Package ‘adapr’

November 30, 2017

Type Package
Title Implementation of an Accountable Data Analysis Process
Version 2.0.0
Date 2017-11-28
Author Jon Gelfond, Martin Goros
Maintainer Jonathan Gelfond <gelfondjal@uthscsa.edu>

BugReports https://github.com/gelfondjal/adapr/issues
Description Tracks reading and writing within R scripts that are organized into
a directed acyclic graph. Contains an interactive shiny application adaprApp().
Uses git2r package, Git and file hashes to track version histories of input
and output. See package vignette for how to get started. V1.02 adds parallel
execution of project scripts and function map in vignette. Makes project
specification argument last in order. V2.0 adds project specific libraries, packrat op-
tion, and adaprSheet().

License LGPL-2
Depends R (>= 3.1)
Imports gdata, plotly, ggplot2, shiny, shinydashboard, knitr,
  rmarkdown, igraph, digest, devtools, plyr, git2r, methods,
  parallel, versions, archivist, doParallel
Suggests packrat
RoxygenNote 6.0.1
VignetteBuilder knitr
NeedsCompilation no
Repository CRAN

Date/Publication 2017-11-30 00:19:12 UTC

R topics documented:

  adapr-package .......................... 5
  adaprApp ................................ 11
R topics documented:

adaprDependencies ................................................. 11
adaprHomeDir .......................................................... 12
adaprInstall .......................................................... 12
adaprSheet ............................................................ 13
adaprUpdate ........................................................... 14
addPackage ............................................................ 14
AppLoadFlex ........................................................... 15
arcRead ................................................................. 16
arcWrite ................................................................. 16
browsePubFiles ......................................................... 17
checkAdaprHashAlgo .................................................. 18
checkFileHash ......................................................... 19
checkFileHashSource .................................................. 19
checkFileMtimeSource ................................................ 20
checkRmdMode .......................................................... 21
checkVersion .......................................................... 22
commit2char ........................................................... 22
commitProject .......................................................... 23
condenseFileInfo ......................................................... 24
createFileInfo .......................................................... 24
createMarkdown ......................................................... 25
createProgramGraph .................................................... 26
create_source_file_dir .................................................. 27
dataDir ................................................................. 28
defaultAdaprSetup ...................................................... 28
dependency-class ....................................................... 29
Digest ................................................................. 29
fileInfoProjects ......................................................... 30
finalize_dependency ..................................................... 31
firstProject ............................................................ 31
getAdaprOptions ....................................................... 32
getDepSubgraph ......................................................... 33
getFileInfo .............................................................. 33
getFileSysTime .......................................................... 34
getLibrary ............................................................... 35
getProject .............................................................. 35
getProjectInfo .......................................................... 36
getProjectInfoSI ......................................................... 37
getProjectLibrary ....................................................... 37
getProjectPath ......................................................... 38
getProjectPublishPath .................................................. 39
getProjectSwapPath .................................................... 39
getPubResults .......................................................... 40
getSourceInfo .......................................................... 41
getUpstream ........................................................... 41
get_orchard ............................................................ 42
gitAdd ................................................................. 42
gitCommit .............................................................. 43
topics documented:

- `gitConfigure` ........................................... 44
- `gitConfigureTest` ..................................... 44
- `gitHistorySearch` ..................................... 45
- `gitIgnoreLibrary` ..................................... 46
- `gitInfo` ................................................ 46
- `gitInit` ................................................ 47
- `gitProvenance` ......................................... 48
- `git_path` ............................................... 48
- `Graph` .................................................. 49
- `graphProject` ......................................... 50
- `guess.readfcn` ......................................... 50
- `guessWriteFcn` ........................................ 51
- `idPackages` ............................................ 52
- `idSync` ................................................ 52
- `importData` ............................................ 53
- `initialize_dependency_info` .......................... 54
- `initProject` ............................................ 54
- `install` ................................................ 55
- `installLibrary` ........................................ 56
- `installProjectPackages` ............................... 57
- `Library` ............................................... 58
- `listBranches` .......................................... 58
- `listDatafiles` .......................................... 59
- `listProjects` .......................................... 60
- `listScripts` ............................................ 60
- `Load.branch` ........................................... 61
- `loadAdaprTest` ........................................ 61
- `loadFlex` .............................................. 62
- `loadInstallLibraryFile` ............................... 63
- `makeDependencyGraphObj` ............................. 63
- `makeFunction` ......................................... 64
- `makeHyperlink` ........................................ 65
- `makeScript` ............................................ 65
- `makeSummaryGraph` .................................... 66
- `monitorParallelSync` .................................. 67
- `openProjectList` ...................................... 68
- `openScript` ............................................ 68
- `parallelSync` ......................................... 69
- `path.expand.2` ........................................ 70
- `plantOrchard` ......................................... 70
- `plantTree` ............................................. 71
- `programIOTable` ...................................... 72
- `project.directory.tree` ............................... 72
- `projectReportMarkdown` ................................ 73
- `projectReportSend` .................................... 74
- `publishResults` ........................................ 75
- `pullSourceInfo` ....................................... 75
- `rapidPlot` ............................................. 76
R topics documented:

- Read
- Read.cap
- read.dependency
- readDependency
- readLibrary
- ReadTrack
- redirectTree
- relocateProject
- removePackage
- removeProject
- removeScript
- renderRmd
- reportProject
- resultsDir
- reworkProjectPath
- runScript
- runSourceDirectory
- scriptLoader
- scriptSubgraph
- searchScripts
- sendBranch
- sendBranchSI
- setAdaprOptions
- setProject
- showProject
- showResults
- sourceSyncSI
- sourceSyncSIload
- sproutProgram
- swapProjectPath
- syncProject
- syncTest
- syncTestPI
- syncTestProject
- syncTestSI
- syncTrunk
- updateAdaprConfig
- Write
- Write.cap
- write.dependency
- WriteTrack
Description

Tracks reading and writing within R scripts that are organized into a directed acyclic graph. Contains an interactive shiny application adaprApp(). Uses git2r package, Git and file hashes to track version histories of input and output. See package vignette for how to get started. V1.02 adds parallel execution of project scripts and function map in vignette. Makes project specification argument last in order. V2.0 adds project specific libraries, packrat option, and adaprSheet().

Details

The DESCRIPTION file:

<table>
<thead>
<tr>
<th>Package</th>
<th>adapr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Package</td>
</tr>
<tr>
<td>Title</td>
<td>Implementation of an Accountable Data Analysis Process</td>
</tr>
<tr>
<td>Version</td>
<td>2.0.0</td>
</tr>
<tr>
<td>Date</td>
<td>2017-11-28</td>
</tr>
<tr>
<td>Author</td>
<td>Jon Gelfond, Martin Goros</td>
</tr>
<tr>
<td>Maintainer</td>
<td>Jonathan Gelfond <a href="mailto:gelfondjal@uthscsa.edu">gelfondjal@uthscsa.edu</a></td>
</tr>
<tr>
<td>BugReports</td>
<td><a href="https://github.com/gelfondjal/adapr/issues">https://github.com/gelfondjal/adapr/issues</a></td>
</tr>
<tr>
<td>Description</td>
<td>Tracks reading and writing within R scripts that are organized into a directed acyclic graph. Contains an interactive shiny application adaprApp(). Uses git2r package, Git and file hashes to track version histories of input and output. See package vignette for how to get started. V1.02 adds parallel execution of project scripts and function map in vignette. Makes project specification argument last in order. V2.0 adds project specific libraries, packrat option, and adaprSheet().</td>
</tr>
<tr>
<td>License</td>
<td>LGPL-2</td>
</tr>
<tr>
<td>Depends</td>
<td>R (&gt;= 3.1)</td>
</tr>
<tr>
<td>Imports</td>
<td>gdata, plotly, ggplot2, shiny, shinydashboard, knitr, rmarkdown, igraph, digest, devtools, plyr, git2r, method, parallel, versions, archivist, doParallel</td>
</tr>
<tr>
<td>Suggests</td>
<td>packrat</td>
</tr>
<tr>
<td>RoxygenNote</td>
<td>6.0.1</td>
</tr>
<tr>
<td>VignetteBuilder</td>
<td>knitr</td>
</tr>
</tbody>
</table>

Index of help topics:

- AppLoadFlex: Loads a single R object from file for a R Shiny app
- Digest: Digest files from (digest package)
- Graph: Write object and capture file information
- Library: Installs and loads library specific to a project
- Load.branch: Loads a single R object from file
- Read: Read data and capture the file information within dependency object
- Read.cap: Lower level function that reads data and capture the file information within dependency object
- ReadTrack: Tracks files that read by functions not in...
adapr and captures the file information within dependency object

Write
Write object and capture file information
Write.cap
Lower level function that writes data and captures the file information within dependency object
WriteTrack
Tracks files written by functions not in adapr and captures the file information within dependency object

