**TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI**

BỘ MÔN KỸ THUẬT ĐO VÀ THCN

**----o0o----**

**BÁO CÁO SINH VIÊN NGHIÊN CỨU KHOA HỌC**

**Đề tài:**

Hướng dẫn cách quản lý phiên bản sử dụng Git

**Giảng viên hướng dẫn:** PGS.TS – Nguyễn Quốc Cường

**Nhóm sinh viên thực hiện**

Họ và tên SHSV Lớp

1. Phạm Hoàng 20170761 KSTN-ĐKTĐ-K62

2. Trần Nguyễn Đức Thọ 20173389 KSTN-ĐKTĐ-K62

Hà Nội, 4/2019

*Mục lục*

[**CHƯƠNG 1. TÌM HIỂU VỀ GIT VÀ CÀI ĐẶT GIT.** 3](#_Toc7950793)

[1.1. Git là gì? 3](#_Toc7950794)

[1.2. Hệ thống quản lý phiên bản với Git gồm những gì? 4](#_Toc7950795)

[1.2.1. Repository (kho) 4](#_Toc7950796)

[1.2.2. Chức năng Commit (ký thác) 4](#_Toc7950797)

[1.2.3. Working Tree và Index 5](#_Toc7950798)

[1.2.4. Branch 6](#_Toc7950799)

[1.3. Cài đặt GIT 6](#_Toc7950800)

[**CHƯƠNG 2. SỬ DỤNG GIT ĐỂ QUẢN LÝ PHIÊN BẢN PHẦN MỀM TẠI MÁY TÍNH LOCAL.** 7](#_Toc7950801)

[2.1. Thiết lập thông tin người dùng 7](#_Toc7950802)

[2.2. Tạo repository mới: 8](#_Toc7950803)

[2.3. Commit: 8](#_Toc7950804)

[2.4. Làm việc với Branch 11](#_Toc7950805)

[2.4.1. Tạo một Branch mới: 11](#_Toc7950806)

[2.4.2. Kiểm tra danh sách branch thực hiện: 11](#_Toc7950807)

[2.4.3. Chuyển đổi Branch: 11](#_Toc7950808)

[2.4.4. Merge Branch 12](#_Toc7950809)

[2.4.5. Làm việc song song với nhiều Branch 14](#_Toc7950810)

[2.4.6. Rebase: 16](#_Toc7950811)

[**CHƯƠNG 3. SỬ DỤNG GIT VỚI SEVER GITHUB** 18](#_Toc7950812)

[3.1. Tạo tài khoản GitHub: 18](#_Toc7950813)

[3.2. Tạo Remote Repository: 18](#_Toc7950814)

[3.3. Push lên Remote Repository: 20](#_Toc7950815)

[3.4. Pull từ Repository 22](#_Toc7950816)

[3.5. Clone một Repository: 23](#_Toc7950817)

[3.6. Push từ Repository đã Clone: 24](#_Toc7950818)

[3.7. Giải quyết xung đột lúc Push. 25](#_Toc7950819)

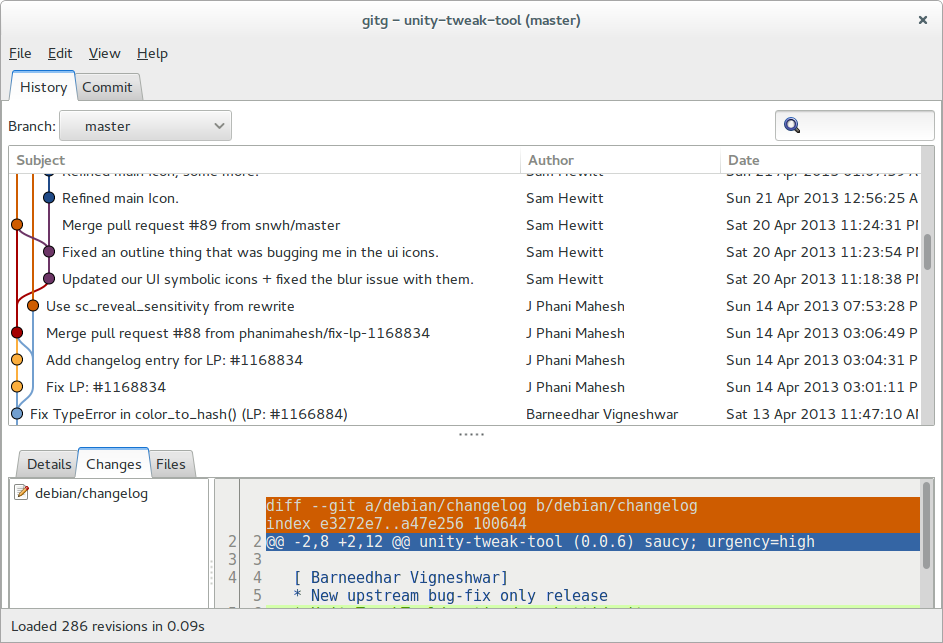
[**Chương 4. Sử dụng Git GUI** 29](#_Toc7950820)

# **CHƯƠNG 1. TÌM HIỂU VỀ GIT VÀ CÀI ĐẶT GIT.**

## 1.1. Git là gì?

Git là một trong những Hệ thống Quản lý Phiên bản Phân tán (còn gọi là Distributed Version Control Systems) , được phát triển nhằm quản lý mã nguồn (source code) hữu hiệu của Linux.

Trên Git, mọi thay đổi của file đều được lưu lại dưới dạng một lists các lịch sử cập nhật. Vì thế chúng ta có thể truy cập đến bất kỳ trạng thái thay đổi nào của file đó, đồng thời nó cũng sẽ hiển thị những thay đổi so với những phiên bản khác.

*Mọi lịch sử thay đổi đều sẽ được ghi lại cụ thể trong Git*

## 1.2. Hệ thống quản lý phiên bản với Git gồm những gì?

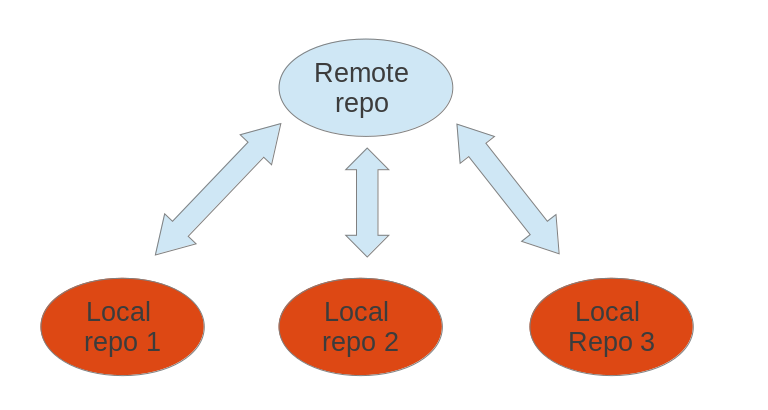
### 1.2.1. Repository (kho)

Repository là nơi sẽ ghi lại trạng thái của thư mục và file (lịch sử thay đổi, nội dung thay đổi, …). Repository của Git được chia làm hai loại:

* Remote repository: Là repository để chia sẻ giữa nhiều người và bố trí trên server chuyên dụng.
* Local repository: Là repository bố trí trên máy tính cá nhân, chỉ dành cho một người dùng sử dụng.

