Package 'gh'

January 24, 2020

Title 'GitHub' 'API'

Version 1.1.0

Description Minimal client to access the 'GitHub' 'API'.

License MIT + file LICENSE

LazyData true

URL https://github.com/r-lib/gh#readme

BugReports https://github.com/r-lib/gh/issues

Suggests covr, keyring, pingr, testthat, withr

Imports cli, ini, jsonlite, httr (>= 1.2)

RoxygenNote 7.0.2.9000

Encoding UTF-8

NeedsCompilation no

Author Gábor Csárdi [cre, ctb],
Jennifer Bryan [aut],
Hadley Wickham [aut]

Maintainer Gábor Csárdi <csardi.gabor@gmail.com>

Repository CRAN

Date/Publication 2020-01-24 06:50:17 UTC

R topics documented:

gh *GitHub API*

Description

Minimal wrapper to access GitHub's API.

This is an extremely minimal client. You need to know the API to be able to use this client. All this function does is:

- Try to substitute each listed parameter into `endpoint`, using the :parameter notation.
- If a GET request (the default), then add all other listed parameters as query parameters.
- If not a GET request, then send the other parameters in the request body, as JSON.
- Convert the response to an R list using `jsonlite::fromJSON()`.

Usage

```
gh(
  endpoint,
  ...,
  per_page = NULL,
  .token = NULL,
  .destfile = NULL,
  .overwrite = FALSE,
  .api_url = NULL,
  .method = "GET",
  .limit = NULL,
  .accept = "application/vnd.github.v3+json",
  .send_headers = NULL,
  .progress = TRUE
)
```

Arguments

`endpoint`	GitHub API endpoint. Must be one of the following forms: • METHOD path, e.g. `GET /rate_limit`, • `path`, e.g. /rate_limit, • METHOD url, e.g. GET https://api.github.com/rate_limit, • `url`, e.g. https://api.github.com/rate_limit. If the method is not supplied, will use `.method`, which defaults to `"GET"`.
`...`	Name-value pairs giving API parameters. Will be matched into `endpoint` placeholders, sent as query parameters in GET requests, and as a JSON body of POST requests. If there is only one unnamed parameter, and it is a raw vector, then it will not be JSON encoded, but sent as raw data, as is. This can be used for example to add assets to releases. Named `NULL` values are silently dropped, and named `NA` values trigger an error.

per_page	Number of items to return per page. If omitted, will be substituted by max(.limit,100) if .limit is set, otherwise determined by the API (never greater than 100).
.token	Authentication token. Defaults to GITHUB_PAT or GITHUB_TOKEN environment variables, in this order if any is set. See gh_token() if you need more flexibility, e.g. different tokens for different GitHub Enterprise deployments.
.destfile	path to write response to disk. If NULL (default), response will be processed and returned as an object. If path is given, response will be written to disk in the form sent.
.overwrite	if .destfile is provided, whether to overwrite an existing file. Defaults to FALSE.
.api_url	Github API url (default: https://api.github.com). Used if endpoint just contains a path. Defaults to GITHUB_API_URL environment variable if set.
.method	HTTP method to use if not explicitly supplied in the endpoint.
.limit	Number of records to return. This can be used instead of manual pagination. By default it is NULL, which means that the defaults of the GitHub API are used. You can set it to a number to request more (or less) records, and also to Inf to request all records. Note, that if you request many records, then multiple GitHub API calls are used to get them, and this can take a potentially long time.
.accept	The value of the Accept HTTP header. Defaults to "application/vnd.github.v3+json". If Accept is given in .send_headers, then that will be used. This paramter can be used to provide a custom media type, in order to access a preview feature of the API.
.send_headers	Named character vector of header field values (except Authorization, which is handled via .token). This can be used to override or augment the default User-Agent header: "https://github.com/r-lib/gh".
.progress	Whether to show a progress indicator for calls that need more than one HTTP request.

Value

Answer from the API as a gh_response object, which is also a list. Failed requests will generate an R error. Requests that generate a raw response will return a raw vector.

Author(s)

Maintainer: Gábor Csárdi <csardi.gabor@gmail.com> [contributor]

Authors:

- Jennifer Bryan
- Hadley Wickham

See Also

Useful links:

- https://github.com/r-lib/gh#readme

- Report bugs at https://github.com/r-lib/gh/issues

gh_gql() if you want to use the GitHub GraphQL API, gh_whoami() for details on GitHub API token management.

