Advanced configuration

- Adopt existing git directories
- Interface customization
- Moderation tools
- Soft-Quota
- Incoming Email
- Search Engines Indexation
- OAuth2 provider
- Instance Commit Signing

Adopt existing git directories

If directories containing bare git repositories exist in the `[repository].ROOT` hierarchy, they can be imported using the admin panel.

- Make sure the directory names are lowercase.
- `[repository].ROOT/{user,org}/{repo}.git` will become a project owned by the user or the org.
- Go to `/admin/repos/unadopted`
- In trusted environments, users can also be given similar permissions via the `ALLOW_ADOPTION_OF_UNADOPTED_REPOSITORIES` setting.

Interface customization

Forgejo currently has limited capabilities for customizing the user interface.

Configurations and UI settings

Some settings are provided to customize the look and feel of the Forgejo user interface.

This includes the default theme, the name of your Forgejo instance, as well as the description that appears in Forgejo's homepage. A complete list of those settings can be found in the config cheat sheet:

- In the UI section and below.
- The `APP_NAME` setting in the Overall section.

They are documented and supported to be backward compatible between versions.

Serving custom resources, logos and pages

A word of warning (Here be dragons!)

Setting a custom logo for your instance, serving custom public files or modifying pages shown by Forgejo (such as the homepage) is possible. However, they impose an additional maintenance burden on administrators and, most importantly, **are unsupported**.

Unsupported means that future updates **are likely to break** your changes **without any warning**.

Templates

The most dangerous types of modifications are the ones concerning **template files** (`.tmpl`) served by Forgejo, as Forgejo issues backward incompatible updates to its templates very regularly.

Before deploying your changes to production or upgrading a modified Forgejo instance, we urge that you test your custom modifications in a testing environment first.

Instructions

For the reasons mentioned above, the instructions on performing such tasks can be found in the developer section on interface customization.

At this stage, it is also worth mentioning that it is possible to customize Forgejo by modifying its source code and compiling the changes, or by extracting and modifying the files of your choice using the command `forgejo embedded extract`. However, these methods are, for the same reasons as mentioned above, also **unsupported**.

Moderation tools

Moderation tools are meant to help the Forgejo users and admins handle spam bots and undesirable interactions.

Email notifications to instance admins on new user sign-up

`[admin].SEND_NOTIFICATION_EMAIL_ON_NEW_USER` can be set to `true` on small Forgejo instances with an open registration. Such instances are subject to occasional spam bots registrations and saves the admin the trouble to check on a regular basis. Read more in the config cheat sheet.

Self moderation tools

On large instances the moderation team may be too busy with fighting spam to handle problematic relationships between users. Self moderation tools may help in this case and allow users to block any unwanted interaction from another user. Read more in the user blocking guide.

Deleting users together with their content

To delete a user, including all the repositories or issues they created the Forgejo admin can either:

- Use the administration panel of the web interface
- Use the `forgejo admin user delete --purge` command line. Read more in the command line documentation.

Abusive content reporting

By default this functionality is disabled and can be enabled by adding the following lines within `app.ini` config file:

```
[moderation]
ENABLED = true
```

Optionally, you can also define the amount of time after which the resolved reports will be automatically deleted if the corresponding cron is enabled:

```
KEEP_RESOLVED_REPORTS_FOR = 168h
```

When enabled, within the '⋯' context menu from user profile pages and organization pages a new 'Report abuse' action will be available while for repositories, issues, pull requests and comments the '⋯' menu will get a 'Report content' action.

Note that the new actions will not be available for your profile, for issues, pull requests and comments posted by you, or for organizations and repositories owned by you.

Clicking on 'Report abuse' or 'Report content' will open a new page where the user is able to select from a predefined list of categories and add some remarks before submitting the report to administrators.

At the same time, a new 'Moderation reports' section will be available within the 'Site administration' page, where administrators can see an overview with the submitted abuse reports that are still open (not yet handled in any way). When multiple reports (submitted by distinct users) exist for the same content only the first one will be shown in the list and a counter can be seen on the right side (indicating the number of open reports for the same content type and ID). Clicking on the counter or the adjacent icon will open the details page where a list with all the reports (one or multiple) linked to the reported content is shown, as well as any shadow copy saved for the current report(s).

