Package: lucode2 (via r-universe)

September 6, 2024

Type Package

Title Code Manipulation and Analysis Tools

Version 0.47.10

Date 2024-08-07

Description A collection of tools which allow to manipulate and analyze code.

License BSD_2_clause + file LICENSE

URL https://github.com/pik-piam/lucode2, https://doi.org/10.5281/zenodo.4389418

BugReports https://github.com/pik-piam/lucode2/issues

Imports callr, citation (>= 0.11.3), data.table, desc, devtools, dplyr, lintr (>= 3.1.0), rlang, tools, usethis (>= 2.1.0), withr, yaml

Suggests covr, gdx, gdxrrw, gert, ggplot2, knitr, lusweave, magclass, poorman, renv, rmarkdown, styler, testthat

Encoding UTF-8

LazyData no

RoxygenNote 7.3.2

Config/testthat/parallel true

Config/testthat/edition 3

Config/testthat/start-first checkRequiredPackages, updateRepo

Config/Keywords tool

Repository https://pik-piam.r-universe.dev

RemoteUrl https://github.com/pik-piam/lucode2

RemoteRef HEAD

RemoteSha e514cb0206e883d89fa425f38fd2c696d91ecf34

Contents

`addEOF` *add EOF comment*

Description

Add EOF text to all files in the code in order to ease debugging

Usage

```
addEOF(path = ".", filetypes = c("inc", "prn", "gms"))
```

Arguments

`path`	path to the main folder of the model
`filetypes`	file types the function should be applied to

Author(s)

Anastasis Giannousakis

See Also

`removeEOF`

Examples

```
## Not run:
addEOF()

## End(Not run)
```

addGitHubActions *addGitHubActions*

Description

This function adds a standard Github action workflow called "check.yaml" to the project which runs lucode2::check(), checks the validation key, and creates a coverage report using codecov. This file is overwritten automatically each time this function is run and should not be edited by hand. Additional Github actions can be added as separate files.

Usage

```
addGitHubActions(lib = ".")
```

Arguments

lib	Path to the package

Details

In addition, this function adds a codecov.yml to the repository, if not already existing. This file is only created if missing and can be edited manually.

Author(s)

Jan Philipp Dietrich

See Also

buildLibrary

Examples

```
## Not run:
addGitHubActions()

## End(Not run)
```

autoFormat *autoFormat*

Description

Apply auto-formatting using styler::style_file to the given files. Does not change indentation.

Usage

```
autoFormat(
  files = getFilesToLint(),
  ignoreLintFreeFiles = TRUE,
  lintAfterwards = TRUE
)
```

Arguments

files	A character vector of paths to files that should be auto-formatted.
ignoreLintFreeFiles	If set to TRUE (the default) files without linter warnings are not auto-formatted.
lintAfterwards	If set to TRUE (the default) return linter results for the auto-formatted files.

Author(s)

Pascal Sauer

See Also

getFilesToLint

Examples

```
## Not run:
lucode2::autoFormat()

## End(Not run)
```

buildLibrary *buildLibrary*

Description

Builds R libraries. Includes checks for consistency. Find solutions to common problems at https://github.com/pik-piam/discussions/discussions/18

Usage

```
buildLibrary(
  lib = ".",
  cran = TRUE,
  updateType = NULL,
  updateLucode2 = TRUE,
  autoCheckRepoUpToDate = TRUE
)
```

Arguments

lib	Path to the package
cran	If cran-like test is needed
updateType	Either an integer or character string:

number	string	description
1	major	for API breaking changes
2 (default)	minor	for new features or improvements
3	patch	for bugfixes and corrections
4	development	only for packages in development stage
0	none	version has already been incremented

updateLucode2	Update lucode2 if possible and run buildLibrary with new version instead.
autoCheckRepoUpToDate	Automatically check if your repository is up to date. If FALSE the user is asked.

Details

This function is designed to help building and checking R libraries. It performs the following steps:

- Version: Determination of a new version number (Can also be defined by the user).
- Date: Determination of a the date of the build (Can also be defined by the user).
- Linter: Check for code style problems.
- R check: Check whether the library is consistent and can be built.
- Package building: Builds the .zip and .tar.gz packages under windows. Under linux, only the .tar.gz package is built.

