Package: gistr (via r-universe)

October 28, 2024

Title Work with 'GitHub' 'Gists'

Description Work with 'GitHub' 'gists' from 'R' (e.g., <https://en.wikipedia.org/wiki/GitHub#Gist>, <https://docs.github.com/en/github/writing-on-github/creating-gists/>). A 'gist' is simply one or more files with code/text/images/etc. This package allows the user to create new 'gists', update 'gists' with new files, rename files, delete files, get and delete 'gists', star and 'un-star' 'gists', fork 'gists', open a 'gist' in your default browser, get embed code for a 'gist', list 'gist' 'commits', and get rate limit information when 'authenticated'. Some requests require authentication and some do not. 'Gists' website: <https://gist.github.com/>.

Version 0.9.0.93

License MIT + file LICENSE

URL https://github.com/ropensci/gistr (devel), https://docs.ropensci.org/gistr (website)

BugReports https://github.com/ropensci/gistr/issues

Roxygen list(markdown = TRUE)

VignetteBuilder knitr

Encoding UTF-8

Language en-US

Imports jsonlite, crul, httr, magrittr, assertthat, knitr, rmarkdown, dplyr

Suggests git2r, testthat

RoxygenNote 7.1.1

X-schema.org-applicationCategory Web

X-schema.org-keywords http, https, API, web-services, GitHub, GitHub API, gist, gists, code, script, snippet

X-schema.org-isPartOf https://ropensci.org

Repository https://ropensci.r-universe.dev

RemoteUrl https://github.com/ropensci/gistr

RemoteRef master

RemoteSha 61f2003a539bc7b08750c451b087a36b316c6a2e

Contents

`gistr-package` *R client for GitHub gists*

Description

R client for GitHub gists.

Details

gistr allows you to peform actions on gists, including listing, forking, starring, creating, deleting, updating, etc.

There are two ways to authorise gistr to work with your GitHub account:

- Generate a personal access token (PAT) at https://docs.github.com/en/github/authenticating-to-github/creating-a-personal-access-token and record it in the `GITHUB_PAT` envar.

- Interactively login into your GitHub account and authorise with OAuth.

Using the GITHUB_PAT is recommended.

Author(s)

Scott Chamberlain <myrmecocystus@gmail.com>

Ramnath Vaidyanathan <ramnath.vaidya@gmail.com>

Karthik Ram <karthik.ram@gmail.com>

`add_files` *Add files to a gist object*

Description

Add files to a gist object

Usage

```
add_files(gist, ...)

update_files(gist, ...)

delete_files(gist, ...)

rename_files(gist, ...)
```

Arguments

`gist`	A gist object or something coerceable to a gist
`...`	Curl options passed on to verb-GET

Examples

```
## Not run:
add_files("~/stuff.Rmd")
# update_files()
# delete_files()
# rename_files()

## End(Not run)
```

browse *Open a gist on GitHub*

Description

Open a gist on GitHub

Usage

```
browse(gist, what = "html")
```

Arguments

gist	A gist object or something that can be coerced to a gist object.
what	One of html (default), json, forks, commits, or comments.

commits *List gist commits*

Description

List gist commits

Usage

```
commits(gist, page = NULL, per_page = 30, ...)
```

Arguments

gist	A gist object or something coerceable to a gist
page	(integer) Page number to return.
per_page	(integer) Number of items to return per page. Default 30. Max 100.
...	Further named args to crul::verb-GET

Examples

```
## Not run:
gists()[[1]] %>% commits()
gist(id = '1f399774e9ecc9153a6f') %>% commits(per_page = 5)

# pass in a url
gist("https://gist.github.com/expersso/4ac33b9c00751fddc7f8") %>% commits

## End(Not run)
```

create_gists *Create gists*

Description

Creating gists in `gistr` can be done with any of three functions:

- `gist_create()` - Create gists from files or code blocks, using the GitHub HTTP API. Because this function uses the GitHub HTTP API, it does not work for binary files. However, you can get around this for images by using knitr's hook to upload images to eg., imgur. In addition, it's difficult to include artifacts from the knit-ing process.
- `gist_create_git()` - Create gists from files or code blocks, using git. Because this function uses git, you have more flexibility than with the above function: you can include any binary files, and can easily upload all artifacts.
- `gist_create_obj()` - Create gists from R objects: data.frame, list, character string, matrix, or numeric. Uses the GitHub HTTP API.