Với những công việc bình thường thì có thể sử dụng local repository để thực hiện trên máy tính cá nhân của mình.

Khi muốn công khai nội dung công việc mà bản thân đã làm trên local repository, thì sẽ upload lên remote repository rồi công khai. Thêm nữa, thông qua remote repository cũng có thể lấy về nội dung công việc của người khác về máy mình để làm việc.

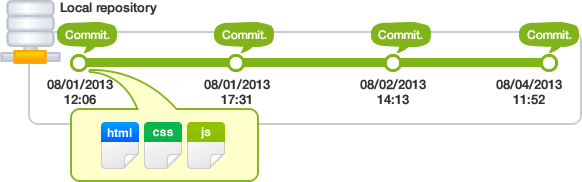


*Sự tương tác giữa remote repository và local repository*

1.2.2. Chức năng Commit (ký thác)

Để xác nhận việc thêm/ thay đổi file hay thư mục vào repository thì sẽ thực hiện thao tác gọi là Commit (ký thác/xác nhận).

Khi thực hiện commit, trong repository sẽ tạo ra commit (hoặc revision) đã ghi lại sự khác biệt từ trạng thái đã commit lần trước đến trạng thái hiện tại.

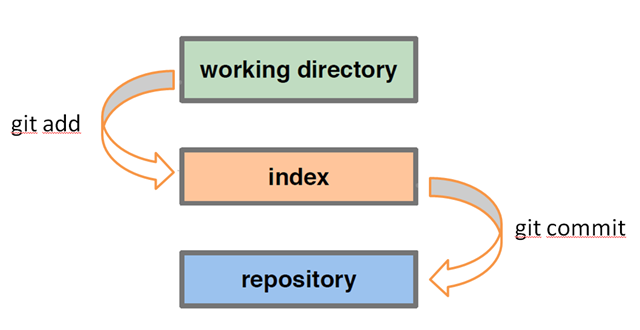


*Lịch sử những lần Commit được ghi lại trong Git*

1.2.3. Working Tree và Index

Trên Git, những thư mục được đặt trong sự quản lý của Git mà mọi người đang thực hiện công việc trong thực tế được gọi là working tree.

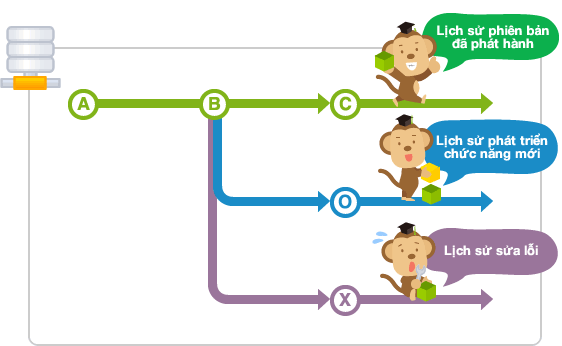
Ngoài ra, giữa repository và working tree tồn tại một khu vực gọi là index. Index là nơi để chuẩn bị cho việc commit lên repository.



*Trước khi Commit, phải Add file vào Index thì mới có thể thực hiện Commit được!*

1.2.4. Branch

Branch được hiểu là một cái dùng để phân nhánh và ghi lại luồng của lịch sử. Branch đã phân nhánh sẽ không ảnh hưởng đến branch khác nên có thể tiến hành nhiều thay đổi đồng thời trong cùng 1 repository.

Hơn nữa, branch đã phân nhánh có thể chỉnh sửa tổng hợp lại thành 1 branch bằng việc hợp lại (merge) với branch khác.

*Cách thức làm việc với Branch để phát triển phần mềm*

1.3. Cài đặt GIT

* Đối với Ubuntu:

Khởi động Terminal (Ctrl+Alt+T).

Sử dụng lệnh: $ **sudo apt-get install git**

* Đối với Windows: Tiến hành Download trên trang chủ của git:

<https://git-scm.com/>, chạy file exe rồi cài đặt.

Git sẽ được cài đặt với Git Bash và giao diện Git GUI.

Để có thể thao tác bằng Console (sử dụng Git Bash) như trên Ubuntu, các bạn cần phải cài đặt thêm trình biên dịch cho Windows, download tại đây: <http://www.mingw.org/>, và tiến hành cài đặt như bình thường.

* Kiểm tra Git đã được cài đặt hay chưa bằng cách: $ **git --version**

# **CHƯƠNG 2. SỬ DỤNG GIT ĐỂ QUẢN LÝ PHIÊN BẢN PHẦN MỀM TẠI MÁY TÍNH LOCAL.**

## 2.1. Thiết lập thông tin người dùng

* Các bạn sử dụng các lệnh sau:

$ **git config --global user.name *"<Tên người dùng>"***

$ **git config --global user.email "<Địa chỉ mail>"**

* Thiết lập SSH Key:

Trong Git, nếu bạn sử dụng giao thức HTTPS để truyền tải dữ liệu giữa server và client thì mỗi lần bạn Pull hay Push đều cần phải nhập lại email và mật khẩu GitHub (phần làm việc với Sever GitHub chúng ta sẽ nói ở phần sau, tuy nhiên ở đây chỉ hướng dẫn cách thiết lập để về sau việc sử dụng được dễ dàng và thuận tiện hơn), vì thế hãy dùng giao thức SSH như sau:

Khởi động Terminal rồi chạy lệnh:

**ssh-keygen -t rsa -C "**[**your\_email@gmail.com**](mailto:youremail@gmail.com)**"**

Câu lệnh trên sẽ tạo ra một public key và một private key. Hãy chia sẻ public key với GitHub bằng cách:

* Truy cập đến đường dẫn: Home/.ssh, mở file “id\_rsa.pub” rồi copy key.
* Đăng nhập vào GitHub -> Setting -> [SSH and GPG keys](https://github.com/settings/keys) -> [Add new SSH key](https://github.com/settings/ssh/new), điền key vừa copy rồi save lại!
* Tạo màu cho phần xuất ra của git thì sử dụng:

$ **git config --global color.ui auto**

2.2. Tạo repository mới:

Ở đây chúng ta sẽ tạo một thư mục có tên là “repo”, sau đó sẽ tạo một repository ở trong thư mục đó. Thực hiện lần lượt các lệnh sau:

$ **mkdir repo**

$ **cd repo**

$ **git init**

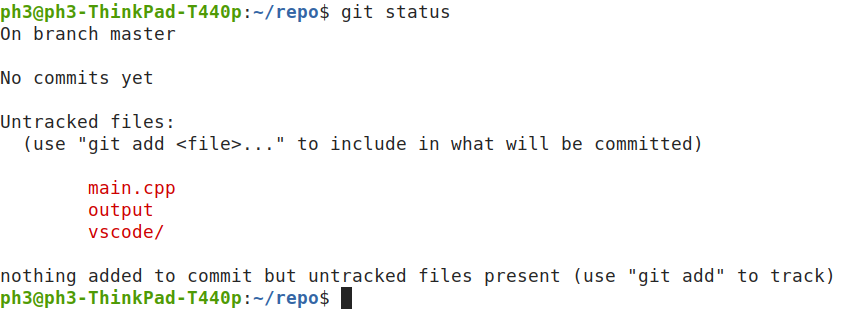
Tạo repository thành công sẽ có thông báo:

Initialized empty Git repository in /home/\*/repo/.git/

2.3. Commit:

Trước hết, hãy chuyển toàn bộ file code của bạn vào thư mục “repo” vừa được tạo (có thể là file, có thể là các thư mục)

Thực hiện: $ **git status** để kiểm tra.



*Ví dụ ở đây chúng ta có thư mục vscode và một số file nhỏ khác*

* Thông báo “No commits yet” cho biết chúng ta chưa thực hiện commit nào.
* “Untracked files” là những file mà chúng ta chưa đánh dấu (tức là chưa thêm vào Index) - thông báo màu đỏ.

Để thực hiện thêm vào những file mà chúng ta cần làm việc vào Index thì thực hiện: $ **git add -A**

Kiểm tra lại một lần nữa: $ **git status**

*Lúc này tất cả các file và thư mục đều đã được thêm vào Index để Commit.*

Để commit chúng ta dùng lệnh: $ **git commit -m “first commit”**

*Thông báo thực hiện commit thành công!*

Kiểm tra tiếp bằng lệnh: $ **git status** chúng ta nhận được thông báo:

On branch master

nothing to commit, working tree clean

* Kiểm tra lịch sử commit: $ **git log**

**Note:** Các bạn hoàn toàn có thể tự tạo trên máy của mình một repository với nhiều thư mục, nhiều file, rồi thực hiện commit như bình thường.

Trường hợp chỉ làm việc với một file duy nhất thì sử dụng: $ **git add** <file>

## 2.4. Làm việc với Branch

### 2.4.1. Tạo một Branch mới:

$ **git branch “demo”**

Vẫn ở thư mục repo trước, ở đây chúng ta tạo một branch mới tên là “demo”.

Để xóa branch chúng ta dùng lệnh: $ **git branch -d “tên branch”**

2.4.2. Kiểm tra danh sách branch thực hiện:

$ **git branch**

Ở đây chúng ta sẽ thấy danh sách hai branch (master và demo). Dấu \* thể hiện việc chúng ta đang làm việc trên branch master (hay còn gọi là branch chính).

demo

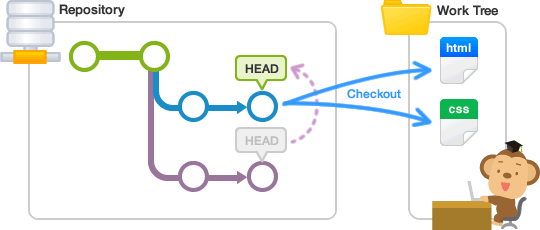
\* master

2.4.3. Chuyển đổi Branch:

Để chuyển đổi branch làm việc thì sẽ thực hiện thao tác gọi là checkout.

$ git **checkout “demo”**

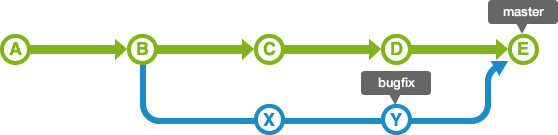
Khi thực hiện checkout, mọi thao tác thay đổi, chỉnh sửa và commit trong branch “demo” sẽ hoàn toàn độc lập mà không ảnh hưởng gì đến branch master.

  
**Note:** HEAD hiển thị phần đầu branch của chúng ta đang làm việc. Mặc định là master. Bằng việc di chuyển HEAD thì branch cũng sẽ được chuyển (checkout).

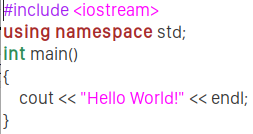
2.4.4. Merge Branch

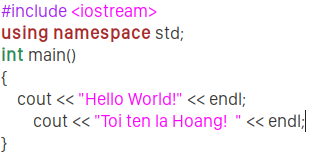
Khi sử dụng merge, có thể tổng hợp nhiều luồng lịch sử.

Merge commit lấy thay đổi của cả hai branch sẽ được tạo ra. Đầu branch master sẽ di chuyển đến commit đó.



**Thí dụ:**

* Các bạn hãy tự tạo trong repository của mình một file \*.txt và chúng ta sẽ bắt đầu làm việc với branch để hiểu rõ hơn về nó. Ở đây tôi đã có một file main.cpp với nội dung như bên dưới, file này mặc định sẽ ở branch master.
*  Chúng ta hãy checkout qua branch demo bằng lệnh: $ **git checkout demo**
* Thực hiện sửa đổi trên file này bằng cách thêm một vài dòng đơn giản, chẳng hạn:



* Sau đó **add** và **commit** để xác nhận, lúc này nội dung file main.cpp ở branch demo và branch master sẽ hoàn toàn khác nhau. Bạn có thể kiểm tra bằng cách : $ **git checkout master** để xem nội dung.
* Gộp hai branch vào branch master bằng lệnh:

$ **git checkout master**

**$ git merge demo**

Updating 30a88ea..6c1f624

Fast-forward

vscode/main.cpp | 4 +++-

1 file changed, 3 insertions(+), 1 deletion(-)

* Lúc này việc gộp (merge) branch đã thành công! Các bạn hãy mở file lên để kiểm tra, nội dung hoàn toàn giống nhau ở cả hai branch (bạn đọc thực hiện checkout để thấy rõ điều này).

