|  |  |  |
| --- | --- | --- |
|  | | |
|  |  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  | | |

**Revision History**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**Table of Content**

[1 git-flow 1](#_Toc38899229)

[1.1 Getting Started 1](#_Toc38899230)

[1.2 How it works 1](#_Toc38899231)

[1.2.1 Develop and Master Branches 1](#_Toc38899232)

[1.2.2 Feature Branches 1](#_Toc38899233)

[1.2.3 Release Branches 2](#_Toc38899234)

[1.2.4 Hotfix Branches 2](#_Toc38899235)

[1.3 Summary 3](#_Toc38899236)

[2 IVC Git Guideline 4](#_Toc38899237)

[2.1 Non-review flow (no Merge request) 4](#_Toc38899238)

[2.1.1 Clone the created Git from the server 4](#_Toc38899239)

[2.1.2 Commit the base source code to git (master branch) 5](#_Toc38899240)

[2.1.3 Initialize the git-flow 6](#_Toc38899241)

[2.1.4 Create a new feature 8](#_Toc38899242)

[2.1.5 Commit code to new feature branches 9](#_Toc38899243)

[2.1.6 Finish feature (merge to develop branch) 10](#_Toc38899244)

[2.1.7 Create release branch 12](#_Toc38899245)

[2.1.8 Finish release branch 14](#_Toc38899246)

[2.1.9 Create hotfix (for bugs after release) 15](#_Toc38899247)

[2.2 Review flow (create Merge request) 19](#_Toc38899248)

[2.2.1 Create feature branch 19](#_Toc38899249)

[2.2.2 Create Merge request (for feature developer) 20](#_Toc38899250)

[2.2.3 Merge the Merge request (for Reviewer) 21](#_Toc38899251)

[2.3 Merging conflict problem 22](#_Toc38899252)

[2.3.1 When using Source tree merge 22](#_Toc38899253)

[2.3.2 When using Merge request 25](#_Toc38899254)

1. git-flow
   1. Getting Started

Git-flow is a Git workflow design that was first published and made popular by Vincent Driessen at nvie.

The Gitflow Workflow defines a strict branching model designed around the project release.

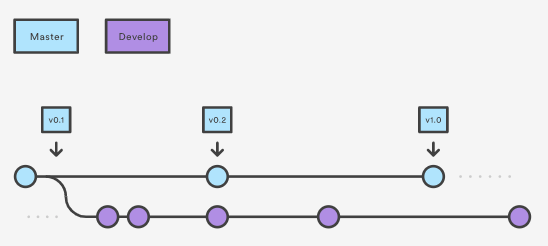
There are many Git tools support Git-flow, but in this Guideline, we will use Source Tree to use Git-flow

Source Tree installation: <https://www.sourcetreeapp.com/>

* 1. How it works
     1. Develop and Master Branches

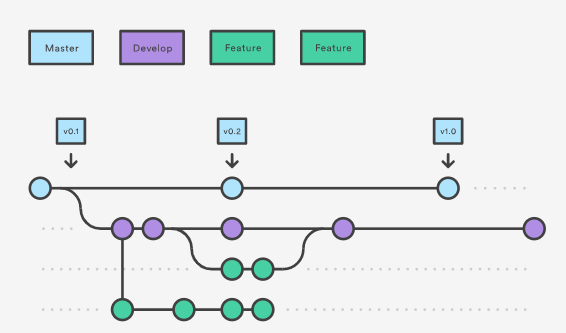
The master branch stores the official release history

The develop branch serves as an integration branch for features.



* + 1. Feature Branches

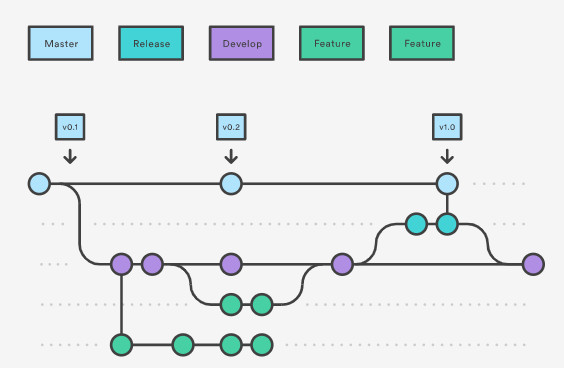
Each new feature should reside in its own branch, which can be pushed to the central repository for backup/collaboration. But, instead of branching off of master, feature branches use develop as their parent branch. When a feature is complete, it gets merged back into develop. Features should never interact directly with master.



*Feature branch example name: feature/login*

* + 1. Release Branches

Once develop has acquired enough features for a release (or a predetermined release date is approaching), you fork a release branch off of develop.

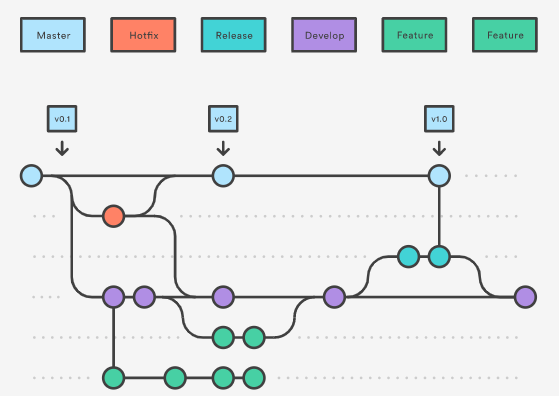


* + 1. Hotfix Branches

Maintenance or “hotfix” branches are used to quickly patch production releases.

This is the only branch that should fork directly off of master.

As soon as the fix is complete, it should be merged into both master and develop and master should be tagged with an updated version number.



* 1. Summary

The overall flow of Gitflow is:

1. A ***develop*** branch is created from ***master***
2. A ***release*** branch is created from ***develop***
3. ***Feature*** branches are created from ***develop***
4. When a feature is complete it is merged into the ***develop*** branch
5. When the ***release*** branch is done it is merged into ***develop*** and ***master***
6. If an issue in master is detected a ***hotfix*** branch is created from ***master***
7. Once the ***hotfix*** is complete it is merged to both ***develop*** and ***master***

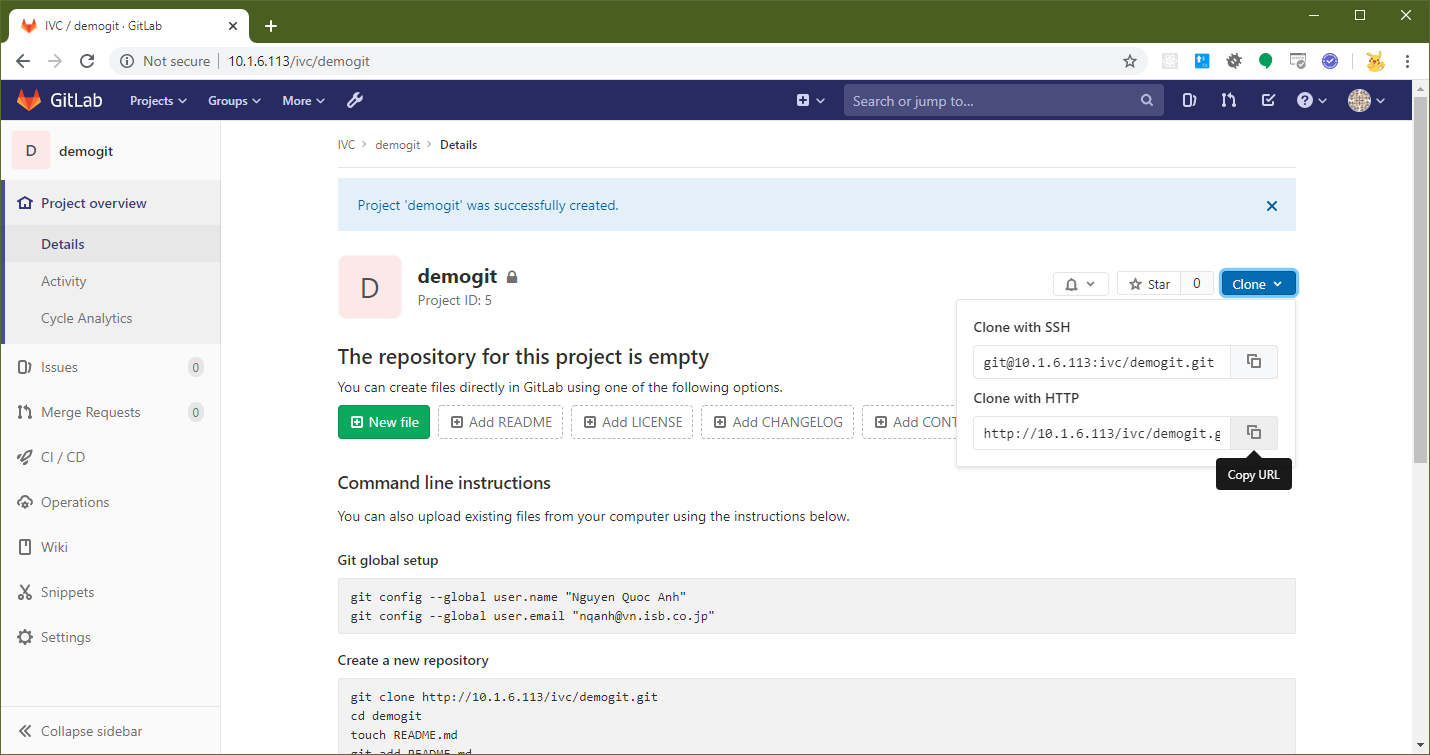
Reference: <https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>

1. IVC Git Guideline

* **Note: The URL of this Guideline may be different with actual URL**
  1. Non-review flow (no Merge request)
     1. Clone the created Git from the server

Copy the Git URL on the website

* 1. Click on Clone button
  2. Then Click “Copy URL”

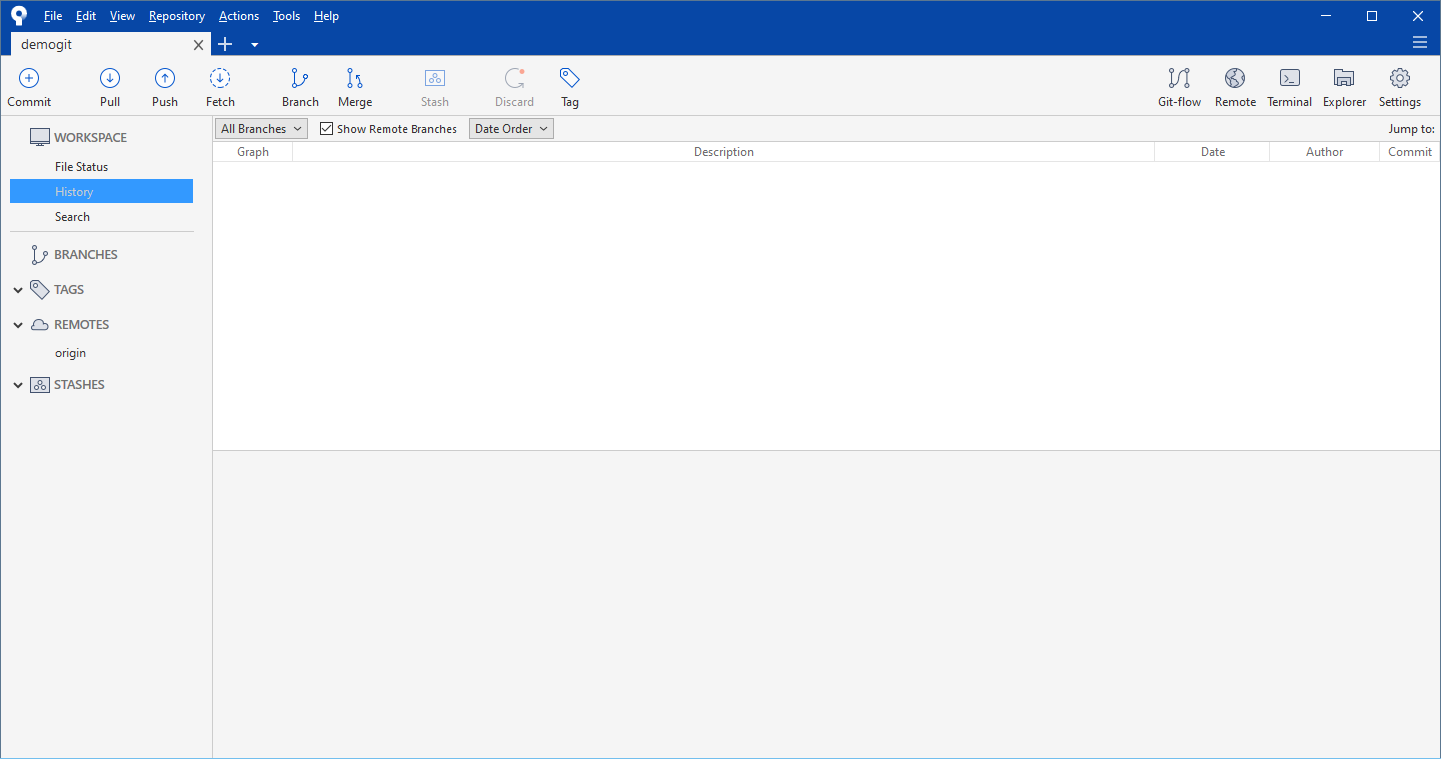


Clone using Source Tree

1. On Source Tree, select Menu File -> Clone/New
2. Input the required fields then click Clone
3. Enter the username/password if requested



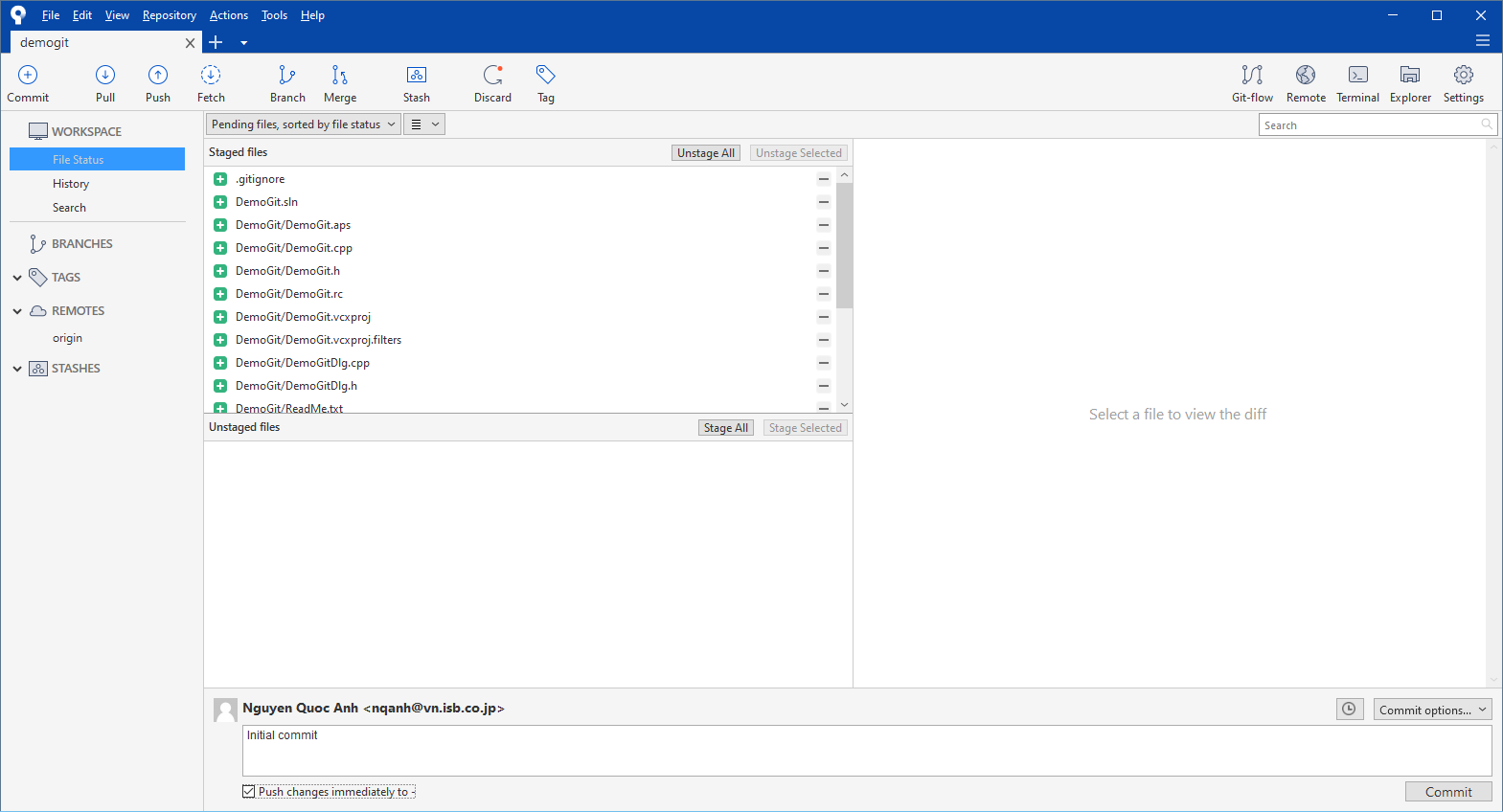
After cloned



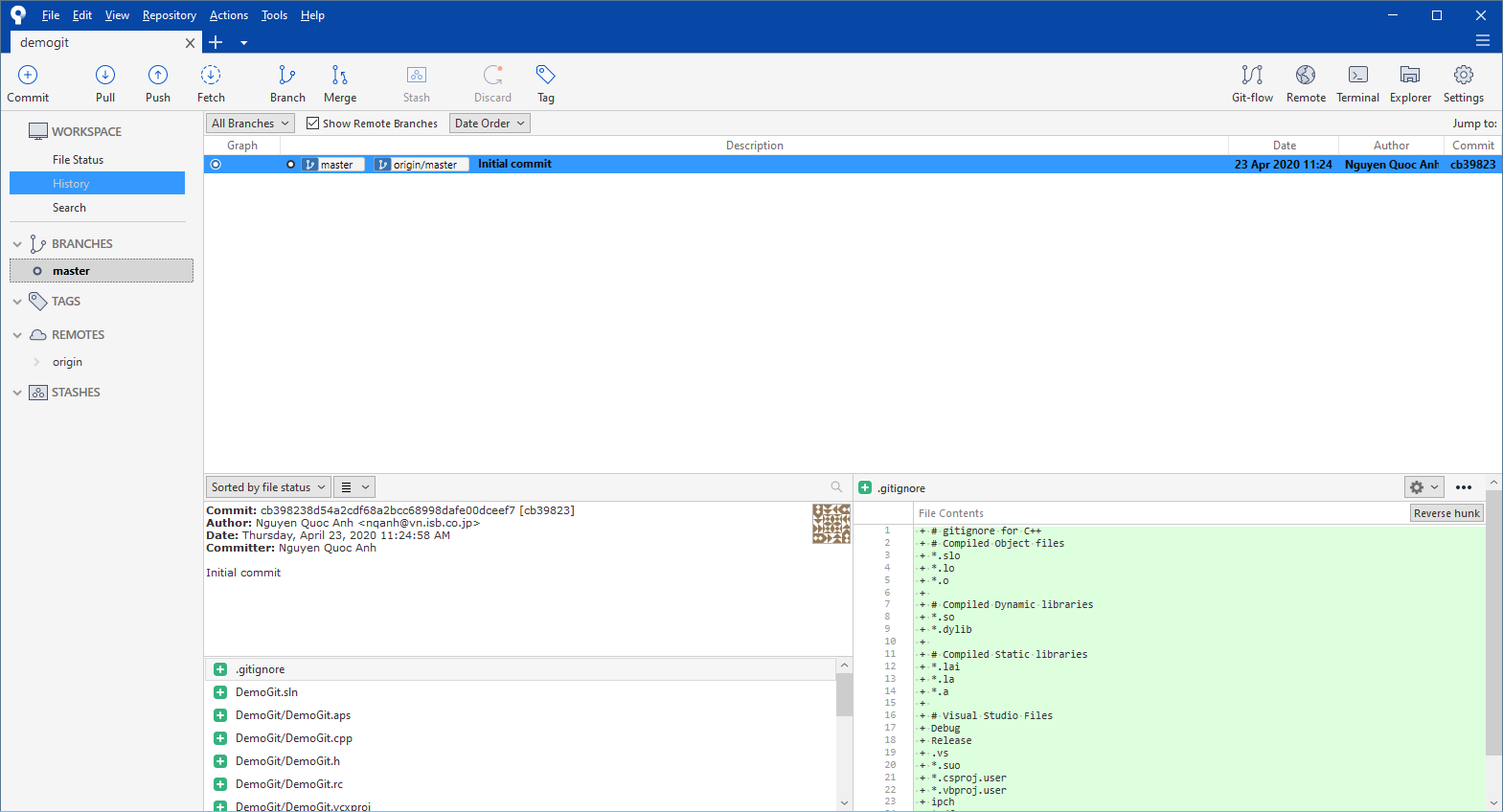
* + 1. Commit the base source code to git (master branch)

