## seabORD website/pkg – cheat sheet

Contents

[seabORD app – cheat sheet 1](#_Toc189213615)

[1. Links seabORD package: 1](#_Toc189213616)

[2. Gitbash 2](#_Toc189213617)

[2.1. STARTING: Getting the repository first time in gitbash 2](#_Toc189213618)

[2.2. PUSHING: If you were up to date (no conflicts) 2](#_Toc189213619)

[2.3. PUSHING: If you are not up to date and you may have conflicts 2](#_Toc189213620)

[2.4.PULLING: If up to date 2](#_Toc189213621)

[2.5. PULLING: If conflicting changes found 3](#_Toc189213622)

[3. The Package 4](#_Toc189213623)

[3.1. folder structure 4](#_Toc189213624)

[3.2. changing the website/pkg 5](#_Toc189213625)

[A.“articles/home page” on website 5](#_Toc189213626)

[B. changing “references” on website 6](#_Toc189213627)

[C. adding/changing DATASETS 6](#_Toc189213628)

### 1. Links seabORD package:

- GitHub: <https://github.com/NERC-CEH/seabORD_pkg>

- website: <https://nerc-ceh.github.io/seabORD_pkg/>

### 2. Gitbash

to paste in github

shift + insert

#### 2.1. STARTING: Getting the repository first time in gitbash

|  |
| --- |
| cd path/to/desired/folder #define folder you want to save the repository in  git clone <https://github.com/NERC-CEH/seabORD_pkg.git> |

#### 2.2. PUSHING: If you were up to date (no conflicts)

|  |
| --- |
| cd path/to/the/repository # cd = change directory  git status # if you want to check which files are different in your local version to that on github  git add . # to add them all for the commit (“stage” them)  git commit -m“commit message here”  git push # push them on the repository |

**Note**: If in repository in the ***.gitignore*** file (may need to set the View of the folder to show hidden files in order to see it) has the line for “docs” it means that when you push to github (or pull) that folder will be ignored. The “docs” folder is what changes the **website**.  
So, to see changes to the vignettes, or other elements of the website it must be pushing the docs folder onto github too. to do so, you either manually remove the “docs” line from the .gitignore file and then push OR, first push all the changes (like above) and then push the docs folder too:

|  |
| --- |
| pwd # pwd = print working directory to check you are still in the right folder (that of the repo) else run cd command…  git add -f docs/ # “f” is force, so it will add that folder even if it is in the git ignore  git commit -m“updating docs”  git push |

#### 2.3. PUSHING: If you are not up to date and you may have conflicts

When trying to push it will tell you that you are not up to date

First pull, merge, and then re-push.

|  |
| --- |
| git pull origin main #get the stuff that is on github  git add .  commit -m"merged update"  git push |

If it tells you that there are conflicts, follow 2.5 to merge

#### 2.4.PULLING: If up to date

|  |
| --- |
| cd path/to/the/repository # cd = change directory  git pull origin main # main = the name of the branch. simply “git pull” may work too as we never messed with branches, but it’s good to be extra specific in my experience to avoid mistakes. |

**Note:** can do it with R open or closed, no difference  
**Note**: If in repository in the ***.gitignore*** file docs is there; then it won’t be pulling those changes.

#### 2.5. PULLING: If conflicting changes found

if when pulling it says there are conflicts:  
“Automatic merge failed; fix conflicts and then commit the result.”

either DISCARD your local changes and stick to pulling:

|  |
| --- |
| #assuming you are in the right directory, or get there  git stash |

**Note:** changes aren’t actually Lost Lost, should be able to get them back with “git stash pop” but why risk it :’)

or MERGE the two versions

|  |
| --- |
| git status #to see which files where “both modified” |

open the conflicting files in R and see: In conflicted file the two versions should be displayed:  
  
<<<<<<< HEAD

This is my local change.  
=======

This is the change from the remote branch.   
>>>>>>> origin/main

check what is in the github (origin/main) vs what is in your local (head). by deleting the orange bit and keeping one or the other options.  
then:

|  |
| --- |
| git add .  git commit -m "Resolved merge conflicts"  git push origin main |

So now your local version has the pulled and merged stuff and so does the github one.

