the pull request

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





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?

```r
#extract temporal boundaries per year
tb = list()
    if(sYear == eYear) {tb[[1]] = c(sDoy,eDoy)}
        if(sYear != eYear) {
            for(y in 1:nYears){
                if(y != nYears) {
                    if((sYear+(y-1) %% 4) > 1) {nDays = 366} else {nDays = 365}   #number of days in the year
                        if(y == 1) {tb[[1]] = c(sDoy,nDays)}
                            if(y > 1) {tb[[length(tb)+1]] == c(0,nDays)}
                              }
                            if(y == nYears) {tb[[length(tb)+1]] = c(0,as.numeric(eDoy))}
                          }
```

Time to code


lunch time