Below the right side counters from the overview page there are also some buttons that can be used to:

- mark a report as Handled or as Ignored (without performing any action on the reported content);
- mark a user account as suspended (prohibited from signing in and existing sessions won't work any more, but please note that SSH access could still be allowed);
- delete (and purge) a user / organization and mark the linked reports as Handled;
- delete a repository and mark the linked reports as Handled;
- delete an issue / pull request and mark the linked reports as Handled;
- delete a comment and mark the linked reports as Handled;

Only the buttons for updating the status of the report are directly visible - as '✓' and '✗' icons with some tooltips - while the content actions are hidden under a '⋯' dropdown.

Soft-Quota

Forgejo has early support for a soft-quota that can protect your server from high disk usage due to abuse or errors. The feature is still in development. If you make use of it, consider sending us feedback via a discussion, or the Matrix channels.

Understanding soft-quota and its limitations

Forgejo has chosen to use a "soft" quota implementation. It means that Forgejo checks the quota usage only before an action is executed, but it will allow a started action to complete.

In some cases (like pushing to Git repositories), it is hard to estimate the exact new size, because it depends on how much data is available and how much we can benefit from compression. As a result, it is possible to exceed the quota if the operation was started before the quota was used up, but after the quota is exceeded, new operations that would increase the quota won't be possible.

Further, there is currently little support for early prevention of operations in the UI: The handling for e.g. web operations that are denied later is not yet optimal.

You can read more about the technical background in the initial PR setting the foundations.

Getting started

The quota feature is currently not yet enabled by default. Set this in your app.ini:

```
[quota]
ENABLED = true
```

You can now choose between managing quota via the config file (simple option, but limited functionality) or via the API (more advanced).

Simple case

To make use of a simple quota for your instance, you can set a global quota for all data:

```
[quota.default]
TOTAL = 2G
```

`quota.default.TOTAL` is `-1` (unlimited) by default and can take sizes suffixed with units such as 500M or 10G.

Advanced usage: via API

If you have more complex needs, you can use the API to configure quota rules.

1. With an admin account, create a new application token.
2. Make yourself familiar with the API endpoints by visiting the swagger documentation, e.g. by visiting `/api/swagger` (online example).
3. Make yourself familiar with the available quota subjects from the respective section in the config cheat sheet.
4. Optionally, set up a local helper for interacting with the API:

```
  export FORGEJO_API_TOKEN="your-admin-api-token" #  delete after setting it up or ensure it
is not preserved in the shell history
api_helper() {
  endpoint="$1"; shift
  curl -s -H "content-type: application/json" \
          -H "accept: application/json" \
          -H "authorization: token ${FORGEJO_API_TOKEN}" \
      "https://url-to-your-forgejo/api/v1${endpoint}" "$@" | jq .
}
```

Example: Set up a quota group

This example sets a quota group that is restricted tightly on Git repo space, has more generous limits for LFS files and can store unlimited packages and attachments, but Actions artifacts are restricted:

```
{
  "name": "newgroup",
  "rules": [
    {
```

```
      "name": "git",
      "limit": 200000000,
      "subjects": ["size:repos:all"]
    },
    {
      "name": "git-lfs",
      "limit": 500000000,
      "subjects": ["size:git:lfs"]
    },
    {
      "name": "all-assets",
      "limit": -1,
      "subjects": ["size:assets:all"]
    },
    {
      "name": "size:assets:artifacts",
      "limit": 350000000,
      "subjects": ["size:assets:all"]
    }
  ]
}
```

After tuning the above JSON to your needs, you can create a new quota group by calling the API:

```
api_helper /admin/quota/groups -XPOST -d '{"your": "json"}'
```

Check the API documentation for more details on how you can add, modify and delete quota groups and their rules.

Example: Managing users

If you do not modify the `default` quota group, or have your new groups listed in the `[quota].DEFAULT_GROUPS` list, they won't apply to users.

However, you can create quota groups and assign users to them:

```
api_helper "/admin/quota/groups/<GROUPNAME>/users/<USERNAME>" -XPUT
```

Or remove them:

```
api_helper "/admin/quota/groups/<GROUPNAME>/users/<USERNAME>" -XDELETE
```

Sync Quota Groups with OAuth2 Providers

If you use OAuth2 providers, you can sync the quota groups with a special claim returned by the provider.