Examples

```
## Repositories of a user, these are equivalent
gh("/users/hadley/repos")
gh("/users/:username/repos", username = "hadley")

## Starred repositories of a user
gh("/users/hadley/starred")
gh("/users/:username/starred", username = "hadley")


## Create a repository, needs a token in GITHUB_PAT (or GITHUB_TOKEN)
## environment variable
gh("POST /user/repos", name = "foobar")


## Issues of a repository
gh("/repos/hadley/dplyr/issues")
gh("/repos/:owner/:repo/issues", owner = "hadley", repo = "dplyr")

## Automatic pagination
users <- gh("/users", .limit = 50)
length(users)


## Access developer preview of Licenses API (in preview as of 2015-09-24)
gh("/licenses") # used to error code 415
gh("/licenses", .accept = "application/vnd.github.drax-preview+json")


## Access Github Enterprise API
## Use GITHUB_API_URL environment variable to change the default.
gh("/user/repos", type = "public", .api_url = "https://github.foobar.edu/api/v3")


## Use I() to force body part to be sent as an array, even if length 1
## This works whether assignees has length 1 or > 1
assignees <- "gh_user"
assignees <- c("gh_user1", "gh_user2")
gh("PATCH /repos/OWNER/REPO/issues/1", assignees = I(assignees))
```

Description

See more about the GraphQL API here: https://developer.github.com/v4/

Usage

```
gh_gql(
  query,
  ...,
  .token = NULL,
  .destfile = NULL,
  .overwrite = FALSE,
  .api_url = NULL,
  .send_headers = NULL
)
```

Arguments

query	The GraphQL query, as a string.
...	Name-value pairs giving API parameters. Will be matched into endpoint placeholders, sent as query parameters in GET requests, and as a JSON body of POST requests. If there is only one unnamed parameter, and it is a raw vector, then it will not be JSON encoded, but sent as raw data, as is. This can be used for example to add assets to releases. Named NULL values are silently dropped, and named NA values trigger an error.
.token	Authentication token. Defaults to GITHUB_PAT or GITHUB_TOKEN environment variables, in this order if any is set. See gh_token() if you need more flexibility, e.g. different tokens for different GitHub Enterprise deployments.
.destfile	path to write response to disk. If NULL (default), response will be processed and returned as an object. If path is given, response will be written to disk in the form sent.
.overwrite	if .destfile is provided, whether to overwrite an existing file. Defaults to FALSE.
.api_url	Github API url (default: https://api.github.com). Used if endpoint just contains a path. Defaults to GITHUB_API_URL environment variable if set.
.send_headers	Named character vector of header field values (except Authorization, which is handled via .token). This can be used to override or augment the default User-Agent header: "https://github.com/r-lib/gh".

Details

Note: pagination and the .limit argument does not work currently, as pagination in the GraphQL API is different from the v3 API. If you need pagination with GraphQL, you'll need to do that manually.

See Also

gh() for the GitHub v3 API.

Examples

```
gh_gql("query { viewer { login }}")
```

---

gh_next *Get the next, previous, first or last page of results*

Description

Get the next, previous, first or last page of results

Usage

```
gh_next(gh_response)

gh_prev(gh_response)

gh_first(gh_response)

gh_last(gh_response)
```

Arguments

gh_response	An object returned by a gh() call.

Details

Note that these are not always defined. E.g. if the first page was queried (the default), then there are no first and previous pages defined. If there is no next page, then there is no next page defined, etc.

If the requested page does not exist, an error is thrown.

Value

Answer from the API.

See Also

The .limit argument to gh() supports fetching more than one page.

Examples

```
x <- gh("/users")
vapply(x, "[[", character(1), "login")
x2 <- gh_next(x)
vapply(x2, "[[", character(1), "login")
```

gh_token *Return the local user's GitHub Personal Access Token (PAT)*

Description

You can read more about PATs here: https://help.github.com/articles/creating-a-personal-access-token-for- and you can access your PATs here (if logged in to GitHub): https://github.com/settings/tokens.

Usage

```
gh_token(api_url = NULL)
```

Arguments

api_url	Github API url. Defaults to GITHUB_API_URL environment variable if set, otherwise https://api.github.com.

Details

Set the GITHUB_PAT environment variable to avoid having to include your PAT in the code. If you work with multiple GitHub deployments, e.g. via GitHub Enterprise, then read 'PATs for GitHub Enterprise' below.

If you want a more secure solution than putting authentication tokens into environment variables, read 'Storing PATs in the system keyring' below.

Value

A string, with the token, or a zero length string scalar, if no token is available.

NA

gh supports storing your PAT in the system keyring, on Windows, macOS and Linux, using the keyring package. To turn on keyring support, you need to set the GH_KEYRING environment variables to true, in your .Renviron file or profile.

If keyring support is turned on, then for each PAT environment variable, gh first checks whether the key with that value is set in the system keyring, and if yes, it will use its value as the PAT. I.e. without a custom GITHUB_API_URL variable, it checks the GITHUB_PAT_API_GITHUB_COM key first, then the env var with the same name, then the GITHUB_PAT key, etc. Such a check looks like this:

```
keyring::key_get("GITHUB_PAT_API_GITHUB_COM")
```

and it uses the default keyring backend and the default keyring within that backend. See keyring::default_backend() for details and changing these defaults.

If the selected keyring is locked, and the session is interactive, then gh will try to unlock it. If the keyring is locked, and the session is not interactive, then gh will not use the keyring. Note that some keyring backends cannot be locked (e.g. the one that uses environment variables).