Note

The behavior of buildLibrary can be configured via the `.buildLibrary` file in the main folder of the package. It uses YAML format and can contain the following entries:

- **ValidationKey**: This entry always exists and is written automatically by `buildLibrary` It confirms that the package has been successfully build via the function.
- **AutocreateReadme** (optional): yes/no - decides whether `buildLibrary` automatically updates the README.md file or not (default: yes)
- **AddInReadme** (optional): Additional entries to be added to the autocreated README. Provided either in markdown format or as paths to RMarkdown (Rmd) or Markdown (md) files
- **AddLogoReadme** (optional): Additional logo to be added to the autocreated README. Provided as path to logo in PNG format
- **AcceptedWarnings** (optional): a list of Warnings which should be ignored by `buildLibrary` (autocompletion via asterisks allowed)
- **AcceptedNotes** (optional): a list of Notes which should be ignored by `buildLibrary` (autocompletion via asterisks allowed)
- **allowLinterWarnings**: yes/no - If set to "no" linter warnings will stop the build process. (default: yes)

Author(s)

Jan Philipp Dietrich, Anastasis Giannousakis, Markus Bonsch, Pascal Sauer

See Also

`package2readme`, `lint`, `autoFormat`

Examples

```
## Not run:
buildLibrary()

## End(Not run)
```

---

check *check*

Description

Builds documentation and runs checks, tests, and linter. Find solutions to common problems at https://github.com/pik-piam/discussions/discussions/18

Usage

```
check(
  lib = ".",
  cran = TRUE,
  config = loadBuildLibraryConfig(lib),
  runLinter = TRUE
)
```

Arguments

lib	Path to the package
cran	If cran-like test is needed
config	A configuration defining AcceptedWarnings, AcceptedNotes, and allowLinterWarnings. By default the .buildLibrary file is read.
runLinter	Set to FALSE to skip the linter.

Details

This function builds documentation including vignettes via devtools::document(). It runs devtools::check() (without tests), then in a separate clean R session it runs devtools::test(), and finally lucode2::lint(). Before linting ".lintr" config files are created if missing. The actual linter rules are defined in lintrRules. In general undesirable functions and operators result in linter warnings, but not in the tests and vignettes subdirectories. Warnings and notes in checks and tests are only allowed if the given config defines them as accepted, otherwise this function will stop.

Author(s)

Jan Philipp Dietrich, Pascal Sauer

See Also

buildLibrary, lint, lintrRules

check, test

Examples

```
## Not run:
lucode2::check()

## End(Not run)
```

`checkRepoUpToDate` *Check if repo is up to date with upstream*

Description

Checks whether the local repo is up-to-date with the remote tracking branch and the default branch of the upstream remote. Will throw an error if not up-to-date and prints a git command to update.

Usage

```
checkRepoUpToDate(pathToRepo = ".", autoCheckRepoUpToDate = TRUE)
```

Arguments

`pathToRepo`	The path to the git repo.
`autoCheckRepoUpToDate`	If FALSE do not check automatically and instead just ask the user.

Author(s)

Pascal Sauer

`checkRequiredPackages` *checkRequiredPackages*

Description

Check if one or more packages are available and try to install the missing packages. Throw an error if at least one of the required packages is still missing after the installation attempt.

Usage

```
checkRequiredPackages(
  requiredPackages,
  requiredFor = "",
  installFunction = install.packages,
  readlineFunction = readline,
  libPaths = .libPaths()
)
```

Arguments

requiredPackages	One or more names of packages that are checked using requireNamespace.
requiredFor	Optional single string. What the packages are required for, usually the name of a function.
installFunction	Optional function, defaults to install.packages. Will be called during checkRequiredPackages like this: installFunction(missingPackages, libPaths[[1]]). Only needed if the required packages are not available on CRAN or the configured repos (getOption("repos")). In that case you might want to use something like this: function(...) remotes::install_github("USDA-ERS/MTED-HARr").
readlineFunction	This argument was added for testing. A function to get an answer from the user.
libPaths	This argument was added for testing. Where to look for and install the required packages.

Author(s)

Pascal Sauer

See Also

requireNamespace

Examples

```
## Not run:
checkRequiredPackages(c("ggplot2", "lusweave"), "lucode2::readRuntime(..., plot = TRUE)")

## End(Not run)
```

---

checkup *checkup*

Description

Checks the current R setup for common problems and reports info such as OS and R version.

Usage

```
checkup()
```

Value

Invisibly, the report as a list.

Author(s)

Pascal Sauer

check_versions *Package version check tool*

Description

Checks if there are CRAN-packages with the same name as those in our PIK-CRAN whose version is newer

Usage

```
check_versions(mail = TRUE, test = FALSE, gitpath = NULL)
```

Arguments

mail	whether an email notification is sent to RSE
test	to test whether auto email sending works
gitpath	if an email notification has to be sent out, the path to the git repo

Author(s)

Anastasis Giannousakis

conditionalCopy *conditionalCopy*

Description

Copy a file from lucode2 into the current package.

