Package 'link2GI'

October 27, 2018

Type Package

Title Linking Geographic Information Systems, Remote Sensing and Other Command Line Tools

Version 0.3-5

Date 2018-10-26

Encoding UTF-8

Maintainer Chris Reudenbach <reudenbach@uni-marburg.de>

Description Functions to simplify the linking of open source GIS and remote sensing related command line interfaces.

License GPL (>= 3) | file LICENSE

Depends R (>= 3.1), methods

Imports raster, rgdal, gdalUtils, tools, rgrass7, sp, sf, RSAGA, devtools, roxygen2

SystemRequirements GNU make

RoxygenNote 6.1.0

Suggests knitr, rmarkdown,

VignetteBuilder knitr

NeedsCompilation no

Author Tim Appelhans [ctb], Chris Reudenbach [cre, aut]

Repository CRAN

Date/Publication 2018-10-26 23:00:03 UTC

R topics documented:

`findGRASS` *Search recursivly existing 'GRASS GIS' installation(s) at a given drive/mountpoint*

Description

Provides an list of valid 'GRASS GIS' installation(s) on your 'Windows' system. There is a major difference between osgeo4W and stand_alone installations. The functions trys to find all valid installations by analysing the calling batch scripts.

Usage

```
findGRASS(searchLocation = "default", ver_select = FALSE,
  quiet = TRUE)
```

Arguments

`searchLocation`	drive letter to be searched, for Windows systems default For Windows Systems it is mandatory to use Capitel letters with colon only is `C:`, for Linux systems default is `/usr`.
`ver_select`	boolean default is FALSE. If there is more than one 'SAGA GIS' installation and `ver_select` = TRUE the user can select interactively the preferred 'SAGA GIS' version
`quiet`	boolean switch for supressing console messages default is TRUE

Value

A dataframe with the 'GRASS GIS' root folder(s), version name(s) and installation type code(s)

Author(s)

Chris Reudenbach

Examples

```
## Not run:
# find recursively all existing 'GRASS GIS' installation folders starting
# at the default search location
findGRASS()

## End(Not run)
```

`findOTB` *Search recursivly existing 'Orfeo Toolbox' installation(s) at a given drive/mountpoint*

Description

Provides an list of valid 'OTB' installation(s) on your 'Windows' system. There is a major difference between osgeo4W and stand_alone installations. The functions trys to find all valid installations by analysing the calling batch scripts.

Usage

```
findOTB(searchLocation = "default", quiet = TRUE)
```

Arguments

`searchLocation`	drive letter to be searched, for Windows systems default is `C:`, for Linux systems default is `/usr`.
`quiet`	boolean switch for supressing console messages default is TRUE

Value

A dataframe with the 'OTB' root folder(s), and command line executable(s)

Author(s)

Chris Reudenbach

Examples

```
## Not run:
# find recursively all existing 'Orfeo Toolbox' installations folders starting
# at the default search location
findOTB()

## End(Not run)
```

`findSAGA` *Search recursivly existing 'SAGA GIS' installation(s) at a given drive/mountpoint*

Description

Provides an list of valid 'SAGA GIS' installation(s) on your 'Windows' system. There is a major difference between osgeo4W and stand_alone installations. The functions trys to find all valid installations by analysing the calling batch scripts.

Usage

```
findSAGA(searchLocation = "default", quiet = TRUE)
```

Arguments

`searchLocation`	drive letter to be searched, for Windows systems default is `C:`, for Linux systems default is `/usr`.
`quiet`	boolean switch for supressing console messages default is TRUE

Value

A dataframe with the 'SAGA GIS' root folder(s), version name(s) and installation type code(s)

Author(s)

Chris Reudenbach

Examples

```
## Not run:
# find recursively all existing 'SAGA GIS' installation folders starting
# at the default search location
findSAGA()

## End(Not run)
```

`gvec2sf` *Converts from an existing GRASS 7 environment an arbitrary vector dataset into a sf object*

Description

Converts from an existing GRASS 7 environment an arbitrary vector dataset into a sf object