**Note:** If all the conflicts are in docs folder you can stash them as this won’t matter as you will create a new one.

### 3. The Package

#### 3.1. folder structure

Main folders for package

* /R  
  contains all the functions and datasets
* /man

contains the documentation (generated automatically) for each folder

* DESCRIPTION

metadata of package

* NAMESPACE

controls the exporting and importing of datasets and functions (generated automatically)

* test/

scripits that tests the functions. To run them all and perform all the tests run  
devtools::test()

* vignettes/

contains all the articles stuff

* README

the readme file displayed as home for the website AND in github page

* data-raw/

the scripts used to add datasets to the package. This is used to keep track of what is in the actual package, as for each dataset you have the proof of where it comes from and what modifications where made. Each scripts starts with:  
my\_data <- “C:fkdgjfdlgj” # shows where it came from originally  
then all the modifications of the case  
and then it has a line  
usethis::use\_data(my\_data, overwrite = TRUE)  
which generates the dataset in the right format for R packages (.rda) and stores them in theh data/ folder.

so data-raw does not contain any datasets, simply the scripts to create them.

* data/

all the datasets generated in data-raw/. Shouldn’t be manually modified bu talsways to data-raw to ensure that reproducibility is not lost.

* \_pkgdown.yml

controls the website structure and such

* docs/  
  generated by pkgdown with all the website things. not changed manually.

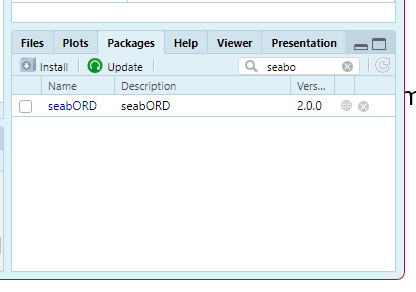
in summary what can be changed manually is:

* scripts in R/
* DESCRIPTION
* README
* test/
* vignettes/
* data-raw/
* \_pkgdown.yml

core commands that must know:

1. **devtools::install()**

installs your package into your R library (like install.packages(“”) does from CRAN), so then seabORD is found here:



WHEN:  
After making changes and you want to test the package in a clean environment.

Before sharing it with others.

1. **devtools::build()**

Creates a “tarball” (.tar.gz), which is the format used for sharing and submitting packages around like on CRAN. But basically, when doing so, it ensures the package is structured correctly.

WHEN:   
After making changes to check everything is alright before installing

1. **devtools::document()**Updates the NAMESPACE file

WHEN:   
After modifying functions or datasets.

Before building or installing to make sure you are keeping up to date NAMESPACE file.

#### 3.2. changing the website/pkg

When you change the pkg (the actual functions or datasets) or website (text, figures etc.), and you want to see the changes reflected on the pkgdown page

To see the changes on website while doing it

|  |
| --- |
| devtools::build() #to check everything is fine. but can skip this step  devtools::document()  pkgdown::build\_site() |

if you don’t have a docs/ folder it will generate one. If you have it already, it will update it and then open the website preview (not LIVE yet)

To make the changes live must update the docs/ version that is on github by pushing it.

so before pushing changes to git hub must always have run “pkgdown::build\_site()” to update the docs file with new changes in the rmd; and then push it to git hub.

##### A.“articles/home page” on website

* to change the **articles** on website change  
  vignettes/ all the .Rmd files.

figures : If adding and deleting etc add them in folder images/ in here as .png and them in the .Rmd file add them browsing to that folder.

* To change **home page** of website:

change the README file.

figures: add them to man/figures. README can only access stuff in man/.

##### B. changing “references” on website

in R/ folder all the functions have ‘# r oxygen documentation. All t he functions that have the line #' @export will be and must be present in the reference page.

* to remove some functions:  
  delete the #' @export line in file in R/ folder  
  delete their name from \_pkgdown.yml
* to add functions:  
  add them in a file in R/ folder with documentation and #' @export line.  
  add them in \_pkgdown.yml
* to change their info on the website  
  On the website the info behind the functions are automatically generated using the r oxygen lines, so to change that text, must change their relative r oxygen in R/.

when adding/removing functions run devtools::document() to update the documentation of pkg

##### C. adding/changing DATASETS

* adding a new dataset:

1. create its file in the data-raw folder by running:  
   usethis::use\_data\_raw("name dataset")
2. in that file create the dataset and run the usethis:: line.  
   (note: with rasters it doesn’t work properly so split info in list of matrix and metadata)
3. check it has added the .rda version of that dataset in the data/ folder.
4. add the dataset documentation in R/datasets.R file. (all the build in datsets MUST be documented) – see how was done for other datasets but like this:

#’documentation

#’documentation

#’documentation

“name\_dataset”

1. add the datasets name in \_pkgdown.yml

* modifying a dataset:

1. go to their data-raw/ file
2. modify it
3. re-run the script with usethis:: line to change the dataset in data/
4. change its documentation in R/datasets.R if structure changes and/or want to add there that it has been updated with x-y changes or whatever

* remove a dataset:

1. delete the file in data-raw/
2. delete the file In data/
3. delete its documentation from R/datasets.R
4. delete it from \_pkgdown.yml