

Package ‘PaRe’

July 21, 2025

Type Package

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

Version 0.1.15

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 7.3.1

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), igrph (>= 1.3.5), utils, R6 (>= 2.5.1), git2r (>= 0.31.0), checkmate (>= 2.1.0), parallel

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

VignetteBuilder knitr

Config/testthat/edition 3

Config/testthat/parallel true

NeedsCompilation no

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

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

Repository CRAN

Date/Publication 2024-12-02 10:10:06 UTC

Contents

addPareArticle	2
checkDependencies	4
checkInstalled	5
Code	5
countPackageLines	7
exportDiagram	8
File	9
Function	11
functionUseGraph	13
funsUsedInFile	14
funsUsedInLine	14
getApplyCall	15
getApplyFromLines	15
getDefaultPermittedPackages	16
getDefinedFunctions	17
getDlplyCall	18
getDlplyCallFromLines	19
getDoCall	19
getDoCallFromLines	20
getExportedFunctions	20
getFunCall	21
getFunctionDiagram	21
getFunctionUse	22
getFunsPerDefFun	24
getGraphData	24
getMultiLineFun	26
graphToDot	26
lintRepo	27
lintScore	28
makeGraph	29
makeReport	30
pkgDiagram	31
Repository	32
whiteList	35
Index	37

addPareArticle	<i>addPareArticle</i>
----------------	-----------------------

Description

Writes an Rmd-file to `./vignettes/articles/PaReReport.Rmd`. The relative path is dictated by the specified path in the [Repository](#) object.

Usage

```
addPaReArticle(repo)
```

Arguments

repo ([Repository](#)) Repository object.

Value

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

Examples

```
fetchedReader <- 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 (fetchedReader) {
  # Run makeReport on the Repository object.
  addPaReArticle(repo)
}
```

checkDependencies *checkDependencies*

Description

Check package dependencies

Usage

```
checkDependencies(repo, dependencyType = c("Imports", "Depends"), nThreads = 1)
```

Arguments

repo	(Repository) Repository object.
dependencyType	(character()) Types of dependencies to be included
nThreads	(numeric(1): 1) Number of threads to use to fetch permitted packages

Value

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

column	data type
package	character()
version	character()

Examples

```
# Set cahce, 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
    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.

Code

R6 Code class

Description

Class representing a piece of code.

Methods

Public methods:

- [Code\\$new\(\)](#)
- [Code\\$print\(\)](#)
- [Code\\$getLines\(\)](#)
- [Code\\$getNLines\(\)](#)
- [Code\\$getName\(\)](#)
- [Code\\$clone\(\)](#)

Method `new()`: Initializer method

Usage:

`Code$new(name, lines)`

Arguments:

`name` (character(1))

Name of Code object.

`lines` (character(n))

Vector of lines Code object.

Returns: invisible(self)

Method `print()`: Overload generic print, to print Code object.

Usage:

`Code$print(...)`

Arguments:

... further arguments passed to or from other methods. See [print](#).

Returns: (character(n))

Method `getLines()`: Get method for lines.

Usage:

`Code$getLines()`

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

Method `getNLines()`: Get method for number of lines.

Usage:

`Code$getNLines()`

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:

`deep` Whether to make a deep clone.

See Also

Other Representations: [File](#), [Function](#), [Repository](#)

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

```
fetchRepo <- 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 fetchRepo to TRUE if all goes well.  
    TRUE  
  },  
  error = function(e) {  
    # Set fetchRepo 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)
}

```

exportDiagram

exportDiagram

Description

Exports the diagram from pkgDiagram to a PDF-file.

Usage

```
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())
}
```

File

R6 File class

Description

Class representing a file containing code.

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](#))

column	data type
name	character
lineStart	integer
lineEnd	numeric
nArgs	integer
cycloComp	integer

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](#))

column	data type
<code>name</code>	(character)
<code>lineStart</code>	(integer)
<code>lineEnd</code>	(numeric)
<code>nArgs</code>	(integer)
<code>cycloComp</code>	(integer)

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

Usage:

`Function$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

See Also

Other Representations: [Code](#), [File](#), [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$getRFiles()
    file <- files[[1]]
    funs <- file$getFunctions()
    funs[[1]]
  }
}
```

functionUseGraph *functionUseGraph*

Description

functionUseGraph

Usage

functionUseGraph(repo)

Arguments

repo ([Repository](#))

Value[\(graph\)](#)

funsUsedInFile	<i>funsUsedInFile</i>
----------------	-----------------------

Description

Support function

Usage

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

Arguments

files	(list) of (File)
verbose	(logical)

