**HƯỚNG DẪN SỬ DỤNG GITHUB**

# **I, Giới thiệu về git và github**

## **1, Git là gì ?**

**Git** là tên gọi của một **Hệ thống quản lý phiên bản phân tán** (*Distributed Version Control System –****DVCS***) là một trong những hệ thống quản lý phiên bản phân tán phổ biến nhất hiện nay. DVCS nghĩa là hệ thống giúp mỗi máy tính có thể lưu trữ nhiều phiên bản khác nhau của một mã nguồn được nhân bản (**clone**) từ một kho chứa mã nguồn (**repository**), mỗi thay đổi vào mã nguồn trên máy tính sẽ có thể ủy thác (**commit**) rồi đưa lên máy chủ nơi đặt kho chứa chính. Và một máy tính khác (nếu họ có quyền truy cập) cũng có thể clone lại mã nguồn từ kho chứa hoặc clone lại một tập hợp các thay đổi mới nhất trên máy tính kia. Trong Git, thư mục làm việc trên máy tính gọi là **Working Tree**.

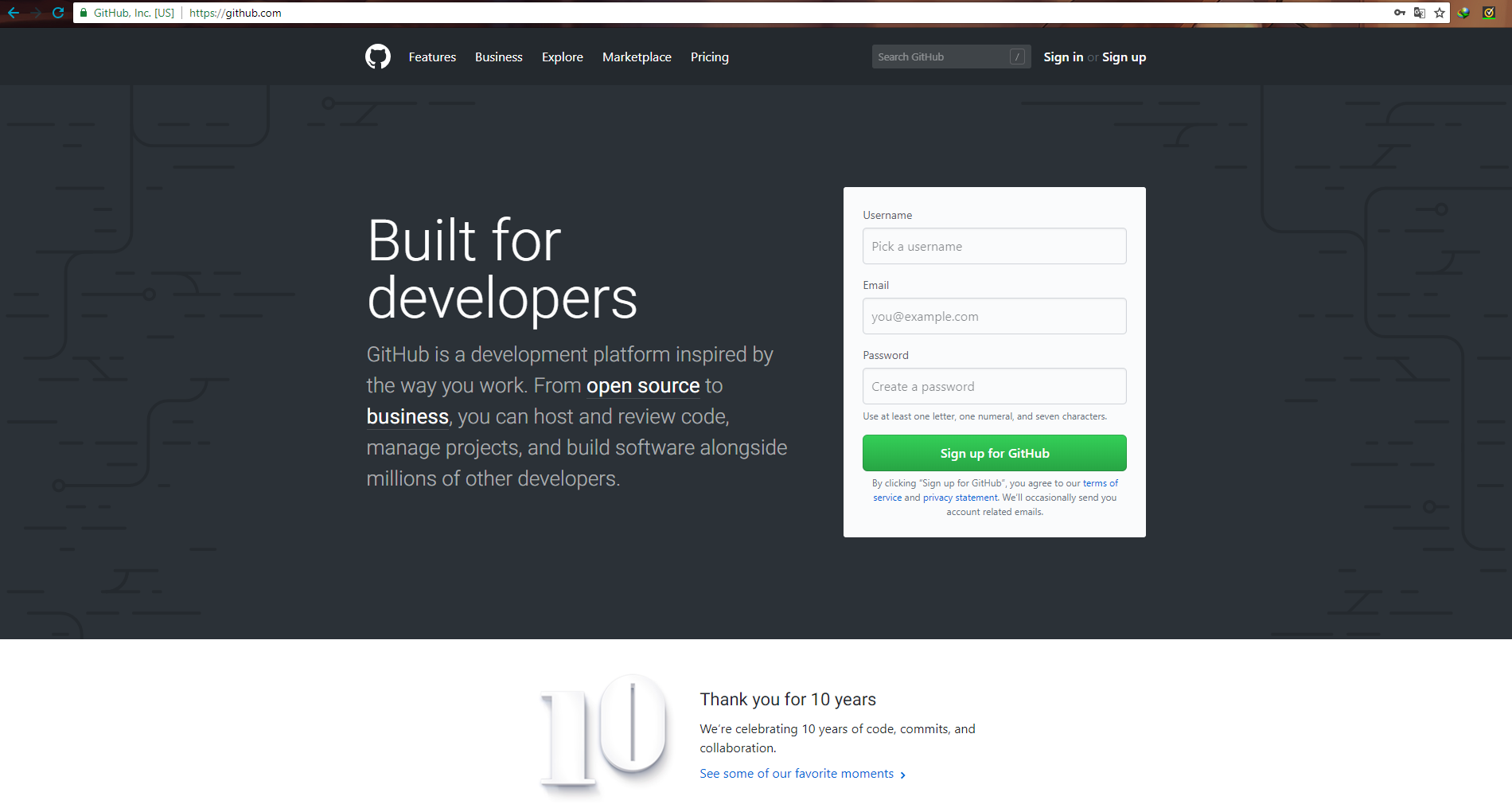
[](https://thachpham.com/wp-content/uploads/2015/04/dvcs.png)

Ngoài ra, có một cách hiểu khác về Git đơn giản hơn đó là nó sẽ giúp bạn lưu lại các phiên bản của những lần thay đổi vào mã nguồn và có thể dễ dàng khôi phục lại dễ dàng mà không cần copy lại mã nguồn rồi cất vào đâu đó. Và một người khác có thể xem các thay đổi của bạn ở từng phiên bản,  họ cũng có thể đối chiếu các thay đổi của bạn rồi gộp phiên bản của bạn vào phiên bản của họ. Cuối cùng là tất cả có thể đưa các thay đổi vào mã nguồn của mình lên một kho chứa mã nguồn.

Cơ chế lưu trữ phiên bản của Git là nó sẽ tạo ra một “*ảnh chụp*” (*snapshot*) trên mỗi tập tin và thư mục sau khi commit, từ đó nó có thể cho phép bạn tái sử dụng lại một ảnh chụp nào đó mà bạn có thể hiểu đó là một phiên bản. Đây cũng chính là lợi thế của Git so với các DVCS khác khi nó không “lưu cứng” dữ liệu mà sẽ lưu với dạng snapshot.

## **2, Github là gì ?**

Git và github là không phải là một và có thể nhiều bạn sẽ bị hiểu lầm. Git là tên gọi của một mô hình hệ thống và như đã giải thích ở trên, **các máy tính có thể clone lại mã nguồn từ một repository** và **Github chính là một dịch vụ máy chủ repository** công cộng, mỗi người có thể tạo tài khoản trên đó để tạo ra các kho chứa của riêng mình để có thể làm việc.



## **3, Mô hình quản lý phân tán của git**

Quản lý phân tán của [Git](http://expressmagazine.net/taxonomy/term/308)là một repositories không cần có chung một nơi để lưu trữ, mà mỗi thành viên sẽ có một repository ở local của họ. Tất cả thao tác ta làm việc với Git đều ở trên máy của ta, local repository, khi quyết định đưa những thay đổi đó lên server ta chỉ cần một thao tác "push" nó lên server. Chúng ta vẫn có thể share thay đổi của chúng ta cho thành viên khác, bằng cách commit hoặc update trực tiếp từ máy của họ mà không phải thông qua repositories gốc trên server (thông qua share ssh cho nhau). Và dĩ nhiên là mọi thao tác đều mang theo thông tin history với Git.

Tính phân tán thì an toàn hơn so với tập trung, vì mỗi bản copy của thành viên đều là full copy từ repository gốc, khi server bị down, các thành viên vẫn có thể làm việc offline, họ vẫn có thể commit và update trên local của họ hoặc thậm chí với nhau mà không cần thông qua server. Khi server hoạt động trở lại, họ có thể cập nhật tất cả lên lại server.

## **4, Cách thức làm việc với github**

Làm việc với GitHub nói riêng hay hệ thống GIT nói chung có 2 workflow chính là local workflow và server workflow.

Bạn có thể làm mọi chuyện thay đổi source code ở local, sau khi đã thay đổi xong, bạn sẽ commit nhưng thay đổi đó lên server và bản lên server phải là bản hoàn chỉnh một tính năng nào đó, hoặc fix bug xong, test xong hoặc ít nhất bản đó phải chạy được. Không được commit code dở dang, chưa qua test lên repository server sẽ làm ảnh hưởng đến các thành viên khác, ngược lại bạn có thể làm điều đó ở repository local (Bạn cũng có thể tạo một branch ở server cho việc commit code dở dang hay tính năng chưa hoàn thành, nó sẽ chiếm space ở server cũng như làm mất thời gian của bạn vào việc tương tác kết nối với server, vậy nên commit nó lên repository local vừa nhanh thao tác lại không mất space của server.)

1) Làm việc với repository ở local: với 2 command thường dùng là git add và git commit

+ Git add: add file đã thay đổi vào stage

+ Git commit: commit các file đã add vào stage lên repository local.

2) Làm việc với repository ở server github:

Sau khi đã làm xong ở local , cuối cùng khi có một bản ổn định và hoàn tất (có thông qua test) ta sẽ quyết định cập nhật nó lên repository server với:

+ push: push thay đổi từ repository local lên repository server

+ fetch: cập nhật thay đổi từ repository server về repository local

+ pull/rebase: sao chép source code từ server về local workspace

## **5, Tại sao nên sử dụng github?**

Một số lợi ích từ việc sử dụng github có thể kể đến ở đây như :

+ Git dễ dàng sử dụng, an toàn và nhanh chóng

+ Có thể khiến quy trình làm việc code theo nhóm trở nên đơn giản hơn bằng việc kết hợp các phân nhánh

+ Có thể làm việc ở bất cứ đâu vì chỉ cần clone mã nguồn từ kho chứa hoặc một nhánh nào đó từ kho chứa.

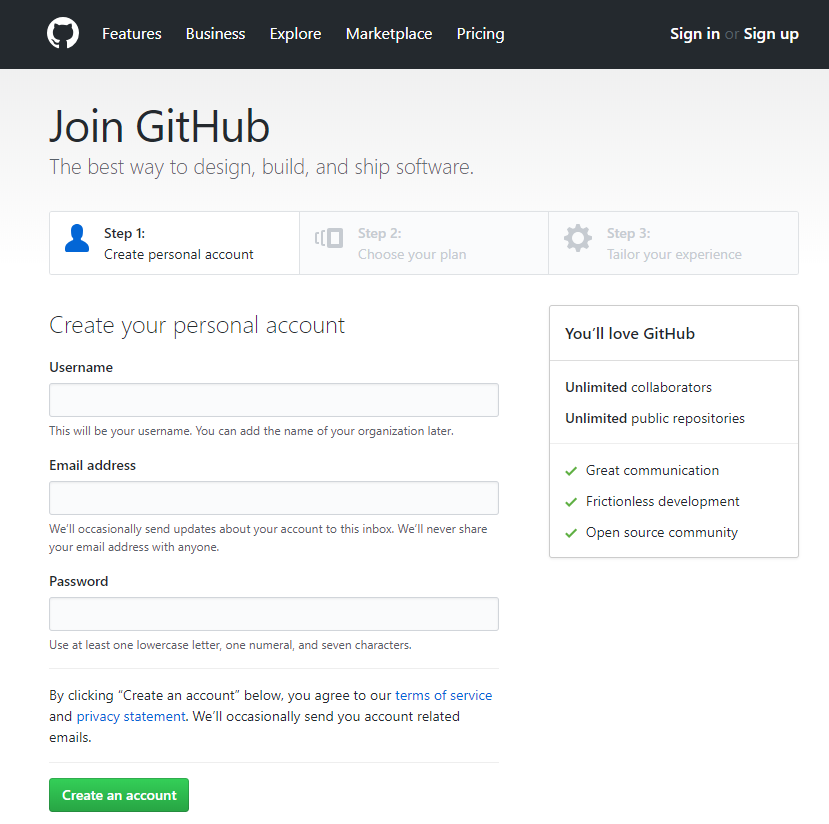
+ Dễ dàng trong việc triển khai sản phẩm

