We mirror several open/private source git repos so that we have them locally (speedy access).
Pushing those to a local GitLab also means people can easily fork them and carry on.
We may also want to push them to gitlab.com. For example, so not to give access to the private server. Alternatively, we want to have our repository synced, avoiding manually updating.

The type of mirroring we are going to perform will depend on a question:
Do we have permission as "*Maintainer*" or "*Owner*" on source repository?

NO → **PULL mirror** (using a script along with a cronjob) – from destination repository
YES → **PUSH mirror** (using gui or api) – from source repository

I am going to use a folder/group named "mirror" and, whenever possible, use a specific user just for mirroring push/pull. Therefore, the mirroring logs will only be associated to this user, providing better monitoring insights later on.

If we do not have access to a private GitLab repository then we will never be able to mirror it, no matter the type (pull/push).

Extras:

- There is also a docker container which provides a pull mirror using either cron or webhook technique. It runs a tiny Webserver to integrate with Gitlabs webhooks and sends emails on error or just for reporting.
  From here on, you will have your local repository updated (via volume-path) or you can apply a push mirror technique to the destination repository (via git command towards remote repository).
  You can find the docker container here:
  https://hub.docker.com/r/ochorocho/gitlab-mirror-pull

PULL MIRRORS

On the GitLab server I have a local posix *usrmirror* user who also owns a group called *mirror* in GitLab (this user is cannot be called “mirror” as the user and group would conflict in GitLab).
In *usrmirror*’s home directory there’s a ~/git/mirror directory which stores all the repos that I want to mirror.
The *usrmirror* user also has a cronjob that runs every few minutes to pull down any updates and push them to the appropriate project in the GitLab mirror group.

So for example, to mirror self host GitLab software, I first create a new project in the GitLab mirror group called ***gitlab*** (this would be accessed at something like https://gitlab/mirror/***gitlab***.git).

Then as the usrmirror user on GitLab I run a mirror clone:

```
[usrmirror@gitlab ~]$ cd ~/git/mirror
[usrmirror@gitlab mirror]$ git clone --mirror https://gitlab.com/gitlab-org/gitlab.git
```

Then the script takes care of future updates and pushes them directly to GitLab via localhost:

```
#!/bin/bash

# Setup proxy for any https remotes
export http_proxy=http://proxy:3128
export https_proxy=http://proxy:3128

cd ~/git/mirror

for x in $(ls -d *.git) ; do
    pushd ${x}
    git remote prune origin
    git remote update -p
    git push --mirror git@localhost:mirror/${x}"
    popd
done

echo $(date) > /tmp/git_mirror_update.timestamp
```

That’s managed by a simple cronjob that the *usrmirror* user has on the GitLab server:

```
[usrmirror@gitlab mirror]$ crontab -l 0 */3 * * * /usr/local/bin/git_mirror_update.sh
```