Copy the base source code to cloned folder

Commit and push the source code to master branch

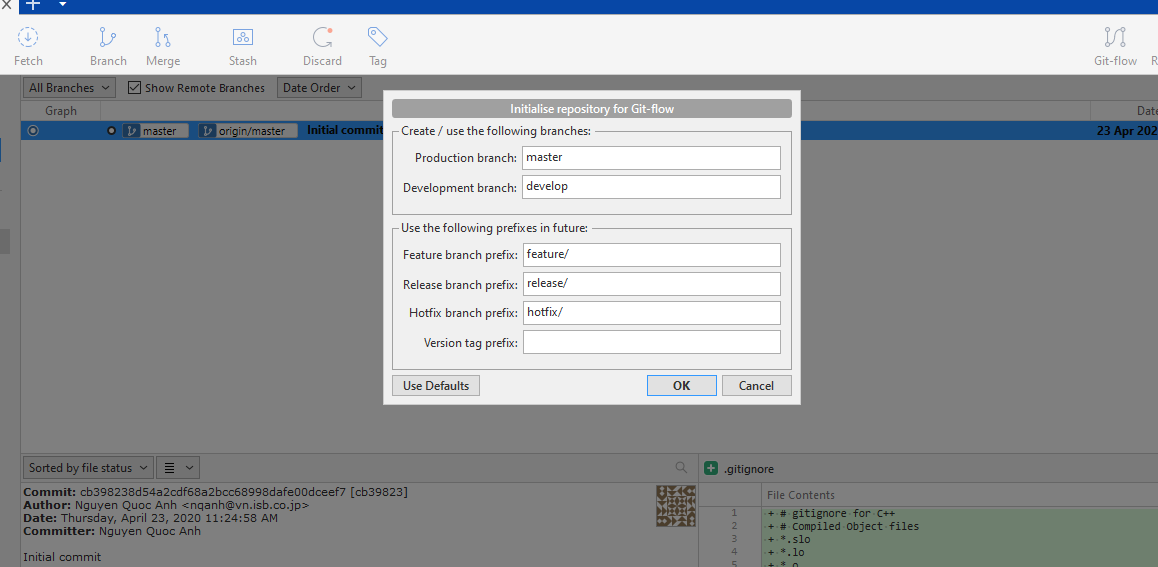


After commit

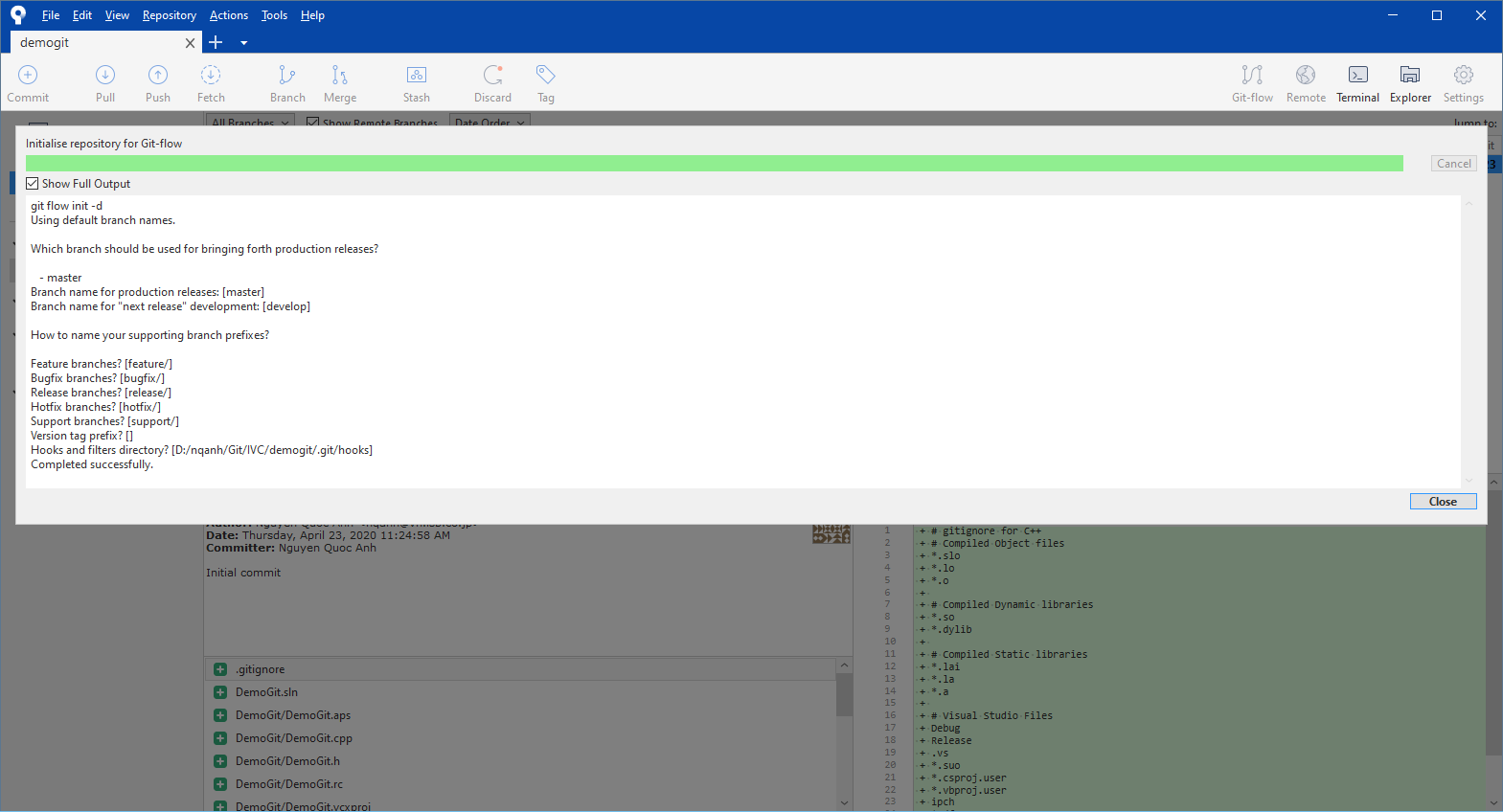


* + 1. Initialize the git-flow

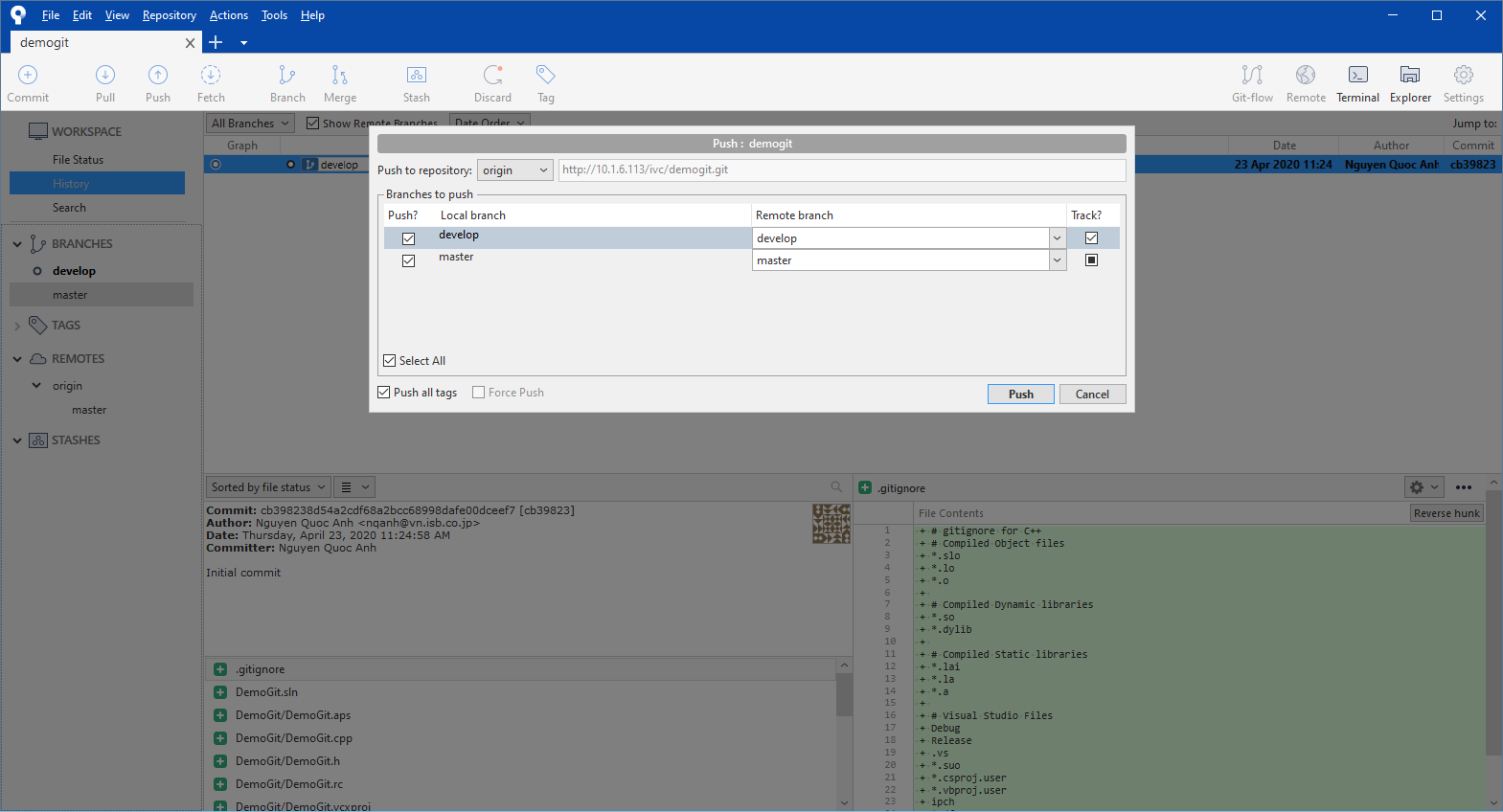
Click on the button Git-flow on the toolbar of Source Tree

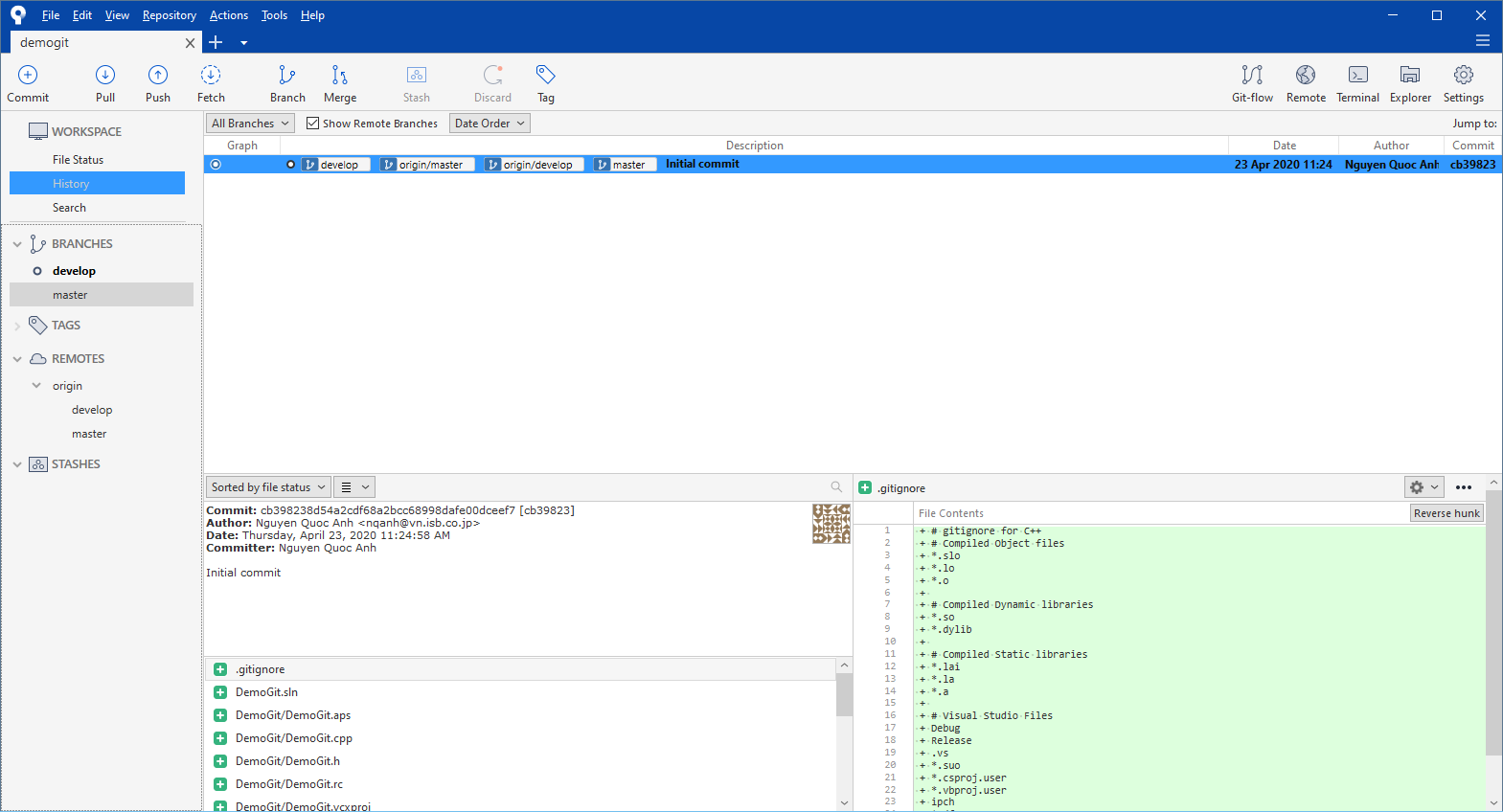


Leave everything default, click OK button



After that, push all branches

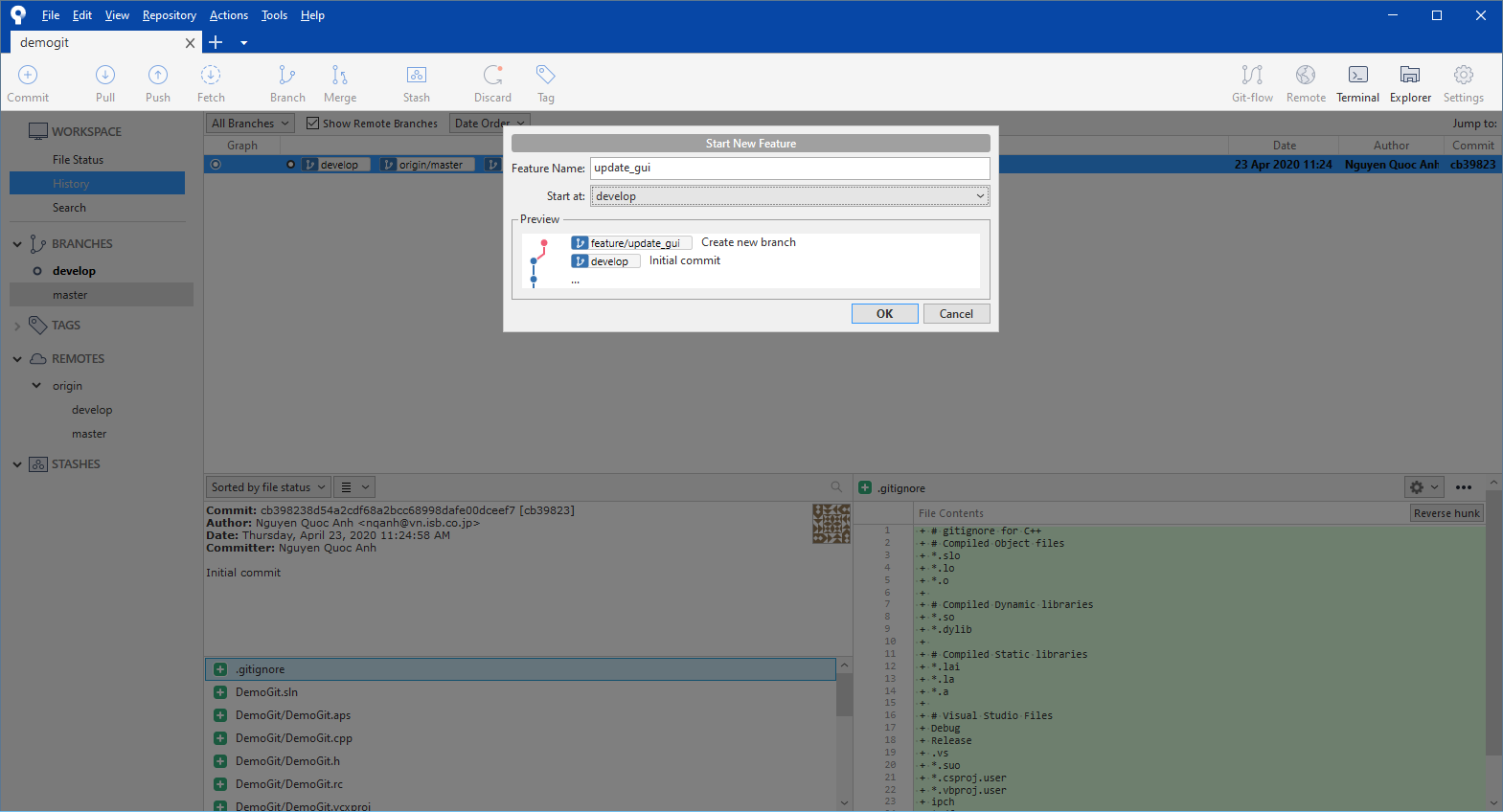


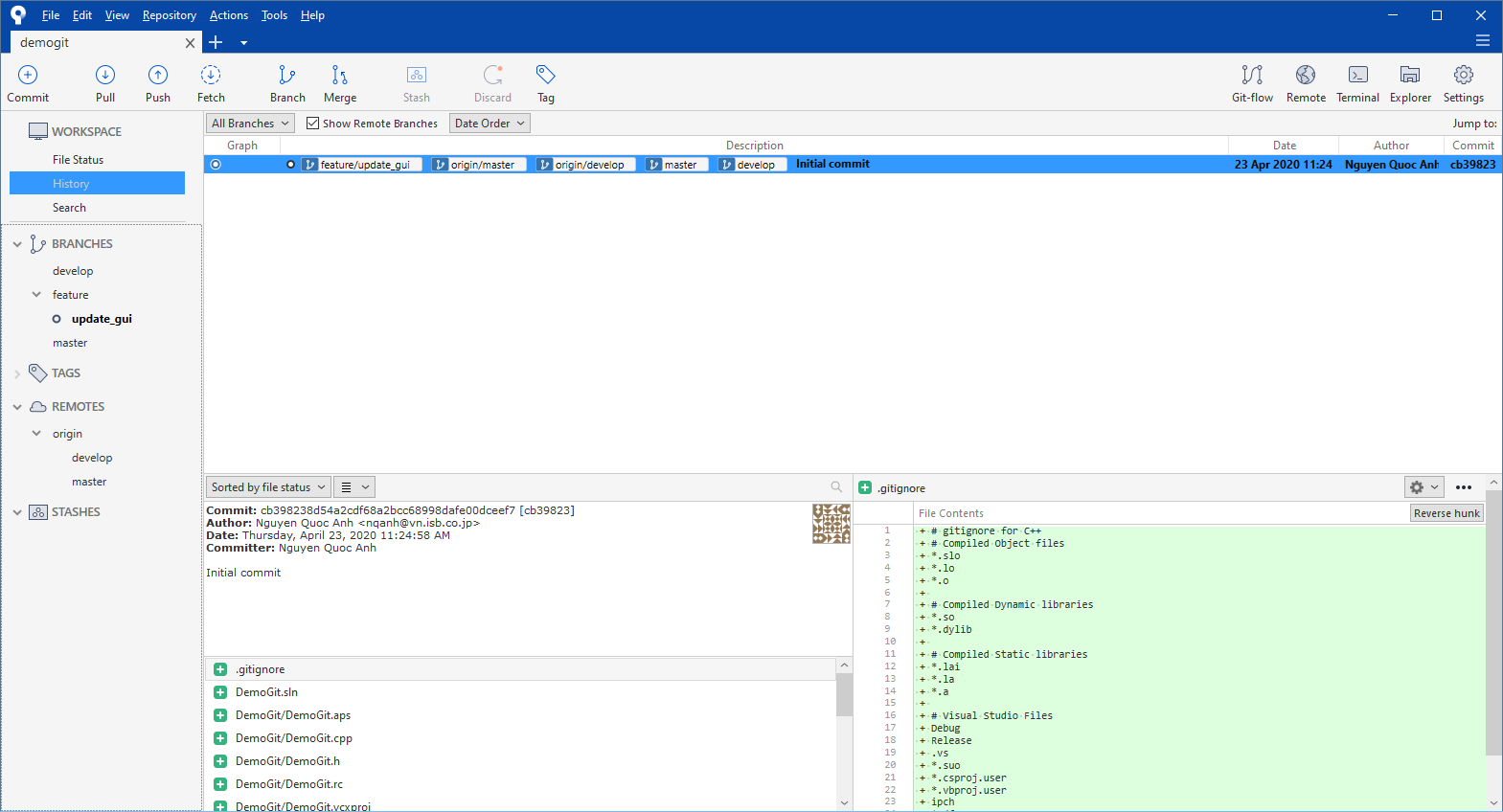


* + 1. Create a new feature

Click on the button Git-flow on the toolbar of Source Tree 🡪 select “Start New Feature” 🡪 Input the Feature name (Ex: update\_gui)