2.4.5. Làm việc song song với nhiều Branch

Ở trên là thí dụ làm việc với hai branch khác nhau. Vậy nếu có ba, bốn, …. nhiều branch thì sẽ thế nào?

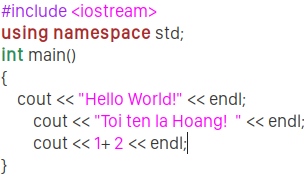
Để thử nghiệm chúng ta sẽ tạo thêm một branch nữa có tên là “demo2”:

$ **git branch demo2**

Tiếp theo, checkout qua branch demo2 rồi mở file code của bạn lên, thêm vào một vài dòng đơn giản, ở đây tôi chỉnh sửa như sau:

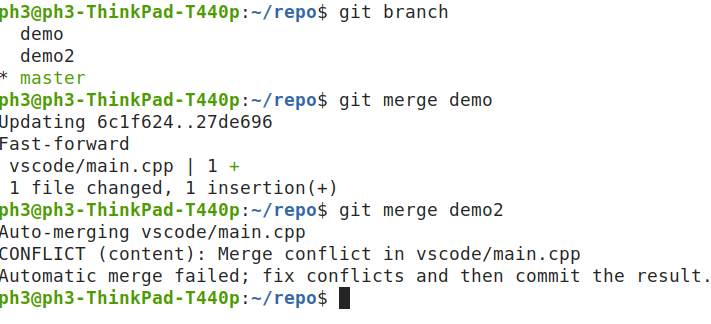
Cuối cùng, luôn nhớ khi thực hiện chỉnh sửa, hãy add và commit để xác nhận nhé!

Tiếp tục checkout qua branch demo để chỉnh sửa, chẳng hạn:

Cuối cùng là save và commit!

Như vậy bây giờ chúng ta đã có 3 branch khác nhau (master, demo, demo2), cách thức làm việc song song này sẽ giúp những thành viên trong team làm việc độc lập với nhau mà không hề ảnh hưởng đến nhau. Vậy đến khi muốn gộp branch để cho ra phiên bản cuối cùng thì sẽ như thế nào? Hãy thử xem nhé!

* Trước tiên, checkout qua branch master: $ **git checkout master**
* Merge branch “demo” vào branch master: $ **git merge demo**, tiếp tục merge branch “demo2”: $ **git merge demo2**

Chúng ta có thể thấy, ở đây đã báo lỗi xung đột (do nột dung khác nhau), lúc này chúng ta phải mở file code lên, chỉnh sửa bằng tay và commit lại!

Xóa những kí tự thừa đi, giữ lại những sửa đổi rồi Save → add file → commit!

→ Thông báo thành công, các bạn hãy mở file của mình lên và kiểm tra!

**Note:** Các bạn hoàn toàn có thể mở rộng với nhiều tệp, nhiều thư mục khác nhau.

2.4.6. Rebase:

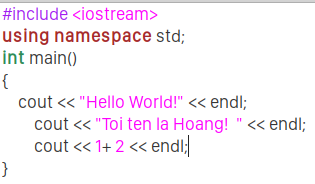
Rebase có chức năng tương tự merge. Lịch sử sẽ trở nên đơn giản nhưng nội dung thay đổi từ commit ban đầu sẽ bị thay đổi. Cho nên cũng có trường hợp rơi vào tình trạng commit gốc không hoạt động.



Để hiểu rõ hơn về Rebase, chúng ta hãy xóa lần merge trước bằng lệnh:

$ **git reset --hard HEAD~**

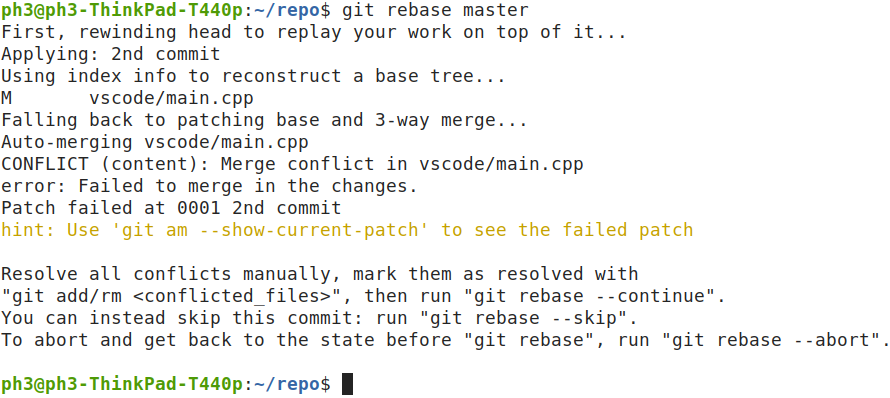
Lúc này sẽ xuất hiện thông báo HEAD is now at 27de696 …, mở file code lên kiểm tra:

Dễ dàng nhận ra, chúng ta đang ở trạng thái gộp branch “demo” vào branch master, còn branch demo2 vẫn chưa được gộp.

Tiến hành checkout qua branch demo2, tiến hành rebase như sau:

$ **git checkout demo2**

$ **git rebase demo2**

****

→ Báo lỗi → chúng ta mở file lên xóa đi những kí tự không cần thiết rồi Save lại!

Thực hiện $ **git add** <file> rồi $ **git rebase --continue**

Xuất hiện thông báo Applying là đã thành công. Bây giờ chúng ta đã có thể thực hiện merge như bình thường.

Thực hiện: $ **git checkout master**

$ **git merge demo2**

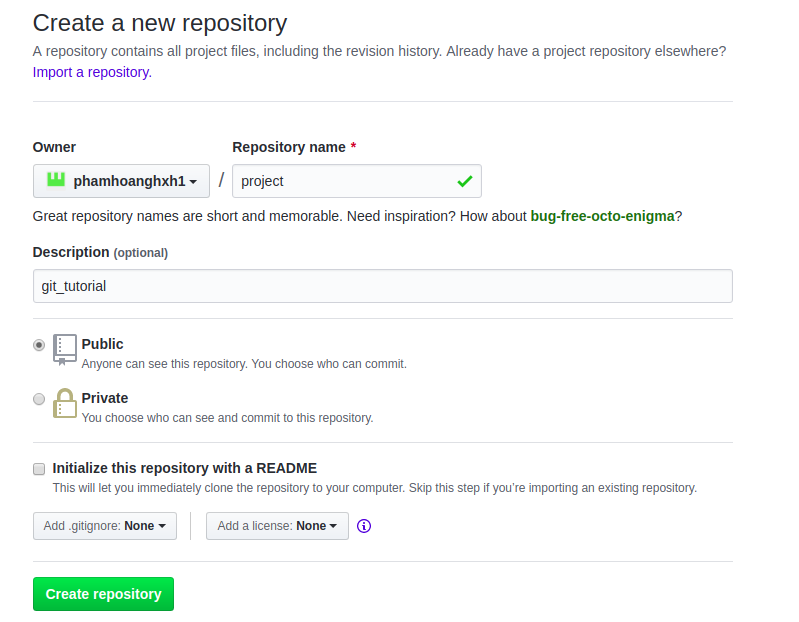
# **CHƯƠNG 3. SỬ DỤNG GIT VỚI SEVER GITHUB**

