Package 'tinyProject'

July 22, 2025

Type Package

Title A Lightweight Template for Data Analysis Projects

Version 0.6.1

Date 2019-06-13

Author Francois Guillem

Maintainer Francois Guillem <guillem.francois@gmail.com>

Description Creates useful files and folders for data analysis projects and provides functions to manage data, scripts and output files. Also provides a project template for 'Rstudio'.

License GPL (>= 2)

LazyLoad yes

Encoding UTF-8

Imports brew, methods, R.utils

Suggests testthat, remotes, covr

RoxygenNote 6.1.1

NeedsCompilation no

Repository CRAN

Date/Publication 2019-06-14 11:40:03 UTC

Contents

`tinyProject-package` *Organise a R data analysis project*

Description

When working with R on a data analysis project, data files, scripts and output files multiply very quickly.

The purpose of this package is to help the user to keep things organised and improve readibility, maintainability an reusability of its R projects. Moreover it improves the productivity by saving a few seconds very often: it reduces the time spent searching, loading and saving data and scripts.

This package is designed to be used with the Rstudio projects system.

Details

Package:	project
Type:	Package
Version:	0.5
Date:	2016-05-19
License:	GPL (>= 2)

A data analysis project uses one or several data sources and a bunch of scripts to answer questions about some subject and to generate material that will help the diffusion of the results of the project. Ths quality of the answers is not the only critorion of quality of a project: it should also be easy to read and the results should be easy to reproduce.

The purpose of this package is to improve these two aspects by encouraging the user to adopt some simple conventions: every project has the same structure, the scripts always do the same type of operations and are heavily commented, the data is always in the same folder and is also documented, etc.

Project setup: To begin with this package, one has to first create an empty project in Rstudio. Then one can use function `prInit` to setup the project and create a file tree. More precisely `prInit` generates three directories: `data` for storing data files, `scripts` for storing R scripts and `output` for storing output files.

Additionally, three scripts are created and opened in the script editor: `data.R` for the import and processing of data, `main` for the analysis of the data and `start` for all instructions that have to be exetuted everytime the project is opened (load libraries, define constants and functions, etc.).

When the user opens the project in Rstudio, the script `start` is automatically executed. By convention, all scripts starting with the prefix "tools" are also executed. It can be useful to automatically load important specific functions, constants or data.

Script management: The user can open and/or create scripts with the function `prScript`. He just has to give to the function the name of the script (without the extention ".R") and eventually a subdirectory. The fucntion searches in the folder "scripts" the corresponding file. If it does not exists, it is created. In all cases it is opened in the script editor.

The scripts created by `prScript` contains some comments that encourages the user to explain what the script does and eventually what were the results.

although it is not mandatory, the user should use some conventions in the naming of its scripts: a script that process data should start with the prefix "data" while a script that analyses data should start with prefix "analysis". Other scripts generally contain helper functions and their name should start with "tools". They will be sourced when the project is opened.

When possible, data operations and analysis operations should be put in different scripts. If the data operations are long, then the data script should save the final data objects and the analysis script should load the saved data.

Scripts can be sourced with function `prSource`. It is a wrapper to the function `source` but with the same interface as `prScript`.

Data management: Every R object can be saved with `prSave`. The function just needs the name of the object (quoted) and it will save it in the right place so one has not to remember the location where objects are saved. `prSave` can also attach to an object a description that will be displayed when the object is loaded.

To load data, simply use `prLoad`.

Author(s)

François Guillem

Maintainer: François Guillem <guillem.francois@gmail.com>

`prBmp` *Save plots as image files*

Description

These functions can be used in place of their corresponding base R functions to save plots as image files.

Usage

```
prBmp(name, ..., replace = FALSE)

prJpeg(name, ..., replace = FALSE)

prPng(name, ..., replace = FALSE)
```

```
prTiff(name, ..., replace = FALSE)

prPdf(name, ..., replace = FALSE)

prSvg(name, ..., replace = FALSE)

prCairoPdf(name, ..., replace = FALSE)

prCairoPs(name, ..., replace = FALSE)
```

Arguments

name	Name of the output file, without extension. One can also specify subdirectory where to save the file.
...	parameters passed to the corresponding base R function. For instance, for prPng, these parameters will be passed to function png
replace	If the file already exists, should it be overwritten ?

