Contributing to 802-COM by creating Feature Branches and Merging – v1.1

Adam Gregory, Samtec

Richard Mellitz, Samtec

Kent Lusted, Synopsys

Revision History

Version #	Notes
1.0	Original release – May 2025 interim
1.1	Added details to step 5 on file modification and creating the release file

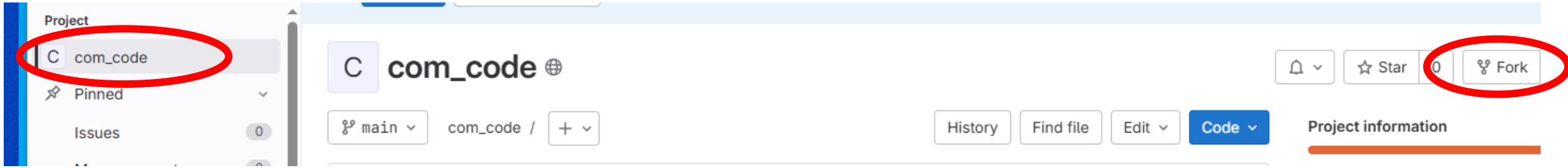
Summary

- The main idea is to work within a Feature Branch of a Forked Repository instead of working on the main code branch. This allows the developer to make any change they want without affecting the stable main branch
- Note that if you are not a Maintainer of 802-COM, it is only possible to work on a Forked Repository
- Summary of steps to work in a branch:
    1. Create a Fork repository of 802-COM into your name space
    2. Create a new branch on your Forked repository
    3. GIT: clone the repository
    4. GIT: Checkout the new branch
    5. Proceed with normal code updates: commits, pushes, etc…
    6. When finished, do a Merge Request from the COM Open-Source IEEE page
    7. After the Merge Request is approved, update your Fork

Step 1: Fork 802-COM into personal namespace

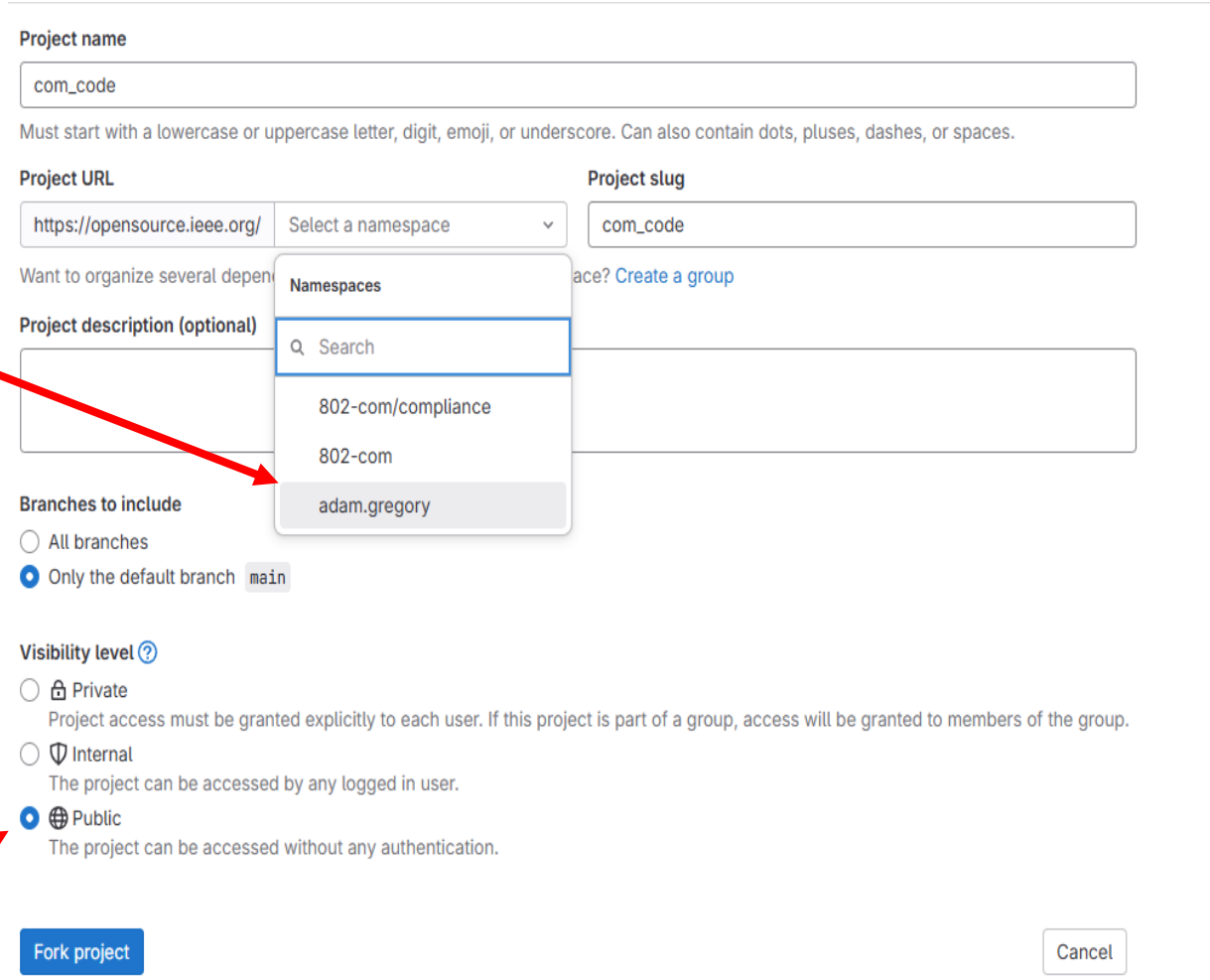
- A project can be Forked to a user's personal space. Then the user can create branches on their personal repository. After finishing, a Merge Request is submitted FROM the forked repo TO the original repo
- Note that creating the Fork is a one-time process. For future contributions, continue using the same fork repository