Usage

```
gvec2sf(x, obj_name, gisdbase, location, gisdbase_exist = TRUE)
```

Arguments

x	sf object corresponding to the settings of the corresponding GRASS container
obj_name	name of GRASS layer
gisdbase	GRASS gisDbase folder
location	GRASS location name containing obj_name)
gisdbase_exist	logical switch if the GRASS gisdbase folder exist default is TRUE

Note

have a look at the sf capabilities to read direct from sqlite

Author(s)

Chris Reudenbach

Examples

```
## Not run:
## example
# get meuse data as sf object
require(sf)
meuse_sf = st_as_sf(meuse,
                    coords = c("x", "y"),
                    crs = 28992,
                    agr = "constant")


# write data to GRASS and create gisdbase
sf2gvec(x = meuse_sf,
          obj_name = "meuse_R-G",
          gisdbase = "~/temp3",
          location = "project1")

# read from existing GRASS
gvec2sf(x = meuse_sf,
          obj_name = "meuse_R-G",
          gisdbase = "~/temp3",
          location = "project1")

## End(Not run)
```

initProj *Defines and creates folders and variables*

Description

Defines and creates (if necessary) all folders variables. Returns a list with the project folder pathes. Optionally exports all pathes to a global sub environment.

Usage

```
initProj(projRootDir = tempdir(), GRASSlocation = "tmp/",
  projFolders = c("data/", "result/", "run/", "log/"),
  path_prefix = "", global = FALSE)
```

Arguments

projRootDir	project github root directory (your github name)
GRASSlocation	folder for GRASS data
projFolders	list of subfolders in project
path_prefix	character a prefix for the path variables names default is ""
global	boolean esport path strings as global variables default is false

Examples

```
## Not run:

link2GI::initProj(projRootDir = tempdir(),
                  projFolders = c("data/",
                                  "data/level0/",
                                  "data/level1/",
                                   "output/",
                                   "run/",
                                   "fun/") )

## End(Not run)
```

link2GI *Bridges to the GI-World*

Description

A straightforward helper tool for linking GI/RS functionality to R. The goals of the package are to correctly initialize both the existing wrapper packages rgrass7 and RSAGA and to smoothly enable the necessary system variables and path parameters for a direct access of the binaries via system calls on all operating systems. In particular, rgrass7 and RSAGA can cause severe problems during initialization of parallel installations of GRASS GIS or SAGA GIS under the Windows operating system(s). link2GI tries to set the correct system settings and returns if system calls are required the necessary paths and commands.
Furthermore the package provides a linkage to the Orfeo Toolbox software as well as a basic list based command parser.

In addition there are some usefull functions for creating project folder structures, direct reading and writing of GRASS vector data and manual building of the package in UNC related enviroments.

Details

Functions for linking GI/RS functionality to R

Note

To utilize the power of the open source GI tools from within R you need to installthem first. The link2GI package just tries to generate correct environment settings as system and path variables for the most of the known issues. As a first promising opportunity to do fullfil most of the requirements you may install QGIS, GRASS GIS 7.x and SAGA-GIS following the excellent installation instructions of the RQGIS package will have a good first try to ensure a smooth working environment.

For a broad number of problems you may just use the RQGIS package as wrapper for the functionalitythat can be reached via QGIS. link2GI is tested under Windows 7/10 as well as on the Ubuntu/Debian Linux distributions. The OSX operation system should run but is not tested.

Author(s)

Chris Reudenbach Tim Appelhans
Maintainer: Chris Reudenbach <reudenbach@uni-marburg.de>

---

linkGDAL *Checks and sets up via* gdalUtils *the 'GDAL' pathes and settings for the command line usage.*

Description

For almost all GI related software tools you need to have installed the 'GDAL' binaries. It is useful for checking and linking the 'GDAL' binary installation to the 'R'environment for command line calls of GDAL functions. linkGDAL checks via gdalUtils the status of the 'GDAL' binaries installation.

Usage

```
linkGDAL(quiet = TRUE, returnPaths = TRUE)
```

Arguments

quiet	boolean switch for supressing console messages default is TRUE
returnPaths	boolean if set to FALSE the GDAL binary path is written to the PATH variable only, otherwise all paths and names of the installed "GDAL" ae returned.

