CI/CD using MCUXpresso for VS Code, MCUXpresso SDK, and GitHub

1. Objective

This guide is the shortened version of the lab. A CICD pipeline has already been established to speed up the process. This pipeline makes use of a Docker image hosted on GitHub Container Registry. The goal is to understand how a team can work independently on a joint project. In this lab, you will fork the existing repo. The reason for this is because this repo is intended to be public, hence managing write access for an undetermined list of users is not ideal as it would be for internal repos. Once you have forked the repo, you will enable the associated workflow in your repo. Last, you will create a feature branch to actively modify the project and create a pull request to trigger an automatic build.

2. Prerequisites

- GitHub account
- MCUXpresso for VS Code
- Docker
- Basic knowledge of Git and CMake

3. Lab Structure

- Forking the project repo
- Modifying the project files
- Creating a pull request
- Examining the workflow
- Post processes --> Project Admin

Forking the project repository

Forking a GitHub repository means creating a personal copy of someone else's repository under your own GitHub account. This allows you to freely experiment with changes without affecting the original project. It's commonly used when:

- You want to contribute to an open-source project.
- You need to customize a repo for your own use.
- You want to explore or test changes safely.

1. Navigate to GitHub and sign in to your account. The sample project repo can be found at https://github.com/nxp-jose/mcuxpresso-cicd. Click the Fork button at the top-right of the repository page. This creates a copy of the repo in your account. The repo has the following structure.

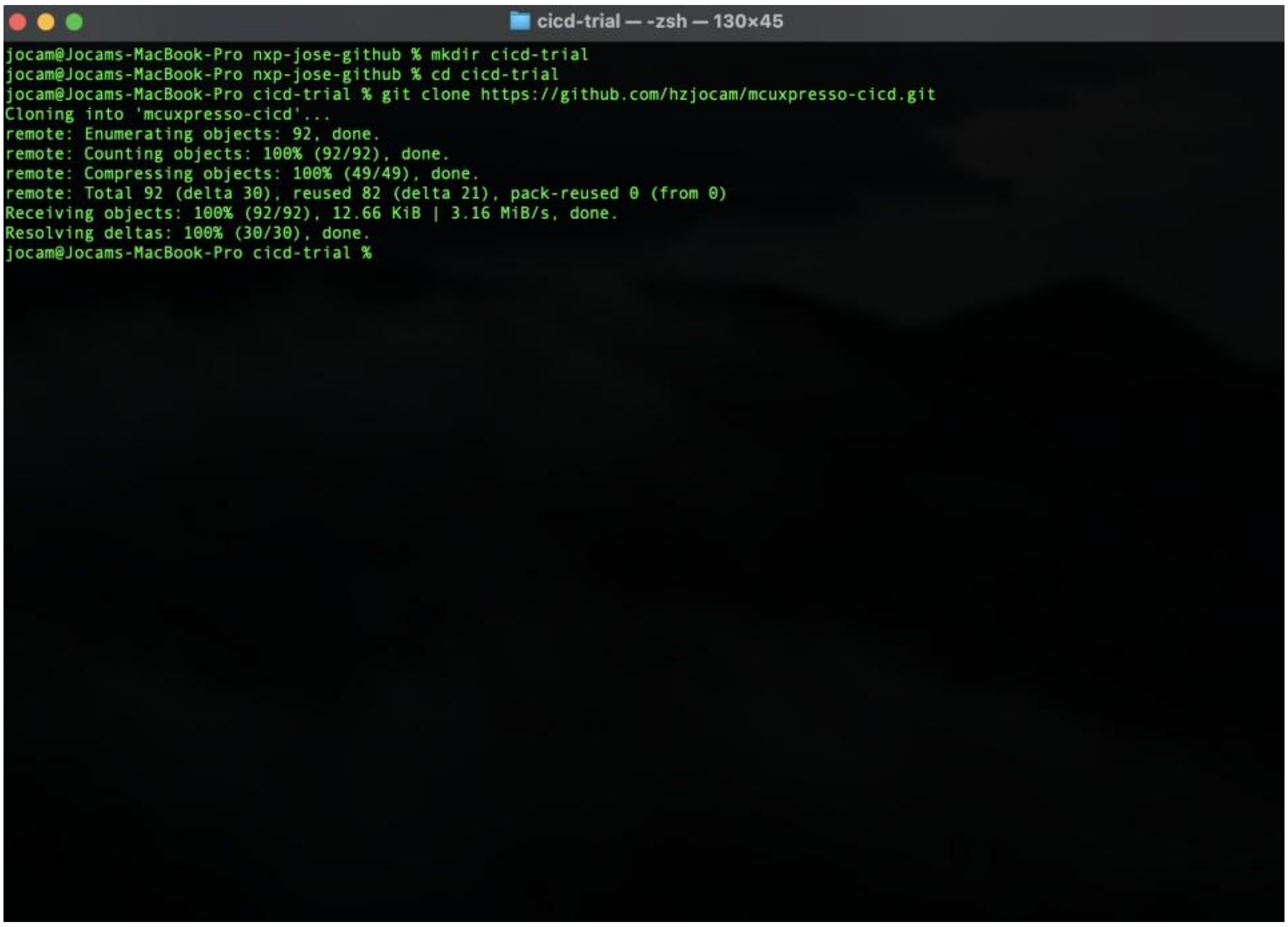
```
mcuxpresso-cicd repository

├── .github/
│   └── workflows/
│         └── docker-build.yml
├── dockerfile
│
├── my_app/
│   └── CMakeLists.txt
│   └── CMakePresets.json
│   └── Kconfig
│   └── example.yml
│   └── hello_world.c
│   └── mcux_include.json
│   └── prj.conf
├── README.md
```

