Package ‘gitlabr’

September 13, 2022

Title  Access to the 'Gitlab' API

Version  2.0.1

Description  Provides R functions to access the API of the project and repository management web application 'GitLab'. For many common tasks (repository file access, issue assignment and status, commenting) convenience wrappers are provided, and in addition the full API can be used by specifying request locations. 'GitLab' is open-source software and can be self-hosted or used on <https://about.gitlab.com>.

License  GPL (>= 3)

URL  https://statnmap.github.io/gitlabr/

BugReports  https://github.com/statnmap/gitlabr/issues

Depends  R (>= 3.1.2)

Imports  arpr, base64enc, dplyr (>= 0.4.3), httr (>= 1.1.0), magrittr, purr (>= 0.2.2), shiny (>= 0.13.0), stringr, tibble (>= 1.1), utils

Suggests  knitr, rmarkdown, testthat (>= 3.0.0), yaml

VignetteBuilder  knitr

Config/testthat/edition  3

Encoding  UTF-8

LazyData  true

RoxygenNote  7.2.1

NeedsCompilation  no

Author  Jirka Lewandowski [aut], Sébastien Rochette [aut, cre] (<https://orcid.org/0000-0002-1565-9313>)

Maintainer  Sébastien Rochette <sebastien@thinkr.fr>

Repository  CRAN

Date/Publication  2022-09-13 11:00:02 UTC
R topics documented:

```r
gitlab ......................................................... 2
gitlab_deprecated ................................. 4
gitlab_0_7_renaming .............................. 7
gitlab_options_set ............................ 7
glLoginInput ......................................... 8
gl_archive ........................................... 9
gl_ci_job ................................................ 10
gl_connection ........................................ 11
gl_create_merge_request ..................... 12
gl_get_comments ................................. 14
gl_get_commits .................................... 15
gl_get_project_id ............................... 16
gl_list_branches .................................. 16
gl_list_issues ..................................... 18
gl_list_projects .................................. 19
gl_new_issue ....................................... 20
gl_new_project .................................... 21
gl_pipelines ....................................... 22
gl_proj_req ......................................... 24
gl_push_file ...................................... 24
gl_repository ....................................... 25
gl_to_issue_id ..................................... 27
set_gitlab_connection ......................... 28
use_gitlab_ci ....................................... 28
```

Index

```
<table>
<thead>
<tr>
<th></th>
<th>Request GitLab API</th>
</tr>
</thead>
<tbody>
<tr>
<td>gitlab</td>
<td></td>
</tr>
</tbody>
</table>

Description

This is gitlabr’s core function to talk to GitLab’s server API via HTTP(S). Usually you will not use this function directly too often, but either use gitlabr’s convenience wrappers or write your own. See the gitlabr vignette for more information on this.

Usage

```r
gitlab(
  req,
  api_root,
  verb = http::GET,
  auto_format = TRUE,
  debug = FALSE,
  gitlab_con = "default",
  page = "all",
```
max_page = 10,
enforce_api_root = TRUE,
argname_verb = if (identical(verb, httr::GET) | identical(verb, httr::DELETE)) {
  "query"
} else {
  "body"
},
...

Arguments

req       vector of characters that represents the call (e.g. c("projects", project_id, "events"))
api_root  URL where the GitLab API to request resides (e.g. https://gitlab.myserver.com/api/v3/)
verb      http verb to use for request in form of one of the httr functions httr::GET(), httr::PUT(), httr::POST(), httr::DELETE()
auto_format whether to format the returned object automatically to a flat data.frame
debug     if TRUE API URL and query will be printed, defaults to FALSE
gitlab_con function to use for issuing API requests (e.g. as returned by gitlab_connection())
page      number of page of API response to get; if "all" (default), all pages (up to max_page parameter!) are queried successively and combined.
max_page  maximum number of pages to retrieve. Defaults to 10. This is an upper limit to prevent gitlabr getting stuck in retrieving an unexpectedly high number of entries (e.g. of a project list). It can be set to NA/Inf to retrieve all available pages without limit, but this is recommended only under controlled circumstances.

enforce_api_root

if multiple pages are requested, the API root URL is ensured to be the same as in the original call for all calls using the "next page" URL returned by GitLab This makes sense for security and in cases where GitLab is behind a reverse proxy and ignorant about its URL from external.

argname_verb

name of the argument of the verb that fields and information are passed on to

... named parameters to pass on to GitLab API (technically: modifies query parameters of request URL), may include private_token and all other parameters as documented for the GitLab API

Details

gitlab() function allows to use any request of the GitLab API https://docs.gitlab.com/ce/api/.

For instance, the API documentation shows how to create a new project in https://docs.gitlab.com/ce/api/projects.html#create-project:

- The verb is POST
- The request is projects
• Required attributes are name or path (if name not set)
• default_branch is an attribute that can be set if wanted

The corresponding use of `gitlab()` is:

```r
gitlab(
  req = "projects",
  verb = httr::POST,
  name = "toto",
  default_branch = "main"
)
```

Note: currently GitLab API v4 is supported. GitLab API v3 is no longer supported, but you can give it a try.

### Value

the response from the GitLab API, usually as a `tibble` and including all pages

### Examples

```r
## Not run:
# Connect as a fixed user to a GitLab instance
set_gitlab_connection(
  gitlab_url = "https://gitlab.com",
  private_token = Sys.getenv("GITLAB_COM_TOKEN")
)

# Use a simple request
gitlab(req = "projects")

# Use a combined request with extra parameters
gitlab(req = c("projects", 1234, "issues"),
  state = "closed")

## End(Not run)
```

---

**gitlabr-deprecated**  
**Deprecated functions**

**Description**

Many functions were renamed with version 0.7 to the `gl_` naming scheme. Note that the old function names are deprecated and might be removed without further notice.
Usage

archive(...)
assign_issue(...)
close_issue(...)
comment_commit(...)
comment_issue(...)
create_branch(...)
create_merge_request(...)
delete_branch(...)
edit_commit_comment(...)
edit_issue(...)
edit_issue_comment(...)
file_exists(...)
get_comments(...)
get_commit_comments(...)
get_commits(...)
get_diff(...)
get_file(...)
get_issue(...)
get_issue_comments(...)
get_issues(...)
get_project_id(...)
gitlab_connection(...)
list_branches(...)
list_files(...)

list_projects(...)  
new_issue(...)  
project_connection(...)  
proj_req(...)  
push_file(...)  
reopen_issue(...)  
repository(...)  
to_issue_id(...)  
unassign_issue(...)  

Arguments  
... Parameters to the new function  

Value  
Warning for deprecated functions and output depending on the superseeding function.  

Details  

<table>
<thead>
<tr>
<th>Argument</th>
<th>New Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>archive</td>
<td>gl_archive</td>
</tr>
<tr>
<td>assign_issue</td>
<td>gl_assign_issue</td>
</tr>
<tr>
<td>close_issue</td>
<td>gl_close_issue</td>
</tr>
<tr>
<td>comment_commit</td>
<td>gl_comment_commit</td>
</tr>
<tr>
<td>comment_issue</td>
<td>gl_comment_issue</td>
</tr>
<tr>
<td>create_branch</td>
<td>gl_create_branch</td>
</tr>
<tr>
<td>create_merge_request</td>
<td>gl_create_merge_request</td>
</tr>
<tr>
<td>delete_branch</td>
<td>gl_delete_branch</td>
</tr>
<tr>
<td>edit_commit_comment</td>
<td>gl_edit_commit_comment</td>
</tr>
<tr>
<td>edit_issue</td>
<td>gl_edit_issue</td>
</tr>
<tr>
<td>edit_issue_comment</td>
<td>gl_edit_issue_comment</td>
</tr>
<tr>
<td>file_exists</td>
<td>gl_file_exists</td>
</tr>
<tr>
<td>get_comments</td>
<td>gl_get_comments</td>
</tr>
<tr>
<td>get_commit_comments</td>
<td>gl_get_commit_comments</td>
</tr>
<tr>
<td>get_commits</td>
<td>gl_get_commits</td>
</tr>
<tr>
<td>get_diff</td>
<td>gl_get_diff</td>
</tr>
<tr>
<td>get_file</td>
<td>gl_get_file</td>
</tr>
<tr>
<td>get_issue</td>
<td>gl_get_issue</td>
</tr>
<tr>
<td>get_issue_comments</td>
<td>gl_get_issue_comments</td>
</tr>
<tr>
<td>get_issues</td>
<td>gl_list_issues</td>
</tr>
</tbody>
</table>
Description

List of old and new function names.

Format

A data frame with 33 rows and 2 variables

Arguments

key: option name
value: option value
Details

Options accounted for by gitlabr:

- `gitlabr.main`: Name of the main branch of your repository. Default to "main" in functions.

Value

Used for side effect. Populates user `options()`

Examples

