Package 'repo'

October 14, 2022

Encoding UTF-8

Type Package

Title A Data-Centered Data Flow Manager

Version 2.1.5

Date 2020-02-05

Author Francesco Napolitano <franapoli@gmail.com>

Maintainer Francesco Napolitano <franapoli@gmail.com>

Description A data manager meant to avoid manual storage/retrieval of data to/from the file system. It builds one (or more) centralized repository where R objects are stored with rich annotations, including corresponding code chunks, and easily searched and retrieved. See Napolitano (2017) <doi:10.1037/a0028240> for further information.

License GPL-3

Imports digest, tools

Suggests igraph, knitr, shiny, testthat

VignetteBuilder knitr

RoxygenNote 7.0.2

NeedsCompilation no

Repository CRAN

Date/Publication 2020-02-08 16:20:02 UTC

R topics documented:

`repo-package` *Repo: The Data-centered Data Flow Manager*

Description

Repo: The Data-centered Data Flow Manager

Details

The Repo package is meant to help with the management of R data files. It builds one (or more) centralized repository where R objects are stored together with corresponding annotations, tags, dependency notes, provenance traces. It also provides navigation tools to easily locate and load previously stored resources.

Create a new repository with `rp <- repo_open()`.

Given the object rp of class repo, the repo command foo must be called like this: rp$foo(). However, the public name of foo will be repo_foo, and this name must be used to get help (?repo_foo).

For a complete list of functions, use library(help = "repo").

Author(s)

Francesco Napolitano <franapoli@gmail.com>

---

repo_attach *Create a new item from an existing file.*

Description

Create a new item from an existing file.

Usage

```
repo_attach(
  filepath,
  description = NULL,
  tags = NULL,
  prj = NULL,
  src = NULL,
  chunk = basename(filepath),
  replace = F,
  to = NULL,
  URL = NULL
)
```

Arguments

filepath	The path to the file to be stored in the repo.
description	A character description of the item.
tags	A list of tags to sort the item. Tags are useful for selecting sets of items and run bulk actions.
prj	The name of a project item in the repository (see project). Default is no associated project item.
src	The name of the item that produced the stored object. Usually a previously attached source code file.
chunk	The name of the code chunk within src that is responsible for building the item. Set to name by default. See build.
replace	If the item exists, overwrite the specified fields.
to	An existing item name to attach the file to.
URL	A URL where the item contents con be downloaded from.

Value

Used for side effects.

Examples

```
rp_path <- file.path(tempdir(), "example_repo")
rp <- repo_open(rp_path, TRUE)

## Not run:
## Creating a PDF file with a figure.
pdf("afigure.pdf")
## Drawing a random plot in the figure
plot(runif(100), runif(100))
dev.off()
## Attaching the PDF file to the repo
rp$attach("afigure.pdf", "A plot of random numbers", "repo_sys")
## don't need the PDF file anymore
file.remove("afigure.pdf")
## Opening the stored PDF with Evince document viewer
rp$sys("afigure.pdf", "evince")

## End(Not run)

## wiping temporary repo
unlink(rp_path, TRUE)
```

---

repo_attr *Get item attribute.*

Description

Get item attribute.

Usage

```
repo_attr(name, attrib)
```

Arguments

name	An item name.
attrib	An attribute name (currently can be only "path").

Value

The item's attribute value.

See Also

repo_entries, repo_get

Examples

```
rp_path <- file.path(tempdir(), "example_repo")
rp <- repo_open(rp_path, TRUE)
rp$put(1, "item1", "Sample item 1", "tag1")
print(rp$attr("item1", "path"))

## wiping temporary repo
unlink(rp_path, TRUE)
```

`repo_build` *Builds a resource using the associated code chunk*

Description

In order to be `build`able, a repository item must have an associated source file and code chunk.

Usage

```
repo_build(
  name,
  src = NULL,
  recursive = T,
  force = F,
  env = parent.frame(),
  built = list()
)
```

Arguments

`name`	Name of an item in the repo.
`src`	Path to a source file containing the code block associated with the resource. Not necessary if `name` is already in the repository and has an associated source item.
`recursive`	Build dependencies not already in the repo recursively (T by default).
`force`	Re-build dependencies recursively even if already in the repo (F by default).
`env`	Environment in which to run the code chunk associated with the item to build. Parent environment by default.
`built`	A list of items already built used for recursion (not meant to be passed directly).