**Note: The “Start at” branch must be “develop”**

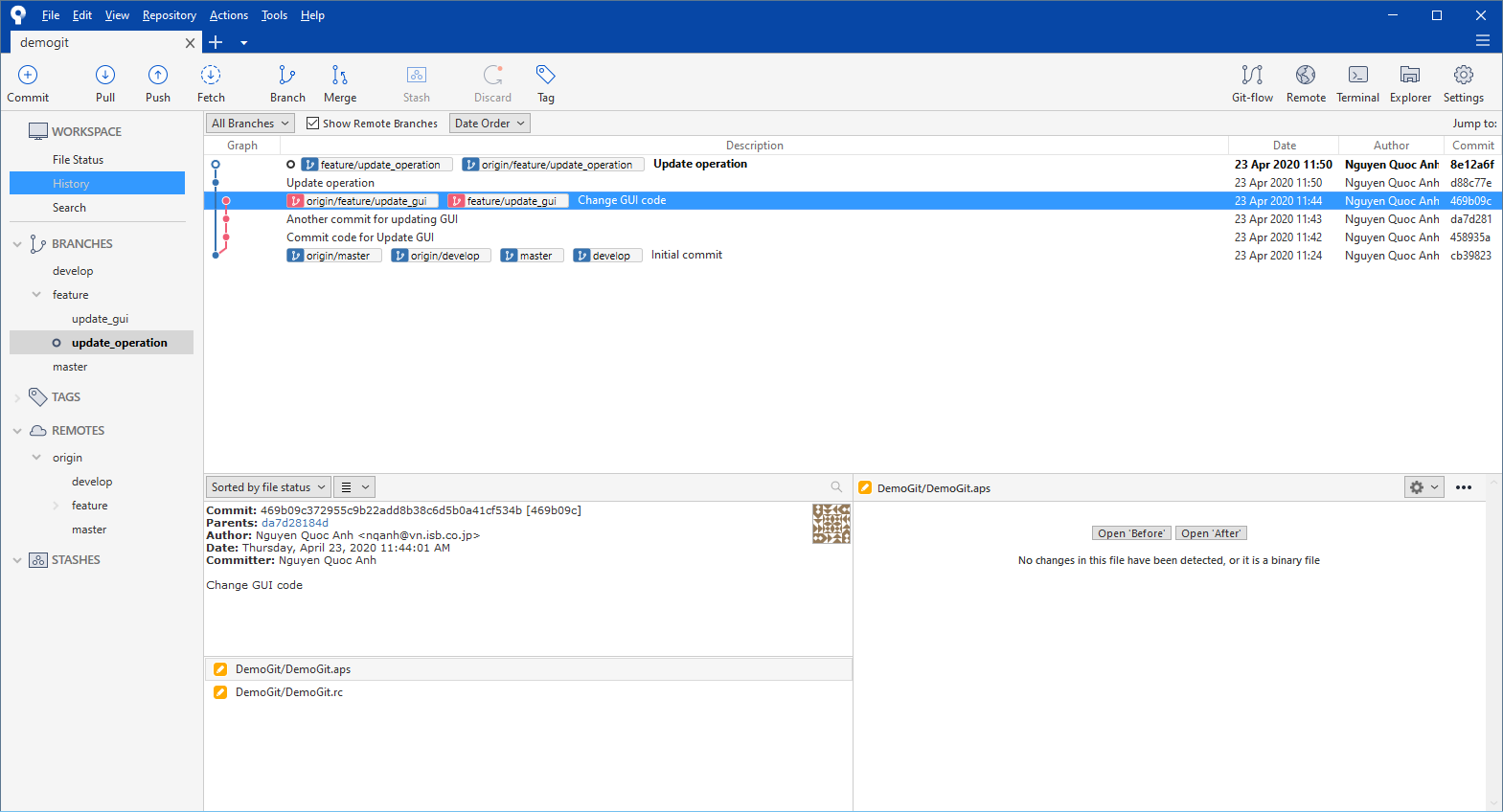




Push the new branch to Server

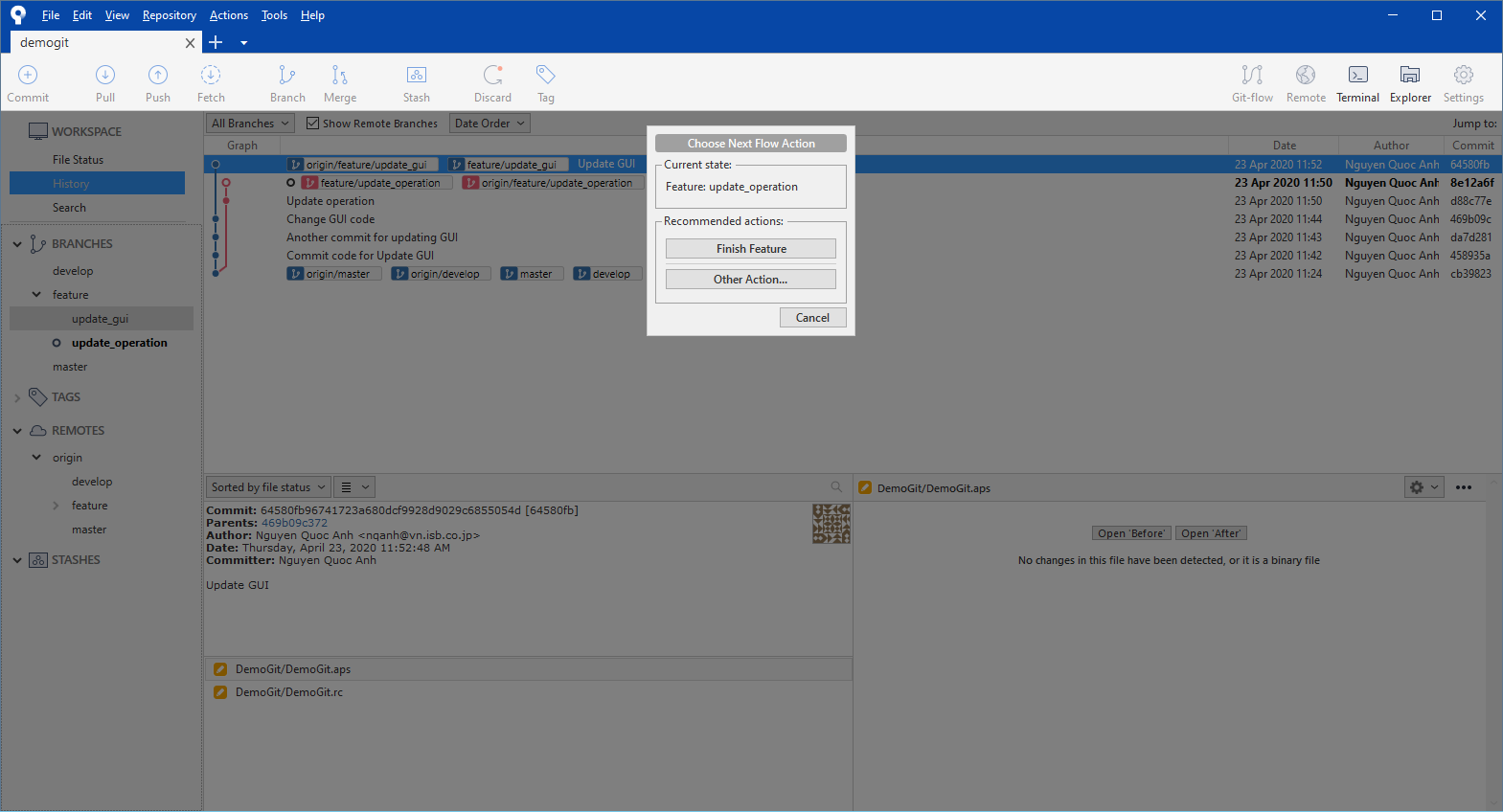
* ***We will create 2 feature branches in this example (update\_gui, update\_operation)***
  + 1. Commit code to new feature branches

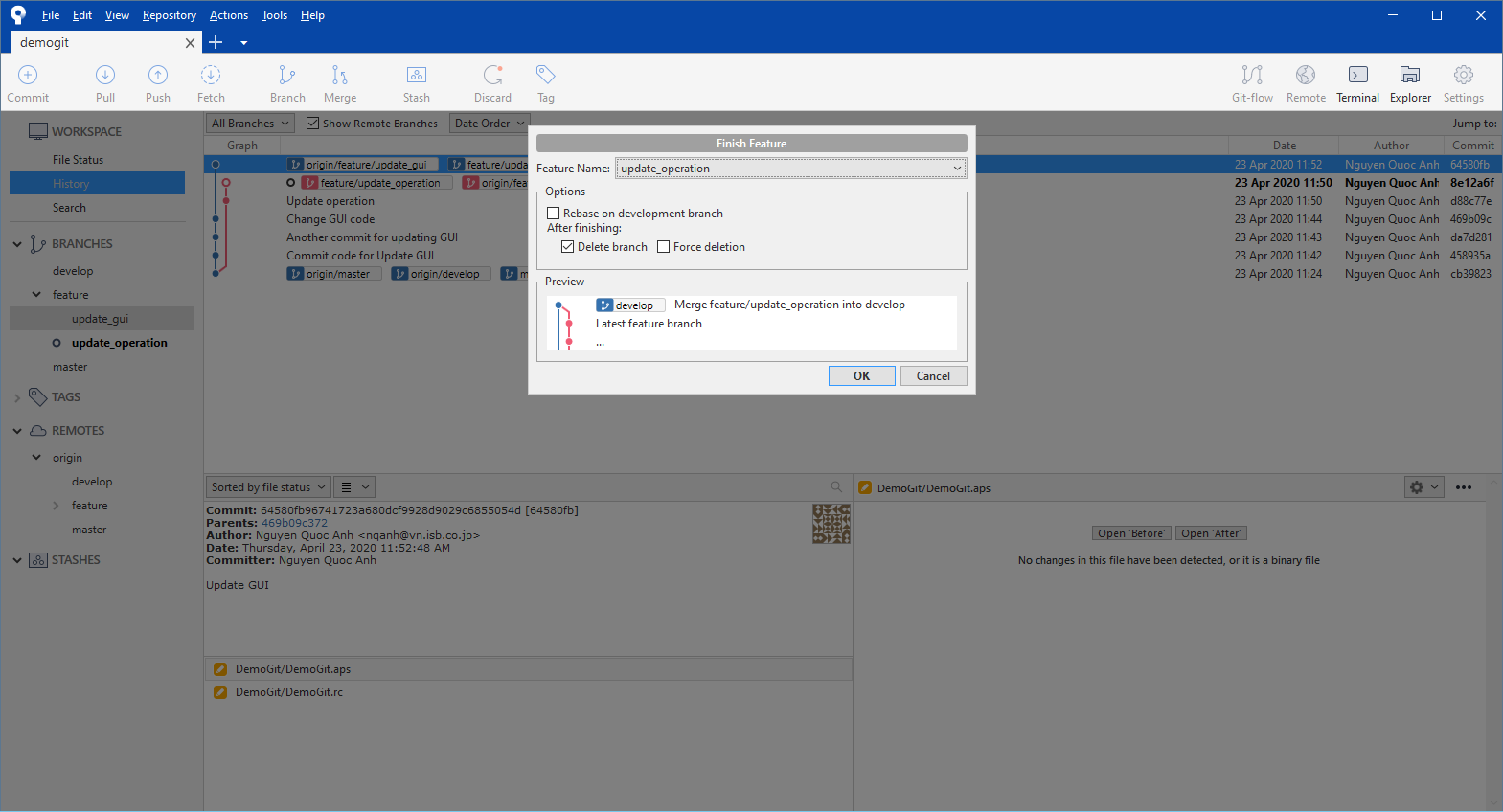
Modify code and commit to the created feature branches



* + 1. Finish feature (merge to develop branch)

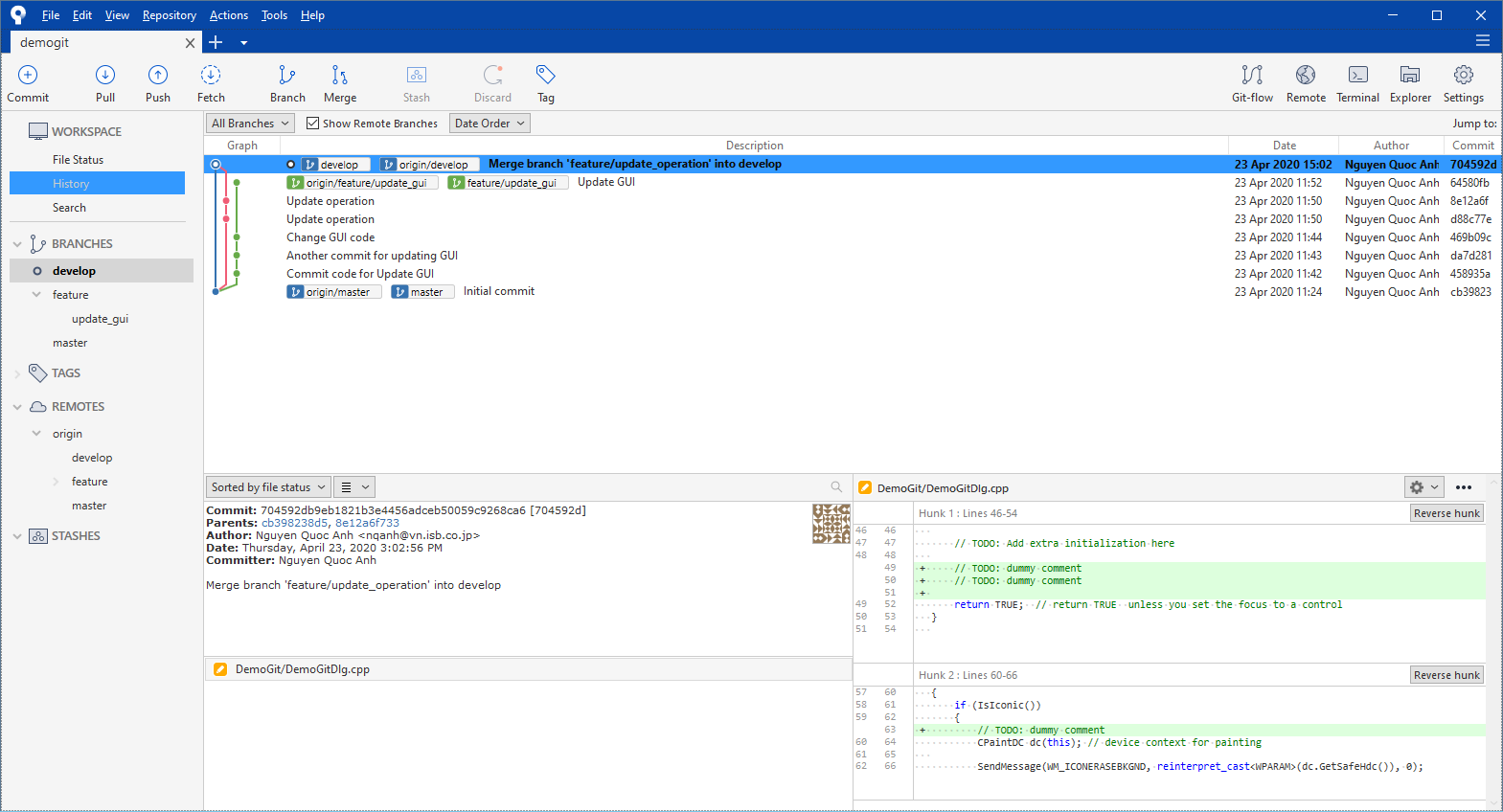
Click on Git-flow button 🡪 Click Finish Feature



