Maintenance Rake tasks **(FREE SELF)**

GitLab provides Rake tasks for general maintenance.

Gather GitLab and system information

This command gathers information about your GitLab installation and the system it runs on. These may be useful when asking for help or reporting issues. In a multi-node environment, run this command on nodes running GitLab Rails to avoid PostgreSQL socket errors.

Omnibus Installation

```
sudo gitlab-rake gitlab:env:info
```

Source Installation

```
bundle exec rake gitlab:env:info RAILS_ENV=production
```

Example output:

```
System information
System:         Ubuntu 20.04
Proxy:          no
Current User:   git
Using RVM:      no
Ruby Version:   2.6.6p146
Gem Version:    2.7.10
Bundler Version:1.17.3
Rake Version:   12.3.3
Redis Version:  5.0.9
Git Version:    2.27.0
Sidekiq Version:5.2.9
Go Version:     unknown

GitLab information
Version:        13.2.2-ee
Revision:       618883a1f9d
Directory:      /opt/gitlab/embedded/service/gitlab-rails
DB Adapter:     PostgreSQL
DB Version:     11.7
URL:            http://gitlab.example.com
HTTP Clone URL: http://gitlab.example.com/some-group/some-project.git
SSH Clone URL:  git@gitlab.example.com:some-group/some-project.git
Elasticsearch:  no
Geo:            no
Using LDAP:     no
Using Omniauth: yes
Omniauth Providers:

GitLab Shell
Version:    13.3.0
Repository storage paths:
- default:  /var/opt/gitlab/git-data/repositories
GitLab Shell path:      /opt/gitlab/embedded/service/gitlab-shell
```

Show GitLab license information **(PREMIUM SELF)**

- Introduced in GitLab 12.6.
- Moved to GitLab Premium in 13.9.

This command shows information about your GitLab license and how many seats are used. It is only available on GitLab Enterprise installations: a license cannot be installed into GitLab Community Edition.

These may be useful when raising tickets with Support, or for programmatically checking your license parameters.

Omnibus Installation

```
sudo gitlab-rake gitlab:license:info
```

Source Installation

```
bundle exec rake gitlab:license:info RAILS_ENV=production
```

Example output:

```
Today's Date: 2020-02-29
Current User Count: 30
Max Historical Count: 30
Max Users in License: 40
License valid from: 2019-11-29 to 2020-11-28
Email associated with license: user@example.com
```

Check GitLab configuration

The `gitlab:check` Rake task runs the following Rake tasks:

- `gitlab:gitlab_shell:check`
- `gitlab:gitaly:check`
- `gitlab:sidekiq:check`
- `gitlab:incoming_email:check`
- `gitlab:ldap:check`
- `gitlab:app:check`

It checks that each component was set up according to the installation guide and suggest fixes for issues found. This command must be run from your application server and doesn't work correctly on component servers like Gitaly. If you're running Geo, see also the Geo Health check Rake task.

You may also have a look at our troubleshooting guides for:

- GitLab
- Omnibus GitLab

Additionally you should also verify database values can be decrypted using the current secrets.

To run `gitlab:check`, run:

Omnibus Installation

```
sudo gitlab-rake gitlab:check
```

Source Installation

```
bundle exec rake gitlab:check RAILS_ENV=production
```

Use `SANITIZE=true` for `gitlab:check` if you want to omit project names from the output.

Example output:

```
Checking Environment ...

Git configured for git user? ... yes
Has python2? ... yes
python2 is supported version? ... yes
```

```
Checking Environment ... Finished

Checking GitLab Shell ...

GitLab Shell version? ... OK (1.2.0)
Repo base directory exists? ... yes
Repo base directory is a symlink? ... no
Repo base owned by git:git? ... yes
Repo base access is drwxrws---? ... yes
post-receive hook up-to-date? ... yes
post-receive hooks in repos are links: ... yes

Checking GitLab Shell ... Finished

Checking Sidekiq ...

Running? ... yes

Checking Sidekiq ... Finished

Checking GitLab ...

Database config exists? ... yes
Database is SQLite ... no
All migrations up? ... yes
GitLab config exists? ... yes
GitLab config outdated? ... no
Log directory writable? ... yes
Tmp directory writable? ... yes
Init script exists? ... yes
Init script up-to-date? ... yes
Redis version >= 2.0.0? ... yes

Checking GitLab ... Finished
```

Rebuild `authorized_keys` file

In some cases it is necessary to rebuild the `authorized_keys` file, for example, if after an upgrade you receive `Permission denied (publickey)` when pushing via SSH and find `404 Key Not Found` errors in the `gitlab-shell.log` file. To rebuild `authorized_keys`, run:

Omnibus Installation

```
sudo gitlab-rake gitlab:shell:setup
```

Source Installation

```
cd /home/git/gitlab
sudo -u git -H bundle exec rake gitlab:shell:setup RAILS_ENV=production
```

Example output:

```
This will rebuild an authorized_keys file.
You will lose any data stored in authorized_keys file.
Do you want to continue (yes/no)? yes
```

Clear Redis cache

If for some reason the dashboard displays the wrong information, you might want to clear Redis' cache. To do this, run:

Omnibus Installation

```
sudo gitlab-rake cache:clear
```

Source Installation

```
cd /home/git/gitlab
sudo -u git -H bundle exec rake cache:clear RAILS_ENV=production
```

Precompile the assets

Sometimes during version upgrades you might end up with some wrong CSS or missing some icons. In that case, try to precompile the assets again.

