Source Control with Gitlab + Git

14-642 Embedded Real-Time Systems

Contents

1 Introduction

For this course, we will be using the git version control system to distribute labs, manage submissions, and give feedback. Version control systems let you checkpoint, backup and view version history of your project.

There are multiple front-ends for git like Gitlab, Github, and Bitbucket. For this course we have a Gitlab server running at `https://embedded.andrew.cmu.edu:5443/`. This is a private instance hosted on a campus VM with Andrew authentication. This private server will give you, your partner, and the staff an easy place to collaborate that doesn't require using your GitHub student perks or similar. Starter code this semester is available in the `642-TAs/642-f16` repository.

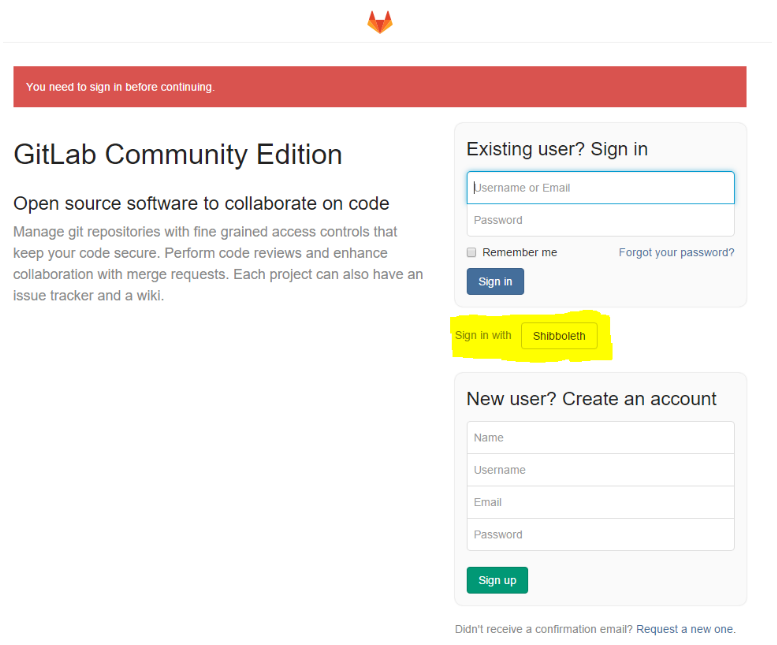
You may use whatever Git client on your machine you wish. Most of the course staff is familiar with the git command line tool and the directions will be provided for it. If you want a Git gui, the course staff is familiar with SourceTree from Atlassian.

PLEASE follow these instructions carefully since you need to correctly configure your project in order to receive credit for the lab.

2 Gitlab

2.1 Signing in to Gitlab

Open a browser and go to `https://embedded.andrew.cmu.edu:5443/`. You can also get here from the link on our class website (`http://embedded.andrew.cmu.edu/`). Press the sign-in with shibboleth button. This will create you a account based on your andrew id.



You need to sign in before continuing.

GitLab Community Edition

Open source software to collaborate on code

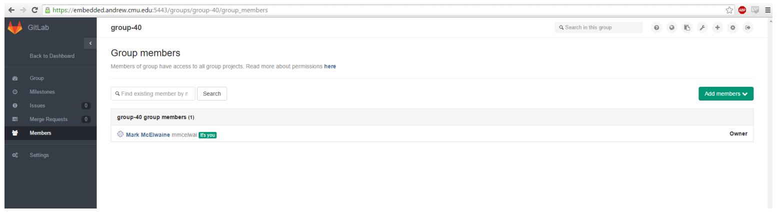
Manage git repositories with fine grained access controls that keep your code secure. Perform code reviews and enhance collaboration with merge requests. Each project can also have an issue tracker and a wiki.

Didn't receive a confirmation email? Request a new one.

Not signing up using your andrew account will give you no credit since we won't be able to find your repository.