It may seem a bit odd to have three separate functions for creating gists. `gist_create()` was created first, and was out for a bit, so when we had the idea to create gists via git (`gist_create_git()`) and from R objects (`gist_create_obj()`), it made sense to have a different API for creating gists via the HTTP API, git, and from R objects. We could have thrown everything into `gist_create()`, but it would have been a massive function, with far too many parameters.

delete *Delete a gist*

Description

Delete a gist

Usage

```
delete(gist, ...)
```

Arguments

`gist`	A gist object or something coerceable to a gist
`...`	Curl options passed on to `verb-GET`

Examples

```
## Not run:
gists("minepublic")[[29]] %>% delete()

## End(Not run)
```

embed *Get embed script for a gist*

Description

Get embed script for a gist

Usage

```
embed(gist)
```

Arguments

gist	A gist object or something that can be coerced to a gist object.

Examples

```
## Not run:
gists()[[1]] %>% embed()

# pass in a url
gist("https://gist.github.com/expersso/4ac33b9c00751fddc7f8") %>% embed

## End(Not run)
```

fork *Fork a gist*

Description

Fork a gist

Usage

```
fork(gist, ...)
```

Arguments

gist	A gist object or something coerceable to a gist
...	Further named args to crul::verb-GET

Value

A gist class object

Examples

```
## Not run:
# fork a gist
w <- gists()[[1]] %>% fork()

# browse to newly forked gist
browse(w)

## End(Not run)
```

`forks` *List forks on a gist*

Description

List forks on a gist

Usage

```
forks(gist, page = NULL, per_page = 30, ...)
```

Arguments

`gist`	A gist object or something coerceable to a gist
`page`	(integer) Page number to return.
`per_page`	(integer) Number of items to return per page. Default 30. Max 100.
`...`	Further named args to crul::verb-GET

Value

A list of gist class objects

Examples

```
## Not run:
gist(id='1642874') %>% forks(per_page=2)
gist(id = "8172796") %>% forks()

# pass in a url
gist("https://gist.github.com/expersso/4ac33b9c00751fddc7f8") %>% forks

## End(Not run)
```

`gist` *Get a gist*

Description

Get a gist

Usage

```
gist(id, revision = NULL, ...)

as.gist(x)
```

Arguments

`id`	(character) A gist id, or a gist URL
`revision`	(character) A sha. optional
`...`	Curl options passed on to verb-GET
`x`	Object to coerce. Can be an integer (gist id), string (gist id), a gist, or an list that can be coerced to a gist.

Details

If a file is larger than ~1 MB, the content of the file given back is truncated, so you won't get the entire contents. In the return S3 object that's printed, we tell you at the bottom whether each file is truncated or not. If a file is, simply get the `raw_url` URL for the file (see example below), then retrieve from that. If the file is very big, you may need to clone the file using git, etc.

Examples

```
## Not run:
gist('f1403260eb92f5dfa7e1')

as.gist('f1403260eb92f5dfa7e1')
as.gist(10)
as.gist(gist('f1403260eb92f5dfa7e1'))

# get a specific revision of a gist
id <- 'c1e2cb547d9f22bd314da50fe9c7b503'
gist(id, 'a5bc5c143beb697f23b2c320ff5a8dacf960b0f3')
gist(id, 'b70d94a8222a4326dff46fc85bc69d0179bd1da2')
gist(id, '648bb44ab9ae59d57b4ea5de7d85e24103717e8b')
gist(id, '0259b13c7653dc95e20193133bcf71811888cbe6')

# from a url, or partial url
x <- "https://gist.github.com/expersso/4ac33b9c00751fddc7f8"
x <- "gist.github.com/expersso/4ac33b9c00751fddc7f8"
x <- "gist.github.com/4ac33b9c00751fddc7f8"
```

```
x <- "expersso/4ac33b9c00751fddc7f8"
as.gist(x)

ids <- sapply(gists(), "[[", "id")
gist(ids[1])
gist(ids[2])
gist(ids[3])
gist(ids[4])

gist(ids[1]) %>% browse()

## If a gist file is > a certain size it is truncated
## in this case, we let you know in the return object that it is truncated
## e.g.
(bigfile <- gist(id = "b74b878fd7d9176a4c52"))
## then get the raw_url, and retrieve the file
url <- bigfile$files$`plossmall.json`$raw_url

## End(Not run)
```

`gists` *List gists*

Description

List public gists, your own public gists, all your gists, by gist id, or query by date.

