Your First Repository

- Creating a Repository
- Connect a local repository to Forgejo
  - Note on the different types of HTTP authentication
  - Option A: Clone the newly created, empty repository
    - 1. Navigate to your local workspace (optional)
    - 2. Clone the repository
    - 3. Navigate to the cloned repository
  - Option B: Connect an existing local source tree
    - 1. Initialize a Git Repository
    - 2. Add Forgejo as the repository's origin
  - Option C: Migrate an existing git repository to Forgejo
- Making your first commit
  - 1. Create or edit a file
  - 2. Add changes to your commit
  - 3. Commit your changes
  - 4. Push your changes to Forgejo

Almost everything on Forgejo happens in a repository. Think of a repository as a home for your project, where all of its source code can be organized using Git, as well as where you can track issues and read and write wikis.

This article will guide you through creating your first repository on Forgejo, connecting your local development environment, and making your first commit.

Creating a Repository

To create a new repository, you need to be logged in to your account.

When you're logged in, you can use one of the two buttons shown in the following screenshots to create a new repository:

screenshot of the expanded menu in the upper right corner to create a new repository

This will lead you to the page below.

screenshot of new repository page

Here's an explanation of the form's fields:

- **Owner** Here you can specify whether you want this to be your own personal project or whether you want it to be part of an organization that you're a part of.
- **Repository name** A name for your repository (which will also be part of its path, in this case `https://codeberg.org/knut/foobar`).
- **Visibility** Repositories are either *public* or *private*. Public means that everyone will be able to access your repository, while your private repositories can only be accessed by you and your collaborators (see Invite Collaborators).
- **Description** A short description that appears next to your repository's name where appropriate.
- **Template** Occasionally you may want to generate your repository from an existing template. In that case, you can specify that template here. Otherwise, simply leave this field empty.
- **Issue Labels** If you want to initialize your project's issue tracker with a set of labels that you can use to categorize issues, you can choose one here. You don't have to choose this right away, though, as you can choose and modify issue labels at a later time as well.
- **.gitignore** A .gitignore file defines which files Git should not keep track of. This is useful, for example, to prevent configuration files or binaries from being tracked in version control. You can choose to add a predefined file matching the programming language you use now, or add one manually later.
- **License** Here you can choose from a list of FSF/OSI approved licenses. A `LICENSE` file will then be added to the repository. For some help on choosing the correct license, have a look at our licensing article.
- **README** is the first file one should read when accessing a repository. It's also the first file displayed when accessing a repository, a bit like the "homepage" of your repository. On Forgejo, this is interpreted as a Markdown file.
- **Initialize repository** To add the `LICENSE`, `README`, and `.gitignore` files mentioned above to your new repository, make sure you tick this box.
- **Default branch** Using this field, you can choose how to name the default branch of your Git repository. We recommend you use the predefined default.
- **Object format** is the repository's object format. We recommend SHA1 as it's the most compatible. It cannot be changed later once the repository has been created.

It's okay to only specify the owner and repository name if you want to get started quickly. After filling out the fields, click the green "Create Repository" button at the bottom of the page.

You should now see a screen similar to the one below. If you haven't chosen to generate `LICENSE`, `README`, and `.gitignore`, the screen might show instructions instead, which will vanish after your

first commit.

screenshot showing a freshly baked repository

Here's what the most important buttons do:

- **Repository Settings (1)** is where you can make adjustments to your repository settings, such as setting a project website, changing the repository description, enabling/disabling a wiki and issue tracker, or deleting the repository. You may want to give this page a visit right now to get an overview of your options.
- **The RSS, Watch, Star, and Fork buttons (2)** allow you to interact with other people's repositories. While they don't do much for your own repository, when visiting another user's repository, you can click on "Watch" to get notified about everything important happening in that repository, "Star" to show the user your appreciation (and to help other users find interesting projects more quickly), and "Fork" to create your own copy of the repository, for example, to make modifications that you want to share with the original author. You can also use the RSS button to get the RSS feed of the repository.
- **The Repository Tabs (3)** contain links to every important feature within this repository (some may not be visible by default):
  - **Code** lets you browse through all versions of this repository's code.
  - **Issues** is a very important communication tool between the author, their users, and their contributors. Think of it as part bug-tracker, part forum. For more information on this, have a look at The Basics of Issue Tracking.
  - **Pull Requests** is where other users can ask the author to "pull" in code, from a fork into the author's program.
  - **Releases** is a space where the author can upload finished versions of their program, for example binaries.
  - **Wiki** is a basic wiki feature built into Forgejo.
  - **Activity** calculates statistics about this repository.
- **Repository's Git URLs (4)** - use these to let Git know where to find this repository. Don't worry if you don't understand this now, we will look at this in the following section.

