Package ‘PaRe’

May 21, 2024

Type Package

Title A Way to Perform Code Review or QA on Other Packages

Version 0.1.13

Language en-US

Description Reviews other packages during code review by looking at their dependencies, code style, code complexity, and how internally defined functions interact with one another.

URL https://github.com/darwin-eu-dev/PaRe

BugReports https://github.com/darwin-eu-dev/PaRe/issues

License Apache License (>= 2)

Encoding UTF-8

RoxygenNote 7.2.3

Imports cli (>= 3.6.0), cyclocomp (>= 1.1.0), desc (>= 1.4.2), DiagrammeR (>= 1.0.9), DiagrammeRsvg (>= 0.1), dplyr (>= 1.1.0), glue (>= 1.6.2), lintr (>= 3.0.2), magrittr (>= 2.0.3), pak (>= 0.2.0), rmarkdown (>= 2.20), rsvg (>= 2.4.0), stringr (>= 1.5.0), igraph (>= 1.3.5), utils, R6 (>= 2.5.1), git2r (>= 0.31.0), checkmate (>= 2.1.0)

Suggests ggplot2, plotly, ggraph, DT, magick, withr, cowplot, knitr, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Config/testthat/parallel true

NeedsCompilation no

Author Maarten van Kessel [aut, cre] (<https://orcid.org/0009-0006-8832-6030>)

Maintainer Maarten van Kessel <m.l.vankessel@erasmusmc.nl>

Repository CRAN

Date/Publication 2024-05-21 12:50:14 UTC
**R topics documented:**

addPareArticle .................................................. 2
checkDependencies ............................................. 4
checkInstalled .................................................. 5
Code ............................................................. 6
countPackageLines ............................................. 7
exportDiagram ................................................... 8
File ............................................................... 9
Function .......................................................... 12
functionUseGraph ............................................... 14
funsUsedInFile .................................................. 14
funsUsedInLine .................................................. 15
defGetApplyCall .................................................. 15
defGetApplyFromLines ........................................... 16
defGetDefaultPermittedPackages ................................. 16
defGetDefinedFunctions ......................................... 17
defGetDlplyCall ................................................... 18
defGetDlplyCallFromLines ....................................... 19
defGetDoCall ...................................................... 19
defGetDoCallFromLines ......................................... 20
defGetExportedFunctions ........................................ 20
defGetFunCall .................................................... 21
defGetFunctionDiagram .......................................... 21
defGetFunctionUse ............................................... 22
defGetFunsPerDefFun ............................................. 24
defGetGraphData .................................................. 24
defGetMultiLineFun ............................................... 26
defGraphToDot ..................................................... 26
deflintRepo ....................................................... 27
deflintScore ...................................................... 28
defmakeGraph ..................................................... 29
defmakeReport .................................................... 30
defpkgDiagram ..................................................... 31
defRepository ..................................................... 32
defwhiteList ..................................................... 35

**Index**

addPareArticle addPareArticle

**Description**

Writes an Rmd-file to ./vignettes/articles/PaReReport.Rmd. The relative path is dictated by the specified path in the Repository object.
addPareArticle

Usage

addPareArticle(repo)

Arguments

repo (Repository) Repository object.

Value

NULL Writes Rmd-file to ./vignettes/articles/PaReReport.Rmd

Examples

fetchedRepo <- tryCatch(
  {
    # Set dir to clone repository to.
    tempDir <- tempdir()
    pathToRepo <- file.path(tempDir, "glue")
    
    # Clone repo
    git2r::clone(
      url = "https://github.com/darwin-eu/IncidencePrevalence.git",
      local_path = pathToRepo
    )
    
    # Create instance of Repository object.
    repo <- PaRe::Repository$new(path = pathToRepo)
    
    # Set fetchedRepo to TRUE if all goes well.
    TRUE
  },
  error = function(e) {
    # Set fetchedRepo to FALSE if an error is encountered.
    FALSE
  },
  warning = function(w) {
    # Set fetchedRepo to FALSE if a warning is encountered.
    FALSE
  }
)

if (fetchedRepo) {
  # Run makeReport on the Repository object.
  addPareArticle(repo)
}
checkDependencies

Description
Check package dependencies

Usage
checkDependencies(
  repo,
  dependencyType = c("Imports", "Depends"),
  verbose = TRUE
)

Arguments
- repo (Repository)
  Repository object.
- dependencyType (character())
  Types of dependencies to be included
- verbose (logical(): TRUE) TRUE or FALSE. If TRUE, progress will be reported.

Value
(data.frame())
Data frame with all the packages that are now permitted.