> **Note:** This guide was made with the Hello World example from MCUXpresso SDK v25.06 initially imported as a freestanding example. The board files are referenced from the SDK which is containerized along with the build system. Board files are expected in freestanding examples in versions 25.09 of the MCUXpresso SDK and MCUXpresso for VS Code.

2. To make changes locally on your machine, clone your fork.

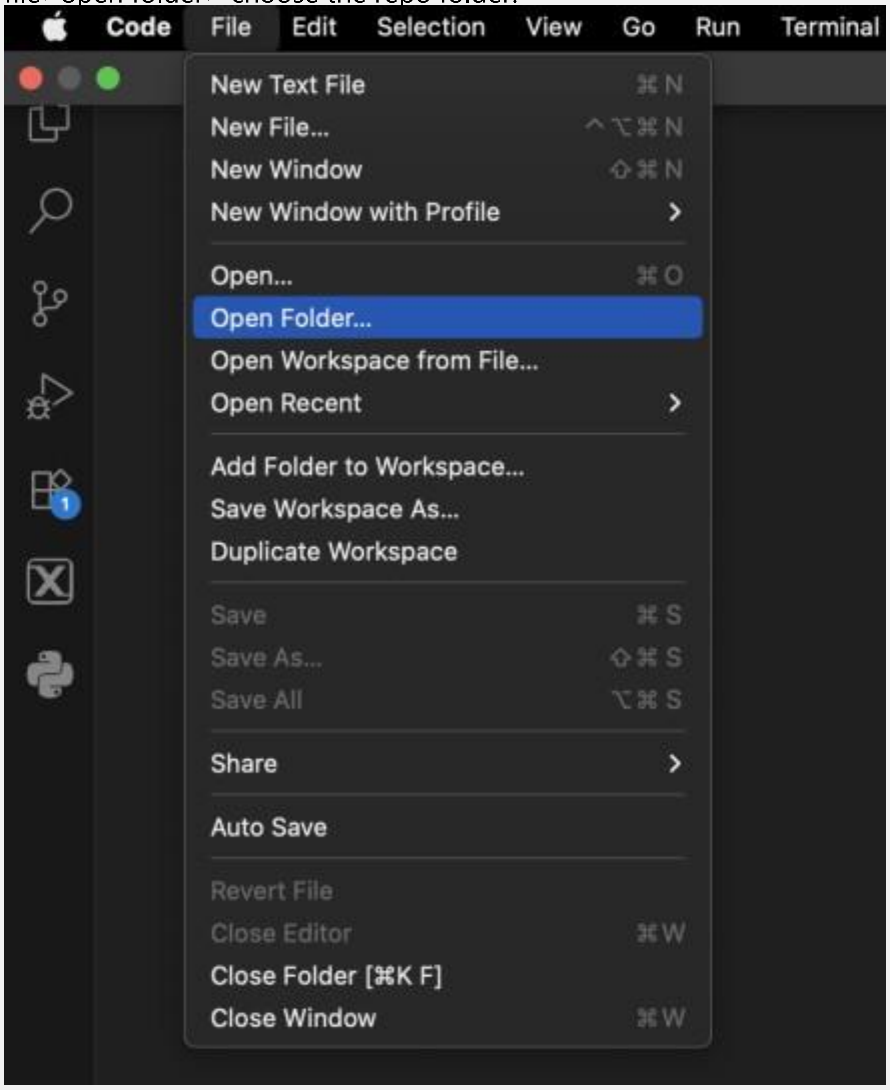
> **Note 2.1:** MCUXpresso for VS Code allows you to use the command line directly from your workspace.