Usage

```
conditionalCopy(relativePath, nameInInstExtdata = basename(relativePath))
```

Arguments

relativePath	The destination to copy to.
nameInInstExtdata	The source file name in lucode2's extdata, if it differs from relativePath's basename.

Details

For normal packages, it will simply overwrite the given file from the corresponding file in lucode2's extdata. For lucode2 itself, it instead checks if the file in extdata matches the file in the main folder. If not, it asks if the file in extdata should be updated.

eprint *extended Print*

Description

An extended print command which formats information in a way that it is good to use for a log-file

Usage

```
eprint(var_name, envir = parent.frame())
```

Arguments

var_name	name of the variable that should be printed as string
envir	environment from which the variable should be read (by default the environment from which the function is called)

Author(s)

Jan Philipp Dietrich, Oliver Richters

See Also

eprint_list

Examples

```
## Not run:
a <- 1:3
eprint("a")

## End(Not run)

### print additional information concerning loaded configuration###
### ePrint (extended Print) offers an extended output functionality which
### allows to create easily log-files with all relevant information
```

eprint_list *Extended list print*

Description

Same as eprint, but expecting a vector with variable names

Usage

```
eprint_list(var_list, envir = parent.frame())
```

Arguments

var_list	Vector containing names of several variables that should be printed
envir	environment from which the variable should be read (by default the environment from which the function is called)

Author(s)

Jan Philipp Dietrich

See Also

eprint

Examples

```
a <- 1:3
b <- "blub"
lucode2:::eprint_list(c("a", "b"))
```

---

extract_arguments *Extract arguments*

Description

Extracts the value (right-hand-side) of a string of the structure "name=value" and converts it to an appropriate format. This file also reads arguments from command line. To use this script you have to include it by typing source("readArgs.R") in your script and call readArgs(allowed_args) including all arguments that can be read from command line.

Usage

```
extract_arguments(inputArg)
```

Arguments

inputArg	string of the structure "name=value"

Value

value	the value (right-hand-side) of the string converted into an appropriate format

Author(s)

Jan Philipp Dietrich

See Also

readArgs

Examples

```
## Not run:
extract_arguments("bla=1:9")
# [1] 1 2 3 4 5 6 7 8 9

extract_arguments("blub=3,5,7")
# [1] 3 5 7

extract_arguments("ble=hallo")
# [1] "hallo"

## End(Not run)
```

`findCoupledruns` *findCoupledruns*

Description

Extracts scenario names from coupled runs in the given outputfolder. The scenario names will be extracted based on the folder names of the results folders.

Usage

```
findCoupledruns(resultsfolder)
```

Arguments

`resultsfolder`	Path to an output folder.

Value

A vector containing the names of the scenarios.

Author(s)

David Klein

findDeps *findDeps*

Description

Find all dependencies of an R project.

Usage

```
findDeps(devDeps = TRUE)
```

Arguments

devDeps	Whether development dependencies should also be checked.

Details

This is a wrapper around 'renv::dependencies()' that does not report dependencies on core R packages, because these are always available.

Value

A dataframe documenting which dependency is needed where.

Author(s)

Pascal Sauer

See Also

dependencies

findIterations *findIterations*

Description

Collects paths to all coupled runs (iterations) in `modelpath` that contain `runname`. For each entry in `runname` the paths are sorted by the modification time of the respective fulldata.gdx

Usage

```
findIterations(runname, modelpath = ".", latest = FALSE)
```

Arguments

runname	Scenarioname or vector of scenarionames.
modelpath	Path or vector of paths where iterations are searched for.
latest	Logical indicating if only the latest iteration of a runname is returned.

Value

A vector containing the paths to the iterations of coupled runs.

Author(s)

David Klein

---

fixBuildLibraryMergeConflict

fixBuildLibraryMergeConflict

Description

Fix merge conflicts in files auto-edited by buildLibrary (.buildlibrary, .zenodo.json, DESCRIPTION, README.md).

Usage

```
fixBuildLibraryMergeConflict(lib = ".")
```

Arguments

lib	The path to the project with a merge conflict.

Details

The ValidationKey in .buildlibrary will be set to an empty string and the higher version number in DESCRIPTION is used. buildLibrary needs to be run after this function to deal with the ValidationKey and the merge markers in .zenodo and README.md.

`fixfile` *fixfile*

Description

Fix file format of text files (e.g. line/file endings).