Details

These functions has three advantages over the base functions:

- Files are automatically created in the output folder of the project even if the working directory has changed. Subdirectories are automatically created if they do not exist.
- By default, these functions do not erase an existing file. This avoids accidents.
- The default values of the parameters (width, height, etc.) can be modified with function prOutputDefaults.

Value

These functions are used to open a plot device. Nothing is returned.

Examples

```
projectPath <- file.path(tempdir(), "test")
prInit(projectPath)

prPng("test")
plot(rnorm(100))
dev.off()
# The plot is saved in "output/test.png"
list.files(projectPath, recursive = TRUE, include.dirs = TRUE)

prPng("mysubdirectory/test")
plot(rnorm(100))
dev.off()
# The plot is saved in "output/mysubdirectory/test.png"
list.files(projectPath, recursive = TRUE, include.dirs = TRUE)
```

`prInit` *Initialise a project*

Description

`prInit` should be used in a Rstudio project. The function creates the basic structure of a data analysis project that will help you keep your your work organised and increase your productivity.

Usage

```
prInit(dir = ".", instructions = TRUE)
```

Arguments

`dir`	Directory where project files will be stored. By default, it is the current working directory.
`instructions`	Should instructions added in the scripts created by the the function?

Details

The function creates three folders :

- scripts: Folder where the scripts are stored
- data: Folder where the source and intermediate data files are stored
- output: Folder where the output of the project are stored

These three folders are essential so do not remove them. But you can add any other folders you desire ("latex", "presentation", etc.)

Additionally, three scripts are created:

- main.R: This is the main script of your project. Ideally, just by looking at this script you should remember how your project is organised For small projects most code can go in this script. But for larger project, it should mostly contain comments what each script is supposed to do. You can use the function `prScript` to programatically open the scripts you are referring to.
- data.R: This script should contain inscruction to import data in the project and/or transform the source data in processed data that will be studied/analysed. As for the script "main", for large project, this script should not contain code, but make reference to the scripts that do the job.
- start.R: script that is executed every time the project is opened. It should load the packages used in the project, define constant values, ... More generally it should include all technical instructions that do not directly participate to the creation, manipulation or analysis of data.

See Also

`prLibrary`, `prSource`, `prSave`, `prLoad`, `prScript`

Examples

```
projectPath <- file.path(tempdir(), "test")
prInit(projectPath)

list.files(tempdir(), recursive = TRUE, include.dirs = TRUE)
```

`prLibrary` *Load and install packages*

Description

The function tries to load all packages passed as argument. For those that are not installed, it tries to install them and then load them.

Usage

```
prLibrary(..., warnings = FALSE)
```

Arguments

`...`	name of the packages to load. The names need to be quoted. If a package is missing, the function tries to install it from CRAN by defaults. If a package needs to be installed from github, it can be declared with the following format: `"github:username/pkgname"`. This way, if the package is not installed yet, the function knows how to install it.
`warnings`	Should the function display warnings?

See Also

`prSource`

Examples

```
## Not run:
prLibrary(data.table, plyr)

## End(Not run)
```

prMoveData *Move and delete data files.*

Description

The functions can be used to programmatically move or delete data files.

Usage

```
prMoveData(name, newDir, subdir = ".")

prDeleteData(name, subdir = ".")
```

Arguments

name	Name of the data file one want to move or delete (without extension)
newDir	Subdirectory where to move a data file
subdir	Subdirectory of the data file. It can also be indicated directly in the name parameter.

See Also

prScript

Examples

```
projectPath <- file.path(tempdir(), "test")
prInit(projectPath)

x <- rnorm(100)

prSave(x)
list.files(projectPath, recursive = TRUE, include.dirs = TRUE)

prMoveData(x, "testdir")
list.files(projectPath, recursive = TRUE, include.dirs = TRUE)

prDeleteData("testdir/x")
list.files(projectPath, recursive = TRUE, include.dirs = TRUE)
```

prMoveScript *Move, rename and delete scripts*

Description

These functions can be used to pragrammatically move, rename and delete scripts files.