Author(s)

Chris Reudenbach

Examples

```
## Not run:

# get all available GDAL informations
gdal<-linkGDAL()
if (nchar(gdal[[1]][[1]])>0) {
# get available GDAL driver
gdal[[1]]$drivers$format_code

# get the binary path (is also written to the PATH variable)
gdal[[1]]$path

# get the available python tools e.g. for gdal_sieve.py
# you may call it like following:
#  ret <- system(paste0("gdal_sieve.py -8 ",
#                       "input.sdat ",
#                       "output.sdat ",
#                       "-of SAGA"), intern = TRUE)
gdal[[1]]$python_utilities

}

## End(Not run)
```

---

linkGRASS7 *Locate and set up 'GRASS 7' API bindings*

Description

Initializes the session environment and the system pathes for an easy acces to 'GRASS GIS 7.x'. The correct setup of the spatial and projection parameters is automatically performed by using either an existing and valid raster, sp or sf object, or manually by providing a list containing the minimum parameters needed.

Usage

```
linkGRASS7(x = NULL, default_GRASS7 = NULL, search_path = NULL,
  ver_select = FALSE, gisdbase_exist = FALSE, gisdbase = NULL,
  location = NULL, spatial_params = NULL, resolution = NULL,
  quiet = TRUE, returnPaths = FALSE)
```

Arguments

`x`	raster or sp object
`default_GRASS7`	default is `NULL` If is `NULL` an automatic search for all installed versions is performed. If you provide a valid list the corresponding version is initialized. An example for OSGeo4W64 is: `c("C:/OSGeo4W64","grass-7.0.5","osgeo4w")`
`search_path`	path or mounting point that will be searched
`ver_select`	boolean if TRUE you may choose interactively the binary version (if found more than one), by default FALSE
`gisdbase_exist`	default is FALSE if set to TRUE the arguments gisdbase and location are expected to be an existing GRASS gisdbase
`gisdbase`	default is `NULL`, invoke `tempdir()` to the 'GRASS' database. Alternativeley you can provide a individual path.
`location`	default is `NULL`, invoke `basename(tempfile())` for defining the 'GRASS' location. Alternativeley you can provide a individual path.
`spatial_params`	default is `NULL`. Instead of a spatial object you may provide the geometry as a list. E.g. c(xmin,ymin,xmax,ymax,proj4_string)
`resolution`	resolution in map units for the GRASS raster cells
`quiet`	boolean switch for supressing console messages default is TRUE
`returnPaths`	boolean if set to FALSE the pathes of the selected version are written to the PATH variable only, otherwise all paths and versions of the installed GRASS versions ae returned.

Details

The concept is straightforward but for an all days usage helpful. Either you need to provide a `raster` or `sp` `sf` spatial object which has correct spatial and projection properties or you may link directlxy to an existing 'GRASS' gisdbase and mapset. If you choose an spatial object to initialize a correct 'GRASS' mapset it is used to create either a temporary or a permanent rgrass7 environment including the correct 'GRASS 7' structure.

The most time consuming part on 'Windows' Systems is the search process. This can easily take 10 or more minutes. To speed up this process you can also provide a correct parameter set. Best way to do so is to call `searchGRASSW` or for 'Linux' `searchGRASSX` manually. and call `linkGRASS7` with the version arguments of your choice. linkGRASS7 initializes the usage of GRASS7.

Note

'GRASS GIS 7' is excellently supported by the rgrass7 wrapper package. Nevertheless 'GRASS GIS' is well known for its high demands regarding the correct spatial and reference setup an a bunch of workspace and environment requirements. This becomes even worse on 'Windows' platforms or if several alternative 'GRASS GIS' installations are available. If one knows what to do the `rgrass7` package setup function `initGRASS` works fine under Linux. This is also valid for well known configurations under the 'Windows' operation system. Nevertheless on university lab or on company computers with restriced privileges and/or using different releases like the 'OSGeo4W' distribution and the 'GRASS 7' stand-alone installation, or different software releases (e.g. 'GRASS 7.0.5 and GRASS 7.2.0), it becomes often cumbersome or even impossible to get the correct linkages.
The function `linkGRASS7` tries to find all valid 'GRASS GIS' binaries by analyzing the startup script files of 'GRASS GIS'. After identifying the 'GRASS GIS' binaries all necessary system variables and settings will be generated and passed to a temporary R enviroment.