# **II, Hướng dẫn sử dụng github**

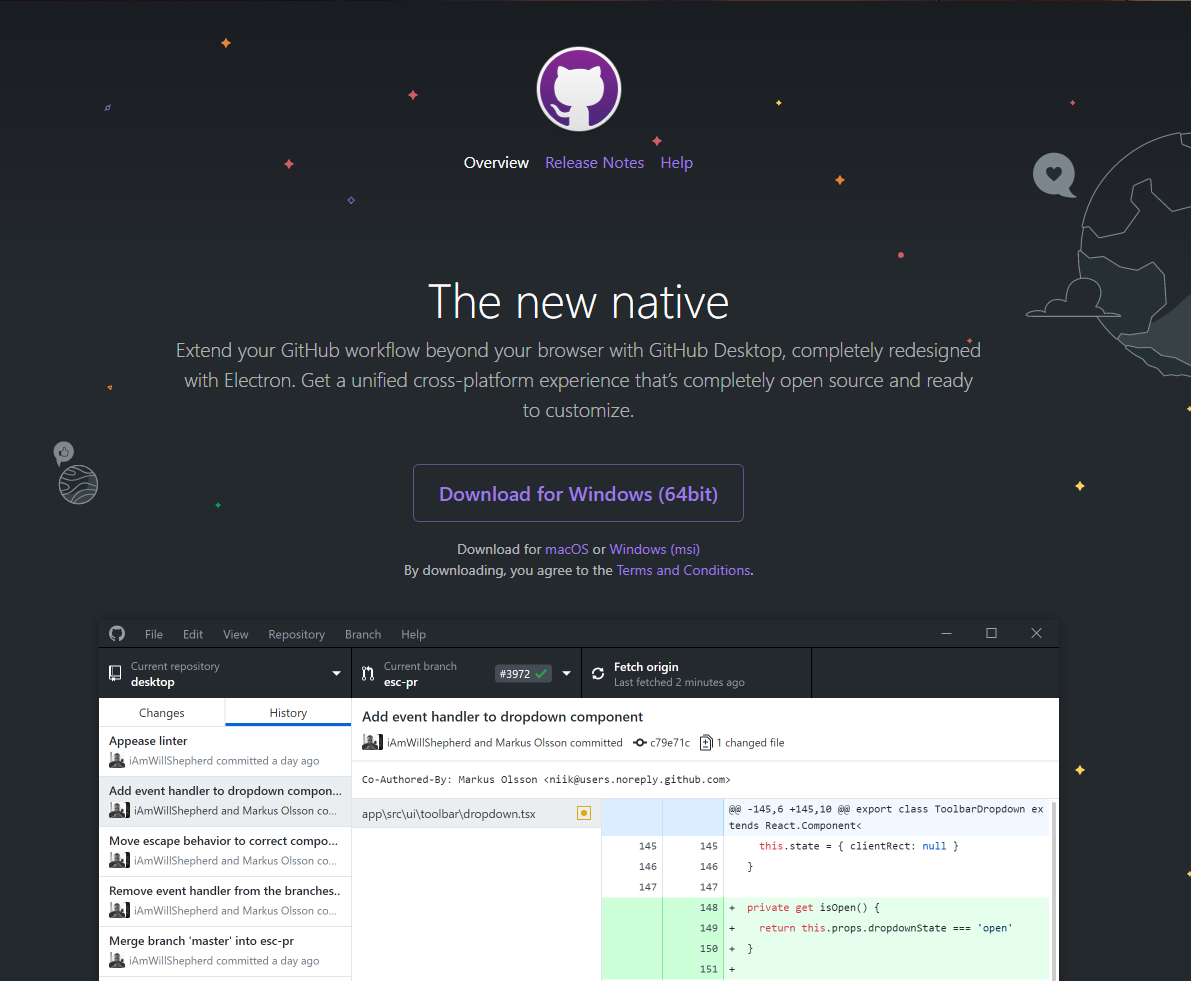
## **1, Bước chuẩn bị**

+ Đầu tiên đăng ký một tài khoản github

>>> Link đăng ký tài khoản github: <https://github.com/join>



+ Sau đó cài đặt Github for windows ( GFW ) để có thể sử dụng trên máy tính của mình



>>> Link tải GFW: <https://desktop.github.com/>

## **2, Sử dụng github**

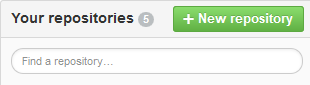
Github có 2 cách để sử dụng :

+ Thứ nhất là tạo một kho lưu code ( repositories ) cho project của mình

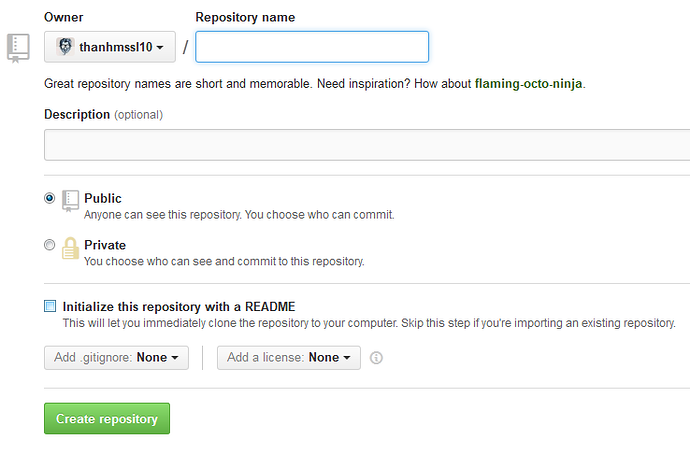
+ Thứ hai là có thể tham gia vào một project nào đó

### **2.1, Tạo repositories**

Trên giao diện web của github ta ấn nút New Repository

  
**New Repository**

Sau khi ấn chọn **New Repository** sẽ ra giao diện như dưới:

[](https://daynhauhoc.s3-ap-southeast-1.amazonaws.com/original/2X/7/76a4800ae5002b93adffbf520f94c13ef214f265.png)

Trong đó:

+ **Repository name:** là tên dự án mà bạn muốn đặt

+ **Description:** miêu tả chi tiết dự án, có thể điền hoặc không

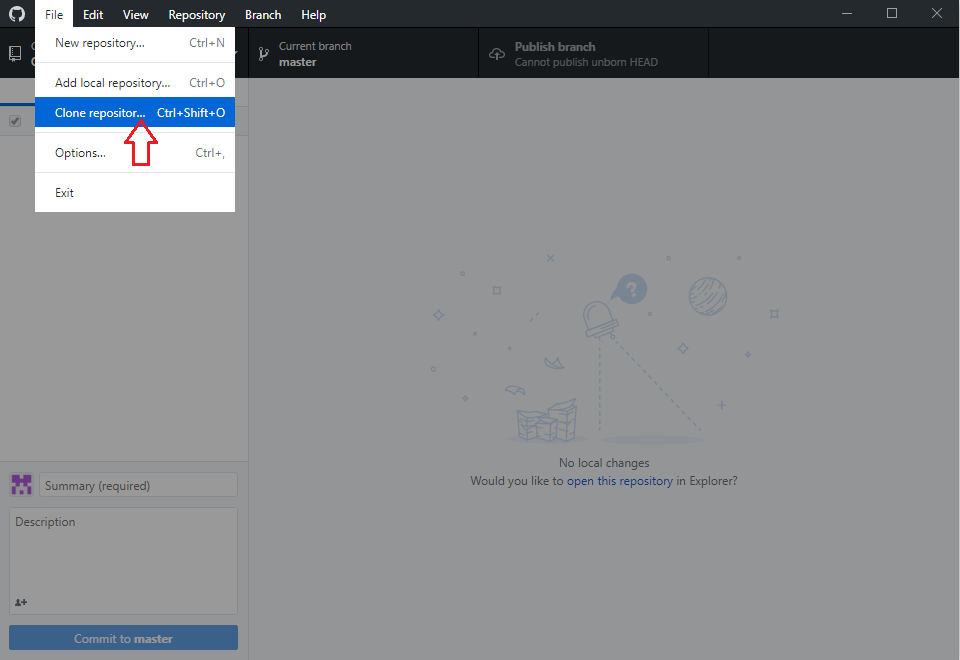
+ **Public or Private:** nếu chọn chế độ private thì người khác sẽ không thấy code của bạn nhưng bạn sẽ phải tốn phí cho nó.

+ **Intialize this repository with a README:** Có thể chọn hoặc không

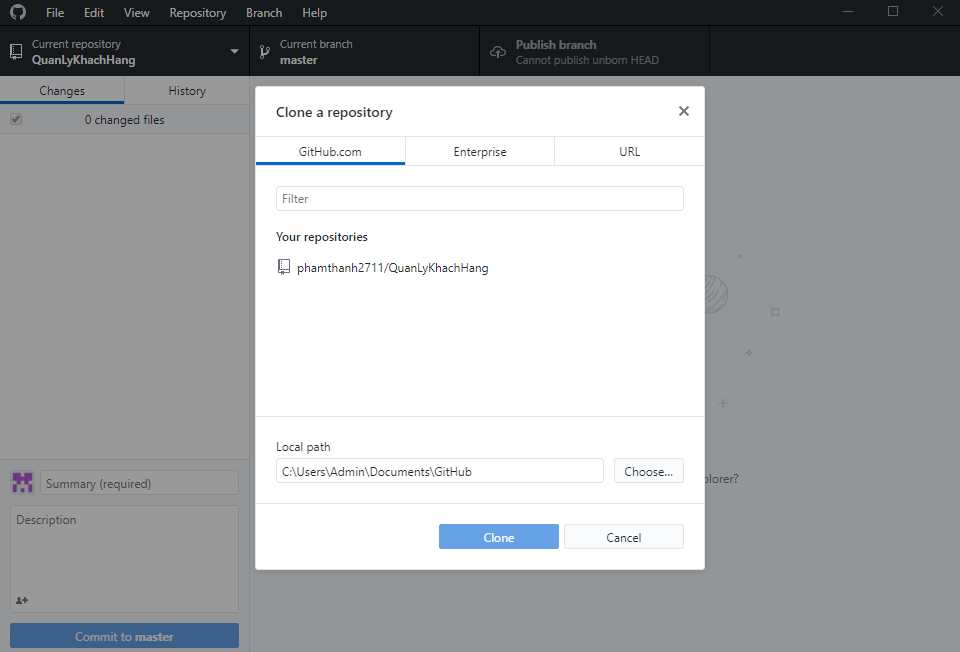
Sau khi điền xong thông tin bạn ấn create repository là xong. Bạn sẽ thu được remote repository vừa tạo, và link này có dạng **git@github.com:user-name/repository-name.**

### **2.2, Tạo clone repository trong github desktop**

Mở Github for Window để tạo clone repository về máy tính

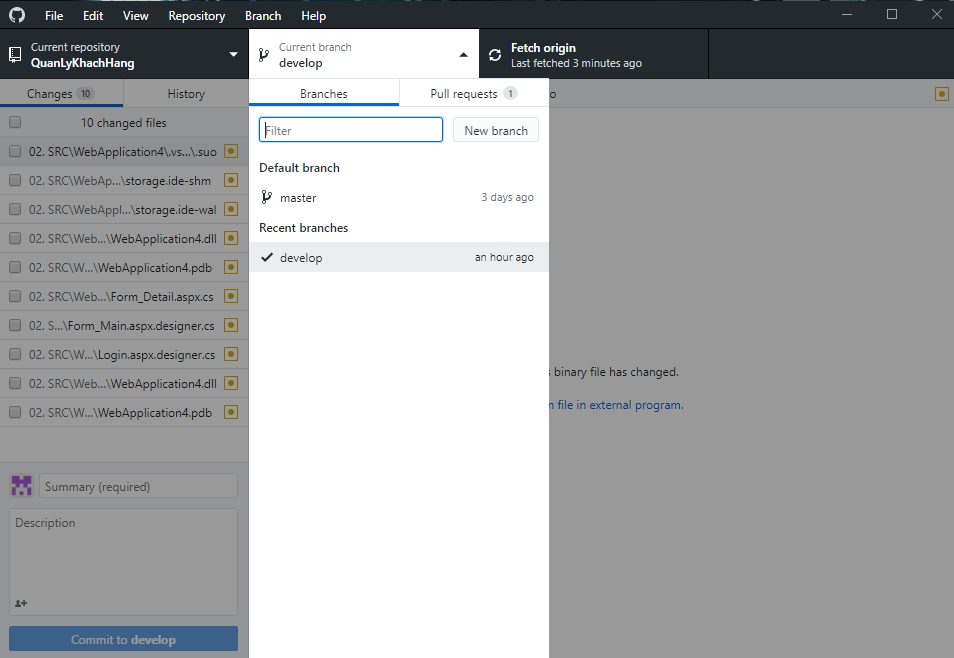


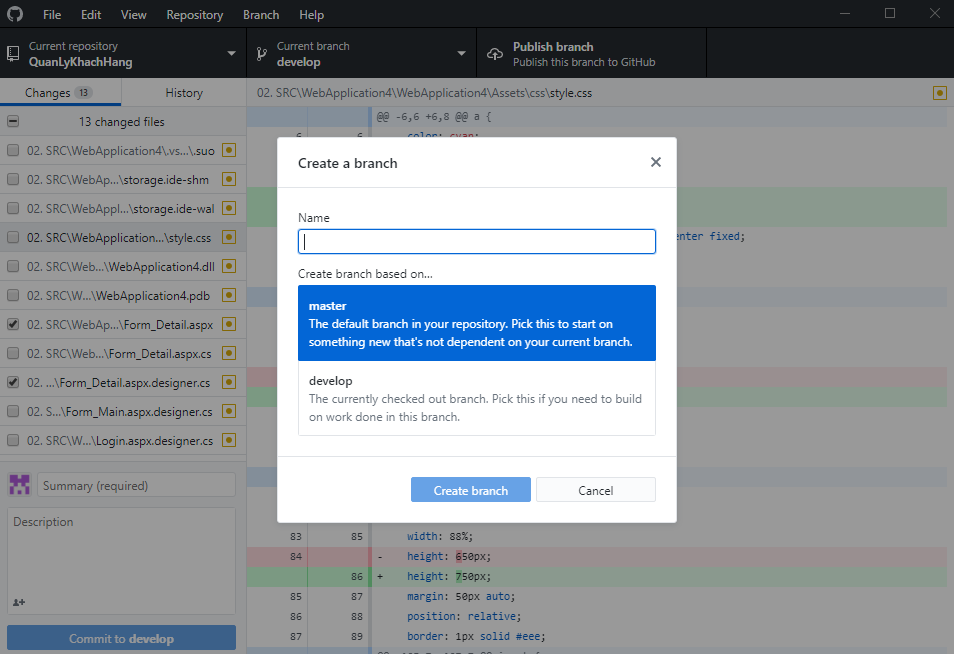
Sau khi tạo clone repository xong thì sẽ xuất hiện