adapr-package
Implementation of an Accountable Data Analysis Process

adaprApp
Launches Main app
adaprDependencies
Returns character string of adapr R package import dependencies
adaprHomeDir
Identify adapr Home directory. Location of options file and project listing.
adaprInstall
Install adapr in a library adaprInstall.R
adaprSheet
Browse adapr cheat sheet
adaprUpdate
Updates latest adapr from github.
addPackage
Add R package to a project
arcRead
Loads R object from archivist repository within the results directory of another adapr Script
arcWrite
Saves R object to archivist repository in results directory of adapr Script
browsePubFiles
Browses publication table for editing
checkAdaprHashAlgo
Checks or changes the specified adapr hash algorithm (adaprHashAlgo option)
checkFileHash
Checks the consistency of the dependency directory with the files within the file system
checkFileHashSource
Checks the consistency of the dependency directory with the files within the file system. Reports the source scripts that need to be updated.
checkFileMtimeSource
Checks the consistency of the dependency directory with the files within the file system. Reports the source scripts that need to be updated!
checkRmdMode
Checks whether interactive R markdown session is ON
checkVersion
Check install of package of specific version
commit2char
git2r commit class to character converter
commitProject
Git commit of project.
condenseFileInfo
Generate condensed information about files from dependency object
createFileInfo
Creates a list containing information about a file
createMarkdown
Creates an Rmarkdown file in specified directory
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createProgramGraph</td>
<td>Make plot of project programs only Summarize all programs. Sync status is assessed and indicated.</td>
</tr>
<tr>
<td>create_source_file_dir</td>
<td>Create source file directories</td>
</tr>
<tr>
<td>dataDir</td>
<td>Returns project's data directory, allows relative directories. Used within an R script.</td>
</tr>
<tr>
<td>defaultAdaprSetup</td>
<td>Set up adapr 1st time</td>
</tr>
<tr>
<td>dependency-class</td>
<td>Dependency class</td>
</tr>
<tr>
<td>fileInfoProjects</td>
<td>List project file information disk space, modification timespan, days inactive</td>
</tr>
<tr>
<td>finalize_dependency</td>
<td>Writes dependency data to file in &quot;Dependency&quot; directory</td>
</tr>
<tr>
<td>firstProject</td>
<td>Create first project</td>
</tr>
<tr>
<td>getAdaprOptions</td>
<td>Returns the primary hub file with project location and id information</td>
</tr>
<tr>
<td>getDepSubgraph</td>
<td>Produces subgraph of dependencies of R script</td>
</tr>
<tr>
<td>getFileInfo</td>
<td>Retrieve the file info for the file by name OR from the data subdirectory data</td>
</tr>
<tr>
<td>getFileSysTime</td>
<td>Retrun time of file system</td>
</tr>
<tr>
<td>getLibrary</td>
<td>Get library for a project</td>
</tr>
<tr>
<td>getProject</td>
<td>Returns the adapr project in R option &quot;adaprProject&quot;</td>
</tr>
<tr>
<td>getProjectInfo</td>
<td>Acquire all dependencies related to a project, Generate graph of project dependencies, Get all file information related to project</td>
</tr>
<tr>
<td>getProjectInfoSI</td>
<td>Given source_info object, retrieves project information</td>
</tr>
<tr>
<td>getProjectLibrary</td>
<td>Given Project name, Return project library directory</td>
</tr>
<tr>
<td>getProjectPath</td>
<td>Given Project id, Return project directory</td>
</tr>
<tr>
<td>getProjectPublishPath</td>
<td>Given Project name, Return project publish directory</td>
</tr>
<tr>
<td>getProjectSwapPath</td>
<td>Retrieve project swap directory</td>
</tr>
<tr>
<td>getPubResults</td>
<td>Read result filepaths to publish</td>
</tr>
<tr>
<td>getSourceInfo</td>
<td>Returns the information related to the adapr script</td>
</tr>
<tr>
<td>getUpstream</td>
<td>Identifies dependencies in a DAG</td>
</tr>
<tr>
<td>get_orchard</td>
<td>Returns the primary hub file with project location and id information</td>
</tr>
<tr>
<td>gitAdd</td>
<td>git add to stage the file</td>
</tr>
<tr>
<td>gitCommit</td>
<td>git commit. Requires git installation.</td>
</tr>
<tr>
<td>gitConfigure</td>
<td>Configure user.name and email for git.</td>
</tr>
<tr>
<td>gitConfigureTest</td>
<td>Checks git configuration. Requires git installation</td>
</tr>
<tr>
<td>gitHistorySearch</td>
<td>Performes git history search</td>
</tr>
<tr>
<td>gitIgnoreLibrary</td>
<td>Git ignore the library file</td>
</tr>
<tr>
<td>gitInfo</td>
<td>Retrieves the information from git about a file</td>
</tr>
</tbody>
</table>
gitInit   Initiate git
gitProvenance  Identify git provenance of file within a project
git_path   Find path for git executable
graphProject  Make plot of project programs only. Summarize all programs. Sync status is assessed and indicated.
guess.read.fcn Return function for reading common file types
guessWriteFcn  Return function for writing common file types
idPackages  Lower level function that captures packages that are explicitly loaded, not loaded automatically
idSync   Returns the source files needed to repair synchrony
importData  Import selected file into the project data directory
initProject  initialize project
initialize_dependency_info  Initializes dependency object source_info
install  Install package of specific version
installLibrary  Install package of specific version
installProjectPackages  Installs all packages
listBranches  Lists the branches available for loading in the adapt project
listDatafiles  Lists the data files available for reading in the adapt project
listProjects  List projects
listScripts  Lists the R scripts in the adapt project
loadAdaprTest  Create adaptrTest example project
loadFlex  Loads a single R object from file, more flexible than Load.branch or base::load
loadInstallLibraryFile  This function is no longer supported. Loads libraries within the file library.list.file
makeDependencyGraphObject  Creates a graph object from a dependency object
makeFunction  Generates the shell of a R function that is project specific in support_functions folder
makeHyperlink  Makes HTML hyper link
makeScript  Generates the shell of a code that is project specific
makeSummaryGraph  Make.summary graph of projects based on files in dependency directory
monitorParallelSync  Experimental (use with caution) track parallelSync while in progress
openProjectList  Browses orchard in file system
openScript
Opens script from a project with default R program. Can open markdown files as well.

parallelSync
Experimental (use with caution) parallel synchronization of project. Takes advantage of directed acyclic graph structure to run R script processes in parallel.

path.expand.2
Location of options file and project listing. Swap / for \ in path expand.

plantOrchard
Create project hub files in root directory

plantTree
initialize project

programIOTable
Create program io table

project.directory.tree
Project directory tree structure contains the relative directory structure included analysis, data, texidr, dependency.dir, support functions, and library bank locations

projectReportMarkdown
Make plot of network within html documents. Summarize all programs.

projectReportSend
Make plot of network within html documents. Summarize all programs. Make a readme file at top project directory copy to target.directory. Uses pander and pandoc unlike project_report

publishResults
Read in results to publish & Copies results to the project's publication directory

pullSourceInfo
Create source_info from project.id

rapidPlot
Make project graph with sync status already computed.

read.dependency
Lower level function that reads the script dependency data from file

readDependency
Collect trees from dependency directory

readLibrary
Read library file

redirectTree
Lower level function that that changes project directory/publish directory or identifies imported project

relocateProject
changes project directory/publish directory/library locataion or identifies imported project

removePackage
Remove R package to a project

removeProject
Removes project from orchard, but doesn't delete project from file system

removeScript
Remove an R script from a project. Removes program, dependency, and results.

renderRmd
Renders and Rmarkdown file

reportProject
Make plot of network within html documents. Summarize all programs.

resultsDir
Returns project's results directory, allows relative directories. Only used within an R script, after create_source_file_dir.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reworkProjectPath</td>
<td>Lower level function that collects all trees in dependency.dir and changes the project path</td>
</tr>
<tr>
<td>runScript</td>
<td>Run an R script within a project using devtools::clean_source</td>
</tr>
<tr>
<td>runSourceDirectory</td>
<td>Runs all source files within the directory source.directory</td>
</tr>
<tr>
<td>scriptLoader</td>
<td>Initializes dependency object source_info</td>
</tr>
<tr>
<td>scriptSubgraph</td>
<td>Produces script only subgraph</td>
</tr>
<tr>
<td>searchScripts</td>
<td>Searches R scripts and R markdown files within a project.</td>
</tr>
<tr>
<td>sendBranch</td>
<td>Copy dependent programs to swap directory</td>
</tr>
<tr>
<td>sendBranchSI</td>
<td>Copy dependent programs to swap directory</td>
</tr>
<tr>
<td>setAdaprOptions</td>
<td>Returns Modifications the primary adapr option file</td>
</tr>
<tr>
<td>setProject</td>
<td>Checks or changes the specified adapr project in R option &quot;adaprProject&quot;</td>
</tr>
<tr>
<td>showProject</td>
<td>Opens project directory</td>
</tr>
<tr>
<td>showResults</td>
<td>Opens results directory of project or R script within a project</td>
</tr>
<tr>
<td>sourceSyncSI</td>
<td>Lower level function that synchronizes project by running necessary R scripts. Loads from source_info list.</td>
</tr>
<tr>
<td>sourceSyncSIUnload</td>
<td>Synchronize project by IDENTIFYING necessary R scripts</td>
</tr>
<tr>
<td>sproutProgram</td>
<td>Lower level function that generates the shell of a code that is project specific</td>
</tr>
<tr>
<td>swapProjectPath</td>
<td>Lower level function that takes list of dependency file data and changes the project path</td>
</tr>
<tr>
<td>syncProject</td>
<td>Checks the synchronization project and runs scripts needed for synchronization</td>
</tr>
<tr>
<td>syncTest</td>
<td>Lower level function that checks the synchrony of source files and their created objects</td>
</tr>
<tr>
<td>syncTestPI</td>
<td>Lower level function that tests the synchrony of files in dependency tree given project information.</td>
</tr>
<tr>
<td>syncTestProject</td>
<td>Tests the synchrony of files in dependency tree</td>
</tr>
<tr>
<td>syncTestSI</td>
<td>Tests the synchrony of files in dependency tree</td>
</tr>
<tr>
<td>syncTrunk</td>
<td>Partial project synchronization of dependencies to an Rscript. Runs only scripts needed for synchronization.</td>
</tr>
<tr>
<td>updateAdaprConfig</td>
<td>Updates the project list file to include project specific libraries.</td>
</tr>
<tr>
<td>write.dependency</td>
<td>Lower level function that writes the dependency object to file</td>
</tr>
</tbody>
</table>

Tracks reading and writing within R scripts that are organized into a directed acyclic graph.
**Author(s)**

Jon Gelfond, Martin Goros
Maintainer: Jonathan Gelfond &lt;gelfondjal@uthscsa.edu&gt;

**References**

None.

**Examples**

```r
#Digest()
```

---

### `adaprApp`

**Launched Main app**

**Description**

Launches Main app

**Usage**

`adaprApp()`

---

### `adaprDependencies`

**Returns character string of adapr R package import dependencies**

**Description**

Returns character string of adapr R package import dependencies

**Usage**

`adaprDependencies()`

**Value**

character vector of package names

**Examples**

```r
## Not run:
adaprDependencies()
## End(Not run)
```
adaprHomeDir

*Identify adapr Home directory. Location of options file and project listing.*

**Description**

Identify adapr Home directory. Location of options file and project listing.

**Usage**

```r
adaprHomeDir()
```

**Details**

This is automatically handled by defaultAdaprSetup(), but can be controlled in an "R profile" by adding adaprHomeDir R option (e.g., `options(adaprHomeDir="myPath")`).

**Value**

Full filepath to Adapr options and project listings directory.

**Examples**

```r
## Not run:
adaprHomeDir()

## End(Not run)
```

adaprInstall

*Install adapr in a library adaprInstall.R*

**Description**

Install adapr in a library adaprInstall.R

**Usage**

```r
adaprInstall(library.location = getProjectLibrary("adaprHome"), 
            betaTF = getAdaprOptions()$adaprBeta)
```

**Arguments**

- `library.location`  
  file path to library

- `betaTF`  
  logical indicating whether to install from github or CRAN.
**adaprSheet**

**Details**
Installs dependencies

**Value**
logical for successful creation or not

**Examples**
```r
## Not run:
adaprInstall(library.location=getProjectLibrary("adaprHome"), betaTF=FALSE)
```

## End(Not run)

---

**adaprSheet**  
*Browse adap cheat sheet*

**Description**
Browse adap cheat sheet

**Usage**
```r
adaprSheet()
```

**Details**
Opens pdf from R package within system

**Examples**
```r
## Not run:
# Requires pandoc location or RStudio
adaprSheet()
```

## End(Not run)
addPackage

Description

Add R package to a project

Usage

addPackage(project.id = getProject(), library.name, library.install = NA,
            library.specific = FALSE)

Arguments

- project.id: project.id to add R package to
- library.name: R package name to add
- library.install: Command to install library: NA for CRAN, bioC for bioconductor
- library.specific: logical indicate whether package is for specific R script

Details

Not for direct use. Installs and loads all packages

addUpdate

Description

Updates latest adapr from GitHub.

Usage

addUpdate()

Value

message from github

Examples

## Not run:
addUpdate()

## End(Not run)
**AppLoadFlex**

**Value**

Library information data

**Examples**

```r
## Not run:
addPackage("adaprHome","ggplot2")

## End(Not run)
```

---

**AppLoadFlex**  
*Loads a single R object from file for a R Shiny app*

**Description**

Loads a single R object from file for a R Shiny app

**Usage**

```r
AppLoadFlex(project.id = getProject(), path, file, read.fcn = readRDS, ...)
```

**Arguments**

- `project.id`: project name from which to load file
- `path`: directory that contains file to be loaded
- `file`: contains R object
- `read.fcn`: function to read the file, default readRDS
- `...`: arguments passed to read.fcn

**Value**

object for file that was read

**Examples**

```r
## Not run:
processed <- AppLoadFlex("adaprTest","Results/read_data.R","cardata.RData")

## End(Not run)
```
arcRead

*Description*

Loads R object from archivist repository within the results directory of another adapr Script

*Usage*

```r
arcRead(rscript = "read_data.R", description = "xyplot", project.id = getProject())
```

*Arguments*

- `rscript`: name of R script that loaded the function
- `description`: character description. Need for access with arcRead
- `project.id`: project to search within

*Details*

For use within R adapr script. This complements the use of arcWrite. The description should match the description in arcWrite.

*Examples*

```r
## Not run:
# In Script that writes the archivist object:
# arcWrite(myplot,"xyplot")

# In Script that reads the object:
processed <- arcRead("read_data.R","xyplot")

## End(Not run)
```

arcWrite

*Description*

Saves R object to archivist repository in results directory of adapr Script
browsePubFiles

Usage

arcWrite(Obj, description)

Arguments

Robj R object to be written
description character description. Need for access with aRead

Details

For use within R adapr script. Makes tags in archivist 'source:rscript' and 'aName:description'. Should be paired with arcRead().