<table>
<thead>
<tr>
<th>column</th>
<th>data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>package</td>
<td>character()</td>
</tr>
<tr>
<td>version</td>
<td>character()</td>
</tr>
</tbody>
</table>

Examples

# Set cache, usually not required.
withr::local_envvar(
  R_USER_CACHE_DIR = tempfile()
)

fetchedRepo <- tryCatch(
  {
    # Set dir to clone repository to.
    tempDir <- tempdir()
    pathToRepo <- file.path(tempDir, "glue")

    # Clone repo

checkInstalled

```r
# Download repository.
git2r::clone(
  url = "https://github.com/tidyverse/glue.git",
  local_path = pathToRepo
)

# Create instance of Repository object.
repo <- PaRe::Repository$new(path = pathToRepo)

# Set fetchedRepo to TRUE if all goes well.
TRUE
),
error = function(e) {
  # Set fetchedRepo to FALSE if an error is encountered.
  FALSE
},
warning = function(w) {
  # Set fetchedRepo to FALSE if a warning is encountered.
  FALSE
}
)

if (fetchedRepo) {
  # Use checkDependencies on the Repository object.
  checkDependencies(repo)
  checkDependencies(repo, dependencyType = c("Imports", "Suggests"))
}
```

---

**checkInstalled**

**checkInstalled**

**Description**

Checks if suggested packages are installed.

**Usage**

`checkInstalled()`

**Value**

`logical`

Logical depending if suggested packages are installed.
Description

Class representing a piece of code.

Methods

Public methods:

- \texttt{Code$new()}  
- \texttt{Code$print()}  
- \texttt{Code$getLines()}  
- \texttt{Code$getNLines()}  
- \texttt{Code$getName()}  
- \texttt{Code$clone()}

**Method** \texttt{new()}: Initializer method

*Usage:*  
\texttt{Code$new(name, lines)}

*Arguments:*  
name (character(1))  
   Name of Code object.
lines (character(n))  
   Vector of lines Code object.

*Returns:* invisible(self)

**Method** \texttt{print()}: Overload generic print, to print Code object.

*Usage:*  
\texttt{Code$print(...)}

*Arguments:*  
... further arguments passed to or from other methods. See \texttt{print}.

*Returns:* (character(n))

**Method** \texttt{getLines()}: Get method for lines.

*Usage:*  
\texttt{Code$getLines()}

*Returns:* (character(n)) Vector of lines in the Code object.

**Method** \texttt{getNLines()}: Get method for number of lines.

*Usage:*  
\texttt{Code$getNLines()}
countPackageLines

Returns: (numeric(1)) Number of lines in the Code object.

**Method getName():** Get method for Name.

*Usage:*

Code$getName()

*Returns: (character(1)) Name of the Code object.*

**Method clone():** The objects of this class are cloneable with this method.

*Usage:*

Code$clone(deep = FALSE)

*Arguments:*

depth Whether to make a deep clone.

**See Also**

Other Representations: `File`, `Function`, `Repository`

---

countPackageLines  countPackageLines

countPackageLines

**Description**

Counts the package lines of a `Repository` object.

**Usage**

`countPackageLines(repo)`

**Arguments**

repo (Repository) Repository object.

**Value**

(tibble ) Tibble containing the amount of lines per file in the Repository object.
Examples

```r
fetchedRepo <- tryCatch(
  {
    # Set dir to clone repository to.
    tempDir <- tempdir()
    pathToRepo <- file.path(tempDir, "glue")

    # Clone repo
    git2r::clone(
      url = "https://github.com/tidyverse/glue.git",
      local_path = pathToRepo
    )

    # Create instance of Repository object.
    repo <- PaRe::Repository$new(path = pathToRepo)

    # Set fetchedRepo to TRUE if all goes well.
    TRUE
  },
  error = function(e) {
    # Set fetchedRepo to FALSE if an error is encountered.
    FALSE
  },
  warning = function(w) {
    # Set fetchedRepo to FALSE if a warning is encountered.
    FALSE
  }
)

if (fetchedRepo) {
  # Run countPackageLines on the Repository object.
  countPackageLines(repo = repo)
}
```

Description

Exports the diagram from pkgDiagram to a PDF-file.

Usage

```r
exportDiagram(diagram, fileName)
```

Arguments

- `diagram` *(grViz)*
  - Graph object from pkgDiagram.
fileName          (character)
Path to save the diagram to, as PDF.