```r
# Principal branch is called "master"
gitlabr_options_set("gitlabr.main", "master")
# Go back to default option (default branch will be "main")
gitlabr_options_set("gitlabr.main", NULL)
```

---

**glLoginInput**  
*Shiny module to login to GitLab API*

Description

The UI contains a login and a password field as well as an (optional) login button. The server side function returns a reactive GitLab connection, just as `gl_connection()` and `gl_project_connection()`.

Usage

```r
glLoginInput(id, login_button = TRUE)
```

```r
glReactiveLogin(
  input,
  output,
  session,
  gitlab_url,
  project = NULL,
  api_version = 4,
  success_message = "GitLab login successful!",
  failure_message = "GitLab login failed!",
  on_error = function(...) {
    stop(failure_message)
  }
)
```

Arguments

- `id`: shiny namespace for the login module
- `login_button`: whether to show a login button (TRUE) or be purely reactive (FALSE)
- `input`: from shinyServer function, usually not user provided
output from shinyServer function, usually not user provided
session from shinyServer function, usually not user provided
gitlab_url root URL of GitLab instance to login to
project if not NULL, a codegl_project_connection is created to this project
api_version A character with value either "3" or "4" to specify the API version that should be used
success_message message text to be displayed in the UI on successful login
failure_message message text to be displayed in the UI on login failure in addition to HTTP status
on_error function to be returned instead of GitLab connection in case of login failure

Details

glLoginInput is supposed to be used inside a shinyUI, while glReactiveLogin is supposed to be passed on to shiny::callModule()

Value
An input or output element for use in shiny UI.

gl_archive Archive a repository

Description
Archive a repository

Usage

gl_archive(project, ...)

Arguments

  project Project name or id
  ...

  further parameters passed on to gitlab() API call, may include parameter sha for specifying a commit hash

Value
If save_to_file is NULL, a raw vector of the archive, else the path to the saved archived file
### Examples

```r
## Not run:
set_gitlab_connection(
  gitlab_url = "https://gitlab.com",
  private_token = Sys.getenv("GITLAB_COM_TOKEN")
)
gl_archive(project = "<<your-project-id>>", save_to_file = "example-project.zip")

## End(Not run)
```

---

### gl_ci_job

**Define GitLab CI jobs content**

#### Description

Exploration of job content is deprecated as of `gitlabr` 1.1.7. Content of .gitlab-ci.yml file is now created using templates with `use_gitlab_ci(type = "check-coverage-pkgdown")`. See `use_gitlab_ci()`.

#### Usage

```r
gl_ci_job()
```

#### Value

Creates the content of a .gitlab-ci.yml file as character.

#### See Also

`use_gitlab_ci()`

#### Examples

```r
## Not run:
# Deprecated
gl_ci_job()

## End(Not run)
```
**gl_connection**

*Connect to a specific GitLab instance API*

**Description**

Creates a function that can be used to issue requests to the specified GitLab API instance with the specified user private token and (for `gl_project_connection`) only to a specified project.

**Usage**

```r
gl_connection(
  gitlab_url,
  private_token,
  api_version = 4,
  api_location = paste0("/api/v", api_version, "/")
)
```

```r
gl_project_connection(
  gitlab_url,
  project,
  private_token,
  api_version = 4,
  api_location = paste0("/api/v", api_version, "/")
)
```

**Arguments**

- `gitlab_url`: URL to the GitLab instance (e.g. https://gitlab.myserver.com)
- `private_token`: private_token with which to identify. You can generate one in the web interface under GITLABINSTANCEURL/~profile/personal_access_tokens.html when logged on.
- `api_version`: Currently "4" for the latest GitLab API version. See Details section on API versions.
- `api_location`: location of the GitLab API under the `gitlab_url`, usually and by default "/api/$api_version/"
- `project`: id or name of project to issue requests to

**Details**

The returned function should serve as the primary way to access the GitLab API in the following. It can take vector/character arguments in the same way as the function `gitlab()` does, as well as the convenience functions provided by this package or written by the user. If it is passed such that function it calls it with the arguments provided in ... and the GitLab URL, api location and private_token provided when creating it via `gl_connection`.

Note: currently GitLab API v4 is supported. GitLab API v3 is no longer supported, but you can give it a try.
Value

A function to access a specific GitLab API as a specific user, see details

API versions

"v4" is the standard API since GitLab version 9.0 and only this version is officially supported by gitlabr since version 1.1.6. "v3" as a parameter value is not removed, since for many instances, gitlabr code will still work if you try.

Examples

