Package: gitlabr (via r-universe)

June 16, 2024

Title Access to the 'GitLab' API

Version 2.1.0.9000

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://thinkr-open.github.io/gitlabr/, https://github.com/ThinkR-open/gitlabr

BugReports https://github.com/ThinkR-open/gitlabr/issues

Depends R (>= 3.1.2)

Imports arpr, base64enc, dplyr (>= 0.4.3), httr (>= 1.1.0), magrittr, purrr (>= 0.2.2), stringr, tibble (>= 1.1), tidyr, utils

Suggests DT, knitr, rmarkdown, shiny (>= 0.13.0), testthat (>= 3.0.0), yaml

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

Repository https://thinkr-open.r-universe.dev

RemoteUrl https://github.com/ThinkR-open/gitlabr

RemoteRef HEAD

RemoteSha 15d93106d1b2f80aa642f69a5fffb1e89356ffd2

Contents

`gitlab` *Request GitLab API*

Description

This is 'gitlabr' 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' convenience wrappers or write your own. See the 'gitlabr' vignette for more information on this.

Usage

```
gitlab(
  req,
  api_root,
  verb = httr::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 `get_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:

```
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

```
## 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

List of deprecated functions that will be removed in future versions.

Usage

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

gl_ci_job()
```

Arguments

project	id (preferred way) or name of the project. Not repository name.
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.´
...	Parameters to the new function

Value

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

Details

gl_builds	in favour of gl_pipelines
gl_ci_job	in favour of use_gitlab_ci

gitlabr_options_set *Set gitlabr options*

Description

Set gitlabr options

Usage

```
gitlabr_options_set(key, value)
```

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

```
# 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

```
glLoginInput(id, login_button = TRUE)

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 [gl_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	id (preferred way) or name of the project. Not repository name.
...	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

```
## 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_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

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

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 (preferred way) or name of the project. Not repository name.

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

```
## 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

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

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	id (preferred way) or name of the project. Not repository name.
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

```
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`	id (preferred way) or name of the project. Not repository name.
`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
...	passed on to gitlab() API call. See Details.
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.

Value

Tibble of comments with descriptive variables.

Examples

```
## 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)
```

---

gl_get_commits *Get commits and diff from a project repository*

Description

Get commits and diff from a project repository

Usage

```
gl_get_commits(project, commit_sha = c(), ...)

gl_get_diff(project, commit_sha, ...)
```

Arguments

project	id (preferred way) or name of the project. Not repository name.
commit_sha	if not null, get only the commit with the specific hash; for gl_get_diff() this must be specified
...	passed on to gitlab() API call, may contain ref_name for specifying a branch or tag to list commits of

Value

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

Examples

```
## 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_group_id *Get a group id by name*

Description

Get a group id by name

Usage

```
gl_get_group_id(group_name, ...)
```

Arguments

group_name	group name
...	passed on to gitlab()

Details

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

Value

Integer. ID of the group if found.

Examples

```
## Not run:
gl_get_group_id("<<your-group-name>>")

## 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_group_req *Create a group specific request*

Description

Prefixes the request location with "groups/:id/subgroups" and automatically translates group names into ids

Usage

```
gl_group_req(group, ...)
```

Arguments

group	The ID, name or URL-encoded path of the group
...	passed on to gl_get_group_id()

Value

A vector of character to be used as request for functions involving groups

Examples

```
## Not run:
gl_group_req("test_group" = "<<your-group-id>>")

## End(Not run)
```

gl_list_branches *List, create and delete branches*

Description

List, create and delete branches

List, create and delete branches

Usage

```
gl_list_branches(project, ...)

gl_get_branch(project, branch, ...)