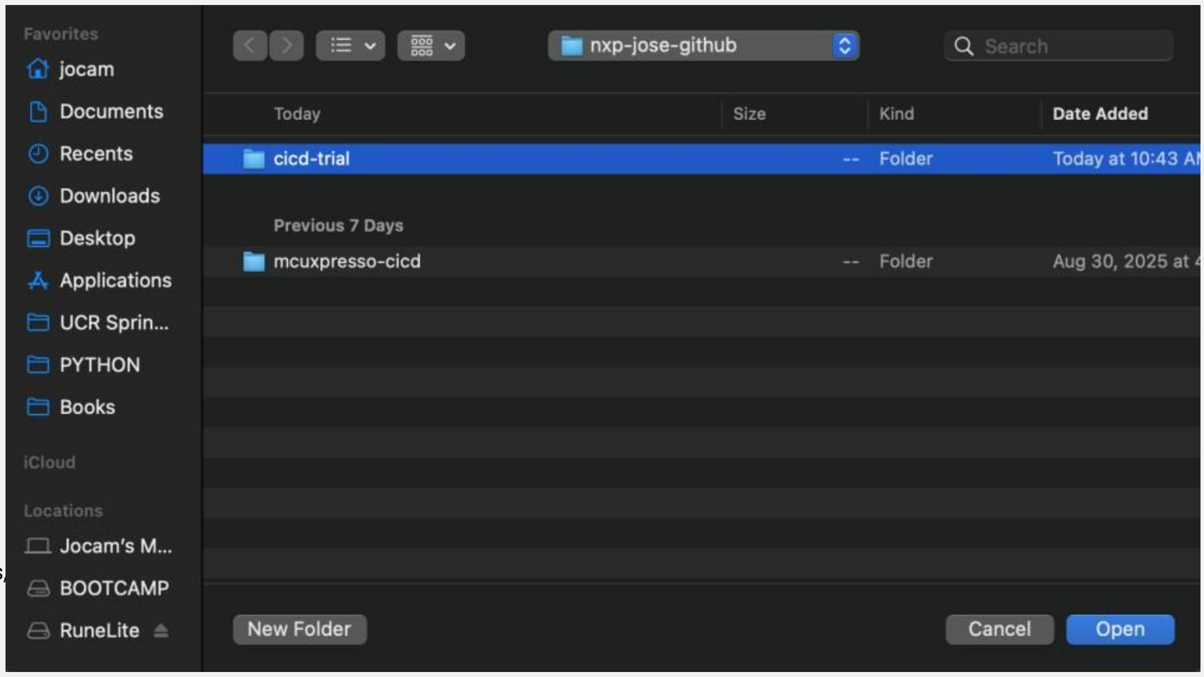
```
git clone https://github.com/your-username/mcuxpresso-cicd.git
```



```
jocam@Jocams-MacBook-Pro nxp-jose-github % mkdir cicd-trial
jocam@Jocams-MacBook-Pro nxp-jose-github % cd cicd-trial
jocam@Jocams-MacBook-Pro cicd-trial % git clone https://github.com/hzjocam/mcuxpresso-cicd.git
Cloning into 'mcuxpresso-cicd'...
remote: Enumerating objects: 92, done.
remote: Counting objects: 100% (92/92), done.
remote: Compressing objects: 100% (49/49), done.
remote: Total 92 (delta 30), reused 82 (delta 21), pack-reused 0 (from 0)
Receiving objects: 100% (92/92), 12.66 KiB | 3.16 MiB/s, done.
Resolving deltas: 100% (30/30), done.
jocam@Jocams-MacBook-Pro cicd-trial %
```