Usage

```
fixfile(f)
```

Arguments

`f`	path to the file that should be fixed

Author(s)

Jan Philipp Dietrich

`functionHeaderDefaults` *functionHeaderDefaults*

Description

used to quickly read in the default values of a function

Usage

```
functionHeaderDefaults(...)
```

Arguments

`...`	parameters that shall be assigned to the global environment of R

Value

no direct return, values are assigned to .GlobalEnv

Author(s)

Benjamin Leon Bodirsky

Examples

```
## Not run:
test <- function(a = "klk", b = "kjlkv", kk = 3) {
paste(a, b, kk)
}
functionHeaderDefaults(a = "klk", b = "kjlkv", kk = 3)
print(a)
paste(a, b, kk)

## End(Not run)
```

`getClusterLoad` *getClusterLoad*

Description

Returns information about cluster load in case that the command "sclass" is available. Otherwise, it returns NULL

Usage

```
getClusterLoad()
```

Value

NULL, if command "sclass" is not available, otherwise returns a named vector with current load on available partitions

Author(s)

Jan Philipp Dietrich

`getFilesToLint` *getFilesToLint*

Description

Get the R files the current git user is responsible for to pass them to the auto-formatter and/or linter.

Usage

```
getFilesToLint(pathToGitRepo = ".")
```

Arguments

`pathToGitRepo`	path to a git repository

Details

The files of interest are identified using git via system() (and shell() for windows). All currently untracked files and changed files (both staged and unstaged) are collected, as well as files that were changed in non-merge commits authored by the current git user since the last version commit (a commit where .buildLibrary was changed, presumably to increase the version number). Of those the absolute paths to .R, .Rmd and .Rnw files are returned as a character vector.

Author(s)

Pascal Sauer

See Also

`lint`, `autoFormat`

Examples

```
lucode2:::getFilesToLint()
```

---

getPackageAuthors *getPackageAuthors*

Description

Creates a suggestion for an Authors entry for the DESCRIPTION of a package. Suggestion is based on author information in roxygen headers and authors are ranked based on number of mentionings.

Usage

```
getPackageAuthors(folder = "R")
```

Arguments

`folder`	R folder of the package

Details

Please be aware that the output will most likely require some manual processing before it can be used in the DESCRIPTION!

Author(s)

Jan Philipp Dietrich

`getScenNames` *getScenNames*

Description

Get the scenario names (titles) of runs from the specified output folder(s).

Usage

```
getScenNames(dirs)
```

Arguments

`dirs`	vector of paths to the used output folders.

Value

A vector containing the titles used as scenario names for e.g. plots

Author(s)

Lavinia Baumstark

`gitAuthors` *gitAuthors*

Description

Print each git author's first name, last name, and mail address in a way that could be copy pasted into DESCRIPTION. Please note that this list usually still needs to be carefully checked e.g. for duplicates and cleaned.

Usage

```
gitAuthors()
```

Value

Invisibly, a data.frame with the following columns: raw, firstName, lastName, mailAdress.

Author(s)

Pascal Sauer

incrementVersion *incrementVersion*

Description

Increment a version number at the specified position.

Usage

```
incrementVersion(currentVersion, position, defLengths = 3)
```

Arguments

currentVersion	The current package version as a string like "1.23.456"
position	An integer defining which part of the version number will be increased. Use 1 for major version, 2 for minor, 3 for patch/bugfix, 4 for development.
defLengths	An integer defining how many parts make up the resulting version number.

Value

The new version string.

Author(s)

Jan Philipp Dietrich, Anastasis Giannousakis, Markus Bonsch, Pascal Sauer

See Also

buildLibrary

Examples

```
lucode2:::incrementVersion("1.23.45", 3)
```

isVersionUpdated *isVersionUpdated*

Description

Checks if the version number in the DESCRIPTION file of a given package has been updated.

Usage

```
isVersionUpdated(
  repo = "https://rse.pik-potsdam.de/r/packages/",
  config = loadBuildLibraryConfig()
)
```

Arguments

repo	package repository to determine latest version
config	A configuration defining enforceVersionUpdate. By default the .buildLibrary file is read.

Author(s)

Falk Benke

---

lint *lint*

Description

Check the given files for linter warnings using lintr::lint.

Usage

```
lint(files = getFilesToLint())
```

Arguments

files	A character vector of paths to files that should be checked by the linter. If set to "." the whole package is linted.

Details

For files in the vignettes and tests folder less strict rules are applied, e.g. using ::: usually leads to a linter warning, but not in vignettes/tests. Which linter rules are used depends on ".lintr" config files. `check` creates lintr config files that use `lintrRules`.

Value

A named list, where the names are the paths to the linted files and the values are lists containing the linter warnings.

Author(s)

Pascal Sauer

See Also

`getFilesToLint`, `lintrRules`, `autoFormat`, `lint`

Examples

```
## Not run:
lucode2::lint()

## End(Not run)
```

`lintrRules` *lintrRules*

Description

This function defines the rules to be used by the linter called lintr. `check` creates ".lintr" config files that use this function.