Value

Value of object description

Examples

## Not run:
processed <- arcWrite(rnorm(100),"100 Gaussians")

## End(Not run)

browsePubFiles Browses publication table for editing

Description

Browses publication table for editing

Usage

browsePubFiles(project.id = getProject())

Arguments

project.id Project to publish

Details

File is in support directory/files_to_publish.csv

Value

dataframe of files to publish
checkAdaprHashAlgo

### Description

Checks or changes the specified adapr hash algorithm (adaprHashAlgo option)

### Usage

checkAdaprHashAlgo(hashAlgorithm = "")

### Arguments

hashAlgorithm characters specifying adaprHash algorithm if changing

### Details

Current default is sha1. If algorithm not recognized then will not change option.

### Value

value is specified algorithm or default algorithm

### Examples

````
## Not run:
checkAdaprHashAlgo()

## End(Not run)
```
checkFileHash

**Description**

Checks the consistency of the dependency directory with the files within the file system

**Usage**

```r
checkFileHash(dependency.dir = NULL, dependency.object = NULL)
```

**Arguments**

- `dependency.dir`: Directory with dependency information files
- `dependency.object`: Data frame with dependency information

**Details**

Not for direct use. Only needs one or the other argument.

**Value**

list of information about file hash mismatches

**Examples**

```r
## Not run:
checkFileHash(pullSourceInfo("adaprHome")$dependency.dir)

## End(Not run)
```

---

checkFileHashSource

**Description**

Checks the consistency of the dependency directory with the files within the file system. Reports the source scripts that need to be updated.
**Usage**

`checkFileHashSource(dependency.dir = NULL, dependency.object = NULL)`

**Arguments**

- `dependency.dir` Directory with dependency information files
- `dependency.object` data frame with dependency information

**Details**

Only needs one or the other argument.

**Value**

List of information about file hash mismatches

**Examples**

```r
## Not run:

checkFileHashSource(pullSourceInfo("adaprHome")$dependency.dir)

## End(Not run)
```

**Description**

Checks the consistency of the dependency directory with the files within the file system Reports the source scripts that need to be updated!

**Usage**

`checkFileMtimeSource(dependency.dir = NULL, dependency.object = NULL)`

**Arguments**

- `dependency.dir` Directory with dependency information files
- `dependency.object` data frame with dependency information
checkRmdMode

Details

Only needs one or the other argument.

Value

list of information about file hash mismatches

Examples

## Not run:
checkFileMtimeSource(pullSourceInfo("adaprHome")$dependency.dir)

## End(Not run)

checkRmdMode

Checks whether interactive R markdown session is ON

Description

Checks whether interactive R markdown session is ON

Usage

checkRmdMode(changeOption = FALSE)

Arguments

changeOption logical to print out and change the Rmdstart option

Value

value TRUE if R session in in interactive R markdown mode

Examples

## Not run:
checkRmdMode()

## End(Not run)
### checkVersion

*Check install of package of specific version*

**Description**

Check install of package of specific version

**Usage**

```r
checkVersion(package0, version0 = "", versionCheck = FALSE, 
              lib = .libPaths()[1])
```

**Arguments**

- `package0`: name of package
- `version0`: package version
- `versionCheck`: logical to install specific version
- `lib`: path to local library

**Value**

logical on installed status

### commit2char

*git2r commit class to character converter*

**Description**

git2r commit class to character converter

**Usage**

```r
commit2char(commitclass)
```

**Arguments**

- `commitclass`: Commit object

**Details**

Uses git2r package.

**Value**

commit message
commitProject

Examples

## Not run:
committed <- git2r::commit(repo.message = commit.message)
out <- paste("Git", commit2char(committed))

## End(Not run)

commitProject  
*Git commit of project.*

Description

Git commit of project.

Usage

commitProject(commit.message = "", project.id = getProject(),
               addAll = FALSE, checkSync = TRUE)

Arguments

- commit.message: message describing edits
- project.id: project to commit
- addAll: logical for adding every R script and support file to commit
- checkSync: logical for checking sync status

Details

Need git option active. Uses git2r package. addAll = TRUE will increase compute time.

Value

commit message

Examples

## Not run:
commitProject("adaprHome","Did I change something?")

## End(Not run)
condenseFileInfo  Generate condensed information about files from dependency object

Description
Generate condensed information about files from dependency object

Usage
condenseFileInfo(trees)

Arguments
trees  dependency object

Value
data frame with information about each file in project

Examples
## Not run:
trees <- readDependency(pullSourceInfo("adaptHome"))$dependency.dir
condenseFileInfo(trees)
## End(Not run)

createFileInfo  Creates a list containing information about a file

Description
Creates a list containing information about a file

Usage
createFileInfo(path, file, description = "", dbname = "", si = NULL)

Arguments
path  is the directory file resides
file  is the filename
description  is the description of the file
dbname  is the database name
si  is the source_info list default is null
createMarkdown

Value

file info list with path filename fullname description and db.name

Examples

```r
## Not run:
file0 <- file.path(pullSourceInfo("adaprHome")$project.path,
project.directory.tree$analysis,"read_data.R")
createFileInfo(dirname(file0),basename(file0),"a program that reads")

## End(Not run)
```

createMarkdown

*Creates an Rmarkdown file in specified directory*

Description

Creates an Rmarkdown file in specified directory

Usage

```r
createMarkdown(target.file = paste0(getSourceInfo()$file$file, "md"),
target.dir = getSourceInfo()$markdown.dir, style = "html_document",
description = "Markdown", si, overwrite = FALSE)
```

Arguments

- `target.file` Markdown file to create
- `target.dir` Directory to send target file
- `style` Markdown target style
- `description` Markdown description
- `si` source_info object for tracking
- `overwrite` overwrite existing R markdown?

Details

Uses rmarkdown library to access objects in the R script. Will track dependencies for objects used within Rmd file.

Value

File information
createProgramGraph

Make plot of project programs only Summarize all programs. Sync status is assessed and indicated.

Description

Make plot of project programs only Summarize all programs. Sync status is assessed and indicated.

Usage

createProgramGraph(project.id, testSync = TRUE)

Arguments

project.id  Project id of program
testSync  Logical to test synchronization status

Details

Uses ggplot2

Value

List of data.frame of programs vertices, data.frame of edges, ggplot, rgrapher=igraph

Examples

## Not run:
createProgramGraph("adaprHome")

## End(Not run)
create_source_file_dir

Create source file directories

Description

Create source file directories

Usage

create_source_file_dir(project.id0 = get("project.id"),
                      source.file0 = get("source.file"), source.description = "")

Arguments

project.id0       project id name string
source.file0     filename of the source
source.description    character description of what the source file does

Details

Initializes git for the project, adds program git tracking, creates project library and initializes dependency tracking. Creates directories for Project and Results. Initialize the file tracking object source_info. Gathers file information on all project files. Initialize Git for project (if using Git). Adds dependencies files to Git. Initialize archivist repo for script. Run all R scripts in support_function directory. Run all R scripts in script specific support_function/myRscript.R directory. Create R markdown file with same file prefix (if not already done). Create publication file (if not already done).

Value

source_info list describing the project

Examples

## Not run:
source_info <- create_source_file_dir("adaprHome","tree_controller.R")

## End(Not run)
dataDir | Returns project’s data directory, allows relative directories. Used within an R script.

Description
Returns project’s data directory, allows relative directories. Used within an R script.

Usage
dataDir(project.id = getProject())

Arguments
project.id | project specifies which data directory

Value
path to data directory

Examples
## Not run:
dataDir(getProject())
## End(Not run)

defaultAdaprSetup | Set up adapr 1st time

Description
Set up adapr 1st time

Usage
defaultAdaprSetup()

Details
Use on ADAPR start up. Requires RStudio to work. Will make project directories in computer Document directory and create adaprHome project.
Examples

```
## Not run:
# Requires pandoc location or RStudio
default.adapr.setup()

## End(Not run)
```

---

**dependency-class**

**Dependency class**

**Description**

Dependency class

**Methods**

update(df.update) Updates the dependency object with a read in or write out

---

**Digest**

**Digest files from (digest package)**

**Description**

Compute file hash without checking file.access

**Usage**

```
Digest(object = NULL, algo = checkAdaprHashAlgo(), serialize = FALSE, 
file = TRUE, length = Inf, skip = "auto", ascii = FALSE, 
raw = FALSE)
```

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>R object or file</td>
</tr>
<tr>
<td>algo</td>
<td>digest hash algorithm</td>
</tr>
<tr>
<td>serialize</td>
<td>FALSE allows hash comparison of known output</td>
</tr>
<tr>
<td>file</td>
<td>logical TRUE iff object is file</td>
</tr>
<tr>
<td>length</td>
<td>Size of object/file to hash</td>
</tr>
<tr>
<td>skip</td>
<td>How many input bytes to skip for computing hash</td>
</tr>
<tr>
<td>ascii</td>
<td>ASCII or binary compression</td>
</tr>
<tr>
<td>raw</td>
<td>logical digest output in binary form</td>
</tr>
</tbody>
</table>
Details

Uses digest from package "digest". Authors Dirk Eddelbuettel edd@debian.org for the R interface; Antoine Lucas for the integration of crc32; Jarek Tuszyński for the file-based operations; Henrik Bengtsson and Simon Urbanek for improved serialization patches; Christophe Devine for the hash function implementations for sha-1, sha-256 and md5; Jean-loup Gailly and Mark Adler for crc32; Hannes Muehleisen for the integration of sha-512; Jim Hester for the integration of xxhash32, xxhash64 and murmur32.

Value

The filehash

Examples

```r
## Not run:
fileP <- file.path(pullSourceInfo("adapRHome")$project.path,
project.directory.tree$analysis,"read_data.R")
Digest(file=fileP)

## End(Not run)
```

```r

<table>
<thead>
<tr>
<th>fileInfoProjects</th>
<th>List project file information disk space, modification timespan, days inactive</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>inactive</td>
</tr>
</tbody>
</table>
```

Description

List project file information disk space, modification timespan, days inactive

Usage

```r
fileInfoProjects(project.id = listProjects()$project.id)
```

Arguments

- `project.id` character vector of projects

Value

dataframe with project information

Examples

```r
## Not run:
fileInfoProjects()

## End(Not run)
```
### finalize_dependency

**Description**

Writes dependency data to file in "Dependency" directory

**Usage**

```r
finalize_dependency(RMD = TRUE, write = TRUE)
```

**Arguments**

- `RMD` Logical denoting whether the finalizing occurs in markdown vs R script
- `write` Logical indicated to write the dependency object

**Details**

Operates git tracking of program and dependency file.

Strips project directory out of dependency file. Passes workspace to R markdown file and render. Stores file modification and hash for all dependencies. Writes dependency information to 'Dependency' directory. Adds dependency file and Session information to Git.

**Value**

dependency.object

**Examples**

```r
## Not run:
#Executed only at the end of adapr R script
#finalize_dependency()

## End(Not run)
```

### firstProject

**Description**

Create first project

**Usage**

```r
firstProject(project.path = "", publish.path = "")
```
getAdapOptions

Arguments

    project.path  Path where first project will go
    publish.path  Path to share project results

Value

    logical for successful creation or not

Examples

    ## Not run:
    opt <- getAdapOptions()
    firstProject(opt$project.path,opt$publish.path)

    ## End(Not run)

getAdapOptions  Returns the primary hub file with project location and id information

Description

    Returns the primary hub file with project location and id information

Usage

    getAdapOptions(setoptions = FALSE)

Arguments

    setoptions  Logical specifying Execute Options

Value

    adaproptions

Examples

    ## Not run:
    opt <- getAdapOptions()
    print(opt)

    ## End(Not run)
**getDepSubgraph**  
*Produces subgraph of dependencies of R script*

---

**Description**

Produces subgraph of dependencies of R script

**Usage**

```r
getDepSubgraph(rscript, project.id = getProject(), plotTF = FALSE)
```

**Arguments**

- `rscript`: R script name
- `project.id`: Project graph
- `plotTF`: logical to plot or not

**Details**

Only needs one or the other argument.