Usage

```
gists(what = "public", since = NULL, page = NULL, per_page = 30, ...)
```

Arguments

`what`	(character) What gists to return. One of public, minepublic, mineall, or starred. If an id is given for a gist, this parameter is ignored.
`since`	(character) A timestamp in ISO 8601 format: YYYY-MM-DDTHH:MM:SSZ. Only gists updated at or after this time are returned.
`page`	(integer) Page number to return.
`per_page`	(integer) Number of items to return per page. Default 30. Max 100.
`...`	Curl options passed on to `verb-GET`

Details

When `what = "mineall"`, we use `getOption("github.username")` internally to get your GitHub user name. Make sure to set your GitHub user name as an R option like `options(github.username = "foobar")` in your `.Rprofile` file. If we can't find you're user name, we'll stop with an error.

Examples

```
## Not run:
# Public gists
gists()
gists(per_page=2)
gists(page=3)
# Public gists created since X time
gists(since='2014-05-26T00:00:00Z')
# Your public gists
gists('minepublic')
gists('minepublic', per_page=2)
# Your private and public gists
gists('mineall')
# Your starred gists
gists('starred')
# pass in curl options
gists(per_page=1, verbose=TRUE)

## End(Not run)
```

---

gist_auth *Authorize with GitHub.*

Description

This function is run automatically to allow gistr to access your GitHub account.

Usage

```
gist_auth(app = gistr_app, reauth = FALSE)
```

Arguments

app	An `httr::oauth_app()` for GitHub. The default uses an application `gistr_oauth` created by Scott Chamberlain.
reauth	(logical) Force re-authorization?

Details

There are two ways to authorise gistr to work with your GitHub account:

- Generate a personal access token with the gist scope selected, and set it as the `GITHUB_PAT` environment variable per session using `Sys.setenv` or across sessions by adding it to your `.Renviron` file or similar. See https://help.github.com/articles/creating-an-access-token-for-command-line-use for help
- Interactively login into your GitHub account and authorise with OAuth.

Using `GITHUB_PAT` is recommended.

Value

a named list, with a single slot for `Authorization`, with a single element with the token - this is the expected format needed when passed down to the http request

Examples

```
## Not run:
gist_auth()

## End(Not run)
```

`gist_create` *Create a gist*

Description

Create a gist

Usage

```
gist_create(
  files = NULL,
  description = "",
  public = TRUE,
  browse = TRUE,
  code = NULL,
  filename = "code.R",
  knit = FALSE,
  knitopts = list(),
  renderopts = list(),
  include_source = FALSE,
  imgur_inject = FALSE,
  rmarkdown = FALSE,
  ...
)
```

Arguments

`files`	Files to upload. this or `code` param must be passed
`description`	(character) Brief description of gist (optional)
`public`	(logical) Whether gist is public (default: TRUE)
`browse`	(logical) To open newly create gist in default browser (default: TRUE)
`code`	Pass in any set of code. This can be a single R object, or many lines of code wrapped in quotes, then curly brackets (see examples below). this or `files` param must be passed

filename	Name of the file to create, only used if code parameter is used. Default to code.R
knit	(logical) Knit code before posting as a gist? If the file has a .Rmd or .Rnw extension, we run the file with knit, and if it has a .R extension, then we use render
knitopts, renderopts	(list) List of variables passed on to knit, or render
include_source	(logical) Only applies if knit=TRUE. Include source file in the gist in addition to the knitted output.
imgur_inject	(logical) Inject imgur_upload into your .Rmd file to upload files to https://imgur.com/. This will be ignored if the file is a sweave/latex file because the rendered pdf can't be uploaded anyway. Default: FALSE
rmarkdown	(logical) If TRUE, use rmarkdown::render() instead of knitr::knit() to render the document.
...	Further args passed on to verb-POST

See Also

gist_create_obj(), gist_create_git()