- Navigate to 802-COM website
  - https://opensource.ieee.org/802-com/com_code
- Select com_code under project
- Click the Fork button. Follow steps on the next slide.



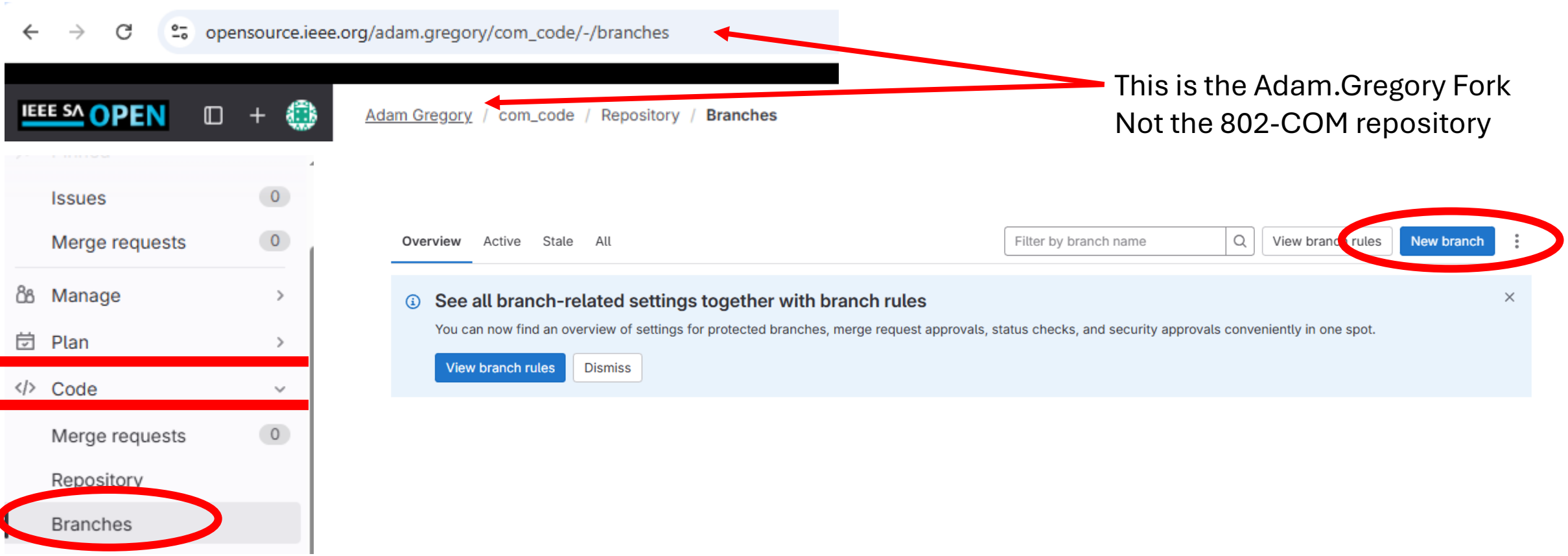
Step 1: Fork 802-COM into personal namespace

- After clicking Fork, the dialog shown here will appear
- The main idea is to select the namespace for the current user
  - For the example here: adam.gregory
- This creates an entirely new project that can be cloned as normal to your computer
- The fork can include all branches or just the main branch.
  - This is a user choice. The typical flow is to include only the main branch since the idea is to work on a completely new branch after forking.
- It is strongly recommended to keep the Fork at Public Visibility
  - Setting to Private could create problems with submitting Merge Requests



