Lab: Git

This is definitely not an exhaustive tutorial about learning Git ... Google would be better to reveal several great tutorials about Git. What we focus on instead is simplistic workflow about publishing a "Pull Request" in Git.

Part 0: Setup Gitlab account

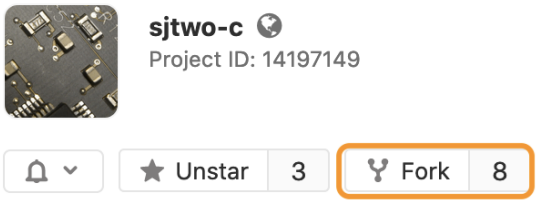
For better or worse, we have decided to use Gitlab.com for the repository. You are also required to use this Gitlab repository because that keeps the entire class aligned to a single server type and reduces fragmentation while increasing efficiency of the teacher and the ISA team.

For this part, establish your Gitlab.com account. In addition, also install Git to your machine such that you can successfully execute the `git` command from a terminal.

Part 1: Fork SJ2-C Project

When you fork a project, you essentially create a copy of the original SJ2-C repository. This will be your version of the forked project, and you can use this throughout the semester for your private workspace to do the lab assignments.

Browse to the SJtwo-c repository, and click on the Fork button.



After you fork the repository, make sure you set the permissions to "public". Do this by going into your newly forked repository settings, and look for "Visibility" setting.

Part 2: Branch Workflow

The process of checking-in new code to your forked repository will involve "Branch Workflow". There are actually a number of ways to contribute code to your repository, and the branch workflow is just one of them that we will choose to use.

We are not going to discuss that in detail because it is already captured well at **this awesome article**. We will summarize the process that you will use to do this. The `$` indicates the commands you should try.

```
# See what is going on
$ git status
On branch master

# Create a new "branch" of code to work on
# You can use any name, and feature/foo is just a convention
$ git checkout -b feature/gpio_blinky_in_periodics
Switched to a new branch 'feature/gpio_blinky_in_periodics'

# Add or modify a file we want
```

```
$ touch file.txt

# Tell git to add it to be committed
$ git add file.txt

# Check what is going on
$ git status
On branch feature/gpio_blinky_in_periodics
Changes to be committed:
(use "git reset HEAD <file>..." to unstage)

new file: file.txt

# Commit the change with a message
$ git commit -m "added file.txt"
[feature/gpio_blinky_in_periodics 5f76839] added file.txt
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 file.txt

# Check what is going on
$ git status
On branch feature/gpio_blinky_in_periodics
nothing to commit, working tree clean
```

Part 3: Merge Request (MR)

The typical name for a request to merge code is called a "Pull Request" or a "Merge Request". This is the chance to review the code, and merge the code. At the end of `Part 2`, you have a branch that only exists on your computer. In case you lose your computer or your storage device dies, then you **will lose** any work even though you have "committed" a change.

The distinction is that a commit only commits to your storage device, but does not send the data or the branch to the Git server. To actually push the code to the Git server, simply type `git push origin head`.

```
$ git push origin head
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 262 bytes | 262.00 KiB/s, done.
Total 2 (delta 1), reused 0 (delta 0)
remote:
remote: To create a merge request for feature/gpio_blinky_in_periodics, visit:
remote:   https://gitlab.com/sjtwo-c-dev/sjtwo-c/-/merge_requests/new?merge_request%5Bsource_br
remote:
To gitlab.com:sjtwo-c-dev/sjtwo-c.git
 * [new branch]      head -> feature/gpio_blinky_in_periodics
```

This command will generate a URL for you, so copy and paste this URL to your web browser. For example, the URL above is:
`https://gitlab.com/sjtwo-c-dev/sjtwo-c/-/merge_requests/new?merge_request%5Bsource_branch%5D=feature%2Fgpio_blinky_in_periodics`

This will lead you to generate your "Merge Request". At the end of the webpage that loads, click on "Submit Merge Request". At this point, you can view the changes, get feedback from others, and if the code looks good, you can then merge the code. But wait ... rarely will you be able to merge code without iterating and revising it, and that is what the Part 4 is for.

Part 4: Revise an MR

Granted that you have an MR already out there, and you have got feedback from others, this section will teach you how to revise or amend your code.

```
# Modify any code
# In this case, we will dump 'hello' to our previously committed file: file.txt
$ echo "hello" >> file.txt

# Check what is going on
$ git status
On branch feature/gpio_blinky_in_periodics
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   file.txt

# Add the file we want to re-commit (another commit on top of previous)
$ git add file.txt
$ git commit -m "Added hello to file.txt"

# Update the remote branch and the Merge Request
$ git push origin head
```

After the `git push` command, your MR will be updated on the browser. This way, you can continue to revise your MR per suggestions of other people. When you are satisfied with your MR, you can seek approval and officially hit the Merge button on the Gitlab.com webpage.

Part 5: Final Step

After you have merged your MR, it is time to go back to the master branch and grab the latest changes. Other users may have merged their code also, so pulling the latest master branch is going to get you the latest and greatest code.

```
# Go to the master branch
$ git checkout master

# Pull the latest master
$ git pull origin master
```

Part 6: Going beyond . . .

There is of course A LOT more to Git, but once you master the basics, you can then Google your way through the rest of the world you will face such as:

- Handling merge conflicts

- Checkout other people's branches

Rebase on the latest master branch.

```
$ git status
# Assume that you are on your feature branch

$ git checkout master
$ git pull origin master

# Go back to the previous branch you were working with (feature)
$ git checkout -

# Apply our commits to the latest master
$ git rebase master
```