Details

Code chunks are defined as in the following example: “‘ ## chunk "item 1" x <- code_to_make_x() rp$put(x, "item 1") ## “‘

‘item 1‘ must be associated to the source (‘src‘ parameter of ‘put‘) containing the chunk code.

Value

Nothing, used for side effects.

repo_bulkedit *Edit all items info using a text file.*

Description

Edit all items info using a text file.

Usage

```
repo_bulkedit(outfile = NULL, infile = NULL)
```

Arguments

outfile	Name of a file to put entries data to.
infile	Name of a file to read entries data from.

Details

Exactly one of `outfile` or `infile` must be supplied. All repository entry fields are copied to a tab-separated file when using the `outfile` parameter. All repo entries are updated reading from `infile` when the `infile` parameter is used. Within the TAGS field, tags must be comma-separated. The system writes a checksum to the `outfile` that prevents from using it as `infile` if repo has changed in the meantime.

Value

Used for side effects.

See Also

repo_set

Examples

```
rp_path <- file.path(tempdir(), "example_repo")
rp <- repo_open(rp_path, TRUE)
rp$put(1, "item1", "Sample item 1", c("tag1", "tag2"))

items_data_file <- tempfile()
rp$bulkedit(items_data_file)
## Manually edit items_data_file, then update items:
rp$bulkedit(infile=items_data_file)

## wiping temporary repo
unlink(rp_path, TRUE)
```

repo_check *Check repository integrity.*

Description

Checks that all indexed data are present in the repository root, that files are not corrupt and that no unindexed files are present.

Usage

```
repo_check()
```

Details

Every time the object associated to an item is stored, an MD5 checksum is saved to the repository index. check will use those to verify that the object was not changed by anything other than Repo itself.

Value

Used for side effects.

Examples

```
## Repository creation
rp_path <- file.path(tempdir(), "example_repo")
rp <- repo_open(rp_path, TRUE)

rp_path <- file.path(tempdir(), "example_repo")
rp <- repo_open(rp_path, TRUE)
rp$put(0, "item1", "A sample item", "repo_check")
rp$check()

## wiping temporary repo
unlink(rp_path, TRUE)
```

repo_chunk *Shows code chunk associated with an item*

Description

Shows code chunk associated with an item

Usage

```
repo_chunk(name)
```

Arguments

name	Item name.

Value

List of lines of code, invisibly.

---

repo_copy *Copy items to another repository*

Description

Copies an object file from one repository to another and creates a new entry in the index of the destination repository. Supports tags and multiple names.

Usage

```
repo_copy(destrepo, name, tags = NULL, replace = F, confirm = T)
```

Arguments

destrepo	An object of class repo (will copy to it)
name	The name (or list of names) of the item/s to copy
tags	If not NULL, copy all items matching tags. NULL by default.
replace	What to do if item exists in destination repo (see put). F by default.
confirm	If F, don't ask for confirmation when multiple items are involved. F by default.

Value

Used for side effects.

Examples

```
## Repository creation
rp_path1 <- file.path(tempdir(), "example_repo1")

rp1 <- repo_open(rp_path1, TRUE)
rp1$put(0, "item1", "A sample item", "tag1")
rp_path2 <- file.path(tempdir(), "example_repo2")
rp2 <- repo_open(rp_path2, TRUE)
rp1$copy(rp2, "item1")

## wiping temporary repo
unlink(rp_path1, TRUE)
unlink(rp_path2, TRUE)
```

repo_cpanel *Open a visual interface to the repo*

Description

Opens a browser window with a Shiny interface to a repo. The interface is preliminary and has some exploration features together with a "Load into workspace" button for a selected item.

Usage

```
repo_cpanel(reporoot = NULL, env = globalenv())
```

Arguments

reporoot	An object of class repo. Can be NULL like for repo_open.
env	Environment to export variables to. Defaults to globalenv.

Value

Used for side effects.

repo_dependencies *Build and/or plots a dependency graph*

Description

Creates a weighted adjacency matrix, in which (i,j) = x means that item i is in relation x with item j. The resulting graph is plotted.