This claim should be a comma-separated list of group names, for example:

```
{
  "quota_groups": "dev,student"
}
```

You can configure the mapping of these groups to quota groups in Forgejo, allowing you to manage quotas based on the user's OAuth2 group membership. For example:

```
{
  "dev": ["premium", "git-max-50g"],
  "student": ["git-max-5g"]
}
```

You will need to configure the auth source at `https://<your-instance>/admin/auths` by filling these fields:

- **Claim name providing group names for this source to be used for quota management**: For the example above, this would be `quota_groups`.
- **Map claimed groups to quota groups**: Here you can map the groups returned by the OAuth2 provider to the quota groups you have created. Fill in the JSON using the format shown above.
- **Remove users from synchronized quota groups if user does not belong to corresponding group**: If enabled, users will be removed from quota groups if they no longer belong to the corresponding OAuth2 group. Notice: only the groups specified in the mapping will be considered for removal. If you simply remove a group from the mapping, users will not be removed from the quota group.

Incoming Email

Forgejo supports the execution of several actions through incoming mails. This page describes how to set this up.

Requirements

Handling incoming email messages requires an IMAP-enabled email account. The recommended strategy is to use email sub-addressing but a catch-all mailbox does work too. The receiving email address contains a user/action specific token which tells Forgejo which action should be performed. This token is expected in the `To` and `Delivered-To` header fields.

Forgejo tries to detect automatic responses to skip and the email server should be configured to reduce the incoming noise too (spam, newsletter).

Configuration

To activate the handling of incoming email messages you have to configure the `email.incoming` section in the configuration file.

The `REPLY_TO_ADDRESS` contains the address an email client will respond to. This address needs to contain the `%{token}` placeholder which will be replaced with a token describing the user/action. This placeholder must only appear once in the address and must be in the user part of the address (before the `@`).

An example using email sub-addressing may look like this: `incoming+%{token}@example.com`

If a catch-all mailbox is used, the placeholder may be used anywhere in the user part of the address, for example:

- `incoming+%{token}@example.com`
- `incoming_%{token}@example.com`
- `%{token}@example.com`

Search Engines Indexation

Search engines indexation of your Forgejo installation

By default, your Forgejo installation will be indexed by search engines. If you don't want your repository to be visible for search engines read further.

Block search engines indexation using robots.txt

To make Forgejo serve a custom `robots.txt` (default: empty 404) for top level installations, create a file with path `public/robots.txt` at the root of the `Custom File Root Path` as displayed in the `/admin/config` page.

Examples on how to configure the `robots.txt` can be found at https://moz.com/learn/seo/robotstxt.

```
User-agent: *
Disallow: /
```

If you installed Forgejo in a subdirectory, you will need to create or edit the `robots.txt` in the top level directory.

```
User-agent: *
Disallow: /forgejo/
```

Disallow crawling archives to save disk space

If the archive files are crawled, they will be generated dynamically and kept around which can amount to a lot of disk. To prevent that from happening, add the following to the `robots.txt` file:

```
User-agent: *
Disallow: /*/*/archive/
```

See also a more complete example at Codeberg.

