
node-gitlab-2-github

Install

1. You need nodejs and npm installed
2. Clone this repo with `git clone https://github.com/piceaTech/node-gitlab-2-github.git`
3. `cd node-gitlab-2-github`
4. `npm i`

Preliminaries

Before using this script, you must mirror your GitLab repo to your new GitHub repo. This can be done with the following steps:

```
1 # Clone the repo from GitLab using the `--mirror` option. This is like
2 # `--bare` but also copies all refs as-is. Useful for a full backup/
   move.
3 git clone --mirror git@your-gitlab-site.com:username/repo.git
4
5 # Change into newly created repo directory
6 cd repo
7
8 # Push to GitHub using the `--mirror` option. The `--no-verify` option
   skips any hooks.
9 git push --no-verify --mirror git@github.com:username/repo.git
10
11 # Set push URL to the mirror location
12 git remote set-url --push origin git@github.com:username/repo.git
13
14 # To periodically update the repo on GitHub with what you have in
   GitLab
15 git fetch -p origin
16 git push --no-verify --mirror
```

After doing this, the autolinking of issues, commits, and branches will work. See **Usage** for next steps.

Usage

The user must be a member of the project you want to copy. This user must be the one

1. `cp sample_settings.ts settings.ts`

-
2. edit settings.ts
 3. run `npm run start`

Docker

If you don't have Node.js installed in your local environment and don't want to install it you can use the Dockerized approach.

1. Make sure that you have Docker installed in your computer. You can test running `docker version` in the terminal.
2. `cp sample_settings.ts settings.ts`
3. edit settings.ts
4. `docker build -t node-gitlab-2-github:latest .`, or, you can use `make build-image`
5. `docker run node-gitlab-2-github:latest`, or, you can use `make docker-run`

If you want to let it run in the background (detached mode), just use the following command:

1. `docker run -d node-gitlab-2-github:latest`

Docker with bind mounts

In order to optimize the usage of the dockerized application, one can use the `bind mounts` feature of Docker (Docker docs). This way, whenever you change the `settings.ts` file in the host environment it will change in the container filesystem as well.

The process to use this trick is pretty much the same we presented before, the only different is the addition of a flag in the docker command to tell it what is the directory/file to be bound.

1. Make sure that you have Docker installed in your computer. You can test running `docker version` in the terminal.
2. `cp sample_settings.ts settings.ts`
3. edit settings.ts
4. `docker build -t node-gitlab-2-github:latest .`, or, you can use `make build-image`
5. This command must work for **Linux** or **Mac**: `docker run --mount type=bind,source="$(pwd)/settings.ts",target="/app/settings.ts",readonly node-gitlab-2-github:latest`, or, you can use `make docker-run-bind`

-
- If you want to run this last command in the Windows environment, please consult the Docker documentation on how to solve the problem of the `pwd` command expanding incorrectly there
 - Docker documentation - Topics for windows.

Where to find info for the `settings.ts`

gitlab

gitlab.url The URL under which your gitlab instance is hosted. Default is the official <http://gitlab.com> domain.

gitlab.token Go to Settings / Access Tokens. Create a new Access Token with [api](#) and [read_repository](#) scopes and copy that into the `settings.ts`

gitlab.projectID Leave it null for the first run of the script. Then the script will show you which projects there are. Can be either string or number.

gitlab.listArchivedProjects When listing projects on the first run (`projectID = null`), include archived ones too. The default is *true*.

gitlab.sessionCookie GitLab's API does not allow downloading of attachments and only images can be downloaded using HTTP. To work around this limitation and enable binary attachments to be migrated one can use the session cookie set in the browser after logging in to the gitlab instance. The cookie is named [_gitlab_session](#).

github

github.baseUrl Where is the github instance hosted? The default is the official github.com domain

github.apiUrl Point this to the api. The default is api.github.com.

github.owner Under which organisation or user will the new project be hosted

github.ownerIsOrg A boolean indicator (default is *false*) to specify that the owner of this repo is an Organisation.

github.token Go to Settings / Developer settings / Personal access tokens. Generate a new token with `repo` scope and copy that into the `settings.ts`

github.token_owner Set to the user name of the user whose token is used (see above). This is required to determine whether the user running the migration is also the creator of comments and issues. If this is the case and `useIssueCreationAPI` is true (see below), the extra line specifying who created a comment or issue will not be added.

github.repo What is the name of the new repo

github.recreateRepo If true (default is false), we will try to delete the destination github repository if present, and (re)create it. The github token must be granted `delete_repo` scope. The newly created repository will be made private by default.

If you've set `github.recreateRepo` to true and the repo belongs to an Organisation, the `github.ownerIsOrg` flag **must** be set as true.

This is useful when debugging this tool or a specific migration. You will always be prompted for confirmation.

s3 (optional)

S3 can be used to store attachments from issues. If omitted, `has_attachment` label will be added to GitHub issue.

s3.accessKeyId and **s3.secretAccessKey** AWS credentials that are used to copy attachments from GitLab into the S3 bucket.