```r
## Not run:
# Set the connection for the session
set_gitlab_connection("https://gitlab.com", private_token = Sys.getenv("GITLAB_COM_TOKEN"))
# Get list of projects
gl_list_projects(max_page = 1)
# Unset the connection for the session
unset_gitlab_connection()

# Set connection for a specific project
my_project <- gl_project_connection(
  gitlab_url = "https://gitlab.com",
  project = 1234,
  private_token = Sys.getenv("GITLAB_COM_TOKEN")
)
# List files of a project
my_project_list_files <- my_project(gl_list_files, max_page = 1)

## End(Not run)
```

---

**gl_create_merge_request**

*Manage merge requests*

**Description**

Manage merge requests

**Usage**

```r
gl_create_merge_request(
  project,
  source_branch,
  target_branch = get_main(),
  title,
  description,
  ...
)```
gl_create_merge_request

)

gl_edit_merge_request(project, merge_request_iid, ...)

gl_close_merge_request(project, merge_request_iid)


gl_delete_merge_request(project, merge_request_iid, ...)


gl_list_merge_requests(project, ...)

Arguments

project name or id of project (not repository!)
source_branch name of branch to be merged
target_branch name of branch into which to merge
title title of the merge request
description description text for the merge request...

... passed on to `gitlab()`. Might contain more fields documented in GitLab API

doc.

merge_request_iid iid of the merge request

Value

Tibble of created or remaining merge requests of the project with informative variables.

Examples

## Not run:
set_gitlab_connection(
  gitlab_url = "https://gitlab.com",
  private_token = Sys.getenv("GITLAB_COM_TOKEN")
)

# Create MR and get its information
mr_infos <- gl_create_merge_request(project = <<your-project-id>>,
  source_branch = "my-extra-branch",
  title = "Merge extra to main", description = "These modifications are wonderful"
)

# List all opened MR
gl_list_merge_requests(project = <<your-project-id>>, status = "opened")

# Edit MR created
gl_edit_merge_request(project = <<your-project-id>>, merge_request_iid = mr_infos$iid,
  assignee_id = "<user-id>"
)

# Close MR
gl_close_merge_request(project = <<your-project-id>>, merge_request_iid = mr_infos$iid)

# Delete MR as it never existed

gl_delete_merge_request(project = <<your-project-id>>, merge_request_iid = mr_infos$iid)

## End(Not run)
**gl_get_comments**  
*Get the comments/notes of a commit or issue*

### Description
Get the comments/notes of a commit or issue

### Usage
```r
gl_get_comments(project, object_type = "issue", id, note_id = c(), ...)

gl_get_issue_comments(project, id, ...)

gl_get_commit_comments(project, id, ...)

gl_comment_commit(project, id, text, ...)

gl_comment_issue(project, id, text, ...)

gl_edit_comment(project, object_type, text, ...)

gl_edit_issue_comment(project, ...)

gl_edit_commit_comment(project, ...)
```

### Arguments
- **project**: project name or id
- **object_type**: one of "issue" or "commit". Snippets and merge_requests are not implemented yet.
- **id**: id of object:
  - commits: sha
  - issues notes/comments:
    - (project-wide) id for api version 4,
    - (global) iid for api version 3
- **note_id**: id of note
- **text**: Text of comment/note to add or edit (translates to GitLab API note/body respectively)

### Details
- **gl_comment_commit**: might also contain path, line and line_type (old or new) to attach the comment to a specific in a file. See https://docs.gitlab.com/ce/api/commits.html
- **gl_get_issue_comments**: might also contain comment_id to get a specific comment of an issue.
gl_get_commits

Value
Tibble of comments with descriptive variables.

Examples
```r
## Not run:
# fill in login parameters
set_gitlab_connection(gitlab_url = "https://gitlab.com", 
  private_token = Sys.getenv("GITLAB_COM_TOKEN"))
gl_get_comments(project = "<<your-project-id>>", object_type = "issue", 1)
gl_get_comments(project = "<<your-project-id>>", "commit", 
  id = "8ce5ef240123cd78c1537991e5de8d8323666b15")
gl_comment_issue(project = "<<your-project-id>>", 1, 
  text = "Almost done!"
)
```
```
## End(Not run)
```