Usage

```
repo_dependencies(
  tags = NULL,
  tagfun = "OR",
  depends = T,
  attached = T,
  generated = T,
  plot = T,
  ...
)
```

Arguments

tags	Only show nodes matching tags
tagfun	Function specifying how to match tags (by default "OR": match any of tags).
depends	If TRUE, show "depends on" edges.
attached	If TRUE, show "attached to" edges.
generated	If TRUE, show "generated by" edges.
plot	If TRUE (default), plot the dependency graph.
...	Other parameters passed to the plot.igraph function.

Details

The relation between any two items i and j can have values 1, 2 or 3, respectively meaning:

- depends on: to build item i, item j was necessary.
- attached to: item i is an attachment item and is attached to item j.
- generated by: item i has been generated by item j. Item j is usually an attachment containing the source code that generated item i.

Value

Adjacency matrix representing the graph, with edges labeled 1, 2, 3 corresponding to "depends", "attached" and "generated" respectively.

Examples

```
## Repository creation
rp_path <- file.path(tempdir(), "example_repo")
rp <- repo_open(rp_path, TRUE)

## Producing some irrelevant data
data1 <- 1:10
data2 <- data1 * 2
data3 <- data1 + data2

## Putting the data in the database, specifying dependencies
rp$put(data1, "item1", "First item",
    "repo_dependencies")
rp$put(data2, "item2", "Item dependent on item1",
    "repo_dependencies", depends="item1")
rp$put(data3, "item3", "Item dependent on item1 and item2",
    "repo_dependencies", depends=c("item1", "item2"))

## Creating a temporary plot and attaching it
fpath <- file.path(rp$root(), "temp.pdf")
pdf(fpath)
plot(data1)
dev.off()
rp$attach(fpath, "visualization of item1", "plot",
```

```
   to="item1")

## Obtaining the dependency matrix
depmat <- rp$dependencies(plot=FALSE)
print(depmat)
## The matrix can be plotted as a graph (requires igraph package)
rp$dependencies()
## The following hides "generated" edges
rp$dependencies(generated=FALSE)

## wiping temporary repo
unlink(rp_path, TRUE)
```

`repo_depends` *Returns item's dependencies*

Description

Returns item's dependencies

Usage

```
repo_depends(name)
```

Arguments

`name`	The name of a repository item.

Value

The items on which the input item depends.

`repo_entries` *Low-level list of item entries.*

Description

Low-level list of item entries.

Usage

```
repo_entries()
```

Value

A detailed list of item entries.

Examples

```
rp_path <- file.path(tempdir(), "example_repo")
rp <- repo_open(rp_path, TRUE)
rp$put(1, "item1", "Sample item 1", "entries")
rp$put(2, "item2", "Sample item 2", "entries")
rp$put(3, "item3", "Sample item 3", "entries")
print(rp$entries())

## wiping temporary repo
unlink(rp_path, TRUE)
```

---

repo_export *Export* repo *items to RDS file.*

Description

Export repo items to RDS file.

Usage

```
repo_export(name, where = ".", tags = NULL, askconfirm = T)
```

Arguments

name	Name (or list of names) of the item/s to export.
where	Destination directory
tags	List of tags: all items tagged with all the tags in the list will be exported.
askconfirm	If T ask confirmation when exporting multiple items.

Value

TRUE on success, FALSE otherwise.

Examples

```
rp_path <- file.path(tempdir(), "example_repo")
rp <- repo_open(rp_path, TRUE)
rp$put(1, "item1", "Sample item 1", "export")
rp$export("item1", tempdir()) # creates item1.RDS in a tempdir

## wiping temporary repo
unlink(rp_path, TRUE)
```

repo_find *Match items by matching any field*

Description

Match items by matching any field

Usage

```
repo_find(what, all = F, show = "ds")
```

Arguments

what	Character to be matched against any field (see Details).
all	Show also items tagged with "hide".
show	Select columns to show.

Details

This function actually calls print specifying the find parameters. The find parameter can be any character string to be matched against any item field, including string-converted size (like "10x3").

Value

Used for side effects.

