Package 'gitProjectExtras'

December 2, 2015

Type Package

Title Extra Functions for Integrating with git based projects

Version 2.32

Date 2015-11-09

Author Douglas Kelley; Rhys Whitley

Maintainer Douglas Kelley <douglas.kelley@mq.edu.au>

Imports raster, ncdf4

Description {Allows linking code and outputs with information about git revision numbers and storage locations. Also contains tools for setting up a proper project structure and common git commands that can be called within R.}

License GPL-2

R topics documented:

`gitProjectExtras-package`
Extra Functions for Integrating with git tracked projects

Description

Allows linking code and outputs with information about git revision numbers and remote location. Also contains tools for setting up a project structure when freshly cloned. Feel free to clone and contribute to the package: https://bitbucket.org/teambcd/gitprojectextras.

Details

Package:	gitProjectExtras
Type:	Package
Version:	1.0
Date:	2014-10-15
License:	GPL 2
Website:	https://bitbucket.org/teambcd/gitprojectextras

Author(s)

Douglas Kelley <douglas.i.kelley@gmail.com>, Rhys Whitley <rhys.whitley@gmail.com>

Examples

```
## From command line
## Not run:
>> git clone https://bitbucket.org/teambcd/gitprojectextras.git
>> cd gitprojectextras
>> R

## End(Not run)
## From R
library("gitProjectExtras")
makeDir("tryOut")
setwd("tryOut")
setupProjectStructure()

fileConn <- file("libs/yay.r")
writeLines(c(
"yay <- function(toFile=FALSE) { ",
" funName = match.call.string() ",
" gitRev  = gitVersionNumber() ",
" getURL  = gitRemoteURL() ",
" ",
" if (toFile) ",
" pdf(paste('figs/egFig_',funName,",
"    '.pdf', sep='')) ",
" plot(0) ",
" mtext(paste('git URL:', getURL)) ",
" mtext(paste('git rev:', gitRev),",
"   side=1,line=-1) ",
" ",
" if (toFile) dev.off() ",
   "} "),
 fileConn)
```

```
close(fileConn)

sourceAllLibs()

yay()

yay(TRUE)

list.files("figs/")
```

`git` *git Commands*

Description

Runs git commands from within R.

Usage

```
git(commands, ...)

git.push(...)
git.pull(...)

git.status(...)

git.diff(files = '.', ...)
git.add(files = '.', ...)
git.commit(message, messageType = '-m', ...)

git.addCommit(files = '.', message, messageType = '-m',
              addOptions = '', commitOptions = '')

git.checkout(...)
git.log(...)
```

Arguments

`commands`	string (or string vector) describing commands being sent to git
`...`	additional commands passed to `git` and ultimatly to `system`
`files`	vector or files to be added
`message`	commit message
`messageType`	commit message type, default '-m' for command line message or '-F' for file. See git documentation for more options.
`addOptions`	git commands used by git.add
`commitOptions`	git commands used by git.commit

Details

`git` is essentially the same as `system('git <<commands>>')` `git.xxx` works much the same as `git xxx` in command line) Arguments are also passed to `system`. It is therefore possible to intern the printout (i.e, capture the output of the command as an R character vector rather than print to console). For `git.log()`, output is to console and interned to R vector. See example below

Note

These functions invoke commands and may not work on all OSs

Author(s)

Douglas Kelley <douglas.i.kelley@gmail.com>

See Also

`gitRemoteURL` `gitVersionNumber`

Examples

```
## setup project
makeDir('yayandwow')
setwd('yayandwow')
git('init')
git.status()

## simple outpur file function
write.text <- function(txt, filename, ...) cat(txt, file = filename, sep = "\n")

## write some example files
write.text('yay', 'yay.txt', sep ='')
write.text('wow', 'wow.txt', sep ='')

## commit these files
git.add() ## adds all files
git.commit('yay and wow')

## add new file and change old one
write.text('yayandwow', 'newFile.txt', sep ='')
write.text('wownotyay', 'wow.txt', sep ='')

## show status and commit new file
git.status()
git.add('newFile.txt')
git.commit('yayandwow')

## show status and previous commits
git.status()
gitVersionNumber()
log = git.log()

print(log)
print(log[[1]])
summary(log)
```

`gitRemoteURL` *Return Git Remote URL*

Description

Provides the URL of the remote git repo.