Usage

```
lintrRules(allowUndesirable = FALSE, modification = list())
```

Arguments

`allowUndesirable`	If true it is okay to use undesirable operators (such as "«-") and undesirable (but not deprecated) functions (such as "setwd").
`modification`	A named list mapping linter names to NULL (removes that linter) or a corresponding linter function. Will be applied to the result with utils::modifyList.

Details

To change which linters are applied for a package edit the .lintr file to use the modification argument, see example.

Value

A named list mapping linter names to linter functions.

See Also

`check`, `lint`

Examples

```
## Not run:
lintr::lint_dir(linters = lucode2::lintrRules())

# return usual linters with a different object_name_linter and without the todo_comment_linter
snakeCaseLinter <- lintr::object_name_linter(styles = "snake_case")
lucode2::lintrRules(modification = list(object_name_linter = snakeCaseLinter,
                                        todo_comment_linter = NULL))
```

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

`loadBuildLibraryConfig` *buildLibrary*

Description

Load the build configuration from .buildLibrary. If the file does not exist it is created.

Usage

```
loadBuildLibraryConfig(lib = ".")
```

Arguments

`lib`	Path to the package

Value

The configuration loaded from .buildLibrary as a list.

Author(s)

Jan Philipp Dietrich, Pascal Sauer

See Also

`buildLibrary`

`manipulateConfig` *Replace in File*

Description

Function to set configuration parameters in configuration files (e.g. default.cfg and magpie.gms). This replacement is useful, when using R to manage different model runs at once. Please check your results after replacement!

Usage

```
manipulateConfig(configFile, ...)
```

Arguments

configFile	a character string containing the name of the configuration file, that should be manipulated. Supported file formats are at the moment "gms", "inc", "cfg" (R-syntax), "php", "opt" and "cmd". Other formats are currently not supported
...	Variables, that should be set to new values, e.g. title="test" for default.cfg or s_max_timesteps=10 for magpie.gms

Author(s)

Jan Philipp Dietrich, Markus Bonsch, David Klein

See Also

manipulateFile

Examples

```
## Not run:
manipulateConfig("config/default.cfg", input = "test_new_yields", title = "yihaa", revision = 4.2)
manipulateConfig("magpie.gms", s_max_timesteps = 4, s_use_gdx = -1)

## End(Not run)
```

---

manipulateFile *Replace in File*

Description

Function to replace a specific text string in a text file. Useful to manipulate GAMS sourcecode files.

Usage

```
manipulateFile(file, manipulations, perl = TRUE, ...)
```

Arguments

file	a connection object or a character string describing the file, that should be manipulated.
manipulations	A list of 2 element vectors, containing the search phrase as first element and the replace term as second element.
perl	usually set to TRUE so regular expressions in perl syntax (including backreferencing) can be used. If fixed = TRUE is specified in ..., perl is set to FALSE
...	Further options passed to gsub

Author(s)

Jan Philipp Dietrich

See Also

manipulateConfig

Examples

```
# manipulateFile("example.txt",list(c("bla","blub"),c("a","b")))
```

---

memCheck *Memory usage Check*

Description

Function checks memory usage and shows the biggest objects in the given environment

Usage

```
memCheck(
  order.by = "Size",
  decreasing = TRUE,
  n = NULL,
  envir = parent.frame(),
  gc = TRUE
)
```

Arguments

order.by	Column based on which the data should be sorted
decreasing	Determines whether the values should be in an increasing or decreasing order
n	Limit of number of elements that should be shown. NULL means no limit
envir	Environment which should be analyzed, but default the parent environment relative to this function.
gc	Determines whether the garbage collector should be executed at the end of the function for additional information

Details

This function is based on an idea posted at stack overflow: http://stackoverflow.com/questions/1358003/tricks-to-manage-the-available-memory-in-an-r-session

Author(s)

Jan Philipp Dietrich

Examples

```
## Not run:
memCheck()

## End(Not run)
```

`mergestatistics` *Merge Statistics*

Description

Support function to merge run statistics which have been derived with `runstatistics`

Usage

```
mergestatistics(
  dir = ".",
  file = NULL,
  renew = FALSE,
  quickcheck = FALSE,
  pattern = "*\\.[rR]da",
  removeCols = NULL,
  keepCols = NULL
)
```

Arguments

`dir`	Path to the run statistics repository
`file`	path to an rds-file the data should be written to and from which (if existing) already merged data can be read from
`renew`	if set to TRUE the full data.table will be created again from scratch, if set to FALSE merging will start with the existing file (if it exists) and just add missing entries
`quickcheck`	If active, the function compares last modification dates of repository data and and merged statistics and cancels execution in case that there is no newer file in the data repository (assuming that merge statistics are already complete). This is useful if this function is run frequently and execution time plays a role, but might lead to cases in which the function is not run even if the merge statistics are incomplete.
`pattern`	detection pattern for rda files that should be merged
`removeCols`	vector of columns that will be filtered out
`keepCols`	only these columns will be kept, if NULL all columns will be kept

Value

A data table containing the merged run statistics or NULL in case the data was not recalculated

Author(s)

Jan Philipp Dietrich

`package2readme` *package2readme*

Description

Creates a README.md for a R package.