If you have more than one valid installation and run `linkGRASS7()` without arguments, you will be ask to select one.

Author(s)

Chris Reudenbach

Examples

```
## Not run:
# get meuse data as sp object
library(link2GI)
require(sp)

# proj folders
projRootDir<-tempdir()
paths<-link2GI::initProj(projRootDir = projRootDir,
                         projFolders = c("project1/"))

data(meuse)
coordinates(meuse) <- ~x+y
proj4string(meuse) <-CRS("+init=epsg:28992")

# get meuse data as sf object
require(sf)
meuse_sf = sf::st_as_sf(meuse,
                    coords =
                    c("x", "y"),
                    crs = 28992,
                    agr = "constant")


# Automatic search and find of GRASS binaries
# using the meuse sp data object for spatial referencing
# This is the highly recommended linking procedure for on the fly jobs
# NOTE: if more than one GRASS installation is found you have to choose.
```

```
grass<-linkGRASS7(meuse,returnPaths = TRUE)
if (grass$exist){

# CREATE and link to a permanent GRASS folder at "projRootDir", location named "project1"
linkGRASS7(meuse_sf, gisdbase = projRootDir, location = "project1")

# ONLY LINK to a permanent GRASS folder at "projRootDir", location named "project1"
linkGRASS7(gisdbase = projRootDir, location = "project1", gisdbase_exist = TRUE )


# setting up GRASS manually with spatial parameters of the meuse data
proj4_string <- as.character(sp::CRS("+init=epsg:28992"))
linkGRASS7(spatial_params = c(178605,329714,181390,333611,proj4_string))

# creating a GRASS gisdbase manually with spatial parameters of the meuse data
# additionally using a peramanent directory "projRootDir" and the location "meuse_spatial_params "
proj4_string <- as.character(sp::CRS("+init=epsg:28992"))
linkGRASS7(gisdbase = projRootDir,
           location = "meuse_spatial_params",
           spatial_params = c(178605,329714,181390,333611,proj4_string))
}

## Some more examples related to interactive selection or OS specific settings

# SELECT the GRASS installation and define the search location
linkGRASS7(meuse_sf, ver_select = TRUE, search_path = "~")

# SELECT the GRASS installation
linkGRASS7(meuse_sf, ver_select = TRUE)

# Typical osge4W installation (QGIS), using the meuse sp data object for spatial referencing
linkGRASS7(meuse,c("C:/Program Files/QGIS 2.18","grass-7.2.1","osgeo4W"))

# Typical osgeo4W installation (rootdir), using the meuse sp data object for spatial referencing
linkGRASS7(meuse,c("C:/OSGeo4W64/","grass-7.2.2","osgeo4W"))


## End(Not run)
```

---

linkOTB *Locate and set up 'Orfeo ToolBox' API bindings*

Description

Locate and set up 'Orfeo ToolBox' API bindings

Usage

```
linkOTB(bin_OTB = NULL, root_OTB = NULL, type_OTB = NULL,
  searchLocation = NULL, ver_select = FALSE, quiet = TRUE,
  returnPaths = TRUE)
```

Arguments

bin_OTB	string contains path to where the otb binaries are located
root_OTB	string provides the root folder of the bin_OTB
type_OTB	string
searchLocation	string hard drive letter default is C:
ver_select	boolean default is FALSE. If there is more than one 'OTB' installation and ver_select = TRUE the user can select interactively the preferred 'OTB' version
quiet	boolean switch for supressing messages default is TRUE
returnPaths	boolean if set to FALSE the pathes of the selected version are written to the PATH variable only, otherwise all paths and versions of the installed GRASS versions ae returned.

Details

It looks for the otb_cli.bat file. If the file is found in a bin folder it is assumed to be a valid 'OTB' binary installation.

if called without any parameter linkOTB() it performs a full search over the hardrive C:. If it finds one or more 'OTB' binaries it will take the first hit. You have to set ver_select = TRUE for an interactive selection of the preferred version.