Và từ bây giờ bạn có thể cập nhật các dự án của mình vào trong thư mục github trên máy tính

### **2.3, Tạo phân nhánh trong github desktop**





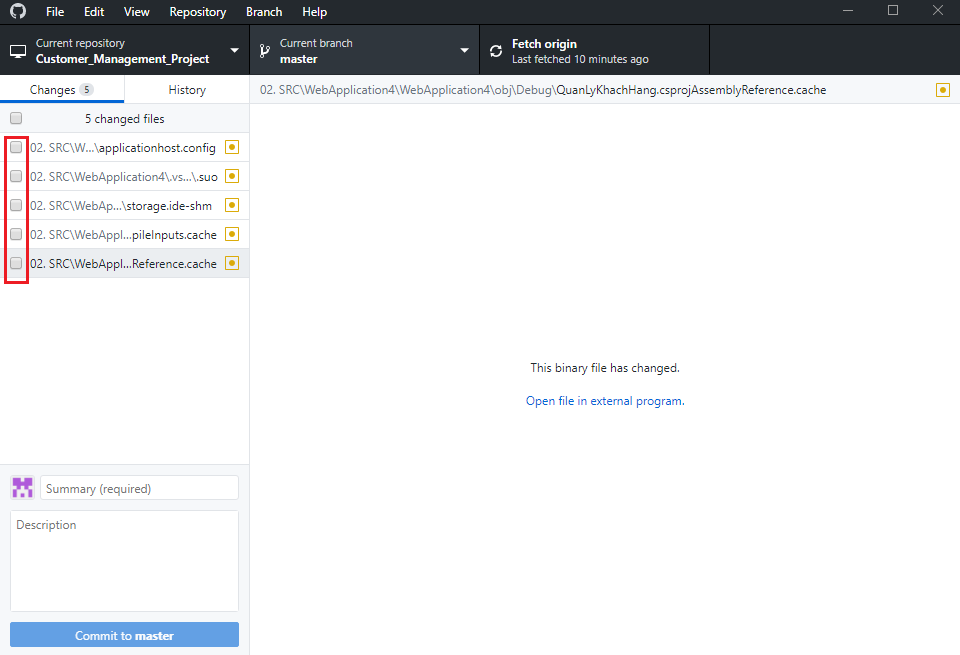
Chọn new branch và sau đó sẽ hiện lên box **create a branch**.

Trong đó:

+ Name: điền tên của nhánh định tạo.

+ Create branch based on: Chọn nhánh cha của nhánh định tạo.

### **2.4, Đưa những thay đổi lên github**



Khi có bất cứ sự thay đổi gì trong project thì sẽ được hiện lên trong ô danh sách bên trái

Bạn có thể tích vào check box để chọn những file thay đổi sẽ được đưa lên github.

Sau đó điền tiêu đề tóm tắt vào ô textbox summary bên dưới và kích nút commit to master.

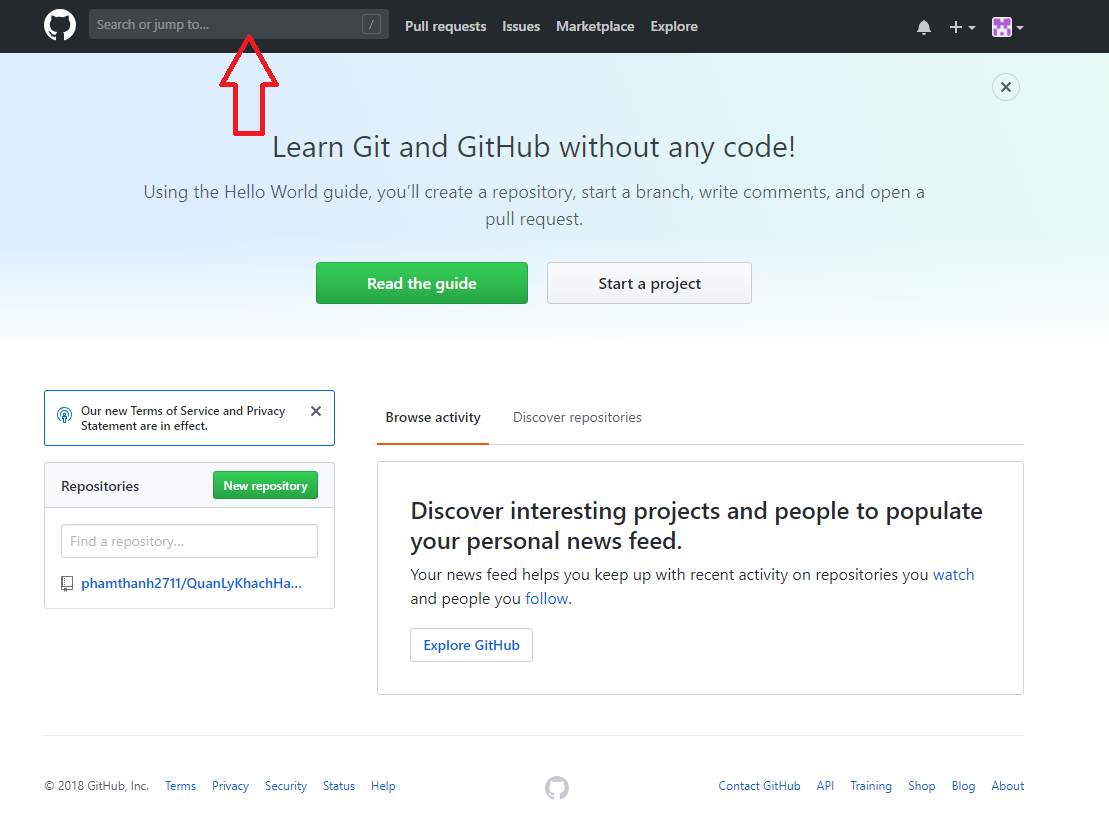
Cuối cùng nhấn vào fetch origin để đưa commit lên trên github.

Lưu ý:

* Commit to master mới chỉ update trạng thái trên local repository của bạn

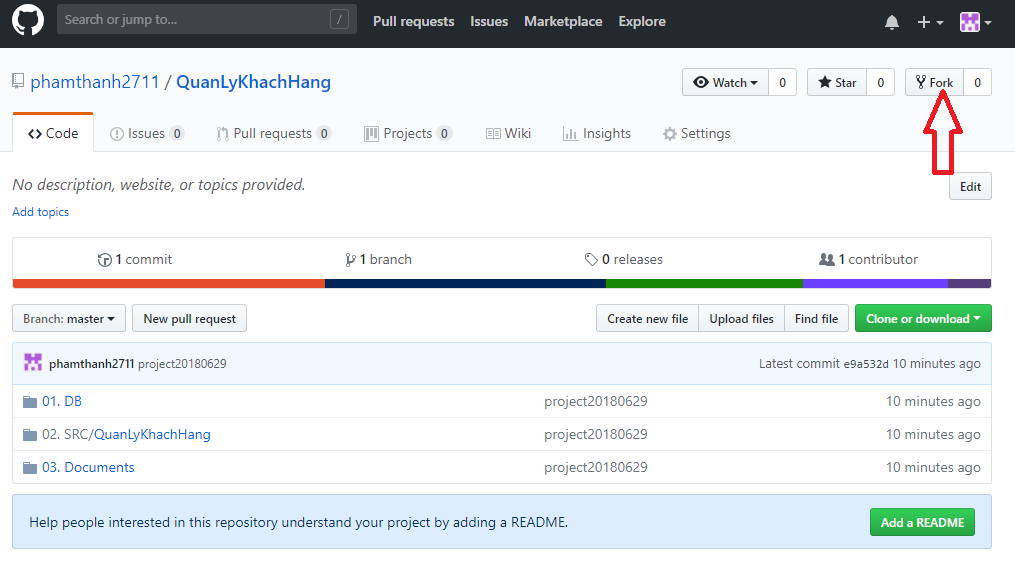
- Fetch origin để update thay đổi lên repository trên github.com

### **2.5, Tham gia vào một project khác**



Để tham gia vào một dự án khác thì trên ô tìm kiếm bạn gõ tên dự án.

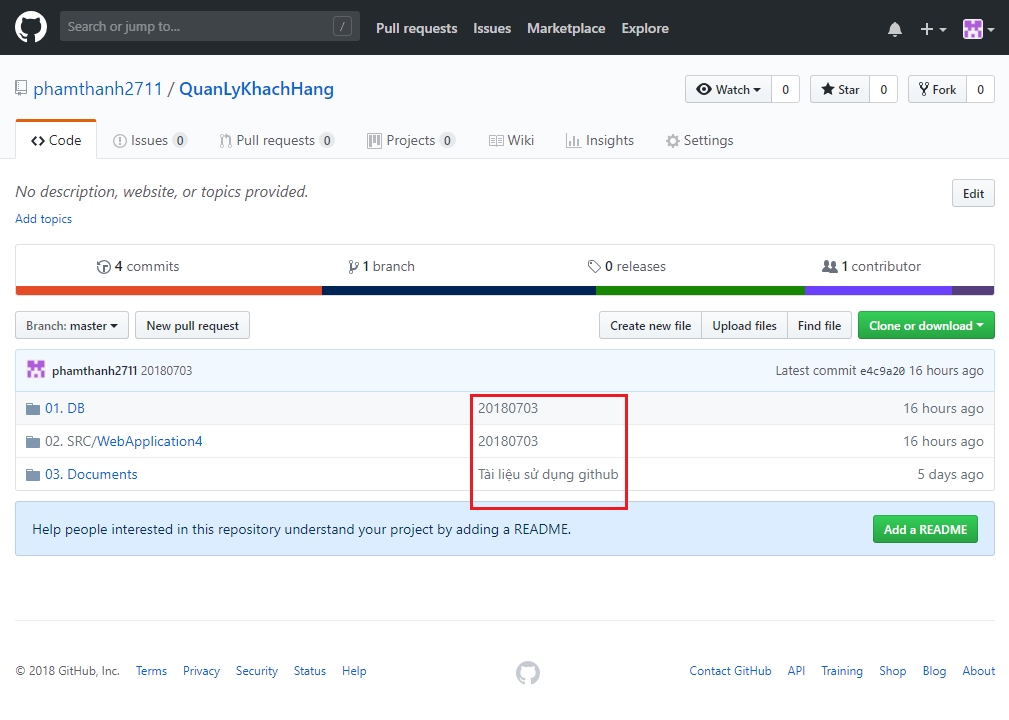
VD: để tham gia dự án của mình , bạn gõ phamthanh2711/QuanLyKhachHang ,sau đó chọn fork



Sau khi fork thì tiếp theo làm gì để lấy project về máy mình?

### **2.6, Xem những thay đổi trong project**

Cách xem thay đổi có thể thực hiện trên cả web và Git for windows, nên them cả hướng dẫn sử dụng xem thay đổi trên windows



Vùng khoanh đỏ trên đây là những thay đổi mới nhất của project và bên cạnh là thời gian thay đổi được đưa lên github.

Hoặc có thể nhấn vào commits để xem cả những thay đổi được đưa lên trước đó.