Description
Get commits and diff from a project repository

Usage
```r
gl_get_commits(project, commit_sha = c(), ...)
gl_get_diff(project, commit_sha, ...)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>project</td>
<td>project name or id</td>
</tr>
<tr>
<td>commit_sha</td>
<td>if not null, get only the commit with the specific hash; for gl_get_diff() this must be specified</td>
</tr>
<tr>
<td>...</td>
<td>passed on to gitlab() API call, may contain ref_name for specifying a branch or tag to list commits of</td>
</tr>
</tbody>
</table>

Value
Tibble of commits or diff of the branch with informative variables.

Examples
```r
## Not run:
my_commits <- gl_get_commits("<<your-project-id>>")
gl_get_commits("<<your-project-id>>", my_commits$id[1])
```
```
## End(Not run)
```
gl_get_project_id  Get a project id by name

Description
Get a project id by name

Usage
gl_get_project_id(project_name, ...)

Arguments
project_name  project name
...  passed on to gitlab()

Details
Number of pages searched is limited to (per_page =) 20 * (max_page =) 10 by default. If the project_name is an old project lost in a big repository (position > 200), gl_get_project_id() may not find the project id.

Value
Integer. ID of the project if found.

Examples
## Not run:
gl_get_project_id("<<your-project-name>>")
## End(Not run)

---

gl_list_branches  List, create and delete branches

Description
List, create and delete branches
List, create and delete branches
%gl_list_branches

Usage

gl_list_branches(project, ...)

gl_get_branch(project, branch, ...)

gl_create_branch(project, branch, ref = get_main(), ...)

Arguments

project     name or id of project (not repository!)
...
branch      name of branch to create / delete / get information
ref         ref name of origin for newly created branch

Value

Tibble of branches available in the project with descriptive variables

Examples

## Not run:
set_gitlab_connection(gitlab_url = "https://gitlab.com",
  private_token = Sys.getenv("GITLAB_COM_TOKEN"))
project_id <- ... ## Fill in your project ID

# List branches of the project
gl_list_branches(project = "<<your-project-id>>")
# Create branch "new_feature"
gl_create_branch(project = "<<your-project-id>>",
  branch = "new_feature")
# Confirm that the branch was created
gl_get_branch("<<your-project-id>>", branch = "new_feature")
# List all branches - this may take some time before your branch really appears there
gl_list_branches(project = "<<your-project-id>>")
# Delete branch again
gl_delete_branch(project = "<<your-project-id>>",
  branch = "new_feature")
# Check that we're back where we started
gl_list_branches(project = "<<your-project-id>>")

## End(Not run)
gl_list_issues Get issues of a project or user

Description

Get issues of a project or user

Usage

```r
gl_list_issues(
  project = NULL,  # project name or id, may be null for all issues created by user. If using the ID, set it as numeric, otherwise this is used as project name.
  issue_id = NULL,  # optional issue id (projectwide; for API v3 only you can use global iid when api_version is 3)
  verb = httr::GET,  # ignored; all calls with this function will have gitlab()’s default verb httr::GET
  api_version = 4,  # a switch to force deprecated GitLab API v3 behavior that allows filtering by global iid. If 3 filtering happens by global iid, if false, it happens by projectwide ID. For API v4, this must be FALSE (default)
  ...  # further parameters passed on to gitlab(), may be state, labels, issue id, ...
)
```

```r
gl_get_issue(project, issue_id, ...)
```

Arguments

- `project`: project name or id, may be null for all issues created by user. If using the ID, set it as numeric, otherwise this is used as project name.
- `issue_id`: optional issue id (projectwide; for API v3 only you can use global iid when api_version is 3)
- `verb`: ignored; all calls with this function will have `gitlab()`'s default verb `httr::GET`
- `api_version`: a switch to force deprecated GitLab API v3 behavior that allows filtering by global iid. If 3 filtering happens by global iid, if false, it happens by projectwide ID. For API v4, this must be FALSE (default)
- `...`: further parameters passed on to `gitlab()`, may be state, labels, issue id, ...

Details

`gl_get_issue` provides a wrapper with swapped arguments for convenience, esp. when using a project connection

Value

Tibble of issues of the project with descriptive variables.

Examples

```r
## Not run:
# Set the connection for the session
set_gitlab_connection(
  gitlab_url = test_url,
  private_token = test_private_token
```
gl_list_projects

## Description

List projects information

## Usage

```r
gl_list_projects(...)  
gl_get_projects(...)  
gl_list_user_projects(user_id, ...)  
gl_list_group_projects(group_id, ...)  
gl_get_project(project, ...)
```

## Arguments

- `...`: passed on to `gitlab()`
- `user_id`: id of the user to list project from
- `group_id`: id of the group to list project from
- `project`: project name or id

## Details

- `gl_list_projects()` is an alias for `gl_get_projects()`

## Value

tibble of each project with corresponding information
Examples

