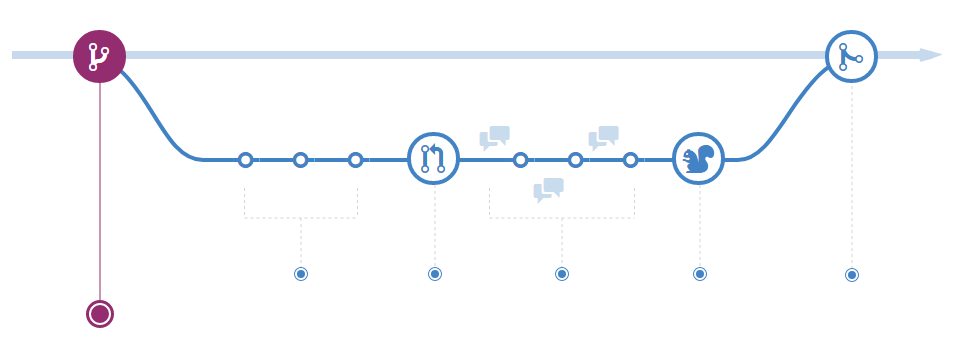
# I. Understanding Github Flow

GitHub flow is a lightweight, branch-based workflow that supports teams and projects where deployments are made regularly. This guide explains how and why GitHub flow works.

## 1. Create a branch



When you're working on a project, you're going to have a bunch of different features or ideas in progress at any given time – some of which are ready to go, and others which are not. Branching exists to help you manage this workflow.

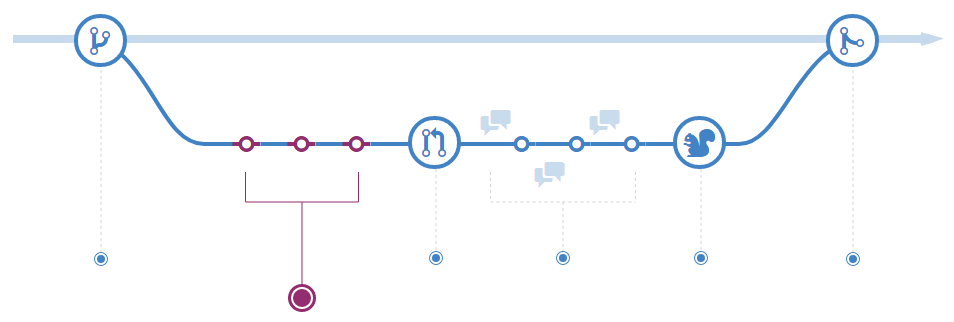
When you create a branch in your project, you're creating an environment where you can try out new ideas. Changes you make on a branch don't affect the master branch, so you're free to experiment and commit changes, safe in the knowledge that your branch won't be merged until it's ready to be reviewed by someone you're collaborating with.

***ProTip***

Branching is a core concept in Git, and the entire GitHub flow is based upon it. There's only one rule: anything in the master branch is always deployable.

Because of this, it's extremely important that your new branch is created off of master when working on a feature or a fix. Your branch name should be descriptive (e.g., refactor-authentication, user-content-cache-key, make-retina-avatars), so that others can see what is being worked on.

## 2. Add commits



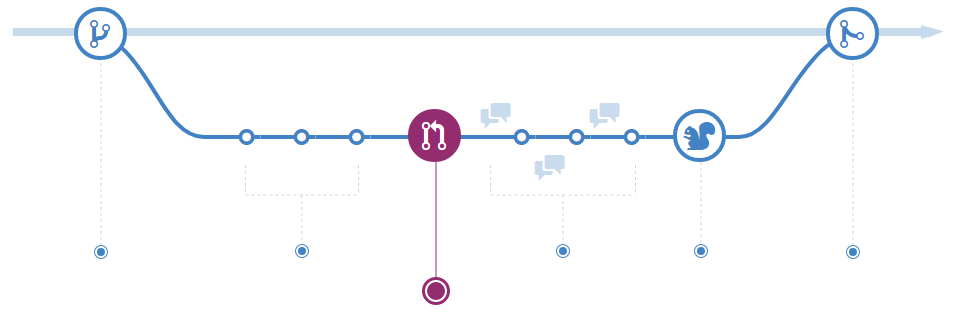
Once your branch has been created, it's time to start making changes. Whenever you add, edit, or delete a file, you're making a commit, and adding them to your branch. This process of adding commits keeps track of your progress as you work on a feature branch.

