4.8 Git on the Server - GitLab

GitLab

GitWeb is pretty simplistic though. If you're looking for a modern, fully featured Git server, there are several open source solutions out there that you can install instead. As GitLab is one of the popular ones, we'll cover installing and using it as an example. This is harder than the GitWeb option and will require more maintenance, but it is a fully featured option.

Installation

GitLab is a database-backed web application, so its installation is more involved than some other Git servers. Fortunately, this process is well-documented and supported. GitLab strongly recommends installing GitLab on your server via the official Omnibus GitLab package.

The other installation options are:

- GitLab Helm chart, for use with Kubernetes.
- Dockerized GitLab packages for use with Docker.
- From the source files.
- Cloud provider such as AWS, Google Cloud Platform, Azure, OpenShift and Digital Ocean.

For more information read the GitLab Community Edition (CE) readme.

Administration

GitLab's administration interface is accessed over the web. Simply point your browser to the hostname or IP address where GitLab is installed, and log in as the admin user. The default username is `admin@local.host`, and the default password is `5iveL!fe` (which you must change right away). After you've logged in, click the "Admin area" icon in the menu at the top right.

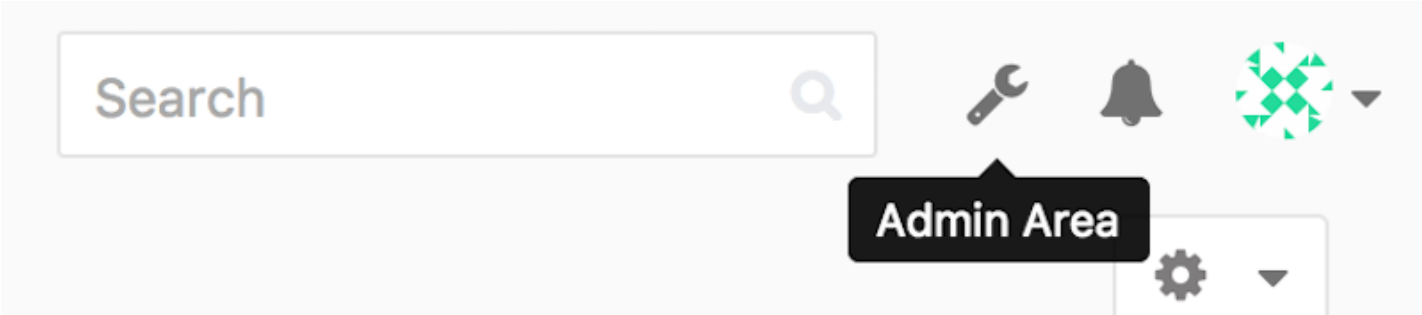

Figure 50. The "Admin area" item in the GitLab menu

Users

Everybody using your GitLab server must have an user account. User accounts are quite simple, they mainly contain personal information attached to login data. Each user account has a **namespace**, which is a logical grouping of projects that belong to that user. If the user jane had a project named project, that project's url would be `http://server/jane/project`.