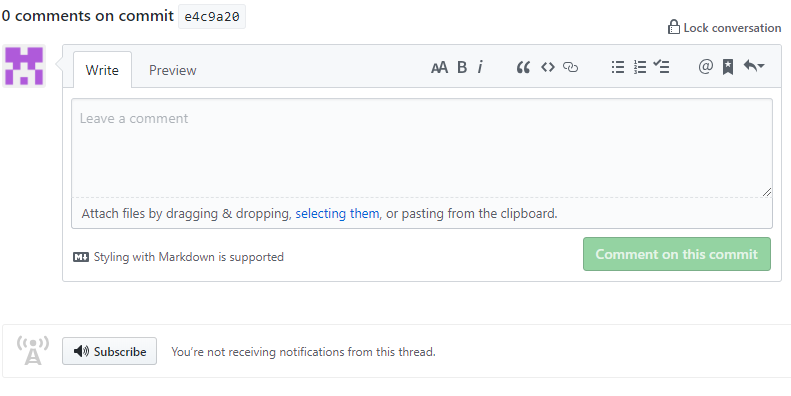
Sau khi nhấn vào , bạn có thể thấy được các sự thay đổi trong chương trình

Trong đó :

+ Trên cùng là tên file có sự thay đổi, bên trái là tương quan tỉ lệ giữa sự thêm và bớt, được thể hiện bằng các ô chấm màu xanh(thêm vào) và màu đỏ(bớt đi)

+ Những dòng có dấu “+” là những dòng mới được thêm vào

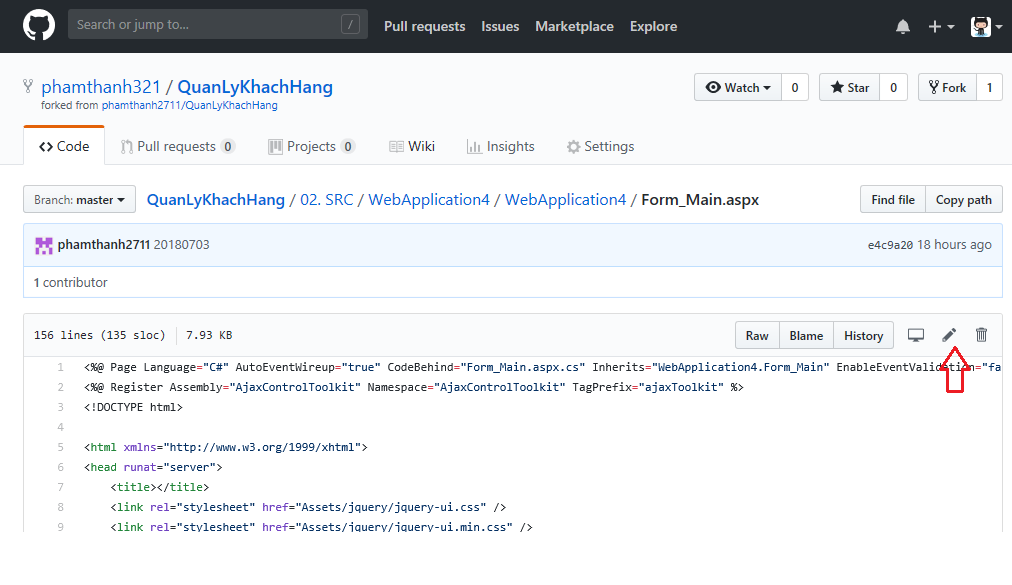
+ Những dòng có dấu “-” là những dòng mới được bớt đi



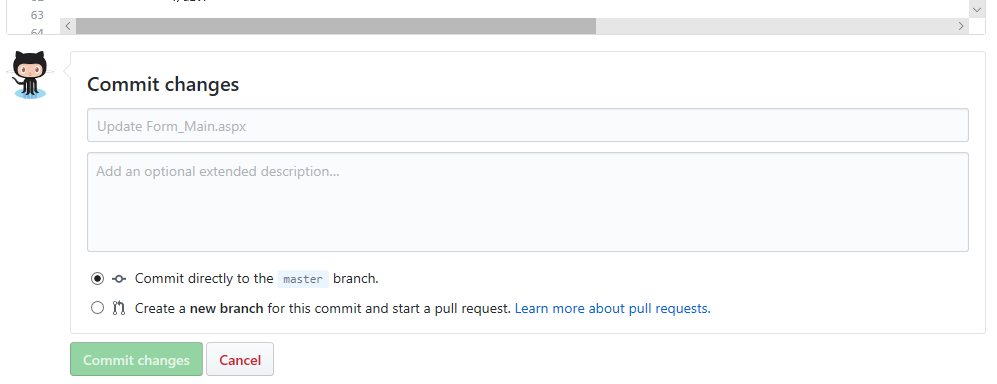
Kéo xuống dưới cùng sẽ có một box cho phép các bạn có thể để lại comment cho commit này.

### **2.7, Cách pull request**

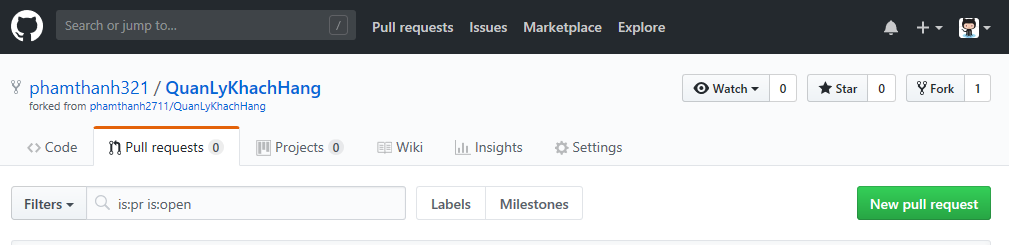
Bây giờ bên B (người tham gia dự án) nếu chỉnh sửa file, khi ấy file đó giữa người A( người tạo repository ban đầu) và B đã có sự khác nhau và B có thể gửi pull request (PR) tới master. PR nghĩa là partner nói với master:" B đã sửa file, yêu cầu A hợp nhất vào repo của A", nếu như A đồng ý với yêu cầu đó,thì file trên repo của A sẽ thay đổi theo file trên repo của B.

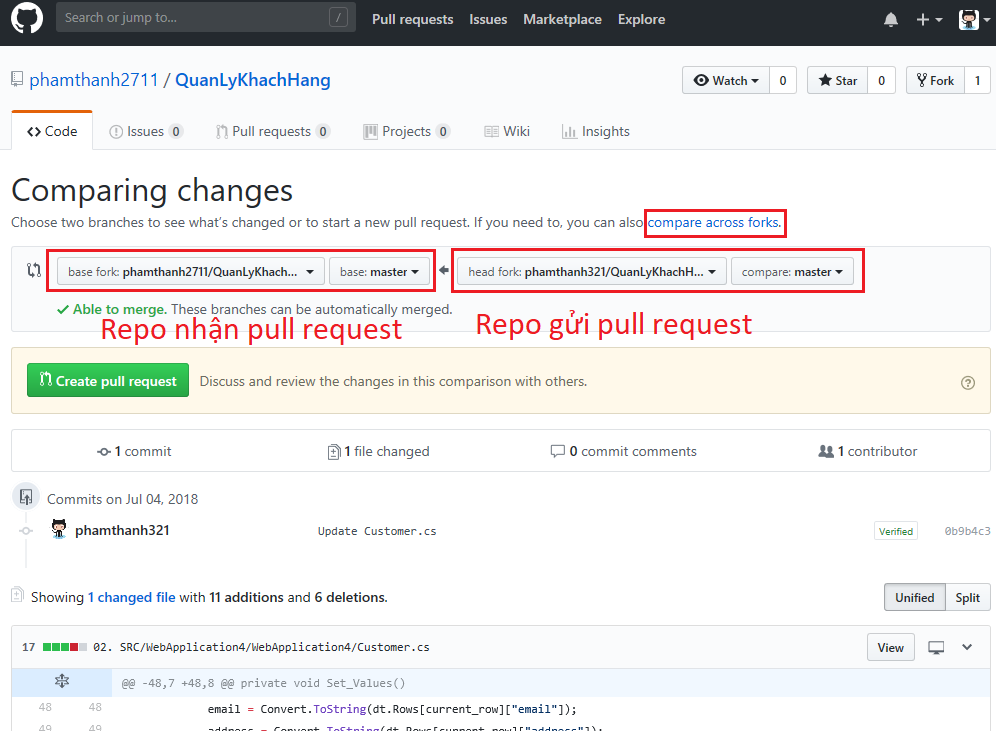


Nhấn vào chọn chỉnh sửa file trực tiếp trên web



Sau khi **commit changes**, ta bắt đầu pull request. Vào **pull requests** chọn **new pull request**.



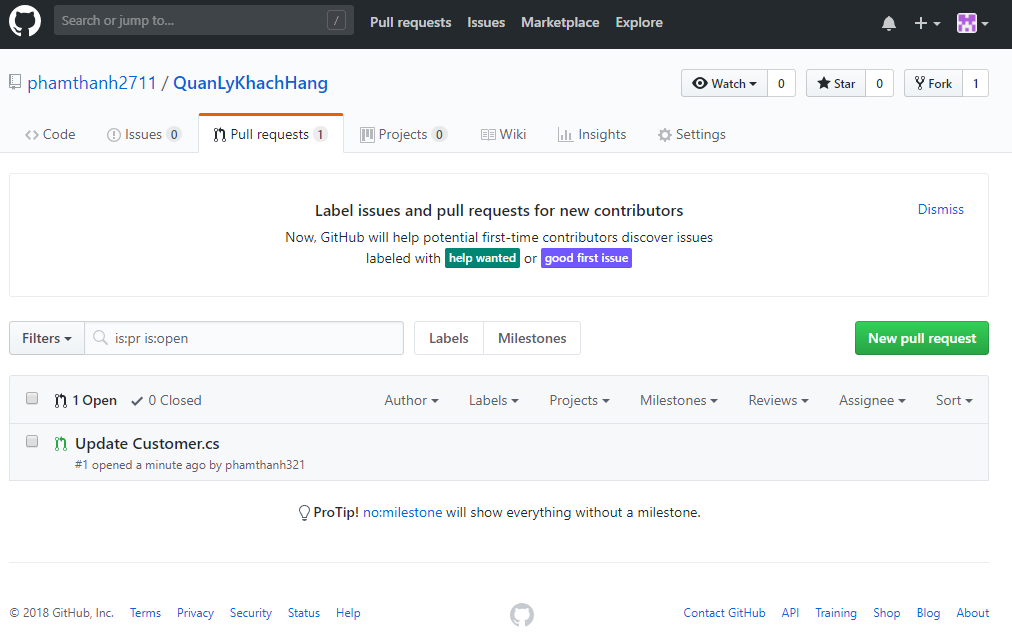


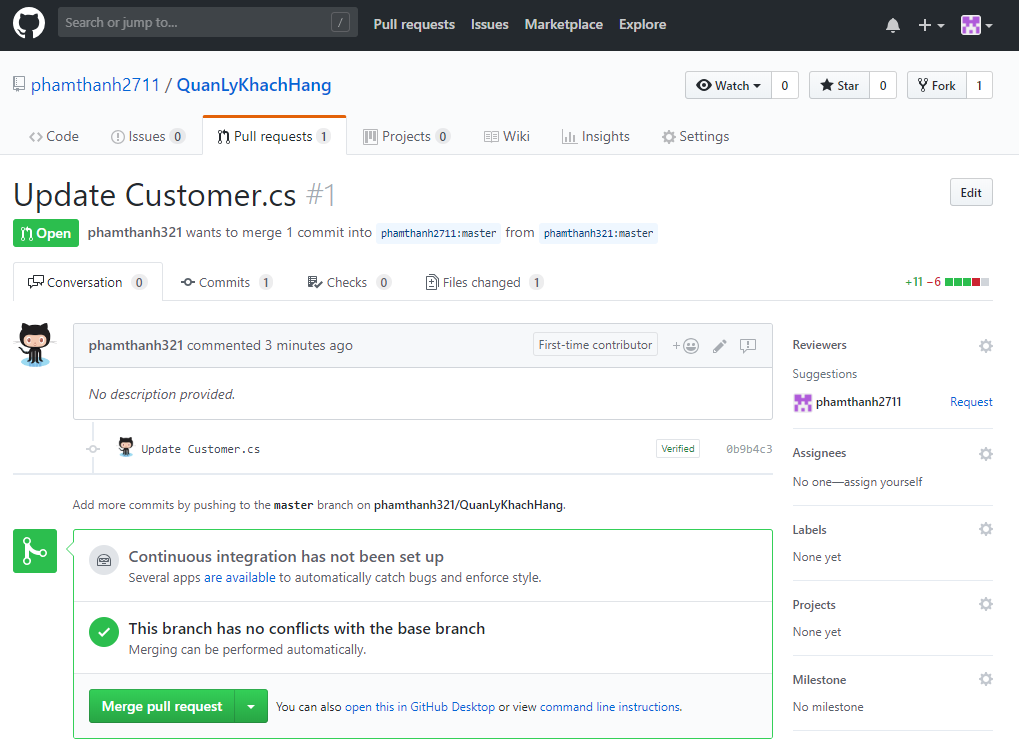
Thông thường thì github tự động hiện bên nhận PR là repo của A và bên gửi PR là repo của B, nhưng nếu không hiện đúng như vậy thì B phải tự chỉnh bằng tay, trong lúc đổi bên nhận gửi PR, thì sẽ có thời điểm là bên nhận và bên gửi là cùng 1 repo, khi đó phải dùng nút **compare across forks** để giải quyết.

### **2.8, Hợp nhất pull request**

Sau khi B gửi pull request, thì bên A sẽ có thông báo có Pull Request.