## 3.1. Tạo tài khoản GitHub:

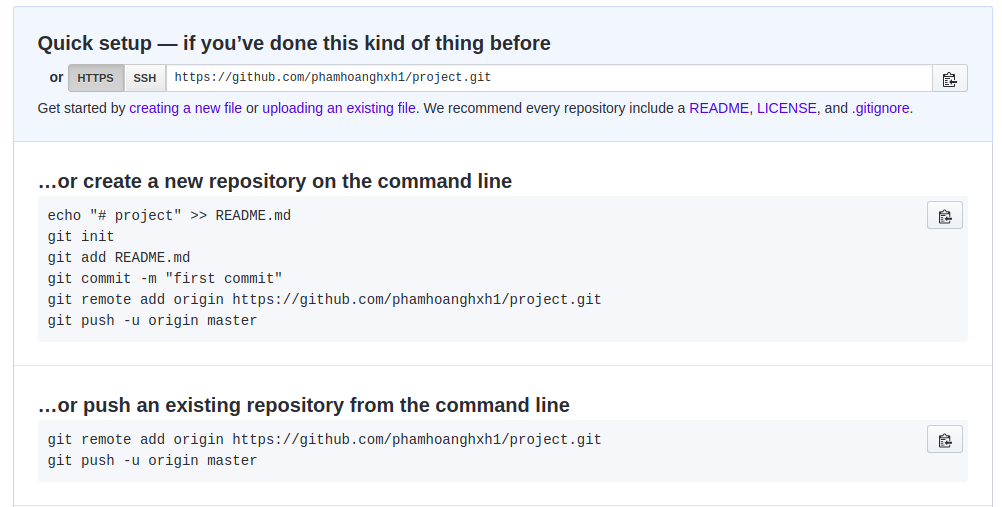
Bạn đọc tạo tài khoản GitHub tại địa chỉ: https://github.com/

## 3.2. Tạo Remote Repository:

Sau khi tạo tài khoản xong, chọn dấu cộng “+” → New Repository → xuất hiện hộp thoại, các bạn điền thông tin đầy đủ rồi chọn “Create repository”.

Ở đây chúng ta chọn kiểu repository là kiểu Public (công khai) để người khác có thể thực hiện sửa đổi trên repository này. Các bạn cũng hoàn toàn có thể chọn kiểu Private (riêng tư) nếu các bạn muốn chỉ riêng mình mới có thể truy cập và thực hiện quản lý repository đó!

Sau khi tạo một Remote Repository, các bạn sẽ nhận được một list các lệnh để thao tác với repository của mình, hãy theo dõi ở bảng dưới.



Có hai giao thức để quản lý repository của bạn là HTTPS và SSH:

Như chúng tôi đã nói ở phần trước (thiết lập SSH key), trong git nếu giao thức bằng HTTPS thì mỗi lần push hay pull đều bắt buộc phải nhập mật khẩu → việc này rất mất thời gian!

Nếu bạn đã thiết lập SSH key thì hãy sử dụng giao thức này để quản lý repository một cách thuận tiện và nhanh nhất!

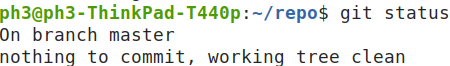
Giao thức SSH key có đường dẫn repository kiểu như sau:

[git@github.com](mailto:git@github.com):phamhoanghxh1/project.git

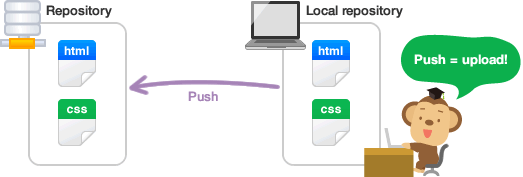
3.3. Push lên Remote Repository:

Truy cập kho repo ở chương 2 bằng lệnh: $ **cd repo**

Ở chương trước chúng ta đã thực hiện commit, chính vì thế ở đây chúng ta không cần phải commit nữa. Hãy chạy lệnh: $ **git status**  để kiểm tra xem có đúng hay không?



**Note:**  Trường hợp bạn đã có thực hiện một số chỉnh sửa trên file thì hãy commit nó để xác nhận. Kiểm tra lại bằng lệnh $ **git status** và nhận được thông báo như trên là ok! Chúng ta có thể bắt đầu push lên remote repository được rồi!



Để push lên remote repository chúng ta sử dụng:

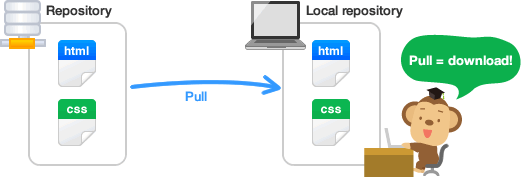
$ **git remote add origin git@github.com:phamhoanghxh1/project.git**

$ **git push -u origin master**

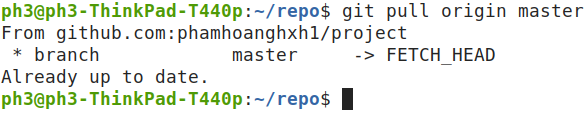
→ Thành công → Hãy kiểm tra lại repository của mình trên trang chủ GitHub.

Sau này, khi thực hiện những thay đổi của mình trên máy local mà bạn muốn chia sẻ thì chỉ cần commit → push, mọi lịch sử thay đổi, nội dung của file đều sẽ được lưu trên sever GitHub, bạn có thể lấy nó về khi cần. Hãy theo dõi tiếp phần “Pull từ Repository” để có thể thấy rõ hơn điều đó.