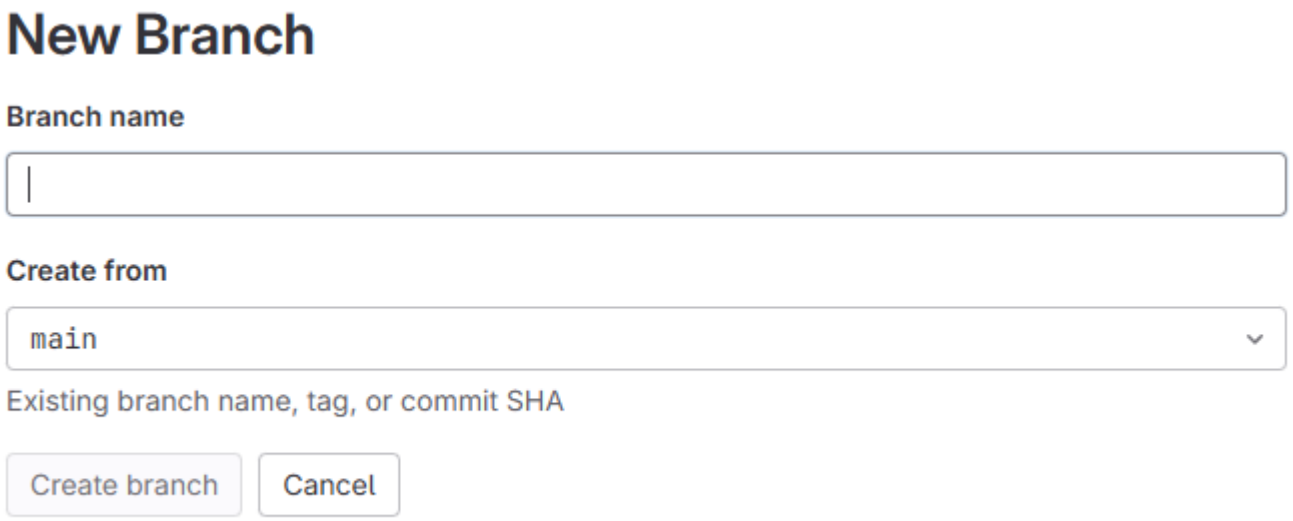
Step 2: Creating a New Branch

- Open the forked repository in your personal namespace
- In the left panel, expand the Code option and select Branches
- Click the New Branch button in the top right



Step 2: Creating a New Branch

- Give the branch a name. Usually a description of the feature or bug fix the branch is addressing.
- Choose what this branch is based on. In most cases, it will be based on “main” branch. It is possible to base the branch on other commits or branches though.



Step 3: Clone (or Pull) the remote repository

- If the repository hasn't been cloned yet, do a git clone
  - Open git bash
  - Navigate to the folder where the repository will be cloned
  - Run: git clone
  - Note that the path shown here is for the adam.gregory Fork. Your Fork will be based on your name.

```
adamg@SCDC-ADAMGR MINGW64 ~
$ git clone https://opensource.ieee.org/adam.gregory/com_code.git
```

- If you have instead made the branch on a repository that has already been cloned, just do a git pull to retrieve the branch.

```
adamg@SCDC-ADAMGR MINGW64 /c/matlab_tools/OSCOM/com_code (Optimize_FOM_Reduce)
$ git pull
```

Step 4: Checkout branch

- From the git bash terminal, run the following command:
  - git checkout <branch name>
  - For a branch named My_Feature:
    - git checkout My_Feature
- This will change all files in your working directory to match the data in the current branch. Run this command to go back to the main branch:
  - git checkout main
- Run this command to see all branches that are available. (Note that you need to check out a branch before it is listed)
  - git branch

Step 5: Working on a branch

- Once the branch is checked out, files can be modified, and git commits/pushes can be done without having any effect on the main branch.
- This means you can do a git checkout to swap between different working states. Git checkout modifies the files on the disk of your working area to match the state of the chosen branch.
- Important: Make sure you have checked out the intended branch before modifying files!
- For those not familiar with using Git, there are a few tutorial slides at the end of this document.

- Note: Any tracked file that is modified must be committed or stashed before changing to a different branch. See slides at the end of the presentation regarding Git Stash.
  - A new file that has not been committed doesn't need to be stashed. Stashing only applies to files whose contents would be modified by changing branches.

Step 5: Working on a branch (Instructions on file modification)

- Only modify the functions in the “src” folder
- Never modify or create functions in the “release” folder

- The “release” folder contains the release .m files which contain all subfunctions in a single file. These files are auto-generated using the “make_release” function.
- It is not required to run the make_release function, but it will allow users to run your branch updates without downloading the entire git repository.
- See the next slide for instructions on running make_release.

Step 5: Working on a branch (Making a release file)

- Command line for running make_release:
  - `Management.make_release('com_ieee8023_.m','<NAME>');`
  - <NAME> = the string that will be appended to the end of com_ieee8023_. If <NAME> = 'TEST', then the output file will be "com_ieee8023_TEST.m"
- The recommendation is to follow this syntax for <NAME>:
  - `'<NextVersion>_beta_<BranchName or FeatureDescription>_<ReleaseNumber>'`
  - <NextVersion> = the version number following the current release
  - <BranchName or Feature Description> = identification string for this test release
  - <ReleaseNumber> = start with "01". Then increment if more releases are created 02, 03, etc…
- Example:
  - `Management.make_release('com_ieee8023_.m','4p9p1_beta_Feature_ABC_01');`
  - This will create "com_ieee8023_4p9p1_beta_Feature_ABC_01.m"