Commits also create a transparent history of your work that others can follow to understand what you've done and why. Each commit has an associated commit message, which is a description explaining why a particular change was made. Furthermore, each commit is considered a separate unit of change. This lets you roll back changes if a bug is found, or if you decide to head in a different direction.

***ProTip***

Commit messages are important, especially since Git tracks your changes and then displays them as commits once they're pushed to the server. By writing clear commit messages, you can make it easier for other people to follow along and provide feedback.

## 3. Open a Pull Request



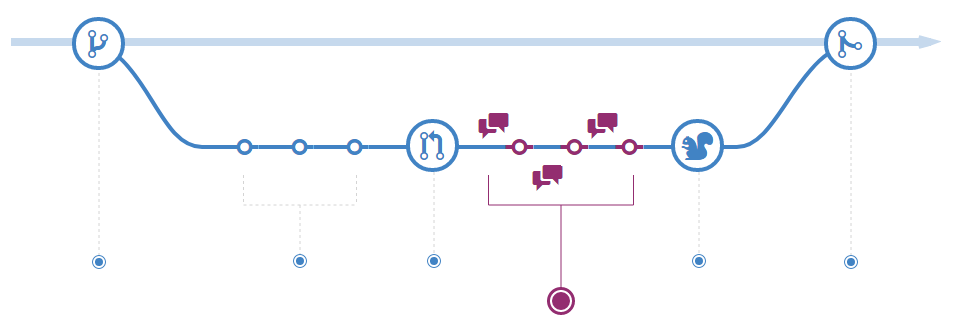
Pull Requests initiate discussion about your commits. Because they're tightly integrated with the underlying Git repository, anyone can see exactly what changes would be merged if they accept your request.

You can open a Pull Request at any point during the development process: when you have little or no code but want to share some screenshots or general ideas, when you're stuck and need help or advice, or when you're ready for someone to review your work. By using GitHub's @mention system in your Pull Request message, you can ask for feedback from specific people or teams, whether they're down the hall or ten time zones away.

***ProTip***

Pull Requests are useful for contributing to open source projects and for managing changes to shared repositories. If you're using a Fork & Pull Model, Pull Requests provide a way to notify project maintainers about the changes you'd like them to consider. If you're using a Shared Repository Model, Pull Requests help start code review and conversation about proposed changes before they're merged into the master branch.

## 4. Discuss and review your code



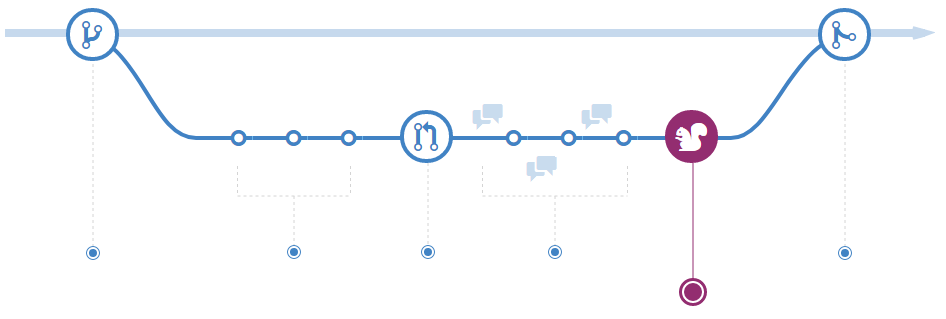
Once a Pull Request has been opened, the person or team reviewing your changes may have questions or comments. Perhaps the coding style doesn't match project guidelines, the change is missing unit tests, or maybe everything looks great and props are in order. Pull Requests are designed to encourage and capture this type of conversation.

You can also continue to push to your branch in light of discussion and feedback about your commits. If someone comments that you forgot to do something or if there is a bug in the code, you can fix it in your branch and push up the change. GitHub will show your new commits and any additional feedback you may receive in the unified Pull Request view.

***ProTip***

Pull Request comments are written in Markdown, so you can embed images and emoji, use pre-formatted text blocks, and other lightweight formatting.