Usage

```
package2readme(package = ".", add = NULL, logo = NULL)
```

Arguments

package	either the path to the main folder of a package (containing a DESCRIPTION file) or the name of the package
add	a character vector with additions to the README file. Each element of the vector can be either 1) a line of markdown code, 2) a path to a markdown file, or 3) a path to a Rmarkdown file
logo	a character string for a path to a logo file used in the title of the README file

Author(s)

Jan Philipp Dietrich

Examples

```
package2readme("lucode2")
```

`packageInfo` *packageInfo*

Description

Function to print version number and time since last update formatted to standard output. Considers CRAN, the RSE server, and r-universe.

Usage

```
packageInfo(
  package,
 repos = c("https://cran.rstudio.com/", "https://rse.pik-potsdam.de/r/packages/",
    "https://pik-piam.r-universe.dev")
)
```

Arguments

package	Package name
repos	vector of package repositories in which availability of the package should be checked

Author(s)

Jan Philipp Dietrich

---

path *path*

Description

Small function to build a consistent path-string based on folder, filename and filetype. The function makes sure that slashes and the dot for the file ending are set correctly (you can supply your folder name either with or without a tailing slash in it. It does not matter.

Usage

```
path(..., ftype = NULL)
```

Arguments

...	the folders and the file name that should be pasted to a file/folder path
ftype	file type

Value

A string containing the path combined of folder, filename and filetype

Author(s)

Jan Philipp Dietrich

piamPackages *piamPackages*

Description

Fetches the names of packages available on https://pik-piam.r-universe.dev/ui#builds

Usage

```
piamPackages()
```

Value

A character vector of names of packages available on https://pik-piam.r-universe.dev/ui#builds

produce_missing_reports
Produces the reporting mif files where they are missing

Description

For coupled runs: Searches the output folder for all existing run folders, checks which of them are currently running on the cluster, ignores them, checks for the remaining runs whether there is a reporting. Produces the reporting if it is missing. #'

Usage

```
produce_missing_reports(modeldir = "./")
```

Arguments

modeldir	Path to the main folder of REMIND or MAgPIE.

Author(s)

David Klein

readArgs *Read Arguments from command line*

Description

Function reads arguments from command line of the structure value=content and transforms them to R-Values, if they are called as allowed arguments.

Usage

```
readArgs(
  ...,
  .argv = commandArgs(trailingOnly = TRUE),
  .envir = parent.frame(),
  .flags = NULL,
  .silent = FALSE
)
```

Arguments

...	arguments allowed to be read from command line (other values are ignored). Value is set if found on command line input, nothing is done, if value is not found.
.argv	command line arguments, usually read with commandArgs, can be specified for testing purposes
.envir	environment in which the variables should be written (by default the environment from which the function is called)
.flags	named vector with possible command line switches. Element names are short flags used with one dash, corresponding elements the long form including two dashes: c(t = "–test") will interpret "-t" in command line as "–test"
.silent	boolean which allows to suppress status messages

Value

vector of activated flags, if any

Author(s)

Jan Philipp Dietrich, Oliver Richters

See Also

manipulateConfig

Examples

```
# Create an R-file "test.R" with following code:
value1 <- "old"
value2 <- 2
value3 <- "willstaythesame"
flags <- readArgs("value1", "value2", "value4", .flags = c(t = "--test", p = "--parallel"))
message(value1)
message(value2)
message(value3)
if ("--test" %in% flags) {
  message("You are in test mode")
}
if ("--parallel" %in% flags) {
  message("You are in parallel mode")
}

# Open the command line and execute the following code:
# Rscript test.R -t --parallel value1=new value2=3 value3=isnotallowed

# Output:
#
# ### READ COMMAND LINE - ASSIGNED CONFIGURATION ###
# value1 <- new
# value2 <- 3
# value4 not defined
# Flags: --parallel, --test
# ### READ COMMAND LINE - CONFIGURATION END ###
#
# new
# 3
# willstaythesame
# You are in test mode
# You are in parallel mode


### function that reads all allowed arguments from command line ###
```

---

readRuntime *readRuntime*

Description

Reads all runtime information from given experiments. The runtime is given in hours and is the runtime of GAMS.