2.2 Creating a Group

The projects in this class are to be completed by groups of 2. So navigate to the groups tab on the left side of the screen.



https://embedded.andrew.cmu.edu:5443/groups/group-40/group_members

group-40

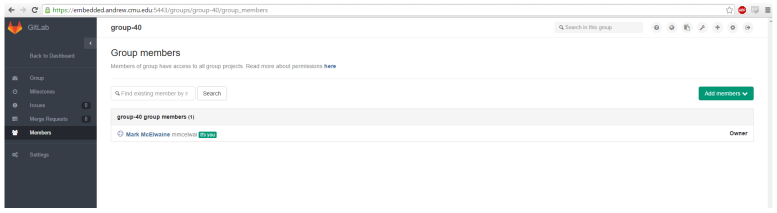
Group members

Members of group have access to all group projects. Read more about permissions here

group-40 group members (1)

Mark McElwaine mmcelwai It's you — Owner

¡¡¡¡¡¡¡ HEAD



https://embedded.andrew.cmu.edu:5443/groups/group-40/group_members

group-40

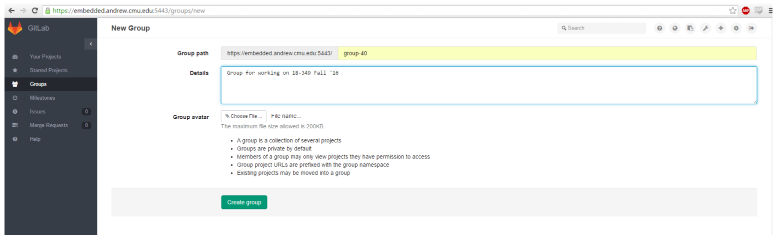
Group members

Members of group have access to all group projects. Read more about permissions here

group-40 group members (1)

Mark McElwaine mmcelwai It's you — Owner

Then click on the New Group button at the top right corner of the screen. This brings up a page asking for a group path and a description. name the group path ”team<number>”. If you have a single-digit team number, use a zero before it, e.g. ”team07”. Fill out the group details section with a simpe group description. Then press Create Group. ======= Then click on the New Group button at the top right corner of the screen. This brings up a page asking for a group path and a description. name the group path group -<your group number>. This doubles as the group's name. Fill out the group details section with a simpe group description. Then press Create Group. ¿¿¿¿¿¿¿ 6edacff936de9483dc0705b8094c4a7e1b6d9b24



https://embedded.andrew.cmu.edu:5443/groups/new

New Group

Group path: https://embedded.andrew.cmu.edu:5443/ group-40

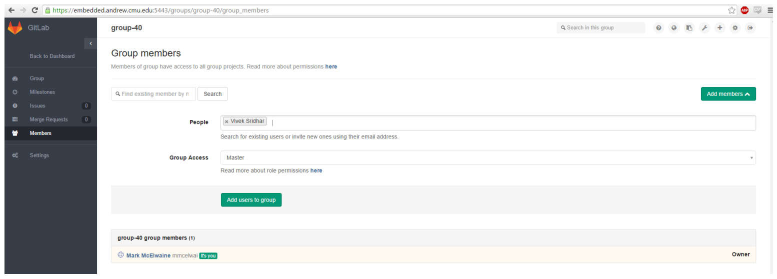
Details: Group for working on 18-349 Fall '16

Group avatar: Choose File ... File name...

The maximum file size allowed is 200KB.

- A group is a collection of several projects
- Groups are private by default
- Members of a group may only view projects they have permission to access
- Group project URLs are prefixed with the group namespace
- Existing projects may be moved into a group

Now you have to add your partner to the newly created group. Click on the members tab on the left side of the screen. Now click on Add Members on the upper right side of the screen. This brings up a people section and Group Access section. From the people bar add your partner and the andrew id viveksri. Give them master access and the click Add users to group. Your group is now ready.



https://embedded.andrew.cmu.edu:5443/groups/group-40/group_members

group-40

Group members

Members of group have access to all group projects. Read more about permissions here

People: Vivek Sridhar

Search for existing users or invite new ones using their email address.

Group Access: Master

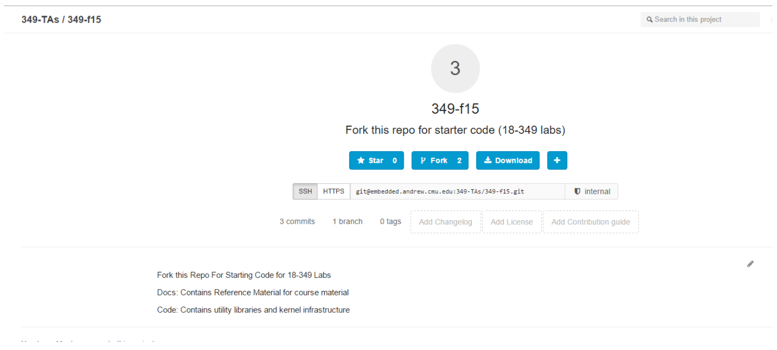
Read more about role permissions here

group-40 group members (1)

Mark McElwaine mmcelwai It's you — Owner

2.3 Forking

Navigate to `https://embedded.andrew.cmu.edu:5443/642-TAs/642-f16/`. This is the starter code repository from which you will be establishing your fork of the project. Press the fork button.