**Value**

List with subgraph in igraph format, data frame format, and layout for plotting.

**Examples**

```r
# Not run:
subGraph <- scriptSubgraph(project.id=getProject())
plot(subGraph[[1]], vertex.label=basename(igraph::V(subGraph[[1]])$name), layout=subGraph[[2]])
# End(Not run)
```

---

**getFileInfo**  
*Retrieve the file info for the file by name OR from the data subdirectory data*

---

**Description**

Retrieve the file info for the file by name OR from the data subdirectory data

**Usage**

```r
getFileInfo(source_info, data = "", file0 = "", path.grep = "")
```
Arguments

source_info  Project information in list
data         Data file name
file0        Filename not in data
path.grep    Path to search for file in

Value

File information list outcome of search

Examples

```r
## Not run:
source_info <- create_source_file_dir("adaprHome","tree_controller.R")
getFileInfo(source_info,file0="read_data.R")

## End(Not run)
```

getFileSysTime  

Retrun time of file system

Description

Writes to temporary file and extracts mod time with file.info

Usage

getFileSysTime(directory = "")

Arguments

directory    path to directory

Value

The file system write time

Examples

```r
## Not run:
getFileSysTime()

## End(Not run)
```
getLibrary

Get library for a project

Description
Get library for a project

Usage
getLibrary(project.id = getProject())

Arguments
project.id character vector of project

Value
dataframe of libraries

Examples
## Not run:
getLibrary("adaprHome")

## End(Not run)

getProject

Returns the adapr project in R option "adaprProject"

Description
Returns the adapr project in R option "adaprProject"

Usage
getProject()

Details
Default is adaprHome. Returns default if project does not exist.

Value
Value is specified project or default project
Examples

```r
## Not run:
getProject()
## End(Not run)
```

---

**getProjectInfo**

*Acquire all dependencies related to a project, Generate graph of project dependencies, Get all file information related to project*

---

**Description**

Acquire all dependencies related to a project, Generate graph of project dependencies, Get all file information related to project

**Usage**

```r
getProjectInfo(dependency.dir)
```

**Arguments**

- `dependency.dir` is the string location of dependency files

**Value**

list with stacked dependency files, graph of dependencies, and condensed file information

**Examples**

```r
## Not run:
projInfo <- getProjectInfo(pullSourceInfo("adaprHome")$dependency.dir)
plot(projInfo$graph)
## End(Not run)
```
**getProjectInfoSI**

**Given source_info object, retrieves project information**

**Description**
Given source_info object, retrieves project information

**Usage**

`getProjectInfoSI(source_info)`

**Arguments**

- **source_info** is list with source information

**Value**

list with stacked dependency files, graph of dependencies, and condensed file information

**Examples**

```r
## Not run:
source_info <- create_source_file_dir("adaprHome","tree_controller.R")
getProjectInfoSI(source_info)

## End(Not run)
```

**getProjectLibrary**

**Given Project name, Return project library directory**

**Description**
Given Project name, Return project library directory

**Usage**

`getProjectLibrary(project.id0 = getProject())`

**Arguments**

- **project.id0** is string with project name

**Details**

- Reads "~/ProjectPaths/projectid_2_directory.csv" into dataframe
- Will create directory if it doesn’t already exist.
getProjectPath

Value

string containing project library directory. Will return empty string if default library.

Examples

```
## Not run:
getProjectLibrary("adaprHome")

## End(Not run)
```

getProjectPath  Given Project id, Return project directory

Description

Given Project id, Return project directory

Usage

```r
getProjectPath(project.id0 = getProject())
```

Arguments

project.id0 is string with project name

Details

Reads "/~ProjectPaths/projectid_2_directory.csv" into dataframe

Value

string containing project directory

Examples

```
## Not run:
getProjectPath("adaprHome")

## End(Not run)
```
### getProjectPublishPath

*Given Project name, Return project publish directory*

**Description**

Given Project name, Return project publish directory

**Usage**

```r
getProjectPublishPath(project_name = NULL)
```

**Arguments**

- `project_name` is string with project name

**Details**

Reads "~/ProjectPaths/projectid_2_directory.csv" into dataframe

**Value**

String containing project directory

**Examples**

```r
## Not run:
getProjectPublishPath("adaprHome")

## End(Not run)
```

### getProjectSwapPath

*Retrieve project swap directory*

**Description**

Retrieve project swap directory

**Usage**

```r
getProjectSwapPath(project_name = NULL)
```

**Arguments**

- `project_name` Project ID
getPubResults
       Read result filepaths to publish

Details
Do not use. Deprecated.

Value
file path for swap directory

Arguments
project.id Project to publish

Details
File is in support directory/files_to_publish.csv

Value
dataframe of files to publish

Examples
## Not run:
getPubResults("adaprHome")

## End(Not run)
**getSourceInfo**

*Returns the information related to the adapr script*

---

**Description**

Returns the information related to the adapr script

**Usage**

```r
getSourceInfo()
```

**Value**

list with information about the project

**Examples**

```r
## Not run:
getSourceInfo()

## End(Not run)
```

---

**getUpstream**

*Identifies dependencies in a DAG*

---

**Description**

Identifies dependencies in a DAG

**Usage**

```r
getUpstream(endVertex, isg)
```

**Arguments**

- `endVertex` vertex
- `isg` Project graph

**Details**

Lower level function. Uses recursion, may contain non-unique vertices.

**Value**

list with subgraph in igraph format, data frame format, and layout for plottingss
get_orchard

*Returns the primary hub file with project location and id information*

**Description**

Returns the primary hub file with project location and id information.

**Usage**

```
get_orchard()
```

**Details**

Not for direct use. See listProjects for direct use.

**Value**

```
orchard
```

**Examples**

```r
## Not run:
orchard <- get_orchard()
print(subset(orchard, project.id == "adaprHome"))

## End(Not run)
```

---

**gitAdd**

*git add to stage the file*

**Description**

git add to stage the file.

**Usage**

```
gitAdd(gitdir, filename, branch = NULL, git_args = character(),
git_binary = NULL)
```

**Arguments**

- **gitdir**: string with git directory
- **filename**: string of file to query
- **branch**: git branch
- **git_args**: string argument for git
- **git_binary**: location of git executable
**Value**

git output from git add

**Author(s)**

Uses git2r package.

**Examples**

```r
## Not run:
si <- pullSourceInfo("adaprHome")
file0 <- file.path(si$project.path, project.directory.tree$analysis,"read_data.R")
gitAdd(si$project.path,file0)
## End(Not run)
```

---

**Description**

git commit. Requires git installation.

**Usage**

```r
gitCommit(gitdir, message, branch = NULL, git_args = character(), 
git_binary = NULL)
```

**Arguments**

- **gitdir**: string with git directory
- **message**: for commit
- **branch**: git branch
- **git_args**: arguments to git
- **git_binary**: location of git executable

**Value**

Not for direct use. Git commit output.

**Examples**

```r
## Not run:
si <- pullSourceInfo("adaprHome")
gitCommit(si$project.path,"test commit")
## End(Not run)
```
gitConfigure  

Configure user.name and email for git.

Description

Configure user.name and email for git.

Usage

gitConfigure(user.name, user.email)

Arguments

user.name  Chris Someone
user.email  someone[at]somewhere.com

Value

output from git

Examples

## Not run:
# Uses git2r package
  gitConfigure("jonG","gelfond@somewhere.com")

## End(Not run)

gitConfigureTest  

Checks git configuration. Requires git installation

Description

Checks git configuration. Requires git installation

Usage

gitConfigureTest(globalTF = TRUE)

Arguments

globalTF  logical specifying global git configuration

Value

output from git2r::config
gitHistorySearch

Author(s)
Uses git2r::config in git2r package.

Examples

## Not run:
#Requires git installation
gitConfigureTest()

## End(Not run)

---

**gitHistorySearch**

Performs git history search

**Description**

Performes git history search

**Usage**

```r
gitHistorySearch(gitdir, pattern, branch = NULL, git_args = character(),
                git_binary = NULL)
```

**Arguments**

- **gitdir**: string with git directory
- **pattern**: to match in history
- **branch**: git branch
- **git_args**: arguments to git
- **git_binary**: location of git executable

**Value**

git log output

**Author(s)**

Uses git_path adapted form devtools author Hadley Wickham

**Examples**

```r
## Not run:
si <- pullSourceInfo("adapRHome")
gitHistorySearch(si$project.path,"read_data.R")

## End(Not run)
```
**gitIgnoreLibrary**  
*Git ignore the library file*

**Description**

Git ignore the library file

**Usage**

```java
gitIgnoreLibrary(project.id = getProject())
```

**Arguments**

- `project.id` to ignore Project’s library

**Details**

Libraries can be large and difficult to track with Git so we can ignore these.

**Value**

- `success`

---

**gitInfo**  
*Retrieves the information from git about a file*

**Description**

Retrieves the information from git about a file

**Usage**

```java
gitInfo(gitdir, filename, branch = NULL, git_args = character(), git_binary = NULL)
```

**Arguments**

- `gitdir` string with git directory
- `filename` string of file to query
- `branch` git branch
- `git_args` string argument for git
- `git_binary` location of git executable
**git Init**

**Value**

git log for filename

**Examples**

```r
## Not run:
si <- pullSourceInfo("adaprHome")
file0 <- file.path(s$project.path,project.directory.tree$analysis,"read_data.R")
gitInfo(s$project.path,file0)

## End(Not run)
```

---

**gitInit**

*Initiate git*

**Description**

Initiate git

**Usage**

```r
gitInit(gitdir, branch = NULL, git_binary = NULL)
```

**Arguments**

- `gitdir`: string with git directory
- `branch`: git branch
- `git_binary`: location of git executable

**Details**

* Deprecated.

**Value**

git init lockaout

**Examples**

```r
## Not run:
gitInit(getProjectPath("adaprHome"))

## End(Not run)
```
gitProvenance  

**Description**

Identify git provenance of file within a project

**Usage**

```r
gitProvenance(project.id, filepath = 0)
```

**Arguments**

- `project.id`: Project id to search for history within
- `filepath`: File that will be hashed and search within Git history. File choose dialogue if not specified

**Details**

Requires a Git commit snapshot within the project

**Value**

List of 1) filename, 2) Git commit including commit message, date, author and 2) file info

**Examples**

```r
## Not run:
projpath <- file.path(getProjectPath("adaprHome"),"Programs")
gitProvenance("adaprHome",file.path(projpath,"read_data.R"))

## End(Not run)
```

---

**git_path**

**Description**

Find path for git executable

**Usage**

```r
git_path(git_binary_name = NULL)
```
Graph

Arguments

- **git_binary_name**: git binary name

Value

- **git executable**: git executable

Author(s)

Adapted form devtools author Hadley Wickham

---

**Graph**

*Write object and capture file information*

---

Description

Write object and capture file information

Usage

```r
Graph(file.name = "data.csv", description = "Result file",
      write.fcn = guessWriteFcn(file.name), date = FALSE, ...)
```

Arguments

- **file.name**: file to write to the source "Result" directory
- **description**: describes object to write
- **write.fcn**: function for writing file of object type. Will open graphics device.
- **date**: logical for adding date to filename
- **...**: arguments passed to write.fcn

Value

File information list

Examples

```r
## Not run:
# Within adappr R Script body:
Graph("hist.pdf","Gaussian RV")
hist(rnorm(100))
dev.off()

## End(Not run)
```
graphProject  
Make plot of project programs only Summarize all programs. Sync status is assessed and indicated.

Description

Make plot of project programs only Summarize all programs. Sync status is assessed and indicated.

Usage

graphProject(project = getProject(), testSync = TRUE)

Arguments

- project: Project id of program
- testSync: logical to test synchronization status

Details

Uses ggplot2. Is a wrapper for create_program_graph.