Examples

```
rp_path <- file.path(tempdir(), "example_repo")

rp <- repo_open(rp_path, TRUE)
rp$put(1, "item1", "Sample item 1", c("tag1", "tag2"))
rp$put(2, "item2", "Sample item 2", c("tag1", "hide"))
rp$put(3, "item3", "Sample item 3", c("tag2", "tag3"))
rp$print()
rp$find("tEm2")
rp$find("ag2", show="t")

## wiping the temp repo
unlink(rp_path, TRUE)
```

repo_get *Retrieve an item from the repo.*

Description

Retrieve an item from the repo.

Usage

```
repo_get(name, enableSuggestions = T)
```

Arguments

name	An item's name.
enableSuggestions	If set to TRUE (default), enables some checks on name that are meant to gracefully handle errors and provide suggestions of similar names. If FALSE, the execution will be significantly faster in large repositories.

Value

The previously stored object, or its file system path for attachments.

Examples

```
rp_path <- file.path(tempdir(), "example_repo")
rp <- repo_open(rp_path, TRUE)
rp$put(1, "item1", "Sample item 1", "get")
print(rp$get("item1"))

## wiping temporary repo
unlink(rp_path, TRUE)
```

repo_handlers *Provides simplified access to repository items.*

Description

Creates a list of functions, each one associated with a repository item, that can be used to access items directly.

Usage

```
repo_handlers()
```

Details

Repository handlers are functions associated with items. As opposed to item names, they can take advantage of IDE auto-completion features and do not require quotation marks. A handler to the repo object itself is provided in the list.

Value

A list of functions.

Examples

```
## Repository creation
rp_path <- file.path(tempdir(), "example_repo")
rp <- repo_open(rp_path, TRUE)

## Putting some irrelevant data
rp$put(1, "item1", "Sample item 1", "repo_handlers")
rp$put(2, "item2", "Sample item 2", "repo_handlers")

## Getting item handlers
h <- rp$handlers()
## handlers have the same names as the items in the repo (and they
## include an handler to the repo itself).
names(h)

## Without arguments, function "item1" loads item named "item1".
i1 <- h$item1()

## Arguments can be used to call other repo functions on the item.
h$item1("info")

## After putting new data, the handlers must be refreshed.
rp$put(3, "item3", "Sample item 3", "repo_handlers")
h <- rp$handlers()
names(h)

## wiping temporary repo
unlink(rp_path, TRUE)
```

---

repo_has *Check whether a repository has an item*

Description

Check whether a repository has an item

Usage

```
repo_has(name)
```

Arguments

name	Item name.

Value

TRUE if name is in the repository, FALSE otherwise.

---

repo_info *Provides detailed information about an item.*

Description

Provides detailed information about an item.

Usage

```
repo_info(name = NULL, tags = NULL)
```

Arguments

name	Item name (or list of names). If both name and tags are NULL, information about the whole repo will be provided.
tags	List of tags: info will run on all items matching the tag list.

Value

Used for side effects.

Examples

```
rp_path <- file.path(tempdir(), "example_repo")
rp <- repo_open(rp_path, TRUE)
rp$put(1, "item1", "Sample item 1", "info")
rp$info("item1")

## wiping temporary repo
unlink(rp_path, TRUE)
```

repo_lazydo *Run expression with cache.*

Description

lazydo searches the repo for previous execution of an expression. If a previous execution is found, the result is loaded and returned. Otherwise, the expression is executed and the result stashed.

Usage

```
repo_lazydo(expr, force = F, env = parent.frame())
```

Arguments

expr	An object of class expression (the code to run).
force	If TRUE, execute expr anyway
env	Environment for expr, defaults to parent.

Details

The expression results are stashed as usual. The name of the resource is obtained by digesting the expression, so it will look like an MD5 string in the repo. Note that the expression, and not its result, will uniquely identify the item in the repo.

The new item is automatically tagged with "stash", "hide" and "lazydo".

Value

Results of the expression (either loaded or computed on the fly).