This Rake task only applies to source installations. Read more about troubleshooting this problem when running the Omnibus GitLab package. The guidance for Omnibus GitLab might be applicable for Kubernetes and Docker Omnibus deployments of GitLab, though in general, container-based installations don't have issues with missing assets.

Source Installation

```
cd /home/git/gitlab
sudo -u git -H bundle exec rake gitlab:assets:compile RAILS_ENV=production
```

For omnibus versions, the unoptimized assets (JavaScript, CSS) are frozen at the release of upstream GitLab. The omnibus version includes optimized versions of those assets. Unless you are modifying the JavaScript / CSS code on your production machine after installing the package, there should be no reason to redo `rake gitlab:assets:compile` on the production machine. If you suspect that assets have been corrupted, you should reinstall the omnibus package.

Check TCP connectivity to a remote site

Sometimes you need to know if your GitLab installation can connect to a TCP service on another machine (for example a PostgreSQL or web server) to troubleshoot proxy issues. A Rake task is included to help you with this.

Omnibus Installation

```
sudo gitlab-rake gitlab:tcp_check[example.com,80]
```

Source Installation

```
cd /home/git/gitlab
sudo -u git -H bundle exec rake gitlab:tcp_check[example.com,80] RAILS_ENV=production
```

Clear exclusive lease (DANGER)

GitLab uses a shared lock mechanism: `ExclusiveLease` to prevent simultaneous operations in a shared resource. An example is running periodic garbage collection on repositories.

In very specific situations, an operation locked by an Exclusive Lease can fail without releasing the lock. If you can't wait for it to expire, you can run this task to manually clear it.

To clear all exclusive leases:

WARNING: Don't run it while GitLab or Sidekiq is running

```
sudo gitlab-rake gitlab:exclusive_lease:clear
```

To specify a lease `type` or lease `type + id`, specify a scope:

```
# to clear all leases for repository garbage collection:
sudo gitlab-rake gitlab:exclusive_lease:clear[project_housekeeping:*]

# to clear a lease for repository garbage collection in a specific project: (id=4)
sudo gitlab-rake gitlab:exclusive_lease:clear[project_housekeeping:4]
```

Display status of database migrations

See the background migrations documentation for how to check that migrations are complete when upgrading GitLab.

To check the status of specific migrations, you can use the following Rake task:

```
sudo gitlab-rake db:migrate:status
```

To check the tracking database on a Geo secondary site, you can use the following Rake task:

```
sudo gitlab-rake db:migrate:status:geo
```

This outputs a table with a `Status` of `up` or `down` for each Migration ID.

```
database: gitlabhq_production

 Status   Migration ID    Migration Name
--------------------------------------------------
   up     migration_id    migration_name
```

Run incomplete database migrations

Database migrations can be stuck in an incomplete state, with a `down` status in the output of the `sudo gitlab-rake db:migrate:status` command.

1. To complete these migrations, use the following Rake task:

   ```
   sudo gitlab-rake db:migrate
   ```

2. After the command completes, run `sudo gitlab-rake db:migrate:status` to check if all migrations are completed (have an `up` status).

3. Hot reload `puma` and `sidekiq` services:

   ```
   sudo gitlab-ctl hup puma
   sudo gitlab-ctl restart sidekiq
   ```

Rebuild database indexes

WARNING: This is an experimental feature that isn't enabled by default. It requires PostgreSQL 12 or later.

Database indexes can be rebuilt regularly to reclaim space and maintain healthy levels of index bloat over time.

To rebuild the two indexes with the highest estimated bloat, use the following Rake task:

```
sudo gitlab-rake gitlab:db:reindex
```

To target a specific index, use the following Rake task:

```
sudo gitlab-rake gitlab:db:reindex['public.a_specific_index']
```

The following index types are not supported:

1. Indexes used for constraint exclusion
2. Partitioned indexes
3. Expression indexes

Optionally, this Rake task sends annotations to a Grafana (4.6 or later) endpoint. Use the following custom environment variables to enable annotations:

1. `GRAFANA_API_URL` - The base URL for Grafana, for example `http://some-host:3000`.
2. `GRAFANA_API_KEY` - Grafana API key with at least `Editor role`.

You can also enable reindexing as a regular cron job.

Import common metrics

Sometimes you may need to re-import the common metrics that power the Metrics dashboards.

This could be as a result of updating existing metrics, or as a troubleshooting measure.

To re-import the metrics you can run:

```
sudo gitlab-rake metrics:setup_common_metrics
```