Examples

```
## Not run:
file <- tempfile()
cat("hello world", file = file)
gist_create(files=file, description='a new cool gist')

file1 <- tempfile()
file2 <- tempfile()
cat("foo bar", file = file1)
cat("foo bar", file = file2)
gist_create(files=c(file1, file2), description='spocc demo files')

# include any code by passing to the code parameter
gist_create(code={'
x <- letters
numbers <- runif(10)
numbers
'})

# Knit an .Rmd file before posting as a gist
file <- system.file("examples", "stuff.Rmd", package = "gistr")
gist_create(file, description='a new cool gist', knit=TRUE)

file <- system.file("examples", "plots.Rmd", package = "gistr")
gist_create(file, description='some plots', knit=TRUE)

# an .Rnw file
file <- system.file("examples", "rnw_example.Rnw", package = "gistr")
gist_create(file)
```

```
gist_create(file, knit=TRUE)

# Knit code input before posting as a gist
gist_create(code={'
```{r}
x <- letters
(numbers <- runif(8))
```
'}, knit=TRUE)

url <- "https://raw.githubusercontent.com/ropensci/geojsonio/master/inst/examples/zillow_or.geojson"
json <- crul::HttpClient$new(url)$get()$parse("UTF-8")
gist_create(code = json, filename = "zillow_or.geojson")

# Knit and include source file, so both files are in the gist
file <- system.file("examples", "stuff.Rmd", package = "gistr")
gist_create(file, knit=TRUE, include_source=TRUE)

gist_create(code={'
```{r}
x <- letters
(numbers <- runif(8))
```
'}, filename="code.Rmd", knit=TRUE, include_source=TRUE)

Uploading images created during knit process
using imgur - if you're file uses imgur or similar, you're good
file <- system.file("examples", "plots_imgur.Rmd", package = "gistr")
cat(readLines(file), sep = "\n") # peek at file
gist_create(file, knit=TRUE)
if not, GitHub doesn't allow upload of binary files via the HTTP API
(which gistr uses) - so see gist_create_git(), which uses git
file <- system.file("examples", "plots.Rmd", package = "gistr")
gist_create(file, knit=TRUE, imgur_inject = TRUE)
works with ggplot2 as well
file <- system.file("examples", "ggplot_imgur.Rmd", package = "gistr")
gist_create(file, knit=TRUE)

# Render `.R` files
file <- system.file("examples", "example1.R", package = "gistr")
cat(readLines(file), sep = "\n") # peek at file
gist_create(file, knit = TRUE)
gist_create(file, knit = TRUE, include_source = TRUE)
many files
(file1 <- system.file("examples", "example1.R", package = "gistr"))
(file2 <- system.file("examples", "example2.R", package = "gistr"))
cat(readLines(file1), sep = "\n") # peek at file
cat(readLines(file2), sep = "\n") # peek at file
gist_create(files=list(file1, file2), knit = TRUE)
three at once, some .R and some .Rmd
file3 <- system.file("examples", "plots_imgur.Rmd", package = "gistr")
gist_create(files=list(file1, file2, file3), knit = TRUE)
gist_create(files=list(file1, file2, file3), knit = TRUE,
```

```
  include_source = TRUE)

# Use rmarkdown::render instead of knitr::knit
file <- system.file("examples", "rmarkdown_eg.Rmd", package = "gistr")
gist_create(file, knit = TRUE, rmarkdown = TRUE, imgur_inject = TRUE,
   renderopts = list(output_format = "md_document"))

## End(Not run)
```

`gist_create_git` *Create a gist via git instead of the GitHub Gists HTTP API*

Description

Create a gist via git instead of the GitHub Gists HTTP API

Usage

```
gist_create_git(
  files = NULL,
  description = "",
  public = TRUE,
  browse = TRUE,
  knit = FALSE,
  code = NULL,
  filename = "code.R",
  knitopts = list(),
  renderopts = list(),
  include_source = FALSE,
  artifacts = FALSE,
  imgur_inject = FALSE,
  git_method = "ssh",
  sleep = 1,
  ...
)
```

Arguments

`files`	Files to upload. this or code param must be passed
`description`	(character) Brief description of gist (optional)
`public`	(logical) Whether gist is public (default: TRUE)
`browse`	(logical) To open newly create gist in default browser (default: TRUE)
`knit`	(logical) Knit code before posting as a gist? If the file has a `.Rmd` or `.Rnw` extension, we run the file with `knit`, and if it has a `.R` extension, then we use `render`

code — Pass in any set of code. This can be a single R object, or many lines of code wrapped in quotes, then curly brackets (see examples below). this or files param must be passed

filename — Name of the file to create, only used if code parameter is used. Default to code.R

knitopts, renderopts — (list) List of variables passed on to knit, or render

include_source — (logical) Only applies if knit=TRUE. Include source file in the gist in addition to the knitted output.

