Package 'gh'

May 29, 2026

Title 'GitHub' 'API'

Version 1.6.0

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

License MIT + file LICENSE

URL https://gh.r-lib.org/, https://github.com/r-lib/gh

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

Depends R (>= 4.1)

Imports cli (>= 3.0.1), gitcreds, glue, httr2 (>= 1.0.6), ini, jsonlite, lifecycle, rlang (>= 1.1.0)

Suggests connectcreds, covr, knitr, rmarkdown, rprojroot, spelling, testthat (>= 3.0.0), vctrs, webfakes (>= 1.5.0), withr

VignetteBuilder knitr

Config/Needs/website tidyverse/tidytemplate

Config/testthat/edition 3

Config/usethis/last-upkeep 2025-04-29

Encoding UTF-8

Language en-US

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Gábor Csárdi [cre, ctb], Jennifer Bryan [aut], Hadley Wickham [aut], Posit Software, PBC [cph, fnd] (ROR: <https://ror.org/03wc8by49>)

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

Repository CRAN

Date/Publication 2026-05-29 13:20:07 UTC

Contents

`fake_github_app` *A fake GitHub web app*

Description

A `webfakes::new_app()` application that implements the subset of the GitHub REST API needed by the gh test suite. It is exported so that downstream packages that depend on gh can use it to test their own GitHub-backed code without hitting the real API.

Usage

```
fake_github_app()
```

Details

The app accepts any token whose shape is a valid GitHub PAT (40 hex characters or a `ghp_` / `github_pat_` / `ghs_` / `ghr_` / `gho_` / `ghu_` prefix). Any other non-empty token is rejected with a 401 "Bad credentials". A missing `Authorization` header yields a 401 "Requires authentication" for endpoints that need a user.

Value

A `webfakes_app` object.

Examples

```
app <- fake_github_app()
proc <- webfakes::new_app_process(app)
proc$url()
proc$stop()
```

gh *Query the GitHub API*

Description

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,
  .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,
  .params = list(),
  .max_wait = 600,
  .max_rate = NULL
)
```

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. For GET requests, named NA values trigger an error. For other methods, named NA values are included in the body of the request, as JSON null.

per_page, .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 gh_token().

.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. gh writes the response to a temporary file, and renames that file to .destfile after the request was successful. The name of the temporary file is created by adding a -<random>.gh-tmp suffix to it, where <random> is an ASCII string with random characters. gh removes the temporary file on error.

.overwrite
: If .destfile is provided, whether to overwrite an existing file. Defaults to FALSE. If an error happens the original file is kept.

.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 parameter 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.

.params
: Additional list of parameters to append to .... It is easier to use this than ... if you have your parameters in a list already.

.max_wait
: Maximum number of seconds to wait if rate limited. Defaults to 10 minutes.

.max_rate
: Maximum request rate in requests per second. Set this to automatically throttle requests.

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.

Caching

gh() uses httr2's HTTP cache by default. It is stored in tools::R_user_dir("gh", "cache"), capped at 100 MB, and shared across R sessions. When the cache holds a previous response for a request, gh() lets httr2 attach If-None-Match / If-Modified-Since headers automatically and replays the cached body on a 304 Not Modified reply, so the call doesn't count against your GitHub rate limit. Set options(gh_cache = FALSE) to disable, or delete the directory above to clear it.

If you pass If-None-Match (or If-Modified-Since) yourself via .send_headers, httr2's cache does not intervene: a 304 response is returned as an empty gh_response with the response headers (including ETag) still attached on attr(res, "response"). You are responsible for holding on to the previous body and matching it against the ETag.

See Also

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", .limit = 2)
gh("/users/{username}/repos", username = "hadley", .limit = 2)

## Starred repositories of a user
gh("/users/hadley/starred", .limit = 2)
gh("/users/{username}/starred", username = "hadley", .limit = 2)


## Create a repository, needs a token (see gh_token())
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))


## There are two ways to send JSON data. One is that you supply one or
## more objects that will be converted to JSON automatically via
## jsonlite::toJSON(). In this case sometimes you need to use
## jsonlite::unbox() because fromJSON() creates lists from scalar vectors
## by default. The Content-Type header is automatically added in this
## case. For example this request turns on GitHub Pages, using this
## API: https://docs.github.com/v3/repos/pages/#enable-a-pages-site

gh::gh(
  "POST /repos/{owner}/{repo}/pages",
  owner = "r-lib",
  repo = "gh",
  source = list(
    branch = jsonlite::unbox("gh-pages"),
    path = jsonlite::unbox("/")
  ),
  .send_headers = c(Accept = "application/vnd.github.switcheroo-preview+json")
)

## The second way is to handle the JSON encoding manually, and supply it
## as a raw vector in an unnamed argument, and also a Content-Type header:

body <- '{ "source": { "branch": "gh-pages", "path": "/" } }'
gh::gh(
  "POST /repos/{owner}/{repo}/pages",
  owner = "r-lib",
  repo = "gh",
  charToRaw(body),
  .send_headers = c(
    Accept = "application/vnd.github.switcheroo-preview+json",
    "Content-Type" = "application/json"
  )
)


## Pass along a query to the search/code endpoint via the ... argument
x <- gh::gh(
            "/search/code",
            q = "installation repo:r-lib/gh",
            .send_headers = c("X-GitHub-Api-Version" = "2022-11-28")
            )
 str(x, list.len = 3, give.attr = FALSE)
```