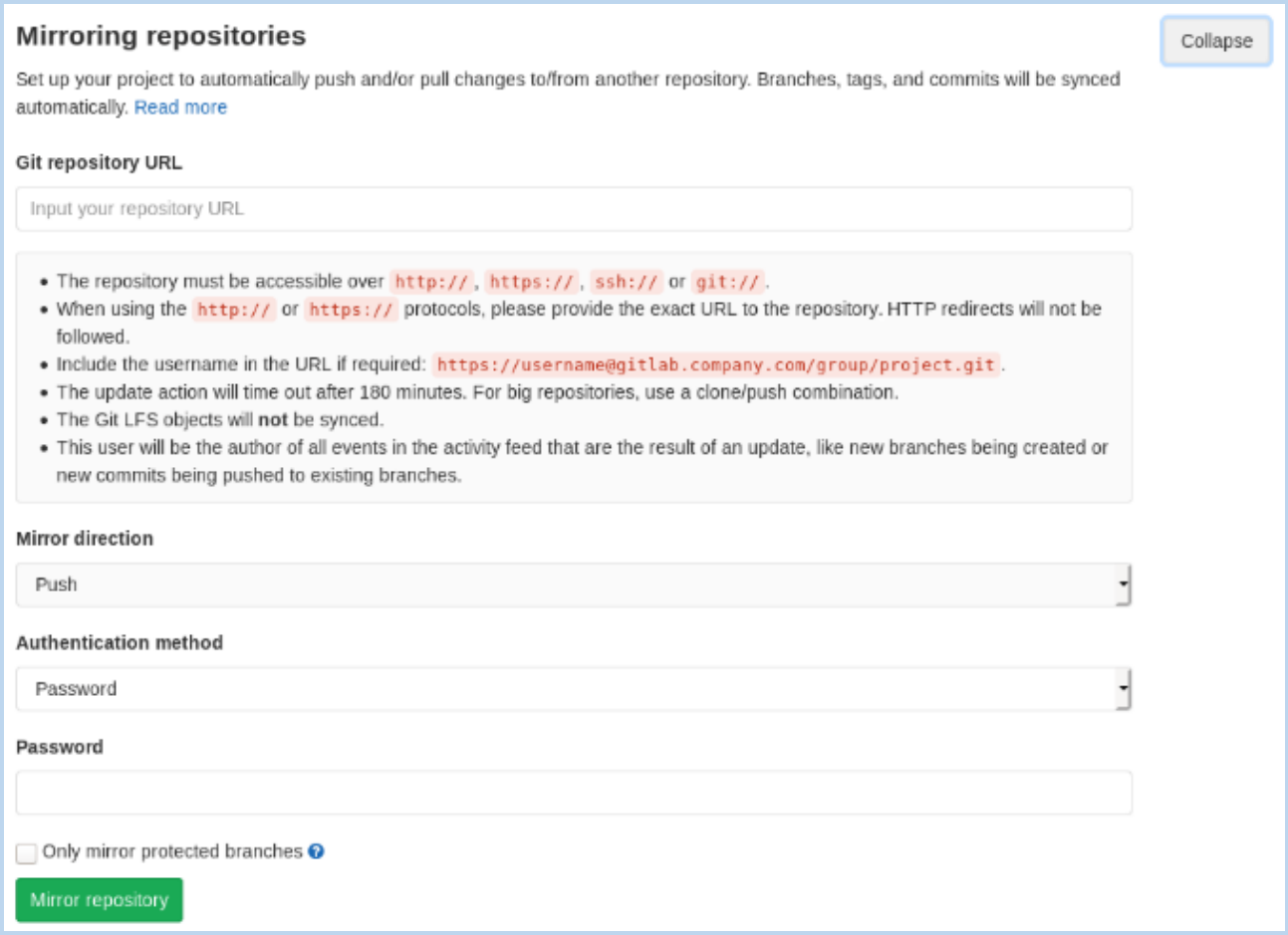
You can see cron jobs logs by running:

```
tail -f /var/log/syslog | grep CRON
```

PUSH MIRRORS

copied from GitLab.com on 03/Sept/2020

»» Push mirrors via GUI ««



When push mirroring is enabled, only push commits directly to the mirrored repository to prevent the mirror diverging. All changes will end up in the mirrored repository whenever:

- Commits are pushed to GitLab.
- A forced update is initiated.

Changes pushed to files in the repository are automatically pushed to the remote mirror at least:

- Within five minutes of being received.
- Within one minute if **Only mirror protected branches** is enabled.

In the case of a diverged branch, you will see an error indicated at the **Mirroring repositories** section.

Push only protected branches

CORE FREE

Version history ...

You can choose to only push your protected branches from GitLab to your remote repository.

To use this option, check the **Only mirror protected branches** box when creating a repository mirror.

Keep divergent refs

CORE FREE

Introduced in GitLab 13.0.

By default, if any ref on the remote mirror has diverged from the local repository, the *entire push* will fail, and nothing will be updated.

For example, if a repository has `master`, `develop`, and `stable` branches that have been mirrored to a remote, and then a new commit is added to `develop` on the mirror, the next push attempt will fail, leaving `master` and `stable` out-of-date despite not having diverged. No change on any branch can be mirrored until the divergence is resolved.

With the **Keep divergent refs** option enabled, the `develop` branch is skipped, allowing `master` and `stable` to be updated. The mirror status will reflect that `develop` has diverged and was skipped, and be marked as a failed update.

> **Note:** After the mirror is created, this option can currently only be modified via the API.

Setting up a push mirror to another GitLab instance with 2FA activated

1. On the destination GitLab instance, create a personal access token with `write_repository` scope.
2. On the source GitLab instance:
    1. Fill in the **Git repository URL** field using this format: `https://oauth2@<destination host>/<your_gitlab_group_or_name>/<your_gitlab_project>.git`.
    2. Fill in the **Password** field with the GitLab personal access token created on the destination GitLab instance.
    3. Click the **Mirror repository** button.

SSH authentication is mutual:

- You have to prove to the server that you're allowed to access the repository.
- The server also has to prove to *you* that it's who it claims to be.

You provide your credentials as a password or public key. The server that the other repository resides on provides its credentials as a "host key", the fingerprint of which needs to be verified manually.

If you're mirroring over SSH (that is, using an `ssh://` URL), you can authenticate using:

- Password-based authentication, just as over HTTPS.
- Public key authentication. This is often more secure than password authentication, especially when the other repository supports Deploy Keys.