Value
(NULL)

Examples
fetchedRepo <- tryCatch(
  {
    # Set dir to clone repository to.
    tempDir <- tempdir()
    pathToRepo <- file.path(tempDir, "glue")

    # Clone repo
    git2r::clone(
      url = "https://github.com/tidyverse/glue.git",
      local_path = pathToRepo
    )

    # Create instance of Repository object.
    repo <- PaRe::Repository$new(path = pathToRepo)

    # Set fetchedRepo to TRUE if all goes well.
    TRUE
  },
  error = function(e) {
    # Set fetchedRepo to FALSE if an error is encountered.
    FALSE
  },
  warning = function(w) {
    # Set fetchedRepo to FALSE if a warning is encountered.
    FALSE
  }
)

if (fetchedRepo) {
  # Run pkgDiagram on the Repository object.
  pkgDiagram(repo = repo) %>%
    # Export the diagram to a temp file.
    exportDiagram(fileName = tempfile())
}
Super class

PaRe::Code -> File

Methods

Public methods:

• File$new()
• File$getFunctions()
• File$getFunctionTable()
• File$getType()
• File$getFilePath()
• File$getBlameTable()
• File$clone()

Method new(): Initializer method

Usage:
File$new(repoPath, filePath)

Arguments:
repoPath (character)
Path to repository.

filePath (character)
Relative path to file

Returns: invisible(self)

Method getFunctions(): Get method to get a list of Function objects

Usage:
File$getFunctions()

Returns: (list)
List of Function objects.

Method getFunctionTable(): Get method to retrieve the function table.

Usage:
File$getFunctionTable()

Returns: (data.frame)

<table>
<thead>
<tr>
<th>column</th>
<th>data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>character</td>
</tr>
<tr>
<td>lineStart</td>
<td>integer</td>
</tr>
<tr>
<td>lineEnd</td>
<td>numeric</td>
</tr>
<tr>
<td>nArgs</td>
<td>integer</td>
</tr>
<tr>
<td>cycloComp</td>
<td>integer</td>
</tr>
</tbody>
</table>

Method getType(): Gets type of file
Usage:
File$getType()
Returns: (character)

Method getFilePath(): Gets relative file path
Usage:
File$getFilePath()
Returns: (character)

Method getBlameTable(): Gets table of git blame
Usage:
File$getBlameTable()
Returns: (tibble)

Method clone(): The objects of this class are cloneable with this method.
Usage:
File$clone(deep = FALSE)
Arguments:
deep  Whether to make a deep clone.

See Also

Other Representations: Code, Function, Repository

Examples

fetchedRepo <- tryCatch(
{
  # Set dir to clone repository to.
  tempDir <- tempdir()
  pathToRepo <- file.path(tempDir, "glue")

  # Clone repo
  git2r::clone(
    url = "https://github.com/tidyverse/glue.git",
    local_path = pathToRepo
  )

  # Create instance of Repository object.
  repo <- PaRe::Repository$new(path = pathToRepo)

  # Set fetchedRepo to TRUE if all goes well.
  TRUE
},
  error = function(e) {
    # Set fetchedRepo to FALSE if an error is encountered.
    FALSE
  },
)
warning = function(w) {
    # Set fetchedRepo to FALSE if a warning is encountered.
    FALSE
}

if (fetchedRepo) {
    files <- repo$getRFiles()
    files[[1]]
}

Function  

R6 Function class.

Description

Class representing a function.

Super class

PaRe::Code -> Function

Methods

Public methods:

- Function$new()
- Function$getFunction()
- Function$clone()

Method new(): Initializer for Function object.

Usage:
Function$new(name, lineStart, lineEnd, lines)

Arguments:
name (character)
    Name of Function.
lineStart (numeric)
    Line number where function starts in File.
lineEnd (numeric)
    Line number where function ends in File.
lines (c)
    Vector of type character Lines of just the function in File.

Returns: invisible(self)

Method getFunction(): Get method to get defined functions in a File object.

Usage:
Function$getFunction()

Returns: (data.frame)
Method \texttt{clone()}: The objects of this class are cloneable with this method.

\textit{Usage:}
\texttt{Function$clone(deep = FALSE)}

\textit{Arguments:}
- \texttt{deep} Whether to make a deep clone.