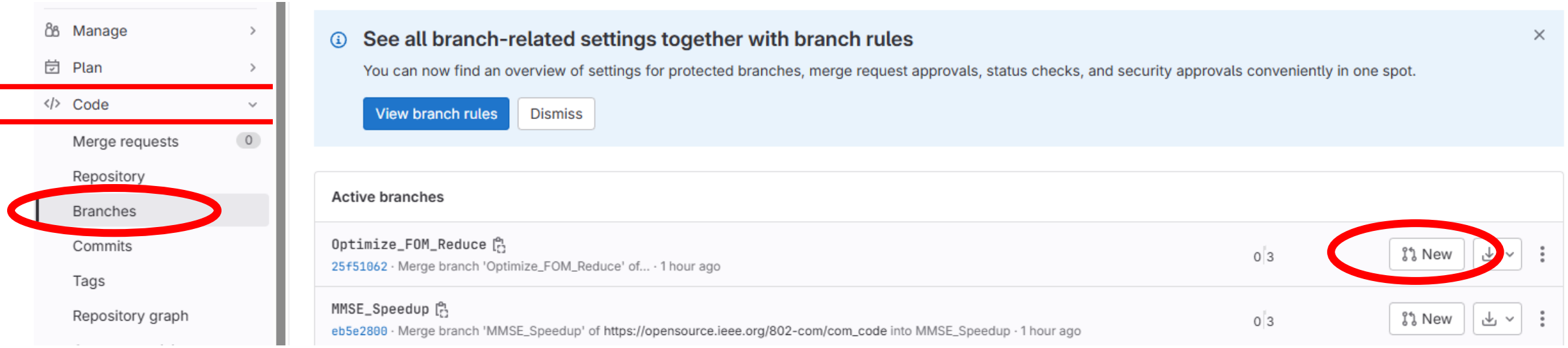
Step 5: Working on a branch

- After finishing the work on the branch, the recommendation is to write some tests to validate the completed work. This will make the process of merging into the main branch much faster.
- For example, if the update is to fix a bug that causes COM to crash in some specific scenario, the tests generally demonstrate 2 things:
    1. The scenario that crashes will now run successfully after the update
    2. Other scenarios that did not crash continue to produce the same output as they did before the update
- The ability to switch back and forth between the Feature Branch and the Main Branch simplifies the testing. Results can be stored by running the code on each branch and doing comparisons.

- Note: the Maintainers of 802-COM have a responsibility to provide a general test suite that contributors can run to help validate updates. This has not been completed yet.

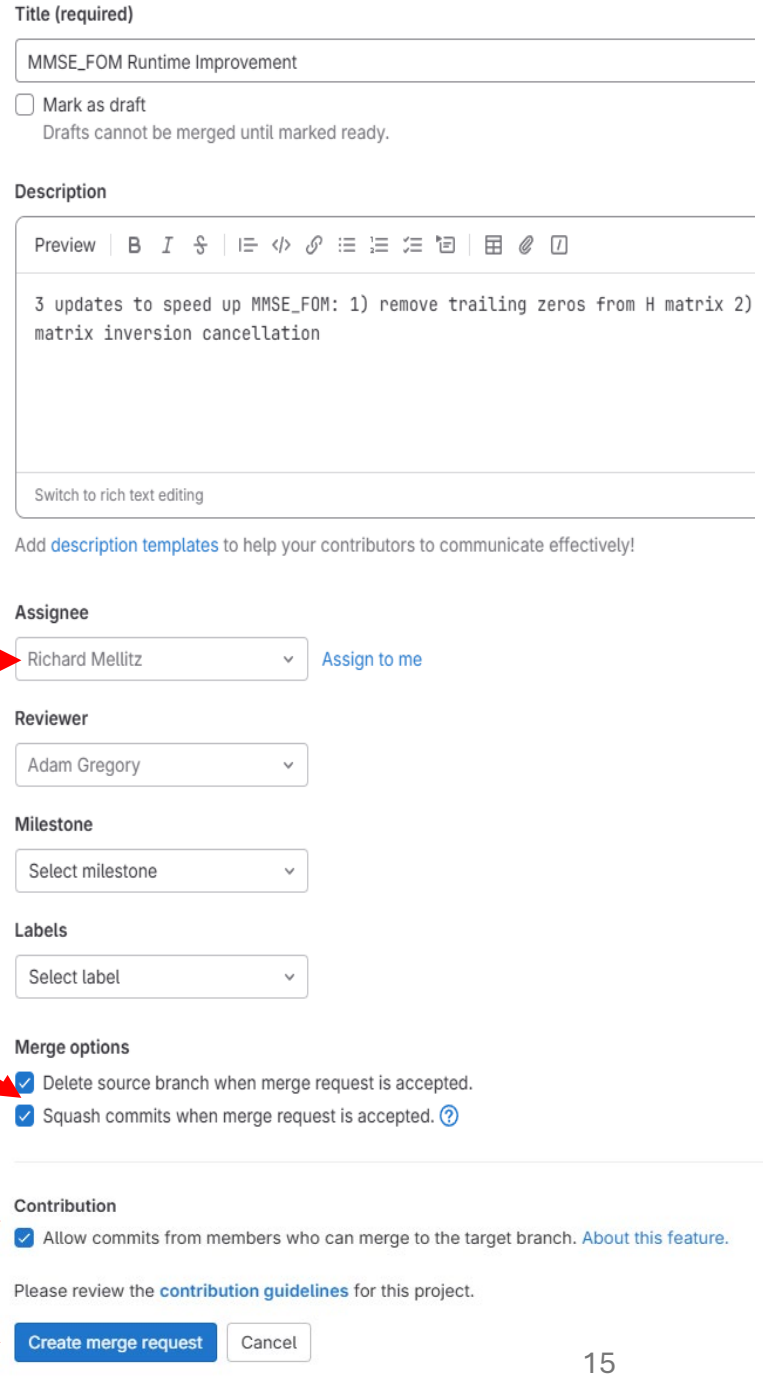
Step 6: Merge Requests

- When a branch is finished, it must be merged into the main branch
- To create a Merge Request, open the project in your personal workspace:
  - In the left panel, expand the Code option and select Branches
  - Click the New button on the target branch. This will open a new dialog for setting up the specific Merge Request
  - See next slide