Value

add otb pathes to the enviroment and creates global variables path_OTB

Note

You may also set the path manually. Using a 'OSGeo4W64' http://trac.osgeo.org/osgeo4w/ installation it is typically C:/OSGeo4W64/bin/

Author(s)

Chris Reudenbach

Examples

```
## Not run:
# call if you do not have any idea if and where OTB is installed
otb<-linkOTB()
if (otb$exist) {
# call it for a default OSGeo4W installation of the OTB
print(otb)
}

## End(Not run)
```

`linkSAGA` *Identifies SAGA GIS Installations and returns linking Informations*

Description

Finds the existing SAGA GIS installation(s), generates and sets the necessary path and system variables for a seamless use of the command line calls of the 'SAGA GIS' CLI API, setup valid system variables for calling a default `rsaga.env` and by this makes available the `RSAGA` wrapper functions.
All existing installation(s) means that it looks for the `saga_cmd` or `saga_cmd.exe` executables. If the file is found it is assumed to be a valid 'SAGA GIS' installation. If it is called without any argument the most recent (i.e. highest) SAGA GIS version will be linked.

Usage

```
linkSAGA(default_SAGA = NULL, searchLocation = "default",
  ver_select = FALSE, quiet = TRUE, returnPaths = TRUE)
```

Arguments

`default_SAGA`	string contains path to `RSAGA` binaries
`searchLocation`	drive letter to be searched, for Windows systems default is `C:`, for Linux systems default is `/usr`.
`ver_select`	boolean default is FALSE. If there is more than one 'SAGA GIS' installation and `ver_select` = TRUE the user can select interactively the preferred 'SAGA GIS' version
`quiet`	boolean switch for supressing console messages default is TRUE
`returnPaths`	boolean if set to FALSE the pathes of the selected version are written to the PATH variable only, otherwise all paths and versions of the installed SAGA versions ae returned.#'@details If called without any parameter `linkSAGA()` it performs a full search over `C:`. If it finds one or more 'SAGA GIS' binaries it will take the first hit. You have to set `ver_select = TRUE` for an interactive selection of the preferred version. Additionally the selected SAGA pathes are added to the environment and the global variables `sagaPath`, `sagaModPath` and `sagaCmd` will be created.

Value

A list containing the selected `RSAGA` path variables `$sagaPath`,`$sagaModPath`,`$sagaCmd` and potentially other installations `$installed`

Note

The excellent 'SAGA GIS' wrapper RSAGA is in line for a major update however it covers currently (Feb 2018) only 'SAGA GIS' versions from 2.0.4 - 2.2.3. The fast evolution of 'SAGA GIS' makes it highly impracticable to keep the wrapper adaptions in line. `RSAGA` will meet all linking

needs perfectly if you use 'SAGA GIS' versions from 2.0.4 - 2.2.3.
RSAGA is not supporting the SAGA_MLB path variable. So if you use a SAGA GIS Version >= 3.0.0 the module library is not recognized. You must call rsaga.env using the rsaga.env(modules = saga$sagaModPath) assuming that saga contains the returnPaths of linkSAGA In addition most recently the very promising Rsagacmd wrapper package is providing a new list oriented wrapping tool.

Examples

```
## Not run:

# call if you do not have any idea if and where SAGA GIS is installed
# it will return a list with the selected and available SAGA installations
# it prepares the system for running the selected SAGA version via RSAGA or CLI
linkSAGA()

# overriding the default environment of rsaga.env call

saga<-linkSAGA()
if (saga$exist) {
require(RSAGA)
RSAGA::rsaga.env(path = saga$installed$binDir[1],modules = saga$installed$moduleDir[1])
}

## End(Not run)
```

---

paramGRASSw *Usually for internally usage get 'GRASS GIS' and* rgrass7 *parameters on 'Windows' OS*

Description

Initialize the enviroment variables on a 'Windows' OS for using 'GRASS GIS' via rgrass7

Usage

```
paramGRASSw(set_default_GRASS7 = NULL, DL = "C:", ver_select = FALSE,
  quiet = TRUE)
```

Arguments

set_default_GRASS7	default = NULL forces a full search for 'GRASS GIS' binaries. You may alternatively provide a vector containing pathes and keywords. c("C:/OSGeo4W64","grass-7.0.5","osgeo4w") is valid for a typical osgeo4w installation.
DL	raster or sp object
ver_select	boolean default is FALSE. If there is more than one 'SAGA GIS' installation and ver_select = TRUE the user can select interactively the preferred 'SAGA GIS' version
quiet	boolean switch for supressing console messages default is TRUE

Details

The concept is very straightforward but for an all days usage pretty helpful. You need to provide a raster or a sp object. The derived properties are used to initialize a temporary but static rgrass7 environment. During the rsession you will have full access to GRASS7 both via the wrapper package as well as the command line. paramGRASSw initializes the usage of GRASS7.