Note 2.2: You can continue working on the command line, however, VS Code provides a terminal which you can use while you edit files. This will be specifically useful when using MCUXpresso for VS Code to develop your projects. Open the repo folder within VS Code, go to file>open folder> choose the repo folder.





Today	Size	Kind	Date Added
cicd-trial	--	Folder	Today at 10:43 A
Previous 7 Days			
mcuxpresso-cicd	--	Folder	Aug 30, 2025 at

> **Note 2.3:** When you launch VS Code, it will prompt you to select a cmakelists.txt file. This happens because MCUXpresso projects rely on cmake. You do not have to explicitly select one, simply press the Esc key on your keyboard.
>
> **Note 2.4:** For the purpose of this lab, the example project in this repo is not directly imported into the MCUXpresso extension. Hence, Intellisense will highlight syntax and usage warnings which can be ignored.
>
> **Note 2.5:** To enable Intellisense and syntax highlighting completion and to take advantage of the debugging tools. You have to import a compatible SDK and import the project into the repo overwriting/replacing the existing files.

3. For now, open the VS Code Terminal to the repo's directory. Create a feature branch to make your changes in. This step is completely optional, however, it is good practice.

```
git checkout -b feature/my-new-feature
```

Modifying the project's files

In this guide, we will make a simple change to focus on the on CICD.

1. To see the pipeline in action, change the string in hello_world.c and save the file.

2. Bring up the Terminal in VS Code and make sure you are working in your feature branch.

```
git branch
```

   Once you confirm you are in your feature branch, check the status.

```
git status
```

3. Stage your changes.

```
git add my_app/hello_world.c
```

4. Commit your changes.

```
git commit -m "modified string in main."
```

5. Push your changes.

```
git push
```

> **Note:** A source control GUI is also implemented in VS Code. Steps 1-5 in this section can be completed using the source control GUI. The source control GUI provides a visual reprentation of the changes made and you'll have access to the source control graph.

6. Navigate to your repo on GitHub and verify that the changes were pushed. At this point the workflow should automatically trigger a build using GitHub Actions.

