Super Groups & Gitlab

What is Gitlab

In efforts to make super group development easier, we are utilizing a code management tool called Gitlab. Similar to Github in functionality and looks, it provides an easy way for the developers to write & push code and give the HUBzero team the security we needed to allow third party developers to commit code to production machines.

Why Gitlab

Overview

There are many benefits for both parties (the developer & the Hub team) to using a tool like Gitlab for managing the code of a super group.

Security

Live site access can be very dangerous for even a very experienced developer. The use of Gitlab removes the need to access the live site all together. The developer can code and test in whatever environement they want, add their changes and the hub can pull in the approved changes right through the HUB admin interface.

Gitlab also allows the HUB team to monitor and approve code after the developer has pushed their updates. Changes MUST be approved by the HUB team before they can be moved to the live site.

Developer Freedom

Developers will actually work on whats called a "forked" copy of the super group repository. This means they they are working on their very own version of the super group code and can do whatever they want to it without affecting the live site super group or any other developers also working on the super group.

Frequent Updates

Managing the super groups by Gitlab (outside of the main CMS), allows for more regular updates. At any point after the hub is configured to work with Gitlab, any hub admin who has access to the groups adminstrator panel can update the the super groups code.

Extra Features

Along with a nice code browser/editor, Gitlab comes with an Issue tracker and Wiki section for each project. Each project has the ability to use those sections however these please.

Setup

Hub Setup

Each HUB can choose to integrate with Gitlab or not. If your HUB chooses to integrate then there are few steps to get setup and running.

1. Gitlab integration must be enabled in the Groups config, under the "super" tab. The Gitlab API URL must also be supplied along with the API key of an admin account on Gitlab (found under the "account" tab in profile section in Gitlab). This allows the HUB to do the initial group/project/repository creation when the super group is created.
2. In order for the HUB (www-data user) to make the first commit to the project, including the basic super group template and folder structure, the www-data user must have an SSH Key on the HUB machine. That SSH key must also be added to an admin account on Gitlab. This first commit actually creates the GIT repository in Gitlab.
3. The last step for HUB setup is to SSH as the www-data user to the Gitlab machine from the HUB machine.. This will approve the RSA fingerprint of the Gitlab machine for the www-data user and add the machine to the known_hosts file. If this step is omitted the hubs attempt to make the inital commit will be denied.

Group Setup

When a super group is created on the HUB, most of the initial setup for that super group is done automatically. If you are working on a super group that was created prior to May 2014, then this setup will need to be done manually for that super group to work with Gitlab. Please enter a ticket through the support system detailing the the super group and that you would like to have your group integrated into Gitlab.

Developer Setup

Access

Part of the developer setup is getting permission to access Gitlab. This must be done manually by the HUBzero development team. Please submit a ticket indicating the super group and any users (name, preferred username, & email address) that will need to have access in Gitlab for the project.

Login & Password Change

Once you have submitted a ticket for access to Gitlab, you will recieve an email within 48 hours with all the details you need to login to your account. The email will contain a temporary

password that you will be forced to update upon login. After you login and change your password you can move on to the next step, uploading an SSH key.

SSH Keys

In order to make commits and push to Gitlab, you need to add an SSH key to your account. To add an SSH key, login to Gitlab, go to your profile, then SSH Keys. Click the "Add SSH Key" button, enter any title you want, paste your public SSH key in the box, and click "Add Key". If you are unsure of how to create an SSH key there is a link at the top of the "Add an SSH Key" page that links to a help page with detailed instructions.

Note: You can add multiple SSH keys if you plan to make commits from multiple machines.

Developing

Overview

After you have completed all the necessary setup steps its time to start actually developing. The following items are necessary steps to getting your code added to the live site super group code as easily as possible.

For example purposes we are going to use "mytestgroup" as the group cname and "hubzero.org" as the hub we are working on. This would map to "hubzero" as the group name and "mytestgroup" as the project in Gitlab. We are also going to use "theuser" as the user's username in Gitlab.

Fork Project

The first step is you need to create a fork of the main project. You can find main project by navigating to your dashboard in Gitlab then the projects tab. Project names are formatted by the group/project, where group is the hub name/URL and project is super group cname.

1. Click on the project you want to start development for, you should be taken to the project page.
2. Click the "Fork repository" button on the right side of the page. This will fork the repository and take you to your forked version of this repository.

Clone Repository

You are now ready to clone the repository to a development machine. This can be anywhere, but recommended that you use the hubs dev machine or local dev machine (local HUB on VM).

1. Get the repository url. From your forked repository page you should see a text box with the git repo url in it. Copy that URL to your clipboard
2. Go to the machine where you want to clone the repository to and type the following into a terminal window:

   ```
   git clone git@gitlab.hubzero.org:theuser/mytestgroup.git; mv myte
   stgroup/* mytestgroup/.git* .; rmdir mytestgroup;
   ```

3. The repository content will be copied to a "mytestgroup" directory within the current directory

Add Upstream Repository

Upstream repository is a fancy word for the main repository you forked from. You need to tell your forked copy that it has a main repository and where it is. To add the upstream repository, in a Terminal window navigate to your cloned repo and type the following:

```
git remote add upstream git@gitlab.hubzero.org:hubzero/mytestgroup.git
```

You can test to see if everything was added correctly by typing:

```
git remote -v
```

and you should now see something like:

```
origin    git@gitlab.hubzero.org:testuser/mytestgroup.git (fetch)
origin    git@gitlab.hubzero.org:testuser/mytestgroup.git (push)
upstream  git@gitlab.hubzero.org:hubzero/mytestgroup.git (fetch)
upstream  git@gitlab.hubzero.org:hubzero/mytestgroup.git (push)
```

This is a very important part of working with Gitlab is keeping your forked repository synced with the main repository.

Develop

Make changes, add new code, fix bugs etc. Commit as you develop.

Sync with Main Project

Before you push your changes to Gitlab it is recommended that you sync your forked project with the main project.

Failure to sync your fork before pushing changes and creating a merge request can result in your merge request being denied until synced properly.

To sync, navigate to your cloned repo in a terminal window and type the following:

```
git fetch upstream
```

Then make sure your on the master branch by typing:

```
git checkout master
```

Then merge the upstream master branch with your master branch by:

```
git merge upstream/master
```

You might have to resolve some merge conflicts at this point. See the Git documation or search Google for issues you might run into.

Note: You can sync your forked project with the main project as often as you like. Syncing often usually reduces potential merge conflicts.

Push Changes

Pushing your changes it simple and easy. Simple type the following in a terminal window from within your cloned repo:

```
git push origin master
```

This pushes the changes you've committed to your forked projects repository.

Create Merge Request

A merge request is how the changes you pushed to your forked project get into the main project. Login to gitlab, go to your forked project, click the merge tab, then "New merge request". Select the master branch in your forked copy and click "Compare branches". You should be taken to the next step where you can give the merge request a title and a description. The description is very important for the approval team to understand what the merge is related to. This page will also show the commits that will be merged and the file diffs. When your ready click "Submit merge request".

Wait for Approval

Approval may be the next day or make take up to a week depending on complexity and schedules. Approvals are done Monday-Friday 8am - 5pm EST.

When your merge request is accepted or denied you will get an email notice regarding its status.

Pull Changes

Pulling in the changes that were merged into the main project can be done through the admin interface for the HUB. You must have admin rights to access the administrator interface.