Examples

```
## Not run:
# automatic retrieval of valid 'GRASS GIS' environment settings
# if more than one is found the user has to choose.
paramGRASSw()

# typical OSGeo4W64 installation
paramGRASSw(c("C:/OSGeo4W64","grass-7.0.5","osgeo4W"))

## End(Not run)
```

`paramGRASSx` *Usually for internally usage, get 'GRASS GIS' and* `rgrass7` *parameters on 'Linux' OS*

Description

Initialize and set up rgrass7 for 'Linux'

Usage

```
paramGRASSx(set_default_GRASS7 = NULL, MP = "/usr",
  ver_select = FALSE, quiet = TRUE)
```

Arguments

`set_default_GRASS7`	default = NULL will force a search for 'GRASS GIS' You may provide a valid combination as c("/usr/lib/grass74","7.4.1","grass74")
`MP`	mount point to be searched. default is "usr"
`ver_select`	if TRUE you must interactivley selcect between alternative installations
`quiet`	boolean switch for supressing console messages default is TRUE

Details

During the rsession you will have full access to GRASS7 GIS via the rgrass7 wrappe. Additionally you may use also use the API calls of GRASS7 via the command line.

Examples

```
## Not run:
# automatic retrieval of the GRASS7 enviroment settings
paramGRASSx()


# typical stand_alone installation
paramGRASSx("/usr/bin/grass72")

# typical user defined installation (compiled sources)
paramGRASSx("/usr/local/bin/grass72")

## End(Not run)
```

`parseOTBAlgorithms` *Get OTB modules*

Description

retrieve the OTB module folder content and parses the module names

Usage

```
parseOTBAlgorithms(gili = NULL)
```

Arguments

`gili`	optional gis linkage as done by 'linkOTB()"

Examples

```
## Not run:
 ## link to the OTB binaries
   otbLinks<-link2GI::linkOTB()
   path_OTB<-otbLinks$pathOTB
 ##p arse all modules
algo<-parseOTBAlgorithms(gili = otbLinks)

##print the list
print(algo)

## End(Not run)
```

parseOTBFunction *Get OTB function calls*

Description

retrieve the choosen function and parses all arguments with the defaults

Usage

```
parseOTBFunction(algos = NULL, gili = NULL)
```

Arguments

algos	number of the algorithm as provided by 'getOTBAlgorithm'
gili	optional gis linkage as done by 'linkOTB()'

Examples

```
## Not run:
otbLinks<-link2GI::linkOTB()
if (otbLinks$exist) {

path_OTB<-otbLinks$pathOTB


## parse all modules
algo<-parseOTBAlgorithms(gili = otbLinks)


## take edge detection
otb_algorithm<-algo[27]
algo_cmd<-parseOTBFunction(algo = otb_algorithm,gili = otbLinks)
## print the current command
print(algo_cmd)
}

############
### usecase
############

## link to OTB
otblink<-link2GI::linkOTB()
path_OTB<-otblink$pathOTB

## get data
setwd(tempdir())
## get some typical data as provided by the authority
url<-"http://www.ldbv.bayern.de/file/zip/5619/DOP%2040_CIR.zip"
res <- curl::curl_download(url, "testdata.zip")
```

```
unzip(res,junkpaths = TRUE,overwrite = TRUE)

## get all available OTB modules
algo<-parseOTBAlgorithms(gili = otblink)

## for the example we use the edge detection,
## because of the windows call via a batch file
## we have to distinguish the module name
ifelse(Sys.info()["sysname"]=="Windows",
algo_keyword<- "EdgeExtraction.bat",
algo_keyword<- "EdgeExtraction")

# write it to a variable
otb_algorithm<-algo[algo[]==algo_keyword]
# now create the command list
algo_cmd<-parseOTBFunction(algo = otb_algorithm,gili = otblink)

## define the current run arguments
algo_cmd$`-in`<- file.path(getwd(),"4490600_5321400.tif")
algo_cmd$`-filter`<- "sobel"

## create out name
outName<-paste0(getwd(),"/out",algo_cmd$`-filter`,".tif")
algo_cmd$`-out`<- outName

## generate full command
command<-paste(paste0(path_OTB,"otbcli_",otb_algorithm," "),
               paste(names(algo_cmd),algo_cmd,collapse = " "))

## make the system call
system(command,intern = TRUE)

##create raster
retStack<-assign(outName,raster::raster(outName))

## plot raster
raster::plot(retStack)

## End(Not run)
##+##
```