Usage

```
prMoveScript(name, newDir, subdir = ".")

prRenameScript(name, newName, subdir = ".")

prDeleteScript(name, subdir = ".")
```

Arguments

name	Name of the script on which one wants to perform an action.
newDir	Subdirectory where to move a script.
subdir	Subdirectory of the script on which one wants to perform an action. It can also be indicated directly in the name parameter.
newName	New name of the script.

See Also

prScript

Examples

```
projectPath <- file.path(tempdir(), "test")
prInit(projectPath)

prScript("test")

prMoveScript("test", "testdir")

prRenameScript("testdir/test", "myTest")

prDeleteScript("testdir/myTest")
```

prOutputDefaults *Set default values of output functions*

Description

This function can modify the default values of the parameters of output function like prWriteTable or prPdf. This can be useful for instance to set in one and only one location the default size of output images, the default font...

Usage

```
prOutputDefaults(table = NA, image = NA, pdf = NA, cairo = NA)
```

Arguments

table	Named list. The names correspond to argument names of write.table and the values to the new default values of these parameters. If this argument is NULL the defaults are reset to their original values
image	Named list. The names correspond to argument names of png and the values to the new default values of these parameters. If this argument is NULL the defaults are reset to their original values
pdf	Named list. The names correspond to argument names of pdf and the values to the new default values of these parameters. If this argument is NULL the defaults are reset to their original values
cairo	Named list. The names correspond to argument names of cairo_pdf and the values to the new default values of these parameters. If this argument is NULL the defaults are reset to their original values

Value

prOutputDefaults invisibly returns the list of modified defaults values.

Examples

```
# Remove row names of table output:
prOutputDefaults(table = list(row.names = FALSE))

# Modify the default size of pdf output:
prOutputDefaults(pdf = list(width = 8, height = 6))

# Reset default values for pdf and table output
prOutputDefaults(table = NULL, pdf = NULL)
```

prPath *Get the full path of a file*

Description

The function returns the absolute path of a file contained in the current project.

Usage

```
prPath(x)
```

Arguments

x	Relative path of a file contained in the project.

Value

Absolute path of the file.

Examples

```
projectPath <- file.path(tempdir(), "test")
prInit(projectPath)

prPath("data")
```

prSave *Easily save and load data*

Description

prSave and prLoad save and load data in the folder data of the project. Each object is saved in distinct file with the same name as the object and the extension rda.

Usage

```
prSave(name, replace = FALSE, desc = "No description", subdir = ".")

prLoad(name, subdir = ".", trace = TRUE, envir = .GlobalEnv)
```

Arguments

name	Name of the object to save or to load. Quotes are optional. Optionaly, one can also specify the subfolder where to save or load the object.
replace	If the file already exists, should it be overwrited ? The prupose of this parameter is to avoid some dramas. Use it with caution.
desc	Short description of the object.
subdir	subdirectory where to save or load the object.
trace	Should information about the loaded object be printed ?
envir	the environment where the object should be loaded. By default, it is the global environement so that the object should be easily accessible in all settings

See Also

prLibrary, prSource

Examples

```
projectPath <- file.path(tempdir(), "test")
prInit(projectPath)

test <- rnorm(100)
prSave(test)

list.files(projectPath, recursive = TRUE, include.dirs = TRUE)

# Save again but add a description
prSave(test, replace = TRUE, desc = "Just a test !")

prLoad(test)

# It is also possible to save/load in subfolders
prSave(test, TRUE, subdir = "testdir", desc = "Saved in subfolder")
list.files(projectPath, recursive = TRUE, include.dirs = TRUE)

# Or equivalently
prSave("testdir/test", TRUE, desc = "Saved in subfolder")

prLoad(test, subdir="testdir")
prLoad("testdir/test")
```

---

prScript *create and/or open R scripts*

Description

prScript creates and opens a new script for edition. If the script already exists, it opens it in the Rstudio editor.