Value

List of data.frame of programs vertices, data.frame of edges, ggplot , rgrapher=igraph

Examples

## Not run:
graphProject("adaprHome")

## End(Not run)

guess.read.fcn  
Return function for reading common file types

Description

Return function for reading common file types

Usage

guess.read.fcn(filename)

Arguments

- filename: for file to be read
guessWriteFcn

Details

Uses the file suffix to return csv, read.delim, read.xls

Value

function for reading

Examples

```r
## Not run:
identical(utils::read.csv, guess.read.fcn("data.csv"))

## End(Not run)
```

guessWriteFcn

Return function for writing common file types

Description

Return function for writing common file types

Usage

```
guessWriteFcn(filename)
```

Arguments

filename file to be written

Details

Uses the file suffix to return write.csv, png, pdf, save, saveRDS

Value

function for writing file

Examples

```r
## Not run:
identical(utils::write.csv, guessWriteFcn("data.csv"))

## End(Not run)
```
idPackages

Lower level function that captures packages that are explicitly loaded, not loaded automatically

Description

Lower level function that captures packages that are explicitly loaded, not loaded automatically

Usage

idPackages(library.data.file)

Arguments

library.data.file

CSV File with a set of library names and repository locations

Details

Captures unaccounted for library within library information file. Not for direct use.

Value

Libraries loaded that were not automatically loaded

idSync

Returns the source files needed to repair synchrony

Description

Returns the source files needed to repair synchrony

Usage

idSync(file.info, dag.to.sync)

Arguments

file.info

Project file information

dag.to.sync

Directed Acyclic graph in need of synchronization

Value

data.frame with sources that need to be run, rows in run order
importData

Examples

```r
## Not run:
projInfo <- getProjectInfo(pullSourceInfo("adaprHome")$dependency.dir)
trees <- readDependency(pullSourceInfo("adaprHome")$dependency.dir)
file.info <- condenseFileInfo(trees)
idsync(file.info, projInfo$graph)

## End(Not run)
```

---

**importData**

*Import selected file into the project data directory*

**Description**

Import selected file into the project data directory

**Usage**

`importData(datafile = "", project.id = getProject(), overwriteTF = TRUE)`

**Arguments**

- `datafile` filename can be unspecified
- `project.id` string indicating which project data directory to copy into
- `overwriteTF` logical indicating whether to overwrite exist file.

**Details**

If `datafile` is "" then a file choose dialogue is created

**Examples**

```r
## Not run:
# Will open file browser to copy into adaprHome data directory
importData(project.id="adaprHome")

## End(Not run)
```
**initialize_dependency_info**

*Initializes dependency object source_info*

**Description**

Initializes dependency object source_info

**Usage**

`initialize_dependency_info(source_info_arg)`

**Arguments**

- `source_info_arg`
  
is a source_info list with describing R script and project

**Details**

Not for direct use.

**Value**

Dependency file location

**initProject**

*initialize project*

**Description**

initialize project

**Usage**

`initProject(project.id, project.path = NA, publish.directory = NA, first.program = "read_data.R", project.libraryTF = FALSE, library.path = ")`
install

Arguments

- `project.id`: Project name, if missing then default
- `project.path`: Project home directory, if missing then default
- `publish.directory`: Project branch exchange directory
- `first.program`: Name of first program in project (read_data.R default)
- `project.libraryTF`: character string "packrat","TRUE","FALSE" for using packrat package or Logical to use a local (not default) library
- `library.path`: Path to local (not default) library

Details

Sets up project for first time. Defaults to main library. If using a local library, then leaving library path equal to "" puts the library within the project folder.

Wrapper for plantTree

Value

logical for success or not

Examples

```r
## Not run:
initProject("adaprTest")

## End(Not run)
```

install

Install package of specific version

Description

Install package of specific version

Usage

```r
install(package, version = NULL, installVersion = FALSE,
       lib = .libPaths()[1], repos =getOption("repos"),
       show.available = FALSE, packageSource = "", ...)
```
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>package</td>
<td>package name to install</td>
</tr>
<tr>
<td>version</td>
<td>package version</td>
</tr>
<tr>
<td>installVersion</td>
<td>logical, TRUE for install a specific version, if FALSE then latest</td>
</tr>
<tr>
<td>lib</td>
<td>path to local library</td>
</tr>
<tr>
<td>repos</td>
<td>character of repository</td>
</tr>
<tr>
<td>show.available</td>
<td>logical to display whether the package is available</td>
</tr>
<tr>
<td>packageSource</td>
<td>character describing where the package is from.</td>
</tr>
<tr>
<td>...</td>
<td>Argument to install.packages/install.version functions</td>
</tr>
</tbody>
</table>

Details

Installs from CRAN and bioconductor packages. Local libraries will not be installed.

Value

Library information data

---
**installLibrary**

*Install package of specific version*

Description

Install package of specific version

Usage

```
installLibrary(input = getLibrary(), lib = getProjectLibrary(),
               versionCheck = FALSE)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td>data.frame with 3 columns package version repos to install</td>
</tr>
<tr>
<td>lib</td>
<td>path to local library</td>
</tr>
<tr>
<td>versionCheck</td>
<td>logical to install specific version</td>
</tr>
</tbody>
</table>

Details

Calls adapr::install and installs from CRAN and bioconductor packages. Local packages will not be installed.

Value

Library information data
installProjectPackages

Installs all packages

Description
Installs all packages

Usage
installProjectPackages(project.id = getProject())

Arguments
project.id project.id to install packages for R package to

Details
Not for direct use. Installs autoloaded packages.

Value
Library information data

Examples
## Not run:
installProjectPackages("adaprHome")

## End(Not run)
listBranches

Description
Installs and loads library specific to a project

Usage
Library(package, repository = "cran", github = "", project.id = getProject())

Arguments
- package: character for package to load/install
- repository: character for location of repository. "cran" for CRAN and "bioc" for bioconductor.
- github: character for devtools::install_github("xxx")
- project.id: project id project id install within

Details
Use within program Body

Examples
## Not run:
Library("adapr","cran",project.id="adaprHome")

## End(Not run)

listBranches

Description
Lists the branches available for loading in the adapr project

Usage
listBranches(project.id = getProject())

Arguments
- project.id: project to find branches within
Value

dataframe of descriptions available branches

Examples

```r
## Not run:
listBranches("adaprHome")

## End(Not run)
```

---

**listDatafiles**

*Lists the data files available for reading in the adapr project*

Description

Lists the data files available for reading in the adapr project

Usage

```r
listDatafiles(project.id = getProject())
```

Arguments

- **project.id**: Project to look for data files within

Value

description of data files

Examples

```r
## Not run:
listDatafiles("adaprHome")

## End(Not run)
```
listProjects  \textit{List projects}

\textbf{Description}

List projects

\textbf{Usage}

\begin{verbatim}
listProjects(project.id = "", allInfo = TRUE)
\end{verbatim}

\textbf{Arguments}

\begin{itemize}
  \item project.id: character for specific project. Empty string default will list all projects.
  \item allInfo: logical whether to return all data
\end{itemize}

\textbf{Value}

data frame with project information

\textbf{Examples}

\begin{verbatim}
## Not run:
listProjects(TRUE)

## End(Not run)
\end{verbatim}

listScripts  \textit{Lists the R scripts in the adapr project}

\textbf{Description}

Lists the R scripts in the adapr project

\textbf{Usage}

\begin{verbatim}
listScripts(project.id = getProject())
\end{verbatim}

\textbf{Arguments}

\begin{itemize}
  \item project.id: project.id
\end{itemize}

\textbf{Value}

dataframe of R scripts and descriptions
Load.branch

**Examples**

```r
## Not run:
listScripts("adaprHome")

## End(Not run)
```

---

**Load.branch**

*Loads a single R object from file*

**Description**

Loads a single R object from file

**Usage**

```r
loadBranch(file)
```

**Arguments**

- `file` contains R object

**Value**

object for file that was read

**Examples**

```r
## Not run:
processed <- loadBranch("read_data.R/process_data.Rdata")

## End(Not run)
```

---

loadAdaprTest

*Create adaprTest example project*

**Description**

Create adaprTest example project

**Usage**

```r
loadAdaprTest(localLibraryTF = FALSE, overwrite = TRUE)
```
Arguments

localLibraryTF Logical for local library or not
overwrite Logical indicating whether to overwrite existing project

Details

To be run after default adapr set up.

Examples

## Not run:
#
loadAdaprTest()

## End(Not run)

loadFlex  Loads a single R object from file, more flexible than Load.branch or base::load

Description

Loads a single R object from file, more flexible than Load.branch or base::load

Usage

loadFlex(file, read.fcn = readRDS, ...)

Arguments

file contains R object
read.fcn function to read the file, default readRDS
... arguments passed to read.fcn

Value

object for file that was read

Examples

## Not run:
processed <- load.flex("read_data.R/process_data.RData")

## End(Not run)
loadInstallLibraryFile

*This function is no longer supported. Loads libraries within the file* library.list.file

**Description**

This function is no longer supported. Loads libraries within the file library.list.file

**Usage**

loadInstallLibraryFile(library.data.file = NA, subgroup = NULL, verbose = FALSE, install.all = FALSE)

**Arguments**

- **library.data.file**
  - CSV File with a set of library names and repository locations
- **subgroup**
  - data frame with Package, repos, and specific columns
- **verbose**
  - Print which libraries are installed and loaded
- **install.all**
  - logical indicated whether to install all project packages

**Details**

Installs and loads all packages. Not for direct use. See installProjectPackages().

**Value**

Library information data

-----

makeDependencyGraphObj

*Creates an graph object from a dependency object*

**Description**

Creates an graph object from a dependency object

**Usage**

makeDependencyGraphObj(dependency.out)

**Arguments**

- **dependency.out**
  - Dependency object(s) to make graph out of
makeFunction

Value

graph object of project/program dependencies

Examples

```r
## Not run:
trees <- readDependency(pullSourceInfo("adaprHome")$dependency.dir)
dag<-makeDependencyGraphObj(trees)

## End(Not run)
```

makeFunction  Generates the shell of a R function that is project specific in support_functions folder

Description

Generates the shell of a R function that is project specific in support_functions folder

Usage

```r
makeFunction(functionName = NA, description = "", project.id = getProject())
```

Arguments

- `functionName`: character string for R function.
- `description`: character string description of function
- `project.id`: Name of project

Details

Function file will add ".R" extension. Will not overwrite existing program. See makeScript() for making an R script.

Value

Logical indicating success or not
**makeHyperlink**

*Makes HTML hyper link*

**Description**

Makes HTML hyper link

**Usage**

makeHyperlink(files, links)

**Arguments**

- files: character vector of filenames
- links: description of links

**Details**

Used in making HTML files

**Value**

link command vector

**Examples**

```r
## Not run:
makeHyperlink("myPath","click here to my path")
## End(Not run)
```

**makeScript**

*Generates the shell of a code that is project specific*

**Description**

Generates the shell of a code that is project specific

**Usage**

makeScript(r = "", description = "", project.id = getProject(),
seed = 2011, run = TRUE, openTF = TRUE)
makeSummaryGraph

Arguments

- **r**: is source file name or Filename to create
- **description**: Character string describing what program does
- **project.id**: Character string for name of project
- **seed**: Random start seed
- **run**: Logical for execution of r script
- **openTF**: Logical for opening R script

Details

Will not overwrite existing program. Executes program and opens stub program. Mostly wrapper for sproutProgram.

Value

Logical indicating failure or not

Examples

```r
## Not run:
makeScript("read.data.R", description="reads data", "adaprHome")

## End(Not run)
```

---

**makeSummaryGraph**

Make.summary graph of projects based on files in dependency directory

Description

Make.summary graph of projects based on files in dependency directory

Usage