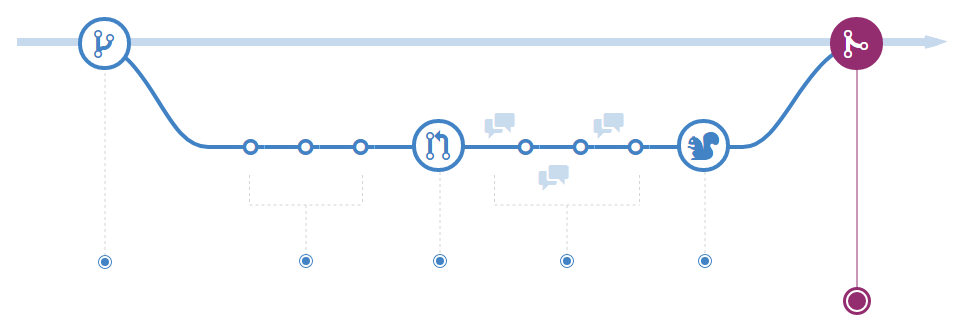
## 5. Deploy



With GitHub, you can deploy from a branch for final testing in production before merging to master.

Once your pull request has been reviewed and the branch passes your tests, you can deploy your changes to verify them in production. If your branch causes issues, you can roll it back by deploying the existing master into production.

## 6. Merge



Now that your changes have been verified in production, it is time to merge your code into the master branch.

Once merged, Pull Requests preserve a record of the historical changes to your code. Because they're searchable, they let anyone go back in time to understand why and how a decision was made.

***ProTip***

By incorporating certain keywords into the text of your Pull Request, you can associate issues with code. When your Pull Request is merged, the related issues are also closed. For example, entering the phrase Closes #32 would close issue number 32 in the repository. For more information, check out our [help article](https://help.github.com/articles/closing-issues-via-commit-messages).

# II. Understanding GitHub Project

There are 2 types of repository : Public and Private (detail in the image)

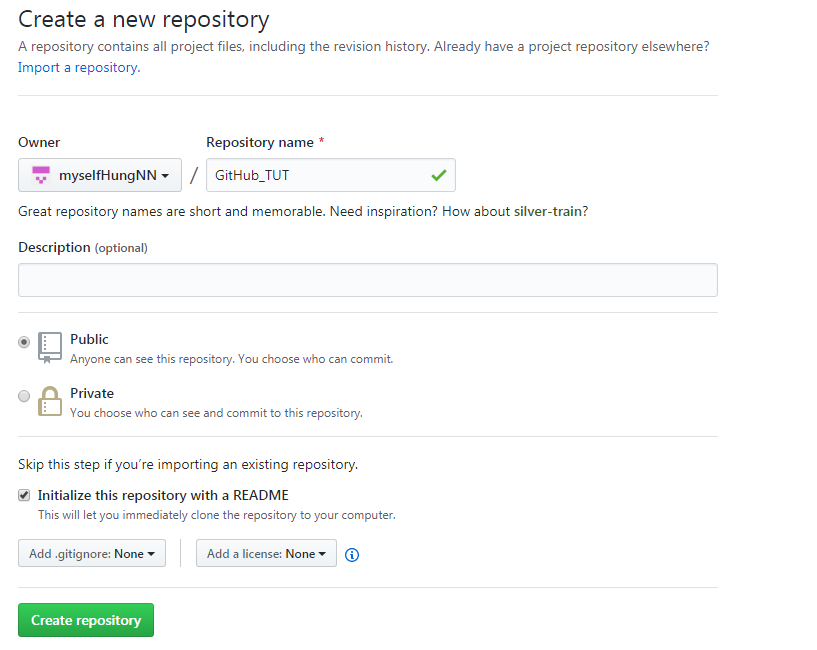
## **1. Create a new repository**

Đây là mở một repository trên nền cloud GitHub để chuẩn bị lưu online mã nguồn để quản lý phiên bản.

Trong đó, thuộc tính public: Anayone can see this repository. You choose who can commit.