Connect a local repository to Forgejo

After creating a new repository, as laid out in the previous section, you can now move on to connect the repository with your local development copy.

In this guide, we'll focus on connecting to Forgejo via HTTP using Git on the command line, but note that there are other ways to connect to Forgejo, as laid out in more detail in these articles:

- Clone & Commit via CLI and
- Clone & Commit via Web

> Although we use HTTP in this guide, it is a good idea to set up SSH-based authentication once you feel confident to do so.

Note on the different types of HTTP authentication

When you try to access a git repository over HTTP, you have 3 approaches for authentication. This does not apply to accessing a repository over SSH.

1. Using an API token as your HTTP password
   - In Settings -> Applications, click `New access token`. Give it any name, set access to `All`, and repository to `Read and write`, and click `Generate token`. Copy the long token that appears.
   - When operating on a repo and are prompted for a password, enter the API token as your password
   - See Access Token Scope for more advanced info.
2. Using the same username and password you use on your Forgejo instance's website. This is the simplest approach but has some downsides as seen below.
3. Using OAuth. This requires a git client with OAuth support, or the installation of a companion tool for the git binary. See OAuth2 Git authentication for more info.

Comparison of options:

Method	Tools needed	Fast LFS?	Suitable for...
API token (Recommended)	Any git tool	Yes ✓	Users who want a general approach that works with any git tool including the bare git binary, and without LFS issues. Users who want to reduce the negative consequences if their git client credentials are leaked, since API tokens can be limited to repository access only.

Method	Tools needed	Fast LFS?	Suitable for...
OAuth (Recommended)	Git tool with OAuth support	Yes ✓	Users using a graphical git client with OAuth support, or who have the technical know-how to install a companion OAuth tool to the bare git binary.
Account's username and password	Any git tool	No ✗	Users who want a general approach that works with any git tool, but accept that LFS will be slow, and 2FA won't work.

Option A: Clone the newly created, empty repository

If you want to start a fresh project (so if you don't already have source code that you want to upload to Forgejo), the quickest way to get started is to clone your newly created repository like this:

1. Navigate to your local workspace (optional)

If you're just getting started, it's a good idea to keep your projects neatly sorted in a dedicated directory, like in this example:

```
knut@iceberg:~$ mkdir repositories
knut@iceberg:~$ cd repositories
knut@iceberg:~/repositories$
```

2. Clone the repository

To clone your newly created repository, execute `git clone` with the URL that is shown in your repository:

screenshot of the two options to receive a clone URL for git

```
knut@iceberg:~/repositories$ git clone https://codeberg.org/knut/foobar
Cloning into 'foobar'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 4 (delta 0), reused 0 (delta 0)
```

```
Unpacking objects: 100% (4/4), 11.94 KiB | 764.00 KiB/s, done.
```

3. Navigate to the cloned repository

After cloning, the repository should now be in a new directory with the same name as your repository. In this case, it's called `foobar`.

```
knut@iceberg:~/repositories$ cd foobar
knut@iceberg:~/repositories/foobar$ ls
LICENSE  README.md
```

Option B: Connect an existing local source tree

If you have already written source code that you now would like to upload to Forgejo, follow the steps below. An upload to Forgejo is referred to as a `push`.

1. Initialize a Git Repository

Unless you already have a Git Repository initialized in your local source tree, run `git init` in your project's directory:

```
knut@iceberg:~/my-project$ git init
Initialized empty Git repository in /home/knut/my-project/.git/
```

2. Add Forgejo as the repository's origin

Now you need to tell Git where to push your changes. You would do that by specifying Forgejo as a remote.

> In this example, we'll specify Forgejo as the `origin` remote. You can name your remote any other way, especially if you already have an `origin`, but `origin` is the recommended name for the main remote repository.

```
knut@iceberg:~/my-project$ git remote add origin https://codeberg.org/knut/foobar
```

When you clone a repository from the Internet, the URL that you got your copy of the repository from will be automatically used for the `origin` remote.