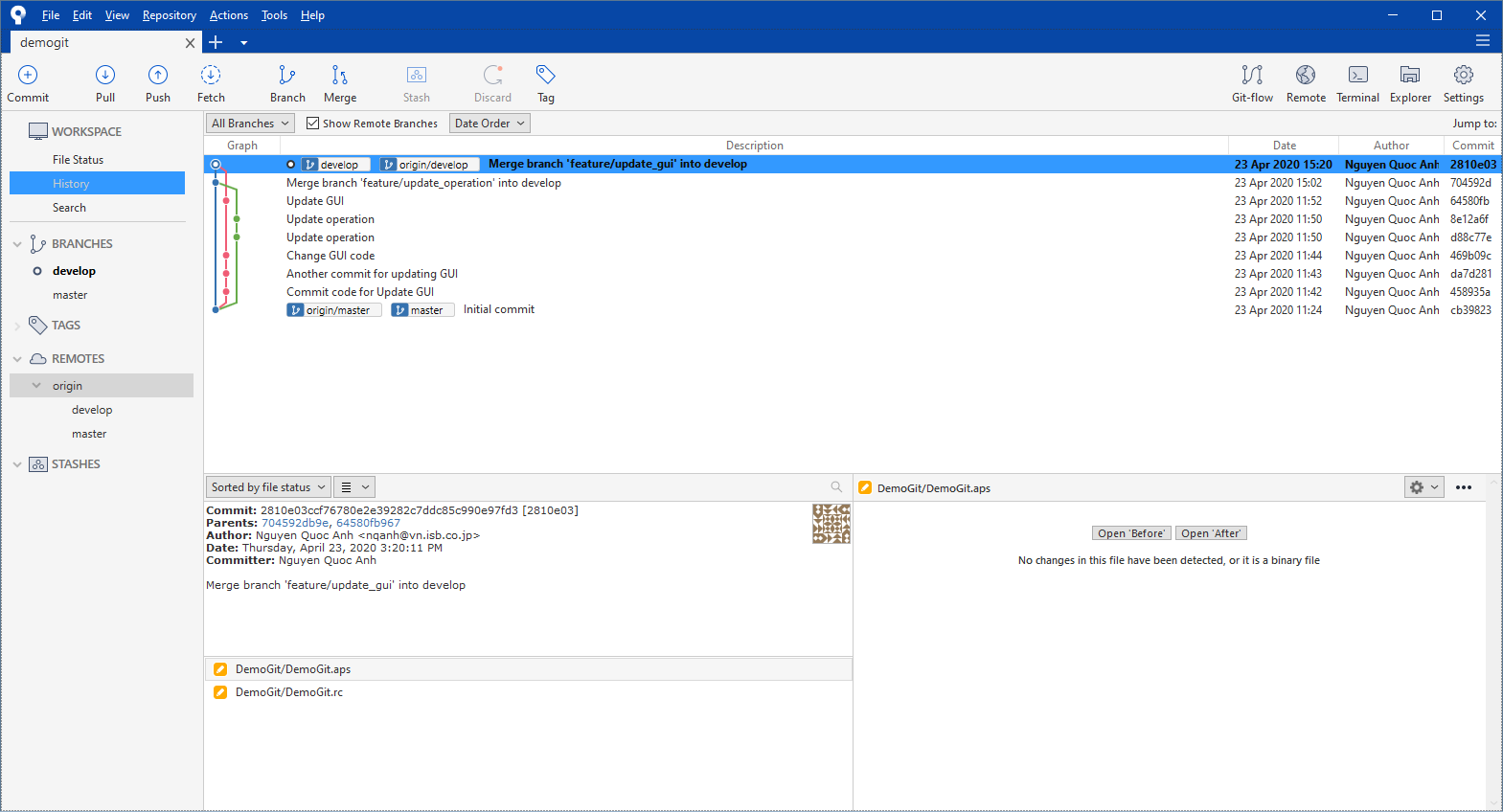


Click OK

After that, push the **develop** branch to Server

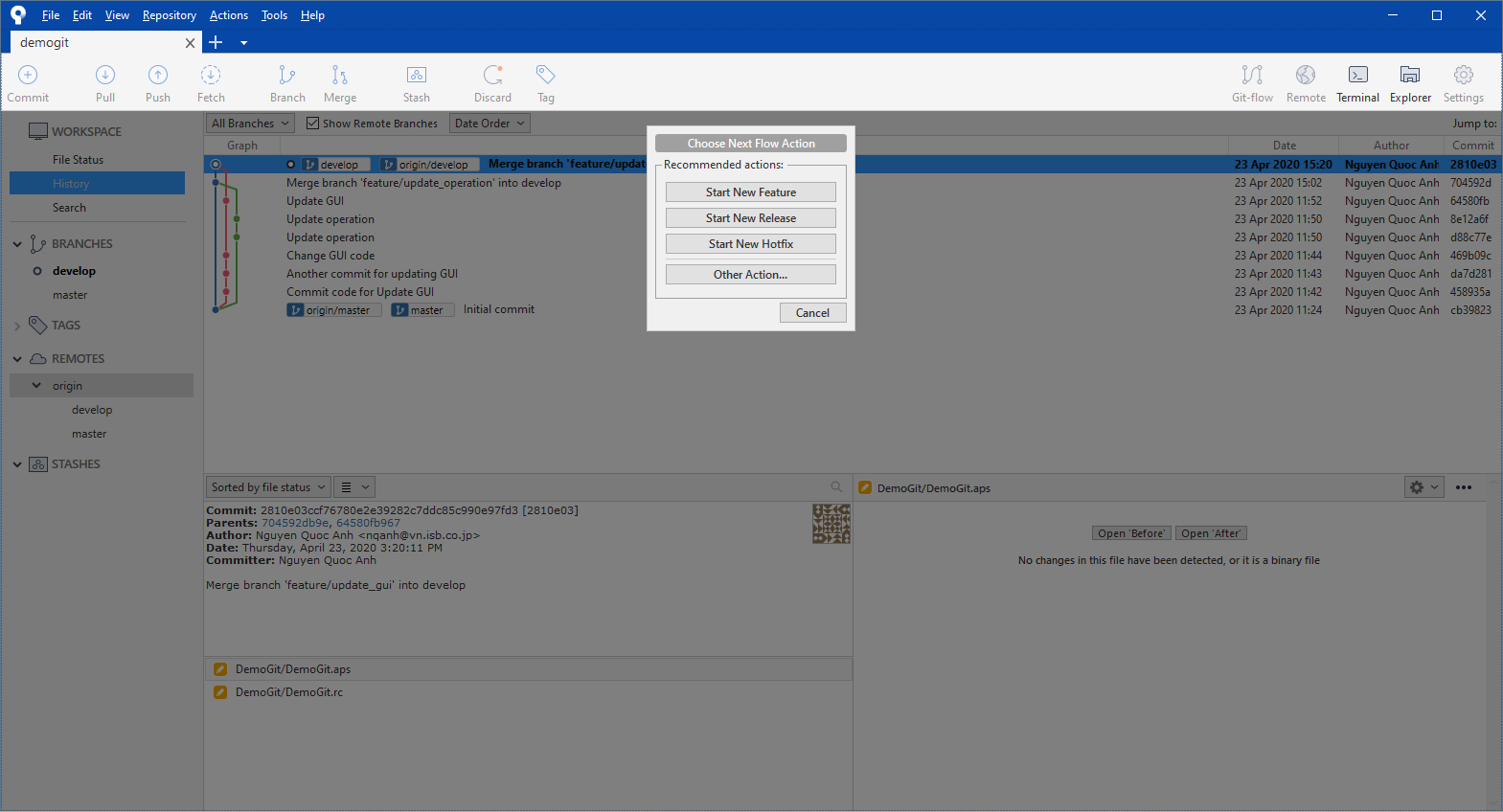


Do the same for other branches

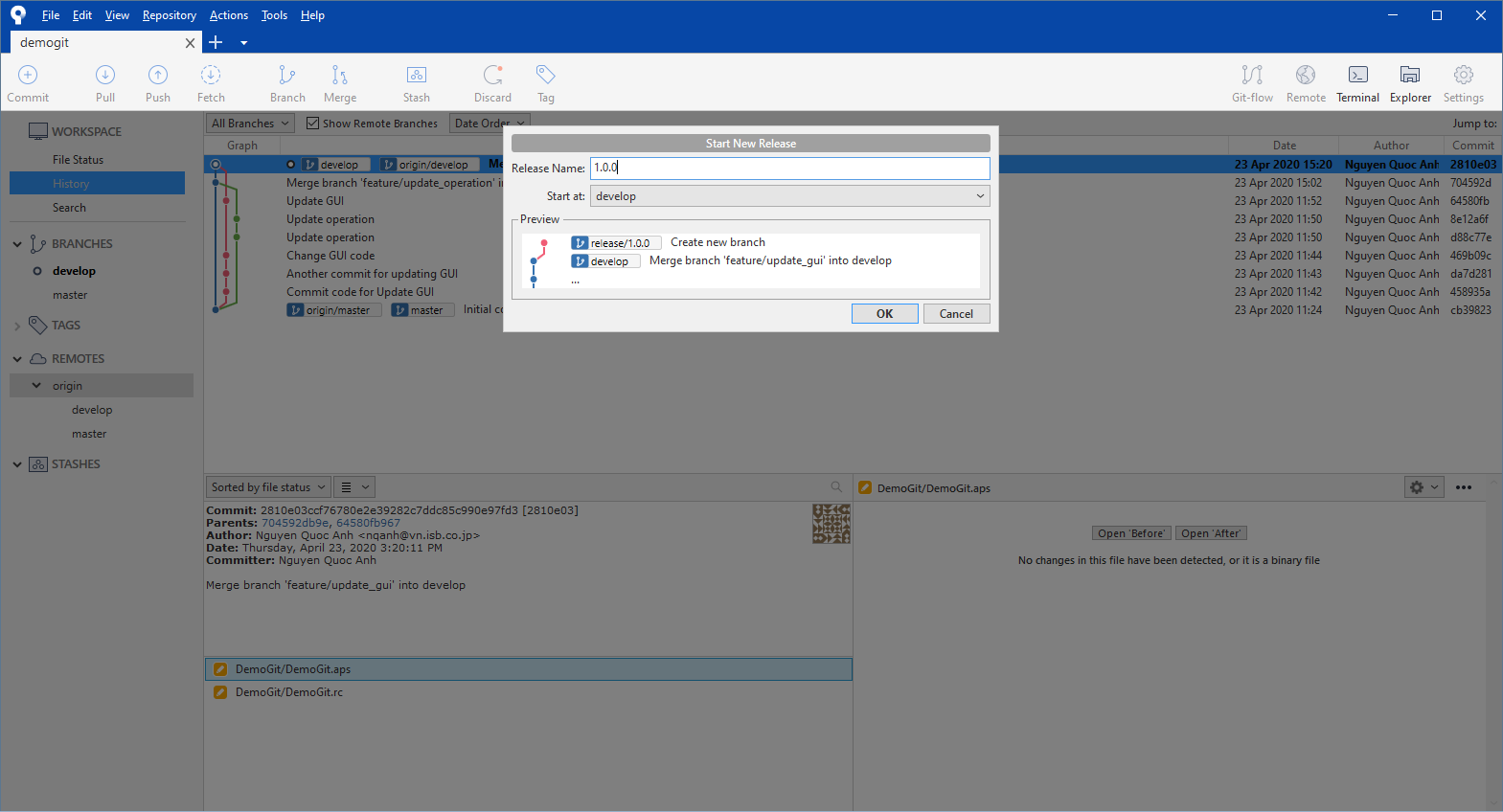


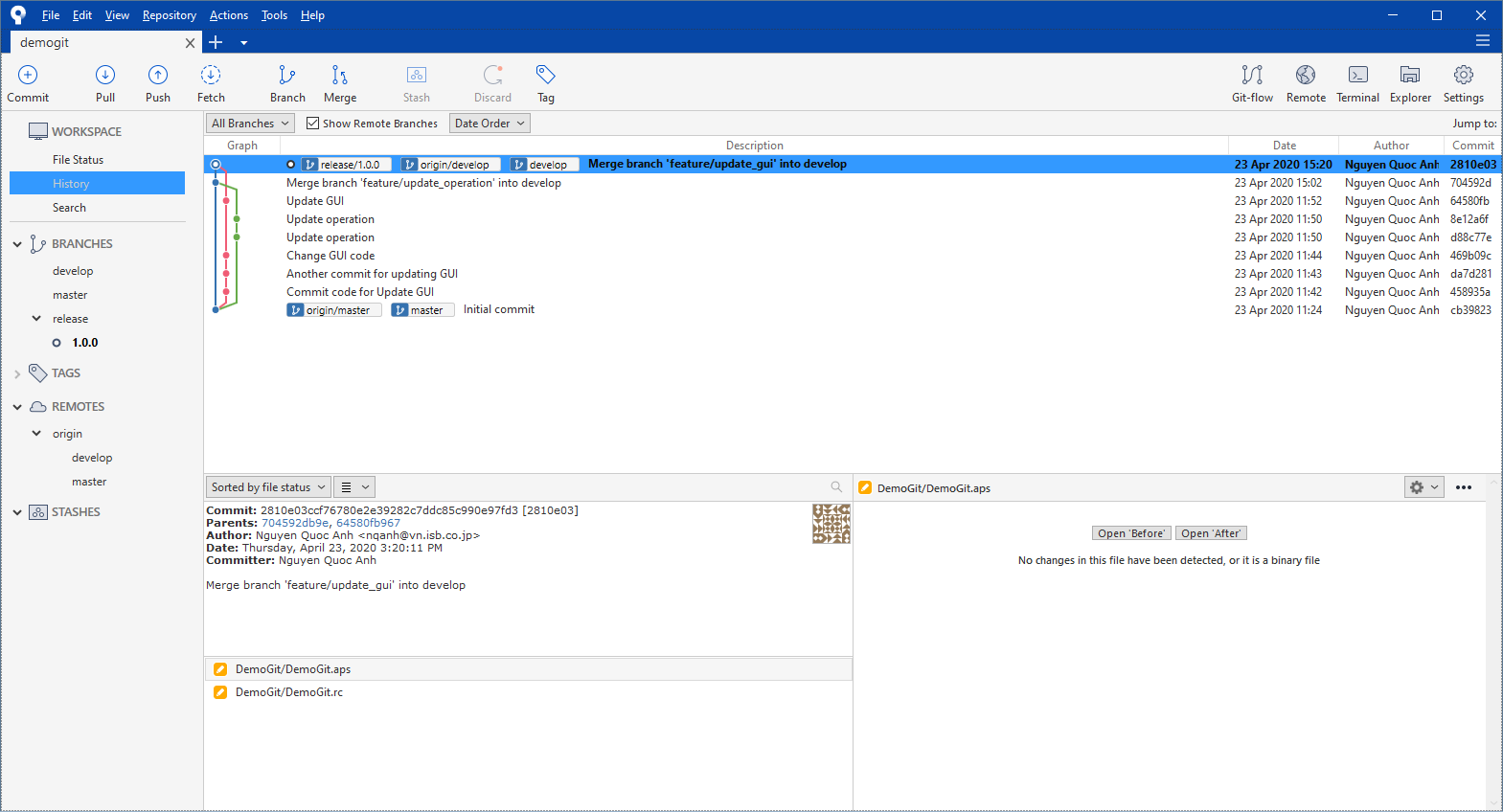
* + 1. Create release branch

On the develop branch 🡪 Click on Git-flow button 🡪 Start New Release

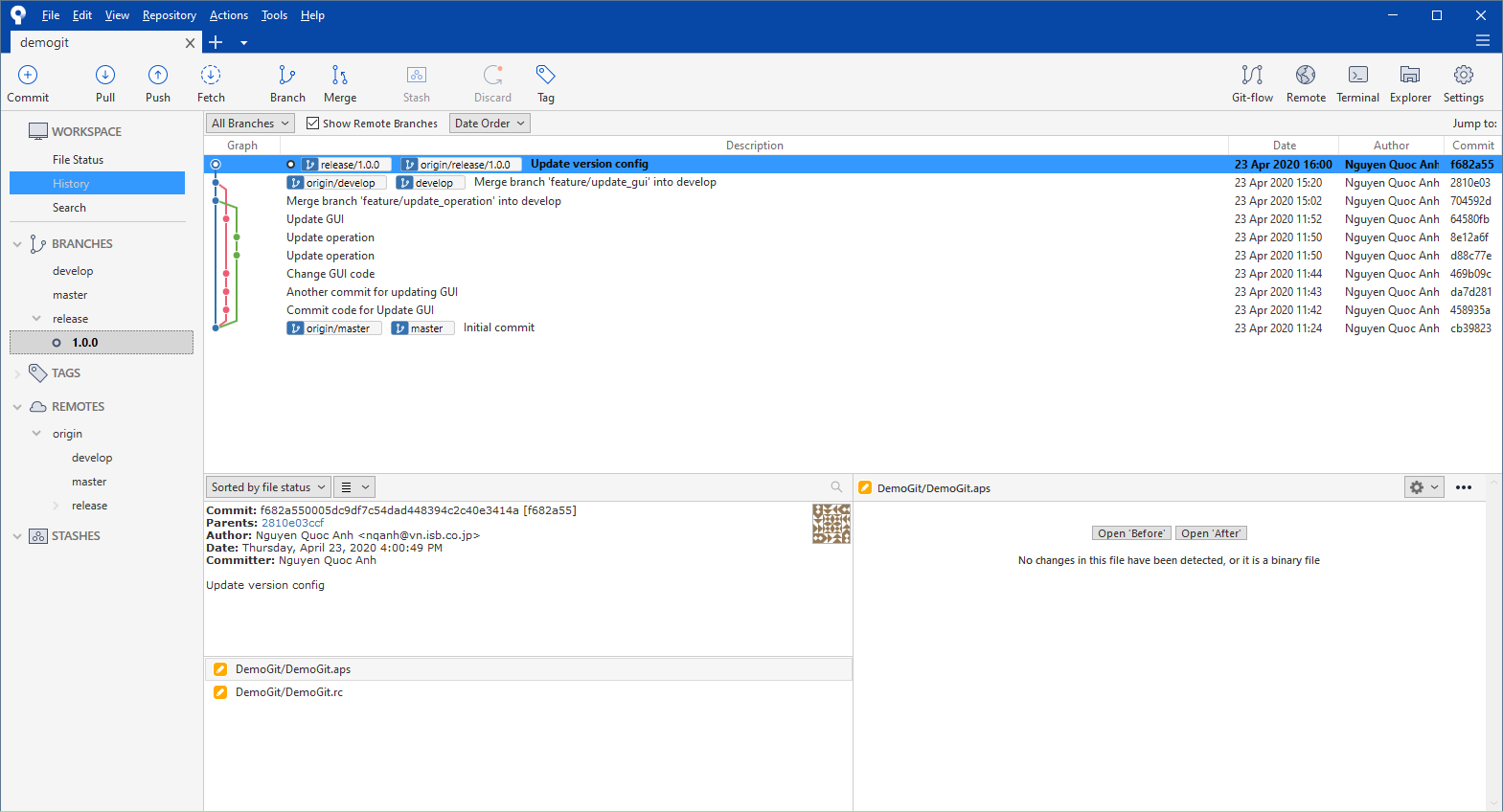


Input the branch name 🡪 Click OK



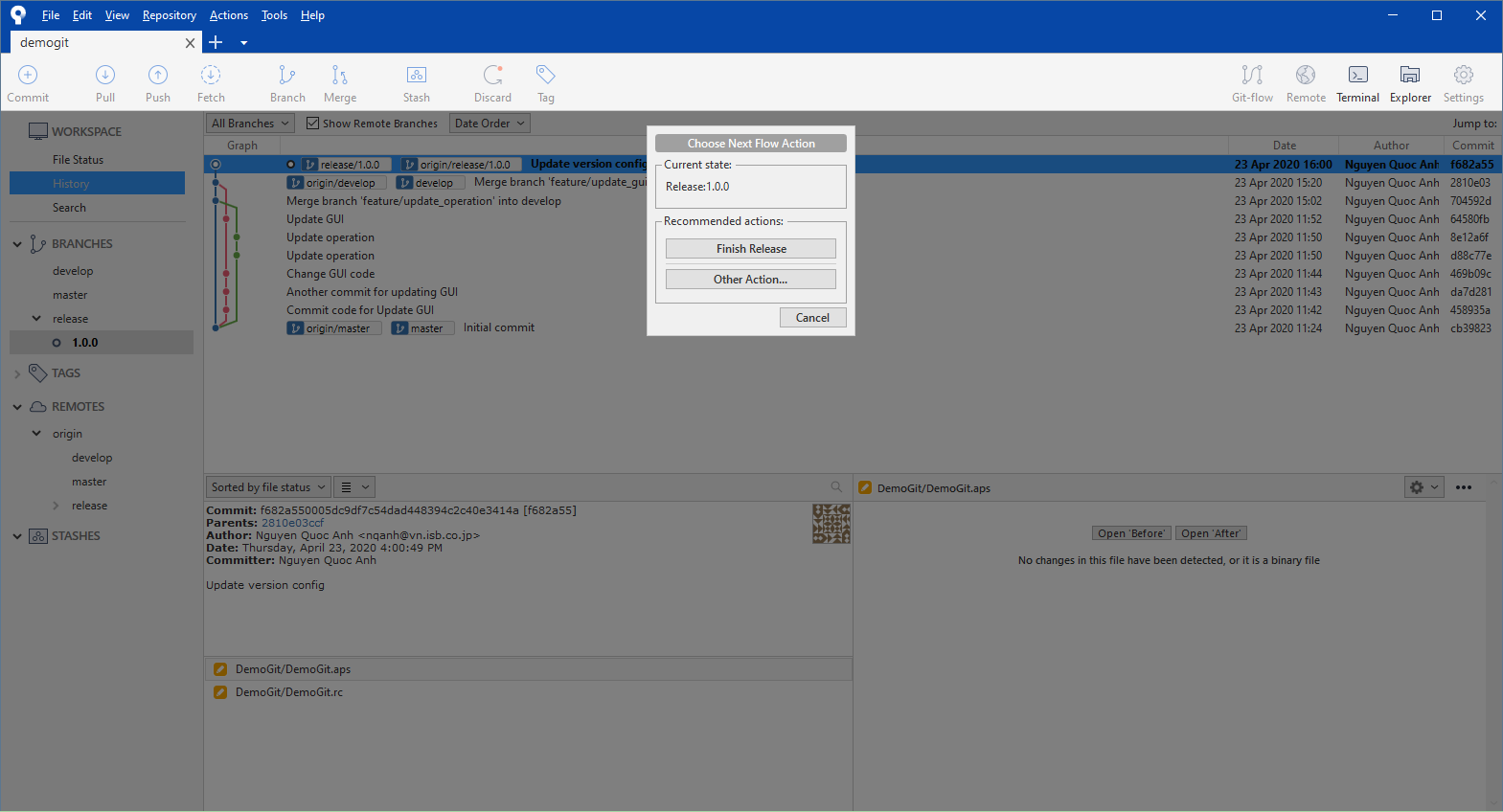


After that, we can commit changes to release branch (version config, etc…)

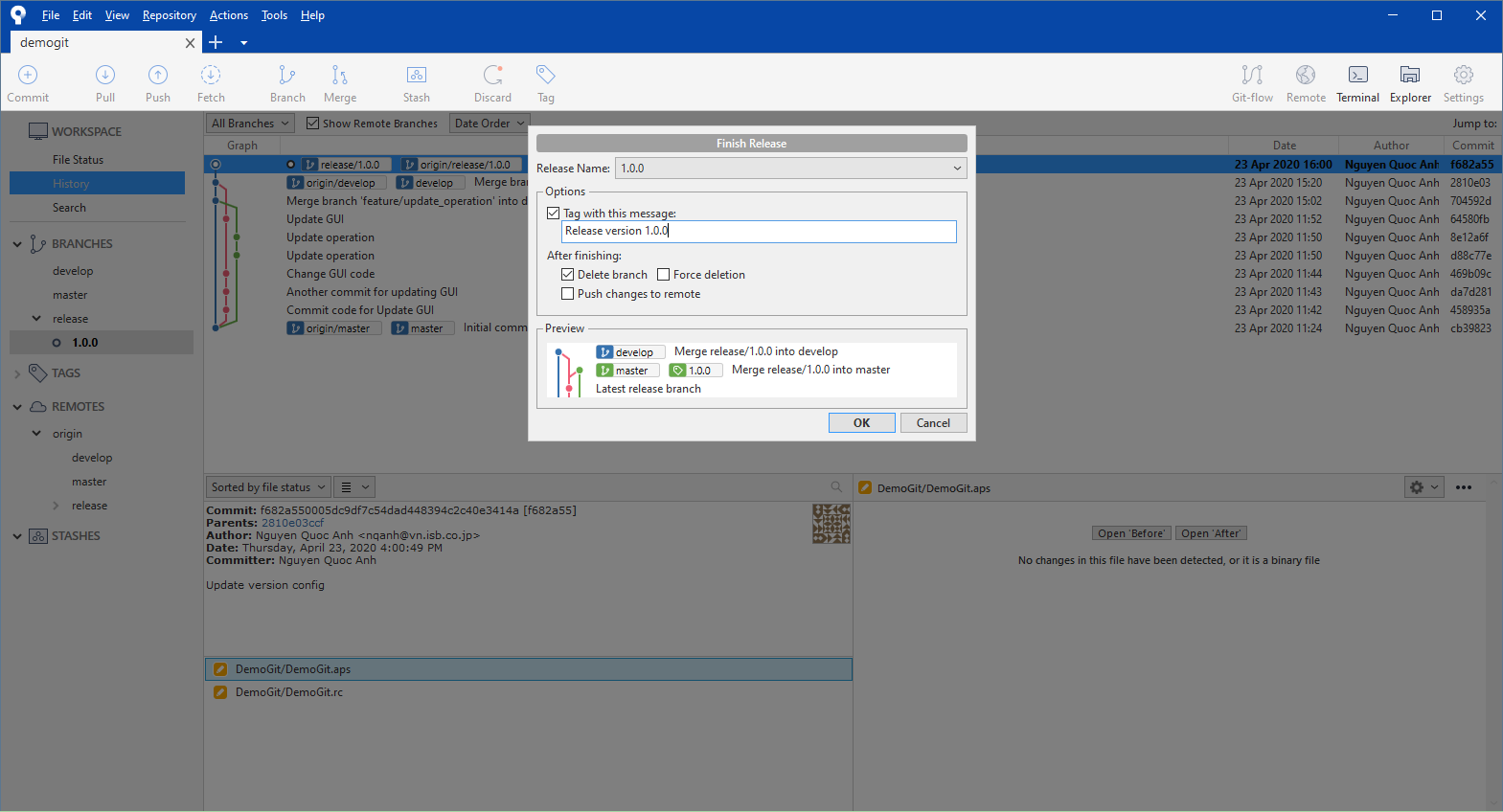


* + 1. Finish release branch

On the current release branch 🡪 Click on Git-flow button 🡪 Finish Release

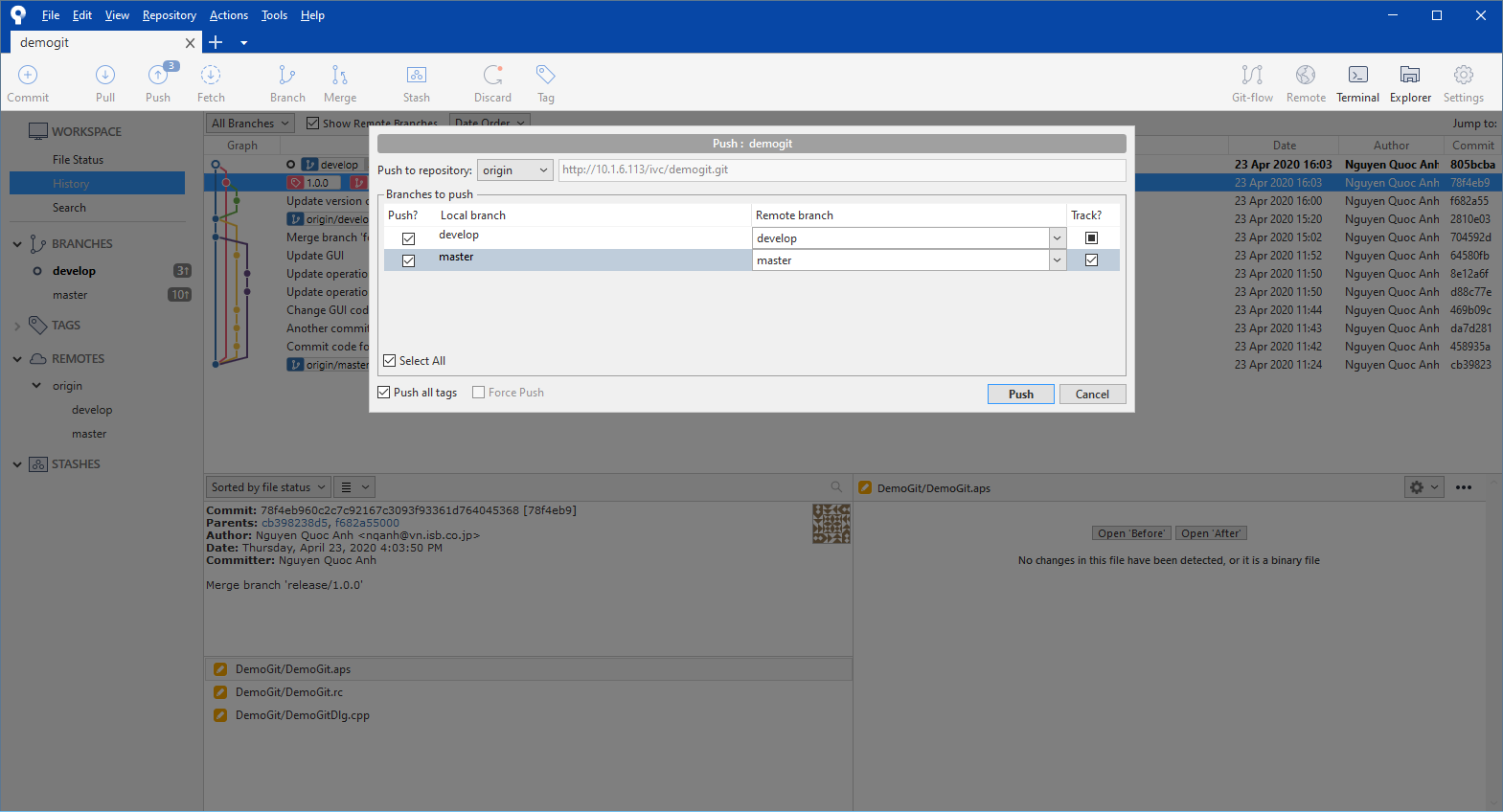


Input the information



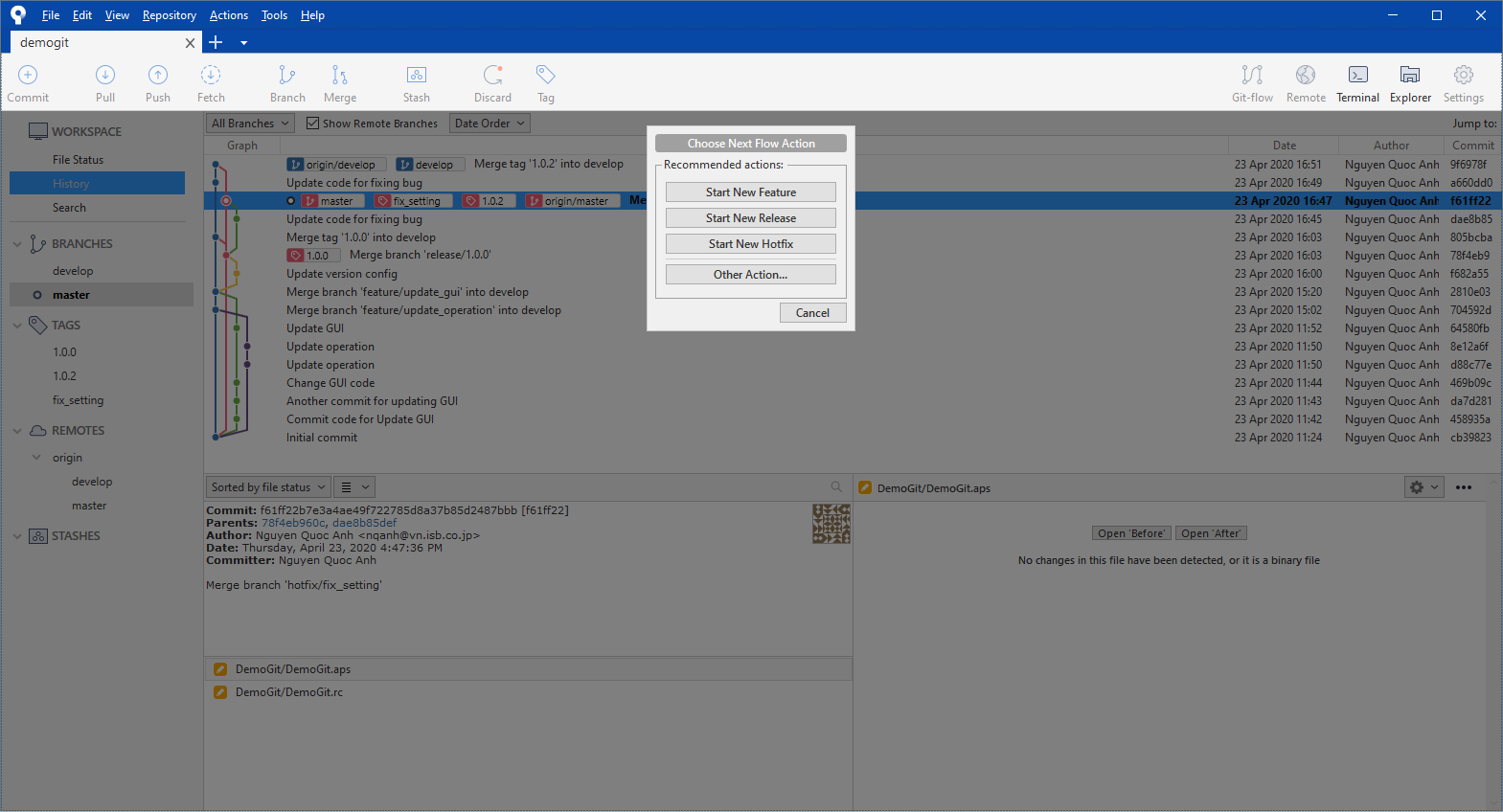
It will merge the current **release** branch to **master** and **develop** branch, after that, create a Tag for this release