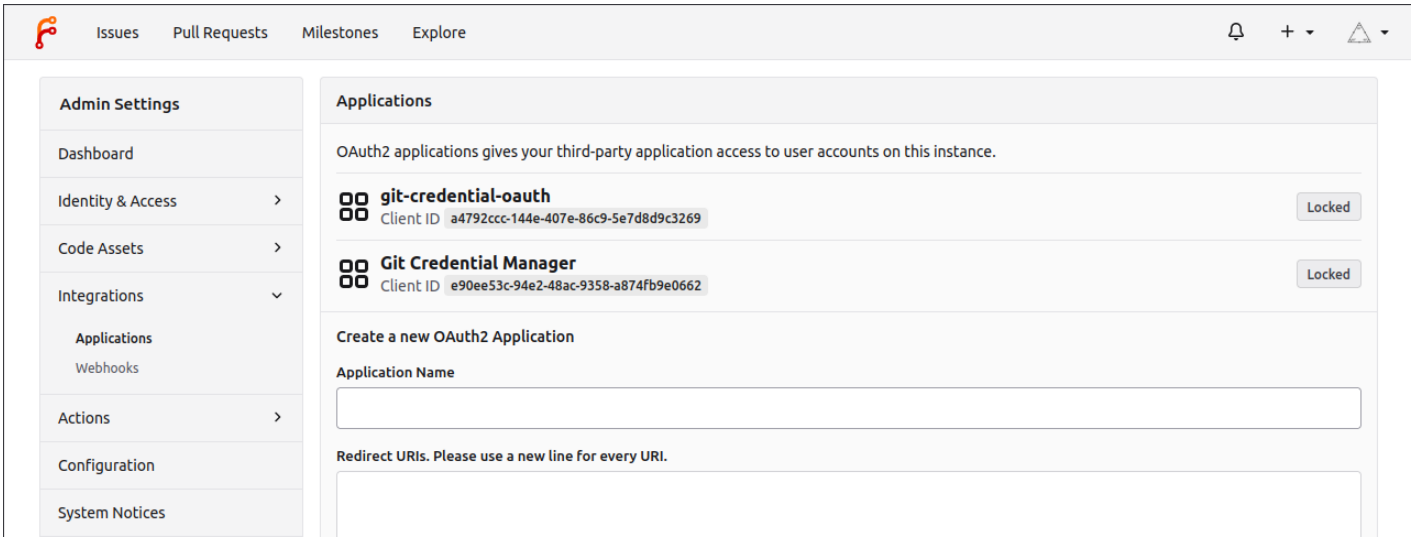
OAuth2 provider

Pre-registered applications

The following OAuth2 applications are pre-registered because it is generally useful for Forgejo to be an OAuth2 provider for the corresponding third party software. Their usage is explained in the Forgejo user guide.

- **git-credential-manager** is the name of the OAuth2 application for the Git Credential Manager (a Git credential helper)
- **git-credential-oauth** is the name of the OAuth2 application for the git-credential-oauth (a Git credential helper)

All pre-registered applications are activated by default in the `[oauth2].DEFAULT_APPLICATIONS` setting as displayed in the `/admin/applications` page.



Instance Commit Signing

Forgejo has the ability to sign commits when Forgejo themselves generates those commits, such as:

- Repository initialisation
- Wiki changes
- CRUD actions using the web editor or the API
- Merges from pull requests

Configuration

In order for Forgejo to sign commits, it has to know how it should be signing commits and when to sign commits. Unless otherwise indicated, all configuration settings discussed on this page are for the `[repository.signing]` section.

Signing key

Forgejo offers two formats to sign commits with: GPG and SSH. If you meet the requirements of SSH, then it is strongly preferred to use that instead of GPG.

SSH

For Forgejo to do SSH commit signing, it needs a Git version equal to or newer than 2.34.0 and `ssh-keygen` binary equal to or newer than version 8.2p1.[^1]

[^1]: The git version check is already done by Forgejo, but for `ssh-keygen` only the presence of the binary is checked.

You need a dedicated OpenSSH key pair for instance signing. If you don't have such key pair yet you can generate one via `ssh-keygen`[^2] or you also could store the SSH key in TPM, there is a dedicated section with instructions on how to do that.

[^2]: https://docs.codeberg.org/security/ssh-key/ contains instructions for generating an SSH key pair, you should not generate a FIDO2 (`-sk` type) key pair as that will not work with Forgejo.

Forgejo needs to be told that it should use SSH signing and which SSH key to use, this should be configured as followed:

```
FORMAT = ssh
SIGNING_KEY = /absolute/path/to/public/ssh/key.pub
```

The value for the `SIGNING_KEY` setting needs to be an absolute path to the public key, where the private key needs to be available in the path without the `.pub` suffix.

Forgejo also needs to be told who the committer of the commit is, which requires a name and email and should be configured as followed:

```
SIGNING_NAME = "forgejo.org Instance"
SIGNING_EMAIL = "noreply@forgejo.org"
```

GPG

There are two ways to tell Forgejo which GPG key should be used for commit signing.

```
SIGNING_KEY = default
```

Will use the git config to determine the signing key: if the value of `commit.gpgsign` is set to true, then it will use the values of `user.signingkey`, `user.name` and `user.email` for the signing key, committer name and committer email respectively.

```
SIGNING_KEY = GPG-KEY-ID
SIGNING_NAME = "forgejo.org Instance"
SIGNING_EMAIL = "noreply@forgejo.org"
```

Will use the GPG keyid to search for the key in the GPG keyring. Forgejo searches for this key in a directory, which can be computed as follows: If a `GNUPGHOME` environment variable is set, this is used. Otherwise the `.gnupg` directory in the directory corresponding to the value of the `HOME_PATH` setting in the `[git]` section is used (`[git].HOME_PATH/.gnupg` so to say). It should be noted that by default, GPG does not use that keyring and you should take extra care when importing or generating the key, for example by setting the value of the `GNUPGHOME` environment to the directory Forgejo uses.