If your local copy of the repository is missing some commits that exist in the remote repository, pushing will result in an error. There are two ways to fix this:

- Run `git pull` to combine your local changes with the changes that exist in the remote repository. If this does not work, please follow the instructions in your terminal.
- If you know what you are doing, you can also overwrite the remote repository using `git -f push`. This action will **permanently** alter your remote repository and is not suitable if you are working on a project together with other people.

Option C: Migrate an existing git repository to Forgejo

You can migrate existing repos to Forgejo while preserving their entire history, tags, and more.

1. Navigate to your Foregejo instance's web interface
2. In the top right, click `+` -> `New migration`
3. Choose the provider you're importing from, and follow the instructions

Making your first commit

Now that you've connected your repository to your local development copy, it's time to make your first commit.

> If you didn't generate the `LICENSE` and `README.md` files when creating the repository, it's a good idea to add them now. Just put them in your local development copy's directory and add them to your commit, as shown below.

1. Create or edit a file

Let's modify the `README.md` file.

> If you didn't generate `README.md`, these commands will still work fine. However, instead of "Hello World!" being at the end of the file, it'll be the only line in the file.

```
knut@iceberg:~/repositories/foobar$ echo "Hello World!" >> README.md
knut@iceberg:~/repositories/foobar$ cat README.md
# foobar

Hello World!
```

Now, if you ask Git about your repository's status, you should see something similar to this:

```
knut@iceberg:~/repositories/foobar$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
```

You can create or modify more files, if you'd like.

2. Add changes to your commit

Next, you should add each modified file that you'd like to add to your commit with `git add`:

```
knut@iceberg:~/repositories/foobar$ git add README.md
```

This command should output nothing.

> If you want to add all modified files in your source tree, you can run `git add .` but **be careful** as this might add unwanted files as well, so it's a good idea to double-check by running `git status` before committing.

3. Commit your changes

By committing your changes, you create a new step in the version history of your program. They act like snapshots of your program's state at a given point in time, and you will later be able to jump back and forth between them.

> It is recommended to keep commits small and focused, so that, if necessary, they can be reverted or easily adapted into another context without too many side-effects.

To commit your changes, run `git commit` and specify a commit message; this will later help you (and others) to identify that commit when looking back at your version history.

```
knut@iceberg:~/repositories/foobar$ git commit -m "My first commit on Forgejo"
[main 1e12979] My first commit on Forgejo
 1 file changed, 1 insertion(+)
```

If you look at the version history of your program using `git log`, you may now see something similar to this:

```
knut@iceberg:~/repositories/foobar$ git log
commit 1e1297929c8c74d9a439fa71c1f0ffe1dbf3d5ad (HEAD -> main)
Author: Knut <knut@noreply.codeberg-test.org>
Date:   Sat Sep 26 14:01:00 2020 +0200

    My first commit on Forgejo

commit c75b50920e3aa7a7ab3484e898fb3ad77132722a (origin/main, origin/HEAD)
Author: Knut <knut@noreply.codeberg-test.org>
Date:   Sat Sep 26 12:29:57 2020 +0200

    Initial commit
```

4. Push your changes to Forgejo

If you're happy with the changes you made, the next step is to present them to the world by pushing them to Forgejo:

```
knut@iceberg:~/repositories/foobar$ git push -u origin main
Username for 'https://codeberg.org': knut
Password for 'https://knut@codeberg.org':
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 315 bytes | 315.00 KiB/s, done.
```

```
Total 3 (delta 0), reused 0 (delta 0)
remote: . Processing 1 references
remote: Processed 1 references in total
To https://codeberg.org/knut/foobar
   c75b509..1e12979  main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
```

The `-u` option sets the upstream remote, which we want to be Codeberg.org, as configured previously.

The `main` argument sets the name of the branch onto which shall be pushed upstream. For this example, it should be the same branch name that you specified when creating the repository.

When connecting via HTTPS, Git will ask you for your username and password, which you can enter interactively.

After refreshing the repository page, you should now see something similar to this:

screenshot showing the updated README on Codeberg

Congratulations - you've just made your first source code contribution on Forgejo!

□□□ #2
□ □□□ □□□ 2026-10-31 13:37:10 UTC
□ □□□ □□□ 2026-10-31 13:37:11 UTC