To get started:

1. Navigate to your project's **Settings > Repository** and expand the **Mirroring repositories** section.
2. Enter an `ssh://` URL for mirroring.

Note: SCP-style URLs (that is, `git@example.com:group/project.git`) are not supported at this time.

Entering the URL adds two buttons to the page:

- **Detect host keys**.
- **Input host keys manually**.

If you click the:

- **Detect host keys** button, GitLab will fetch the host keys from the server and display the fingerprints.
- **Input host keys manually** button, a field is displayed where you can paste in host keys.

SSH public key authentication

To use SSH public key authentication, you'll also need to choose that option from the **Authentication method** dropdown. When the mirror is created, GitLab generates a 4096-bit RSA key that can be copied by clicking the **Copy SSH public key** button.

You then need to add the public SSH key to the other repository's configuration:

- If the other repository is hosted on GitLab, you should add the public SSH key as a Deploy Key.
- If the other repository is hosted elsewhere, you may need to add the key to your user's `authorized_keys` file. Paste the entire public SSH key into the file on its own line and save it.

If you need to change the key at any time, you can remove and re-add the mirror to generate a new key. You'll have to update the other repository with the new key to keep the mirror running.

> **Note:** The generated keys are stored in the GitLab database, not in the filesystem. Therefore, SSH public key authentication for mirrors cannot be used in a pre-receive hook.

»» Push mirrors via API ««

Project remote mirrors API

Push mirrors defined on a project's repository settings are called "remote mirrors", and the state of these mirrors can be queried and modified via the remote mirror API outlined below.

List a project's remote mirrors

Introduced in GitLab 12.9.

Returns an Array of remote mirrors and their statuses:

```
GET /projects/:id/remote_mirrors
```

Example request:

```
curl --header "PRIVATE-TOKEN: <your_access_token>" "https://gitlab.example.com/a
```

Example response:

```
[
  {
    "enabled": true,
    "id": 101486,
    "last_error": null,
    "last_successful_update_at": "2020-01-06T17:32:02.823Z",
    "last_update_at": "2020-01-06T17:32:02.823Z",
    "last_update_started_at": "2020-01-06T17:31:55.864Z",
    "only_protected_branches": true,
    "keep_divergent_refs": true,
    "update_status": "finished",
    "url": "https://*****:*****@gitlab.com/gitlab-org/security/gitlab.git"
  }
]
```

> **Note:** For security reasons, the `url` attribute will always be scrubbed of username and password information.

Create a remote mirror

Introduced in GitLab 12.9.

Create a remote mirror for a project. The mirror will be disabled by default. You can enable it by including the optional parameter `enabled` when creating it:

```
POST /projects/:id/remote_mirrors
```

Attribute	Type	Required	Description
`url`	String	yes	The URL of the remote repository to be mirrored.
`enabled`	Boolean	no	Determines if the mirror is enabled.
`only_protected_branches`	Boolean	no	Determines if only protected branches are mirrored.
`keep_divergent_refs`	Boolean	no	Determines if divergent refs are skipped.

Example request:

```
curl --request POST --data "url=https://username:token@example.com/gitlab/exampl
```

Example request:

```
curl --request POST --data "url=https://username:token@example.com/gitlab/exampl
```

Example response:

```
{
    "enabled": false,
    "id": 101486,
    "last_error": null,
    "last_successful_update_at": null,
    "last_update_at": null,
    "last_update_started_at": null,
    "only_protected_branches": false,
    "keep_divergent_refs": false,
    "update_status": "none",
    "url": "https://*****:*****@example.com/gitlab/example.git"
}
```

Update a remote mirror's attributes

Introduced in GitLab 12.9.

Toggle a remote mirror on or off, or change which types of branches are mirrored:

```
PUT /projects/:id/remote_mirrors/:mirror_id
```

Attribute	Type	Required	Description
`mirror_id`	Integer	yes	The remote mirror ID.
`enabled`	Boolean	no	Determines if the mirror is enabled.
`only_protected_branches`	Boolean	no	Determines if only protected branches are mirrored.
`keep_divergent_refs`	Boolean	no	Determines if divergent refs are skipped.

Example request:

```
curl --request PUT --data "enabled=false" --header "PRIVATE-TOKEN: <your_access_
```

Example response:

```
{
    "enabled": false,
    "id": 101486,
    "last_error": null,
    "last_successful_update_at": "2020-01-06T17:32:02.823Z",
    "last_update_at": "2020-01-06T17:32:02.823Z",
    "last_update_started_at": "2020-01-06T17:31:55.864Z",
    "only_protected_branches": true,
    "keep_divergent_refs": true,
    "update_status": "finished",
    "url": "https://*****:*****@gitlab.com/gitlab-org/security/gitlab.git"
}
```