Signing operations

There are several operations for which Forgejo will generate a commit and thus be able to sign the commit. For each operation you can specify under which conditions Forgejo should sign the commit.

For each operation, you can combine the values as a comma-separated list. There are two special values that are valid values for each operation and cannot be combined with any other value for

that operation: `always` and `never`. The first value, if set, will always sign the commit and the second value, if set, will never sign the commit.

Initial commit

When should Forgejo sign the initial commit when creating a repository. The possible values for the `INITIAL_COMMIT` setting are:

- `pubkey`: Only if the user has added a GPG or SSH key to its account.
- `twofa`: Only if the user is enrolled into two-factor authentication.

Wiki

When should Forgejo sign commits to the wiki. The possible values for the `WIKI` setting are:

- `pubkey`: Only if the user has added a GPG or SSH key to its account.
- `twofa`: Only if the user is enrolled into two-factor authentication.
- `parentsigned`: Only if the parent commit is signed.

CRUD actions

When should Forgejo sign commits that are created for file changes via the web editor or API. The possible values for the `CRUD_ACTIONS` setting are:

- `pubkey`: Only if the user has added a GPG or SSH key to its account.
- `twofa`: Only if the user is enrolled into two-factor authentication.
- `parentsigned`: Only if the parent commit is signed.

Pull request merges

When should Forgejo sign merge commits from pull requests. The possible values for the `MERGES` setting are:

- `pubkey`: Only if the user has added a GPG or SSH key to its account.
- `twofa`: Only if the user is enrolled into two-factor authentication.
- `basesigned`: Only if the parent commit in the base repository is signed.
- `headsigned`: Only if the head commit in the head branch is signed.
- `commitssigned`: Only if all the commits in the head branch to the merge point are signed.
- `approved`: Only if the pull request targets a protected branch and has at least one approval.

Obtaining the instance signing key

If a GPG instance signing key is set, the GPG public key can be obtained at the API route, `/api/v1/signing-key.gpg`. If a repository specific GPG key is set, it can be obtained at the API route, `/api/v1/repos/{username}/{reponame}/signing-key.gpg`

If a SSH instance signing key is set, the SSH public key can be obtained at the API route, `/api/v1/signing-key.ssh`.

Using ssh-tpm-agent

It is possible to use ssh-tpm-agent so that the SSH private key can only be used in combination with a Trusted Platform Module (TPM). To use this, the server that Forgejo runs on must have access to TPM 2.0. This section only explains how to make the SSH private key available to Forgejo, not how to configure Forgejo to use it.

Follow the instruction from ssh-tpm-agent to create a key or import an existing key. An instance key is expected to be a long-lived key[^3] and therefore it is advisable to follow the 'Import existing key' guide as it allows you to backup the private key in a safe place and in case of a recovery, restore the instance SSH key.

[^3]: Rotating instance keys is currently not possible.

ssh-tpm-agent acts as an `ssh-agent(1)` and in order for Forgejo to use ssh-tpm-agent to sign commits with, it needs to have a `SSH_AUTH_SOCK` environment set when launching the Forgejo binary. How to pass this to Forgejo depends on how you run Forgejo, we consider two situation: a Systemd service on bare-metal or containerized (for example, via Docker).

In either case, the host will need to install the systemd unit service by running `ssh-tpm-agent --install-user-units`.

Systemd service

In the `[Service]` section, add the following (it is fine to have multiple `Environment` keys):

```
Environment=SSH_AUTH_SOCK="/socket/path"
```

Where `/socket/path` is replaced with the value of `ssh-tpm-agent --print-socket`.

Containerized

We take the default docker-compose file as an example. We add an environment variable and a volume mount to the compose file:

```
networks:
  forgejo:
    external: false

services:
  server:
    image: codeberg.org/forgejo/forgejo:10
    container_name: forgejo
    environment:
      - USER_UID=1000
      - USER_GID=1000
+      - SSH_AUTH_SOCK=$SOCKET_PATH
    restart: always
    networks:
      - forgejo
    volumes:
      - ./forgejo:/data
      - /etc/localtime:/etc/localtime:ro
+      - $SOCKET_PATH:$SOCKET_PATH
    ports:
      - "3000:3000"
      - "222:22"
```

Where `$SOCKET_PATH` is to be replaced with the value of `ssh-tpm-agent --print-socket`. Another volume would need to be added that exposes the public OpenSSH key, the container path should match with the path that is specified for the `SIGNING_KEY` setting.