

# GitLab Tutorial

This is a tutorial aimed to show some features of [GitLab](#) that we consider interesting for our projects. The topics covered are:

Getting the code.....	2
Create a new branch and committing to it.....	2
Working on an existing branch.....	3
Branch protection.....	4
Commits.....	5
Comparing Commits .....	5
Commenting on Commits .....	6
Tagging Commits .....	7
Creating Merge Requests.....	8
Creating a Bug Report.....	9
Checking File History.....	10
Determine differences between local and repo .....	10

## Getting the code

In order to complete this tutorial you need to clone a repository first. This command will create a local version of the selected repository. We have made a special project, `gcadj_test`, that you can use for this tutorial.

```
git clone git@adjoin.t.colorado.edu:yanko.davila/gcadj_std.git
```

Please note that you need an account on [GitLab](#) in order to clone the repo. If don't have an account you can request one by following the instructions [here](#).

## Create a new branch and committing to it

For developing major new code features, we recommend that users create their own code branch, and commit this branch back to the GitLab server on a regular basis. The benefit of this approach is that

- If you have a question about your code, we (Yanko, Daven, ...) can see the entire code and comment directly on it without you having to email code snippets back and forth
- It is easier in the long run to merge your branch back into the master
- You have a copy of your developments backed up on a remote server!

So, when you finished working on some new code development, the first step is to create a new branch, then move your changes to that new branch and commit them,. This can be done through the command line by

```
git branch branch_name ( Create branch )  
git checkout branch_name ( Move changes to branch )  
git commit  
git push origin branch_name (Commit branch to server)
```

Or this

```
git checkout -b branch_name ( Create and move changes to branch )  
git commit  
git push origin branch_name (Commit branch to server)
```

## Working on an existing branch

Sometimes you will want to work on further developing an existing branch. In order to do so, pull / fetch the latest version of the repository from the remote server (see section XX). You can see the list of available branches with

```
git branch -a
```

An asterisk will appear next to the branch that you are currently working on.

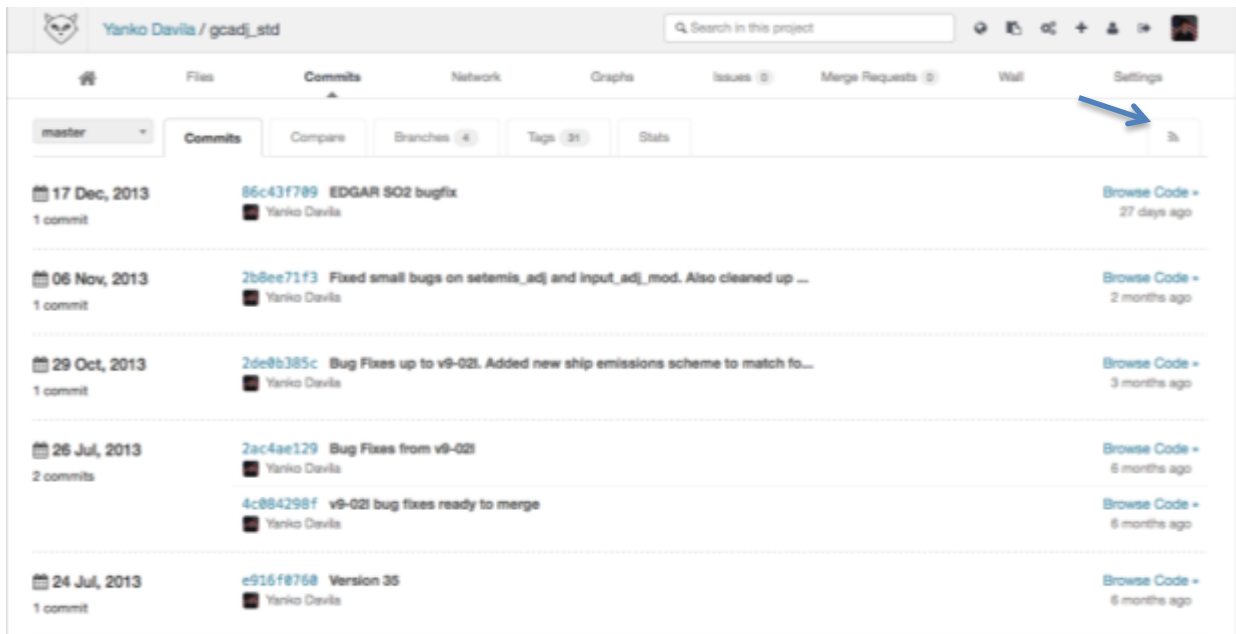
Next, checkout the branch that you want to work on:

```
git checkout branch_name
```