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

gl_delete_branch(project, branch, ...)
```

Arguments

project	id (preferred way) or name of the project. Not repository name.
...	passed on to gitlab()
branch	name of branch to create / delete / get information
ref	ref name of origin for newly created branch. Default to 'main'.

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_files *List of files in a folder*

Description

List of files in a folder

Usage

```
gl_list_files(project, path = "", ref = get_main(), ...)
```

Arguments

project	id (preferred way) or name of the project. Not repository name.
path	path of the folder
ref	name of ref (commit branch or tag). Default to 'main'.
...	passed on to gitlab() API call

Value

Tibble of files available in the branch with descriptive variables.

Examples

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

gl_list_files(project = "<<your-project-id>>", path = "<<path-to-folder>>")

## End(Not run)
```

---

gl_list_groups *List and manage groups*

Description

List and manage groups

Usage

```
gl_list_groups(...)

gl_list_sub_groups(group, ...)
```

Arguments

...	passed on to gitlab()
group	The ID, name or URL-encoded path of the group

Details

When using `gl_list_sub_groups()`, if you request this list as:

- An unauthenticated user, the response returns only public groups.
- An authenticated user, the response returns only the groups you're a member of and does not include public groups.

Value

tibble of each group with corresponding information

Examples

```
## Not run:
set_gitlab_connection(
  gitlab_url = "https://gitlab.com",
  private_token = Sys.getenv("GITLAB_COM_TOKEN")
)
# List all groups
gl_list_groups(max_page = 1)
# List sub-groups of a group
gl_list_sub_groups(group_id = "<<group-id>>", max_page = 1)

## End(Not run)
```

`gl_list_group_members` *List members of a specific group*

Description

List members of a specific group

Usage

```
gl_list_group_members(group, ...)
```

Arguments

`group`	The ID or URL-encoded path of the group
`...`	passed on to `gitlab()` API call for "groups"

Value

A tibble with the group members information

Examples

```
## Not run:
set_gitlab_connection(
  gitlab_url = "https://gitlab.com",
  private_token = Sys.getenv("GITLAB_COM_TOKEN")
)
gl_list_group_members(group = "<<your-group-id>>")

## End(Not run)
```

`gl_list_issues` *Get issues of a project or user*

Description

Get issues of a project or user

Usage

```
gl_list_issues(
  project = NULL,
  issue_id = NULL,
  verb = httr::GET,
  api_version = 4,
  ...
)

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

Arguments

`project`	id (preferred way) or name of the project. Not repository name. May be null for all issues created by user.
`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

```
## Not run:
# Set the connection for the session
set_gitlab_connection(
  gitlab_url = test_url,
  private_token = test_private_token
)
# list issues
gl_list_issues("<<your-project-id>>", max_page = 1)
# list opened issues
gl_list_issues("<<your-project-id>>", state = "opened")
# Get one issue
gl_get_issue("<<your-project-id>>", issue_id = 1)
# Create new issue
gl_new_issue("<<your-project-id>>",
  title = "Implement new feature",
  description = "It should be awesome."
)
# Assign user to issue 1
gl_assign_issue("<<your-project-id>>", issue_id = 1, assignee_id = "<<user-id>>")

## End(Not run)
```

---

gl_list_projects *List projects information*

Description

List projects information

Usage

```
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	id (preferred way) or name of the project. Not repository name.

Details

gl_list_projects() is an alias for gl_get_projects()

Value

tibble of each project with corresponding information

Examples

```
## 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_list_project_members

List members of a specific project

Description

List members of a specific project

Usage

```
gl_list_project_members(project, ...)
```

Arguments

project	id (preferred way) or name of the project. Not repository name.
...	passed on to gitlab() API call for "project"

Value

A tibble with the project members information

Examples

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

## End(Not run)
```

---

gl_new_group *Manage groups*

Description

Manage groups

Usage

```
gl_new_group(name, path, visibility = c("private", "internal", "public"), ...)

gl_new_subgroup(
  name,
  path,
  visibility = c("private", "internal", "public"),
  group,
  ...
)

gl_edit_group(group, ...)

gl_delete_group(group)
```

Arguments

name	Name of the new group
path	Path to the new group
visibility	Visibility of the new subgroup: "public", "private"...
...	passed on to gitlab() API call for "Create group"
group	The ID, name or URL-encoded path of the group

Details

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

Note that on GitLab SaaS, you must use the GitLab UI to create groups without a parent group. You cannot use the API with `gl_new_group()` to do this, but you can use `gl_new_subgroup()`.

