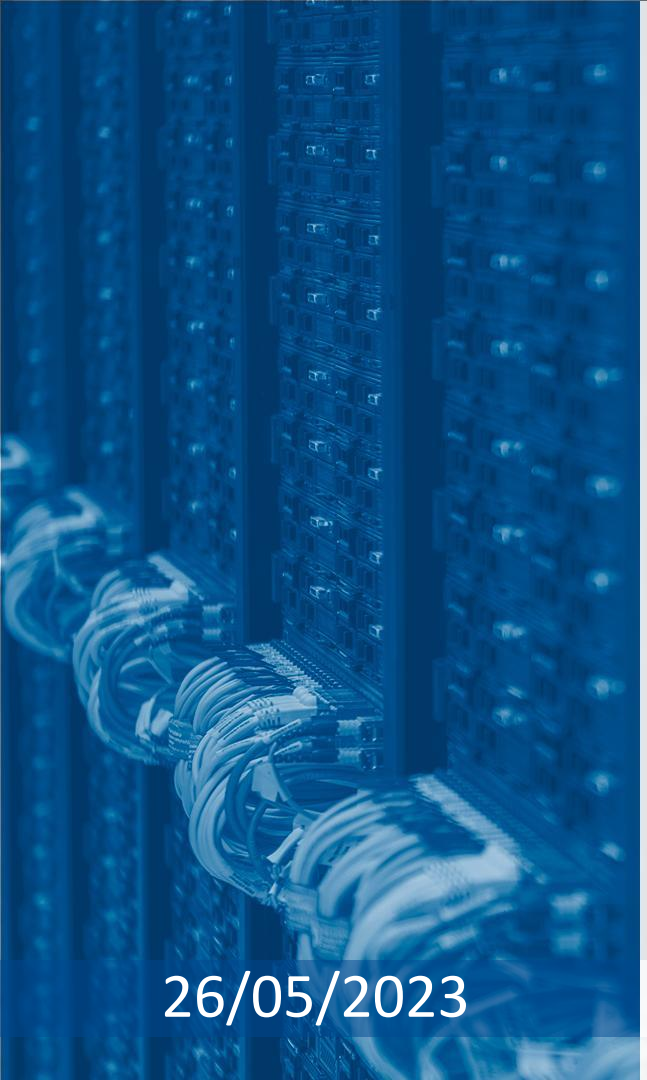


Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Create R package in BSC-ES

26/05/2023

An-Chi Ho

Department R Packages Review

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

BSC-ES R packages

- ★ Functions are split on packages depending on their objective
- ★ Functions from different packages (including external packages) can be used to perform analyses or obtain climate service products

	Package name	Short description	Link to CRAN and GitLab
Data loading and manipulation	**easyNCDF**	Read/write netCDF files into/from multidimensional R array.	https://CRAN.R-project.org/package=easyNCDF https://earth.bsc.es/gitlab/es/easyNCDF
	startR	Data retrieval and processing tools	https://CRAN.R-project.org/package=startR https://earth.bsc.es/gitlab/es/startR
	multiApply	Apply functions to multiple multidimensional arrays or vectors allowing parallel computation	https://CRAN.R-project.org/package=multiApply https://earth.bsc.es/gitlab/ces/multiApply
Analysis and processing	**s2dv**	Functions for Forecast Verification and visualization	https://CRAN.R-project.org/package=s2dv https://earth.bsc.es/gitlab/es/s2dv
	CSTools	Methods for forecast calibration, statistical and stochastic downscaling, optimal forecast combination and tools to obtain tailored products.	https://CRAN.R-project.org/package=CSTools https://earth.bsc.es/gitlab/external/cstools
Climate indicators	**CSIndicators**	Sectorial Indicators for Climate Service	https://CRAN.R-project.org/package=CSIndicators https://earth.bsc.es/gitlab/es/csindicators
	ClimProjDiags	Climate extreme indices, evaluation of the agreement between models, weight and combination functions.	https://CRAN.R-project.org/package=ClimProjDiags https://earth.bsc.es/gitlab/es/ClimProjDiags