349-f15

Fork this repo for starter code (18-349 labs)

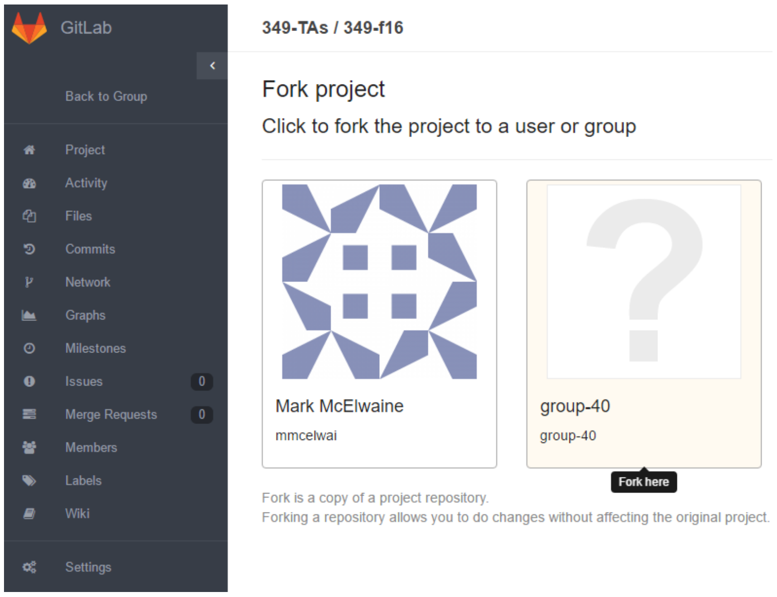
git@embedded.andrew.cmu.edu:349-TAs/349-f15.git

Fork this Repo For Starting Code for 18-349 Labs

Docs: Contains Reference Material for course material

Code: Contains utility libraries and kernel infrastructure

Click the box with your group name inside of it. This creates a new repository for you with all the starter code needed. Conveniently, the path is linked to your group name as `https://embedded.andrew.cmu.edu:5443/<group_name>/642-f16`.



Fork project

Click to fork the project to a user or group

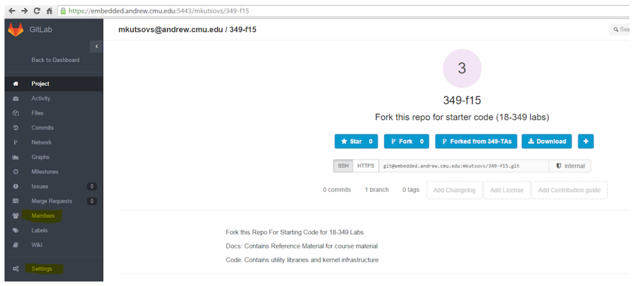
Fork is a copy of a project repository.
Forking a repository allows you to do changes without affecting the original project.

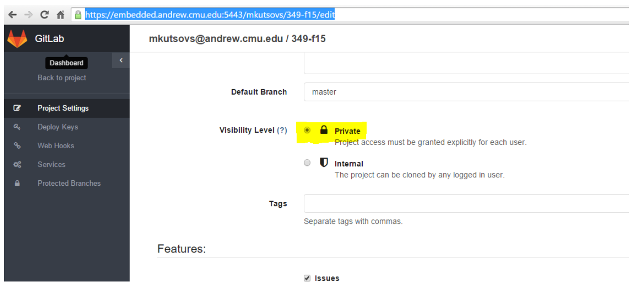
A fork is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project. Most commonly, forks are used to either propose changes to someone else's project or to use someone else's project as a starting point for your own work. The latter is how it is used in this course.

2.4 Repository Permissions

The repository you forked had an *internal* security setting meaning that anyone logged in with their Andrew ID can see it. However, when you push code we don't want you to share it with the class. You must set your repository to *private*. When you get a partner in the later labs, you can add them as a collaborator individually.