On some OSes, e.g. typically on macOS, you need to allow R to access the system keyring. You can allow this separately for each access, or for all future accesses, until you update or re-install R. You typically need to give access to each R GUI (e.g. RStudio) and the command line R program separately.

To store your PAT on the keyring run

```
keyring::key_set("GITHUB_PAT")
```

Storing PATs in the system keyring

gh supports storing your PAT in the system keyring, on Windows, macOS and Linux, using the keyring package. To turn on keyring support, you need to set the `GH_KEYRING` environment variables to `true`, in your `.Renviron` file or profile.

If keyring support is turned on, then for each PAT environment variable, gh first checks whether the key with that value is set in the system keyring, and if yes, it will use its value as the PAT. I.e. without a custom `GITHUB_API_URL` variable, it checks the `GITHUB_PAT_API_GITHUB_COM` key first, then the env var with the same name, then the `GITHUB_PAT` key, etc. Such a check looks like this:

```
keyring::key_get("GITHUB_PAT_API_GITHUB_COM")
```

and it uses the default keyring backend and the default keyring within that backend. See `keyring::default_backend()` for details and changing these defaults.

If the selected keyring is locked, and the session is interactive, then gh will try to unlock it. If the keyring is locked, and the session is not interactive, then gh will not use the keyring. Note that some keyring backends cannot be locked (e.g. the one that uses environment variables).

On some OSes, e.g. typically on macOS, you need to allow R to access the system keyring. You can allow this separately for each access, or for all future accesses, until you update or re-install R. You typically need to give access to each R GUI (e.g. RStudio) and the command line R program separately.

To store your PAT on the keyring run

```
keyring::key_set("GITHUB_PAT")
```

See Also

`slugify_url()` for computing the environment variables that gh uses to search for API URL specific PATs.

`gh_tree_remote` *Find the GitHub remote associated with a path*

Description

This is handy helper if you want to make gh requests related to the current project.

Usage

```
gh_tree_remote(path = ".")
```

Arguments

path	Path that is contained within a git repo.

Value

If the repo has a github remote, a list containing username and repo. Otherwise, an error.

Examples

```
gh_tree_remote()
```

---

gh_whoami *Info on current GitHub user and token*

Description

Reports wallet name, GitHub login, and GitHub URL for the current authenticated user, the first bit of the token, and the associated scopes.

Usage

```
gh_whoami(.token = NULL, .api_url = NULL, .send_headers = NULL)
```

Arguments

.token	Authentication token. Defaults to GITHUB_PAT or GITHUB_TOKEN environment variables, in this order if any is set. See gh_token() if you need more flexibility, e.g. different tokens for different GitHub Enterprise deployments.
.api_url	Github API url (default: https://api.github.com). Used if endpoint just contains a path. Defaults to GITHUB_API_URL environment variable if set.
.send_headers	Named character vector of header field values (except Authorization, which is handled via .token). This can be used to override or augment the default User-Agent header: "https://github.com/r-lib/gh".

Details

Get a personal access token for the GitHub API from https://github.com/settings/tokens and select the scopes necessary for your planned tasks. The repo scope, for example, is one many are likely to need. The token itself is a string of 40 letters and digits. You can store it any way you like and provide explicitly via the .token argument to gh().

However, many prefer to define an environment variable GITHUB_PAT (or GITHUB_TOKEN) with this value in their .Renviron file. Add a line that looks like this, substituting your PAT:

```
GITHUB_PAT=8c70fd8419398999c9ac5bacf3192882193cadf2
```

Put a line break at the end! If you're using an editor that shows line numbers, there should be (at least) two lines, where the second one is empty. Restart R for this to take effect. Call gh_whoami() to confirm success.

To get complete information on the authenticated user, call gh("/user").

For token management via API (versus the browser), use the Authorizations API. This API requires Basic Authentication using your username and password, not tokens, and is outside the scope of the gh package.

Value

A gh_response object, which is also a list.

Examples

```
gh_whoami()


## explicit token + use with GitHub Enterprise
gh_whoami(.token = "8c70fd8419398999c9ac5bacf3192882193cadf2",
          .api_url = "https://github.foobar.edu/api/v3")
```

---

print.gh_response *Print the result of a GitHub API call*

Description

Print the result of a GitHub API call

Usage

```
## S3 method for class 'gh_response'
print(x, ...)
```

Arguments

x	The result object.
...	Ignored.

Value

The JSON result.

---

slugify_url *Compute the suffix that gh uses for GitHub API URL specific PATs*

Description

Compute the suffix that gh uses for GitHub API URL specific PATs

Usage

```
slugify_url(url)
```

Arguments

url	Character vector HTTP/HTTPS URLs.

Value

Character vector of suffixes.

See Also

gh_token()

Examples

```
# The main GH site
slugify_url("https://api.github.com")

# A custom one
slugify_url("https://github.acme.com")
```

Index