BSC-ES R packages

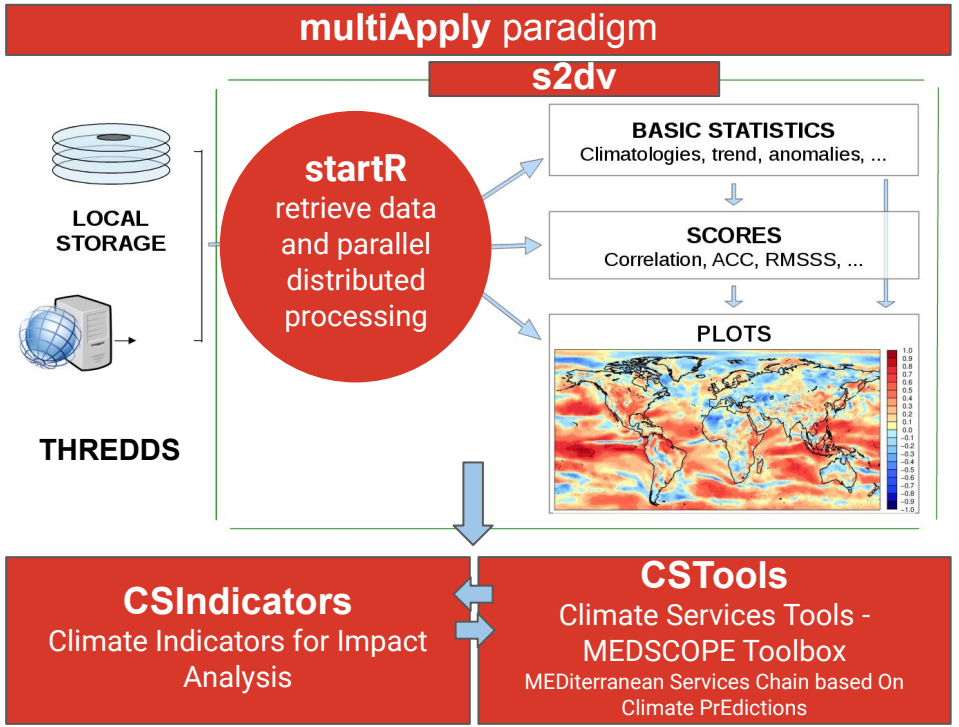
Important features in our tools:

LOADING

- Input data format: netCDF
- Different datasets to be loaded
  - ➢ Data loading flexibility required

ANALYZING

- Accepted R object type by functions: Named multi-dimensional array mainly
- Different forecast horizons and frequency to be analyzed
  - ➢ Function flexibility required
- Multiple-core/node and parallel computation on HPCs
  - ➢ use package “multiApply”



New methods and packages for different needs are being developed continuously.

Function structure: From a developer's view

Bias.R 6.82 KiB

(Bias.R from s2dv)

```
#'Compute the Mean Bias
#'
#'The Mean Bias or Mean Error (Wilks, 2011) is defined as the mean difference
#'between the ensemble mean forecast and the observations. It is a deterministic
#'metric. Positive values indicate that the forecasts are on average too high
#'and negative values indicate that the forecasts are on average too low.
#'It also allows to compute the Absolute Mean Bias or bias without temporal
#'mean. If there is more than one dataset, the result will be computed for each
#'pair of exp and obs data.
#'
#'@param exp A named numerical array of the forecast with at least time
#'  dimension.
                    ...
#'@export
Bias <- function(exp, obs, time_dim = 'sdate', memb_dim = NULL, dat_dim = NULL, na.rm = FALSE,
                 absolute = FALSE, time_mean = TRUE, ncores = NULL) {

  # Check inputs
  ## exp and obs (1)
  if (!is.array(exp) | !is.numeric(exp))
    stop("Parameter 'exp' must be a numeric array.")
                    ...
  ## (Mean) Bias
  bias <- Apply(data = list(exp, obs),
                target_dims = c(time_dim, dat_dim),
                fun = .Bias,
                time_dim = time_dim,
                dat_dim = dat_dim,
                na.rm = na.rm,
                absolute = absolute,
                time_mean = time_mean,
                ncores = ncores)$output1

  return(bias)
}

.Bias <- function(exp, obs, time_dim = 'sdate', dat_dim = NULL, na.rm = FALSE,
                  absolute = FALSE, time_mean = TRUE) {
  # exp and obs: [sdate, (dat)]
```