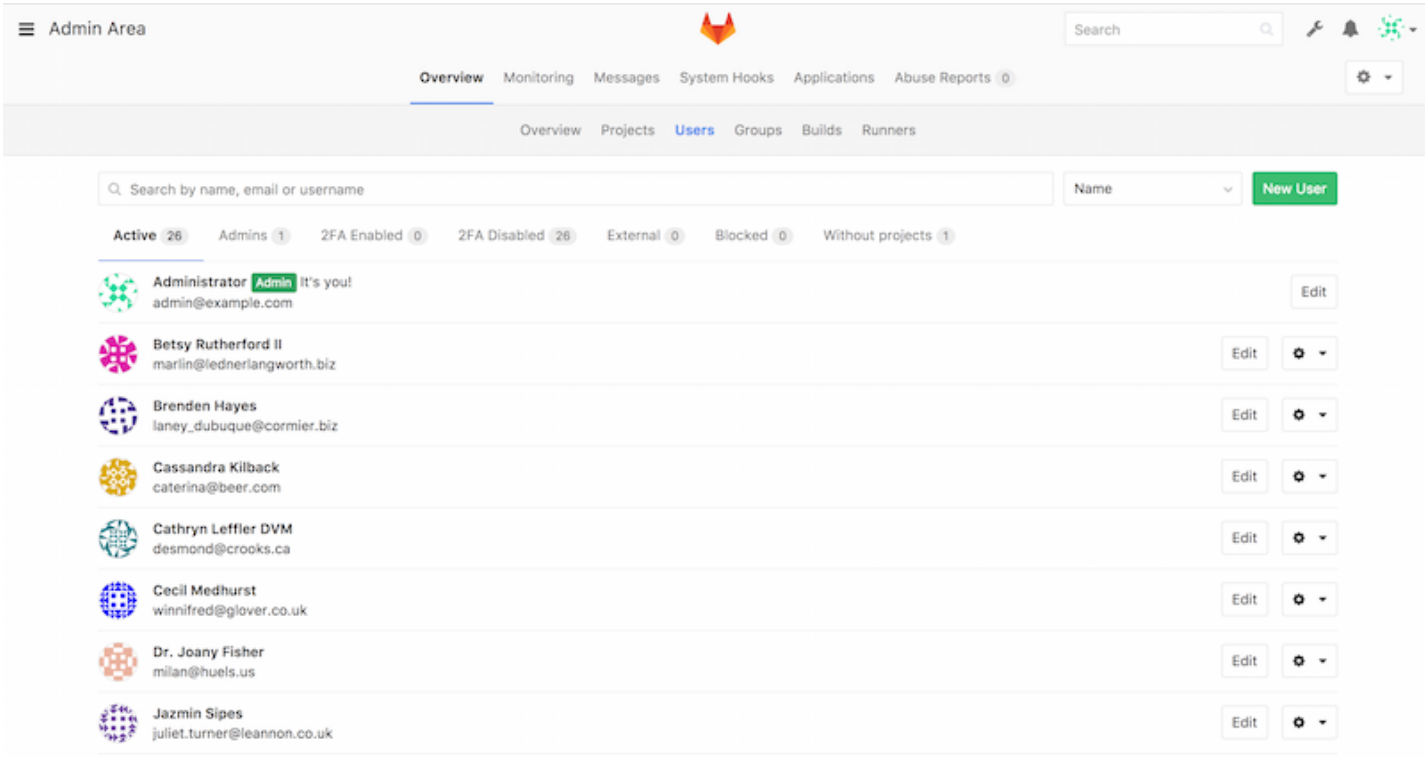

Figure 51. The GitLab user administration screen

You can remove a user account in two ways: “Blocking” a user prevents them from logging into the GitLab instance, but all of the data under that user’s namespace will be preserved, and commits signed with that user’s email address will still link back to their profile.

“Destroying” a user, on the other hand, completely removes them from the database and filesystem. All projects and data in their namespace is removed, and any groups they own will also be removed. This is obviously a much more permanent and destructive action, and you will rarely need it.

Groups

A GitLab group is a collection of projects, along with data about how users can access those projects. Each group has a project namespace (the same way that users do), so if the group training has a project materials, its url would be `http://server/training/materials`.