```r
makeSummaryGraph(dependency.dir = NULL, dependency.object = NULL,
plot.graph = FALSE)
```

Arguments

- **dependency.dir**: Dependency directory
- **dependency.object**: Dependency.data
- **plot.graph**: Logical to plot graph or not
monitorParallelSync

Details

Experimental (use with caution) track parallelSync while in progress

Usage

monitorParallelSync(project.id = getProject(), check.interval = 5)

Arguments

project.id Project to synchronize.
check.interval how many seconds to delay until last check

Details

Must use separate R process from parallelSync(). Refreshes project plot with compute node labels are working or completed

Value

ggplot of project graph

Examples

## Not run:
monitorParallelSync("adaprHome")

## End(Not run)
openProjectList  

_Browses orchard in file system_

Description

Browses orchard in file system

Usage

openProjectList()

Value

orchard

Examples

```r
## Not run:
openProjectList()
## End(Not run)
```

openScript  

_Insert script from a project with default R program. Can open markdown files as well._

Description

Opens script from a project with default R program. Can open markdown files as well.

Usage

openScript(rscript = "", project.id = getProject())

Arguments

- `rscript` : R script or Markdown filename to open
- `project.id` : string for project id to search within.

Details

If `rscript` argument is blank, then lists available scripts for convenience.
parallelSync

Examples

```r
## Not run:
# Opens read_data.R within the adaprHome project
openScript("read_data.R","adaprHome")

## End(Not run)
```

---

**parallelSync**

*Experimental* (use with caution) parallel synchronization of project. Takes advantage of directed acyclic graph structure to run R script processes in parallel.

### Description

Experimental (use with caution) parallel synchronization of project. Takes advantage of directed acyclic graph structure to run R script processes in parallel.

### Usage

```r
parallelSync(project.id = getProject(), n.cores = 2)
```

### Arguments

- **project.id**: Project to synchronize.
- **n.cores**: Number of cores to use. Should be >1, but less than number of logical CPUs.

### Details

Experimental. See also monitorParallelSync(), syncProject() and syncTestProject(). Uses Results/tree_controller.R directory to pass work/completion data between nodes.

### Value

data.frame with success/failure status.

### Examples

```r
## Not run:
parallelSyncProject("adaprHome")

## End(Not run)
```
path.expand.2  

Location of options file and project listing. Swap / for \ in path expand.

Description

Location of options file and project listing. Swap / for \ in path expand.

Usage

path.expand.2(x)

Arguments

x  
file path, could be relative or ~

Details

See path.expand() in base R. Can be controlled in R profile by adding adaprHomeDir option. See adaprHomeDir().

Value

Full filepath to x

Examples

```r
## Not run:
path.expand.2("~")

## End(Not run)
```

plantOrchard  
Create project hub files in root directory

Description

Create project hub files in root directory

Usage

plantOrchard()

Value

logical for succesful creation or not
Examples

```r
## Not run:
plantOrchard()

## End(Not run)
```

---

**plantTree**

*initialize project*

**Description**

initialize project

**Usage**

```r
plantTree(project.id, project.path = NA, swap.directory = NA, 
           first.program = "read_data.R", project.libraryTF = FALSE, 
           library.path = ")
```

**Arguments**

- `project.id`  Project name, if missing then default
- `project.path`  Project home directory, if missing then default
- `swap.directory`  Project branch exchange directory
- `first.program`  Name of first program in project (read_data.R default)
- `project.libraryTF`  Logical to use a local (not default) library
- `library.path`  = path to local (not default) library

**Details**

Not for direct use. See `initProject()`.

**Value**

logical for success or not
**Description**

Create program io table

**Usage**

```
programIOTable(dependency.out)
```

**Arguments**

`dependency.out`  Tree of dependencies

**Details**

groups inputs and outputs
Not for direct use.

**Value**

Matrix summarizing inputs and outputs

**Examples**

```r
## Not run:
trees <- readDependency(pullSourceInfo("adaprHome")$dependency.dir)
programIOTable(trees)
## End(Not run)
```

---

**Description**

Project directory tree structure contains the relative directory structure included analysis, data, texidr, dependency.dir, support functions, and library bank locations

**Usage**

```
project.directory.tree
```
**Format**

An object of class list of length 7.

---

**projectReportMarkdown** *Make plot of network within html documents. Summarize all programs.*

---

**Description**

Make plot of network within html documents. Summarize all programs.

**Usage**

```r
projectReportMarkdown(source_info, graph.width = 960, graph.height = 500)
```

**Arguments**

- `source_info` Source information list
- `graph.width` Sankey Plot dimensions
- `graph.height` Sankey Plot dimensions

**Details**

Dose not assume source_info in workspace

**Value**

output file

**Examples**

```r
## Not run:
source_info <- create_source_file_dir("adaprHome", "tree_controller.R")
projectReportMarkdown(source_info)

## End(Not run)
```
projectReportSend  

Make plot of network within html documents. Summarize all programs. Make a readme file at top project directory copy to target.directory Uses pander and pandoc unlike project_report

Description

Make plot of network within html documents. Summarize all programs. Make a readme file at top project directory copy to target.directory Uses pander and pandoc unlike project_report

Usage

projectReportSend(target.directory = get("source_info")$project.path, si, send.data = FALSE, graph.width = 960, graph.height = 500)

Arguments

target.directory  Location to send project report
si  Source information list
send.data  Logical to send data directory or not
graph.width  Sankey Plot dimensions
graph.height  Sankey Plot dimensions

Details

Not for direct use. Does not assume source_info in workspace

Examples

## Not run:
source_info <- create_source_file_dir("adaprHome","tree_controller.R")
projectReportSend(si=source_info)

## End(Not run)
**publishResults**

Read in results to publish & Copies results to the project’s publication directory

**Usage**

```
publishResults(project.id = getProject())
```

**Arguments**

- **project.id** Project to publish

**Details**

File is in support directory/files_to_publish.csv

**Value**

dataframe of files to publish

**Examples**

```R
# Not run:  
publishResults("adaprHome")  
# End(Not run)
```

---

**pullSourceInfo**

Create source_info from project.id

**Description**

Create source_info from project.id

**Usage**

```
pullSourceInfo(project.id)
```

**Arguments**

- **project.id** Project ID to use
Details

Creates tree_controller.R directory. This directory is used for operation on the tree.

Value

source_info for generic source_info tree operations

rapidPlot

Make project graph with sync status already computed.

Description

Make project graph with sync status already computed.

Usage

rapidPlot(previousGraph, project.id = getProject(), message = "Running")

Arguments

- previousGraph: output value of createProjectGraph
- project.id: character string of project id examined
- message: string subtitle added to graph

Details

Used by syncProject() to display synchronization progress

Examples

```r
## Not run:
# Requires pandoc location or RStudio
graphData <- graphProject("adaprHome")
rapidPlot(graphData)

## End(Not run)
```
Read data and capture the file information within dependency object

Description
Read data and capture the file information within dependency object

Usage
Read(file.name = "data.csv", description = "Data file",
     read.fcn = guess.read.fcn(file.name), ...)

Arguments
- file.name: name of file
- description: description of data file
- read.fcn: function for reading file
- ...: arguments to read function

Details
Main function for reading file data in projects. Wrapper function for Read.cap, automatically generates file information. Assumes file is in project "Data" directory. Use this in the body of the program. Guesses which function to use to read the file, but user can specify any function that given a file name returns an R object.

Value
object read from file

Examples
```r
## Not run:
source_info <- create_source_file_dir("adaprHome","tree_controller.R")
write.csv(cars,file.path(source_info$data.dir,"test.csv"))
cardata <- Read("test.csv","cars data frame",as.is=TRUE)
file.remove(file.path(source_info$data.dir,"test.csv"))

## End(Not run)
```
Read.cap

*Lower level function that reads data and capture the file information within dependency object*

**Description**

Lower level function that reads data and capture the file information within dependency object

**Usage**

```
Read.cap(file.info, read.fcn, source.info, ...)
```

**Arguments**

- `file.info`: file information list
- `read.fcn`: function for reading the file
- `source.info`: source information list
- `...`: arguments passed to read.fcn

**Value**

object read from files

**Examples**

```
## Not run:
source_info <- create_source_file_dir("adaptrHome","tree_controller.R")
testfile <- file.path(source.info$data.dir,"test.csv")
write.csv(cars,testfile)
fileinfo <- create_file_info(dirname(testfile),basename(testfile),"cars dataset")
Read.cap(fileinfo,read.csv,source.info)
file.remove(file.path(source.info$data.dir,"test.csv"))

## End(Not run)
```

---

read.dependency

*Lower level function that reads the script dependency data from file*

**Description**

Lower level function that reads the script dependency data from file

**Usage**

```
read.dependency(dependency.pointer)
```
readDependency

Arguments
  dependency.pointer  filename with dependency data

Details
  Not for direct use. See readDependency().

Value
  dependency data.frame

Description
  Collect trees from dependency directory

Usage
  readDependency(dependency.dir)

Arguments
  dependency.dir  Directory with dependency files

Details
  adapts to relative or absolute directories

Value
  data frame of stacked dependency files

Examples
  ## Not run:
  readDependency(pullSourceInfo("adaprHome")$dependency.dir)

  ## End(Not run)
readLibrary

**Description**

Read library file

**Usage**

```r
readLibrary(project.id = getProject())
```

**Arguments**

- `project.id`: project.id to read library file

**Details**

Not for direct use. Uses `read.csv` to read autoloaded libraries.

**Value**

Autoloaded library information data

**Examples**

```r
## Not run:
readLibrary("adaprHome")

## End(Not run)
```

---

readtrack

**Description**

Tracks files that read by functions not in adapr and captures the file information within dependency object

**Usage**

```r
ReadTrack(file.name = "data.csv", description = "Data file")
```
Arguments

file.name  name of file (vectorized)
description  description of data file (vectorized)

Details

Allows tracking of files read by other functions than Read. Assumes file is in project "Data" directory

Value

Filepath of file to read

Examples

## Not run:
source_info <- create_source_file_dir("adapHome","tree_controller.R")
write.csv(cars,file.path(source_info$data.dir,"test.csv"))
# Read with any function
temp <- utils::read.csv(file.path(source_info$data.dir,"test.csv"))
ReadTrack("test.csv","cars dataframe")
# Will track the file as though read with Read().
file.remove(file.path(source_info$data.dir,"test.csv"))

## End(Not run)

redirectTree  Lower level function that that changes publish directory or identifies imported project

Description

Lower level function that changes project directory/publish directory or identifies imported project

Usage

redirectTree(project.id0, project.path = NA, swap.directory = NA,
project.libraryTF = FALSE, library.path = "")

Arguments

project.id0  Project name
project.path  Project Parent directory (Directory that contains project)
swap.directory  Project publish directory
project.libraryTF  Logical to use a local (not default) library
library.path  = path to local (not default) library
relocateProject

Details
Not for direct use. See relocate.project

Value
logical for success or not

---

relocateProject  changes project directory/publish directory/library locataion or identifies imported project

Description
changes project directory/publish directory/library locataion or identifies imported project

Usage
relocateProject(project.id0, project.path = NA, swap.directory = NA, project.libraryTF = FALSE, library.path = "")

Arguments
- project.id0  Project name
- project.path Project home directory
- swap.directory Project publish directory
- project.libraryTF Logical to use a local (not default) library
- library.path  = path to local (not default) library

Details
Is wrapper for redirectTree. Does not move the project only indicates new location.

Value
logical for success or not

Examples
```r
## Not run:
relocateProject("adaprTest","mydirectory1","mydirectory2publish")

## End(Not run)
```
**removePackage**

*Remove R package to a project*

**Description**

Remove R package to a project

**Usage**

```r
removePackage(project.id = getProject(), library.name)
```

**Arguments**

- `project.id`: project.id to add R package to
- `library.name`: R package name to add

**Details**

Not for direct use. Remove line from autoloading packages file.

**Value**

Library information data

**Examples**

```r
## Not run:
removePackage("adaprHome","ggplot2")

## End(Not run)
```

---

**removeProject**

*Removes project from orchard, but doesn’t delete project from file system*

**Description**

Removes project from orchard, but doesn’t delete project from file system

**Usage**