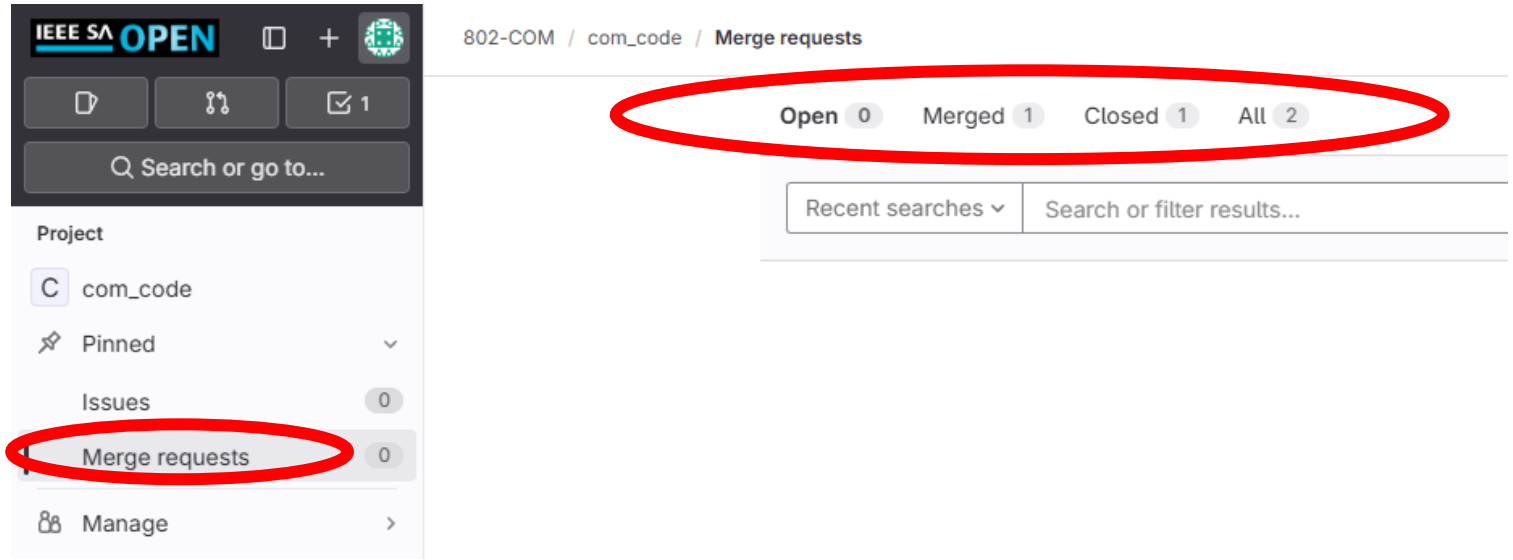
Step 6: Merge Requests

- In the Merge Request dialog, fill out the following fields.
  - Title
  - Description
  - Assignee
  - Reviewer
- The Assignee must be one of the maintainers of 802-COM.
- The options to “Delete source branch” and “Squash commits” should both be enabled
- Enable the option to “Allow commits from members who can merge to the target branch”
  - This increases the likelihood of the Merge Request being accepted since Maintainers of 802-COM can add additional commits before the merge.
- Click the Create Merge Request button