When you are done working commit your changes

```
git commit
```

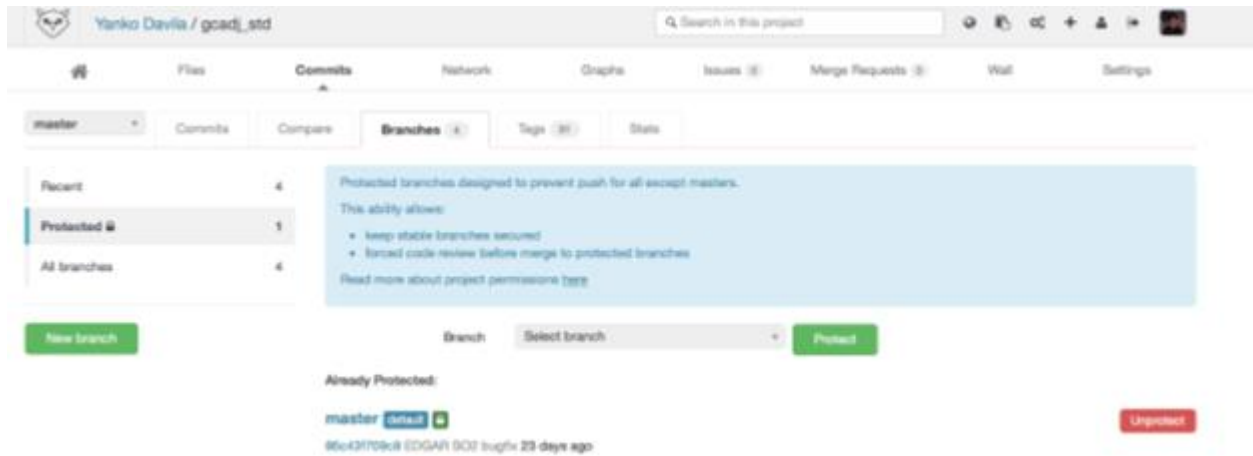
Note: If your project has several users working on the same branch, it is recommended that all of your colleagues use an rss feed reader and follow the feed located on the commits section.



## Branch protection

Depending on your project you may or not use this, but we think is important for you to know. When you create a new project on GitLab, the master branch is writable by all “Developers”. If you want a centralized development where only “Masters” can write to the master branch it is recommended that you protect the master branch. This is done by going to Commits -> Branches -> Protected and you will see something like this.

Note: When we refer to the master branch we mean the main branch of development, later on when new features are developed they will eventually be merged with this one.



From there, select the branch that you want to protect / unprotect.

## Commits

The history of commits to each project can be seen from the “Commits” page. Here is a sample of some of the information that you can get from the commit list.