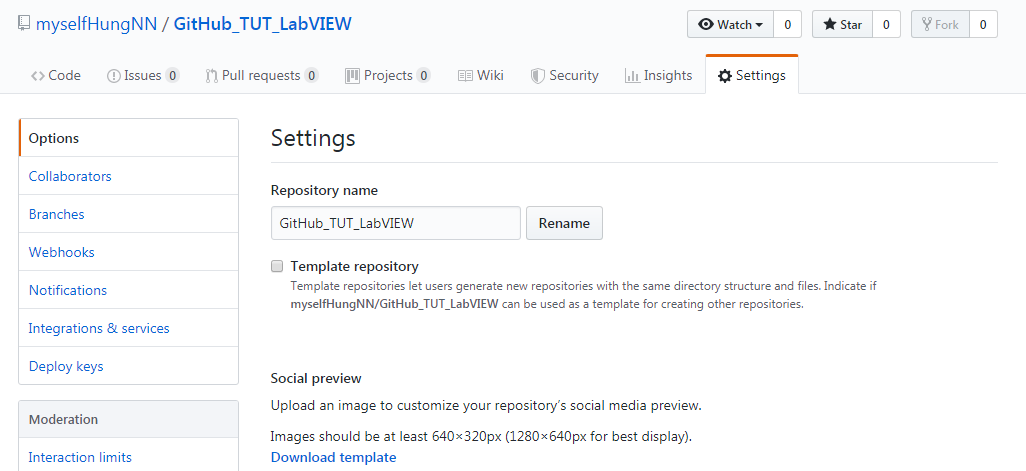
Thuộc tính private” You can choose who can see an commit to this repository

Có thể thay đổi thuộc tính public hay private trong mục Danger Zone trong Repository Settings



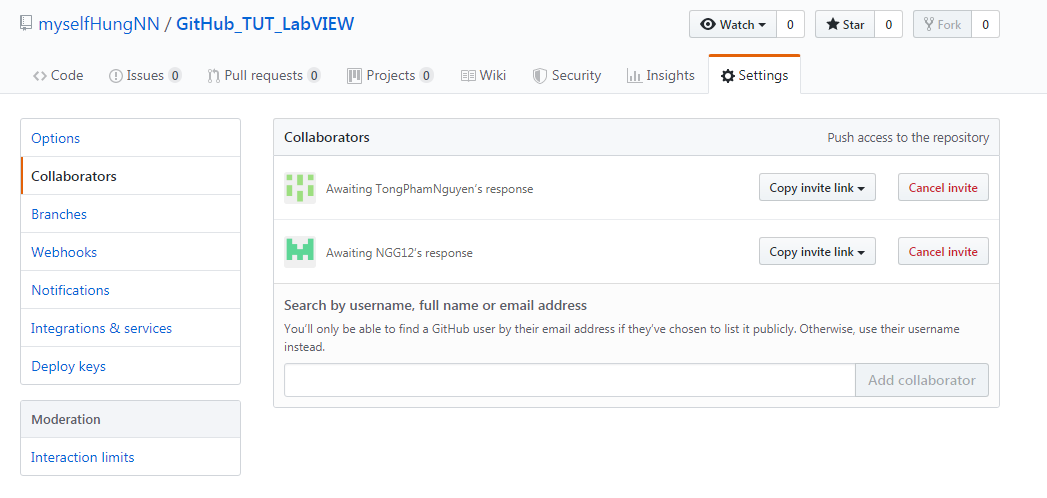
## **2. Change Repository Name**

Việc đổi tên là bình thường nếu ban đầu đặt tên không chứa thông tin một cách đầy đủ nhất



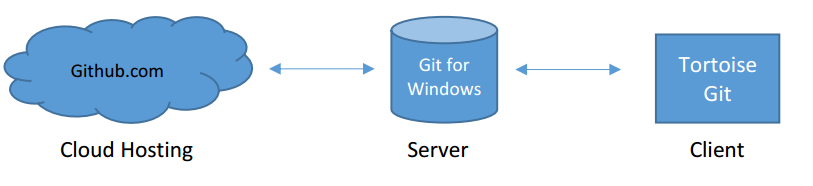
## 3. Add Colaborators

Việc này được thực hiện để có nhiều hơn 1 người có thể đóng góp Code lên Project



# III. Getting Started with Github, Tortoise Git and LabVIEW

## 1. Relationship of Github.com, GIT for Window and TortoiseGit.



Software downloads:

Git for Windows is here:

<http://www.git-scm.com/>

TortoiseGit is here:

<https://tortoisegit.org>

Ngoài ra, để thực hiện quản lý account GitHub từ máy tính có thể cài thêm GitHub desktop theo link sau: <https://help.github.com/en/desktop/getting-started-with-github-desktop/installing-github-desktop>

Ở đây, server “Git for Windows” đóng vai trò làm local server. Nghĩa là khi developer thực hiện Commit, phần mã nguồn thay đổi được lưu cục bộ trong local Server này với thông tin về Track Changes.