\textbf{See Also}

Other Representations: \texttt{Code, File, Repository}

\textbf{Examples}

```r
fetchedRepo <- tryCatch(
  {
    # Set dir to clone repository to.
    tempDir <- tempdir()
    pathToRepo <- file.path(tempDir, "glue")

    # Clone repo
    git2r::clone(
      url = "https://github.com/tidyverse/glue.git",
      local_path = pathToRepo
    )

    # Create instance of Repository object.
    repo <- PaRe::Repository$new(path = pathToRepo)

    # Set fetchedRepo to TRUE if all goes well.
    TRUE
  },
  error = function(e) {
    # Set fetchedRepo to FALSE if an error is encountered.
    FALSE
  },
  warning = function(w) {
    # Set fetchedRepo to FALSE if a warning is encountered.
    FALSE
  }
)

if (fetchedRepo) {
  files <- repo$getRFiles()
  file <- files[[1]]
```
funsUsedInFile

```r
funs <- file$getFunctions()
funs[[1]]
```

---

**Description**

functionUseGraph

**Usage**

```
functionUseGraph(repo)
```

**Arguments**

- `repo` *(Repository)*

**Value**

```
(graph)
```

---

**Description**

Support function

**Usage**

```
funsUsedInFile(files, verbose = FALSE)
```

**Arguments**

- `files` *(list) of (File)*
- `verbose` *(logical)*

**Value**

```
(list)
```
Description

Support function for funsUsedInFile.

Usage

funsUsedInLine(lines, name, i, verbose = FALSE)

Arguments

- **lines**: (c) of (character)
- **name**: (character)
- **i**: (numeric)
- **verbose**: (logical: FALSE)

Value

(data.frame)

<table>
<thead>
<tr>
<th>column</th>
<th>data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>pkg</td>
<td>character</td>
</tr>
<tr>
<td>fun</td>
<td>character</td>
</tr>
<tr>
<td>line</td>
<td>numeric</td>
</tr>
</tbody>
</table>

Description

getApplyCall

Usage

getApplyCall(fun, defFuns)

Arguments

- **fun**: (Function)
  
Function object.
- **defFuns**: (data.frame)
  
See getDefinedFunctions
getDefaultPermittedPackages

Value

(data.frame)

getApplyFromLines

getApplyFromLines

Description

getApplyFromLines

Usage

getApplyFromLines(lines)

Arguments

lines (c)
Vector of (character). See getDefinedFunctions

Value

(character)

getDefaultPermittedPackages

getDefaultPermittedPackages

Description

Gets permitted packages. An internet connection is required.

Usage

getDefaultPermittedPackages()

Value

(tibble)

<table>
<thead>
<tr>
<th>column</th>
<th>data type</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>package</td>
<td>character</td>
<td></td>
</tr>
<tr>
<td>version</td>
<td>character</td>
<td></td>
</tr>
</tbody>
</table>
Examples

# Set cache
withr::local_envvar(
  R_USER_CACHE_DIR = tempfile()
)
if (interactive()) {
  getDefaultPermittedPackages()
}

Description

Gets all the defined functions from a Repository object.

Usage

g_definedFunctions(repo)

Arguments

repo (Repository)
Repository object.

Value

(data.frame)

<table>
<thead>
<tr>
<th>column</th>
<th>data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>character</td>
</tr>
<tr>
<td>lineStart</td>
<td>integer</td>
</tr>
<tr>
<td>lineEnd</td>
<td>numeric</td>
</tr>
<tr>
<td>nArgs</td>
<td>integer</td>
</tr>
<tr>
<td>cycloComp</td>
<td>integer</td>
</tr>
<tr>
<td>fileName</td>
<td>character</td>
</tr>
</tbody>
</table>

Examples

fetchedRepo <- tryCatch(
  {
    # Set dir to clone repository to.
    tempDir <- tempdir()
    pathToRepo <- file.path(tempDir, "glue")
# Clone repo
git2r::clone(
  url = "https://github.com/tidyverse/glue.git",
  local_path = pathToRepo
)

# Create instance of Repository object.
repo <- PaRe::Repository$new(path = pathToRepo)

# Set fetchedRepo to TRUE if all goes well.
TRUE

),
error = function(e) {
  # Set fetchedRepo to FALSE if an error is encountered.
  FALSE
},
warning = function(w) {
  # Set fetchedRepo to FALSE if a warning is encountered.
  FALSE
})

if (fetchedRepo) {
  repo <- PaRe::Repository$new(pathToRepo)

  getDefinedFunctions(repo)
}

getDlplyCall

---

getDlplyCall

---

**Description**

getDlplyCall

**Usage**

getDlplyCall(fun, defFuns)

**Arguments**

- **fun** *(Function)*
  
  Function object.

- **defFuns** *(data.frame)*
  
  See getDefinedFunctions

**Value**

*(data.frame)*
**getDplyCallFromLines**

Description

getDplyCallFromLines

Usage

getDplyCallFromLines(lines)

Arguments

- **lines** (c)
  Vector of (character).

Value

(character)

**getDoCall**

Description

getDoCall

Usage

getDoCall(fun, defFuns)

Arguments

- **fun** (Function)
  Function object.
- **defFuns** (data.frame)
  See getDefinedFunctions

Value

(data.frame)
getDoCallFromLines

Description
getDoCallFromLines

Usage
getDoCallFromLines(lines)

Arguments
lines (c) Vector of (character). See getDefinedFunctions

Value
(character)

getExportedFunctions

Description
Gets all the exported functions of a package, from NAMESPACE.