## Comparing Commits

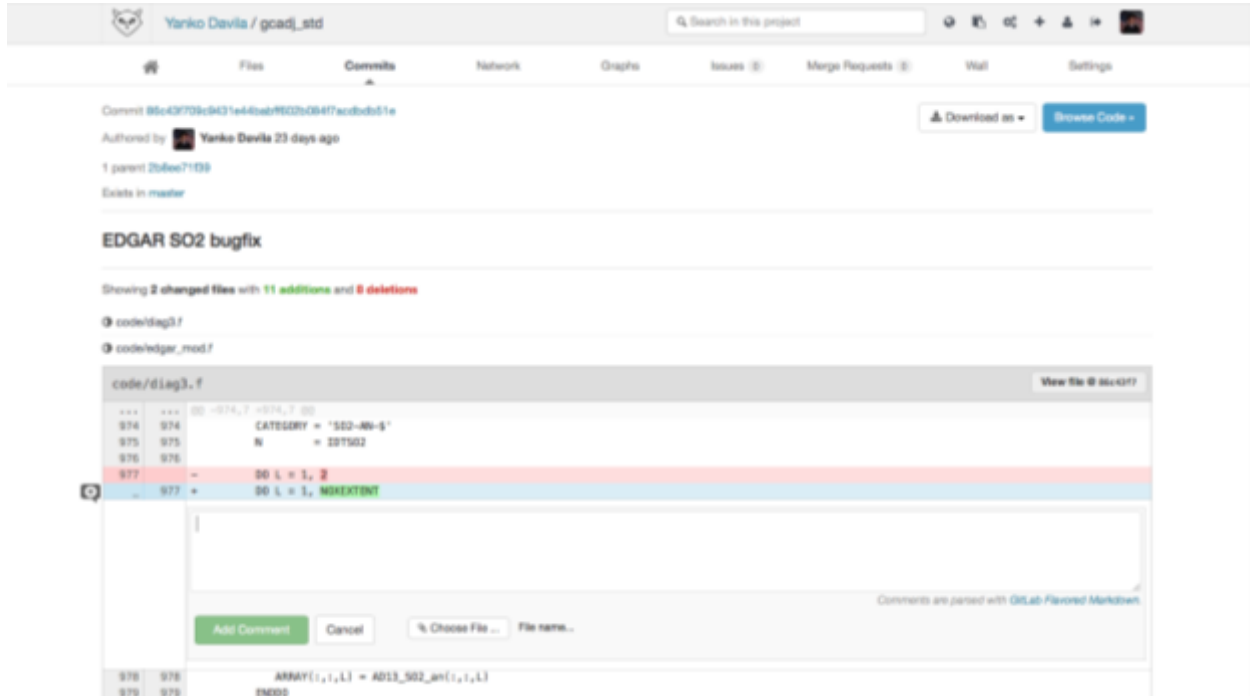
To compare commits, go to Commits -> Compare and you will be presented this:



There you can compare commits by entering the first 8 characters of the commit, also you can compare tags and even branches. Note that you can mix all the above, so for example you can compare a tag to a whole branch.

## Commenting on Commits

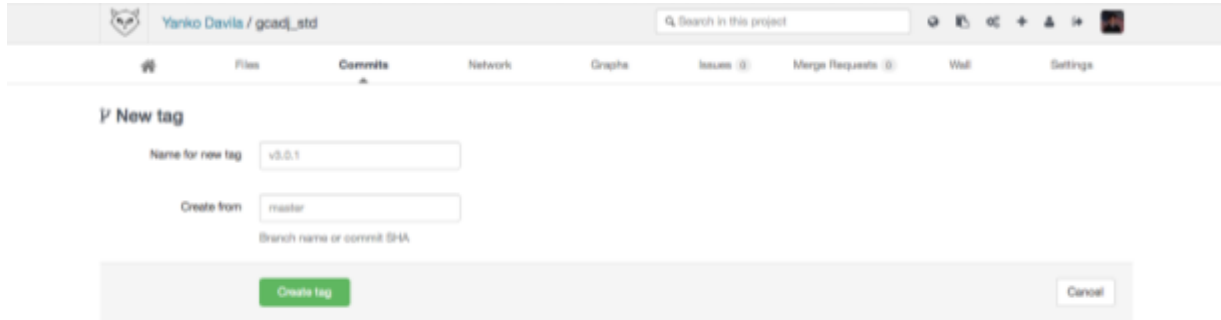
To comment on commits, go to Commits -> Commits -> 'Click commit number' and you will be presented this:



When you pass your pointer over the lines you will see a + on the left. If you click the + you will be presented with the comment tool, after you hit add comment an email will be sent to the committer. This is a good way to start a discussion about certain code changes or updates.

## Tagging Commits

To tag commits, go to Commits -> Tags -> 'Click New Tag' (right corner) and you will be presented this:



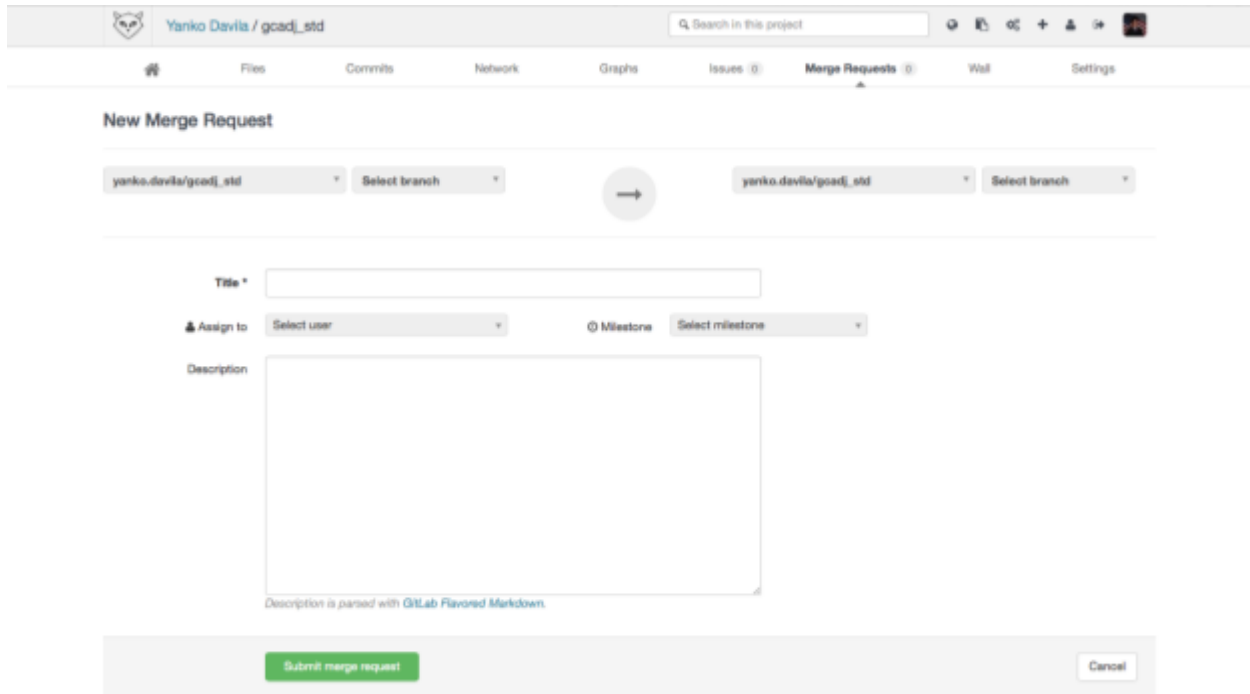
The screenshot shows the GitLab interface for creating a new tag. At the top, the user is identified as 'Yanko Davila / goadj\_std' with a search bar for the project. The navigation menu includes 'Files', 'Commits', 'Network', 'Graphs', 'Issues', 'Merge Requests', 'Wall', and 'Settings'. The 'Commits' section is active, and the 'New tag' form is displayed. The form has two input fields: 'Name for new tag' with the value 'v3.0.1' and 'Create from' with the value 'master'. Below the second field, it says 'Branch name or commit SHA'. At the bottom of the form, there are two buttons: a green 'Create tag' button and a grey 'Cancel' button.

There write the tag name and the commit number.

Note: tagging is not available for "Reporters". Only "Developer and Masters" can do this.

## Creating Merge Requests

To create a new merge request, go to Merge Requests -> 'Click New Merge Request' (right corner) and you will be presented this:



The screenshot shows the 'New Merge Request' form in GitLab. At the top, there is a navigation bar with the user 'Yanko Davila / gcodj\_std' and a search bar. Below the navigation bar, there are tabs for 'Files', 'Commits', 'Network', 'Graphs', 'Issues', 'Merge Requests', 'Wall', and 'Settings'. The 'Merge Requests' tab is active. The form itself is titled 'New Merge Request' and has two dropdown menus for selecting source and destination branches, both currently set to 'yanko.davila/gcodj\_std'. Below these are fields for 'Title', 'Assign to' (with a dropdown for 'Select user'), and 'Milestones' (with a dropdown for 'Select milestones'). There is a large text area for 'Description' with a note that it is parsed with GitLab Flavored Markdown. At the bottom, there are two buttons: 'Submit merge request' (green) and 'Cancel' (grey).