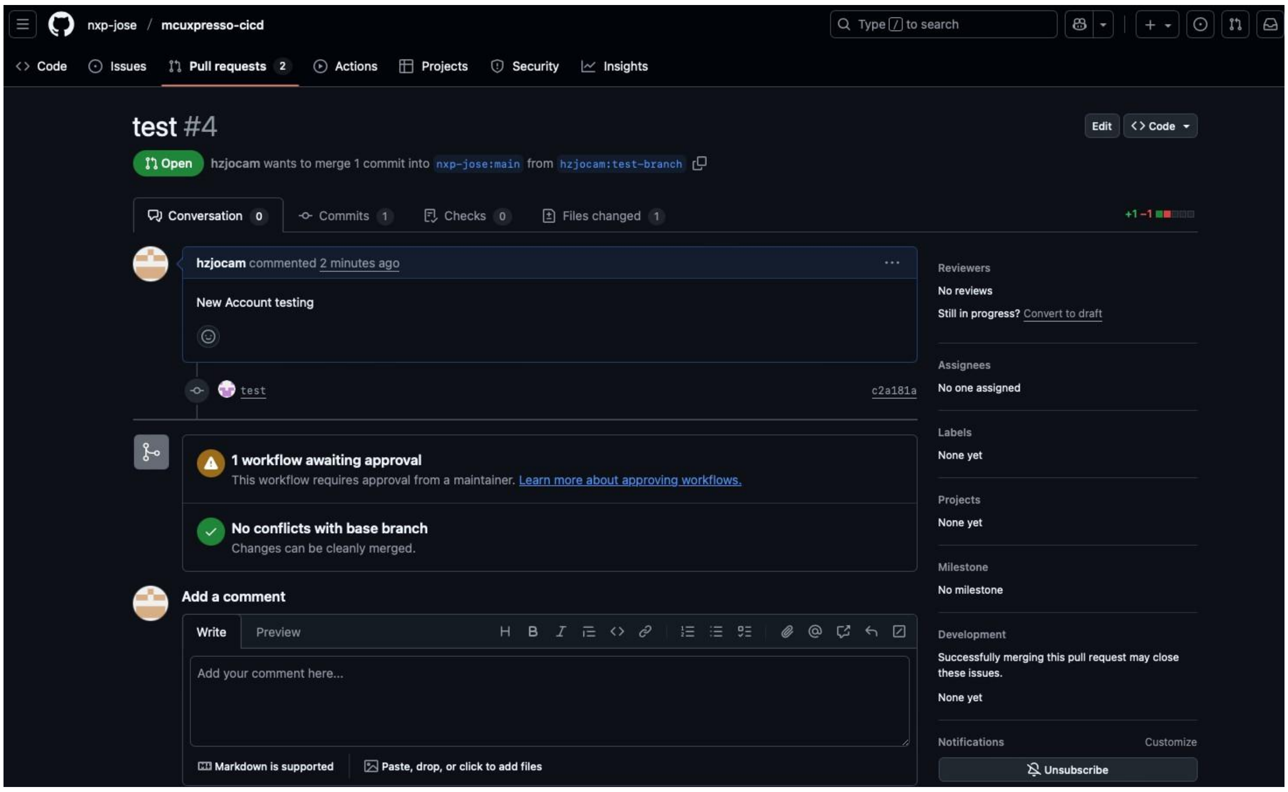
7. Troubeshooting permissions - See Appendix.

Creating a pull request

> **Note:** Your repo should recognize that there is a worfkflow file. Enable workflows when prompted to do so.

A big portion of CICD is automation and continuous testing. Hence, the main repo can be configured to build from Pull Requests. An admin can then choose to review the successful builds and ignore and send back the failed builds. The pull requests will track the build process. During this time you can actively review build messages. Anyone tracking the project will also be notified about the PR.

1. On GitHub, from your feature branch, click on Pull Requests.
2. Select New pull request.
3. This pipeline is setup so that the maintainer approves the workflow run.


nxp-jose / mcuxpresso-cicd

test #4

Open hzjocam wants to merge 1 commit into nxp-jose:main from hzjocam:test-branch

hzjocam commented 2 minutes ago

New Account testing

test c2a181a

1 workflow awaiting approval
This workflow requires approval from a maintainer. Learn more about approving workflows.

No conflicts with base branch
Changes can be cleanly merged.

.Once the maintainer approves, you'll have access to the build details on their repo.