```r
removeProject(project.id0)
```

**Arguments**

- `project.id0`: which project to remove from orchard
removeScript

Value

Project listing data frame.

Examples

```r
## Not run:
removeProject("adaprHome")
relcateProject("adaprHome")

## End(Not run)
```

removeScript

Remove an R script from a project. Removes program, dependency, and results.

Description

Remove an R script from a project. Removes program, dependency, and results.

Usage

```
removeScript(project.id = getProject(),
             source.file = get("source_info")$file$file, ask = TRUE)
```

Arguments

- `project.id`: project id
- `source.file`: R script within that project
- `ask`: is a logical whether to ask user

Details

Cannot be undone through adaptR! Will not remove markdown or other program side-effects.

Value

value from file.remove

Examples

```r
## Not run:
remove.program("adaprHome","read_data.R")

## End(Not run)
```
renderRmd

Renders and Rmarkdown file

Description
Renders and Rmarkdown file

Usage
renderRmd(Rmd.file, description = "Rmarkdown", ...)

Arguments
- Rmd.file: Filename of .Rmd file within the R Markdown directory (source_info$markdown.dir)
- description: of rendered file
- ...: extra arguments for rmarkdown::render

Details
Not for direct use. Uses rmarkdown library to access objects in the R script. Will track dependencies for objects used within Rmd file.

Value
Rendered output file information

Examples
```r
## Not run:
source_info <- create_source_file_dir("adaprHome","read_data.R")
renderRmd("read_data.Rmd")

## End(Not run)
```

reportProject

Make plot of network within html documents. Summarize all programs.

Description
Make plot of network within html documents. Summarize all programs.

Usage
reportProject(project.id = getProject(), graph.width = 960, graph.height = 500)
resultsDir

Arguments

- `project.id` Source information list
- `graph.width` Sankey Plot dimensions
- `graph.height` Sankey Plot dimensions

Details

Dose not assume `source_info` in workspace

Value

File path to report html file

Examples

```r
## Not run:
reportProject("adaprHome")

## End(Not run)
```

```
resultsDir returns project's results directory, allows relative directories. Only used within an R script, after create_source_file_dir.
```

Description

Returns project’s results directory, allows relative directories. Only used within an R script, after `create_source_file_dir`.

Usage

```r
resultsDir(sourceInfo = getSourceInfo())
```

Arguments

- `sourceInfo` R `source_info` list created by `create_source_file_dir`

Value

path to data directory

Examples

```r
## Not run:
resultsDir(getSourceInfo())

## End(Not run)
```
**reworkProjectPath**

*Lower level function that collects all trees in dependency.dir and changes the project path*

**Description**

Lower level function that collects all trees in dependency.dir and changes the project path.

**Usage**

```r
reworkProjectPath(dependency.dir = get("source_info")$dependency.dir,
                  new.path = getProjectPath(get("source_info")$project.id))
```

**Arguments**

- `dependency.dir` location of dependency files to rework
- `new.path` file path for the new project path

**Details**

Not for direct use. Used with swapping branches.

---

**runScript**

*Run an R script within a project using devtools::clean_source*

**Description**

Run an R script within a project using devtools::clean_source.

**Usage**

```r
runScript(r = getSourceInfo()$file$file, project.id = getProject(),
          logRmd = FALSE)
```

**Arguments**

- `r` R script within that project (r is short R script for convenience)
- `project.id` project id
- `logRmd` logical indicating whether to create R markdown log

**Details**

Lists scripts if no current script is active or r script is "".
Value

value from clean_source from devtools package

Examples

```r
## Not run:
run.program("read_data.R","adaprHome")

## End(Not run)
```

---

```r
runSourceDirectory(source.directory)
```

Arguments

source.directory

is a directory with R source files to load

Details

Looks for files with .R or .r suffixes.

Value

source file list

Examples

```r
## Not run:
path <- getProjectPath("adaprHome")
path <- file.path(path,"Programs",project.directory.tree$support)
runSourceDirectory(path)

## End(Not run)
```
**scriptLoader**

*Initialize dependency object source_info*

**Description**

Initializes dependency object source_info

**Usage**

`scriptLoader(projectID, script)`

**Arguments**

- `projectID` is the project id
- `script` is the filename for the R script loaded

**Value**

value 1 if success

---

**scriptSubgraph**

*Produces script only subgraph*

**Description**

Produces script only subgraph

**Usage**

`scriptSubgraph(project.id = getProject(), plotTF = FALSE)`

**Arguments**

- `project.id` Project id
- `plotTF` logical to plot subgraph

**Details**

For use with getDepSubgraph

**Value**

list with subgraph in igraph format, layout for plottingss
searchScripts

Searches R scripts and R markdown files within a project.

Usage

searchScripts(matcher, project.id = getProject(), ...)

Arguments

matcher string or regular expression to identify within R or R markdown files
project.id string for project id to search within.
... arguments to grep

Details

Uses grep. Counts lines with matches, but repeats within a line are not counted.

Value

Data frame with file names and counts of lines with matches.

Examples

## Not run:
# Opens read_data.R within the adaprHome project
searchScripts("read_data.R","adaprHome")

## End(Not run)
sendBranch

*Copy dependent programs to swap directory*

**Description**

Copy dependent programs to swap directory

**Usage**

```
sendBranch(branch_cut, all = FALSE)
```

**Arguments**

- `branch_cut`: filename of the base of the branch to send
- `all`: logical indicating whether to send all branches in project

---

sendBranchSI

*Copy dependent programs to swap directory*

**Description**

Copy dependent programs to swap directory

**Usage**

```
sendBranchSI(source_info, branch_cut, all = FALSE)
```

**Arguments**

- `source_info`: Project information list
- `branch_cut`: filename of the base of the branch to send
- `all`: logical indicating whether to send all branches in project
setAdaprOptions

Description
Returns Modifies the primary adapr option file

Usage
```
setAdaprOptions(optionname = "", optionvalue = "")
```

Arguments
- optionname: is name of option to modify
- optionvalue: is new value to give optionname

Value
adaproptions

Examples
```r
## Not run:
opt <- getAdaprOptions()
setAdaprOptions("project.path", opt$project.path)
opt2 <- getAdaprOptions()
identical(opt, opt2)

## End(Not run)
```

setProject

Description
Checks or changes the specified adapr project in R option "adaprProject"

Usage
```
setProject(project.id = "", quickTest = TRUE)
```

Arguments
- project.id: characters specifying project.id of working project
- quickTest: logical whether to check if project exists
showProject

Details

Default is adaprHome. Returns default if project does not exist.

Value

value is specified project or default project

Examples

## Not run:
setProject("adaprHome")

## End(Not run)

---

openProject

*Opens project directory*

Description

Opens project directory

Usage

showProject(project.id = getProject())

Arguments

project.id character string specifies project to open

Details

Use BrowseURL to open project directory.

Examples

## Not run:
showProject("adaprHome")

## End(Not run)
showResults

Opens results directory of project or R script within a project

Description

Opens results directory of project or R script within a project

Usage

showResults(project.id = getProject(), rscript = getSourceInfo()$file$file)

Arguments

project.id character string specifies project
rscript character string specifies the R script result directory to open

Details

Use BrowseURL to open results directory

Examples

## Not run:
showResults("adaprHome")

## End(Not run)

sourceSyncSI

Lower level function that synchronizes project by running necessary R scripts. Loads from source_info list.

Description

Lower level function that synchronizes project by running necessary R scripts. Loads from source_info list.

Usage

sourceSyncSI(source_info, run = TRUE, plot.to.file = FALSE)

Arguments

source_info Project information within source_info list
run logical indicated whether to run or just identify asynchrony
plot.to.file logical for writing file in tree_controller.R directory
sourceSyncSIload

Details

Not usually direct use. See syncProject() and syncTestProject().

Value

Data.frame with sources needed to synchronize with run times

Examples

```r
## Not run:
source_info <- create_source_file_dir("adaprHome","tree_controller.R")
sourcesyncsi(source_info)

## End(Not run)
```

sourceSyncSIload   Synchronize project by IDENTIFYING necessary R scripts

Description

Synchronize project by IDENTIFYING necessary R scripts

Usage

```
sourceSyncSIload(source_info)
```

Arguments

- `source_info`: Project information within source_info list

Details

Not usually direct use. See syncProject() and syncTestProject().

Value

Data.frame with sources needed to synchronize with run times

Examples

```r
## Not run:
source_info <- create_source_file_dir("adaprHome","tree_controller.R")
sourcesyncSIload(source_info)

## End(Not run)
```
sproutProgram  
*Lower level function that generates the shell of a code that is project specific*

**Description**

Lower level function that generates the shell of a code that is project specific

**Usage**

```
sproutProgram(project.id = NA, source.file.name = NA, description = "", seed = 2011, capture.load.command = "library("adapr")", controller = FALSE)
```

**Arguments**

- `project.id`: Name of project
- `source.file.name`: Filename to create
- `description`: What program does
- `seed`: Set seed at program initialization
- `capture.load.command`: Command for loading inference tree library
- `controller`: Logical to insert lines that operate on analysis tree

**Details**

Will not overwrite existing program. Not for direct use. See `makeScript()`.

**Value**

Logical indicating success or not

---

swapProjectPath  
*Lower level function that takes list of dependency file data and changes the project path*

**Description**

Lower level function that takes list of dependency file data and changes the project path

**Usage**

```
swapProjectPath(list.deps,
    new.path = getProjectPath(get("source_info")$project.id))
```
**syncProject**

**Arguments**

- `list.deps` list of dependency file data
- `new.path` file path for the new project path

**Details**

Not for direct use. Used with swapping branches by `reworkProjectPath()`

**Value**

Updated list of dependency data

---

**syncProject**

*Checks the synchronization project and runs scripts needed for synchronization*

**Description**

Checks the synchronization project and runs scripts needed for synchronization

**Usage**

```
syncProject(project.id = getProject(), ask = FALSE)
```

**Arguments**

- `project.id` is project to synchronize
- `ask` logical whether to report estimated run time prior to execution

**Value**

Character string with message about success for synchronization

**Examples**

```
## Not run:
syncProject("adaprHome")

## End(Not run)
```
**syncTest**

*Lower level function that checks the synchrony of source files and their created objects*

**Description**

Lower level function that checks the synchrony of source files and their created objects

**Usage**

```r
syncTest(dagger, tree, plotl = FALSE)
```

**Arguments**

- `dagger`: a directed acyclic graph representing dependencies
- `tree`: dependency tree corresponding to dagger
- `plotl`: logical for plotting or not

**Details**

Not for direct use. See `syncProject()` and `syncTestProject()`.

**Value**

list with synchronizing information

**Examples**

```r
## Not run:
si <- pullSourceInfo("adaprHome")
projInfo <- getProjectInfo(si$dependency.dir)
syncTest(projInfo$graph, projInfo$tree)
## End(Not run)
```

---

**syncTestPI**

*Lower level function that tests the synchrony of files in dependency tree given project information.*

**Description**

Lower level function that tests the synchrony of files in dependency tree given project information.

**Usage**

```r
syncTestPI(project_info, plotl0 = FALSE)
```
syncTestProject

Arguments

- `project_info`  Project information from get_project_info function
- `plot10`  Logical indicated whether to plot the updated files

Details

Not for direct use. See syncProject() and syncTestProject().

Value

list or logical indicated whether project is synchronized or not

Examples

```r
## Not run:
si <- pullSourceInfo("adaprHome")
projInfo <- getProjectInfo(si$dependency.dir)
syncTestPI(projInfo)

## End(Not run)
```

---

**syncTestProject**  
*Tests the synchrony of files in dependency tree*

Description

Tests the synchrony of files in dependency tree

Usage

```
syncTestProject(project.id = getProject())
```

Arguments

- `project.id`  is project to test the synchrony of

Value

list with logical indicated whether project is synchronized or not and details about synchrony

Examples

```r
## Not run:
syncTestProject("adaprHome")