To change the privacy settings, go to the *Settings* page from the hope page of your repository (below left). Select *Private* and save (below right).


https://embedded.andrew.cmu.edu:5443/mkutsovs/349-f15
GitLab
Back to Dashboard
Project
Activity
Files
Commits
Network
Graphs
Milestones
Issues
Merge Requests
Members
Labels
Wiki
Settings
mkutsovs@andrew.cmu.edu / 349-f15
349-f15
Fork this repo for starter code (18-349 labs)
Star
Fork
Forked from 349-TAs
Download
SSH
HTTPS
0 commits
1 branch
0 tags
Fork this Repo For Starting Code for 18-349 Labs
Docs: Contains Reference Material for course material
Code: Contains utility libraries and kernel infrastructure


https://embedded.andrew.cmu.edu:5443/mkutsovs/349-f15/edit
GitLab
Dashboard
Back to project
Project Settings
Deploy Keys
Web Hooks
Services
Protected Branches
mkutsovs@andrew.cmu.edu / 349-f15
Default Branch
master
Visibility Level (?)
Private
Project access must be granted explicitly for each user.
Internal
The project can be cloned by any logged in user.
Tags
Separate tags with commas.
Features:
Issues

You can share your repository with your partner later using the *Members* page, also highlighted (above left).

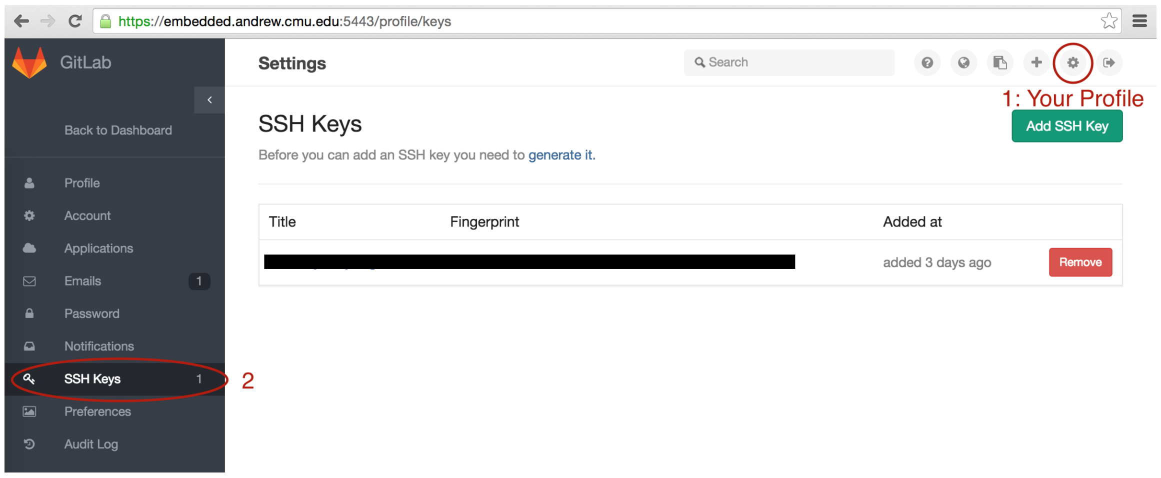
2.5 Adding a SSH Key (to Authenticate Your Clones)

NOTE: the 642 gitlab does not support the HTTPS interface, so you must use SSH.

When you connect git on your working machine to the remote repository it will need to authenticate to see your private repository. Typically, you can use either the http protocol with password or the ssh protocol with ssh keys. However, git doesn't understand how to use Andrew ID passwords. We need to use SSH keys instead.

Generate an SSH key according to this tutorial: `https://help.github.com/articles/generating-ssh-keys/`. If you already have a key in `~/.ssh/id_rsa.pub` you can use that one instead of making another.

In Gitlab you can add the key on the *Profile settings >SSH Keys* page seen below.



SSH Keys

Before you can add an SSH key you need to generate it.

Title	Fingerprint	Added at
[redacted]	[redacted]	added 3 days ago

Note that the github docs ask you to test with the server git@github.com . Since we're using our own server, test against git@embedded.andrew.cmu.edu .