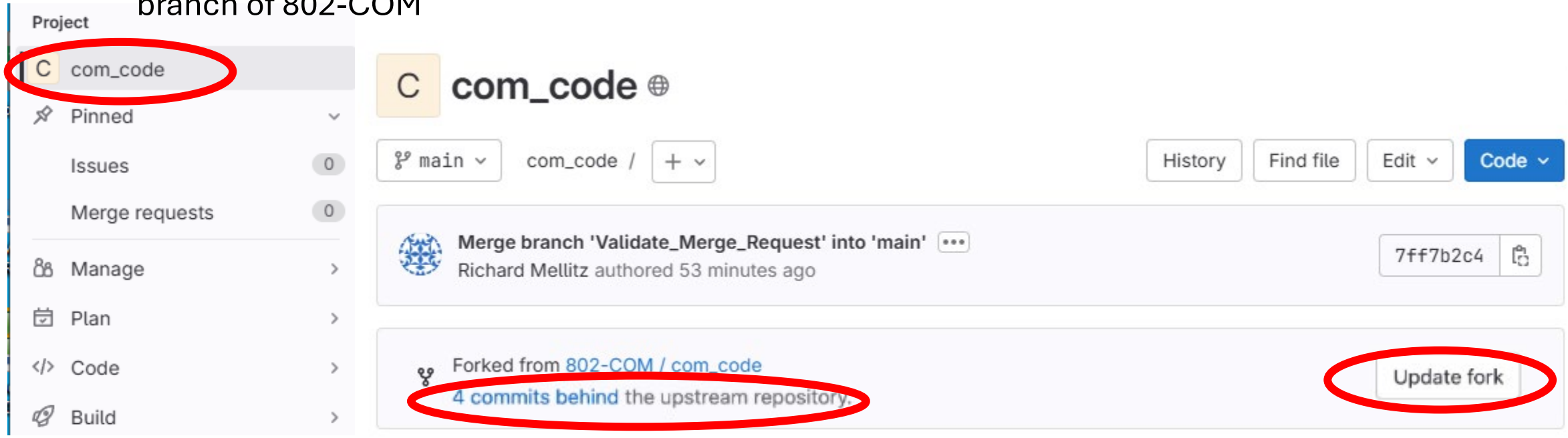
Viewing Merge Requests

- On the main 802-COM page, select Merge Requests
- The different categories of Merge Requests can be selected on the top panel: Open, Merged, Closed, All
- If you have recently submitted a Merge Request, it will be listed under Open



Step 7: Update Fork Repository after Merge Request

- When the Merge Request is approved and the code is merged, the Fork repository in your personal space will be behind the main 802-COM
  - The main branch in your fork reflects the state of the main branch before Merging
- Go to the main page of the Fork Repository in your personal space
- The section below that says where it was forked from will have an Update button if your Fork repository is behind. Click the Update button
- After clicking update, it will say “Up to date with the upstream repository”
- This step is important because it allows the next branch you create to be synced with the main branch of 802-COM



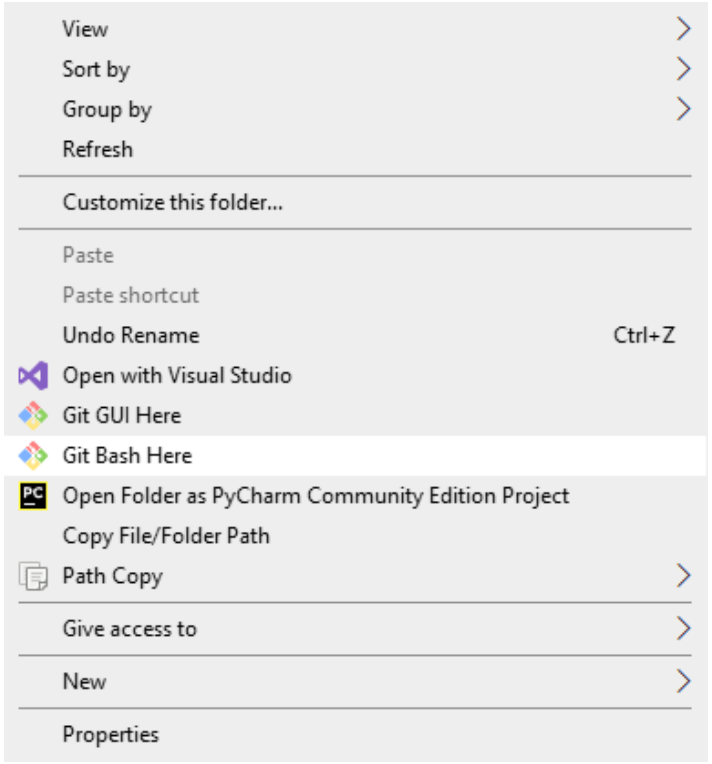
Git Tutorial

- There are many ways to use Git. The following slides contain some basic instructions for updating a Git repository, but it is not the only workflow that exists.

Git Tutorial (Opening Git Bash in the repository)

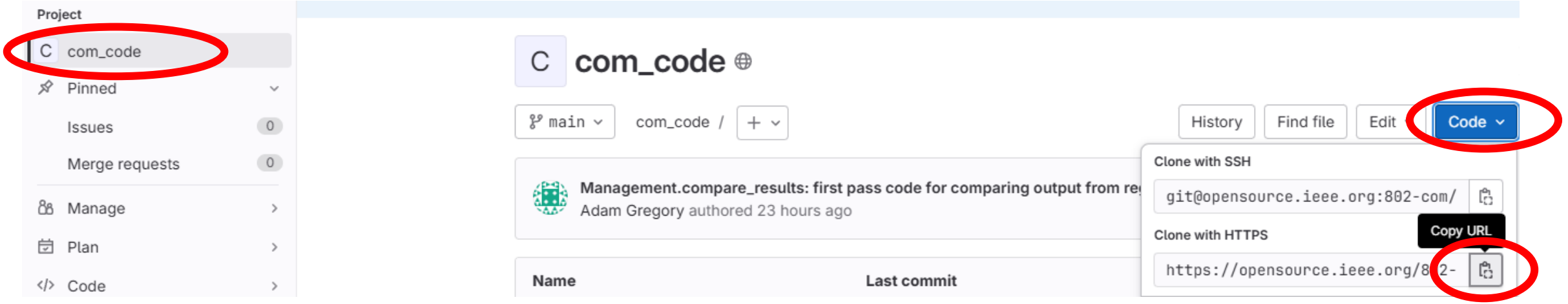
- Open Git Bash and navigate to the chosen folder. There are multiple ways to do this:
  - Option 1: Right click in Windows Explorer from the target directory and choose “Git Bash Here”
  - Option 2: Open Git Bash. Type “cd <target folder>”

- Git Bash has the most flexibility since it is command line driven. You can also open Git GUI. The visual interface makes a lot of operations easier.