IAM User who owns these credential must have write permissions to the bucket.

s3.bucket Existing bucket, with an appropriate security policy. One possible policy is to allow public access.

s3.region Specify Region (example: us-west-1) of bucket list of regions

usermap

Maps the usernames from gitlab to github. If the assignee of the gitlab issue is equal to the one currently logged in github it will also get assigned without a usermap. The Mentions in issues will also be translated to the new github name.

projectmap

When one renames the project while transferring so that the projects don't lose their links to the mentioned issues.

conversion

conversion.useLowerCaseLabels If this is set to true (default) then labels from GitLab will be converted to lowercase in GitHub.

transfer

transfer.milestones If this is set to true (default) then the migration process will transfer milestones.

transfer.labels If this is set to true (default) then the migration process will transfer labels.

transfer.issues If this is set to true (default) then the migration process will transfer issues.

transfer.mergeRequests If this is set to true (default) then the migration process will transfer merge requests.

transfer.releases If this is set to true (default) then the migration process will transfer releases. Note that github api for releases is limited and hence this will only transfer the title and description of the releases and add them to github in chronological order, but it would not preserve the original release dates, nor transfer artefacts or assets.

dryRun

As default it is set to false. Doesn't fire the requests to github api and only does the work on the gitlab side to test for wonky cases before using up api-calls

useIssueImportAPI

Set to true (default) to enable using the GitHub preview API for importing issues. This allows setting the date for issues and comments instead of inserting an additional line in the body.

usePlaceholderIssuesForMissingIssues

If this is set to true (default) then the migration process will automatically create empty dummy issues for every 'missing' GitLab issue (if you deleted a GitLab issue for example). Those issues will be closed on Github and they ensure that the issue ids stay the same on both GitLab and Github.

usePlaceholderMilestonesForMissingMilestones If this is set to true (default) then the migration process will automatically create empty dummy milestones for every 'missing' GitLab milestone (if you deleted a GitLab milestone for example). Those milestones will be closed on Github and they ensure that the milestone ids stay the same on both GitLab and Github.

useReplacementIssuesForCreationFails If this is set to true (default) then the migration process will automatically create so called "replacement-issues" for every issue where the migration fails. This replacement issue will be exactly the same, but the original description will be lost. In the future, the description of the replacement issue will also contain a link to the original issue on GitLab. This way, users who still have access to the GitLab repository can still view its content. However, this is still an open task. (TODO)

It would of course be better to find the cause for migration fails, so that no replacement issues would be needed. Finding the cause together with a retry-mechanism would be optimal, and will maybe come in the future - currently the replacement-issue-mechanism helps to keep things in order.

useIssuesForAllMergeRequests

If this is set to true (default is false) then all merge requests will be migrated as GitHub issues (rather than pull requests). This can be used to sidestep the problem where pull requests are rejected by GitHub if the feature branch no longer exists or has been merged.

filterByLabel

Filters all merge requests and issues by these labels. The applicable values can be found in the GitLab API documentation for issues and merge requests respectively. Default is `null` which returns all issues/merge requests.

skipMergeRequestStates

Merge requests in GitLab with any of the states listed in this array will not be transferred to GitHub (e.g. set to `['merged', 'closed']` to avoid creating issues for closed MRs whose branches have been deleted).

skipMatchingComments

This is an array (empty per default) that may contain string values. Any note/comment in any issue, that contains one or more of those string values, will be skipped (meaning not migrated). Note that this is case insensitive, therefore the string value `foo` would also lead to skipping notes containing a (sub)string `FOO`.

Suggested values:

- `time spent`, since those kind of terms can be used in GitLab to track time, they are rather meaningless in Github though
- action entries, such as `changed the description`, `added 1 commit`, `mentioned in merge request`, etc as they are interpreted as comments

mergeRequests

Object consisting of `logfile` and `log`. If `log` is set to true, then the merge requests are logged in the specified file and not migrated. Conversely, if `log` is set to false, then the merge requests are migrated to GitHub and not logged. If the source or target branches linked to the merge request have been deleted, the merge request cannot be migrated to a pull request; instead, an issue with a custom “gitlab merge request” tag is created with the full comment history of the merge request.

usermap

Maps gitlab user names to github users. This is used to properly set assignees in issues and PRs and to translate mentions in issues.

projectmap

This is useful when migrating multiple projects if they are renamed at destination. Provide a map from gitlab names to github names so that any cross-project references (e.g. issues) are not lost.

Import limit

Because Github has a limit of 5000 Api requests per hour one has to be careful not to go over this limit. I transferred one of my project with it ~ 300 issues with ~ 200 notes. This totals to some 500 objects excluding commits which are imported through githubs importer. I never got under 3800 remaining requests (while testing it two times in one hour).

So the rule of thumb should be that one can import a repo with ~ 2500 issues without a problem.

Bugs

Issue migration fail

See section 'useReplacementIssuesForCreationFails' above for more infos! One reason seems to be some error with `Octokit` (error message snippet: <https://pastebin.com/3VNUNYLh>)

Milestone, MR and issue references

This is WIP

the milestone refs and issue refs do not seem to be rewritten properly at the moment. specifically, milestones show up like %4 in comments and issue refs like #42 do not remap to the #42 from gitlab under the new issue number in github. @ references are remapped properly (yay). If this is a deal breaker, a large amount of the code to do this has been written it just appears to no longer work in current form :(

Feature suggestions / ideas

Throttling mechanism

A throttling mechanism could maybe help to avoid api rate limit errors. In some scenarios the ability to migrate is probably more important than the total duration of the migration process. Some users may even be willing to accept a very long duration (> 1 day if necessary?), if they can get the migration done at all, in return.

Make requests run in parallel

Some requests could be run in parallel, to shorten the total duration. Currently all GitLab- and Github-API-Requests are run sequentially.