Push changes to Server (both branches: master, develop)

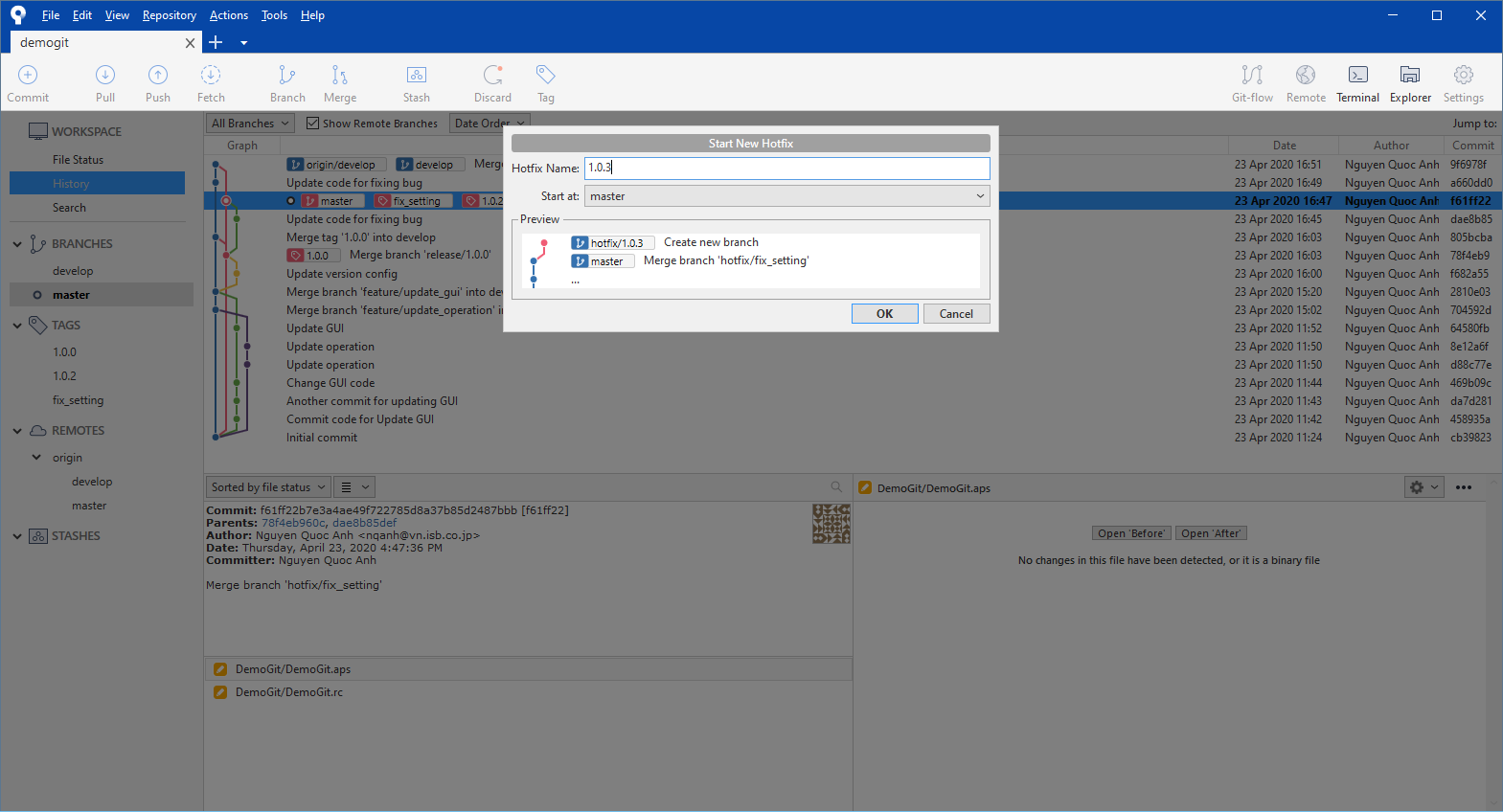


* + 1. Create hotfix (for bugs after release)

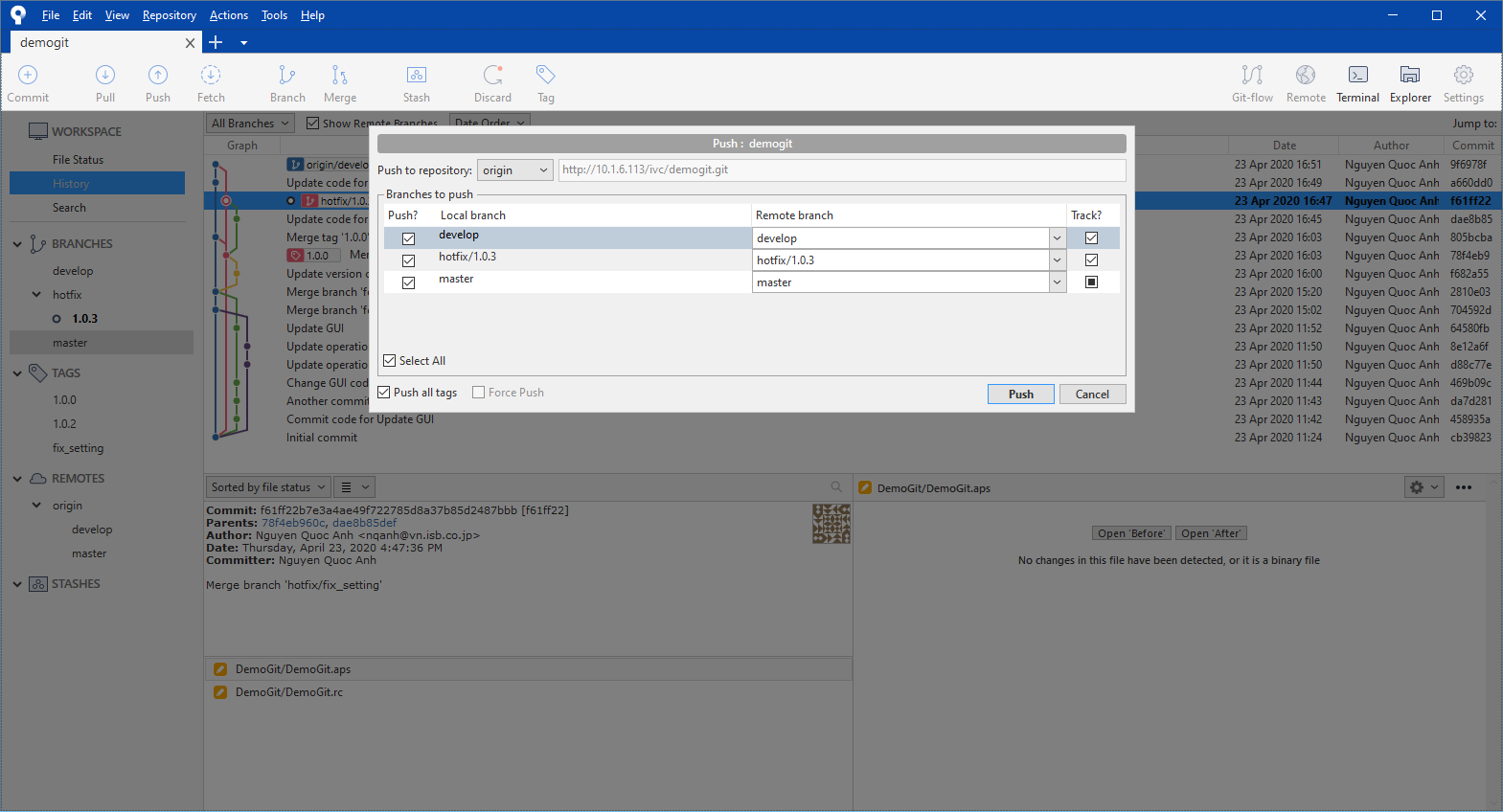
On the **master** branch 🡪 Click on Git-flow button 🡪 Start New Hotfix

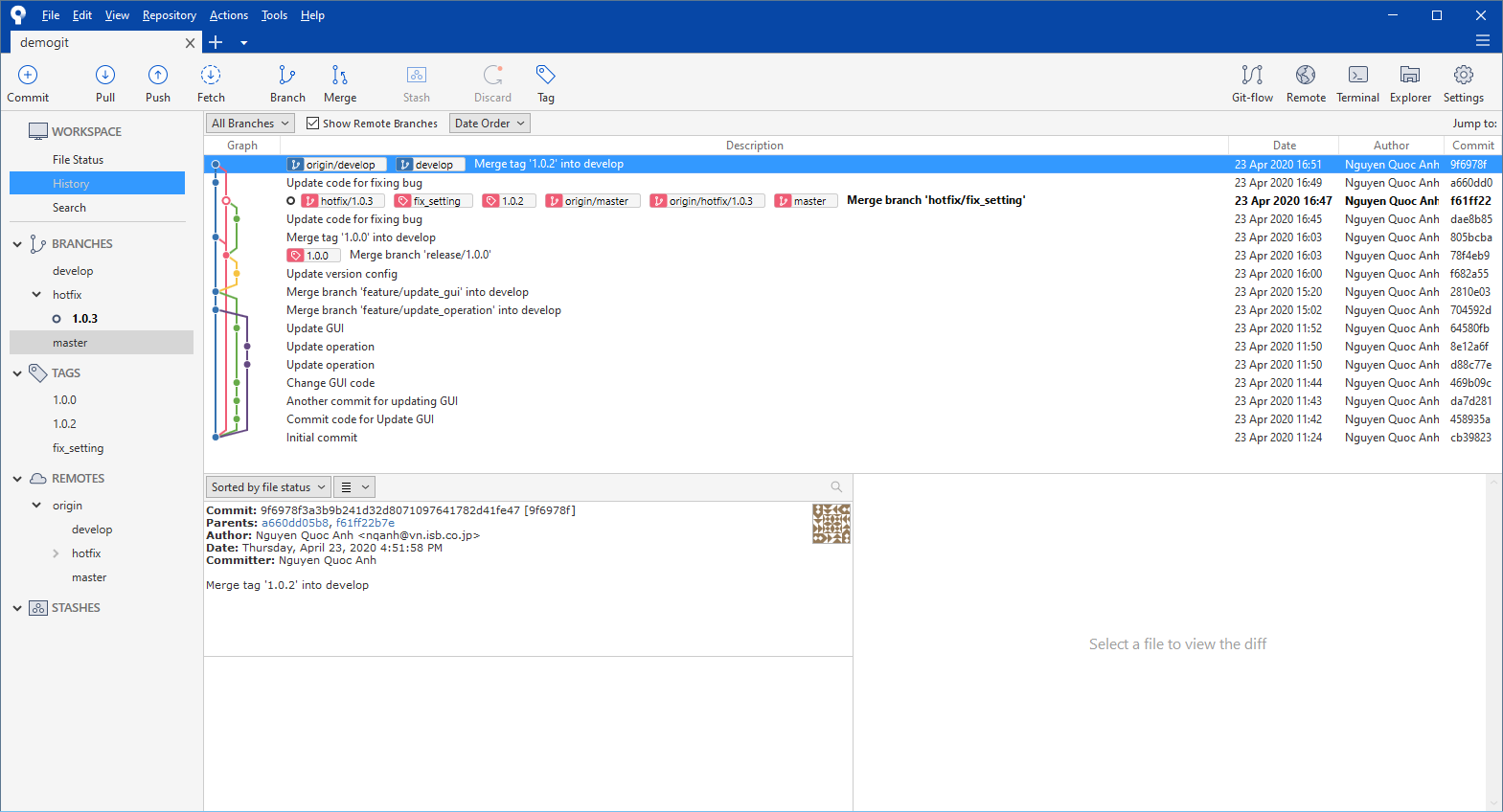


Input the information



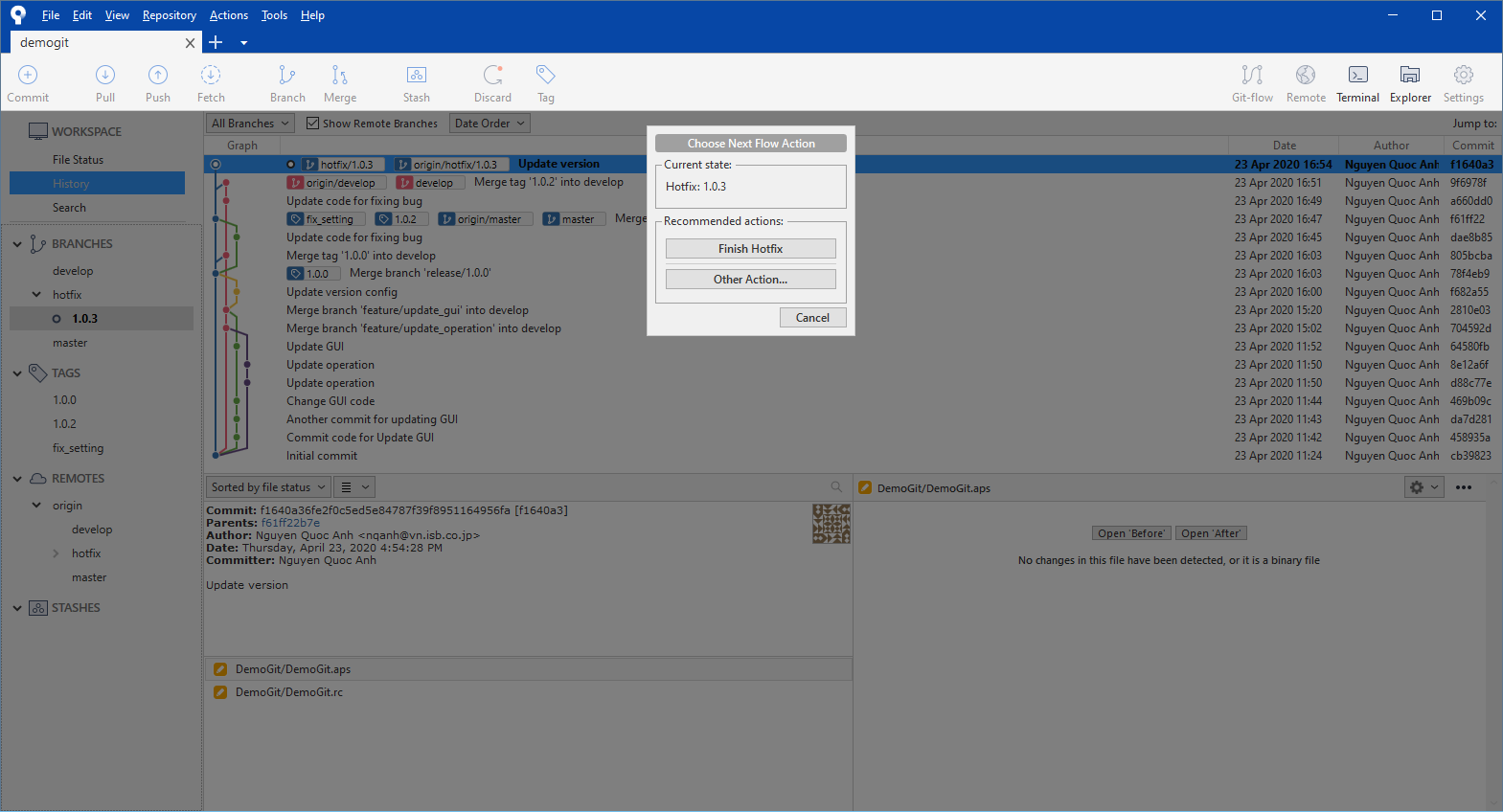
Click OK and push the new branch to server