```r
## Not run:
set_gitlab_connection(
  gitlab_url = "https://gitlab.com",
  private_token = Sys.getenv("GITLAB_COM_TOKEN")
)
# List all projects
gl_get_projects(max_page = 1)
# List users projects
gl_list_user_projects(user_id = "<<user-id>>", max_page = 1)
# List group projects
gl_list_group_projects(group_id = "<<group-id>>", max_page = 1)

## End(Not run)
```

---

### gl_new_issue

#### Post a new issue or edit one

**Description**

Post a new issue or edit one

**Usage**

- `gl_new_issue(project, title, ...)`
- `gl_create_issue(project, title, ...)`
- `gl_edit_issue(project, issue_id, api_version = 4, ...)`
- `gl_close_issue(project, issue_id, ...)`
- `gl_reopen_issue(project, issue_id, ...)`
- `gl_assign_issue(project, issue_id, assignee_id = NULL, ...)`
- `gl_unassign_issue(project, issue_id, ...)`
- `gl_delete_issue(project, issue_id, ...)`

**Arguments**

- **project**: project where the issue should be posted
- **title**: title of the issue
- **...**: further parameters passed to the API call, may contain description, assignee_id, milestone_id, labels, state_event (for edit_issue).
### Issue ID

**issue_id**

issue id (projectwide; for API v3 only you can use global iid when `force_api_v3` is TRUE although this is not recommended!)

**api_version**

a switch to force deprecated GitLab API v3 behavior that allows filtering by global iid. If 3 filtering happens by global iid, if false, it happens by projectwide ID. For API v4, this must be 4 (default)

**assignee_id**

numeric id of users as returned in `/users` API request

### Value

Tibble with the created or remaining issues and descriptive variables.

### Examples

```r
## Not run:
# create an issue
new_issue_infos <- gl_create_issue(project = "<<your-project-id>>", "A simple issue")
new_issue_iid <- new_issue_infos$iid[1]
## close issue
gl_close_issue("<<your-project-id>>", new_issue_iid)
## reopen issue
gl_reopen_issue("<<your-project-id>>", new_issue_iid)
## edit its description
gl_edit_issue("<<your-project-id>>", new_issue_iid, description = "This is a test")
## assign it
gl_assign_issue("<<your-project-id>>", new_issue_iid, assignee_id = "<<user-id>>")
## unassign it
gl_unassign_issue("<<your-project-id>>", new_issue_iid)
## Delete issue as if it never existed
gl_delete_issue("<<your-project-id>>", new_issue_iid)
## End(Not run)
```
Arguments

- **name**: of the new project. The name of the new project. Equals path if not provided.
- **path**: to the new project if name is not provided. Repository name for new project. Generated based on name if not provided (generated as lowercase with dashes).
- **project**: The ID or URL-encoded path of the project.

Details

You can use extra parameters as proposed in the GitLab API:

- **namespace_id**: Namespace for the new project (defaults to the current user’s namespace).

Value

A tibble with the project information. `gl_delete_project()` returns an empty tibble.

Examples

```r
## Not run:
set_gitlab_connection(
  gitlab_url = "https://gitlab.com",
  private_token = Sys.getenv("GITLAB_COM_TOKEN")
)
# Create new project
gl_new_project(name = "toto")
# Edit existing project
gl_edit_project(project = "<<your-project-id>>", default_branch = "main")
# Delete project
gl_delete_project(project = "<<your-project-id>>")
## End(Not run)
```

---

**gl_pipelines**

*Access the GitLab CI builds*

**Description**

List the jobs with `gl_jobs`, the pipelines with `gl_pipelines` or download the most recent artifacts archive with `gl_latest_build_artifact`. For every branch and job combination only the most recent artifacts archive is available. `gl_builds` is the equivalent for GitLab API v3.
Usage

```r
gl_pipelines(project, ...)

gl_jobs(project, ...)

gl_builds(project, api_version = 4, ...)

gl_latest_build_artifact(
  project,
  job,
  ref_name = get_main(),
  save_to_file = tempfile(fileext = ".zip"),
  ...
)
```

Arguments

- `project`: project name or id, required
- `...`: passed on to `gitlab()` API call
- `api_version`: Since `gl_builds` is no longer working for GitLab API v4, this must be set to "3" in order to avoid deprecation warning and HTTP error. It currently default to "4" with deprecation message.
- `job`: Name of the job to get build artifacts from
- `ref_name`: name of ref (i.e. branch, commit, tag)
- `save_to_file`: either a path where to store .zip file or NULL if raw should be returned

Value

returns the file path if `save_to_file` is TRUE, or the archive as raw otherwise.

Examples