Khi Developer thực hiện Push thì mới đưa phần mã nguồn sau trên local Server lên GitHub.com và thực hiện lưu online trên cloud.

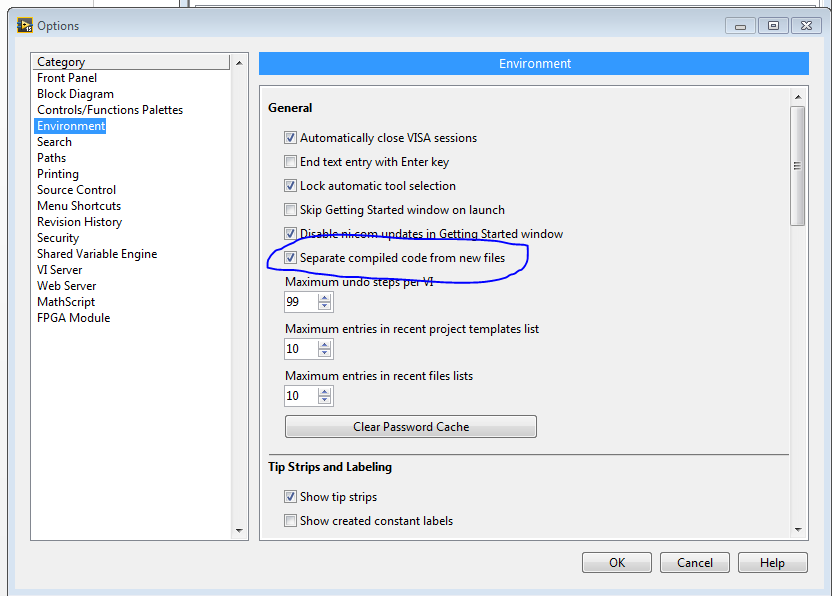
Việc thực hiện này nên thực hiện theo Commit thì thực hiện lúc nào cần quản lý thay đổi, trong khi thực hiện Push thì thực hiện sau từng khoảng thời gian cụ thể để đảm bảo được lưu trên Clound.

## 2. Prepare LabVIEW Project to commit, pust to GIT and GitHub:

Make sure you have “source only” enabled.

Điều này để đảm bảo trên GitHub chỉ lưu phần mã nguồn chứ không phải mã biên dịch, đảm bảo băng thông tiết kiệm nhất và đúng với cơ chế quản lý phiên bản mã nguồn. Thực hiện như sau:

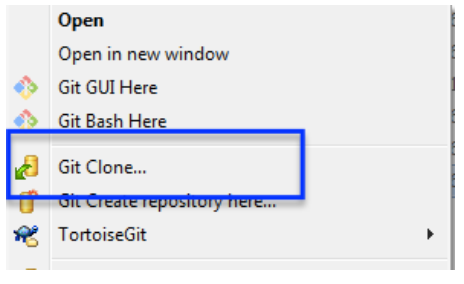
Vào Tool>>Options, chọn Enviroment, tick vào Separate compiled code from new file



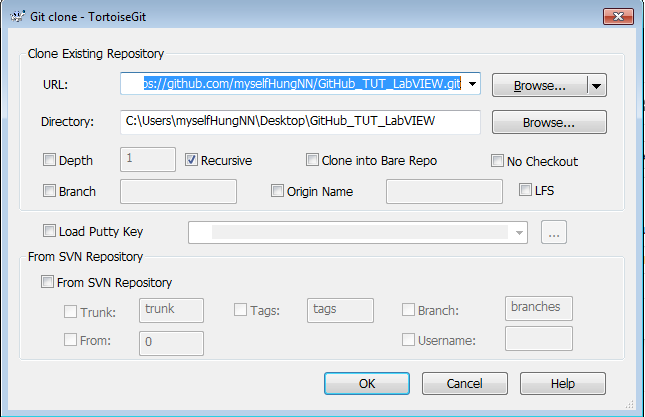
## 3. Setting up your clone of the repository that you created on Github:

Có thể thực hiện điều này trên phần mềm GitHud Desktop hoặc click phải chuột chọn Git Clone và trỏ đến repository trên GitHub.com