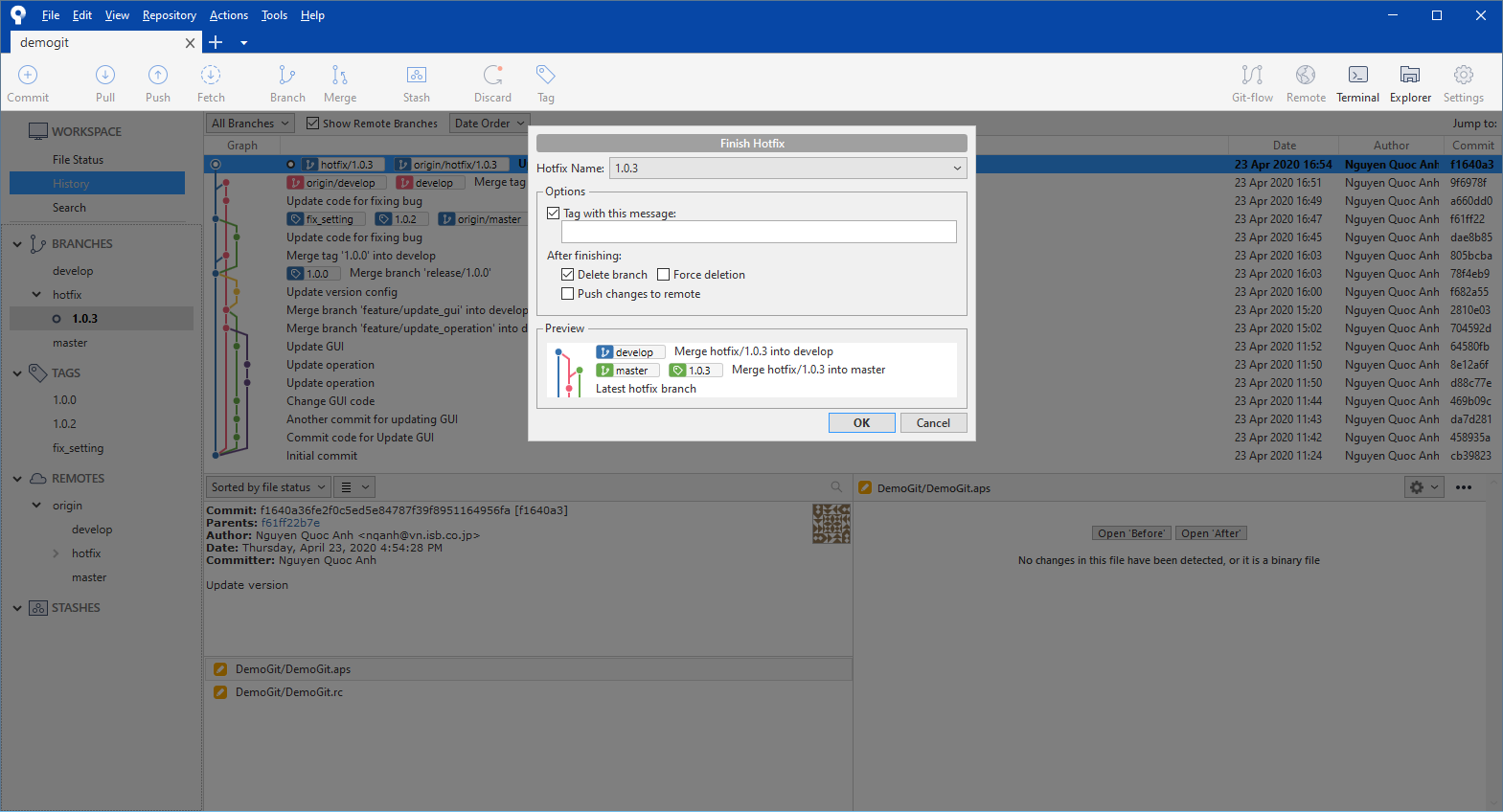


Update code for the hotfix and commit

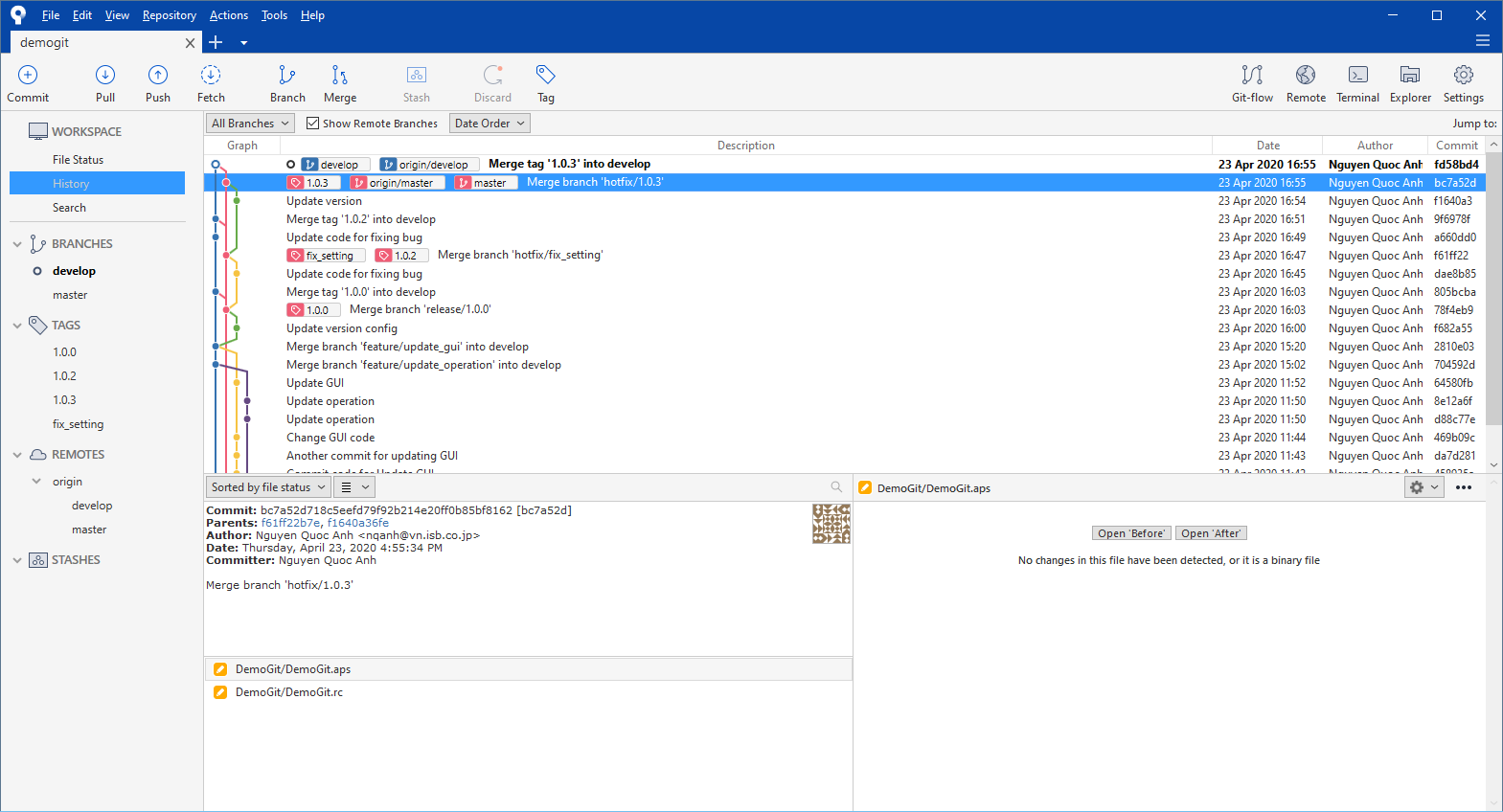
Then Finish the hotfix: Click on Git-flow 🡪 Finish Hotfix



Input information then OK



Push change to Server



The new hotfix changes will be merged to **master** and **develop** branch

* 1. Review flow (create Merge request)

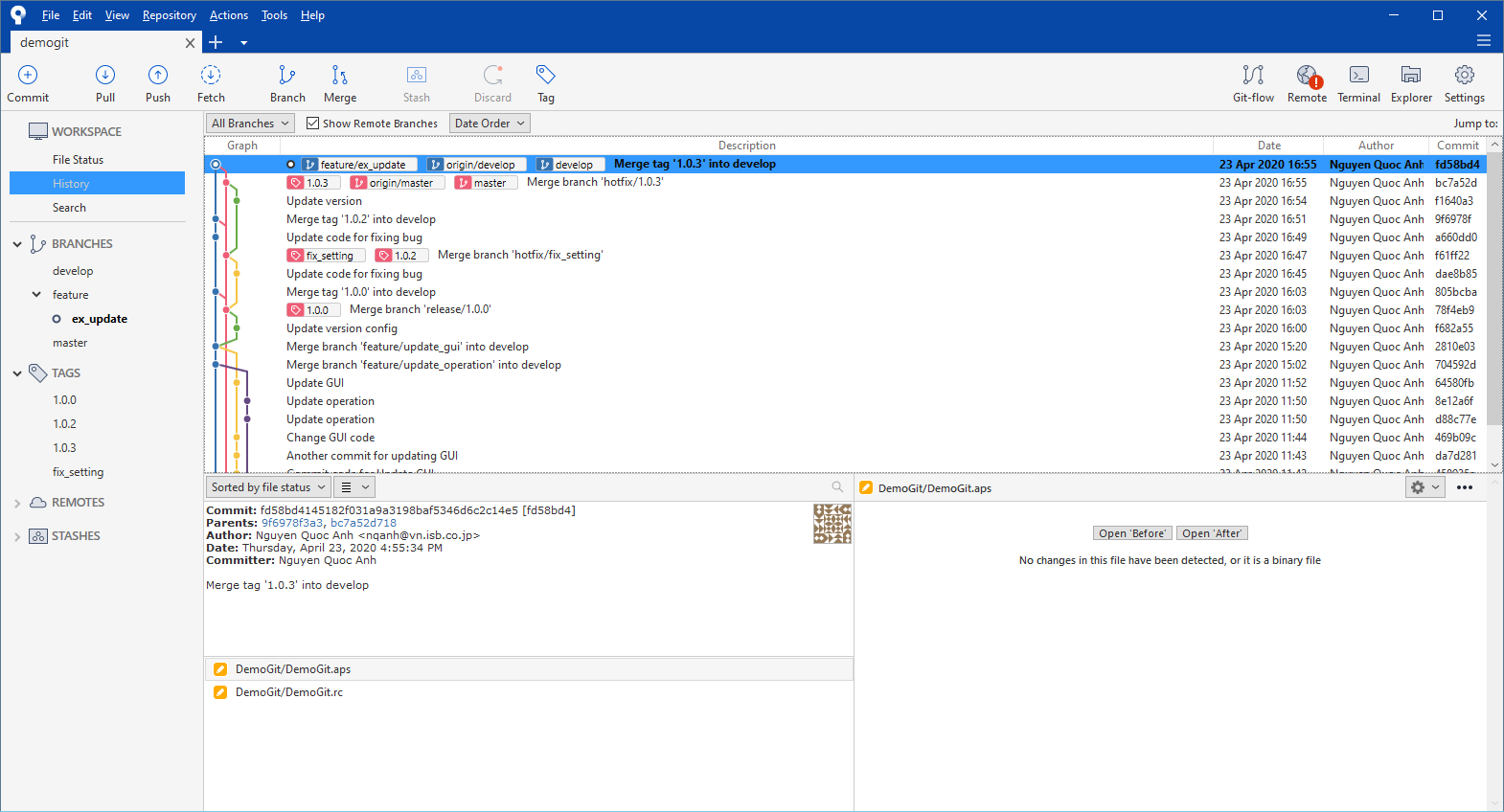
***We will apply this flow for feature-merge (merge feature branch to develop branch)***

We can create Pull request for other merges (ex: merge **release** to **master**, etc…), but we must keep the concept of the git-flow

* + 1. Create feature branch

Create the feature branch same as 2.1.4 Create a new feature

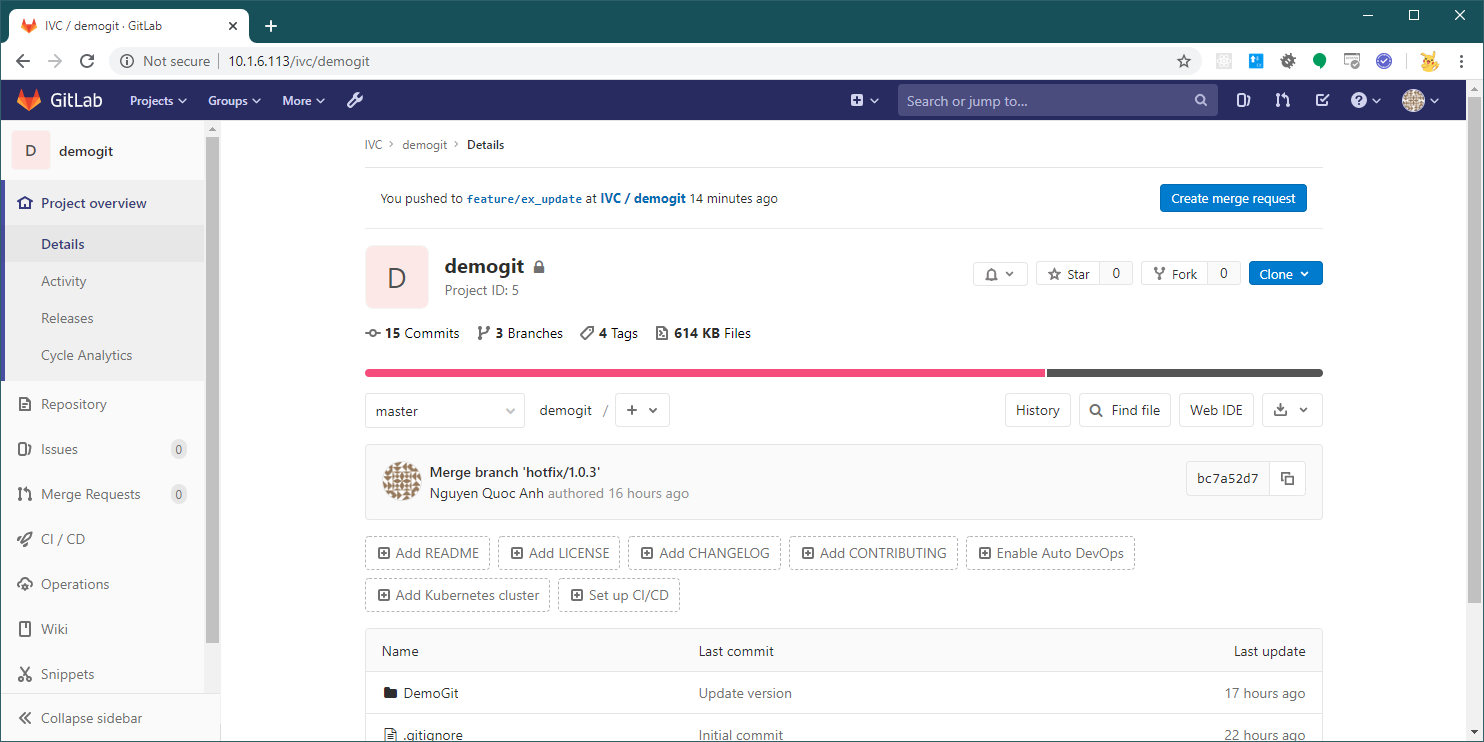
Example feature name: ex\_update



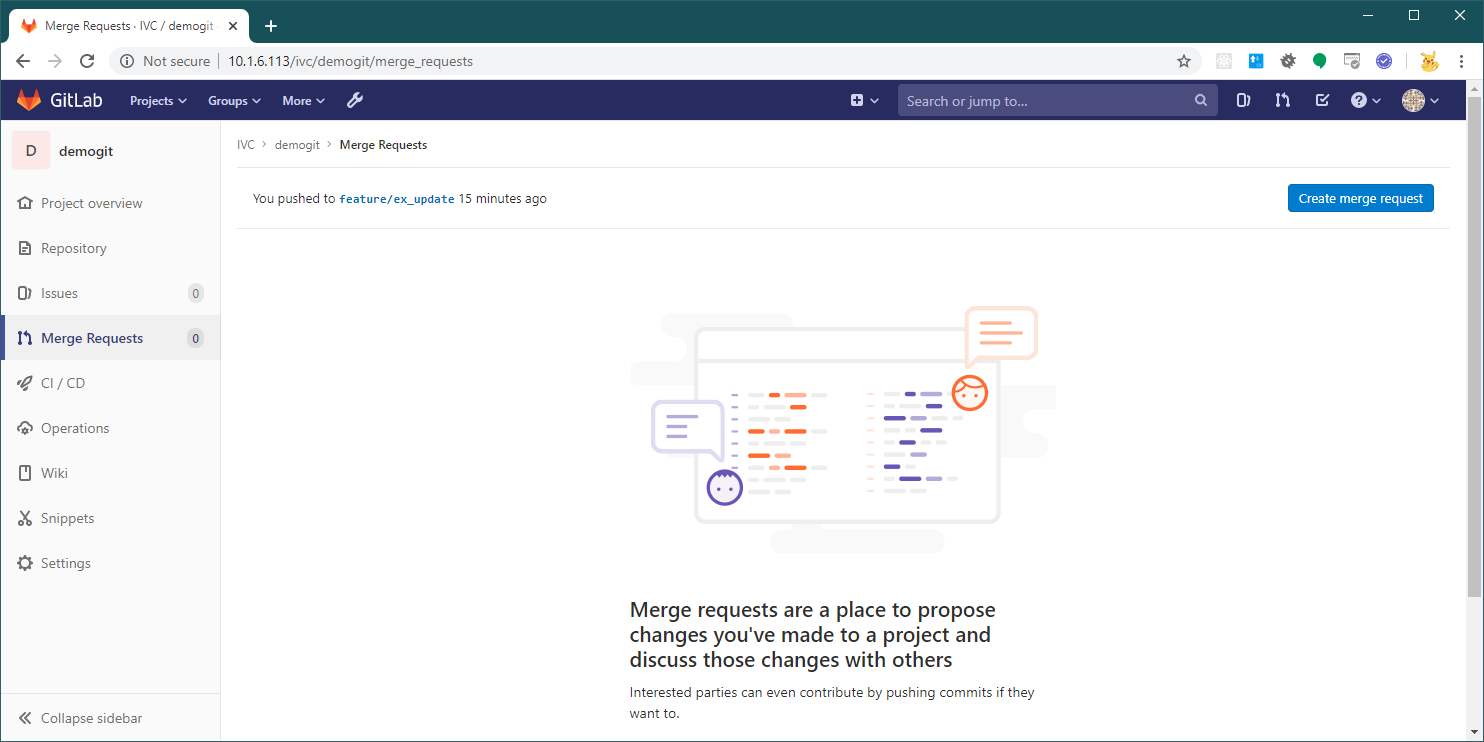
Then commit change this created branch

* + 1. Create Merge request (for feature developer)

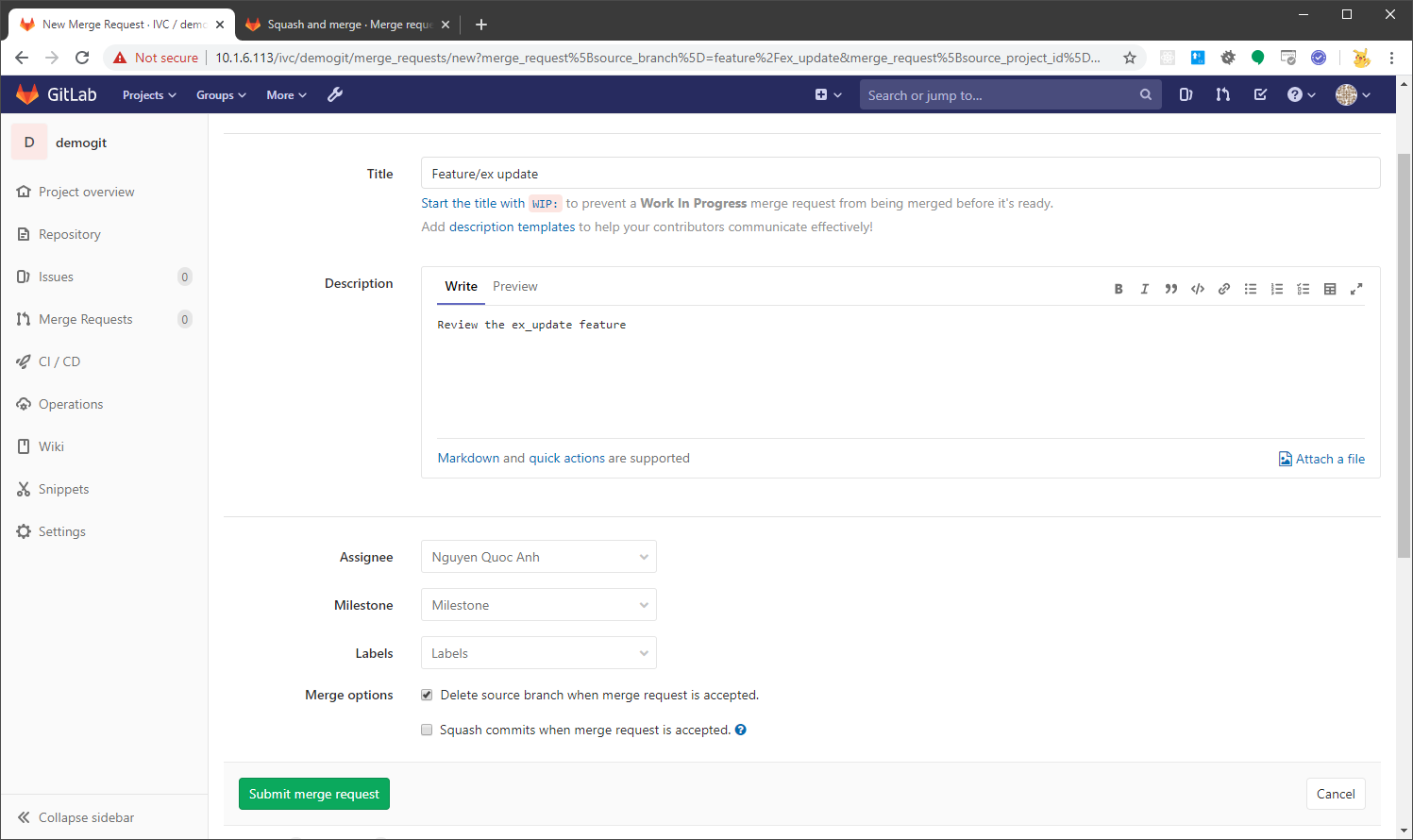
Go to the Gitlab website of the project



Go to “Merge Requests”

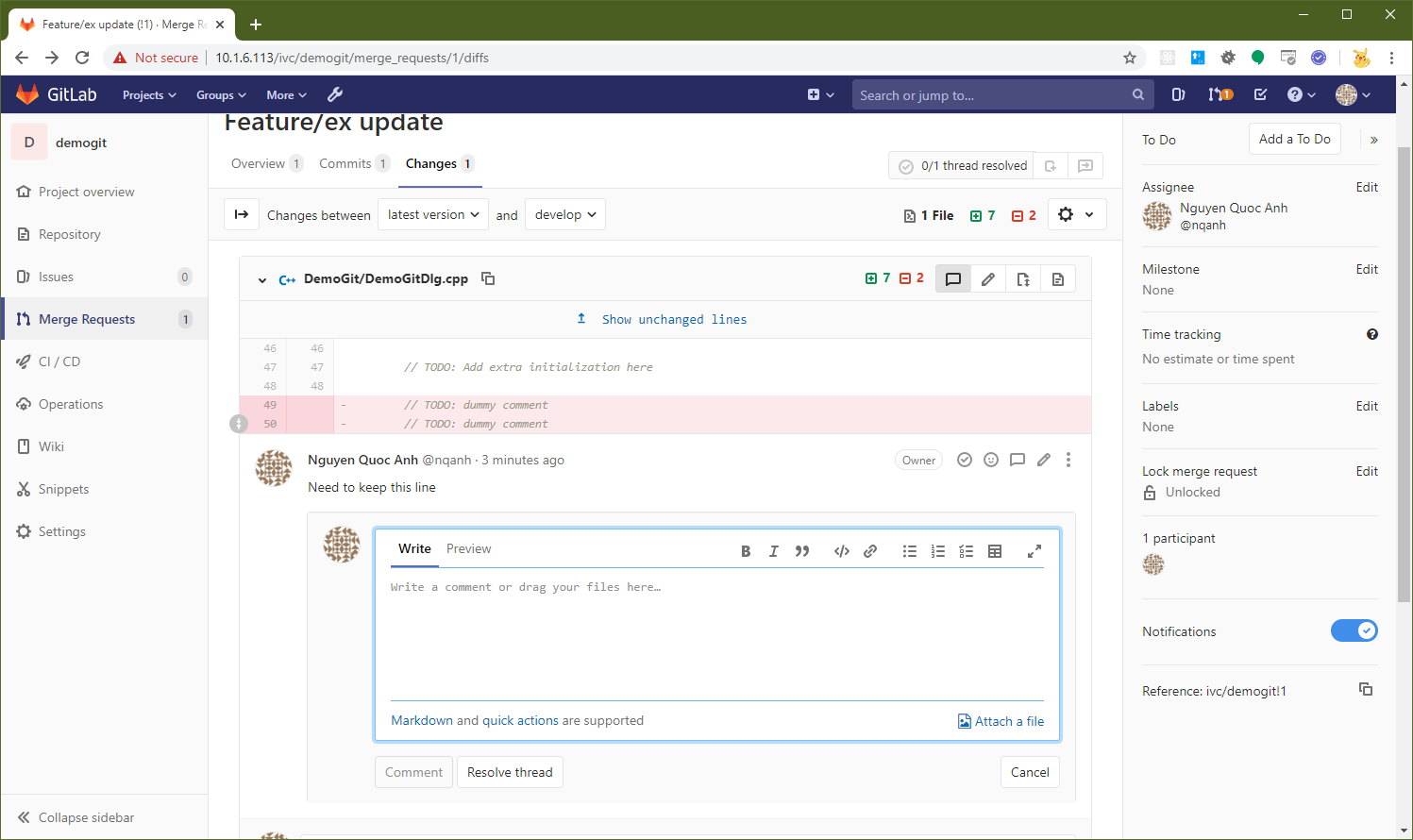


Create merge request (to **develop** branch)