artifacts — (logical/character) Include artifacts or not. If TRUE, includes all artifacts. Or you can pass in a file extension to only upload artifacts of certain file exensions. Default: FALSE

imgur_inject — (logical) Inject imgur_upload into your .Rmd file to upload files to https://imgur.com/. This will be ignored if the file is a sweave/latex file because the rendered pdf can't be uploaded anyway. Default: FALSE

git_method — (character) One of ssh (default) or https. If a remote already exists, we use that remote, and this parameter is ignored.

sleep — (integer) Seconds to sleep after creating gist, but before collecting metadata on the gist. If uploading a lot of stuff, you may want to set this to a higher value, otherwise, you may not get accurate metadata for your gist. You can of course always refresh afterwards by calling gist with your gist id.

... — Further args passed on to verb-POST

Details

Note that when browse=TRUE there is a slight delay in when we open up the gist in your default browser and when the data will display in the gist. We could have this function sleep a while and guess when it will be ready, but instead we open your gist right after we're done sending the data to GitHub. Make sure to refresh the page if you don't see your content right away.

Likewise, the object that is returned from this function call may not have the updated and correct file information. You can retrieve that easily by calling gist() with the gist id.

This function uses git instead of the HTTP API, and thus requires the R package git2r. If you don't have git2r installed, and try to use this function, it will stop and tell you to install git2r.

This function using git is better suited than gist_create() for use cases involving:

- Big files - The GitHub API allows only files of up to 1 MB in size. Using git we can get around that limit.
- Binary files - Often artifacts created are binary files like .png. The GitHub API doesn't allow transport of binary files, but we can do that with git.

Another difference between this function and gist_create() is that this function can collect all artifacts coming out of a knit process.

If a gist is somehow deleted, or the remote changes, when you try to push to the same gist again, everything should be fine. We now use tryCatch on the push attempt, and if it fails, we'll add a new remote (which means a new gist), and push again.

See Also

gist_create(), gist_create_obj()

Examples

```
## Not run:
# prepare a directory and a file
unlink("~/gitgist", recursive = TRUE)
dir.create("~/gitgist")
file <- system.file("examples", "stuff.md", package = "gistr")
writeLines(readLines(file), con = "~/gitgist/stuff.md")

# create a gist
gist_create_git(files = "~/gitgist/stuff.md")

## more than one file can be passed in
unlink("~/gitgist2", recursive = TRUE)
dir.create("~/gitgist2")
file.copy(file, "~/gitgist2/")
cat("hello world", file = "~/gitgist2/hello_world.md")
list.files("~/gitgist2")
gist_create_git(c("~/gitgist2/stuff.md", "~/gitgist2/hello_world.md"))

# Include all files in a directory
unlink("~/gitgist3", recursive = TRUE)
dir.create("~/gitgist3")
cat("foo bar", file="~/gitgist3/foobar.txt")
cat("hello", file="~/gitgist3/hello.txt")
list.files("~/gitgist3")
gist_create_git("~/gitgist3")

# binary files
png <- system.file("examples", "file.png", package = "gistr")
unlink("~/gitgist4", recursive = TRUE)
dir.create("~/gitgist4")
file.copy(png, "~/gitgist4/")
list.files("~/gitgist4")
gist_create_git(files = "~/gitgist4/file.png")

# knit files first, then push up
# note: by default we don't upload images, but you can do that,
# see next example
rmd <- system.file("examples", "plots.Rmd", package = "gistr")
unlink("~/gitgist5", recursive = TRUE)
dir.create("~/gitgist5")
file.copy(rmd, "~/gitgist5/")
list.files("~/gitgist5")
gist_create_git("~/gitgist5/plots.Rmd", knit = TRUE)

# collect all/any artifacts from knitting process
arts <- system.file("examples", "artifacts_eg1.Rmd", package = "gistr")
unlink("~/gitgist6", recursive = TRUE)
```

```
dir.create("~/gitgist6")
file.copy(arts, "~/gitgist6/")
list.files("~/gitgist6")
gist_create_git("~/gitgist6/artifacts_eg1.Rmd", knit = TRUE,
   artifacts = TRUE)

# from a code block
gist_create_git(code={'
x <- letters
numbers <- runif(8)
numbers

[1] 0.3229318 0.5933054 0.7778408 0.3898947 0.1309717 0.7501378 0.3206379 0.3379005
'}, filename="my_cool_code.R")

# Use https instead of ssh
png <- system.file("examples", "file.png", package = "gistr")
unlink("~/gitgist7", recursive = TRUE)
dir.create("~/gitgist7")
file.copy(png, "~/gitgist7/")
list.files("~/gitgist7")
gist_create_git(files = "~/gitgist7/file.png", git_method = "https")

## End(Not run)
```