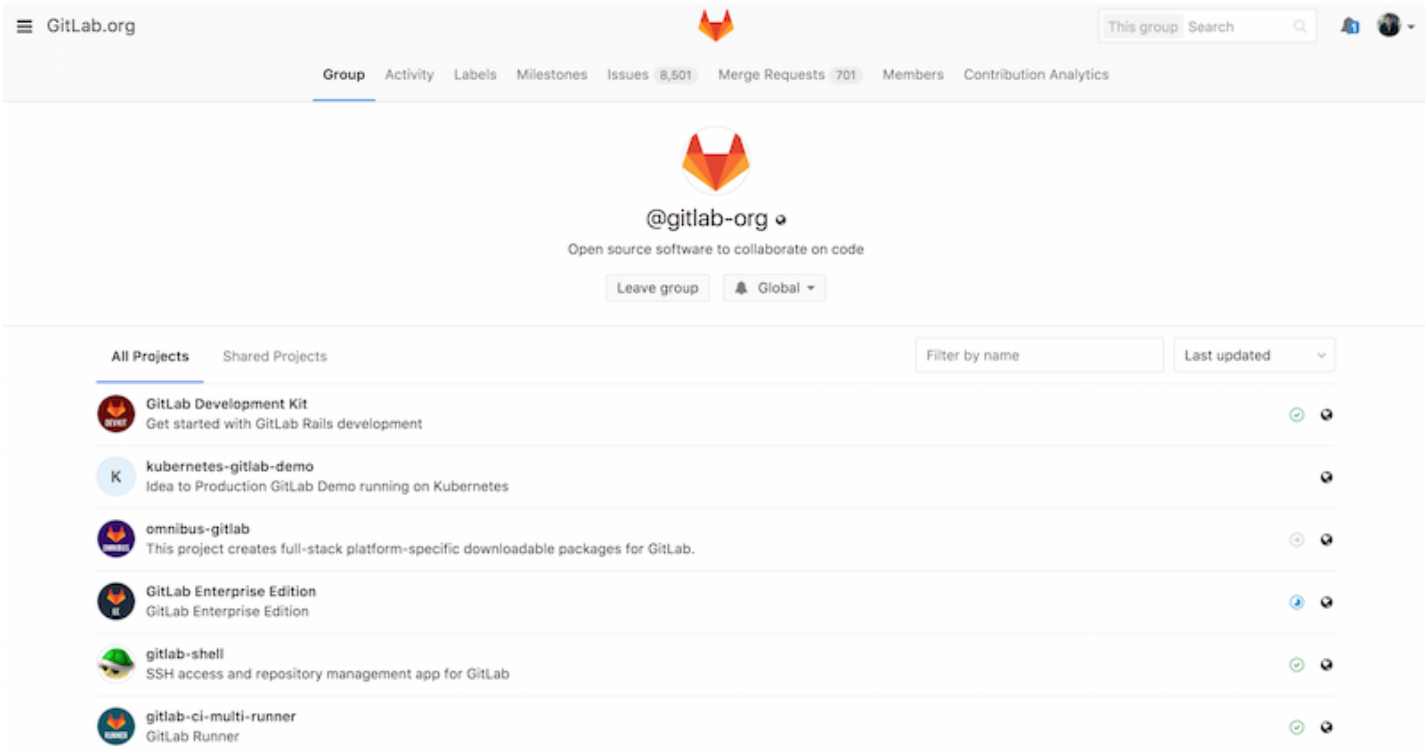

Figure 52. The GitLab group administration screen

Each group is associated with a number of users, each of which has a level of permissions for the group's projects and the group itself. These range from "Guest" (issues and chat only) to "Owner" (full control of the group, its members, and its projects). The types of permissions are too numerous to list here, but GitLab has a helpful link on the administration screen.

Projects

A GitLab project roughly corresponds to a single Git repository. Every project belongs to a single namespace, either a user or a group. If the project belongs to a user, the owner of the project has direct control over who has access to the project; if the project belongs to a group, the group's user-level permissions will take effect.

Every project has a visibility level, which controls who has read access to that project's pages and repository. If a project is *Private*, the project's owner must explicitly grant access to specific users. An *Internal* project is visible to any logged-in user, and a *Public* project is visible to anyone. Note that this controls both `git fetch` access as well as access to the web UI for that project.

Hooks

GitLab includes support for hooks, both at a project or system level. For either of these, the GitLab server will perform an HTTP POST with some descriptive JSON whenever relevant events occur. This is a great way to connect your Git repositories and GitLab instance to the rest of your development automation, such as CI servers, chat rooms, or deployment tools.

Basic Usage

The first thing you'll want to do with GitLab is create a new project. You can do this by clicking on the "+" icon on the toolbar. You'll be asked for the project's name, which namespace it should belong to, and what its visibility level should be. Most of what you specify here isn't permanent, and can be changed later through the settings interface. Click "Create Project", and you're done.

Once the project exists, you'll probably want to connect it with a local Git repository. Each project is accessible over HTTPS or SSH, either of which can be used to configure a Git remote. The URLs are visible at the top of the project's home page. For an existing local repository, this command will create a remote named `gitlab` to the hosted location:

```
$ git remote add gitlab https://server/namespace/project.git
```

If you don't have a local copy of the repository, you can simply do this:

```
$ git clone https://server/namespace/project.git
```

The web UI provides access to several useful views of the repository itself. Each project's home page shows recent activity, and links along the top will lead you to views of the project's files and commit log.

Working Together

The simplest way of working together on a GitLab project is by giving each user direct push access to the Git repository. You can add a user to a project by going to the "Members" section of that project's settings, and associating the new user with an access level (the different access levels are discussed a bit in Groups). By giving a user an access level of "Developer" or above, that user can push commits and branches directly to the repository.

Another, more decoupled way of collaboration is by using merge requests. This feature enables any user that can see a project to contribute to it in a controlled way. Users with direct access can simply create a branch, push commits to it, and open a merge request from their branch back into `master` or any other branch. Users who don't have push permissions for a repository can "fork" it to create their own copy, push commits to *their* copy, and open a merge request from their fork back to the main project. This model allows the owner to be in full control of what goes into the repository and when, while allowing contributions from untrusted users.

Merge requests and issues are the main units of long-lived discussion in GitLab. Each merge request allows a line-by-line discussion of the proposed change (which supports a lightweight kind of code review), as well as a general overall discussion thread. Both can be assigned to users, or organized into milestones.

This section is focused mainly on the Git-related features of GitLab, but as a mature project, it provides many other features to help your team work together, such as project wikis and system maintenance tools. One benefit to GitLab is that, once the server is set up and running, you'll rarely need to tweak a configuration file or access the server via SSH; most administration and general usage can be done through the in-browser interface.