Value[\(list\)](#)

funsUsedInLine	<i>funsUsedInLine</i>
----------------	-----------------------

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)

column	data type
pkg	character
fun	character
line	numeric

getApplyCall	<i>getApplyCall</i>
--------------	---------------------

Description

getApplyCall

Usage

getApplyCall(fun, defFuns)

Arguments

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

Value

(data.frame)

getApplyFromLines	<i>getApplyFromLines</i>
-------------------	--------------------------

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(nThreads = 1)
```

Arguments

nThreads (numeric(1): 1) Number of threads to use to fetch permitted packages

Value[\(tibble\)](#)

column	data type
package	character
version	character

Examples

```
# Set cache
withr::local_envvar(
  R_USER_CACHE_DIR = tempfile()
)

if (interactive()) {
  getDefaultPermittedPackages()
}
```

getDefinedFunctions *getDefinedFunctions*

Description

Gets all the defined functions from a [Repository](#) object.

Usage

```
getDefinedFunctions(repo)
```

Arguments

repo ([Repository](#))
Repository object.

Value

([data.frame](#))

column	data type
name	character
lineStart	integer
lineEnd	numeric
nArgs	integer
cycloComp	integer
fileName	character

Examples

```
fetchRepo <- 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 fetchRepo 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)
}
```

getDplyCall

getDplyCall

Description

getDplyCall

Usage

```
getDplyCall(fun, defFuns)
```

Arguments

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

Value

([data.frame](#))

`getDplyCallFromLines` *getDplyCallFromLines*

Description

`getDplyCallFromLines`

Usage

`getDplyCallFromLines(lines)`

Arguments

`lines` [\(c\)](#)
Vector of [\(character\)](#).

Value

[\(character\)](#)

`getDoCall` *getDoCall*

Description

`getDoCall`

Usage

`getDoCall(fun, defFuns)`

Arguments

`fun` [\(Function\)](#)
Function object.

`defFuns` [\(data.frame\)](#)
See [getDefinedFunctions](#)

Value

[\(data.frame\)](#)

`getDoCallFromLines` *getDoCallFromLines*

Description

`getDoCallFromLines`

Usage

`getDoCallFromLines(lines)`

Arguments

`lines` (c)
Vector of (character). See [getDefinedFunctions](#)

Value

(character)

`getExportedFunctions` *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`*getFunCall*

Description`getFunCall`**Usage**`getFunCall(fun, defFuns)`**Arguments**

<code>fun</code>	(Function) Function object.
<code>defFuns</code>	(data.frame) See getDefinedFunctions .

Value`(data.frame)`

`getFunctionDiagram`*subsetGraph*

Description

Create a subset of the package diagram containing all in coming and out going paths from a specified function.

Usage`getFunctionDiagram(repo, functionName)`**Arguments**

<code>repo</code>	(Repository) Repository object.
<code>functionName</code>	(character) Name of the function to get all paths from.

Value`(htmlwidgets)`
Subsetted diagram. See [grViz](#)

Examples

```
fetchedReader <- 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 fetchedReader to TRUE if all goes well.  
    TRUE  
  },  
  error = function(e) {  
    # Set fetchedReader to FALSE if an error is encountered.  
    FALSE  
  },  
  warning = function(w) {  
    # Set fetchedReader to FALSE if a warning is encountered.  
    FALSE  
  }  
)  
  
if (fetchedReader) {  
  # Run getFunctionDiagram on the Repository object.  
  getFunctionDiagram(repo = repo, functionName = "glue")  
}
```

getFunctionUse

summariseFunctionUse

Description

Summarise functions used in R package.

Usage

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

Arguments

repo [\(Repository\)](#)
Repository object.

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

Value

(tibble)

column	data type
file	character
line	numeric
pkg	character
fun	character

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 getFunctionUse on the Repository object.
  getFunctionUse(repo = repo, verbose = TRUE)
}

```

getFunsPerDefFun	<i>getFunsPerDefFun</i>
------------------	-------------------------

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

getGraphData	<i>getGraphData</i>
--------------	---------------------

Description

Get the dependency interactions as a graph representation.