```r
## Not run:
# connect as a fixed user to a GitLab instance
set_gitlab_connection(
  gitlab_url = "https://gitlab.com",
  private_token = Sys.getenv("GITLAB_COM_TOKEN"))

# Get pipelines and jobs information
gl_pipelines(project = "<<your-project-id>>")
gl_jobs(project = "<<your-project-id>>")
gl_latest_build_artifact(project = "<<your-project-id>>", job = "build")

## End(Not run)
```
**gl_proj_req**  
*Create a project specific request*

**Description**  
Prefixes the request location with "project/:id" and automatically translates project names into ids

**Usage**  
`gl_proj_req(project, req, ...)`

**Arguments**  
- `project`: project name or id  
- `req`: character vector of request location  
- `...`: passed on to `gl_get_project_id()`

**Value**  
A vector of character to be used as request for functions involving projects

**Examples**

```r
## Not run:
gl_proj_req("test_project<<your-project-id>>, req = "merge_requests")
```

**gl_push_file**  
*Upload, delete a file to a GitLab repository*

**Description**  
If the file already exists, it is updated/overwritten by default

**Usage**

```r
gl_push_file(
  project,
  file_path,
  content,
  commit_message,
  branch = get_main(),
  overwrite = TRUE,
  ...
)
```

```r
gl_delete_file(project, file_path, commit_message, branch = get_main(), ...)
```
**Arguments**

- **project**
  Project name or id

- **file_path**
  Path where to store file in gl_repository. If in subdirectory, the parent directory should exist.

- **content**
  Character of length 1. File content (text)

- **commit_message**
  Message to use for commit with new/updated file

- **branch**
  Name of branch where to append newly generated commit with new/updated file

- **overwrite**
  Whether to overwrite files that already exist

... passed on to `gitlab()`

**Value**

Returns a tibble with changed branch and path (0 rows if nothing was changed, since overwrite is FALSE)

**Examples**

```r
## Not run:
# Create fake dataset
tmpfile <- tempfile(fileext = ".csv")
write.csv(mtcars, file = tmpfile)
# Push content to repository with a commit
gl_push_file(
  project = <<your-project-id>>,
  file_path = "test_data.csv",
  content = paste(readLines(tmpfile), collapse = "\n"),
  commit_message = "New test data")
## End(Not run)
```

---

**gl_repository**

Access to repository files in GitLab

**Description**

Access to repository files in GitLab

For `gl_file_exists` dots are passed on to `gl_list_files()` and GitLab API call

Get a file from a GitLab repository
Usage

```r
gl_repository(project, req = c("tree"), ref = get_main(), ...)

gl_list_files(project, ref = get_main(), ...)

gl_file_exists(project, file_path, ref, ...)

gl_get_file(
  project,
  file_path,
  ref = get_main(),
  to_char = TRUE,
  api_version = 4,
  ...
)
```

Arguments

- `project`: name or id of project (not repository!)
- `req`: request to perform on repository (everything after `/repository/` in GitLab API, as vector or part of URL)
- `ref`: name of ref (commit branch or tag)
- `...`: passed on to `gitlab()` API call
- `file_path`: path to file
- `to_char`: flag if output should be converted to char; otherwise it is of class raw
- `api_version`: a switch to force deprecated GitLab API v3 behavior. See details section "API version" of `gl_connection()`

Value

Tibble of files available in the branch with descriptive variables.

Examples

```r
## Not run:
# Set GitLab connection for examples
set_gitlab_connection(
  gitlab_url = "https://gitlab.com",
  private_token = Sys.getenv("GITLAB_COM_TOKEN"))

# Access repository
# _All files
gl_repository(project = <<your-project-id>>) 
# _All contributors
gl_repository(project = <<your-project-id>>, "contributors")
# _List files
gl_list_files(project = <<your-project-id>>) 
# _Get content of one file
```
gl_to_issue_id

This function is only intended to be used with GitLab API v3. With v4, the global iid is no longer functional.

Usage

```r
gl_to_issue_id(project, issue_id, api_version = 3, ...)
```

Arguments

- `project`: project name or id
- `issue_id`: projectwide issue id (as seen by e.g. GitLab website users)
- `api_version`: Since this function is no longer necessary for GitLab API v4, this must be set to 3 in order to avoid deprecation warning and HTTP error.
- `...`: passed on to `gitlab()`

Value

Global GitLab API issue id

Examples