Value

A tibble with the group information. `gl_delete_group()` returns an empty tibble.

Examples

```
## Not run:
set_gitlab_connection(
  gitlab_url = "https://gitlab.com",
  private_token = Sys.getenv("GITLAB_COM_TOKEN")
)
# Create new group
gl_new_group(name = "mygroup")
# Create new subgroup
gl_new_subgroup(name = "mysubgroup", group = "mygroup")
# Edit existing group
gl_edit_group(group = "<<your-group-id>>", default_branch = "main")
# Delete group
gl_delete_group(group = "<<your-group-id>>")

## 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	id (preferred way) or name of the project. Not repository name.
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 (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

```
## 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
## (please note that you must have "Owner" role on the GitLab project)
gl_delete_issue("<<your-project-id>>", new_issue_iid)

## End(Not run)
```

gl_new_project *Manage projects*

Description

Manage projects

Usage

```
gl_new_project(name, path, ...)

gl_edit_project(project, ...)

gl_delete_project(project)
```

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).
...	passed on to gitlab() API call for "Create project"
project	id (preferred way) or name of the project. Not repository name.

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

```
## 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.

Usage

```
gl_pipelines(project, ...)

gl_jobs(project, ...)

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

Arguments

`project`	id (preferred way) or name of the project. Not repository name.
`...`	passed on to `gitlab()` API call
`job`	Name of the job to get build artifacts from
`ref_name`	name of ref (i.e. branch, commit, tag). Default to 'main'.
`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

```
## 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`	id (preferred way) or name of the project. Not repository name.
`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

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

## End(Not run)
```

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

Description

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

Usage

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

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

Arguments

project	id (preferred way) or name of the project. Not repository name.
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

```
## 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

```
gl_repository(project, req = c("tree"), 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`	id (preferred way) or name of the project. Not repository name.
`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). Default to 'main'.
`...`	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

```
## 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")
# _Get content of one file
gl_get_file(project = "<<your-project-id>>", file_path = "README.md")
# _Test if file exists
gl_file_exists(
  project = "<<your-project-id>>",
  file_path = "README.md",
  ref = "main"
)

## End(Not run)
```

---

gl_to_issue_id *Translate projectwide issue id to global GitLab API issue id*

Description

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

Usage

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

Arguments

project	id (preferred way) or name of the project. Not repository name.
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

```
## Not run:
gl_to_issue_id(project = "<my-project>", issue_id = 1, api_version = 3)

## End(Not run)
```

---

multilist_to_tibble *Modify a multilist from API JSON output to a level 1 tibble*

Description

Modify a multilist from API JSON output to a level 1 tibble

Usage

```
multilist_to_tibble(the_list)
```

Arguments

the_list	list of element as issued from a API REST call

Value

a tibble with columns as the names of the list

Examples

```
reprex <- list(
  list(a = 1, b = list("email1", "email2", "email3"), c = list("3")),
  list(a = 5, b = list("email1"), c = list("4")),
  list(a = 3, b = NULL, c = list("3", "2"))
)

multilist_to_tibble(reprex)
```

---

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

```
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 used the parameters is ... using `gl_connection()`

Value

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

Examples

```
## 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

Usage

```
use_gitlab_ci(
  image = "rocker/verse:latest",
  path = ".gitlab-ci.yml",
  overwrite = TRUE,
  add_to_Rbuildignore = TRUE,
  type = "check-coverage-pkgdown",
  upgrade = TRUE
)
```

Arguments

image	Docker image to use in GitLab ci. If NULL, not specified!
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
upgrade	whether to upgrade the R packages to the latest version during the CI. Default to TRUE.

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 there will be a version of the book for each branch deployed. See https://github.com/statnmap/GitLab-Pages-Deploy for setup details.

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