gh_gql *A simple interface for the GitHub GraphQL API v4.*

Description

See more about the GraphQL API here: https://docs.github.com/graphql

Usage

```
gh_gql(query, ...)
```

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. For GET requests, named `NA` values trigger an error. For other methods, named `NA` values are included in the body of the request, as JSON `null`.

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 }}")

# Get rate limit
ratelimit_query <- "query {
  viewer {
    login
  }
  rateLimit {
    limit
    cost
    remaining
    resetAt
  }
}"
```

```
gh_gql(ratelimit_query)
```

---

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, .token = NULL, .send_headers = NULL)

gh_prev(gh_response, .token = NULL, .send_headers = NULL)

gh_first(gh_response, .token = NULL, .send_headers = NULL)

gh_last(gh_response, .token = NULL, .send_headers = NULL)
```

Arguments

gh_response	An object returned by a gh() call.
.token	Authentication token. Defaults to gh_token().
.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 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_rate_limit` *Return GitHub user's current rate limits*

Description

`gh_rate_limits()` reports on all rate limits for the authenticated user. `gh_rate_limit()` reports on rate limits for previous successful request.

Further details on GitHub's API rate limit policies are available at https://docs.github.com/v3/#rate-limiting.

Usage

```
gh_rate_limit(
  response = NULL,
  .token = NULL,
  .api_url = NULL,
  .send_headers = NULL
)

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

Arguments

`response`	`gh_response` object from a previous gh call, rate limit values are determined from values in the response header. Optional argument, if missing a call to "GET /rate_limit" will be made.
`.token`	Authentication token. Defaults to `gh_token()`.
`.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"`.

Value

A `list` object containing the overall `limit`, `remaining` limit, and the limit `reset` time.

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

Description

If gh can find a personal access token (PAT) via `gh_token()`, it includes the PAT in its requests. Some requests succeed without a PAT, but many require a PAT to prove the request is authorized by a specific GitHub user. A PAT also helps with rate limiting. If your gh use is more than casual, you want a PAT.

gh calls `gitcreds::gitcreds_get()` with the `api_url`, which checks session environment variables (`GITHUB_PAT`, `GITHUB_TOKEN`) and then the local Git credential store for a PAT appropriate to the `api_url`. Therefore, if you have previously used a PAT with, e.g., command line Git, gh may retrieve and re-use it. You can call `gitcreds::gitcreds_get()` directly, yourself, if you want to see what is found for a specific URL. If no matching PAT is found, `gitcreds::gitcreds_get()` errors, whereas `gh_token()` does not and, instead, returns `""`.

See GitHub's documentation on Creating a personal access token, or use `usethis::create_github_token()` for a guided experience, including pre-selection of recommended scopes. Once you have a PAT, you can use `gitcreds::gitcreds_set()` to add it to the Git credential store. From that point on, gh (via `gitcreds::gitcreds_get()`) should be able to find it without further effort on your part.

Usage

```
gh_token(api_url = NULL)

gh_token_exists(api_url = NULL)
```

Arguments

`api_url`	GitHub API URL. Defaults to the `GITHUB_API_URL` environment variable, if set, and otherwise to `https://api.github.com`.

Value

A string of characters, if a PAT is found, or the empty string, otherwise. For convenience, the return value has an S3 class in order to ensure that simple printing strategies don't reveal the entire PAT.

Token format validation

gh warns if the PAT it retrieves does not match a known format. Set `options(gh_validate_tokens = "off")` or the `GH_VALIDATE_TOKENS=off` environment variable to avoid this warning. The option takes precedence over the environment variable.

Set `options(gh_validate_tokens = "error")` or the `GH_VALIDATE_TOKENS=error` environment variable to make gh throw an error for an unrecognized PAT format.

Examples

```
## Not run:
gh_token()

format(gh_token())

str(gh_token())

## End(Not run)
```

`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 gh_token().
.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.

On macOS and Windows it is best to store the token in the git credential store, where most GitHub clients, including gh, can access it. You can use the gitcreds package to add your token to the credential store:

```
gitcreds::gitcreds_set()
```

See https://gh.r-lib.org/articles/managing-personal-access-tokens.html and https://usethis.r-lib.org/articles/articles/git-credentials.html for more about managing GitHub (and generic git) credentials.

On other systems, including Linux, the git credential store is typically not as convenient, and you might want to store your token in the GITHUB_PAT environment variable, which you can set in your .Renviron file.

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.

Index