Thêm tên ảnh để thể hiện rõ rang hơn

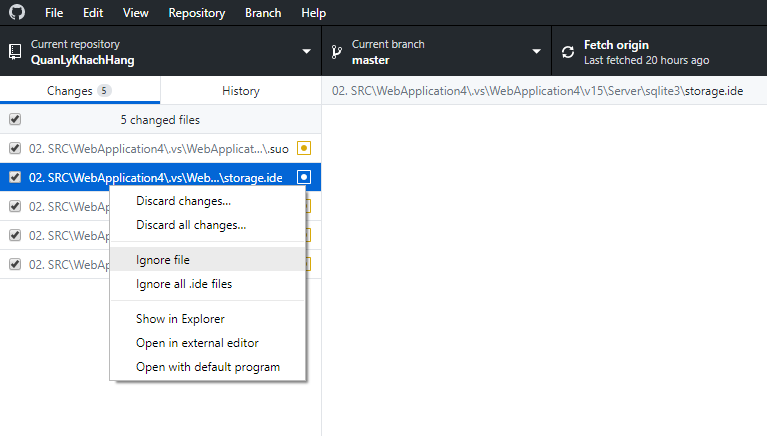




Sau khi chọn **Merge pull request** để chấp thuận hợp nhất thì thay đổi sẽ được hợp nhất vào repo của A.

### **2.9, Tạo file gitignore**

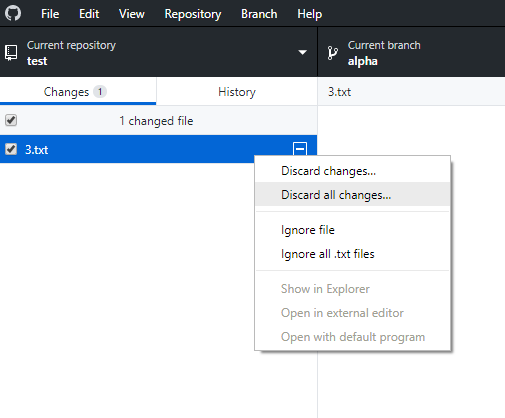
Khi mà bạn không muốn một file bất kỳ nào đó thể hiện sự thay đổi trên git bạn có thể cho nó vào trong file gitignore và để làm được điều đó bạn click chuột phải vào file thay đổi đó.



Bạn chọn ignore file này thì khi có sự thay đổi ở file này nó sẽ không xuất hiện nữa hoặc chọn ignore all nếu muốn tất cả những file có đuôi mở rộng giống file này không xuất hiện.

### **2.10, Khôi phục file khi mới xóa hay thay đổi**

Sau khi xóa file trong thư mục git trên máy bạn hoặc thay đổi nội dung của nó, nếu như muốn khôi phục lại file như cũ, bạn có thể click chuột phải vào sự thay đổi đó và chọn Discard changes hoặc discard all changes nếu như muốn khôi phục lại hết các sự thay đổi.



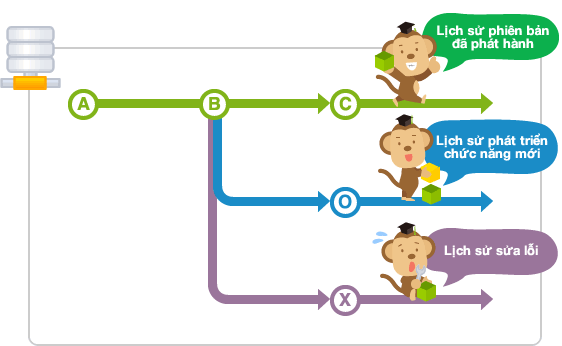
## **3, Tìm hiểu về branch**

### **3.1, Branch là gì ?**

Trong việc phát triển phần mềm, thì ứng với một phần mềm có nhiều thành viên đồng thời tiến hành thêm chức năng hay là tiến hành chỉnh sửa lỗi cùng một lúc. Và ở tình trạng tồn tại của nhiều phiên bản đã phát hành thì cũng phải lưu giữ từng phiên bản.

Chính vì vậy để hỗ trợ quản lý phiên bản hay thêm nhiều chức năng được tiến hành song song, một chức năng được trang bị thêm được gọi là branch ở Git.

Branch là 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.

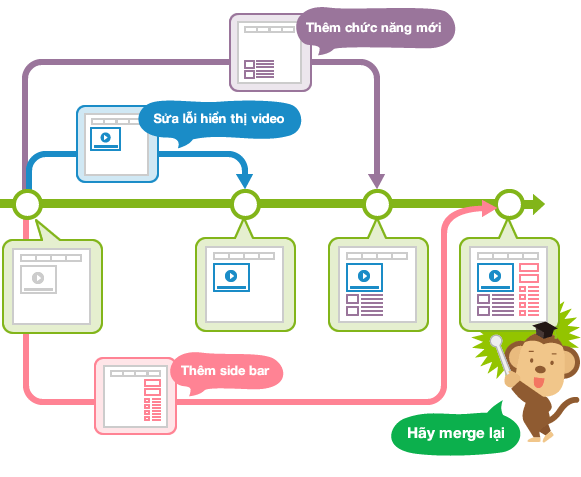


Hơn nữa, branch đã phân nhánh có thể chỉnh sửa tổng hợp lại thành 1 branch bằng việc hợp lại (merge) với branch khác.

Sơ đồ bên dưới là mô hình của thao tác song song đã sử dụng branch.

Các thành viên của nhóm sẽ tạo branch dùng riêng cho công việc của mình từ branch chính để không ảnh hưởng đến công việc của các thành viên khác. Sau đó, những thành viên đã hoàn thành công việc của mình sẽ thực hiện đưa thay đổi của mình vào branch chính. Theo cách như vậy, sẽ không bị ảnh hưởng từ công việc của các thành viên khác, và bản thân mình có thể thực hiện công việc của mình.

Hơn nữa, bằng việc để lại lịch sử theo đơn vị công việc, trong trường hợp có phát sinh vấn đề thì việc điều tra nguyên nhân ở những vị trí thay đổi cũng như việc tiến hành đối sách khắc phục sẽ trở nên dễ dàng hơn.



Khi tiến hành commit lần đầu trong repository thì Git sẽ tạo ra một branch có tên là master. Vì thế những lần commit sau sẽ được thêm vào branch master cho đến khi chuyển đổi branch.

### **3.2, Vận dụng branch**

Trên Git thì có thể tự do tạo branch. Thế nhưng để sử dụng branch hiệu quả thì trước hết cần thiết phải thiết lập qui tắc vận dụng.

Ở đây sẽ giới thiệu phương pháp vận dụng đã sử dụng với 2 loại branch là branch chủ đề (Topic branch) và branch tích hợp (Intergration branch).

**A, Branch tích hợp ( Integration branch )**

Branch tích hợp là branch có thể tạo ra bản phát hành bất cứ khi nào. Hơn nữa, nó cũng được sử dụng như là nguồn phân branch của branch chủ đề. Vì thế, việc duy trì trạng thái ổn định là điều cần thiết.

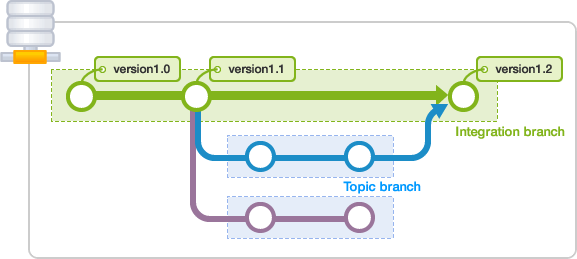
Trường hợp tiến hành thay đổi gì đó thì sẽ thường tạo ra branch chủ đề rồi thực hiện thay đổi. Và việc kiểm tra và build tự động sử dụng công cụ CI như Jenkins chẳng hạn thì sẽ sử dụng branch này để tiến hành.

Thông thường sẽ sử dụng branch master như là một branch tích hợp.

**B, Branch chủ đề ( Topic branch )**

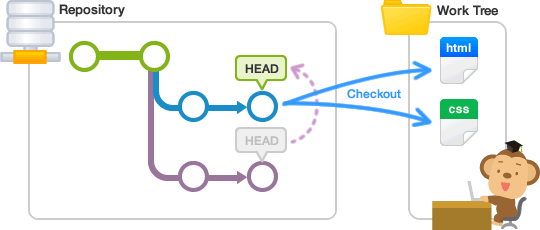
Branch chủ đề là branch tạo ra nhằm tiến hành công việc liên quan đến chủ đề như là chỉnh sửa lỗi hay là thêm chức năng. Khi tiến hành cùng lúc những công việc có liên quan đến nhiều chủ đề thì số lượng branch chủ đề tương ứng sẽ được tạo ra.

Branch chủ đề sẽ tạo ra bằng cách phân branch từ branch tích hợp đã ổn định, khi đã hoàn thành xong công việc sẽ sử dụng đưa vào branch tích hợp.



### **3.3, Chuyển đổi branch**

Để chuyển đổi branch làm việc thì sẽ thực hiện thao tác gọi là checkout. Khi thực hiện checkout, trước tiên nội dung của lần commit cuối cùng trong branch chuyển đến sẽ được mở ra trong work tree. Và commit đã tiến hành sau khi check out thì sẽ được thêm vào branch sau khi di chuyển đến.



**+ Head**

HEAD là tên hiển thị phần đầu của branch đang sử dụng hiện tại. Mặc định là đang hiển thị phần đầu của master. Bằng việc di chuyển HEAD thì branch đang sử dụng sẽ được thay đổi.

**+ Stash**

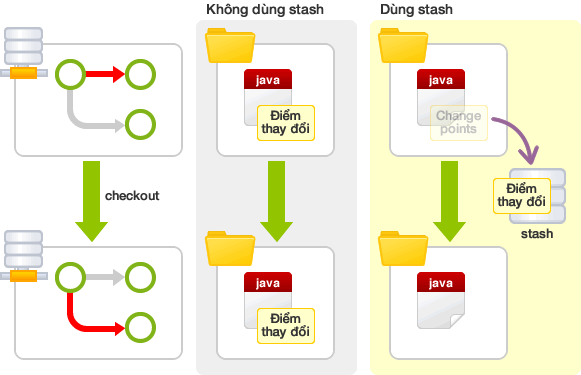
Khi những file được thêm mới hay nội dung thay đổi vẫn chưa commit vẫn còn lưu lại Index và Work tree, mà thực hiện checkout đến branch khác thì nội dung thay đổi đó sẽ di chuyển từ branch ban đầu đến branch chuyển đến.

Tuy nhiên ở branch di chuyển đến, trường hợp có file giống như vậy đã được tiến hành thay đổi cái gì đó thì checkout sẽ thất bại. Khi xảy ra trường hợp này thì sẽ commit lại nội dung thay đổi 1 lần nữa hoặc là sử dụng stash để lưu tạm thời nội dung thay đổi, sau đó phải thực hiện commit.

Giải thích theem về stash, cách sử dụng cụ thể

Stash là khu vực ghi lại tạm thời nội dung thay đổi của file. Bằng việc sử dụng stash, trong work tree và index, những thay đổi chưa được commit có thể lưu lại tạm thời.

Những thay đổi lưu tạm này về sau có thể lấy ra và hiển thị trên branch ban đầu hay là phản ánh lên branch khác.



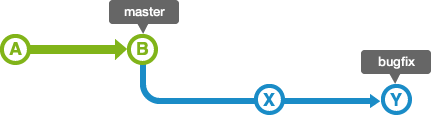
### **3.4, Tích hợp branches**

Branch chủ đề đã hoàn thành công việc, cuối cùng sẽ được tích hợp vào branch tích hợp. Việc tích hợp branch thì có hai phương pháp, phương pháp sử dụng merge và phương pháp sử dụng rebase. Dựa vào việc sử dụng phương pháp nào mà lịch sử của branch sau khi tích hợp sẽ có sự khác biệt lớn.

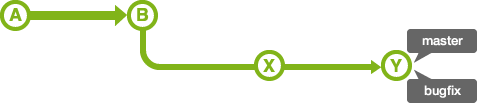
**+ Merge**

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

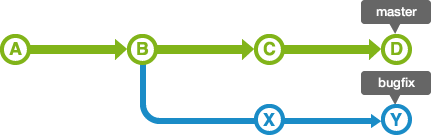
Ví dụ, có branch bugfix phân nhánh ra từ branch master như sơ đồ bên dưới.



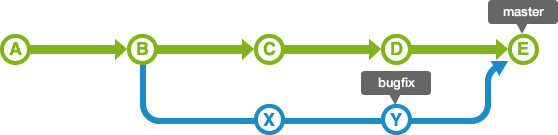
Khi merge branch bugfix này vào branch master, nếu trạng thái branch master không có gì thay đổi thì có thể thực hiện merge hết sức đơn giản. Vì lịch sử của branch bugfix sẽ bao gồm tất cả lịch sử của branch master, nên branch master chỉ có việc di chuyển đơn giản là có thể lấy được nội dung của branch bugfix. Và, người ta gọi merge như thế này là merge fast-forward (chuyển tiếp nhanh).



Nhưng, cũng có trường hợp lịch sử của branch master vẫn đang tiến triển sau khi phân branch bugfix. Trường hợp này thì cần thiết phải tổng hợp nội dung thay đổi của branch master và nội dung thay đổi của branch bugfix thành một.