3.4. Pull từ Repository



Giả sử bạn cần pull repository của mình về để chỉnh sửa, thì hãy chạy lệnh sau: $ **git pull origin master**

3.5. Clone một Repository:

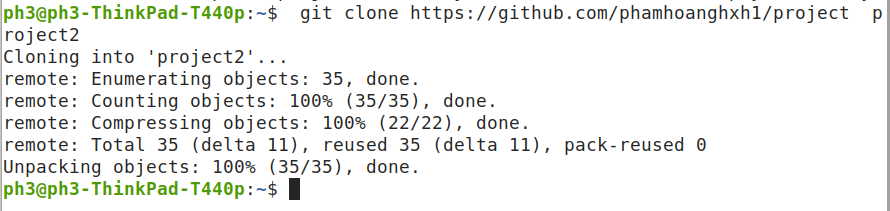
Chẳng hạn bạn cần sao chép một repository của người khác về máy mình để làm việc (hành động này gọi là clone), thì thực hiện:

$ **git clone <url> <directory>**

Trong đó: <url> là đường dẫn của remote repository, <directory> là chỉ tên thư mục nơi sẽ sao chép đến.

Giả định bạn là một thành viên khác trong team, bạn cần clone repository với đường dẫn “<https://github.com/phamhoanghxh1/project>” và tên thư mục là “project2” thì thực hiện:

$ **git clone** [**https://github.com/phamhoanghxh1/project**](https://github.com/phamhoanghxh1/project)project2

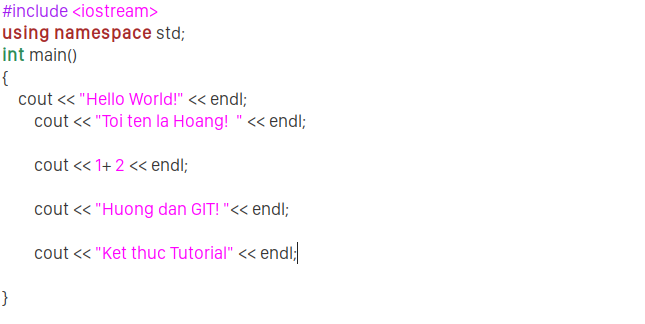
**Note:** Lúc này chúng ta clone repository và lưu vào thư mục Home, chứ không phải lưu vào thư mục repo.

Sau khi hoàn thành, chúng ta kiểm tra thư mục Home và đã thực hiện clone thành công repository project với tên “project2”

## 3.6. Push từ Repository đã Clone:

Bây giờ chúng ta thực hiện chỉnh sửa trên repository đã clone là project2.

Ở đây tôi truy cập thư mục projec2/vscode/main.cpp và thực hiện chỉnh sửa:



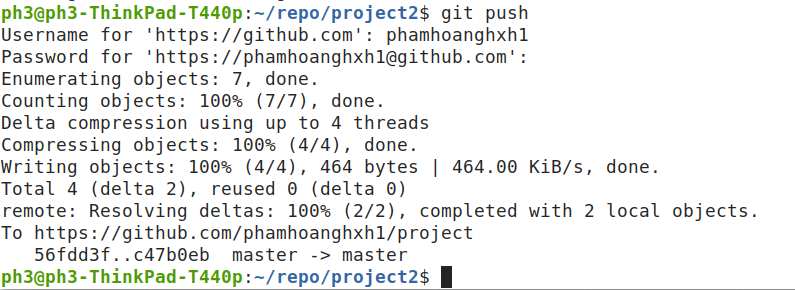
Tiếp theo, thực hiện add file, commit và push:

$ **git add vscode/main.cpp**

$ **git commit -m “first commit”**

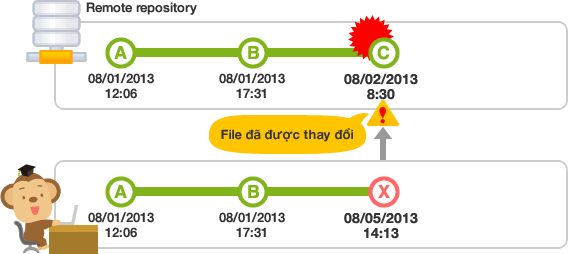
$ **git push**

Nếu hỏi username và password của tài khoản GitHub thì bạn nhập vào, thông báo push thành công như hình bên dưới.

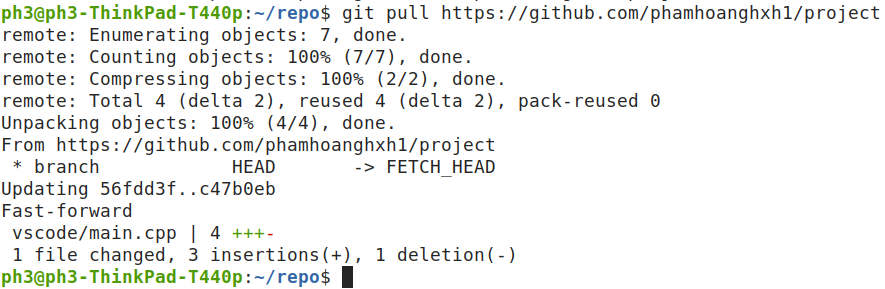
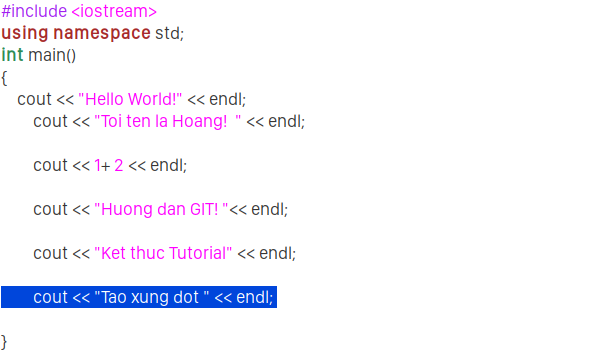


## 3.7. Giải quyết xung đột lúc Push.

Trường hợp trong khoảng thời gian mà sau khi bạn pull lần cuối cho đến khi push lần tiếp theo, mà có người khác lỡ push rồi cập nhật remote repository, thì lần push tới của bạn sẽ bị từ chối (do xảy ra xung đột).