See Also

repo_stash, repo_put

Examples

```
rp_path <- file.path(tempdir(), "example_repo")
rp <- repo_open(rp_path, TRUE)


## First run
system.time(rp$lazydo(
    {
        Sys.sleep(1/10)
        x <- 10
    }
))

## lazydo is building resource from code.
```

```
## Cached item name is: f3c27f11f99dce20919976701d921c62
##   user  system elapsed
##  0.004   0.000   0.108

## Second run
system.time(rp$lazydo(
    {
        Sys.sleep(1/10)
        x <- 10
    }
))

## lazydo found precomputed resource.
##   user  system elapsed
##  0.001   0.000   0.001


## The item's name in the repo can be obtained as the name of the
## last item added:

l <- length(rp$entries())
resname <- rp$entries()[[l]]$name
cat(rp$entries()[[l]]$description)
## {
##    Sys.sleep(1/10)
##    x <- 10
## }
rp$rm(resname) ## single cached item cleared

## wiping temporary repo
unlink(rp_path, TRUE)
```

---

repo_load *Loads an item to current workspace*

Description

Like repo_get, returns the contents of a stored item. But, unlike repo_get, loads it to the current namespace.

Usage

```
repo_load(names, overwrite_existing = F, env = parent.frame())
```

Arguments

names	List or vector of repository item names.
overwrite_existing	Overwrite an existing variable by the same name in the current workspace. If F (defaults) throws an error.
env	Environment to load the variable into (parent environment by default).

Value

Nothing, used for side effects.

`repo_open` *Open an existing repository or create a new one.*

Description

If a repository does not exist at the specified location, creates a directory and stores the repository index in it. If a repository exists, the index is loaded and a `repo` object is built.

Arguments

`root`	Path to store data in. Defaults to "~/.R_repo".
`force`	Don't ask for confirmation.

Value

An object of class `repo`.

Examples

```
## Creates a new repository in a temporary directory without asking for
## confirmation.
rp_path <- file.path(tempdir(), "example_repo")
rp <- repo_open(rp_path, TRUE)
rp$put(0, "zero", "a random item", "a_tag")
rp$info()
## wiping temporary repo
unlink(rp_path, TRUE)
```

`repo_options` *Set repository-wide options*

Description

Set repository-wide options

Usage

```
repo_options(...)
```

Arguments

`...`	options to set

Value

if optional parameters are not passed, the current options are returned

repo_pies *Plots a pie chart of repository contents*

Description

The pie chart shows all repository items as pie slices of size proportional to the item sizes on disk. Items with size smaller then 5

Usage

```
repo_pies(...)
```

Arguments

...	Other parameters passed to the pie function.

Value

Used for side effects.

Examples

```
## Repository creation
rp_path <- file.path(tempdir(), "example_repo")
rp <- repo_open(rp_path, TRUE)

## Producing some irrelevant data of different sizes
data1 <- 1:10
data2 <- 1:length(data(1))*2
data3 <- 1:length(data(1))*3

## Putting the data in the database, specifying dependencies
rp$put(data1, "item1", "First item", "repo_pies")
rp$put(data2, "item2", "Second item", "repo_pies")
rp$put(data3, "item3", "Third item", "repo_pies")

## Showing the pie chart
rp$pies()

## wiping temporary repo
unlink(rp_path, TRUE)
```

repo_print *Show a summary of the repository contents.*

Description

Show a summary of the repository contents.

Usage

```
repo_print(tags = NULL, tagfun = "OR", find = NULL, all = F, show = "ds")
```

Arguments

tags	A list of character tags. Only items matching all the tags will be shown.
tagfun	How to combine tags (see Details).
find	Character to match any filed (see Details).
all	Show also items tagged with "hide".
show	Select columns to show.

Details

The tagfun param specifies how to combine multiple tags when matching items. It can be either a character or a function. As a character, it can be one of OR, AND or NOT to specify that one, all or none of the tags must be matched, respectively. If it is a function, it must take two tag vectors, the first of which corresponds to tags, and return TRUE for a match, FALSE otherwise.

The find param can be any character string to be matched against any item field, including string-converted size (like "10x3").

Value

Used for side effects.

Examples

```
rp_path <- file.path(tempdir(), "example_repo")
rp <- repo_open(rp_path, TRUE)
rp$put(1, "item1", "Sample item 1", c("tag1", "tag2"))
rp$put(2, "item2", "Sample item 2", c("tag1", "hide"))
rp$put(3, "item3", "Sample item 3", c("tag2", "tag3"))
rp$print()
rp$print(all=TRUE)
rp$print(show="tds", all=TRUE)
rp$print(show="tds", all=TRUE, tags="tag1")
## wiping the temp repo
unlink(rp_path, TRUE)

## wiping temporary repo
unlink(rp_path, TRUE)
```

repo_project *Defines and* put-*s a* project *item.*

Description

A project item is a special item containing session information, including package dependencies. Every time a new item is stored in the repository, it will automatically be assigned to the current project, if one has been defined, and session information will be updated.