A typical in-house function would be like:

Header: Documentation following Roxygen2 convention

Function: Work with named multi-dimensional array, with multiple cores option

Sanity check: First step in the function. Check all the input parameters

Computation: Usually short, use multiApply::Apply to allow flexible dimensions and multiple cores.

Atomic function: Work with fixed essential dimensions, store the main analytical code. Non-exported (i.e., no documentation needed)

Function structure: From a developer's view

CST_BiasCorrection.R 15.60 KiB *(CST_BiasCorrection.R from CSTools)*

```
#'Bias Correction based on the mean and standard deviation adjustment
#'
#'@author Verónica Torralba, \email{veronica.torralba@bsc.es}
#'@description This function applies the simple bias adjustment technique
#'described in Torralba et al. (2017). The adjusted forecasts have an equivalent
#'standard deviation and mean to that of the reference dataset.
#'
#'@param exp An object of class \code{s2dv_cube} as returned by \code{CST_Load}
```

```
#'@export
CST_BiasCorrection <- function(exp, obs, exp_cor = NULL, na.rm = FALSE,
                               memb_dim = 'member', sdate_dim = 'sdate',
                               dat_dim = NULL, ncores = NULL) {
  # Check 's2dv_cube'
  if (!inherits(exp, 's2dv_cube') || !inherits(obs, 's2dv_cube')) {
    stop("Parameter 'exp' and 'obs' must be of the class 's2dv_cube'.")
  }
  if (!is.null(exp_cor)) {
    if (!inherits(exp_cor, 's2dv_cube')) {
      stop("Parameter 'exp_cor' must be of the class 's2dv_cube'.")
    }
  }

  BiasCorrected <- BiasCorrection(exp = exp$data, obs = obs$data, exp_cor = exp_cor$data,
                                  memb_dim = memb_dim, sdate_dim = sdate_dim, dat_dim = dat_dim,
                                  na.rm = na.rm, ncores = ncores)
```

A CST prefix function would be like:
Same but with a top level CST_* function

CST Function: Work with s2dv_cube, a wrapper of without CST prefix function

Sanity check: For s2dv_cube object

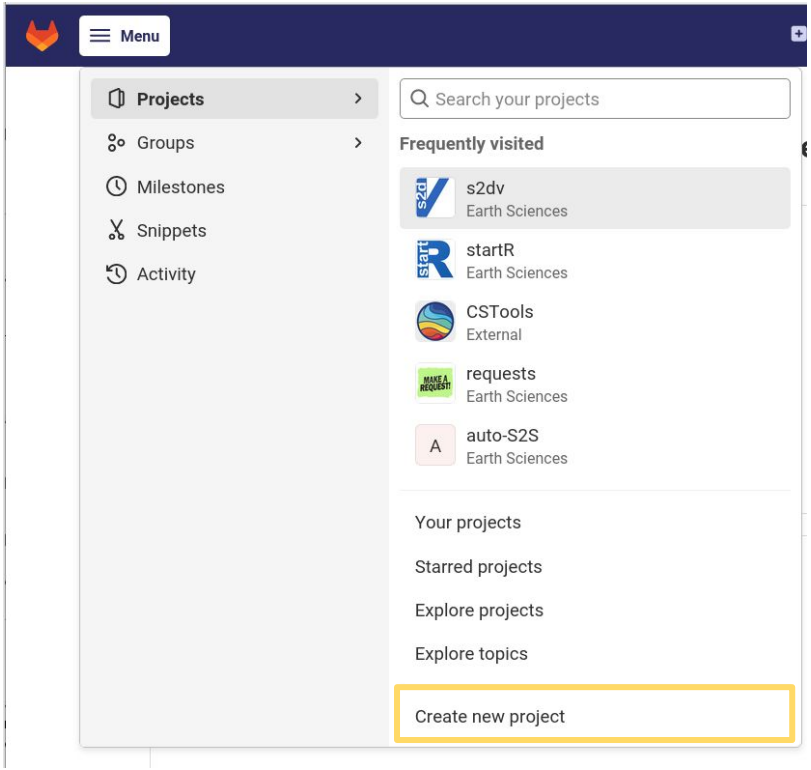
Call function: Call without prefix function, inputs are assigned from the s2dv_cube