Usage
getExportedFunctions(path)

Arguments
path (character) Path to package

Value
(c) Vector of character exported functions.
getFunCall

Description
getFunCall

Usage
getFunCall(fun, defFuns)

Arguments
- **fun** *(Function)* Function object.
- **defFuns** *(data.frame)* See *getDefinedFunctions*.

Value
*(data.frame)*

getFunctionDiagram

Description
Create a subset of the package diagram containing all incoming and outgoing paths from a specified function.

Usage
getFunctionDiagram(repo, functionName)

Arguments
- **repo** *(Repository)* Repository object.
- **functionName** *(character)* Name of the function to get all paths from.

Value
*(htmlwidgets)* Subsetted diagram. See *grViz*
Examples

```r
fetchedRepo <- tryCatch(
  {
    # Set dir to clone repository to.
    tempDir <- tempdir()
    pathToRepo <- file.path(tempDir, "glue")

    # Clone repo
    git2r::clone(
      url = "https://github.com/tidyverse/glue.git",
      local_path = pathToRepo
    )

    # Create instance of Repository object.
    repo <- PaRe::Repository$new(path = pathToRepo)

    # Set fetchedRepo to TRUE if all goes well.
    TRUE
  },
  error = function(e) {
    # Set fetchedRepo to FALSE if an error is encountered.
    FALSE
  },
  warning = function(w) {
    # Set fetchedRepo to FALSE if a warning is encountered.
    FALSE
  }
)

if (fetchedRepo) {
  # Run getFunctionDiagram on the Repository object.
  getFunctionDiagram(repo = repo, functionName = "glue")
}
```

---

**getFunctionUse**

**summariseFunctionUse**

---

**Description**

Summarise functions used in R package.

**Usage**

```r
getFunctionUse(repo, verbose = FALSE)
```

**Arguments**

- **repo** *(Repository)*
  - Repository object.
getFunctionUse

verbose (logical: FALSE)
Prints message to console which file is currently being worked on.

Value

(tibble)

<table>
<thead>
<tr>
<th>column</th>
<th>data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>character</td>
</tr>
<tr>
<td>line</td>
<td>numeric</td>
</tr>
<tr>
<td>pkg</td>
<td>character</td>
</tr>
<tr>
<td>fun</td>
<td>character</td>
</tr>
</tbody>
</table>

Examples

fetchedRepo <- tryCatch(
  {
    # Set dir to clone repository to.
    tempDir <- tempdir()
    pathToRepo <- file.path(tempDir, "glue")

    # Clone repo
    git2r::clone(
      url = "https://github.com/tidyverse/glue.git",
      local_path = pathToRepo
    )

    # Create instance of Repository object.
    repo <- PaRe::Repository$new(path = pathToRepo)

    # Set fetchedRepo to TRUE if all goes well.
    TRUE,
    error = function(e) {
      # Set fetchedRepo to FALSE if an error is encountered.
      FALSE
    },
    warning = function(w) {
      # Set fetchedRepo to FALSE if a warning is encountered.
      FALSE
    }
  },
  error = function(e) {
    # Set fetchedRepo to FALSE if an error is encountered.
    FALSE
  },
  warning = function(w) {
    # Set fetchedRepo to FALSE if a warning is encountered.
    FALSE
  }
)

if (fetchedRepo) {
  # Run getFunctionUse on the Repository object.
  getFunctionUse(repo = repo, verbose = TRUE)
}

getFunsPerDefFun

Description
getFunsPerDefFun

Usage
getFunsPerDefFun(files, defFuns)

Arguments
files (list)
List of File objects.
defFuns (data.frame)
See getDefinedFunctions.

Value
data.frame
column data type
from character
to character

graph
getGraphData

description
Get the dependency interactions as a graph representation.