Usage

```
repo_project(name, description, replace = T)
```

Arguments

name	character containing the name of the project
description	character containing a longer description of the project
replace	logical, if T then an existing project item by the same name will be overwritten.

Value

Used for side effects.

repo_pull *Download item remote content*

Description

Download item remote content

Usage

```
repo_pull(name, replace = F)
```

Arguments

name	Name of the existing item that will be updated.
replace	If TRUE, existing item's object is overwritten.

Details

Repo index files can be used as pointers to remote data. The pull function will download the actual data from the Internet, including regular items or attachment. Another use of the URL item's parameter is to attach a remote resource without downloading it.

Value

Used for side effects.

Examples

```
## Repository creation
rp_path <- file.path(tempdir(), "example_repo")
rp <- repo_open(rp_path, TRUE)
remote_URL <- paste0("https://github.com/franapoli/repo/blob/",
                     "untested/inst/remote_sample.RDS?raw=true")

## The following item will have remote source
rp$put("Local content", "item1", "Sample item 1", "tag",
         URL = remote_URL)
print(rp$get("item1"))

## suppressWarnings(try(rp$pull("item1"), TRUE))
 tryCatch(rp$pull("item1"),
         error = function(e)
             message("There were warnings whle accessing remote content"),
         warning = function(w)
             message("Could not download remote content")
         )
print(rp$get("item1"))

## wiping temporary repo
unlink(rp_path, TRUE)
```

---

repo_put *Create a new item in the repository.*

Description

Given an R object, stores it to an RDS file in the repo root and add an associated item to the repo index, including object name, description, tags and more.

Usage

```
repo_put(
  obj,
  name = NULL,
  description = NULL,
  tags = NULL,
  prj = NULL,
  src = NULL,
  chunk = name,
  depends = NULL,
  replace = F,
```

```
  asattach = F,
  to = NULL,
  addversion = F,
  URL = NULL,
  checkRelations = T
)
```

Arguments

obj	An R object to store in the repo.
name	A character identifier for the new item. If NULL, the name of the obj variable will be used.
description	A character description of the item.
tags	A list of tags to sort the item. Tags are useful for selecting sets of items and run bulk actions.
prj	The name of a project item in the repository (see project). Default is no associated project item.
src	Name of an existing item to be annotated as the "generator" of the new item. Usually it is an attachment item containing the source code that generated the new item. Default is NULL.
chunk	The name of the code chunk within src that is responsible for building the item. Set to name by default. See build.
depends	Character vector: items that depend on this item. Default is NULL.
replace	One of: V, F, "addversion" to define behavior when an item by the same name exists. If V, overwrite it. If F stop with an error. If "addversion" the new item is stored as a new version and the old item is renamed by appending a "#N" suffix. Default is F.
asattach	Specifies that the item is to be treated as an attachment (see attach). Default is F.
to	Vector of character. Specifies which item this item is attached to. Default is NULL.
addversion	Deprecated, use the replace parameter instead.
URL	Remote URL where the pull function expects to download actual item data from. Default is NULL.
checkRelations	Check if items referenced by this item exist. Default is T.

Details

The item name can be any string, however it should be a concise identifier, possibly without special character (could become mandatory soon). Some tags have a special meaning, like "hide" (do not show the item by default), "attachment" (the item is an attachment - this should never be set manually), "stash" (the item is a stashed item, makes the item over-writable by other "stash" items by default).

Value

Used for side effects.