Usage

```
gitRemoteURL(gitLoc = ".git")
```

Arguments

`gitLoc`	name (and optional path location) of revision information store. Defualt is '.git'.

Details

If path is not provided, `gitRemoteURL` will search current directory, and then search each directory out from current position in directory tree until gitLoc is found

Value

The url of the remote location of the git repo. If remote location has not been added, returns NA

Author(s)

Douglas Kelley <douglas.i.kelley@gmail.com>

See Also

`link{gitVersionNumber.Rd}`

Examples

```
gitRemoteURL()
makeDir("yayandwow")
setwd("yayandwow")
gitRemoteURL('../.git')
gitRemoteURL()
```

gitVersionNumber *Return Git Version Number*

Description

Returns the current git revision code.

Usage

```
gitVersionNumber(short = TRUE, gitLoc = ".git")
```

Arguments

short	logical. If TRUE, returns the short (i.e 7 digit) revision code. If FALSE, returns full code
gitLoc	name (and optional path location) of revision information store. Default is '.git'.

Value

The revision code for the last git commit.

Note

Does not indicate if modification has been made since last commit.

Author(s)

Douglas Kelley <douglas.i.kelley@gmail.com>

See Also

gitRemoteURL

Examples

```
gitVersionNumber()
makeDir("yayandwow")
setwd("yayandwow")

# The next two calls return the same value
gitVersionNumber(gitLoc = '../.git')
gitVersionNumber()

# Return Long revision number
gitVersionNumber(FALSE)
```

gitWatermark *git Plot Watermark*

Description

Adds git information to a plotting window as a watermark.

Usage

```
gitWatermark(VersionNumber = TRUE, URL = TRUE,
             timeAndDate = TRUE, comment = '',
             x = 0.5, y = 0.5,
             col = 'black',transparency = 0.8, srt = 45, ...)
```

Arguments

VersionNumber	logical. Default of TRUE adds git version number to watermark
URL	logical. Default of TRUE adds git remote URL to watermark
timeAndDate	logical. Default of TRUE adds date and to watermark
comment	additional text to be added to watermark
x, y	corridinates where watermark will be added as fraction of figure size
col	colour of watermarked text
transparency	transparency of watermarked text, between 0-1 woth 0 being no transparency and 1 being completely see through
srt	angle of watermark. see text
...	Additional arguments passed to text

Details

dev.off.gitWatermark adds watermark when figure is closed

Author(s)

Douglas Kelley <douglas.i.kelley@gmail.com>

See Also

dev.off

Examples

```
## assuming already in a git repo
plot(0)
gitWatermark()

dev.new()
par(mfrow = c(2,2))
for (i in 1:3) plot(0)
plot(1:5, 6:10)
```

```
gitWatermark(comment = 'all info')
gitWatermark(TRUE, FALSE, FALSE, comment = 'git rev no.', x = 0.25, srt = 90)
gitWatermark(col = 'green', comment = 'green stamp', x = 0.75, srt = -90)
gitWatermark(transparency = 0, comment = 'not see through', x = 0.25, y = 0.25, srt =

pdf('yay.pdf')
par(mfrow = c(2,2))
plot(0)
gitWatermark(srt = 0)
dev.off()
```

`makeDir` *Make Directory*

Description

If directory does not exist, then create it. Essentially works as `dir.create` but will not cause an error if the directory already exists.

Usage

```
makeDir(fname, ...)
```

Arguments

`fname`	name (inc. path) of directory to be made
`...`	Arguements passed to `dir.create`

Details

Note, with R3.0+, this function is probably obsolute. Just use dir.create(path, showWarnings=FALSE) instead.

Author(s)

Douglas Kelley <douglas.i.kelley@gmail.com>

See Also

`makeGlobDir` `dir.create`

Examples

```
list.dirs()
makeDir('yayandwow')
list.dirs()
```

makeGlobDir *Make Global Directory*

Description

If it does not already exist, makes directory. Whether directory exists or not, assigns directory path to a global variable.

Usage

```
makeGlobDir(obj, fname, ...)
```

Arguments

obj	name of global variable in which directory path will be stored
fname	directory name/path used by makeDir
...	Arguements passed to dir.create

Note

does not make directories recursively

Author(s)

Douglas Kelley <douglas.i.kelley@gmail.com>

See Also

makeDir dir.create

Examples

```
list.dirs()
makeGlobDir('wowandyay', 'yayandwow')
list.dirs()
```

match.call.string *Match Call String*

Description

Returns the name of the current function.