Usage

```
readRuntime(path, plot = FALSE, types = NULL, coupled = FALSE, outfname = NULL)
```

Arguments

path	Path to a run or a vector of paths.
plot	Logical indicating whether the output should be plotted to a pdf with the name runtime.pdf.
types	A vector of names of different types of runs which should be distinguished by colors. The names have to be part of the folder name in order to allow the function to map the given types to the runs.
coupled	Logical indicating if comparison plots should be added for coupled REMIND and MAgPIE runs. TRUE automatically sets types to c("-rem-","-mag-") and overwrites user defined types
outfname	Optional name of the pdf. If nothing is given the default "runtime" will be used.

Value

A data frame containing the run names and runtime information in hours.

Author(s)

David Klein

---

removeEOF *add EOF comment*

Description

remove EOF text from all files in the code

Usage

```
removeEOF(path = ".", filetypes = c("inc", "prn", "gms"))
```

Arguments

path	path to the main folder of the model
filetypes	file types the function should be applied to

Author(s)

Anastasis Giannousakis

See Also

addEOF

Examples

```
## Not run:
removeEOF()

## End(Not run)
```

`rename_scenario` *Renames the scenariofolder and the scenario contained in it*

Description

Use this function to change the name of a run after it has finished. This function renames the run folder, change the run title in the cfg and in the reporting. This can be useful if the initial name of a run was not meaningful. However, inconsistencies will remain, since the function will NOT rename the scenario in the list file, the gdx, and the results database.

Usage

```
rename_scenario(map, keep_time_stamp = FALSE)
```

Arguments

`map`	Named vector, containing the new scenario names as elements and the corresponding old folder names as the emelents' names.
`keep_time_stamp`	Logical indicating whether timestamp of old folder name should be transferred to new folder name or not (default = FALSE).

Author(s)

David Klein

`rmAllbut` *rmAllbut*

Description

Removes all objects except specified ones from the workspace

Usage

```
rmAllbut(..., list = character(), clean = TRUE)
```

Arguments

...	Objects that should be kept
list	List specifying the objects to be kept. Same as in rm.
clean	Boolean to specify if a gc shall be executed

Details

Helps to clean the workspace. Only objects specified in ... survive. Specify clean =TRUE to really free the memory.

Author(s)

Markus Bonsch

See Also

rm,ls

Examples

```
# Create some objects
a <- 1
b <- 2
c <- 3
# show them
ls()
# delete all but b and c
rmAllbut(b, c)
ls()
# delete all but b
test <- "b"
rmAllbut(list = test)
ls()
```

---

runstatistics *Run Statistics*

Description

Support function to collect run statistics.

Usage

```
runstatistics(file = "runstatistics.Rda", overwrite = TRUE, submit = NULL, ...)
```

Arguments

file	file name the statistics are/should be stored
overwrite	boolean deciding whether entries should be overwritten, if already existing in the file. If set to FALSE an error will be thrown in case that an overwrite is attempted.
submit	path to a folder the run statistics should be submitted to. As soon as the path is set the data will be submitted, so please only set the path as soon as the run statistics are complete
...	entries that should be added to the run statistics file. Standard entries are: model, config, runtime, user, date, modelstat, version_management, revision and status.

Value

An invisible list containing run statistics as stored in the given file

Author(s)

Jan Philipp Dietrich

Examples

```
f <- tempfile()
 runstatistics(file = f, user = Sys.info()[["user"]])
 print(runstatistics(file = f))
 runstatistics(file = f, submit = tempdir())
```

---

sendmail *sendmail*

Description

A function that sends an automatic email with each push to a gitlab repo

Usage

```
sendmail(
  path = NULL,
  gitrepo,
  file,
  commitmessage,
  remote = FALSE,
  reset = FALSE
)
```

Arguments

path	path to the clone of the gitlab repo (can be NULL)
gitrepo	if no path is given, the gitlab repo is needed
file	absolute path to the file to be committed
commitmessage	the commit message (appears in the subject line of the email that will be sent)
remote	whether communication with a remote is needed
reset	whether a reset of a local copy is wanted

Author(s)

Anastasis Giannousakis

---

setup_info *Setup Info*

Description

Returns a list with information about the current session (session info, used library paths and installed libraries)

Usage

```
setup_info()
```

Value

A list with information about the current session and the currently used R setup.

Author(s)

Jan Philipp Dietrich

Examples

```
setup_info()
```

setwd2 *setwd2*

Description

Mini function that allows you to set a directory based on a readline input. Very useful for Windows users, as it replaces backslashes by slashes.