3 Using Git

If you are unfamiliar with the basic git operations please review instructions such as `clone`, `pull`, `add`, `commit`, `push`. There are multiple good git tutorials online including: `https://www.atlassian.com/git/tutorials/`

3.1 Getting Updates

As new labs and material get released, the instructors may add additional content which you will be able to sync from. To get the updates, you need to set up an upstream remote in your git client.

To set up the remote for the first time using the git CLI, run the following commands. We can see that it is added by listing remotes again.

```
$ git remote
origi
$ git remote add upstream git@embedded.andrew.cmu.edu:642-TAs/642-f16.git
$ git remote
origin
upstream
```

Each time you would like to update, pull from the upstream into master. To make these new commits show up in your repo, git push normally.

```
$ git pull upstream master
remote: Counting objects: 4, done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 1), reused 0 (delta 0)
Unpacking objects: 100% (4/4), done.
From embedded.andrew.cmu.edu:642-TAs/642-f16
 * branch            master     -> FETCH_HEAD
 * [new branch]      master     -> upstream/master
Updating 10a5e52..5c34b14
Fast-forward
 README.md |   6 +++---
 lab1.pdf  | Bin 214591 -> 214442 bytes
 2 files changed, 3 insertions(+), 3 deletions(-)
```

If you have issues with conflicts on pull refer to the *Merge Conflicts* section to try to resolve the issue.

3.2 Merge Conflicts

Although not as common in individual projects, different edits can change the same sections of a file, requiring a manual resolution process to determine how the conflicting code segments go together. When you're working with a partner or pulling changes from the course staff, git may complain that you and somebody else changed the same line of code in different ways. This is called a `git merge` and such a conflict needs to be manually resolved.

For example, if I am working on master and try to merge a branch that my partner is working on, I might get the the following error from merge. `git status` also reports which file was unable to merge.

```
$ git merge partners_branch
Auto-merging code/kernel/src/kernel.c
CONFLICT (content): Merge conflict in code/kernel/src/kernel.c
Automatic merge failed; fix conflicts and then commit the result.
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
You have unmerged paths.
  (fix conflicts and run "git commit")

Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both modified:   kernel/src/kernel.c
```

If you open the `kernel/src/kernel.c` where there is a conflict, you can see output like the following:

```
...
void kernel_main(void) {
  while (1) {
    printk("Hello world!\n");
  }
}
...
```

To resolve the conflicts, you need to edit each file in the unmerged list and look for `<<<<<<<` and `>>>>>>>` which mark the beginning and end of each conflict. You have the option of choosing either the top, bottom, or some combination of the 2. Make sure you you remove the `<<<<<<<`, `>>>>>>>`, and `=======` text that git adds.

After you finish resolving conflicts you need to add and commit them to finish the merge in git. You will probably see output like below.

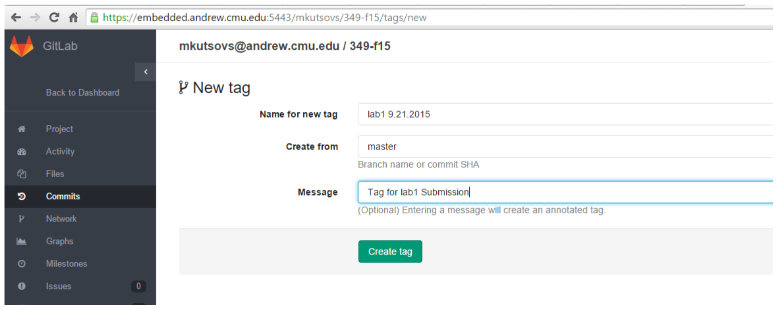
```
$ git add kernel/src/kernel.c
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
All conflicts fixed but you are still merging.
  (use "git commit" to conclude merge)
$ git commit
[master 5e138a8] Merge branch 'partners_branch'
```

4 Submission

First, you need to commit and push your changes:

```
git add <each file you have edited>
git commit -m "a short commit message"
git push
```

Submitting using Gitlab is easy. Tag your committed code with the expected phrase, such as `lab0-submit`. You can find the handin tag at the end of each lab writeup. Go to `https://embedded.andrew.cmu.edu:5443/<andrew_id>/642-f16/tags/` to add a tag online or use the `git tag` command to make it in the CLI.



mkutsovs@andrew.cmu.edu / 349-f15

New tag

Name for new tag: lab1 9.21.2015

Create from: master

Branch name or commit SHA

Message: Tag for lab1 Submission

(Optional) Entering a message will create an annotated tag.