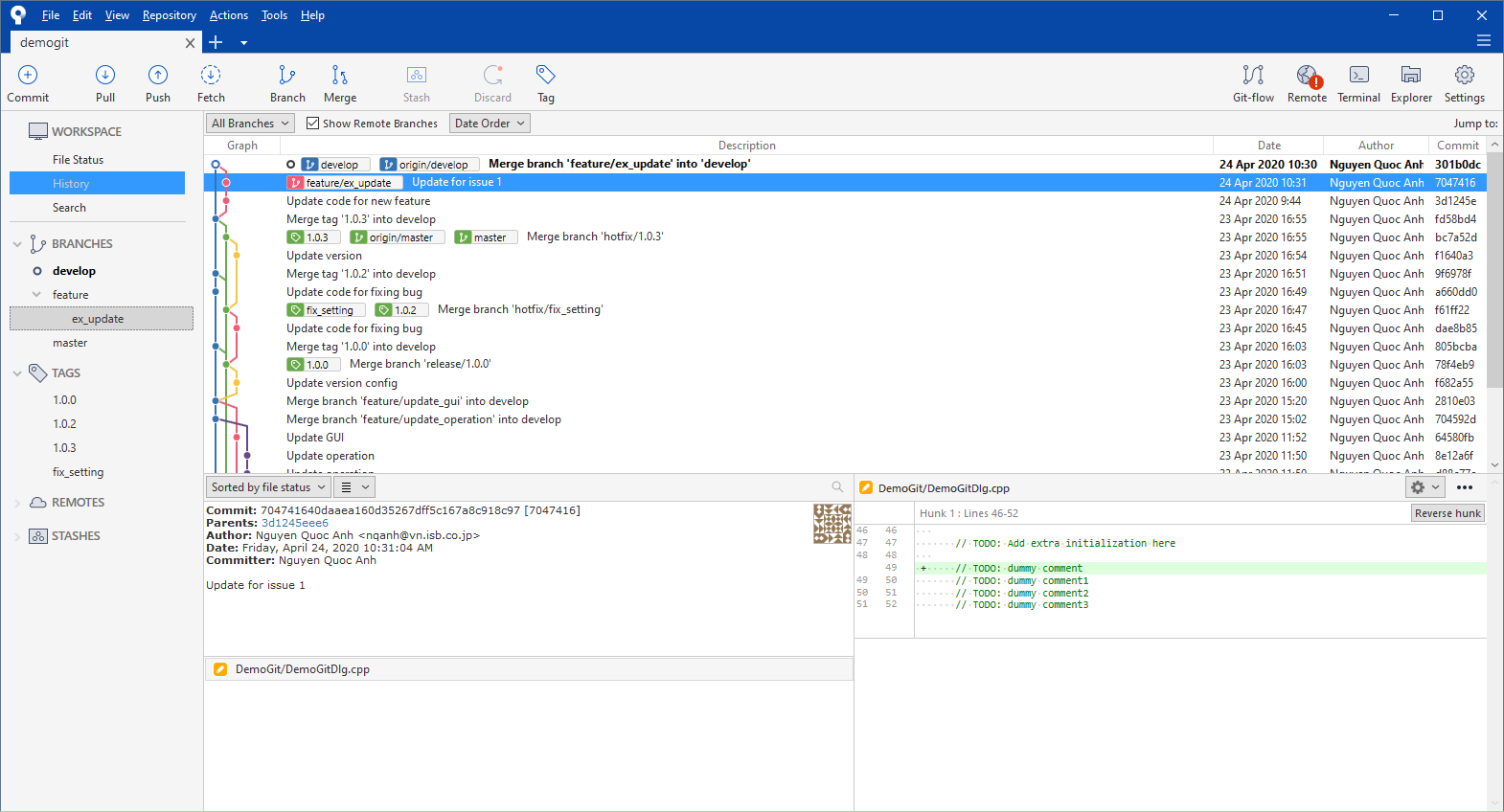
* + 1. Merge the Merge request (for Reviewer)

Reviewer create comment issues and developer update code for those issues

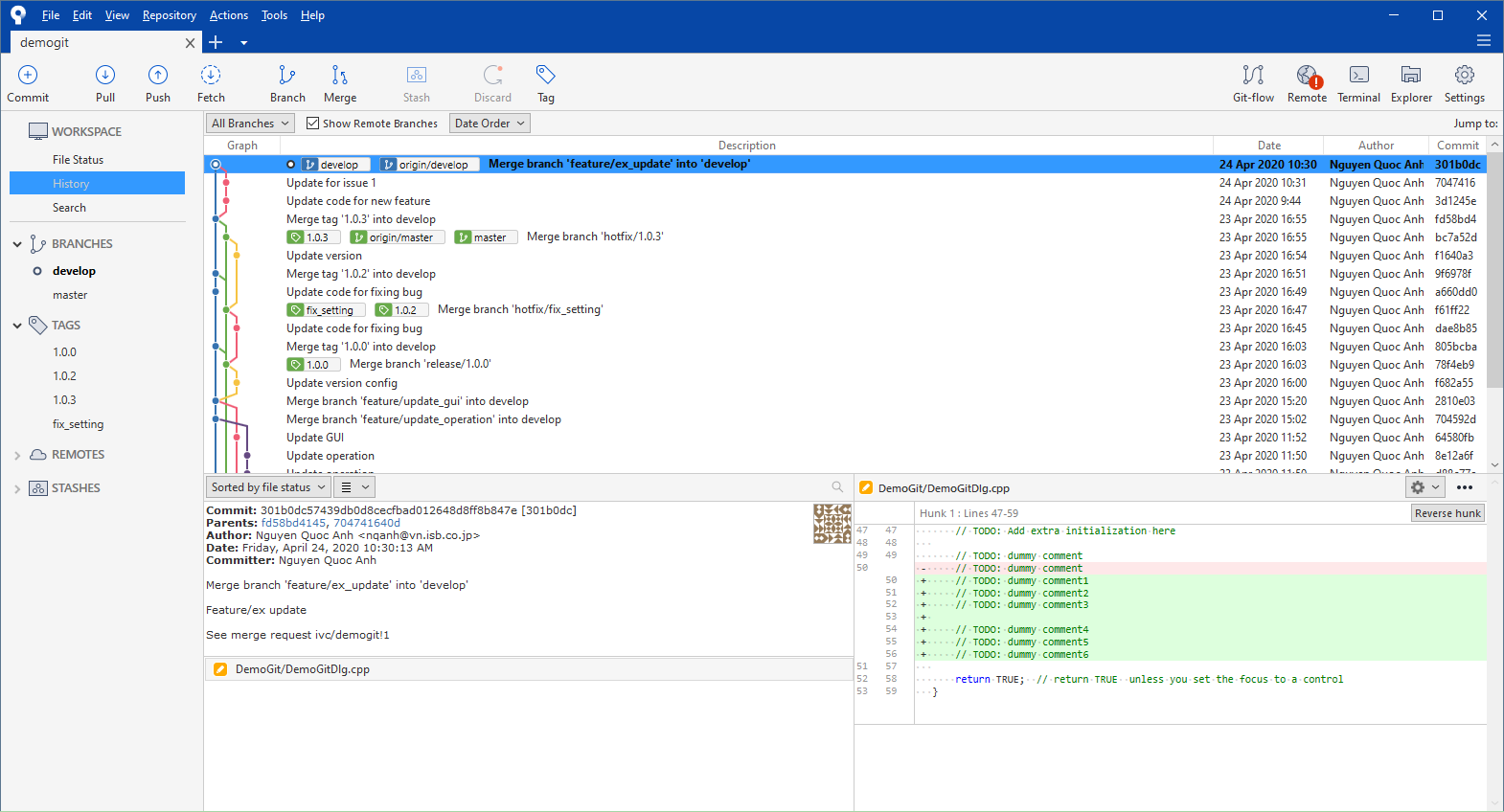


After the issues are OK, reviewer will Resolve them and select Merge

The **feature** branch will be merged to **develop** branch



* **Remember to delete the local feature branch**



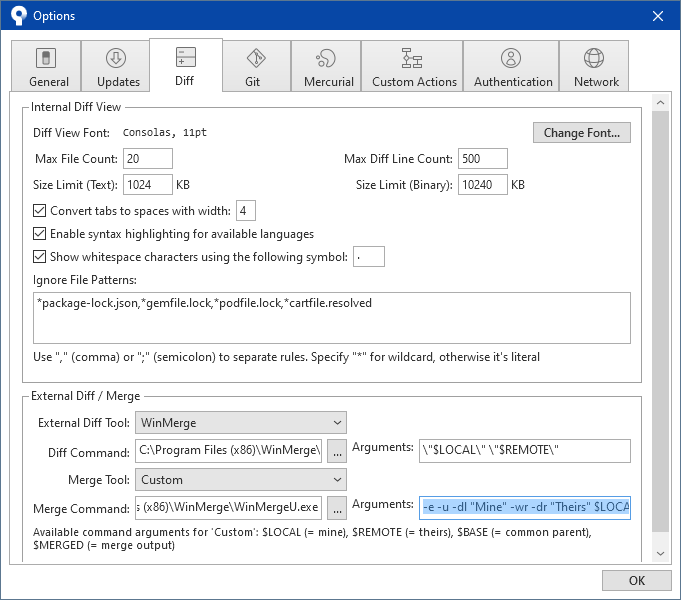
* 1. Merging conflict problem
     1. When using Source tree merge
* ***Merge Tool config (WinMerge)***

*Open the Tool 🡪 Options 🡪 Diff Tab*

*On the “External Diff Tool”: Select WinMerge*

*On the “Merge Tool”: Select Custom*

* *Merge Command: C:\Program Files (x86)\WinMerge\WinMergeU.exe*
* *Arguments: -e -u -dl "Mine" -wr -dr "Theirs" $LOCAL $REMOTE $MERGED*

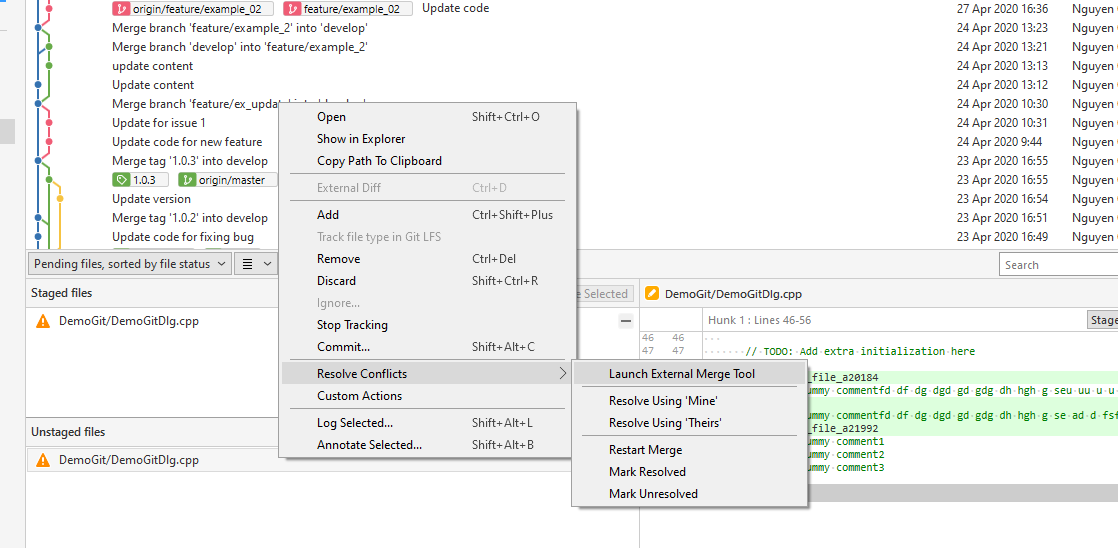


* ***Resolve steps***

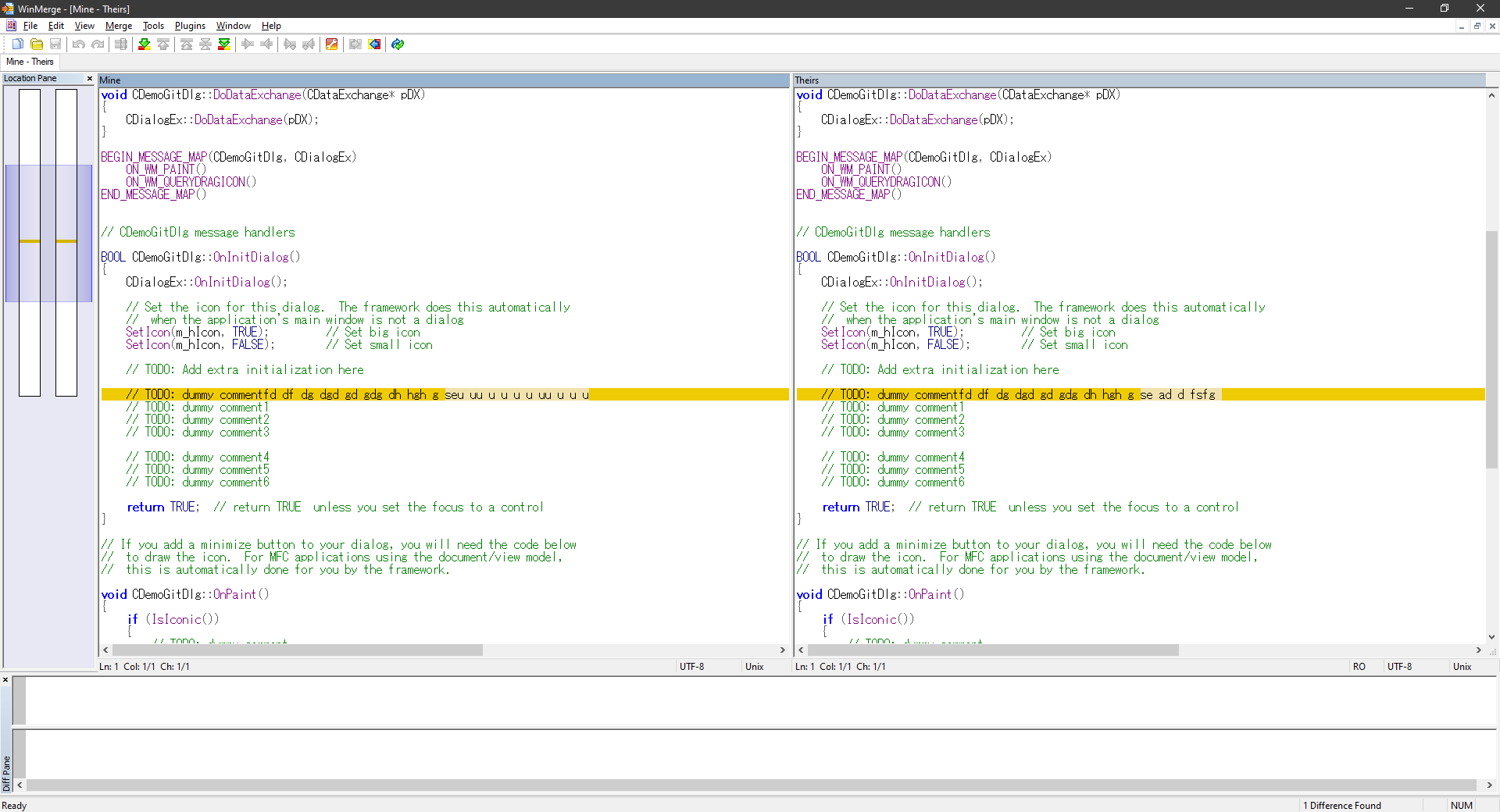
After clicking “Finish Feature”, if there is code conflict, following message will be displayed



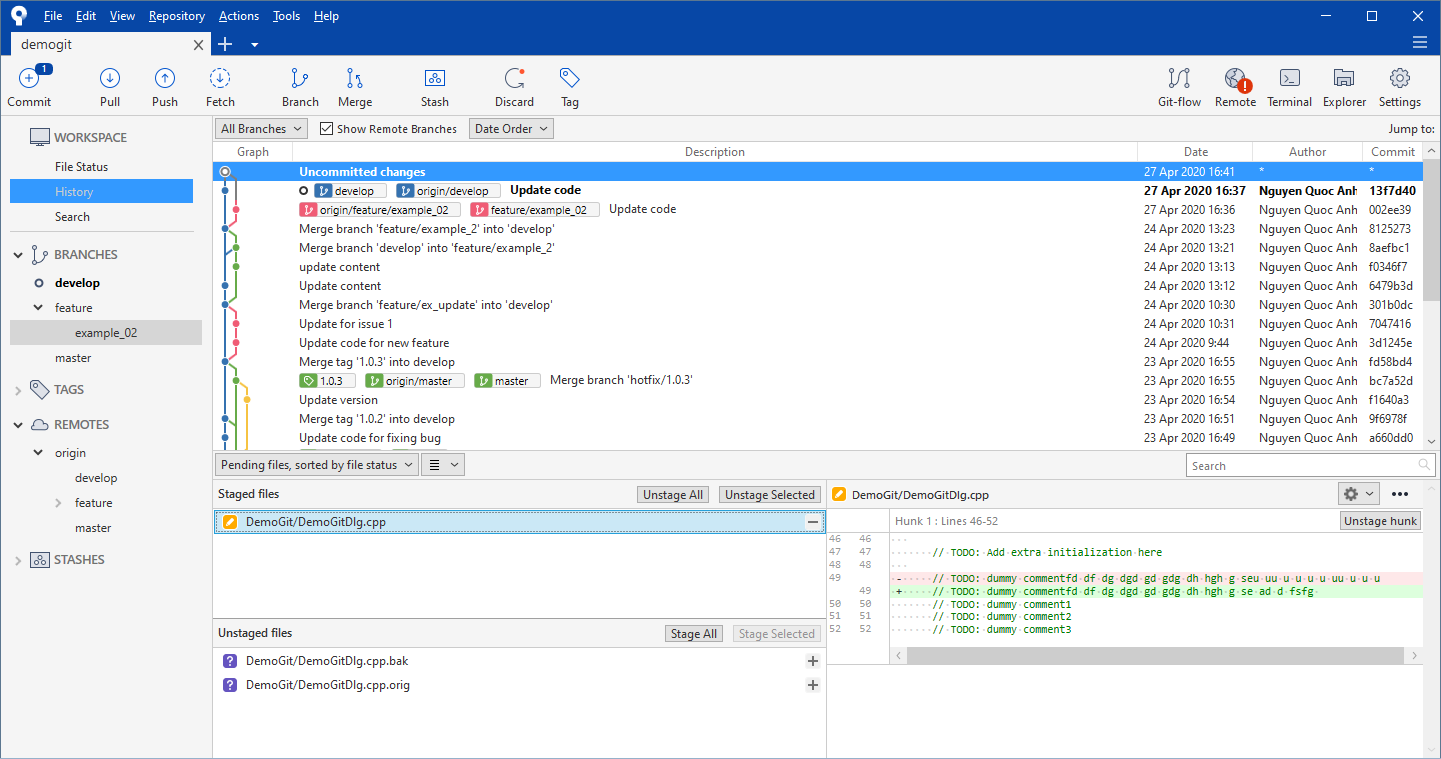
Right click on the Conflict file 🡪 Resolve Conflicts 🡪 Launch External Merge Tool