Để giải quyết vấn đề này thì chúng ta sẽ phải thực hiện gộp lịch sử (merge). Bây giờ chúng ta hãy tự tạo ra sử xung đột này bằng cách thao tác đồng thời trên hai repository là “repo” và “project2”

Đầu tiên, thao tác trên thư mục “repo”:

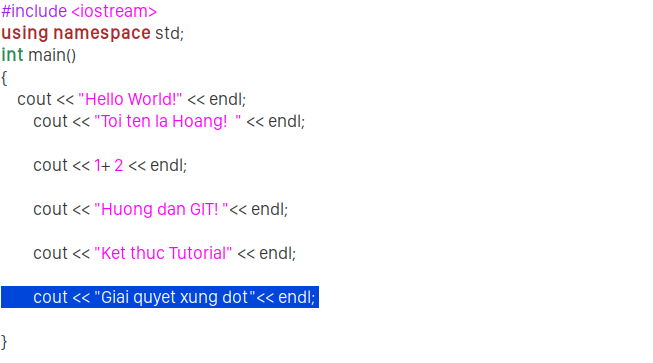
* Pull từ remote repository: $ **git pull <url>**
* Thực hiện chỉnh sửa trên file main.cpp, thêm vào dòng sau:
* Commit:

$ **git add vscode/main.cpp**

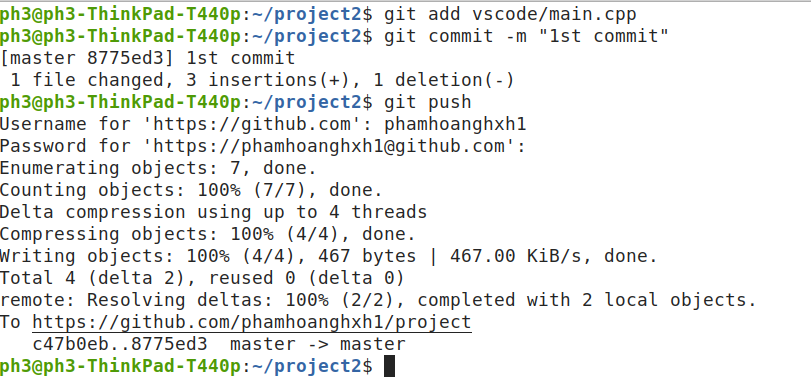
$ **git commit -m “1st commit”**

Thao tác trên thư mục projec2 đã clone:

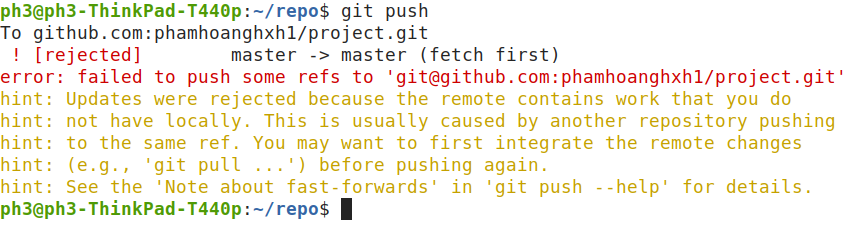
* Thực hiện chỉnh sửa trên file main.cpp bằng việc thêm vào dòng sau:



* Thực hiện commit và push lên remote repository với các lệnh như hình dưới:

→ Chúng ta nhận thấy là không có vấn đề gì xảy ra, bởi lúc này người làm việc với thư mục “project2” chỉ thực hiện chỉnh sửa và push lên repository. Vậy quay lại với người làm việc với thư mục “repo” thì sẽ thế nào?

Với người làm việc trên thư mục “repo”, sau khi đã Pull về, thực hiện chỉnh sửa, bây giờ việc cuối cùng là push lên mà thôi! Chúng ta hãy thử thực hiện push lên từ thư mục “repo” xem như thế nào nhé!

Thật không may là bạn đã bị từ chối (! Rejected). Bởi vì trong lúc bạn Pull về và chỉnh sửa (chưa Push), thì người khác đã chỉnh sửa và Push lên trước!

Để giải quyết xung đột, chúng ta cần phải thực hiện pull về một lần nữa, chỉnh sửa (xóa những kí tự không cần thiết), rồi commit và push lên!

# **Chương 4. Sử dụng Git GUI**

Nếu bạn cảm thấy việc sử dụng console để thao tác hơi khó hiểu và dễ nhầm lẫn, thì bạn có thể sử dụng Git GUI với Visual Studio Code.

Trước hết, bạn hãy cài đặt Visual Studio Code (VSCode).

Truy cập đường dẫn: <https://code.visualstudio.com/docs/setup/linux> để biết thêm chi tiết về cách cài đặt.

Sau khi cài đặt xong VSCode, hãy thực hiện những bước sau để có thể sử dụng Git GUI trên VSCode:

* Khởi động VSCode
* Thêm Extensions cho VSCode bằng cách truy cập File → Preferences → Extensions hoặc nhấn tổ hợp phím Ctrl + Shift + X
* Những Extensions cần thiết bao gồm:

[***Git History (git log)***](https://marketplace.visualstudio.com/items?itemName=donjayamanne.githistory)- View git log, file or line history.

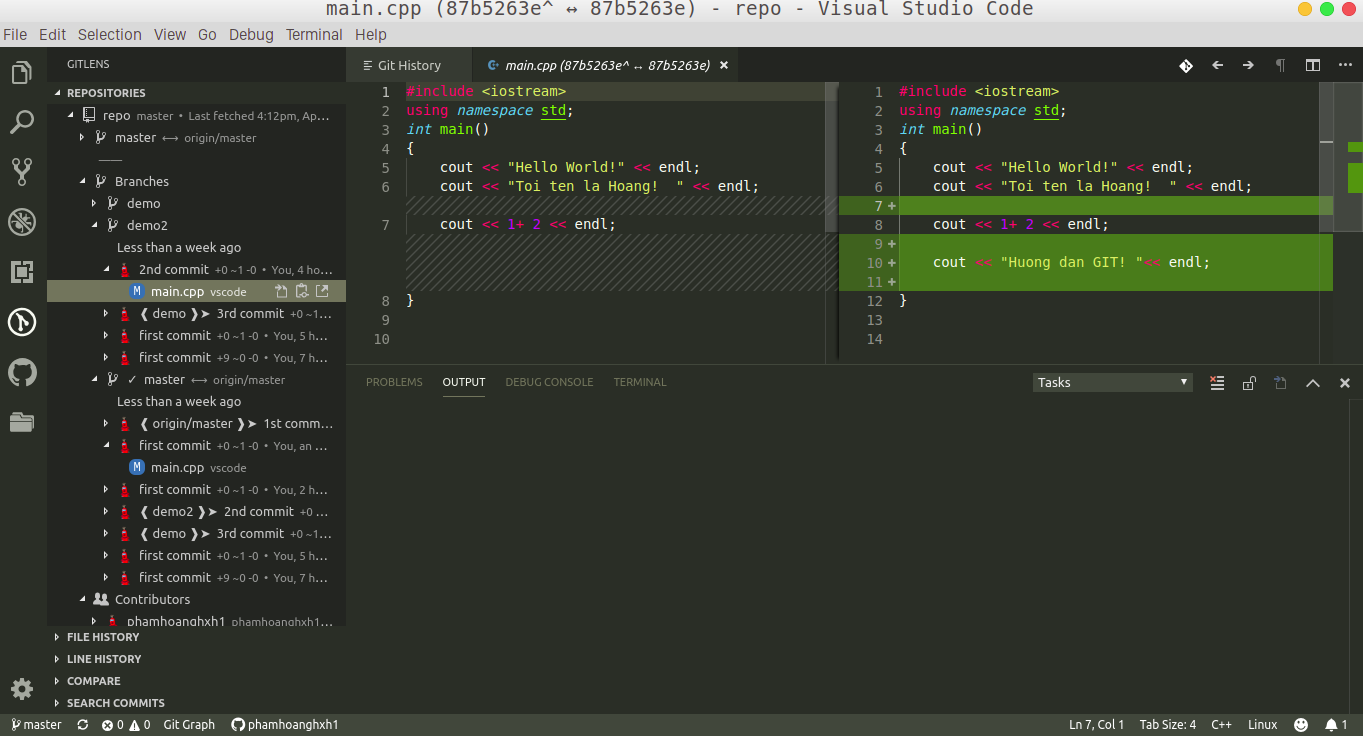
[***Project Manager***](https://marketplace.visualstudio.com/items?itemName=alefragnani.project-manager) - Easily switch between projects.