Git Tutorial (Clone)

- Cloning is the process of copying a remote repository (like on the IEEE Open-Source site) to your local directory. Cloning is a one-time process. Once the repository is cloned to a particular place, you will never run clone in that space again. The git pull command is used to get future updates.
- The main page for a repository always has a Code button. Pressing that gives the URL needed to clone that repository.
- Open git bash in the location where you want the cloned repository to appear. Run:
  - git clone <URL>
- For the main 802-COM repository:
  - git clone https://opensource.ieee.org/802-com/com_code.git



Git Tutorial (pull)

- To get changes made on the remote repository that the local repository doesn't have, run this command:
  - git pull

- If you are referencing a repository on your personal fork, you will probably only need to execute a git pull after making a new branch. Unless there is another using making updates on your personal fork.
- Pull will be used mostly for:
  - Getting updates from the main 802-COM repository
  - Getting updates from another user's COM repository that you want to view

Git Tutorial (Checkout Feature Branch)

- If this is the first time using Git in the repository, it will start on the main branch. The guidelines set forth earlier in this document require that work is done in a different branch. Assuming that the branch has already been created on the remote repository, use this command to switch your working branch from main to the target branch:
  - git pull
  - git checkout <branch name>

- Note that the branch should be checked out BEFORE you start updating files in the repository. The action of checking out a branch will change the files on disk to match the state of the branch

```
adamg@SCDC-ADAMGR MINGW64 /c/matlab_tools/OSCOM/com_code (main)
$ git checkout Optimize_FOM_Reduce
Switched to branch 'Optimize_FOM_Reduce'
Your branch is up to date with 'origin/Optimize_FOM_Reduce'.
```

Git Tutorial (Updating Files)

- The main contribution effort is updating a file and pushing those changes to the remote repository. There are a sequence of 3 commands to complete the update:

1. Add
   - Provide Git with a list of modified files that should be staged. Staged files are those files that will eventually be committed. A changed file that has not been staged by running the Add command is not part of the commit.
2. Commit
   - Update the local repository (on your machine) with the files that have been staged by running Add
3. Push
   - Take all commits since the previous Push, and then Push the data to the remote repository (on IEEE Open-Source Site). Note that a Push can follow each commit, but a Push can also wait until some number of commits are finished.

- Note the important distinction between commit and push. Commit updates only the local repository. Push publishes to the remote repository (on IEEE Open-Source)

Git Tutorial (add)

- Run this command to see a list of files that can be added. This includes files that have been changed and files that are Untracked (not yet part of the repository):
  - git status
- Run git add to add a particular file:
  - git add <path to file>

```
adamg@SCDC-ADAMGR MINGW64 /c/matlab_tools/OSCOM/com_code (main)
$ git add src/test_file.m
```

- For situations where every available file that has been updated or is untracked will be added, this shortcut command can be used to add them all:
  - git add --all

```
adamg@SCDC-ADAMGR MINGW64 /c/matlab_tools/OSCOM/com_code (main)
$ git add --all
```

- Run git status again to see that the files which were added have changed from red to green. The green files are ready to commit.

Git Tutorial (commit)

- After adding files, they can be committed. Run this command:
  - git commit -m “My Commit Message”
- The commit message is what will show up on the repository website as a description for this commit, so it is to your advantage to be descriptive. It will also help other users viewing the update get a sense of what was accomplished before viewing the specific details of the commit.
- When possible, doing multiple small commits makes the history easier to parse.

```
adamg@SCDC-ADAMGR MINGW64 /c/matlab_tools/OSCOM/com_code (main)
$ git commit -m "Updating file my_function.m to support that cool feature"
```

- Note that after completing the commit, only your local repository has been updated.

Git Tutorial (push)

- After completing one or more commits, the push command can be run to push the repository updates to the remote repository on your personal IEEE space.
- Run this command:
  - git push
- A message (like shown below) will be generated showing if the push was successful. A failure to push likely means there was a conflict that must be resolved first. (Generally by doing a git pull and a merge). But when working on a Feature Branch in your personal space, there should never be a conflict unless other users are also contributing to the same feature.