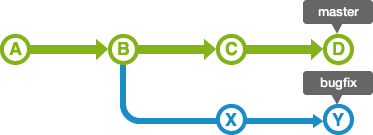


Vì thế, merge commit đã lấy thay đổi của cả hai branch sẽ được tạo ra. Đầu branch master sẽ di chuyển đến commit đó.

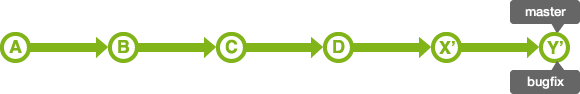


**+ Rebase**

Giống với ví dụ merge, sẽ có branch bugfix phân nhánh từ branch master như sơ đồ bên dưới.

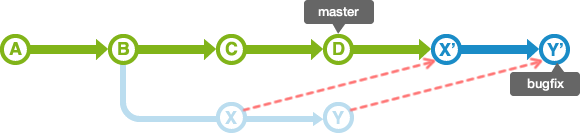


Trường hợp sử dụng rebase ở đây rồi tiến hành tích hợp branch thì lịch sử sẽ giống như sơ đồ tiếp theo. Bây giờ, sẽ trình bày đơn giản trình tự rebase như thế nào.



Trước hết, khi rebase branch bugfix vào branch master, lịch sử branch bugfix sẽ được thay đổi đính kèm sau branch master. Cho nên, lịch sử sẽ thành 1 đường như trong sơ đồ.

Khi này sẽ có trường hợp phát sinh xung đột tại commit di chuyển X và Y. Lúc đó tại từng commit cần thiết phải chỉnh sửa lại những chỗ phát sinh xung đột.



Nếu chỉ rebase không thôi thì vị trí đầu master vẫn cứ như vậy. Vì thế, merge branch bugfix từ branch master rồi di chuyển master đến phần đầu của bugfix.

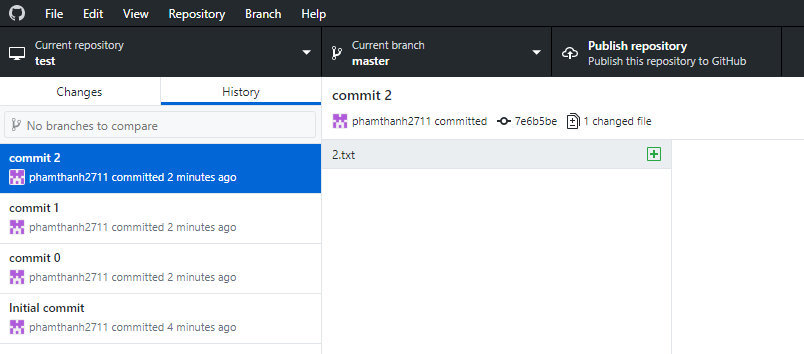
Khi nào nên sử dụng rebase, tại sao không coi việc merge là đủ

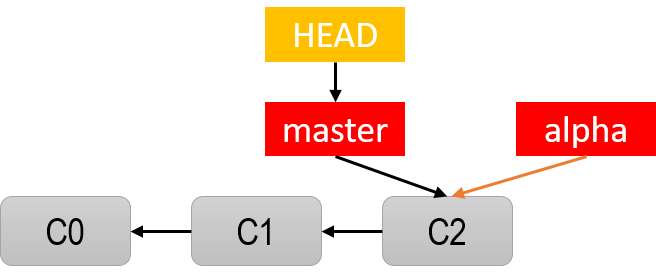


### **3.5, Ví dụ về phân nhánh và gộp nhánh**

**Khởi tạo dự án với 3 commit**

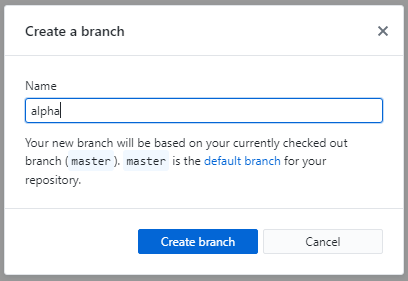
Giả sử tạo repo với tên là test. Lúc này sẽ mặc định sẽ làm việc trên nhánh chính là nhánh master. Sau khi có repo, ta tạo 3 file 0.txt, 1.txt, 2.txt và lần lượt tạo 3 commit đưa lên git.

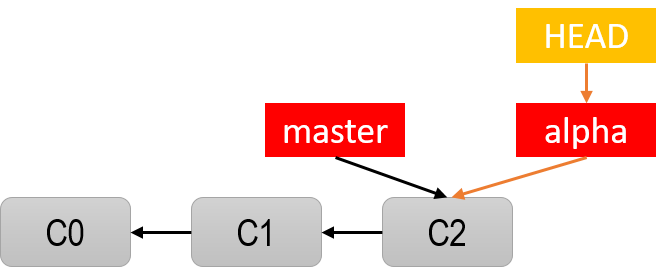




**Tạo một nhánh mới**

Ta tạo một nhánh mới với tên là alpha và chuyển nhánh làm việc sang nhánh alpha

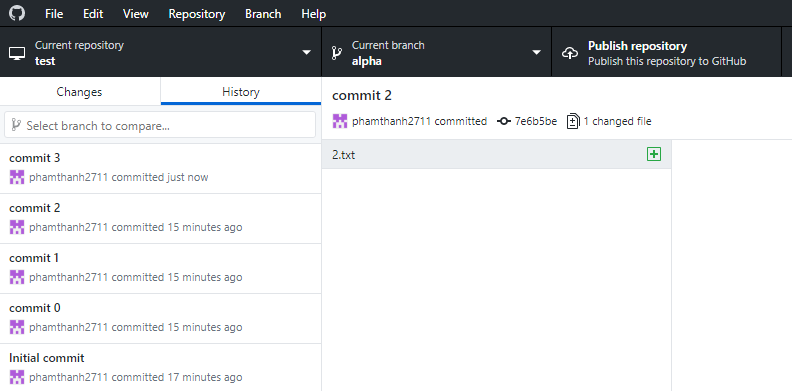


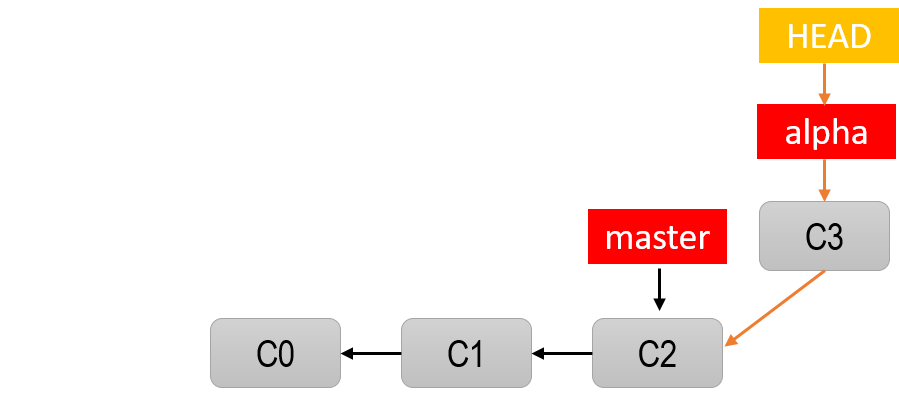


Cả 2 nhánh master và alpha đều trỏ đến commit 2 ( C2 )

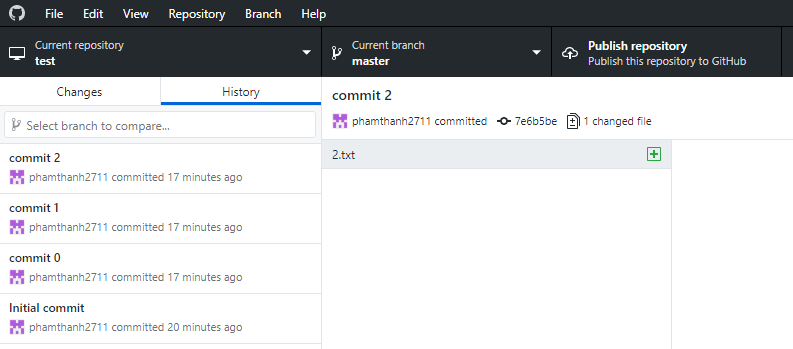
**Tạo sự thay đổi và commit lên nhánh alpha**

Trên nhánh alpha ta thực hiện thay đổi (tạo mới file 3.txt) và hiện tại ở nhánh alpha đã có commit 3 và các commit cũ kế thừa từ nhánh master.



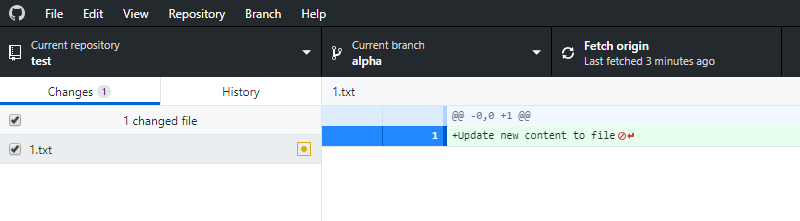


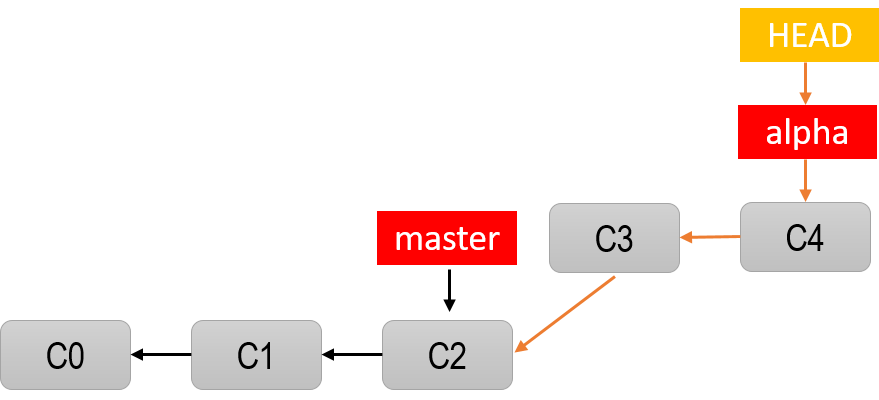
Quay lại nhánh master, ta thấy mới chỉ dừng lại ở commit 2 và các commit cũ hơn , trong này chúng ta sẽ không thấy commit 3 khi tạo ở trên nhánh alpha.



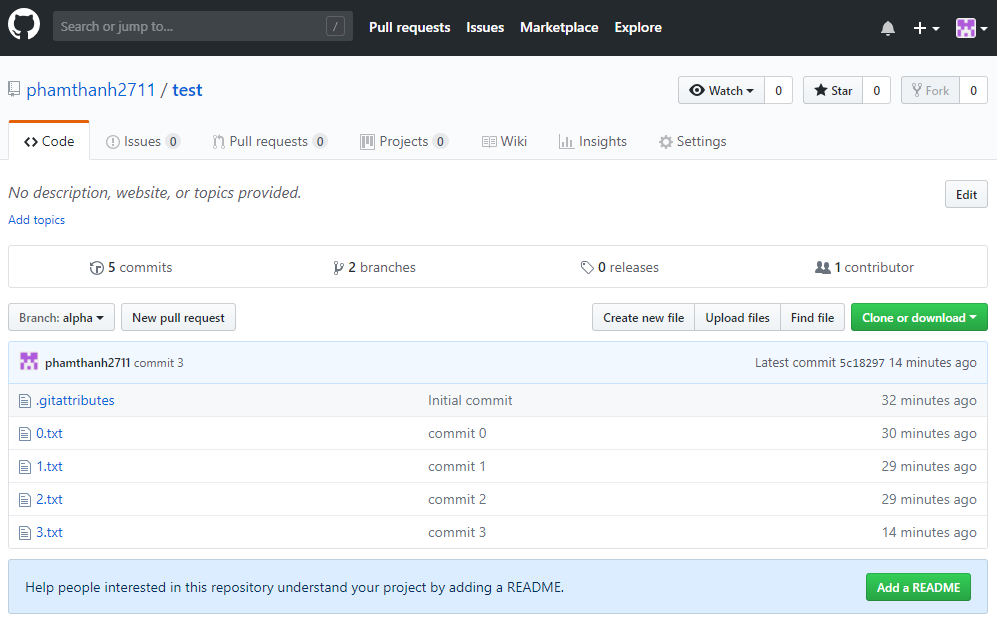
**Tiếp tục tạo sự thay đổi trên nhánh alpha**

Quay lại nhánh alpha, ta tiếp tục thay đổi nội dung file 1.txt (thêm nội dung ‘Update new content to file’) và tạo commit 4 đưa lên nhánh alpha.