## End(Not run)
```
syncTestSI

Tests the synchrony of files in dependency tree

Description
Tests the synchrony of files in dependency tree

Usage
syncTestSI(source_info, plotl0 = FALSE)

Arguments
  source_info  source_info containing project information
  plotl0      Logical indicated whether to plot the updated files

Details
Not usually direct use. See syncProject() and syncTestProject().

Value
list or logical indicated whether project is synchronized or not

Examples
## Not run:
source_info <- create_source_file_dir("adaprHome","tree_controller.R")
syncTestSI(source_info)

## End(Not run)

syncTrunk

Partial project synchronization of dependencies to an Rscript. Runs only scripts needed for synchronization.

Description
Partial project synchronization of dependencies to an Rscript. Runs only scripts needed for synchronization.

Usage
syncTrunk(rscript, project.id = getProject(), ask = FALSE)
Arguments

- **rscript**: script to synchronize the output of
- **project.id**: is project to synchronize
- **ask**: logical whether to report estimated run time prior to execution

Value

Character string with message about success for synchronization

Examples

```r
## Not run:
syncTrunk("read_data.R","adaprHome")
## End(Not run)
```

**updateAdaprConfig**

*Updates the project list file to include project specific libraries.*

Description

Updates the project list file to include project specific libraries.

Usage

```r
updateAdaprConfig()
```

Details

Adds 2 columns to project listing. (project.library and library.path). This enables project specific libraries.

Projects are by default set to use the default library. If packrat is used, this is probably not the case.

Value

orchard

Examples

```r
## Not run:
orchard <- updateOrchardLibraries()
print(subset(orchard,project.id=="adaprHome"))
## End(Not run)
```
Write

Write object and capture file information

Description

Write object and capture file information

Usage

Write(obj = NULL, file.name = "data.csv", description = "Result file",
     write.fcn = guessWriteFcn(file.name), date = FALSE, ...)

Arguments

obj 
    object to write
file.name 
    file to write to the source "Result" directory
description 
    describes object to write
write.fcn 
    function for writing file of object type
date 
    logical for adding date to filename
... 
    arguments passed to write.fcn

Details

Main writing function for adapr to use in the body of the program. This is a wrapper function for Write.cap. Write dependencies of the script are captured with this function. Write will guess which function to use, but the use can specify any function that writes an R object to a file. The '.rda' suffix will write so that loadFlex can load the object within another R script in the same project.

Value

File information list

Examples

## Not run:
# Within an R script:
source_info <- create_source_file_dir("adaprHome","tree_controller.R")
# Inside R script body:
write.csv(cars,file.path(source_info$results.dir,"test.csv"))
Write(cars,"cars.csv","cars dataframe")
# To examine effect:
showResults()

## End(Not run)
Write.cap

Lower level function that writes data and captures the file information within dependency object

Description

Lower level function that writes data and captures the file information within dependency object

Usage

Write.cap(obj = NULL, file.info, write.fcn, source_info, ...)

Arguments

obj          object to write, if null then open graphics device
file.info   file information list
write.fcn   function to write file
source_info source information list
...         arguments passed to write.fcn

Details

Not usually direct use. See Write() and Graph().

Value

file.info file information returned

Examples

```r
## Not run:
source_info <- create_source_file_dir("adaprHome","tree_controller.R")
testfile <- file_path(source_info$results.dir,"test.csv")
fileinfo <- createFileInfo(dirname(testfile),basename(testfile),"cars dataset")
Write.cap(cars,fileinfo,write.csv,source_info,row.names=FALSE)

## End(Not run)
```
write.dependency  
*Lower level function that writes the dependency object to file*

**Description**

Lower level function that writes the dependency object to file

**Usage**

```r
write.dependency(dependency.object, dependency.pointer)
```

**Arguments**

- `dependency.object`
  - dependency object to output
- `dependency.pointer`
  - filename to write

**Details**

Not for direct use. See finalize_dependency().

**Value**

`TRUE`

---

**WriteTrack**  
*Tracks files written by functions not in adapr and captures the file information within dependency object*

**Description**

Tracks files written by functions not in adapr and captures the file information within dependency object

**Usage**

```r
WriteTrack(file.name = "data.csv", description = "Result file")
```

**Arguments**

- `file.name`  
  - name of file
- `description`  
  - description of data file
Details

Allows tracking of files written by other functions than Write. Assumes file is in Results directory

Value

Filepath of file that was written

Examples

```r
## Not run:
source_info <- create_source_file_dir("adaprHome", "tree_controller.R")
write.csv(cars,file.path(source_info$results.dir,"test.csv"))
WriteTrack("cars.csv","cars dataframe")
showResults()

## End(Not run)
```
## Index

+ **Topic** datasets  
  * project.directory.tree, 72
+ **Topic** package, reproducibility, accountability  
  adapr-package, 5

<table>
<thead>
<tr>
<th>Function/Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>adapr (adapr-package), 5</td>
<td></td>
</tr>
<tr>
<td>adapr-package, 5</td>
<td></td>
</tr>
<tr>
<td>adaprApp, 11</td>
<td></td>
</tr>
<tr>
<td>adaprDependencies, 11</td>
<td></td>
</tr>
<tr>
<td>adaprHomeDir, 12</td>
<td></td>
</tr>
<tr>
<td>adaprInstall, 12</td>
<td></td>
</tr>
<tr>
<td>adaprSheet, 13</td>
<td></td>
</tr>
<tr>
<td>adaprUpdate, 14</td>
<td></td>
</tr>
<tr>
<td>addPackage, 14</td>
<td></td>
</tr>
<tr>
<td>AppLoadFlex, 15</td>
<td></td>
</tr>
<tr>
<td>arcRead, 16</td>
<td></td>
</tr>
<tr>
<td>arcWrite, 16</td>
<td></td>
</tr>
<tr>
<td>browsePubFiles, 17</td>
<td></td>
</tr>
<tr>
<td>checkAdaprHashAlgo, 18</td>
<td></td>
</tr>
<tr>
<td>checkFileHash, 19</td>
<td></td>
</tr>
<tr>
<td>checkFileHashSource, 19</td>
<td></td>
</tr>
<tr>
<td>checkFileSysTimeSource, 20</td>
<td></td>
</tr>
<tr>
<td>checkRmdMode, 21</td>
<td></td>
</tr>
<tr>
<td>checkVersion, 22</td>
<td></td>
</tr>
<tr>
<td>commit2char, 22</td>
<td></td>
</tr>
<tr>
<td>commitProject, 23</td>
<td></td>
</tr>
<tr>
<td>condenseFileInfo, 24</td>
<td></td>
</tr>
<tr>
<td>create_source_file_dir, 27</td>
<td></td>
</tr>
<tr>
<td>createFileInfo, 24</td>
<td></td>
</tr>
<tr>
<td>createMarkdown, 25</td>
<td></td>
</tr>
<tr>
<td>createProgramGraph, 26</td>
<td></td>
</tr>
<tr>
<td>dataDir, 28</td>
<td></td>
</tr>
<tr>
<td>defaultAdaprSetup, 28</td>
<td></td>
</tr>
<tr>
<td>dependency (dependency-class), 29</td>
<td></td>
</tr>
<tr>
<td>dependency-class, 29</td>
<td></td>
</tr>
<tr>
<td>Digest, 29</td>
<td></td>
</tr>
<tr>
<td>fileInfoProjects, 30</td>
<td></td>
</tr>
<tr>
<td>finalize_dependency, 31</td>
<td></td>
</tr>
<tr>
<td>firstProject, 31</td>
<td></td>
</tr>
<tr>
<td>get_orchard, 42</td>
<td></td>
</tr>
<tr>
<td>getAdaprOptions, 32</td>
<td></td>
</tr>
<tr>
<td>getDepSubgraph, 33</td>
<td></td>
</tr>
<tr>
<td>getFileInfo, 33</td>
<td></td>
</tr>
<tr>
<td>getFileSysTime, 34</td>
<td></td>
</tr>
<tr>
<td>getLibrary, 35</td>
<td></td>
</tr>
<tr>
<td>getProject, 35</td>
<td></td>
</tr>
<tr>
<td>getProjectInfo, 36</td>
<td></td>
</tr>
<tr>
<td>getProjectInfoSI, 37</td>
<td></td>
</tr>
<tr>
<td>getProjectLibrary, 37</td>
<td></td>
</tr>
<tr>
<td>getProjectPath, 38</td>
<td></td>
</tr>
<tr>
<td>getProjectPublishPath, 39</td>
<td></td>
</tr>
<tr>
<td>getProjectSwapPath, 39</td>
<td></td>
</tr>
<tr>
<td>getPubResults, 40</td>
<td></td>
</tr>
<tr>
<td>getSourceInfo, 41</td>
<td></td>
</tr>
<tr>
<td>getUpstream, 41</td>
<td></td>
</tr>
<tr>
<td>git_path, 48</td>
<td></td>
</tr>
<tr>
<td>gitAdd, 42</td>
<td></td>
</tr>
<tr>
<td>gitCommit, 43</td>
<td></td>
</tr>
<tr>
<td>gitConfigure, 44</td>
<td></td>
</tr>
<tr>
<td>gitConfigureTest, 44</td>
<td></td>
</tr>
<tr>
<td>gitHistorySearch, 45</td>
<td></td>
</tr>
<tr>
<td>gitIgnoreLibrary, 46</td>
<td></td>
</tr>
<tr>
<td>gitInfo, 46</td>
<td></td>
</tr>
<tr>
<td>gitInit, 47</td>
<td></td>
</tr>
<tr>
<td>gitProvenance, 48</td>
<td></td>
</tr>
<tr>
<td>Graph, 49</td>
<td></td>
</tr>
<tr>
<td>graphProject, 50</td>
<td></td>
</tr>
<tr>
<td>guess_read.fcn, 50</td>
<td></td>
</tr>
<tr>
<td>guessWriteFcn, 51</td>
<td></td>
</tr>
<tr>
<td>idPackages, 52</td>
<td></td>
</tr>
<tr>
<td>idSync, 52</td>
<td></td>
</tr>
<tr>
<td>importData, 53</td>
<td></td>
</tr>
<tr>
<td>initialize_dependency_info, 54</td>
<td></td>
</tr>
<tr>
<td>initProject, 54</td>
<td></td>
</tr>
</tbody>
</table>

106
install, 55
installLibrary, 56
installProjectPackages, 57

Library, 58
listBranches, 58
listDataFiles, 59
listProjects, 60
listScripts, 60
Load. branch, 61
loadAdapTest, 61
loadFlex, 62
loadInstallLibraryFile, 63

makeDependencyGraphObj, 63
makeFunction, 64
makeHyperlink, 65
makeScript, 65
makeSummaryGraph, 66
monitorParallelSync, 67

openProjectList, 68
openScript, 68

parallelSync, 69
path.expand, 70
plantOrchard, 70
plantTree, 71
programIOTable, 72
project.directory.tree, 72
projectReportMarkdown, 73
projectReportSend, 74
publishResults, 75
pullSourceInfo, 75

rapidPlot, 76
Read, 77
Read.cap, 78
read.dependency, 78
readDependency, 79
readLibrary, 80
ReadTrack, 80
redirectTree, 81
relocateProject, 82
removePackage, 83
removeProject, 83
removeScript, 84
renderRmd, 85
reportProject, 85

resultsDir, 86
reworkProjectPath, 87
runScript, 87
runSourceDirectory, 88

scriptLoader, 89
scriptSubgraph, 89
searchScripts, 90
sendBranch, 91
sendBranchSI, 91
setAdaprOptions, 92
setProject, 92
showProject, 93
showResults, 94
sourceSyncSI, 94
sourceSyncSIload, 95
sproutProgram, 96
swapProjectPath, 96
syncProject, 97
syncTest, 98
syncTestPI, 98
syncTestProject, 99
syncTestSI, 100
syncTrunk, 100

updateAdaprConfig, 101

Write, 102
Write.cap, 103
write.dependency, 104
WriteTrack, 104