`setenvGRASSw` *Usually for internally usage, create valid 'GRASS GIS 7.xx' rsession environment settings according to the selected GRASS GIS 7.x and Windows Version*

Description

Initializes and set up access to 'GRASS GIS 7.xx' via the rgrass7 wrapper or command line packages. Set and returns all necessary environment variables and additionally returns the GISBASE directory as string.

Usage

```
setenvGRASSw(root_GRASS = NULL, grass_version = NULL,
  installation_type = NULL, jpgmem = 1e+06, quiet = TRUE)
```

Arguments

root_GRASS	grass root directory i.e. "C:\OSGEO4~1",
grass_version	grass version name i.e. "grass-7.0.5"
installation_type	two options "osgeo4w" as installed by the 'OSGeo4W'-installer and "NSIS" that is typical for a stand_alone installtion of 'GRASS GIS'.
jpgmem	jpeg2000 memory allocation size. Default is 1000000
quiet	boolean switch for supressing console messages default is TRUE

Author(s)

Chris Reudenbach

Examples

```
## Not run:
# set choosen'GRASS GIS' installation folders
setenvGRASSw(root_GRASS = "C:\\PROGRA~1\\QGIS2~1.18",
             grass_version =  "grass-7.2.1",
             installation_type =  "osgeo4W")

## End(Not run)
```

---

setenvOTB *Usually for internally usage, initializes and set up access to the 'OTB' command line interface*

Description

Initializes and set up access to the 'OTB' command line interface

Usage

```
setenvOTB(bin_OTB = NULL, root_OTB = NULL)
```

Arguments

bin_OTB	string contains the path to the 'OTB' binaries
root_OTB	string contains the full string to the root folder containing the 'OTB' installation'

Value

Adds 'OTB' pathes to the enviroment and creates the variable global string variable `otbCmd`, that contains the path to the 'OTB' binaries.

Examples

```
## Not run:
## example for the most common default OSGeo4W64 installation of OTB
setenvOTB(bin_OTB = "C:\\OSGeo4W64\\bin\\",
          root_OTB = "C:\\OSGeo4W64")

## End(Not run)
```

`sf2gvec` *Write sf object to GRASS 7 vector utilising an existing or creating a new GRASS7 environment*

Description

Write sf object to GRASS 7 vector utilising an existing or creating a new GRASS7 environment

Usage

```
sf2gvec(x, obj_name, gisdbase, location, gisdbase_exist = FALSE)
```

Arguments

`x`	`sf` object corresponding to the settings of the corresponding GRASS container
`obj_name`	name of GRASS layer
`gisdbase`	GRASS gisDbase folder
`location`	GRASS location name containing `obj_name`)
`gisdbase_exist`	logical switch if the GRASS gisdbase folder exist default is TRUE

Note

have a look at the `sf` capabilities to write direct to sqlite

Author(s)

Chris Reudenbach

Examples

```
## Not run:
## example
# get meuse data as sf object
require(sf)
require(sp)
data(meuse)
meuse_sf = st_as_sf(meuse,
                    coords = c("x", "y"),
                    crs = 28992,
                    agr = "constant")

# write data to GRASS and create gisdbase
sf2gvec(x = meuse_sf,
          obj_name = "meuse_R-G",
          gisdbase = "~/temp3",
          location = "project1")

# read from existing GRASS
gvec2sf(x = meuse_sf,
          obj_name = "meuse_R-G",
          gisdbase = "~/temp3",
          location = "project1")

## End(Not run)
```

Index