Usage
getGraphData(repo, packageTypes = c("Imports"))

Arguments
repo (Repository)
Repository object.
packageTypes (c: c("Imports")) of (character) Any of the following options may be included in a vector:
getGraphData

- "imports"
- "depends"
- "suggests"
- "enhances"
- "linkingto"

Value

(as_tbl_graph)

Examples

```r
fetchedRepo <- tryCatch(
  {
    # Set dir to clone repository to.
    tempDir <- tempdir()
    pathToRepo <- file.path(tempDir, "glue")

    # Clone repo
    git2r::clone(
      url = "https://github.com/tidyverse/glue.git",
      local_path = pathToRepo
    )

    # Create instance of Repository object.
    repo <- PaRe::Repository$new(path = pathToRepo)

    # Set fetchedRepo to TRUE if all goes well.
    TRUE
  },
  error = function(e) {
    # Set fetchedRepo to FALSE if an error is encountered.
    FALSE
  },
  warning = function(w) {
    # Set fetchedRepo to FALSE if a warning is encountered.
    FALSE
  }
)

if (fetchedRepo) {
  # Run getGraphData on the Repository object.
  if (interactive()) {
    getGraphData(repo = repo, packageTypes = c("Imports"))
  }
}
```
getMultiLineFun

**Description**
getMultiLineFun

**Usage**
getMultiLineFun(line, lines)

**Arguments**
- line: (numeric)
  - Current line number.
- lines: (c)
  - Vector of (character) lines.

**Value**
(character)

---

graphToDot

**Description**
graphToDot

**Usage**
graphToDot(graph)

**Arguments**
- graph: (graph)

**Value**
htmlwidgets
See grViz.
Description

Get all the lintr messages of the `Repository` object.

Usage

`lintRepo(repo)`

Arguments

- `repo` (Repository)

Value

(data.frame)

<table>
<thead>
<tr>
<th>column</th>
<th>data type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>character</td>
<td>Name of the file</td>
</tr>
<tr>
<td>line_number</td>
<td>double</td>
<td>Line in which the message was found</td>
</tr>
<tr>
<td>column_number</td>
<td>double</td>
<td>Column in which the message was found</td>
</tr>
<tr>
<td>type</td>
<td>character</td>
<td>Type of message</td>
</tr>
<tr>
<td>message</td>
<td>character</td>
<td>Style, warning, or error message</td>
</tr>
<tr>
<td>line</td>
<td>character</td>
<td>Line of code in which the message was found</td>
</tr>
<tr>
<td>linter</td>
<td>character</td>
<td>Linter used</td>
</tr>
</tbody>
</table>

Examples

```r
fetchedRepo <- tryCatch(
  {
    # Set dir to clone repository to.
    tempDir <- tempdir()
    pathToRepo <- file.path(tempDir, "glue")
    
    # Clone repo
    git2r::clone(
      url = "https://github.com/tidyverse/glue.git",
      local_path = pathToRepo
    )
    
    # Create instance of Repository object.
    repo <- PaRe::Repository$new(path = pathToRepo)
    
    # Set fetchedRepo to TRUE if all goes well.
    TRUE
  },
  
  TRUE
),
```
error = function(e) {
  # Set fetchedRepo to FALSE if an error is encountered.
  FALSE
},
warning = function(w) {
  # Set fetchedRepo to FALSE if a warning is encountered.
  FALSE
}
)

if (fetchedRepo) {
  # Run lintRepo on the Repository object.
  messages <- lintRepo(repo = repo)
}

Description
Function that scores the lintr output as a percentage per message type (style, warning, error). Lintr messages / lines assessed * 100

Usage
lintScore(repo, messages)

Arguments
repo (Repository)
Repository object.
messages (data.frame)
Data frame containing lintr messages. See lintRepo.

Value
(tibble)

  type (character) Type of message.
  pct (double) Score.

Examples
fetchedRepo <- tryCatch(
  {
    # Set dir to clone repository to.
    tempDir <- tempdir()
    pathToRepo <- file.path(tempDir, "glue")
  
```
# Clone repo
git2r::clone(
  url = "https://github.com/tidyverse/glue.git",
  local_path = pathToRepo
)

# Create instance of Repository object.
repo <- PaRe::Repository$new(path = pathToRepo)

# Set fetchedRepo to TRUE if all goes well.
TRUE
},
error = function(e) {
  # Set fetchedRepo to FALSE if an error is encountered.
  FALSE
},
warning = function(w) {
  # Set fetchedRepo to FALSE if a warning is encountered.
  FALSE
}
)

if (fetchedRepo) {
  messages <- lintRepo(repo = repo)
  # Run lintScore on the Repository object.
  lintScore(repo = repo, messages = messages)
}

makeGraph

makeGraph

Description

Makes the graph

Usage

makeGraph(funsPerDefFun, pkgName, expFuns, ...)

Arguments

funsPerDefFun  (data.frame)
  Functions per defined function data.frame.

pkgName  (character)
  Name of package.

expFuns  (data.frame)
  Exported functinos data.frame.

...  Optional other parameters for grViz.
**Value**

(htmlwidget)
Diagram of the package. See grViz.

---

**Description**

Uses rmarkdown’s render function to render a html-report of the given package.

**Usage**