Usage

```
setwd2(return_only = FALSE)
```

Arguments

return_only	if TRUE, the path is not changed, but the clipboard path is returned as string.

Value

if return_only=FALSE: Nothing, but the working directory is set to. Otherwise: no working directory returned, but path transformed.

Author(s)

Benjamin Leon Bodirsky

snakeToCamel *snakeToCamel*

Description

Convenience function to rename variables in an R file from snake to camel case.

Usage

```
snakeToCamel(pathToFile, ask = TRUE)
```

Arguments

pathToFile	Path to the R source file where variables should be renamed.
ask	If TRUE (default) ask before renaming a variable, otherwise always assume "yes" as answer.

SystemCommandAvailable *System Command Available*

Description

Checks whether a system command is available (does not return an error), or not

Usage

```
SystemCommandAvailable(command)
```

Arguments

command	System command to be tested

Value

Boolean indicating whether the command is available, or not

Author(s)

Jan Philipp Dietrich

Examples

```
SystemCommandAvailable("ls")
```

testPackage *Test package*

Description

Installs a package in a temporary library and loads that library on top of the existing one

Usage

```
testPackage(repo, tmpLib = tempdir(), ...)
```

Arguments

repo	GitHub repository to install the package from
tmpLib	temporary library directory where the package should be installed
...	additional arguments forwarded to devtools::install_github

Author(s)

Jan Philipp Dietrich

Examples

```
## Not run:
testPackage("git@github.com:pik-piam/lucode2")

## End(Not run)
```

`updateRepo` *Update package repository*

Description

Function to update an package repository. Run this function on a folder which contains packages sources as subfolders. Packages should be managed via git in order to be updated properly. To add a new package to the repo just checkout/clone it into this folder. The function will automatically detect the new package and add it.

Usage

```
updateRepo(
  path = ".",
  check = TRUE,
  forceRebuild = FALSE,
  clean = TRUE,
  skipFolders = "Archive",
  repoUrl = "https://rse.pik-potsdam.de/r/packages"
)
```

Arguments

`path`	Path to the repository
`check`	Boolean deciding whether package must have been checked or not in order to be distributed
`forceRebuild`	Option to rebuild all packages from source
`clean`	Option to clean repos before updating/pulling to avoid merge conflicts
`skipFolders`	Which folders/packages should not be built.
`repoUrl`	Url of the package repo. Will be added to DESCRIPTION files in the Repository field.

Author(s)

Jan Philipp Dietrich, Pascal Sauer

See Also

buildLibrary

---

validationkey *validationkey*

Description

Support function which creates a key out of a version date combination

Usage

```
validationkey(version, date)
```

Arguments

version	Version number of the package
date	Date of the package

Details

This function is used in buildLibrary to offer the package publication server an option to check whether the package has been properly and successfully (no errors/warnings/notes) checked before its commit.

The calculated key is not safe and can easily be reproduced, but should be complicated enough to encourage users running the buildLibrary check properly.

Author(s)

Jan Philipp Dietrich

See Also

buildLibrary

validkey *validkey*

Description

Support function which validates a key out of a version date combination

Usage

```
validkey(package = ".", stopIfInvalid = FALSE)
```

Arguments

package	Path to the package
stopIfInvalid	logical; whether to stop if the key is invalid.

Details

This function is used to check whether buildLibrary has been run properly and without problems or not

Value

list with version, date and result of validation test

Author(s)

Jan Philipp Dietrich

See Also

buildLibrary

verifyCheck *verifyCheck*

Description

Run R CMD check completely without stopping. Then stop on errors, or unaccepted warnings and notes.

Usage

```
verifyCheck(cran, acceptedWarnings, acceptedNotes)
```

Arguments

cran	Passed to devtools::check
acceptedWarnings	A character vector of regular expressions. A warning will result in an error unless it matches one of these regular expressions.
acceptedNotes	A character vector of regular expressions. A note will result in an error unless it matches one of these regular expressions.

Author(s)

Pascal Sauer

---

verifyLinter *verifyLinter*

Description

Run linter and stop on linter warning unless linter warnings are allowed.

Usage

```
verifyLinter(allowLinterWarnings = FALSE)
```

Arguments

allowLinterWarnings	If FALSE (the default) will stop on linter warnings.

Author(s)

Pascal Sauer

---

verifyTests *verifyTests*

Description

Run tests and stop on error or unaccepted warning.

Usage

```
verifyTests(acceptedWarnings)
```

Arguments

`acceptedWarnings`
: A character vector of regular expressions. A warning will result in an error unless it matches one of these regular expressions.

Author(s)

Pascal Sauer

Index