See Also

get, set, attach, info

Examples

```
## Repository creation
rp_path <- file.path(tempdir(), "example_repo")
rp <- repo_open(rp_path, TRUE)

## Producing some irrelevant data
data1 <- 1:10
data2 <- data1 * 2
data3 <- data1 / 2

## Putting the data in the database, specifying dependencies
rp$put(
    obj = data1,
    name = "item1",
    description = "First item",
    tags = c("repo_put", "a_random_tag"),
    )
rp$put(data2, "item2", "Item dependent on item1",
    "repo_dependencies", depends="item1")
rp$put(data3, "item3", "Item dependent on item1 and item2",
    "repo_dependencies", depends=c("item1", "item2"))

print(rp)

## Creating another version of item1
data1.2 <- data1 + runif(10)
rp$put(data1.2, name = "item1", "First item with additional noise",
    tags = c("repo_put", "a_random_tag"), replace="addversion")
print(rp, all=TRUE)
rp$info("item1#1")

## wiping temporary repo
unlink(rp_path, TRUE)
```

---

repo_related *Finds all items related to a set of item*

Description

Relations are defined as in the dependency graph.

Usage

```
repo_related(names, type = "all", excludeseed = F)
```

Arguments

names	A character vector of item names.
type	Can be one of "all", "to", "from". "to" recursively finds items that names is attached to. "from" recursively finds items that names depends on or is generated by. "all" finds both (connected components including names.
excludeseed	logical. If set to FALSE names will be not included in the output list.

Value

A character vector of item names.

See Also

dependencies

---

repo_rm *Remove item from the repo (and the disk).*

Description

Remove item from the repo (and the disk).

Usage

```
repo_rm(name = NULL, tags = NULL, force = F)
```

Arguments

name	An item's name.
tags	A list of tags: all items matching the list will be removed.
force	Don't ask for confirmation.

Value

Used for side effects.

Examples

```
rp_path <- file.path(tempdir(), "example_repo")
rp <- repo_open(rp_path, TRUE)
rp$put(1, "item1", "Sample item 1", "info")
rp$put(2, "item2", "Sample item 2", "info")
print(rp)
rp$rm("item1")
print(rp)

## wiping temporary repo
unlink(rp_path, TRUE)
```

---

repo_root *Show path to repo root*

Description

Show path to repo root

Usage

```
repo_root()
```

Value

character containing the path to the root of the repo.

Examples

```
rp_path <- file.path(tempdir(), "example_repo")
rp <- repo_open(rp_path, TRUE)
print(rp$root())

## wiping temporary repo
unlink(rp_path, TRUE)
```

---

repo_set *Edit an existing item.*

Description

Edit an existing item.

Usage

```
repo_set(
  name,
  obj = NULL,
  newname = NULL,
  description = NULL,
  tags = NULL,
  prj = NULL,
  src = NULL,
  chunk = NULL,
  depends = NULL,
  addtags = NULL,
  URL = NULL,
  buildURL = NULL
)
```

Arguments

name	An item name.
obj	An R object to replace the one currently associated with the item.
newname	Newname of the item.
description	Item's description.
tags	New item's tags as a list of character.
prj	New item's project as a list of character.
src	New item's provenance as a list of character.
chunk	New item's chunk name.
depends	List of item names indicating dependencies.
addtags	Tags to be added to current item's tags. Can not be used together with the parameter "tags".
URL	A character containing an URL where the item is supposed to be downloaded from.
buildURL	A character containing a base URL that is completed by postfixing the item's relative path. Useful to upload repositories online and make their items downloadable. The item's current URL is overwritten.

Value

Used for side effects.

See Also

repo_put

Examples

```
rp_path <- file.path(tempdir(), "example_repo")
rp <- repo_open(rp_path, TRUE)
rp$put(1, "item1", "Sample item 1", c("tag1", "tag2"))
rp$set("item1", obj=2)
print(rp$get("item1"))
rp$set("item1", description="Modified description", tags="new_tag_set")
rp$info("item1")

## wiping temporary repo
unlink(rp_path, TRUE)
```

`repo_stash` *Quickly store temporary data*

Description

A very simplified call to put that only requires to specify a variable name.

Usage

```
repo_stash(object, rename = deparse(substitute(object)))
```

Arguments

`object`	The object to store in the repo.
`rename`	An optional character containing the new name for the item. Otherwise the name of object is used as item's name.

Details

The `name` parameter is used to search the parent (or a different specified) environment for the actual object to store. Then it is also used as the item name. The reserved tags "stash" and "hide" are set. In case a stashed item by the same name already exists, it is automatically overwritten. In case a non-stashed item by the same name already exists, an error is raised. A different name can be specified through the rename parameter in such cases.