```r
## Not run:
gl_to_issue_id(project = "<my-project>", issue_id = 1, api_version = 3)
## End(Not run)
```
set_gitlab_connection  Get/set a GitLab connection for all calls

Description
This sets the default value of gitlab_con in a call to `gitlab()`

Usage
```r
set_gitlab_connection(gitlab_con = NULL, ...)
get_gitlab_connection()
unset_gitlab_connection()
```

Arguments
- `gitlab_con` A function used for GitLab API calls, such as `gitlab()` or as returned by `gl_connection()`.
- `...` if `gitlab_con` is NULL, a new connection is created using `gl_connection()`

Value
Used for side effects. Set or unset global connection settings.

Examples
```r
## Not run:
set_gitlab_connection("https://gitlab.com", private_token = Sys.getenv("GITLAB_COM_TOKEN"))
## End(Not run)
```

use_gitlab_ci  Add .gitlab-ci.yml file in your current project from template

Description
Add .gitlab-ci.yml file in your current project from template
use_gitlab_ci

Usage

use_gitlab_ci(
  image = "rocker/verse:latest",
  repo_name = "https://packagemanager.rstudio.com/all/__linux__/focal/latest",
  path = ".gitlab-ci.yml",
  overwrite = TRUE,
  add_to_Rbuildignore = TRUE,
  type = "check-coverage-pkgdown"
)

Arguments

image Docker image to use in GitLab ci. If NULL, not specified!
repo_name REPO_NAME environment variable for R CRAN mirror used
path destination path for writing GitLab CI yml file
overwrite whether to overwrite existing GitLab CI yml file
add_to_Rbuildignore add CI yml file and cache path used inside the CI workflow to .Rbuildignore?
type type of the CI template to use

Details

Types available are:

- "check-coverage-pkgdown": Check package along with Code coverage with covr and pkgdown site on GitLab Pages
- "check-coverage-pkgdown-renv": Check package built in a fixed renv state along with Code coverage with covr and pkgdown site on GitLab Pages.
- "bookdown": Build bookdown HTML and PDF site on GitLab Pages
- "bookdown-production": Build bookdown HTML and PDF site on GitLab Pages. Where default page is for branch named 'production' and "dev/" sub-folder is for 'main' (or 'master') branch.

Value

Used for side effects. Creates a .gitlab-ci.yml file in your directory.

Examples

# Create in another directory
use_gitlab_ci(image = "rocker/verse:latest", path = tempfile(fileext = ".yml"))
## Not run:
# Create in your current project with template for packages checking
use_gitlab_ci(image = "rocker/verse:latest", type = "check-coverage-pkgdown")
## End(Not run)
INDEX

(gl_get_comments), 14
(gl_get_project (gl_list_projects)), 19
(gl_get_project_id), 16
(gl_get_project_id()), 24
(gl_get_projects (gl_list_projects)), 19
(gl_jobs (gl_pipelines)), 22
(gl_latest_build_artifact
    (gl_pipelines)), 22
(gl_list_branches), 16
(gl_list_files (gl_repository)), 25
(gl_list_files()), 25
(gl_list_group_projects
    (gl_list_projects)), 19
(gl_list_issues), 18
(gl_list_merge_requests
    (gl_create_merge_request)), 12
(gl_list_projects), 19
(gl_list_user_projects
    (gl_list_projects)), 19
(gl_new_issue), 20
(gl_new_project), 21
(gl_pipelines), 22
(gl_proj_req), 24
(gl_project_connection), 9
(gl_project_connection (gl_connection)), 11
(gl_project_connection()), 8
(gl_push_file), 24
(gl_reopen_issue (gl_new_issue)), 20
(gl_repository), 25
(gl_to_issue_id), 27
(gl_unassign_issue (gl_new_issue)), 20
(set_gitlab_connection), 28
(shiny::callModule()), 9
(to_issue_id (gitlabr-deprecated)), 4
(unassign_issue (gitlabr-deprecated)), 4
(unset_gitlab_connection
    (set_gitlab_connection)), 28
(use_gitlab_ci), 28
(use_gitlab_ci()), 10

httr::DELETE(), 3
httr::GET(), 3
httr::POST(), 3
httr::PUT(), 3

list_branches (gitlabr-deprecated), 4
list_files (gitlabr-deprecated), 4
list_projects (gitlabr-deprecated), 4
new_issue (gitlabr-deprecated), 4

options(), 8
proj_req (gitlabr-deprecated), 4

set_gitlab_connection, 28