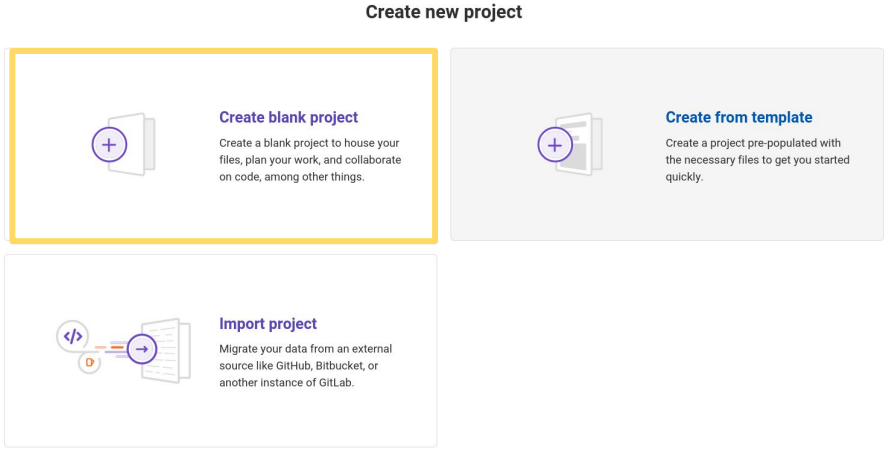
The rest part is the same as previous slide.