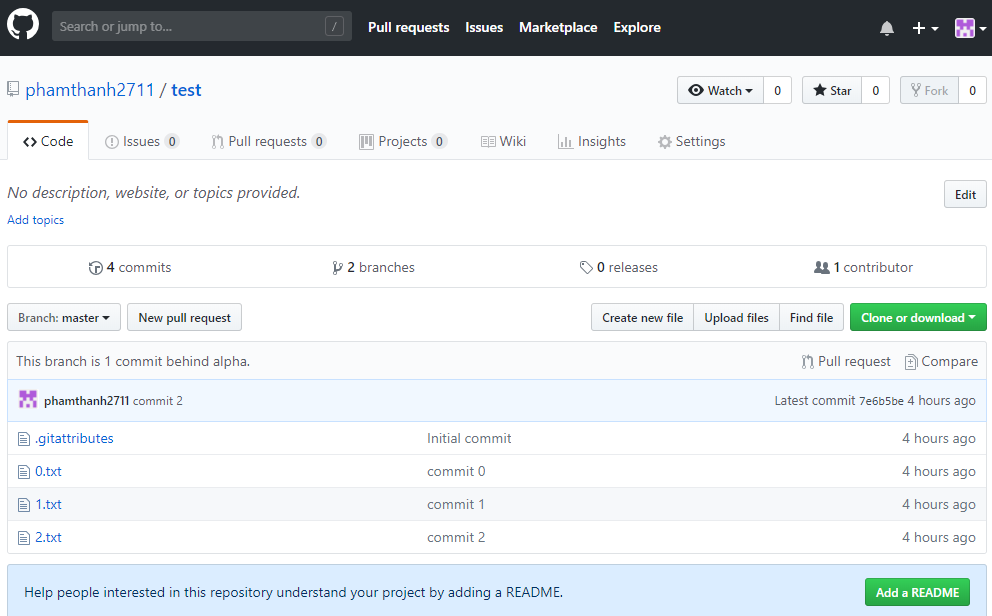


Như vậy ở commit 4, tại nhánh alpha, trong thư mục làm việc có các file: 0.txt, 1.txt, 2.txt, 3.txt, so với master nó có thêm file 3.txt và có chèn thêm nội dung mới trong file 1.txt



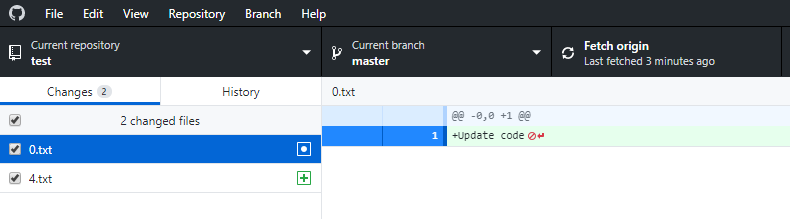
**Quay trở lại làm việc với nhánh master**

Trở về nhánh master, ta thấy có các file 0.txt, 1.txt và 2.txt, trong đó file 1.txt chưa có sự thay đổi nào cả.

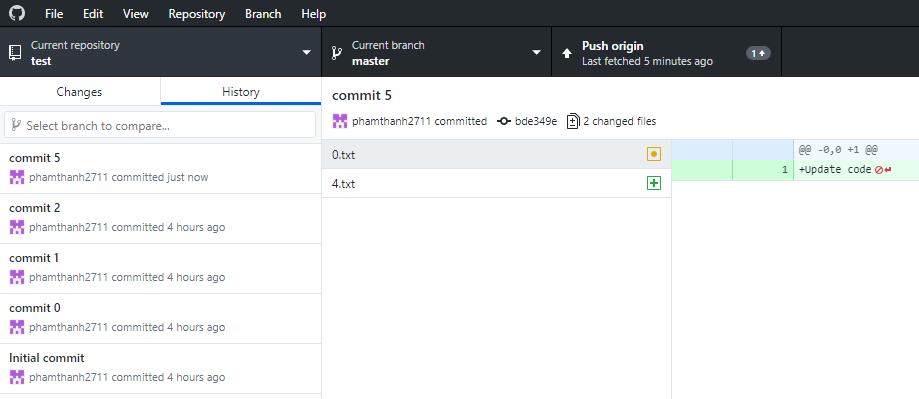


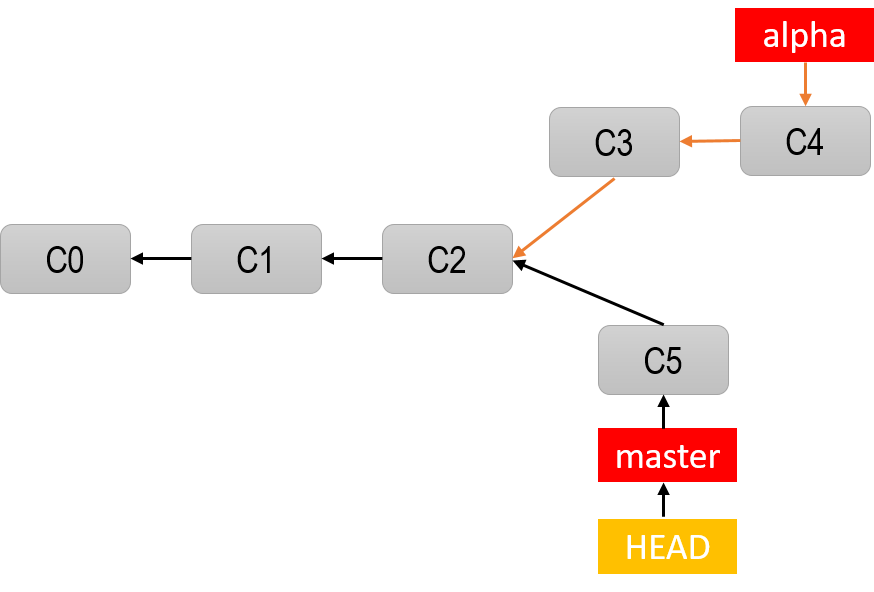
**Tạo sự thay đổi và commit cho nhánh master**

Tại nhánh master, ta bắt đầu tạo sự thay đổi bằng cách tạo thêm file 4.txt và sửa nội dung file 0.txt (thêm nội dung: “Update code” ) .



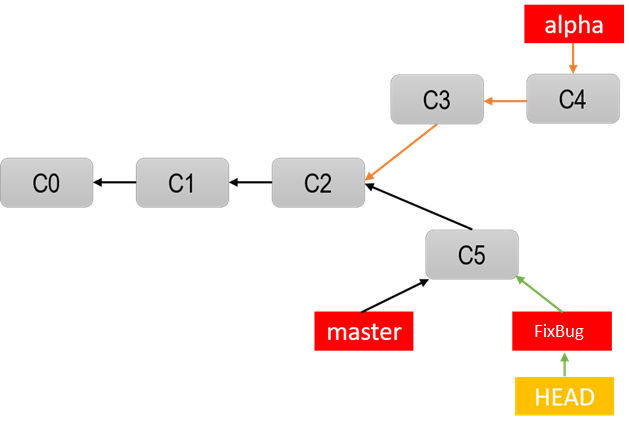
Commit lên nhánh master và đặt tiêu đề là commit 5.





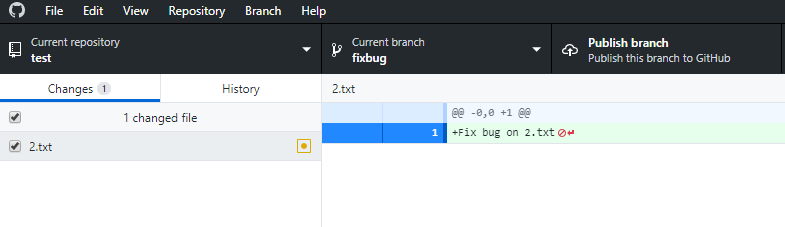
**Tạo nhánh mới rẽ từ nhánh master**

Hiện giờ đang ở nhánh master và phát triển code theo kế hoạch nhưng code trong master cần sửa và kiểm tra ngay. Để quá trình sửa lỗi code không làm ảnh hưởng đến các code hiện tại trong nhánh master thì ta có thể tạo thêm một nhánh mới đặt tên là **fixbug,** sau đó chuyển sang nhánh fixbug để làm việc.

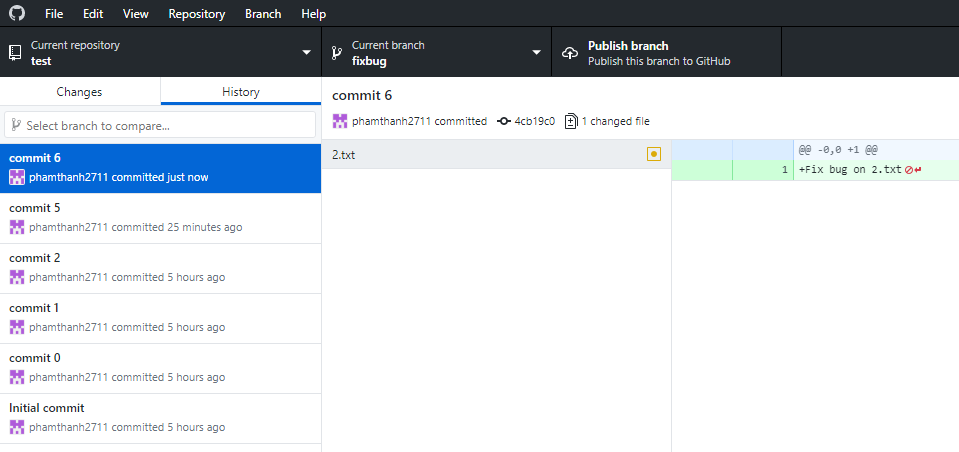


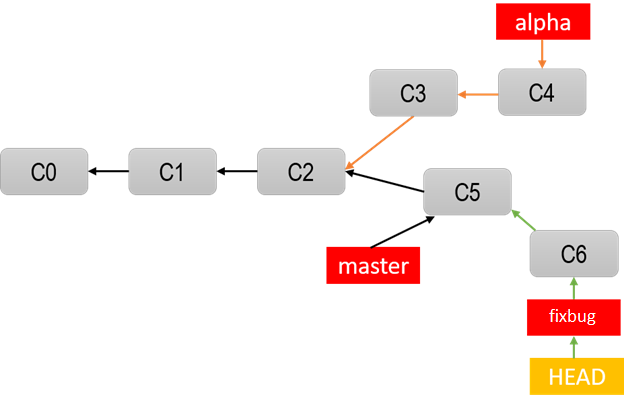
**Tạo sự thay đổi và commit lên nhánh fixbug**

Tạo sự thay đổi trên file 2.txt (thêm nội dung: ‘Fix bug on 2.txt’) và tạo commit 6 lên nhánh fixbug

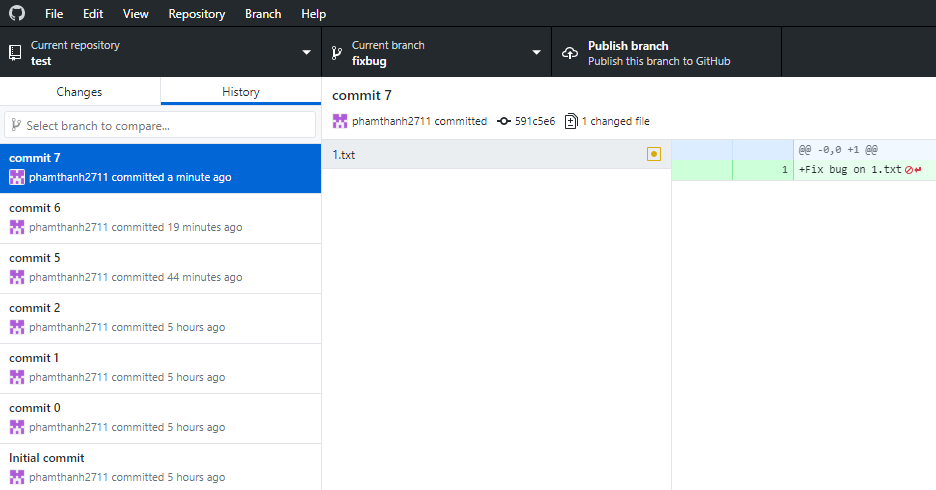


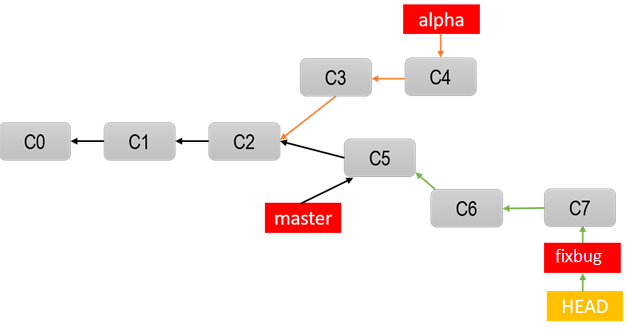
Nhìn vào nhánh fixbug ta sẽ thấy có thêm commit 6 và kế thừa các commit trước kể từ commit 5 của nhánh master