```r
makeReport(repo, outputFile, showCode = FALSE)
```

**Arguments**

- `repo` *(Repository)*
  - Repository object.
- `outputFile` *(character)*
  - Path to html-file.
- `showCode` *(logical: FALSE)*
  - Logical to show code or not in the report.

**Value**

*(NULL)*

**Examples**

```r
fetchedRepo <- tryCatch(
  {
    # Set dir to clone repository to.
    tempDir <- tempdir()
    pathToRepo <- file.path(tempDir, "glue")

    # Clone repo
    git2r::clone(
      url = "https://github.com/darwin-eu/IncidencePrevalence.git",
      local_path = pathToRepo
    )

    # Create instance of Repository object.
    repo <- PaRe::Repository$new(path = pathToRepo)

    # Set fetchedRepo to TRUE if all goes well.
    TRUE
  }
)
```
error = function(e) {
  # Set fetchedRepo to FALSE if an error is encountered.
  FALSE
},
warning = function(w) {
  # Set fetchedRepo to FALSE if a warning is encountered.
  FALSE
}
}

if (fetchedRepo) {
  # Run makeReport on the Repository object.
  makeReport(repo = repo, outputFile = tempfile())
}

pkgDiagram

Description

Creates a diagram of all defined functions in a package.

Usage

pkgDiagram(repo, verbose = FALSE, ...)

Arguments

repo  (Repository)
  Repository object.
verbose  (logical)
  Turn verbose messages on or off.
...
  Optional other parameters for grViz.

Value

(htmlwidget)
  Diagram htmlwidget object. See createWidget

Examples

fetchedRepo <- tryCatch(
  {
    # Set dir to clone repository to.
    tempDir <- tempdir()
    pathToRepo <- file.path(tempDir, "glue")
  },
  error = function(e) {
    # Set fetchedRepo to FALSE if an error is encountered.
    FALSE
  },
  warning = function(w) {
    # Set fetchedRepo to FALSE if a warning is encountered.
    FALSE
  }
)

if (fetchedRepo) {
  # Run makeReport on the Repository object.
  makeReport(repo = repo, outputFile = tempfile())
}
# Clone repo
git2r::clone(
  url = "https://github.com/tidyverse/glue.git",
  local_path = pathToRepo
)

# Create instance of Repository object.
repo <- PaRe::Repository$new(path = pathToRepo)

# Set fetchedRepo to TRUE if all goes well.
TRUE

error = function(e) {
  # Set fetchedRepo to FALSE if an error is encountered.
  FALSE
},
warning = function(w) {
  # Set fetchedRepo to FALSE if a warning is encountered.
  FALSE
}

if (fetchedRepo) {
  # Run pkgDiagram on the Repository object.
  pkgDiagram(repo = repo)
}

---

**Repository**  
*R6 Repository class.*

**Description**

Class representing the Repository

**Methods**

**Public methods:**

- `Repository$new()`
- `Repository$getName()`
- `Repository$getPath()`
- `Repository$getFiles()`
- `Repository$getRFiles()`
- `Repository$getDescription()`
- `Repository$getFunctionUse()`
- `Repository$gitCheckout()`
- `Repository$gitPull()`
- `Repository$gitBlame()`
- Repository$clone()

**Method new()**: Initializer for Repository class

*Usage:*
Repository$new(path)

*Arguments:*
path (character)

  Path to R package project

*Returns*: invisible(self)

**Method getName()**: Get method for name.

*Usage:*
Repository$getName()

*Returns*: (character)

Repository name

**Method getPath()**: Get method for path

*Usage:*
Repository$getPath()

*Returns*: (character)

Path to Repository folder

**Method getFiles()**: Get method to get a list of File objects.

*Usage:*
Repository$getFiles()

*Returns*: (list)

List of File objects.

**Method getRFiles()**: Get method to get only R-files.

*Usage:*
Repository$getRFiles()

*Returns*: (list)

List of File objects.

**Method getDescription()**: Get method to get the description of the package. See: description.

*Usage:*
Repository$getDescription()

*Returns*: (description)

Description object.

**Method getFunctionUse()**: Get method for functionUse, will check if functionUse has already been fetched or not.

*Usage:*
Repository$getFunctionUse()
Repository

Returns: (data.frame)
See getFunctionUse.

Method gitCheckout(): Method to run 'git checkout <branch/commit hash>'
Usage:
Repository$gitCheckout(branch, ...)
Arguments:
branch (character)	Name of branch or a hash referencing a specific commit.
... Further parameters for checkout.
Returns: invisible(self)

Method gitPull(): Method to run 'git pull'
Usage:
Repository$gitPull(...)
Arguments:
... Further parameters for pull.
Returns: invisible(self)

Method gitBlame(): Method to fetch data generated by 'git blame'.
Usage:
Repository$gitBlame()
Returns: (tibble)
column   data type
repository character
author     character
file       character
date       character
lines      integer

Method clone(): The objects of this class are cloneable with this method.
Usage:
Repository$clone(deep = FALSE)
Arguments:
deep Whether to make a deep clone.

See Also
Other Representations: Code, File, Function

Examples
fetchedRepo <- tryCatch(
(  
  # Set dir to clone repository to.
  tempDir <- tempdir()
  pathToRepo <- file.path(tempDir, "glue")

  # Clone repo
  git2r::clone(
    url = "https://github.com/tidyverse/glue.git",
    local_path = pathToRepo
  )

  # Create instance of Repository object.
  repo <- PaRe::Repository$new(path = pathToRepo)

  # Set fetchedRepo to TRUE if all goes well.
  TRUE
  ),
  error = function(e) {
    # Set fetchedRepo to FALSE if an error is encountered.
    FALSE
  },
  warning = function(w) {
    # Set fetchedRepo to FALSE if a warning is encountered.
    FALSE
  }
  )

  if (fetchedRepo) {
    repo
  }
)

---

**whiteList**

<table>
<thead>
<tr>
<th>whiteList</th>
<th>whiteList</th>
</tr>
</thead>
</table>

**Description**

data.frame containing links to csv-files which should be used to fetch white-listed dependencies.

**Usage**

whiteList

**Format**

An object of class tbl_df (inherits from tbl, data.frame) with 3 rows and 4 columns.

**Details**

By default three csv’s are listed:
1. darwin
2. hades
3. tidyverse

The data.frame is locally fetched under: `system.file(package = "PaRe", "whiteList.csv")`

Manual insertions into this data.frame can be made, or the data.frame can be overwritten entirely.

The data.frame itself has the following structure:

<table>
<thead>
<tr>
<th>column</th>
<th>data type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>character</td>
<td>name of the source</td>
</tr>
<tr>
<td>link</td>
<td>character</td>
<td>link or path to the csv-file</td>
</tr>
<tr>
<td>package</td>
<td>character</td>
<td>columnname of the package name column in the csv-file being linked to</td>
</tr>
<tr>
<td>version</td>
<td>character</td>
<td>columnname of the version column in the csv-file being linked to</td>
</tr>
</tbody>
</table>

The csv-files that are being pointed to should have the following structure:

**Examples**