Build an R Package in BSC-ES

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

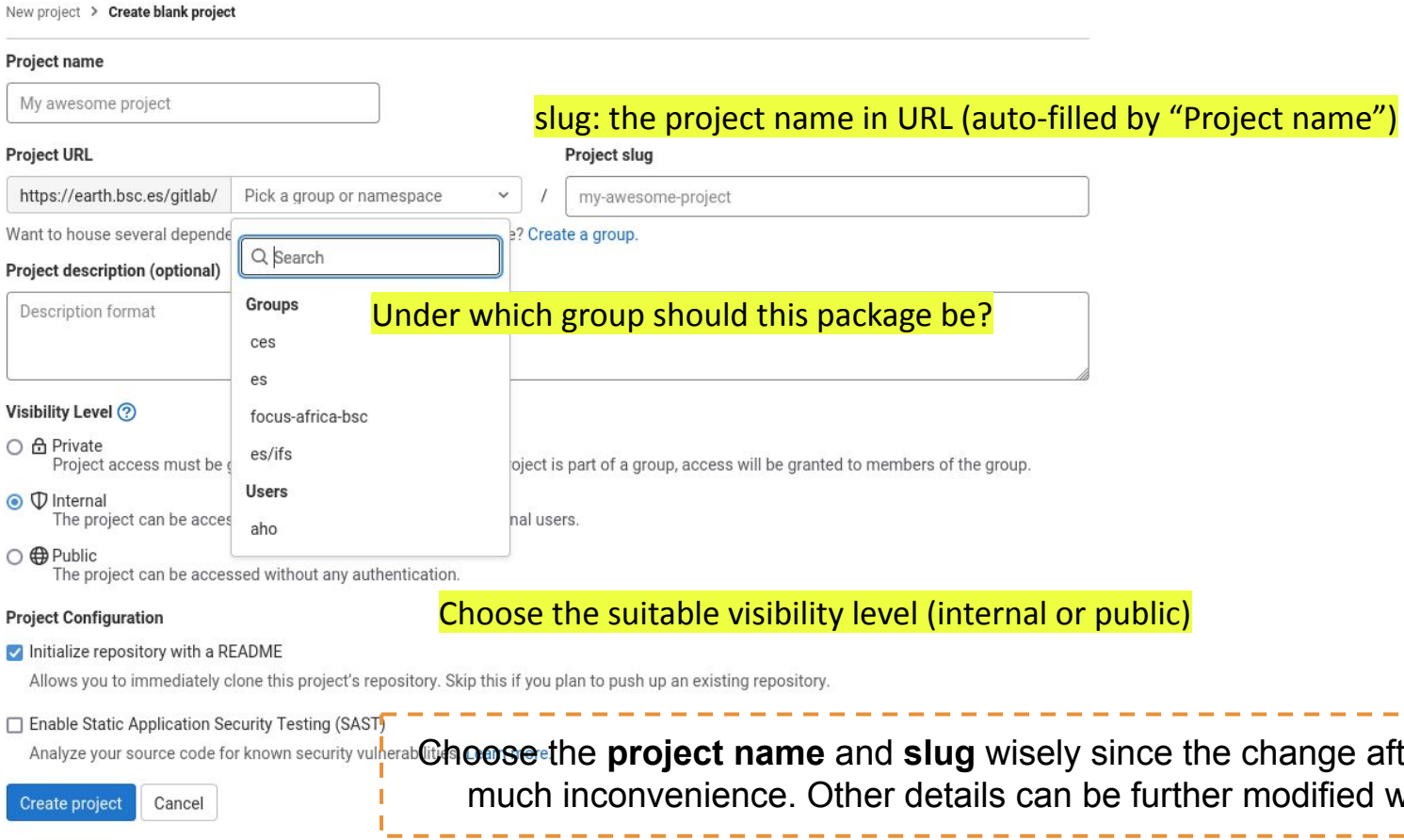
Start on GitLab: Create a new project





Full explanation from GitLab official:
https://docs.gitlab.com/ee/user/project/

Start on GitLab: Create a new project



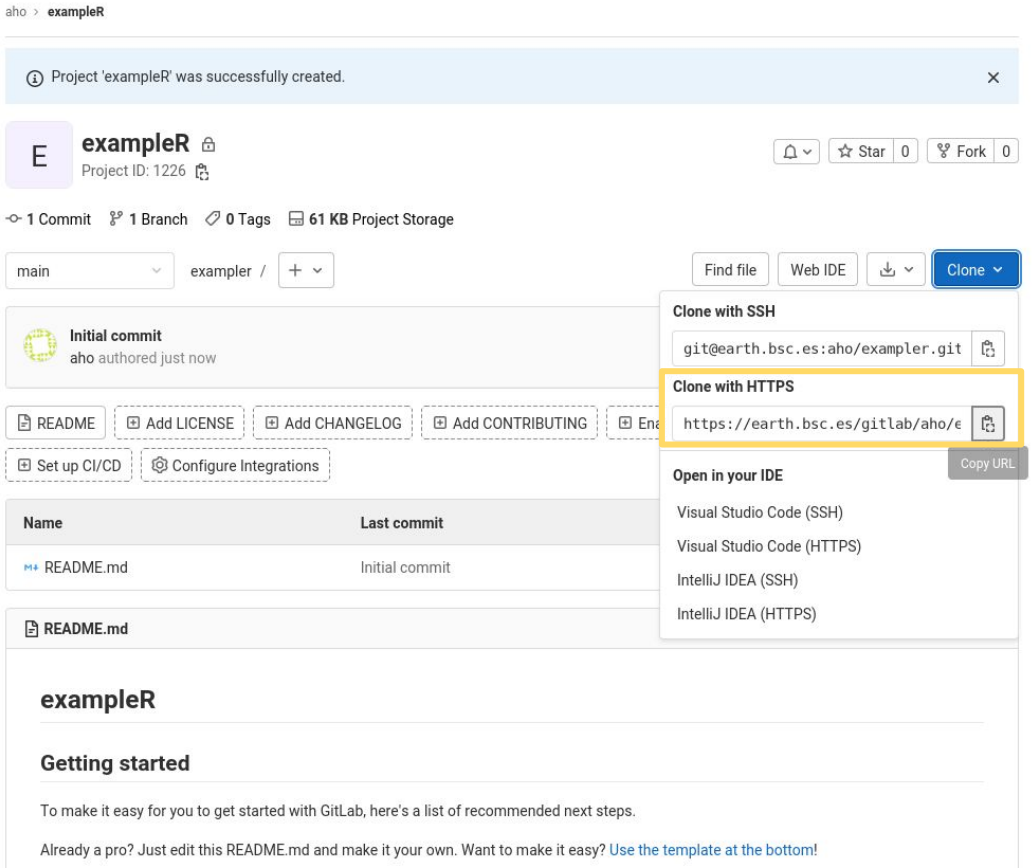
slug: the project name in URL (auto-filled by “Project name”)