Usage

```
match.call.string(n = 1, call = sys.call(sys.parent(n = n)), ...)
```

Arguments

n	call position used by match.call. Default n = 1 returns the function name from which match.call.string is called. n > 1 returns the function name n-1 up the call tree
call	an unevaluated call to the function specified by definition, as generated by call and used by match.call
...	Additional arguments passed to match.call

Value

function name from which match.call.string is called. Returns "N/A" if not called within a function or if n is greater than the call tree

Author(s)

Douglas Kelley <douglas.i.kelley@gmail.com>

See Also

match.call

Examples

```
match.call.string()
yayandwow <- function()
    match.call.string()

yayandwow()

#########################

yay <- function(...) {
wow <- function(...) match.call.string(...)
wow(...)
}

yay()
yay(n=1)
yay(n=2)
yay(n=3)
```

---

setupProjectStructure

Setup Project Structure

Description

Adds directories required for a project and places the path in global variables.

Usage

```
setupProjectStructure(namess = paste(dirn, "_dir", sep = ""), dirn = c("outputs"
```

Arguments

namess	Names of the global variables directory path will be stores. If not defined, variables are named as '<directory name>_dir'
dirn	directories to be created, with paths relative to current working directory. By default, creates folders for 'outputs', 'data', 'temp', 'figs', 'docs'

Value

Sets "namess" as globally callable variables

Author(s)

Douglas Kelley <douglas.i.kelley@gmail.com>

Examples

```
printFiles <- function(files)
lapply(files,function(i) print(i) )

dir.create("yay")
setwd("yay")

print( list.files() )
setupProjectStructure()
print( list.files() )
printFiles( c(outputs_dir, data_dir, temp_dir, figs_dir, docs_dir))

setwd(outputs_dir)
setwd("..")

#########################################################################

setupProjectStructure(namess = c("yayDir", "wowDir"), dirn = c("yay", "wow"))

list.files()
printFiles( c("yay", "wow"))

setwd(yayDir)
setwd("..")
```

---

sourceAllLibs *Source All Libraries*

Description

Sources all R code in a given directory.

Usage

```
sourceAllLibs(path = "libs/", trace = TRUE, ...)
```

Arguments

path	Path to the directory containing code. Default is 'libs'
trace	logical. Default of TRUE lists files as they are sourced. Useful for bug hunting.
...	Arguements passed to source

Details

If there is a sourcing error when trying to source of a file, then the error message is displayed and the file is skipped over. If trace=TRUE, the filename will be listed above the error

Note

This is taken straight from the source help page example.

Author(s)

R Core development team

See Also

source

---

writeRaster.gitInfo

Write raster data to a file with git information

Description

An extension of writeRaster in package raster which adds projects git information to outputted netcdf raster file.

Usage

```
writeRaster.gitInfo(x, filename,
                    VersionNumber = TRUE, URL = TRUE, comment = NULL, ...)
```

Arguments

x	raster, stack or brick object
filename	Output filename
VersionNumber	logical. Default of TRUE adds git version number attribute
URL	logical. Default of TRUE adds git remote URL to attribute
comment	additional text to be added to watermark
...	Additional arguments passed to writeRaster

Details

Works exactly the same way as writeRaster on netcdf files but adds git information is added as comment attributes. Note, unlike gitWatermark, there is no additional arguement for adding a time and date stamp, and this is already performed by writeRaster

Author(s)

Douglas Kelley <douglas.i.kelley@gmail.com>

See Also

raster writeRaster gitWatermark dev.off.gitWatermark

Examples

```
require(raster)
require(ncdf4)
r <- raster(system.file("external/test.grd", package = "raster"))

# take a small part
r <- crop(r, extent(179880, 180800, 329880, 330840) )


printAttributes <- function(filename) {
    nc  = nc_open("allint.nc")
    att = ncatt_get(nc, 0)
    print(att)
}
# write to an integer binary file
rf <- writeRaster.gitInfo(r, filename = "allint.nc", datatype = 'INT4S',
                          overwrite = TRUE)
printAttributes('allint.nc')


# make a brick and save multi-layer file
b <- brick(r, sqrt(r))
bf <- writeRaster.gitInfo(b, filename="multi.nc", URL = FALSE,
                          comment = c(yay = 'wow'), overwrite=TRUE)
printAttributes('multi.nc')
```

Index