Sử dụng Git Clone:

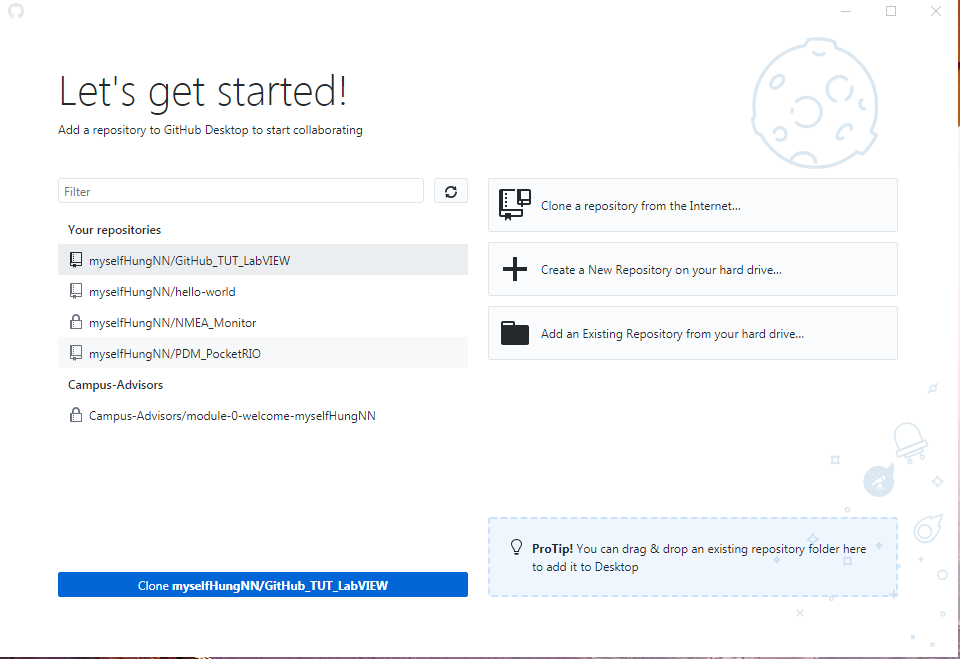


Lên giao diện:



Nhấn OK để sao lưu Clone về máy tính Local và thực hiện điều chỉnh, thay đổi

Hoặc dùng GitHub Desktop

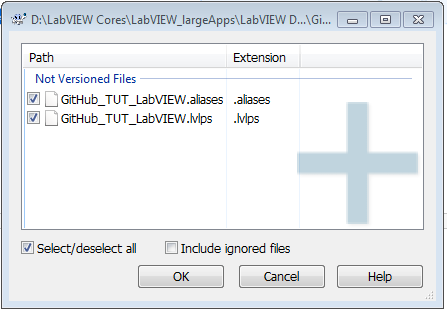


Nhần Clone để sao lưu về máy tính Local.

## 4. Make modifications and commit to your local clone of the repositor

Thực hiện thay đổi trên mã nguồn đã clone từ GitHub về máy local.

Click phải chọn Tortoisegit / Add



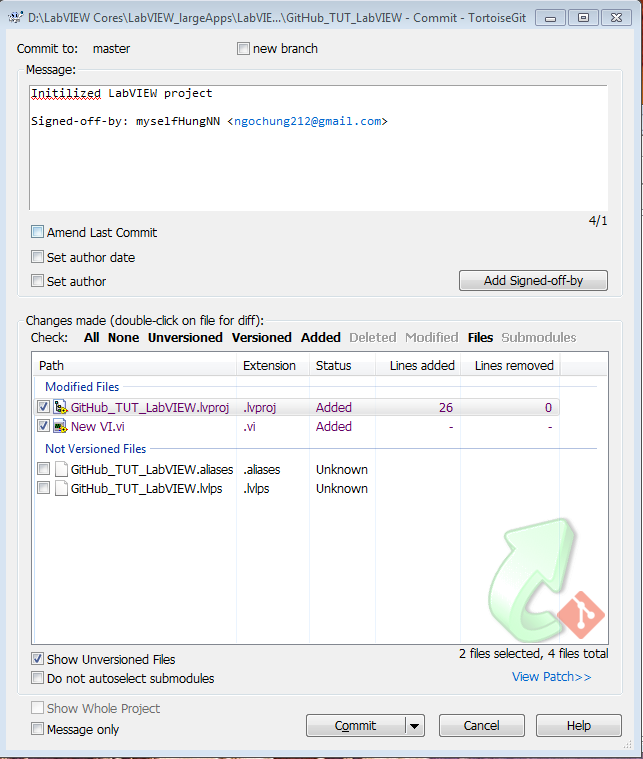
ở đây, chỉ chọn hai dạng file \*.vi và \*.lvproj và \*.ctl, \*.lvlib

các dạng file khác là dạng file được sinh ra thứ cấp do LabVIEW tạo ra khi mở Project.