Under which group should this package be?

Choose the suitable visibility level (internal or public)

Choose the **project name** and **slug** wisely since the change afterward may cause much inconvenience. Other details can be further modified without problem.

Start on GitLab: Create a new project



Once clicking “Create project” in the previous page, you’ll land on the project main page.

Clone the project to your repository (recommended path: under /esarchive/scratch/<*userID*>/)

```
> git clone <HTTPS URL>
```

Now, you can start building the package both on GitLab and with git.

Build the R package structure

First level of the package:

DESCRIPTION	Fundamental information about the package
NAMESPACE	Imports and Exports. Automatically generated by roxygen2.
R/	Store the .R files, i.e., R functions.
man/	Store the .Rd files. Automatically generated by roxygen2.
NEWS.md	News for each release *(optional)*
tests/	Unit test *(optional)*
vignettes/	To demonstrate the usage of the package *(optional)*

Build the R package structure

First level of the package — helper files:

.gitignore	Folders and files to ignore when git push to GitLab.*(ref)*
.gitlab-ci.yml	Run GitLab CI/CD pipeline.
.Rbuildignore	Folders and files to ignore when R package is built *(ref)*

Too many things to build?
→ Take the files from the existing R packages and modify them for your need.

Recommended package: roxygen2

- An R package to make documenting the code as easy as possible.
- Automatically generating `.Rd` files under folder `man/` and file `NAMESPACE`, and will manage the Collate field in `DESCRIPTION`.

How to run it:

Under the git repo of the package, run `devtools::document()` in R session.

Recommended package: testthat

- An R package to build unit tests for R functions.
- Under folder `test/`, folder `testthat/` to store the unit test file for each function; file `testthat.R` to run the tests.

How to run it:

- Under the git repo of the package, run `devtools::test()` in R session.
- library(testthat) then source the function and unit test file.
- etc.

R CMD build and check

- To build a package from the git project, go one layer above the git folder, and run `R CMD build <folder name>`. A .tar.gz file will be generated.
- To check if the package is accepted by CRAN, run `R CMD check --as-cran <.tar.gz file>`. You should get 0 error and a couple of acceptable warnings due to our system environment.
- The `.tar.gz file` is the one to be submitted to CRAN and/or be installed as a package.

How to start? Some tips…

1. Get familiar with one BSC package, understand the function usage and structure.
   a. How to choose the package?
      i. Need to use s2dv_cube (functions with “CST” prefix): CSTools, CSIndicators
      ii. No need to use s2dv_cube: s2dv, ClimProjDiags
   b. Read function documentation: On CRAN and on GitLab
   c. Read and run the vignettes: On CRAN and/or on GitLab
2. Create a GitLab project

   Choose one existing package similar to your new one, copy the folder structure and some files. Modify based on them.
3. Create R functions

   Based on one function you’ve been familiar with, mimic its structure (header, sanity check, atomic function, etc.)

QUESTIONS?

Barcelona Supercomputing Center
Centro Nacional de Supercomputación