Usage

```
prScript(name, template, subdir = ".", instructions = TRUE)
```

Arguments

name	(optional) name of the script to create and/or open. This parameter can also include the subfolder where to save the script (see examples).
template	(Optional) One of "analysis", "data" or "function". Template to use for the creation of a script. prScript adds a few comments that encourage the user to add comments and explain what the script will do. These comments depend on the choosen template. If the argument is missing, the choosen template depends on the name of the script: "data" if the name begins with "data", "function" if it start with "tools" and "analysis" in all other cases.
subdir	subdirectory where the scripts needs to be created or opened. The subdirectory can also be directly specified in the parameter name.
instructions	Should the created script include some instructions?

Details

If the parameter name is missing, then a list of existing scripts is displayed. The user is then invited to choose one by typing the number or name of the script he wants to open.

See Also

prSource, prMoveScript, prRenameScript, prDeleteScript

Examples

```
projectPath <- file.path(tempdir(), "test")
prInit(projectPath)

prScript("test")
list.files(projectPath, recursive = TRUE, include.dirs = TRUE)

prScript("myFunction", template = "function")

# Create script in a subfolder
prScript("test", subdir = "testdir")

# Or equivalently
prScript("testdir/test")
```

prSource *Source a R script*

Description

Source a R script in the "scripts" folder of the project.

Usage

```
prSource(name, subdir = ".")
```

Arguments

name	Name of the script to execute. It can also contain the subfolder where the script is stored.
subdir	subdirectory where the script is located. The subdirectory can also be directly specified in the parameter name.

See Also

prLibrary, prLoad, prSave

Examples

```
projectPath <- file.path(tempdir(), "test")
prInit(projectPath)

prScript("helloWorld")

# Edit the script so that it does something cool

prSource("helloWorld")

# Source a file in a subdirectory
prScript("myScript", subdir = "testdir")
prSource("myScript", subdir = "testdir")

# Or equivalently
prSource("testdir/myScript")
```

prStart *Source start and tools scripts*

Description

This function sources all scripts prefixed by `tools` or situated in a subdirectory called `tools`, then it sources the `start.R` script. It is automatically called when a project is opened.

Usage

```
prStart(dir = ".", trace = TRUE)
```

Arguments

`dir`	Path of the directory where the project is stored.
`trace`	Should the function print what it is doing ?

Value

TRUE if all scripts had been sourced without error and false otherwise.

Examples

```
projectPath <- file.path(tempdir(), "test")
prInit(projectPath)

prStart(projectPath)
```

prWriteTable *Write data in an output text file*

Description

These functions are wrappers to `write.table`, `write.csv` and `write.csv2`. They write a matrix or a data.table in a ".txt" or ".csv" file in the output folder. The file created has the same name as the object.

Usage

```
prWriteTable(name, ..., replace = FALSE)

prWriteCsv(name, ..., replace = FALSE)

prWriteCsv2(name, ..., replace = FALSE)
```

Arguments

name	Name of the object to write. Quotes are optional. This argument can also specify the subdirectory of folder "output" where to write the file.
...	arguments to write.table
replace	If the file already exists, should it be overwritten ?

Examples

```
projectPath <- file.path(tempdir(), "test")
prInit(projectPath)

mydata <- data.frame(x = 1:10, y = rnorm(10))

prWriteTable(mydata)
list.files(projectPath, recursive = TRUE, include.dirs = TRUE)

# Write in a subdirectory of "output"
prWriteTable("mydir/mydata")
list.files(projectPath, recursive = TRUE, include.dirs = TRUE)
```

---

requireVariable *Ask user the value of a variable*

Description

In interactive mode, this functions asks the user to type the value of a given variable. In non-interactive mode, it searchs the value in the command line arguments and it returns an error if the value does not exist.

Usage

```
requireVariable(name, desc = NULL, default = NULL,
  what = character(), nmax = 1, alwaysAsk = TRUE, env = .GlobalEnv)
```

Arguments

name	Name of the required variable
desc	Description to display
default	Default value for the variable
what	Type of data to read in interactive mode.
nmax	Number of values to read in interactive mode.
alwaysAsk	If the variable already exists, should the function ask a new value?
env	Environment where the variable should be defined

Value

Used for side effects

Index