Examining the workflow

The GitHub Actions workflow in this project has been configured to trigger a build in the container automatically.

1. Explore the Actions tab. This will show you the build steps as they were executed. This will also show you how were the project fails if it ever does.
2. Note that for the size of this image, the build typically takes 2 minutes.

Conclusion

This guide shows one of several ways that a CI/CD pipeline can be established. The procedure shown shows
GitHub to use GitHub Actions, Docker for containerization, and MCUXpresso for VS Code to work with
MCUXpresso SDK projects. Keep in mind that Docker image was built from the entire MCUXpresso SDK. However, a custom manifest will reduce the SDK size and improve build times. This guide as well as creating a custom manifest will soon be uploaded to the MCUXpresso Training Hub.

Appendix

1. Upstream repo issues:

   This step is only needed if you run into issues with your upstream. The first thing to try is to set up the upstream.

   ```
   git push --set-upstream origin test-branch
   ```

   If permission is still denied then set up ssh keys for your GitHub account.

   ```
   ssh-keygen -t ed25519 -C "your_email@something.com"
   ```

   Press enter and skip through the prompts. **DO NOT SET THESE AS SECURE KEYS THROUGH THE PROMPTS FOR THIS TUTORIAL, IT IS NOT NECESSARY.** Now copy the key.

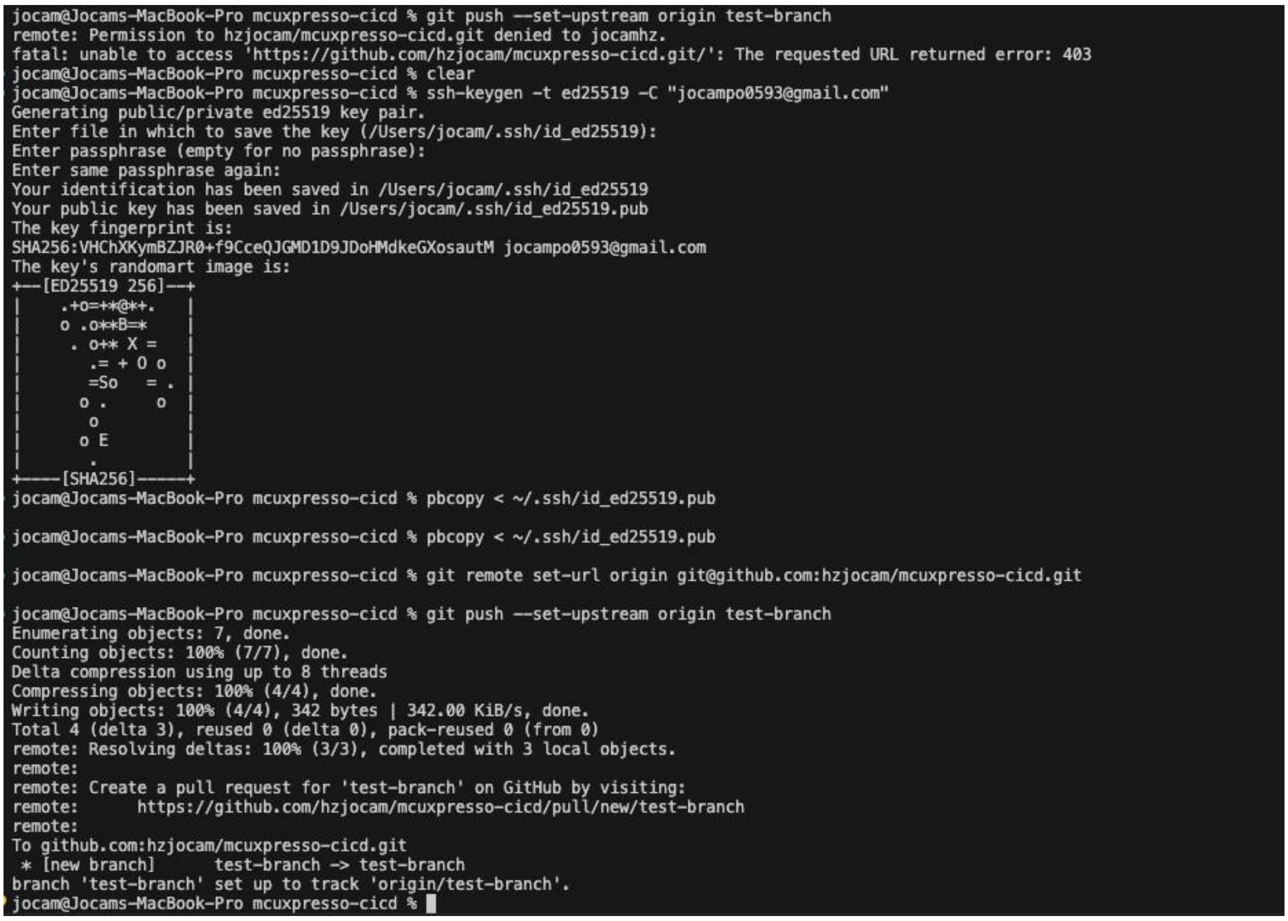
   ```
   pbcopy < ~/.ssh/id_ed25519.pub
   ```