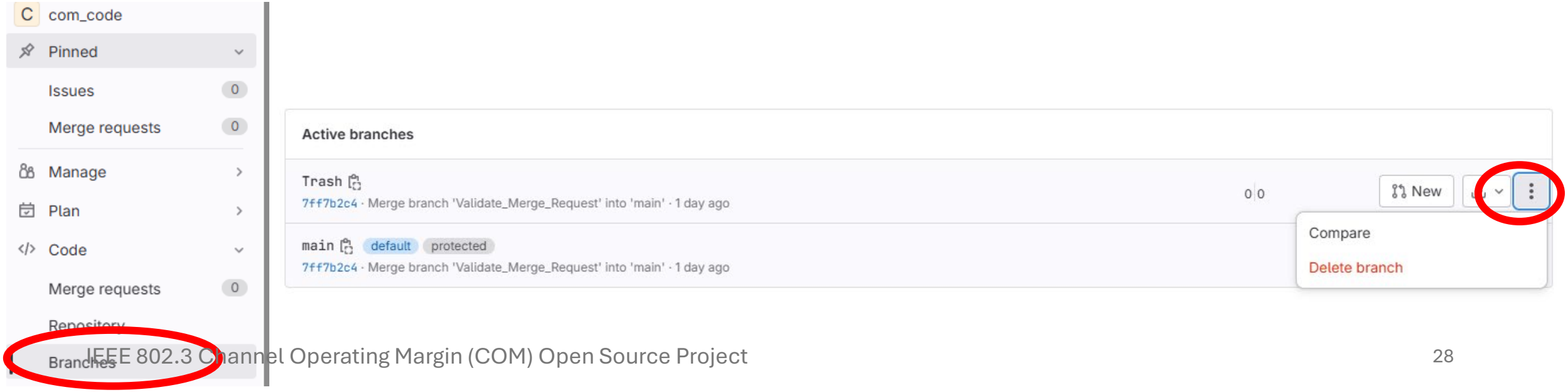
```
$ git push
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 12 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 504 bytes | 504.00 KiB/s, done.
Total 5 (delta 4), reused 0 (delta 0), pack-reused 0
remote:
remote: ========================================================================
remote:
remote:   The reply-by-email functionality is currently not working. To reply
remote:     to Issues or Merge requests, please do so via the web and not by
remote:                                    email.
remote:
remote: ========================================================================
```

Git Tutorial (Merge an update from Main Branch into Feature Branch)

- While working on a feature branch, it is possible there will be changes in the main branch that you want to incorporate
- First use the IEEE Open-Source website to update your Fork
  - See Step 7 in this presentation
- After updating, the main branch in your Fork repository will have the most recent updates in the main branch of 802-COM
- Checkout the branch you want to update with changes in main, and run git merge
  - git checkout <Feature_Branch>
  - git merge main
- The merge command means to take the updates in main and merge them into your Feature Branch. If there are no conflicts, the merge will finish automatically. If there are conflicts, you will need to choose which code to keep in the conflicting lines.
- Before running merge, it is usually a good idea to see the details of the changes you will be merging. You may decide to wait if the changes in the main branch conflict with lines of code you have updated.

Git Tutorial (Deleting a Branch)

- If you decide that a branch in your personal space should be removed, you can delete it from the Branches page. Choose the triple dot option beside the branch you want to remove and select Delete Branch
- This removes the branch from the remote repository, but it should also be cleared from the local repository. Run the following 2 commands
  - git fetch --all --prune
  - git branch -D <BRANCH NAME>



Git Stash: Changing branches before it is ready to commit

- It is possible to be in the middle of an update in a branch and need to switch back to main before work is finished. You could commit the unfinished work, but that could put the branch into an undesirable state
- The alternative is to stash the uncommitted data so that it can be retrieved later. Run this command:
  - git stash
- Now the files are changed back to their original state, but the change data is stored in the git stash
- After changes are made to the main branch and you are ready to work again on the stashed data, checkout the Feature Branch again and run this command:
  - git stash pop

Git Stash: Using multiple stashes

- Each git stash command saves a new stash. By default, running git stash pop reapplies all changes in the stash. It is possible to only apply a particular stash though
- First use this command to see all stashes:
  - git stash list
- It will show something like this:

```
$ git stash list
stash@{0}: On Branch_Revert: mod d.txt
stash@{1}: WIP on master: 8186206 revert last 2 commits
```

- Then use git stash pop with a pointer to the index of the stash you want to apply. For example, if you want to apply stash #1:
  - git stash pop stash@{1}
- The default names for stashes are not descriptive. If using multiple stashes, you can use this command to give the stash a name which makes it easy to identify when running the git stash list command:
  - git stash push –m “my_stash_name”
  - This stash would then be called “my_stash_name”

Accidentally committing and pushing to Main Branch instead of Feature Branch

- This happened early on, and I needed to figure out how to revert to the previous state
  - Note that this is only possible for users that have permission to push to the main branch
- Run these commands to revert the last N commits:
  - git revert --no-commit HEAD~N..
  - Here N = the number of commits to remove. If it was 2 commits:
    - git revert --no-commit HEAD~2..
  - Follow that with a normal commit supplying a message that summarize the full revert. Something like:
    - git commit -m “Reverting last 2 commits”

- You can also 1-by-1 revert each commit in reverse order:
  - git revert <commit hash>
- This is easier if you only have 1 commit to undo, but the first method is better when there are multiple commits.
- If there were branches based on commits that have been reverted, follow these commands after reverting on main:
  - git checkout <branch name>
  - git merge main
  - git push

Other questions

- How can others get the code that was edited?
  - Anyone can clone the forked repositories that other users have created if they are made Public
- Multiple folks editing the same file?
  - I will need to make another document covering merge conflicts.
  - If multiple users have changed the same file, git is able to merge if the same line of code was not touched in the 2 changes.
  - If the same line of code was updated, there is a process for managing merge conflicts. It basically requires a manual edit to decide what is kept in the conflict block.