Value

Used for side effects.

See Also

repo_put, repo_lazydo

Examples

```
## Not run:
rp_path <- file.path(tempdir(), "example_repo")
rp <- repo_open(rp_path, TRUE)
tempdata <- runif(10)
rp$stash(tempdata)
rp$info("tempdata")

## wiping temporary repo
unlink(rp_path, TRUE)

## End(Not run)
```

`repo_stashclear` *Remove all stashed data*

Description

Remove all stashed data

Usage

```
repo_stashclear(force = F)
```

Arguments

`force`	If TRUE, no confirmation is asked.

Value

Used for side effects.

See Also

repo_rm, repo_stash

Examples

```
## Not run:
rp_path <- file.path(tempdir(), "example_repo")
rp <- repo_open(rp_path, TRUE)
tempdata <- runif(10)
rp$stash("tempdata")
rp$print(all=TRUE)
rp$stashclear(TRUE)

## wiping temporary repo
unlink(rp_path, TRUE)

## End(Not run)
```

repo_sys *Run system call on an item*

Description

Runs a system command passing as parameter the file name containing the object associated with an item.

Usage

```
repo_sys(name, command)
```

Arguments

name	Name of a repo item. The path to the file that contains the item will be passed to the system program.
command	System command

Value

Used for side effects.

Examples

```
## Repository creation
rp_path <- file.path(tempdir(), "example_repo")
rp <- repo_open(rp_path, TRUE)

## Creating a PDF file with a figure.
pdffile <- file.path(rp_path, "afigure.pdf")
pdf(pdffile)
plot(runif(30), runif(30))
dev.off()

## Attaching the PDF file to the repo
rp$attach(pdffile, "A plot of random numbers", "repo_sys")
## don't need the original PDF file anymore
file.remove(pdffile)

## Opening the stored PDF with Evince document viewer
## Not run:
rp$sys("afigure.pdf", "evince")

## End(Not run)

## wiping temporary repo
unlink(rp_path, TRUE)
```

repo_tag *Add tags to an item.*

Description

Add tags to an item.

Usage

```
repo_tag(name = NULL, newtags, tags = NULL)
```

Arguments

name	An item name.
newtags	A list of tags that will be added to the item's tag list.
tags	A list of tags: newtags will be added to all items matching the list.

Value

Used for side effects.

See Also

repo_untag, repo_set

Examples

```
rp_path <- file.path(tempdir(), "example_repo")
rp <- repo_open(rp_path, TRUE)
rp$put(1, "item1", "Sample item 1", "tag1")
rp$print(show="t")
rp$tag("item1", "tag2")
rp$print(show="t")

## wiping temporary repo
unlink(rp_path, TRUE)
```

repo_tags *List all tags*

Description

Shows list of all unique tags associated with any item in the repository.

Usage

```
repo_tags(name)
```

Arguments

name	The name of a repository item.

Value

Character vector of unique tags defined in the repo.

See Also

repo_put

Examples

```
rp_path <- file.path(tempdir(), "example_repo")
rp <- repo_open(rp_path, TRUE)

## Putting two items with a few tags
rp$put(1, "item1", "Sample item 1",
    c("repo_tags", "tag1"))
rp$put(2, "item2", "Sample item 2",
    c("repo_tags", "tag2"))

## Looking up tags
rp$tags()

## wiping temporary repo
unlink(rp_path, TRUE)
```

repo_untag *Remove tags from an item.*

Description

Remove tags from an item.

Usage

```
repo_untag(name = NULL, rmtags, tags = NULL)
```

Arguments

name	An item name.
rmtags	A list of tags that will be removed from the item's tag list.
tags	A list of tags: rmtags will be removed from all items matching the list.

Value

Used for side effects.

See Also

repo_tag, repo_set

Examples

```
rp_path <- file.path(tempdir(), "example_repo")
rp <- repo_open(rp_path, TRUE)
rp$put(1, "item1", "Sample item 1", c("tag1", "tag2"))
rp$print(show="t")
rp$untag("item1", "tag2")
rp$print(show="t")

## wiping temporary repo
unlink(rp_path, TRUE)
```

Index