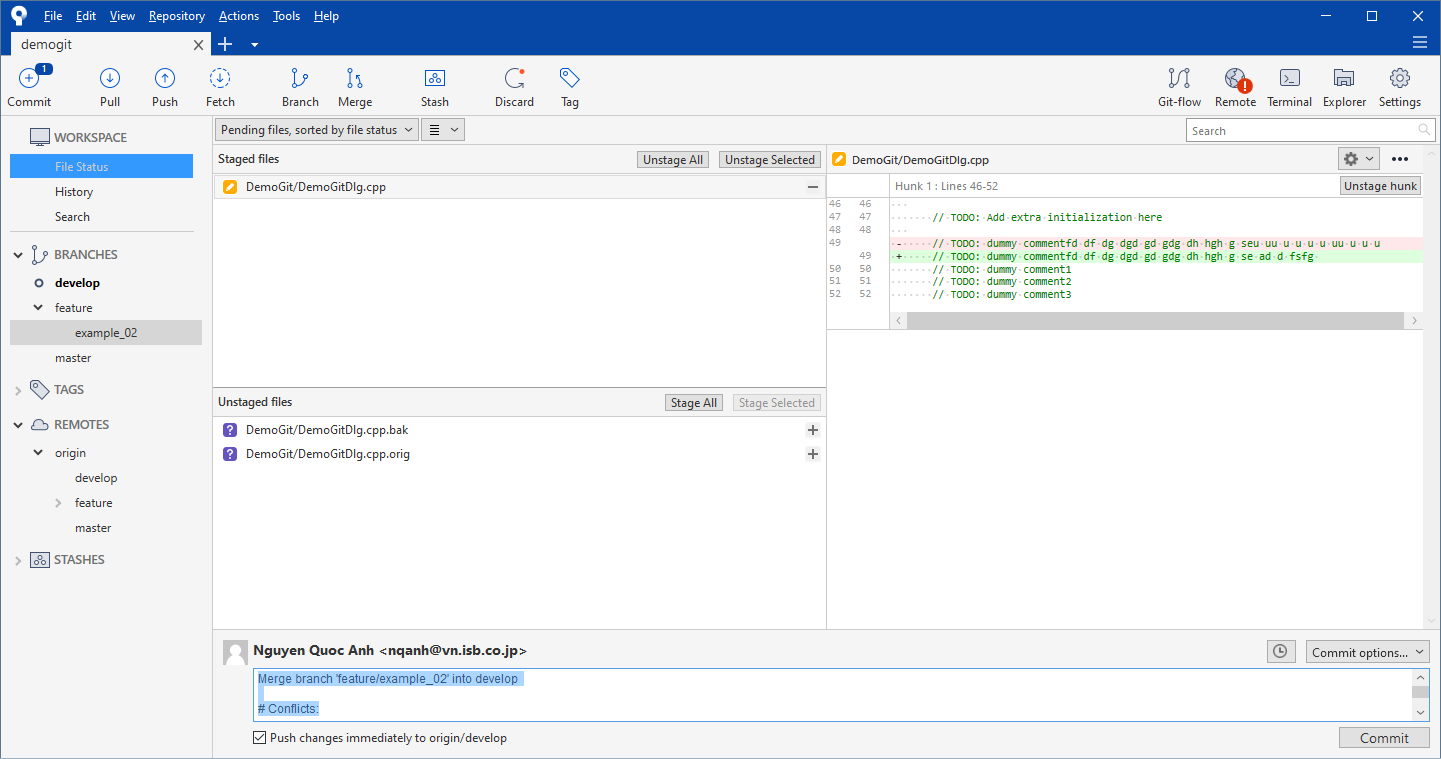
Merge Tool will display



Merge the file from Their 🡪 Mine. After finish, the file is merged

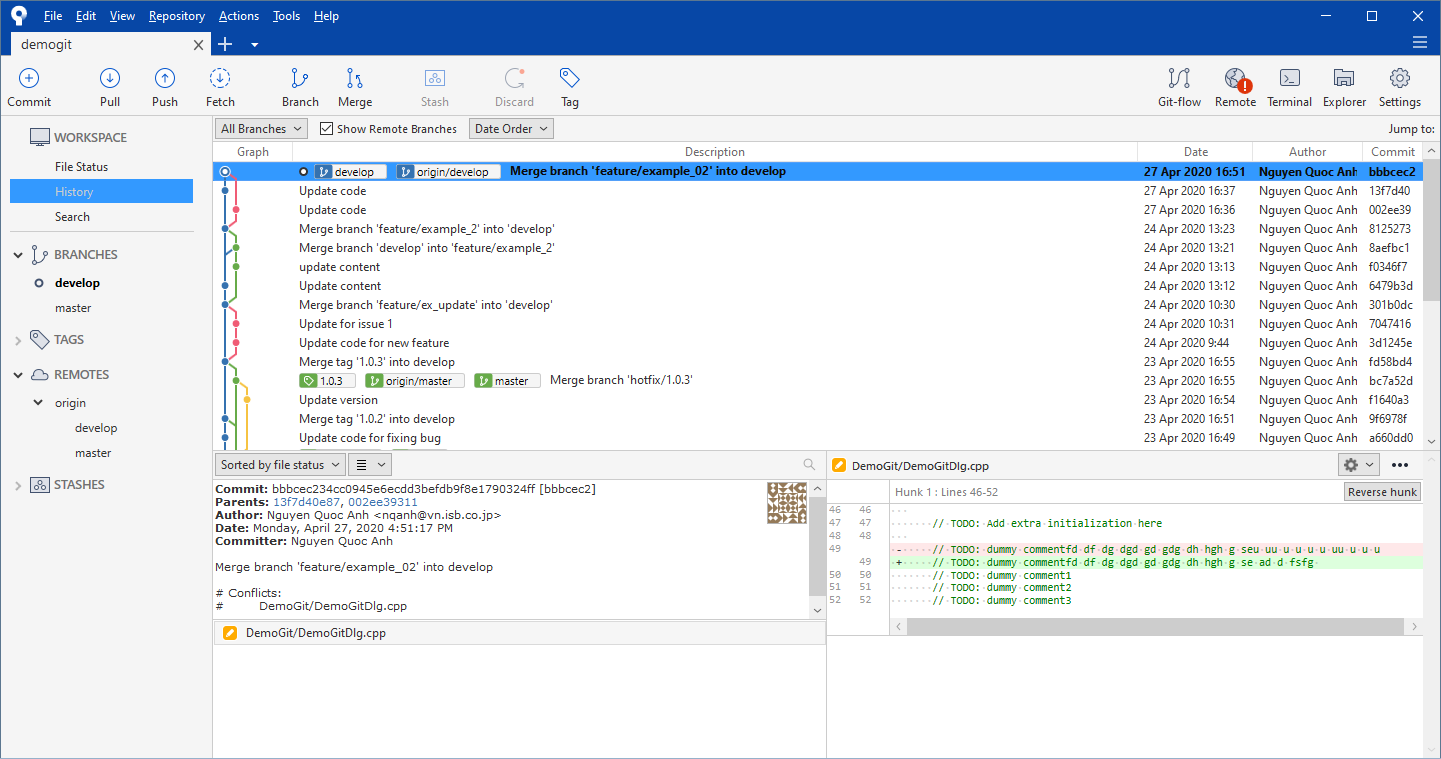


Then we can commit the changes



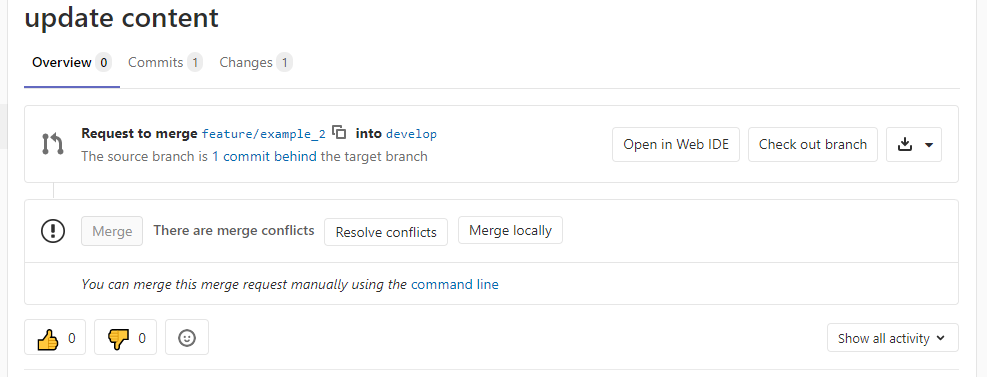
Switch to “**example\_02**” branch 🡪 Finish Feature

It will delete the **example\_02** branch on both Local and Server



* + 1. When using Merge request

When the Gitlab cannot merge the Pull request automatically, it will show the following message



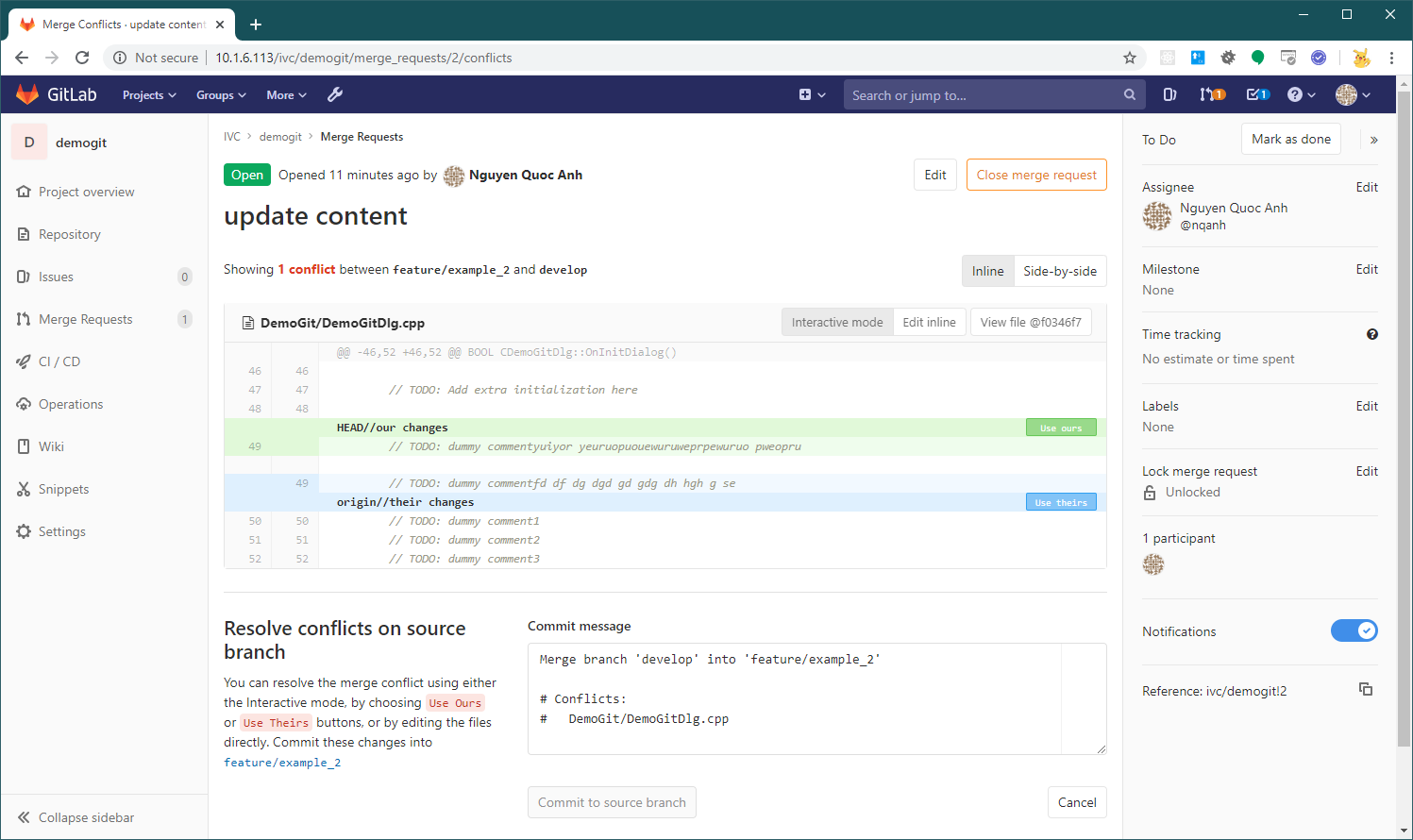
In this step, we can resolve conflict using web tool or Merge locally

“Merge locally” is same as 2.3.1 When using Source tree merge

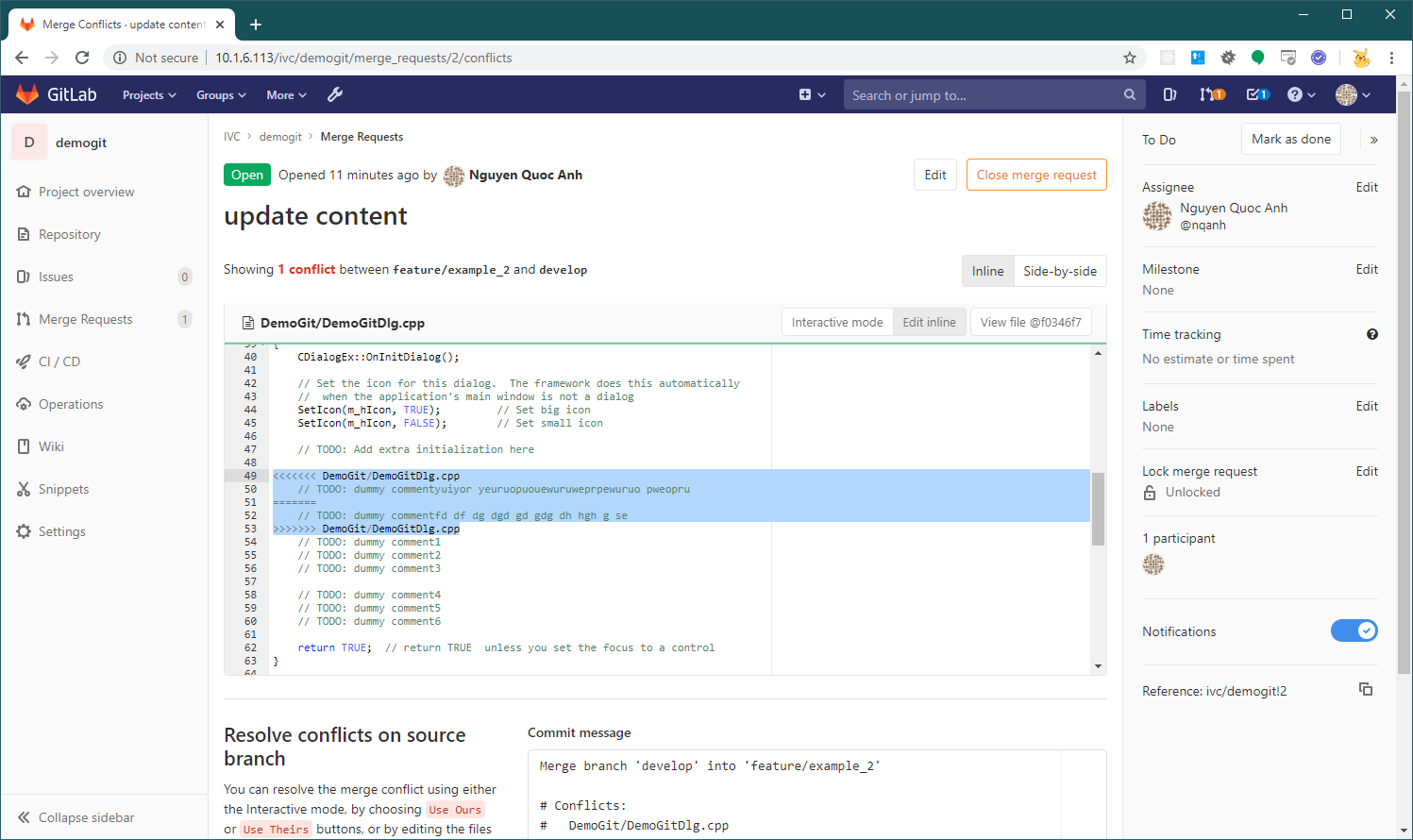
The manual below is for “Resolve conflicts” using web tool

Click on the “Resolve conflicts” button

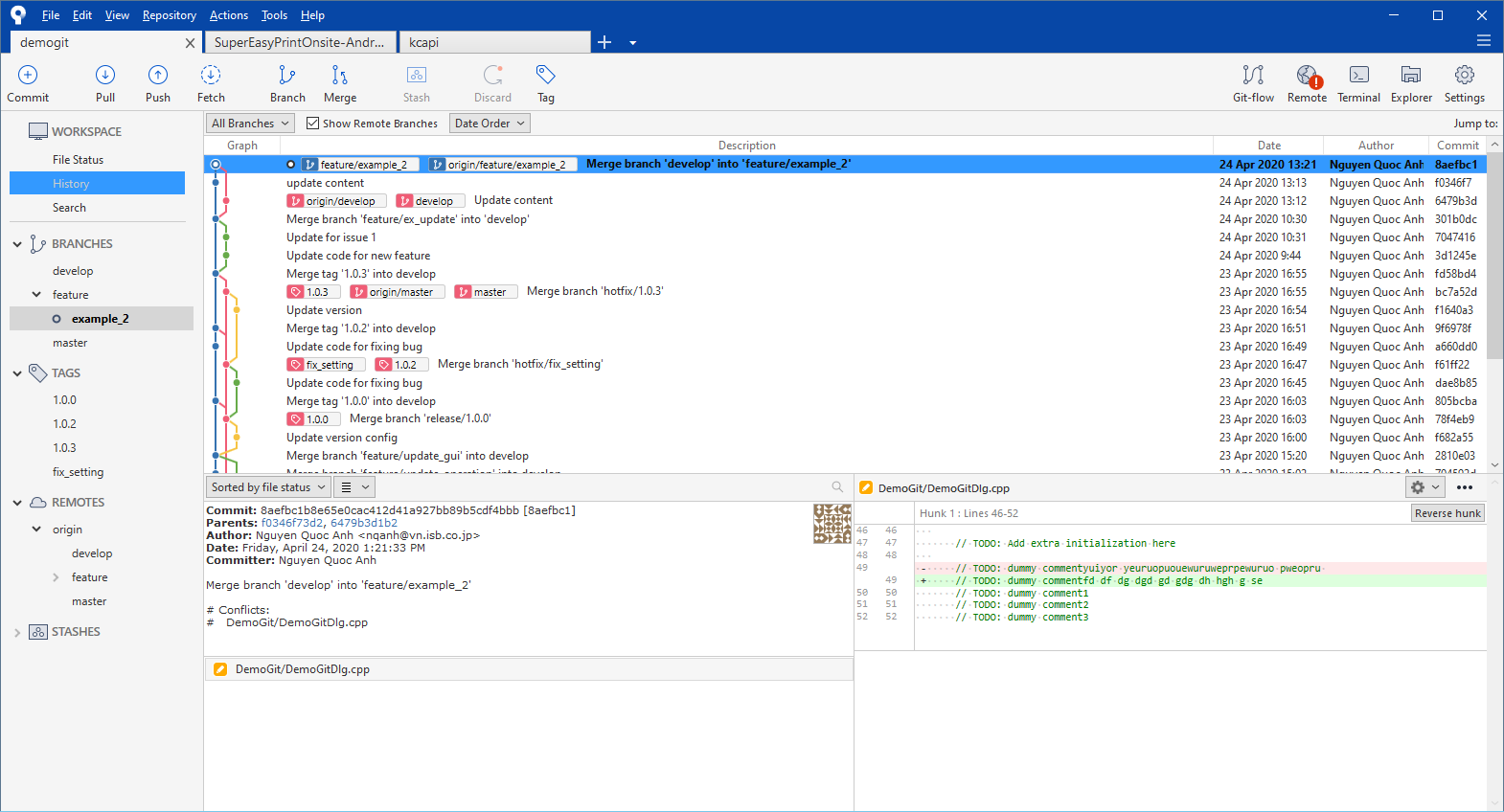
In this screen, we can compare the different and choose to code by using “Use ours” or “Use theirs” button



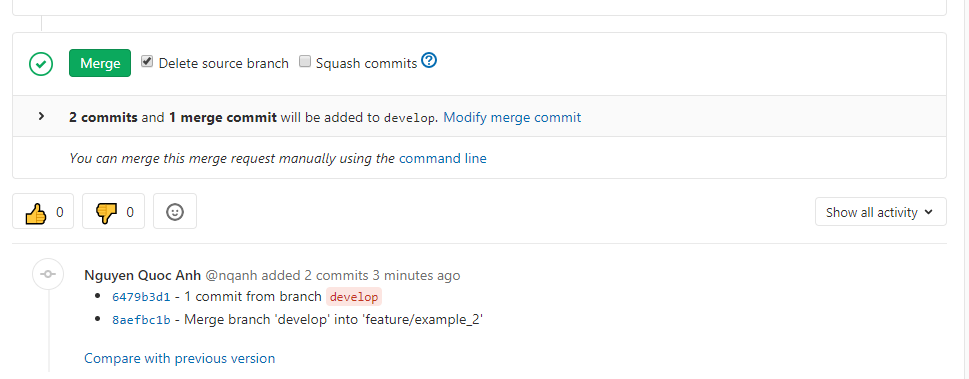
It also has an “Edit Inline” mode, that we can edit directly the source code



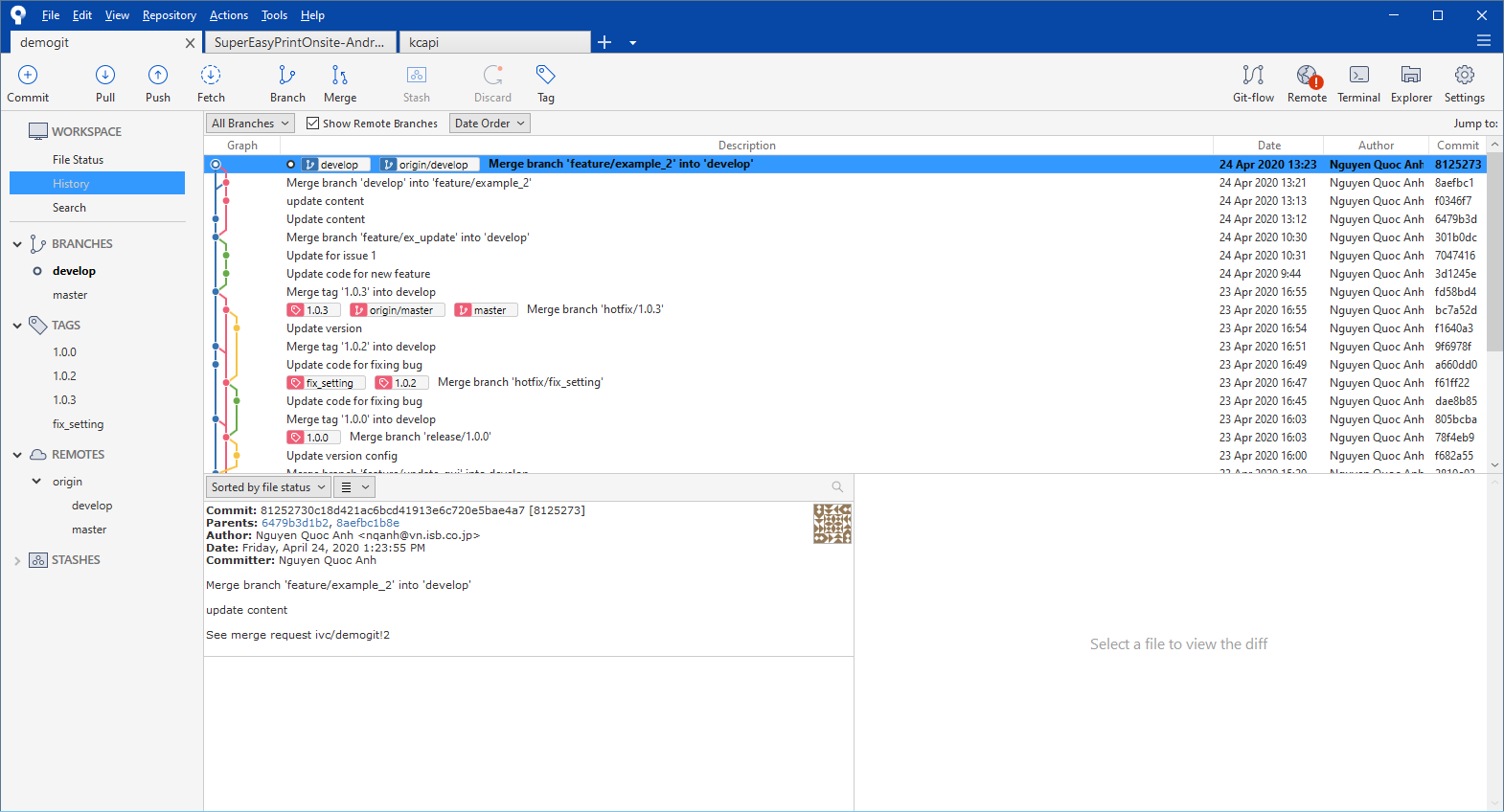
After select code, click the “Commit to source branch” button, this will create the Commit to resolve the conflict



And then we can Merge automatically



The final result



* The idea of the resolving method is: merge the **develop** branch to current **feature** branch, after that merge current **feature** branch to **develop** branch.

We can merge the **develop** branch to current **feature** branch before creating **Merge request** to avoid conflicts.