[***GitLens***](https://marketplace.visualstudio.com/items?itemName=eamodio.gitlens) – GitLens supercharges the built-in Visual Studio Code Git capabilities. It helps you to visualize code authorship at a glance via inline Git blame annotations and code lens, seamlessly navigate and explore the history of a file or branch, gain valuable insights via powerful comparision commands, and so much more.

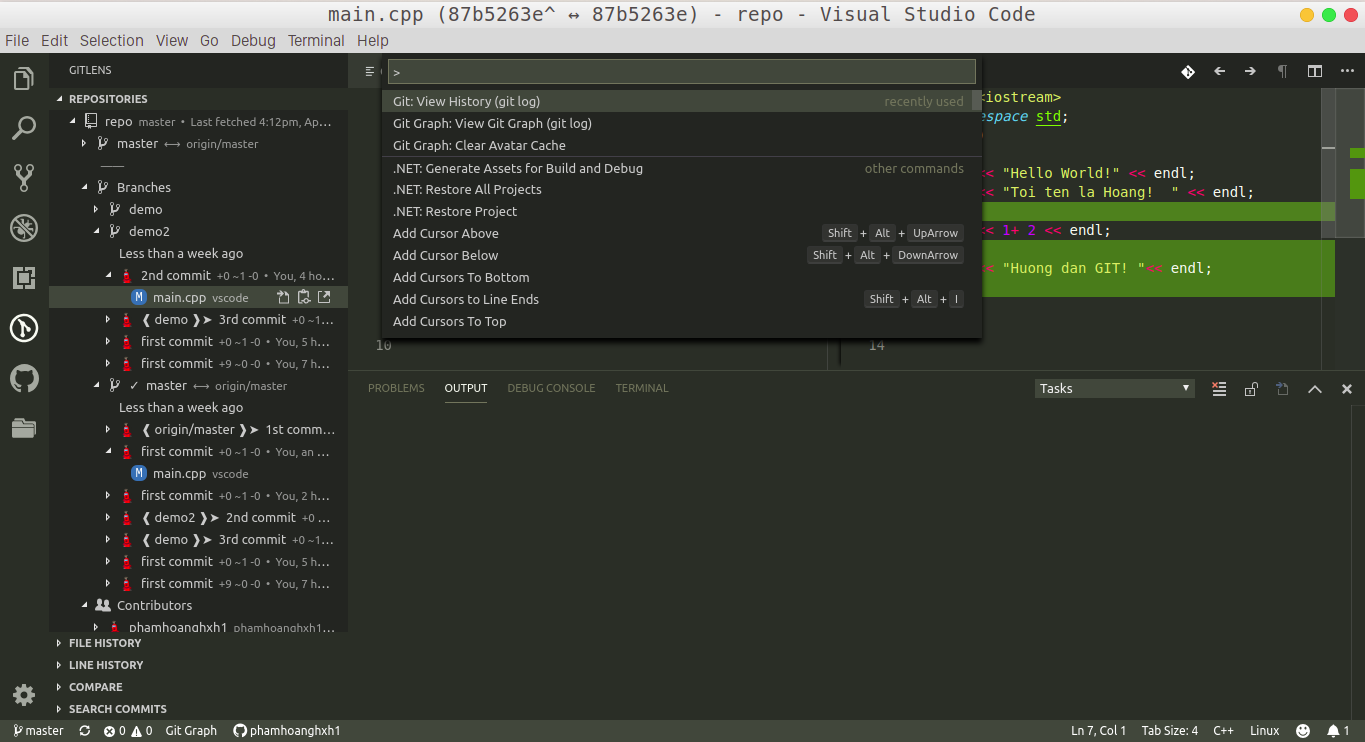
[***Gitignore***](https://marketplace.visualstudio.com/items?itemName=codezombiech.gitignore) - Language support for .gitignore files. Lets you pull .gitignore files from the <https://github.com/github/gitignore> repository.

[***Open in GitHub / Bitbucket / VisualStudio.com***](https://marketplace.visualstudio.com/items?itemName=ziyasal.vscode-open-in-github) - Jump to a source code line in Github / Bitbucket / VisualStudio.com.

* Mở repository của bạn (ở đây tôi mở repository đã tạo từ trước với tên “repo”), thực hiện: File → Open Folder → Repo → OK!
* Chọn Extensions Gitlen (cột bên phía tay trái), mọi lịch sử thao tác trên repository đều sẽ nằm ở đây.



Chúng ta có thể thực hiện những chức năng push, clone, view history tại đây bằng cách thao tác với tổ hợp phím Ctrl + Shift + P:

Xuất hiện hộp tìm kiếm với dấu “>”, ở đây chúng ta có thể thực hiện các chức năng với Git như: Git View History, Git: Checkout, ...

Trên đây là một vài những thao tác cơ bản để quản lý phiên bản cùng với Git và sever GitHub, còn rất nhiều những phần nâng cao và phát triển hơn nữa mà chúng em vẫn còn đang trong quá trình tìm hiểu. Trong quá trình nghiên cứu đó, mặc dù đã cố gắng nhưng chắc chắn rằng vẫn còn nhiều điều thiết sót, tuy nhiên nhờ sự hướng dẫn của thầy – PGS.TS. Nguyễn Quốc Cường đã giúp chúng em hoàn thành đề tài này. Chúng em xin chân thành cảm ơn thầy và rất mong các bạn đón nhận như một tài liệu tham khảo để sử dụng Git một cách tốt nhất!