`gist_create_obj` *Create a gist from an R object*

Description

Create a gist from an R object

Usage

```
gist_create_obj(
  x = NULL,
  description = "",
  public = TRUE,
  browse = TRUE,
  pretty = TRUE,
  filename = "file.txt",
  ...
)
```

Arguments

`x`	An R object, any of data.frame, matrix, list, character, numeric
`description`	(character) Brief description of gist (optional)
`public`	(logical) Whether gist is public (default: `TRUE`)

browse	(logical) To open newly create gist in default browser (default: TRUE)
pretty	(logical) For data.frame and matrix objects, create a markdown table. If FALSE, pushes up json. (default: TRUE)
filename	Name of the file to create. Default: file.txt
...	Further args passed on to crul::verb-POST

Details

This function is specifically for going from R objects to a gist, whereas gist_create() is for going from files or executing code

See Also

gist_create(), gist_create_git()

Examples

```
## Not run:
## data.frame
### by default makes pretty table in markdown format
row.names(mtcars) <- NULL
gist_create_obj(mtcars)
gist_create_obj(iris)
### or just push up json
gist_create_obj(mtcars, pretty = FALSE)

## matrix
gist_create_obj(as.matrix(mtcars))
## list
gist_create_obj(apply(mtcars, 1, as.list))
## character
gist_create_obj("hello, world")
## numeric
gist_create_obj(runif(10))

## Assign a specific file name
gist_create_obj("
## header2

hey there!", filename = "my_markdown.md")

## End(Not run)
```

---

gist_map *Opens a full screen map after uploading a geojson file*

Description

Takes a gist object and a input geojson file name and renders fullscreen map

Usage

```
gist_map(x, browse = TRUE)
```

Arguments

x	An object of class gist generated by gist_create() or gist_create_obj()
browse	Default: TRUE. Set to FALSE if you don't want to automatically browse to the URL.

Examples

```
## Not run:
file <- system.file("examples", "ecoengine_eg.geojson", package = "gistr")
gist_id <- gist_create(file, browse = FALSE)
gist_map(gist_id)

## End(Not run)
```

---

gist_save *Save gist files to disk*

Description

Save gist files to disk

Usage

```
gist_save(gist, path = ".")

gist_open(x)
```

Arguments

gist	A gist object or something coerceable to a gist
path	Root path to write to, a directory, not a file b/c a gist can contain many files. A folder is created with name of the gist id within this root directory. File names will be the same as given in the gist.
x	An object of class gist_files (the output from gist_save()

Details

gist_save: files are written into a new folder, named by the gist id, e.g., a65ac7e56b7b3f746913

gist_open: opens files in your editor/R GUI. Internally, uses file.edit() to open files, using getOption("editor") to open the files. If you're in R.app or RStudio, or other IDE's, files will open in the IDE (I think).

Value

An object of class `gist_files`, S3 object containing file paths

Examples

```
## Not run:
gist("a65ac7e56b7b3f746913") %>% gist_save()
gist("a65ac7e56b7b3f746913") %>% gist_save() %>% gist_open()
gist("https://gist.github.com/expersso/4ac33b9c00751fddc7f8") %>%
  gist_save()

## End(Not run)
```

---

rate_limit *Get rate limit information*

Description

Get rate limit information

Usage

```
rate_limit(...)
```

Arguments

`...`	Named args to crul::verb-GET

Examples

```
## Not run:
rate_limit()

## End(Not run)
```

---

run *Run a .Rmd file*

Description

Run a .Rmd file

Usage

```
run(x, filename = "code.R", knitopts = list())
```

Arguments

x	Input, one of: code wrapped in curly brackets and quotes, a file path to an .Rmd file, or a gist.
filename	Name of the file to create, only used if code parameter is used. Default to code.R
knitopts	(list) List of variables passed on to knitr::knit()

Value

A path, unless a gist object is passed in, in which case a gist object is returned.