   Go to GitHub, click settings, SSH and GPG keys, click new SSH key. Paste the copied key into the field and save. Now set the remote url on the CLI.

   ```
   git remote set-url origin git@github.com:your-repo
   ```

   REPLACE "your-repo" with your actual repo fork. Now attempt to push again.

   ```
   git push --set-upstream origin test-branch
   ```



```
jocam@Jocams-MacBook-Pro mcuxpresso-cicd % git push --set-upstream origin test-branch
remote: Permission to hzjocam/mcuxpresso-cicd.git denied to jocamhz.
fatal: unable to access 'https://github.com/hzjocam/mcuxpresso-cicd.git/': The requested URL returned error: 403
jocam@Jocams-MacBook-Pro mcuxpresso-cicd % clear
jocam@Jocams-MacBook-Pro mcuxpresso-cicd % ssh-keygen -t ed25519 -C "jocampo0593@gmail.com"
Generating public/private ed25519 key pair.
Enter file in which to save the key (/Users/jocam/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /Users/jocam/.ssh/id_ed25519
Your public key has been saved in /Users/jocam/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:VHChXKymBZJR0+f9CceQJGMD1D9JDoHMdkeGXosautM jocampo0593@gmail.com
The key's randomart image is:
+--[ED25519 256]--+
|    .+o=+*@*+.   |
|    o .o**B=*    |
|     . o+* X =   |
|       .= + O o  |
|       =So   = . |
|      o .     o  |
|       o         |
|      o E        |
|       .         |
+----[SHA256]-----+
jocam@Jocams-MacBook-Pro mcuxpresso-cicd % pbcopy < ~/.ssh/id_ed25519.pub

jocam@Jocams-MacBook-Pro mcuxpresso-cicd % pbcopy < ~/.ssh/id_ed25519.pub

jocam@Jocams-MacBook-Pro mcuxpresso-cicd % git remote set-url origin git@github.com:hzjocam/mcuxpresso-cicd.git

jocam@Jocams-MacBook-Pro mcuxpresso-cicd % git push --set-upstream origin test-branch
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 342 bytes | 342.00 KiB/s, done.
Total 4 (delta 3), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
remote:
remote: Create a pull request for 'test-branch' on GitHub by visiting:
remote:      https://github.com/hzjocam/mcuxpresso-cicd/pull/new/test-branch
remote:
To github.com:hzjocam/mcuxpresso-cicd.git
 * [new branch]      test-branch -> test-branch
branch 'test-branch' set up to track 'origin/test-branch'.
jocam@Jocams-MacBook-Pro mcuxpresso-cicd % 
```