Usage

```
getGraphData(repo, packageTypes = c("Imports"), nThreads = 1)
```

Arguments

repo	(Repository) Repository object.
packageTypes	(<code>c("Imports")</code>) of (character) Any of the following options may be included in a vector: <ul style="list-style-type: none"> • "imports" • "depends" • "suggests" • "enhances" • "linkingto"
nThreads	(<code>numeric(1): 1</code>) Number of threads to use to fetch permitted packages

Value

([as_tbl_graph](#))

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 getGraphData on the Repository object.
  if (interactive()) {
    getGraphData(repo = repo, packageTypes = c("Imports"))
  }
}

```

```
    }  
}
```

getMultiLineFun	<i>getMultiLineFun</i>
-----------------	------------------------

Description

getMultiLineFun

Usage

getMultiLineFun(line, lines)

Arguments

line	(numeric) Current line number.
lines	(c) Vector of (character) lines.

Value

([character](#))

graphToDot	<i>graphToDot</i>
------------	-------------------

Description

graphToDot

Usage

graphToDot(graph)

Arguments

graph	(graph)
-------	---------------------------

Value

htmlwidgets
See [grViz](#).

lintRepo *lintRepo*

Description

Get all the lintr messages of the [Repository](#) object.

Usage

```
lintRepo(repo)
```

Arguments

repo ([Repository](#))

Value

([data.frame](#))

column	data type	description
filename	character	Name of the file
line_number	double	Line in which the message was found
column_number	double	Column in which the message was found
type	character	Type of message
message	character	Style, warning, or error message
line	character	Line of code in which the message was found
linter	character	Linter used

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 lintRepo on the Repository object.
    messages <- lintRepo(repo = repo)
  }

```

 lintScore

lintScore

Description

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

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 functions data.frame.
...		Optional other parameters for grViz .

Value

(htmlwidget)
Diagram of the package. See [grViz](#).

makeReport	<i>makeReport</i>
------------	-------------------

Description

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

Usage

```
makeReport(repo, outputFile, showCode = FALSE, nThreads = 1)
```

Arguments

repo	(Repository) Repository object.
outputFile	(character) Path to html-file.
showCode	(logical : FALSE) Logical to show code or not in the report.
nThreads	(numeric (1): 1) Number of threads to use to fetch permitted packages

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/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

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")
```

```
# 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\\$name\(\)](#)
- [Repository\\$path\(\)](#)
- [Repository\\$files\(\)](#)
- [Repository\\$Rfiles\(\)](#)
- [Repository\\$description\(\)](#)
- [Repository\\$functionUse\(\)](#)
- [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 fro 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()

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

whiteList

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:

column	data type	description
source	character	name of the source
link	character	link or path to the csv-file
package	character	columnname of the package name column in the csv-file being linked to
version	character	columnname of the version column in the csv-file being linked to

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

Examples

```
if (interactive()) {  
  # Dropping tidyverse  
  whiteList <- whiteList %>%  
    dplyr::filter(source != "tidyverse")  
  
  # getDefaultPermittedPackages will now only use darwin and hades  
  getDefaultPermittedPackages()  
}
```

Index

* Representations

Code, 5
File, 9
Function, 11
Repository, 32

* datasets

whiteList, 35

addPareArticle, 2

as_tbl_graph, 25

c, 12, 14, 15, 19, 20, 25, 26

character, 8, 10, 12, 14–17, 19–21, 23–30,
33, 34, 36

checkDependencies, 4

checkInstalled, 5

checkout, 34

Code, 5, 11, 12, 34

countPacakelLines, 7

createWidget, 31

data.frame, 10, 12, 15, 17–19, 21, 24, 27–29,
34

description, 33

double, 27, 28

exportDiagram, 8

File, 7, 9, 12, 14, 24, 33, 34

Function, 7, 10, 11, 11, 15, 18, 19, 21, 34

functionUseGraph, 13

funsUsedInFile, 14

funsUsedInLine, 14

getApplyCall, 15

getApplyFromLines, 15

getDefaultPermittedPackages, 16

getDefinedFunctions, 15, 17, 18–21, 24

getDlplyCall, 18

getDlplyCallFromLines, 19

getDoCall, 19

getDoCallFromLines, 20

getExportedFunctions, 20

getFunCall, 21

getFunctionDiagram, 21

getFunctionUse, 22, 34

getFunsPerDefFun, 24

getGraphData, 24

getMultiLineFun, 26

graph, 14, 26

graphToDot, 26

grViz, 8, 21, 26, 29–31

integer, 10, 12, 17, 34

lintRepo, 27, 28

lintScore, 28

list, 10, 14, 24, 33

logical, 5, 14, 23, 30, 31

makeGraph, 29

makeReport, 30

numeric, 10, 12, 14, 15, 17, 23, 26

PaRe::Code, 9, 11

pkgDiagram, 8, 31

print, 6

pull, 34

Repository, 2, 3, 7, 11–13, 17, 21, 22, 25, 27,
28, 30, 31, 32

tibble, 7, 10, 16, 23, 28, 34

whiteList, 35