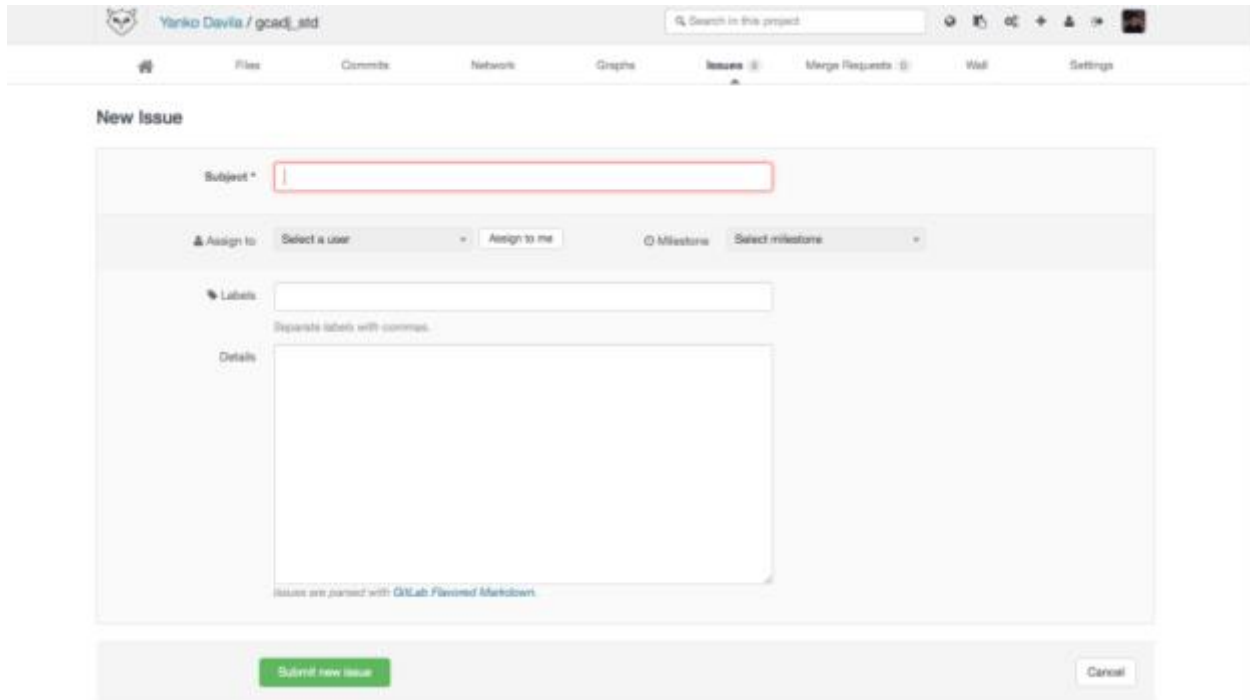
On the left select the source branch (the one with the updated code), on the right select destination branch. You can also assign a user to take care of this merge request. In GEOS-Chem Adjoint the only users who can merge to the master branch are Daven Henze, Nicolas Bousserez and Yanko Davila, but the preferred one is Yanko Davila. Also you can give a title and a brief description to your request.

Note: Merge Request is not available for “Reporters”, only “Developer and Masters” can do this.



## Creating a Bug Report

To create a new bug/error report, go to Issues -> 'Click New Issue' (right corner) and you will be presented this:

The image shows a screenshot of the GitLab web interface for creating a new issue. At the top, the user 'Yanko Davila / goad\_std' is logged in, and there is a search bar. The navigation menu includes 'Issues' which is currently selected. The main content area is titled 'New Issue' and contains several input fields: a 'Subject' field with a red border, an 'Assign to' dropdown menu set to 'Select a user' with an 'Assign to me' button, a 'Labels' field with a note 'Separate labels with commas', and a large 'Details' text area. At the bottom, there is a green 'Submit new issue' button and a grey 'Cancel' button. A small note at the bottom of the details area says 'Issues are parsed with GitLab Flavored Markdown.'

Fill the information requested and hit submit to create the report and send an email to the “Assigned User”. If you know who wrote the section of the code that is having problems you should make that person the assigned user, otherwise use Yanko Davila. This feature can be used as knowledge base to avoid sending emails or answering the same question several times. Once the issue has been resolved, the assigned user can close the issue and it will be archived. All the users of the project have access to Issues and can comment on them.

## Checking File History

To check the history of a file you can use the command 'git log -path to file' or go to Files -> 'Navigate to find the file' -> History and you will be presented with the list of commits where that file was changed.

## Determine differences between local and remote repositories

Sometimes you may want to check the difference between your local copy of a file and the latest one on the server. There are two ways to do so. The first one is

```
git diff --color origin/master -path to file
```

This will go to the master branch on the server and compare your copy vs that one. The other way is to update your local copy of the repo. Please note that this will only update the history so it will not try to apply any changes (read more at <http://stackoverflow.com/questions/292357/whats-the-difference-between-git-pull-and-git-fetch>).

```
git fetch  
git diff --color master -path to file
```