Examples

```
## Not run:
# run a local file
file <- system.file("examples", "stuff.Rmd", package = "gistr")
run(file) %>% gist_create

# run code
run({'
```{r}
x <- letters
(numbers <- runif(8))
```
'}) %>% gist_create

# run a file from a gist, has to get file first
gists('minepublic')[[2]] %>% run() %>% update()

End(Not run)
```

---

star *Star a gist*

Description

Star a gist

Usage

```
star(gist, ...)

unstar(gist, ...)

star_check(gist, ...)
```

Arguments

gist	A gist object or something that can be coerced to a gist object.
...	Curl options passed on to verb-GET

Value

A message, and a gist object, the same one input to the function.

Examples

```
## Not run:
id <- '4ac33b9c00751fddc7f8'
gist(id) %>% star()
gist(id) %>% star_check()
gist(id) %>% unstar()
gist(id) %>% unstar() %>% star()
gist(id) %>% star_check()
gist(id) %>%
  star() %>%
  star_check()

# pass in a url
x <- "https://gist.github.com/expersso/4ac33b9c00751fddc7f8"
gist(x) %>% star
gist(x) %>% unstar

## End(Not run)
```

---

tabl *Make a table from gist or commit class or a list of either*

Description

Make a table from gist or commit class or a list of either

Usage

```
tabl(x, ...)

tabl_data(x)
```

Arguments

x	Either a gist or commit class object or a list of either
...	Ignored

Details

For commits we return a single data.frame. For gists, we always return a list so that we are returning data consistently, regardless of variable return data. So you can always index to the main data.frame with gist metadata and file info by doing `result$data`, and likewise for forks `result$forks` and history `result$history`

Value

A data.frame or list of data.frame's

Examples

```
## Not run:
# from a gist object
x <- as.gist('f1403260eb92f5dfa7e1')
res <- tabl(x)
res$data
res$forks
res$history

# from a list
ss <- gists('minepublic')
tabl(ss[1:3])
lapply(tabl(ss[1:3]), "[[", "data")
# index to data slots, but also make single data.frame
tabl_data(tabl(ss[1:3]))
## manipulate with dplyr
library("dplyr")
tabl_data(tabl(ss[1:30])) %>%
  select(id, description, owner_login) %>%
  filter(grepl("gist gist gist", description))

# commits
x <- gists()[[2]] %>% commits()
tabl(x[[1]])

## many
x <- sapply(gists(per_page = 100), commits)
tabl(x) %>%
  select(id, login, change_status.total, url) %>%
  filter(change_status.total > 50)

# pass in a url
gist("https://gist.github.com/expersso/4ac33b9c00751fddc7f8") %>% tabl
## many
gg <- gists()
(urls <- vapply(gg, "[[", "", "html_url"))
lapply(urls[1:5], as.gist) %>% tabl()

# gist with forks and history
gist('1642874') %>% tabl
```

```
# gist with history, no forks
gist('c96d2e453c95d0166408') %>% tabl

## End(Not run)
```

`update` *Update/modify a gist*

Description

Update/modify a gist

Usage

```
update(gist, description = gist$description, ...)
```

Arguments

`gist`	A gist object or something coerceable to a gist
`description`	(character) Brief description of gist (optional)
`...`	Curl options passed on to `verb-GET`

Value

an object of class `gist`

Examples

```
## Not run:
file1 <- system.file("examples", "alm.md", package = "gistr")
file2 <- system.file("examples", "zoo.json", package = "gistr")

# add new files
gists(what = "minepublic")[[3]] %>%
 add_files(file1, file2) %>%
 update()

# update existing files
### file name has to match to current name
gists(what = "minepublic")[[3]] %>%
 update_files(file1) %>%
 update()

# delete existing files
### again, file name has to match to current name
gists(what = "minepublic")[[3]] %>%
 delete_files(file1, file2) %>%
 update()
```

```
# rename existing files
# For some reason, this operation has to upload the content too
### first name is old file name with path (must match), and second is
### new file name (w/o path)
## add first
gists(what = "minepublic")[[3]] %>%
 add_files(file1, file2) %>%
 update()
## then rename
gists(what = "minepublic")[[3]] %>%
 rename_files(list(file1, "newfile.md")) %>%
 update()
### you can pass in many renames
gists(what = "minepublic")[[3]] %>%
 rename_files(list(file1, "what.md"), list(file2, "new.json")) %>%
 update()

## End(Not run)
```

Index