Click phải chuột chọn Git Commit->Master

Và mở ra giao diện như sau.

(Nhớ rằng, ở đây vẫn chỉ update lên local server mà thôi)



## 5. Set up Tortoise Git for Graphical Diff and do a “Diff”

TortoiseGit chỉ có hàm để thực hiện các tính năng diff and merge cho dạng file văn bản. đối với các dạng file của LabVIEW , hãng NI viết sẵn các hàm để thực hiện tính năng này và chỉ có trong các phiên bản profesional từ LabVIEW 2009 trở lên.

Trong đó “Graphical Diff” is available at the following location: **C:\Program Files\National Instruments\Shared\LabVIEW Compare\LVCompare.exe.**

**và “**[Graphical Merge](http://zone.ni.com/reference/en-XX/help/371361G-01/lvhowto/configmerge_thirdparty/)” is available at **C:\Program Files\National Instruments\Shared\LabVIEW Merge\LVMerge.exe**.

Configuring TortoiseGit for Graphical Diff

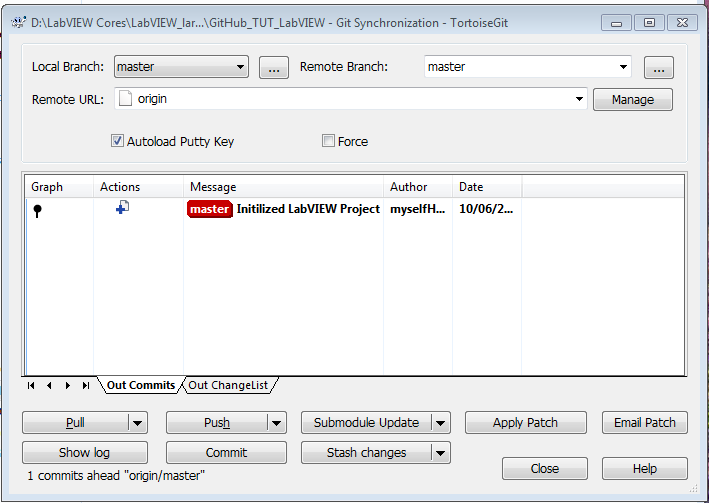
1. Right-click in Windows Explorer to expose the TortoiseGit options
2. Mouse-over 'TortoiseGit >> Settings'
3. In the list of settings on the left, select 'Diff Viewer'
4. On the right side of the dialog, select 'Advanced'
5. When the 'Advance’ dialog appears, select 'Add'
6. In the dialog that appears, type .vi as the extension
7. Where it prompts you for the path to the external program, type the following: ***"C:\Program Files (x86)\National Instruments\Shared\LabVIEW Compare\LVCompare.exe %mine %base -nobdcosm –nobdpos”***
8. Repeat this operation for the file-type \*.vit, \*.ctl, \*.ctt

Configuring TortoiseGit for Graphical Merge

1. Right-click in Windows Explorer to expose the TortoiseGit options
2. Mouse-over 'TortoiseGit>> Settings'
3. In the list of settings on the left, select 'Merge Tool'
4. On the right side of the dialog, select 'Advanced'
5. When the 'Advanced merge settings' dialog appears, select 'Add'
6. In the dialog that appears, type .vi as the extension
7. Where it prompts you for the path to the external program, type the following: ***"C:\Program Files (x86)\National Instruments\Shared\LabVIEW Merge\LVMerge.exe %base %mine %theirs %merged”***
8. Repeat this operation for the file-type \*.vit, \*.ctl, \*.ctt

## 6. Push Your Changes to Github.com

chọn Git Sync và thực hiện theo hình sau:



Click Pusth, đến đây, mọi thay đổi dược theo dõi track change và upload lên Github.com.