```r
if (interactive()) {
  # Dropping tidyverse
  whiteList <- whiteList %>%
    dplyr::filter(source != "tidyverse")

  # getDefaultPermittedPackages will now only use darwin and hades
  getDefaultPermittedPackages()
}
```
Index

* Representations
  - Code, 6
  - File, 9
  - Function, 12
  - Repository, 32
* datasets
  - whiteList, 35
addPareArticle, 2
as_tbl_graph, 25
c, 12, 15, 16, 19, 20, 24, 26
countPackageLines, 7
counter, 9–13, 15–17, 19–21, 23, 24,
  26–30, 33, 34, 36
checkDependencies, 4
checkInstalled, 5
checkout, 34
Code, 6, 11, 13, 34
countPackageLines, 7
createWidget, 31
data.frame, 10, 12, 15–19, 21, 24, 27–29, 34
description, 33
double, 27, 28
desc
exportDiagram, 8
Function, 7, 10, 11, 12, 15, 18, 19, 21, 34
functionUseGraph, 14
funsUsedInFile, 14
funsUsedInLine, 15
getApplyCall, 15
getApplyFromLines, 16
defaultPermittedPackages, 16
definedFunctions, 15, 16, 17, 18–21, 24
dlplyCall, 18
dlplyCallFromLines, 19
doCall, 19
doCallFromLines, 20
getDependedFunctions, 20
getFunCall, 21
getOptionDiagram, 21
getOptionUse, 22, 34
getFunsPerDefFun, 24
getGraphData, 24
getMultiLineFun, 26
graph, 14, 26
graphToDot, 26
grViz, 8, 21, 26, 29–31
integer, 10, 13, 17, 34
lintRepo, 27, 28
lintScore, 28
list, 10, 14, 24, 33
logical, 5, 14, 15, 23, 30, 31
makeGraph, 29
makeReport, 30
numeric, 10, 12, 13, 15, 17, 23, 26
PaRe::Code, 10, 12
pkgDiagram, 8, 31
print, 6
pull, 34
Repository, 2, 3, 7, 11, 13, 14, 17, 21, 22, 24,
  27, 28, 30, 31, 32
tibble, 7, 11, 16, 23, 28, 34
whiteList, 35