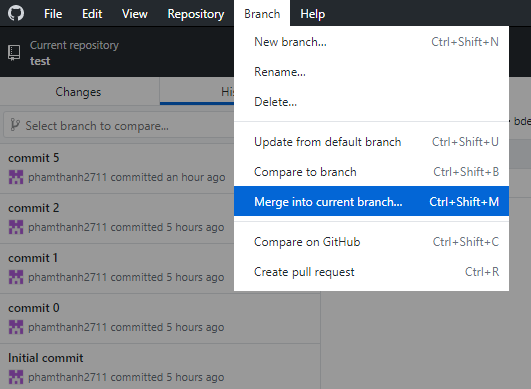
Ta có thể tiếp tục sửa đổi trên file 1.txt (thêm nội dung: ‘Fix bug on 1.txt’) và commit lên nhánh fixbug (commit 7)



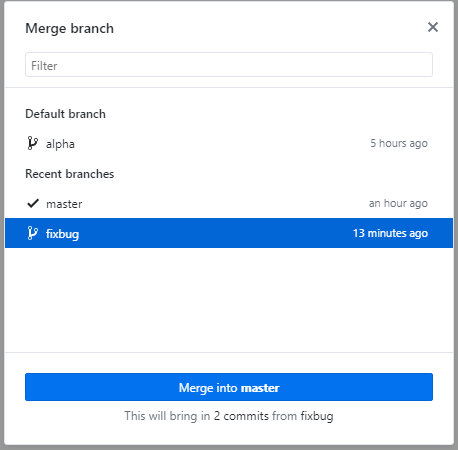


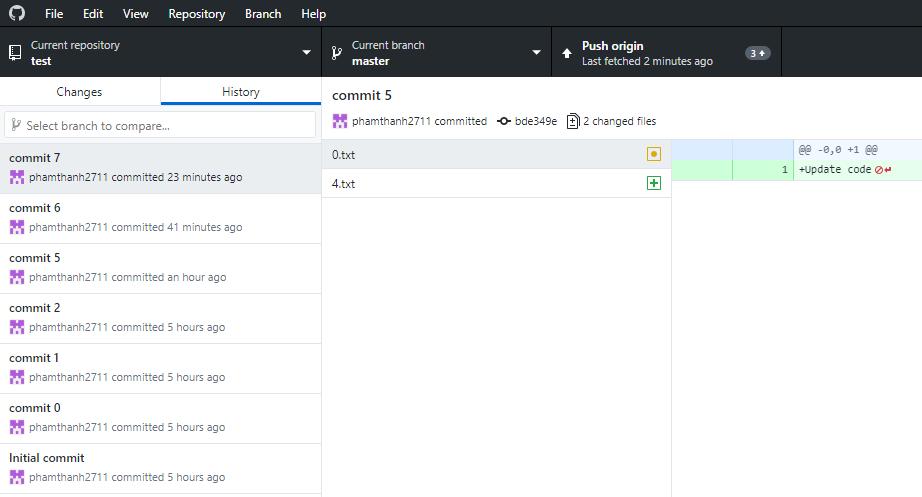
**Trộn nhánh fixbug vào nhánh master**

Sau khi đã hoàn thành nhiệm vụ trên nhánh fixbug nếu muốn các kết quả của nhánh này tích hợp thay đổi vào nhánh master thì tiến hành gộp 2 nhánh này lại ( Merge ). Để làm được điều này, cần phải chuyển về nhánh master rồi cho nhánh master gộp lại với nhánh fixbug bằng chức năng merge.

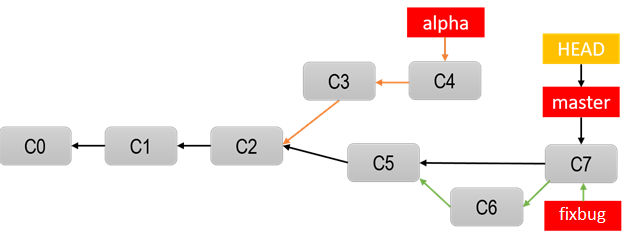


Sau đó chọn nhánh cần merge





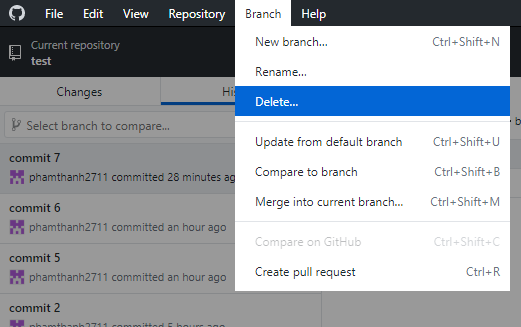
Hiện giờ nhánh master đã gộp vào với nhánh fixbug và cả 2 nhánh đều trỏ vào commit 7 như hình bên dưới



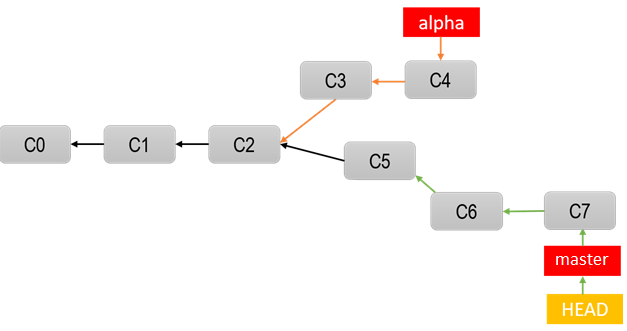
**Xóa nhánh khi không còn sử dụng**

Sau khi nhánh fixbug đã gộp lại vào nhánh master, nếu ta không còn nhu cầu làm việc trên nhánh này thì có thể xóa nhánh này đi.

Ta quay trở lại nhánh fixbug sau đó vào Branch -> Delete.

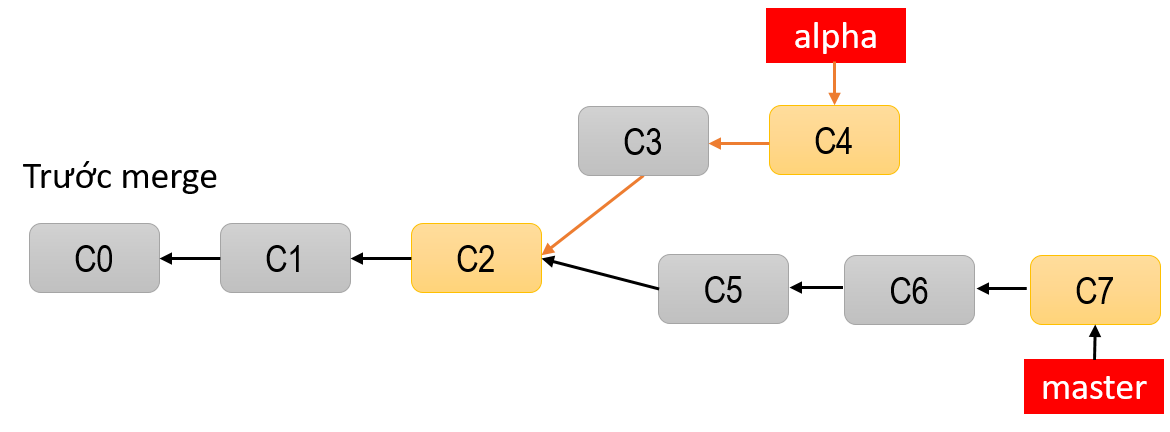


Vậy là giờ nhánh master đã có các commit C0, C1, C2,C5,C6,C7 và hình ảnh trên repo hiện tại như sau:



**Xử lý xung đột khi gộp nhánh**

Bây giờ giả sử có nhu cầu gộp code tại nhánh alpha vào nhánh master, trường hợp này không đơn giản như cách gộp nhánh fixbug ở trên. Do cả 2 nhánh có nhiều commit kể từ thời điểm rẽ nhánh nên khi gộp nó sẽ xem xét sự thay đổi trên cả 2 nhánh tại ba điểm (three-way), thời điểm commit cuối của các nhánh và thời điểm rẽ nhánh, đó là các commit với snapshot C2, C4, C7 như hình dưới:

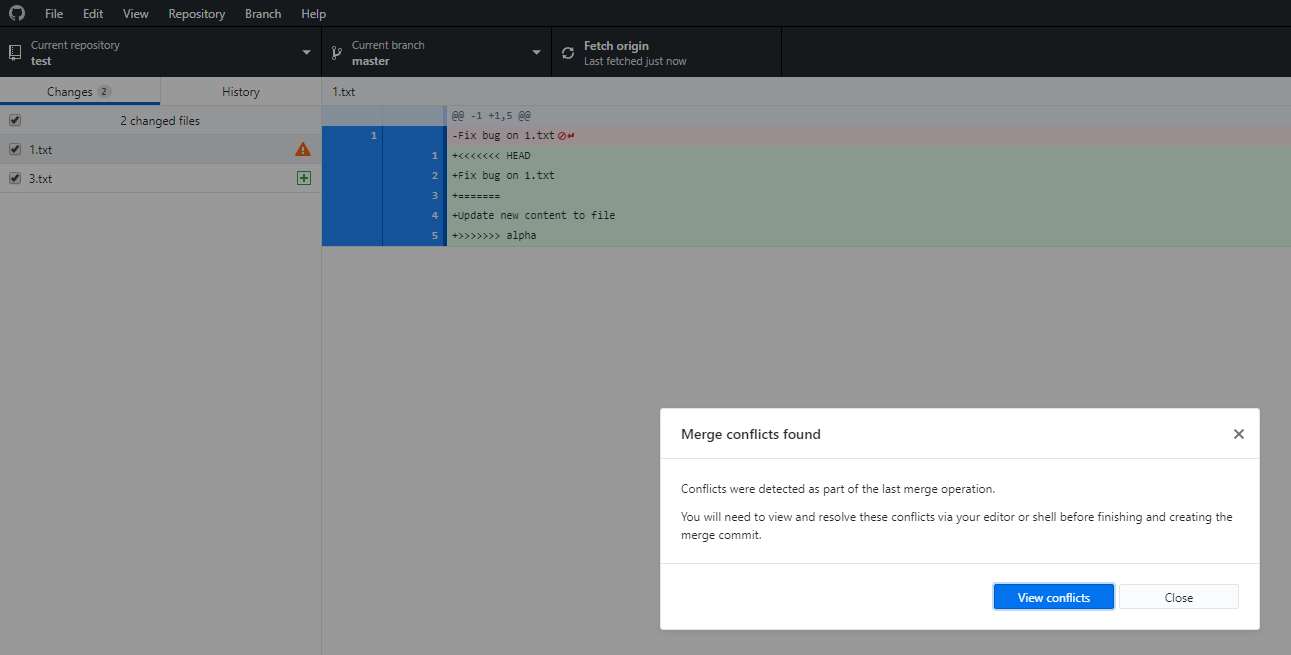


Như hình trên, repo có 2 nhánh và nhánh rẽ bắt đầu từ C2, tức là bắt đầu từ C2 là 2 nhánh bắt đầu có sự thay đổi.

So sánh giữa 2 nhánh master và alpha sau commit cuối ta có bảng so sánh như sau:

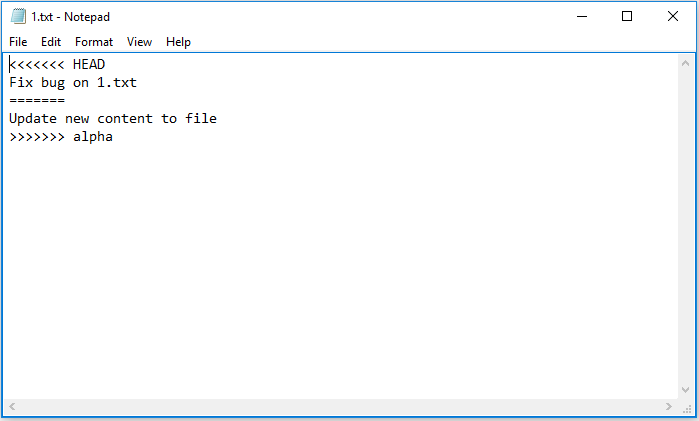
|  |  |  |  |
| --- | --- | --- | --- |
| **File** | **Nhánh master** | **Nhánh alpha** | **Sau khi merge alpha vào master** |
| 0.txt | Có sửa đổi:”Update code” | Không sửa đổi | Không xung đột (lấy theo bản bên nhánh master) |
| 1.txt | Có sửa đổi:”fix bug on 1.txt” | Có sửa đổi:”Update new content to file” | Xung đột vì không biết lấy theo bản ở nhánh nào |
| 2.txt | Có sửa đổi:”fix bug on 2.txt” | Không sửa đổi | Không xung đột (lấy theo bản bên nhánh master) |
| 3.txt | Không sửa đổi | Tạo mới file | Không xung đột (lấy theo bản bên nhánh alpha) |
| 4.txt | Tạo mới file | Không sửa đổi | Không xung đột (lấy theo bản bên nhánh master) |

Theo bảng phân tích trên, sau khi merge nhánh alpha vào nhánh master sẽ xảy ra xung đột ở file 1.txt vì có sự thay đổi trên cả 2 nhánh này và git không biết sẽ lấy sự thay đổi bên nhánh nào.



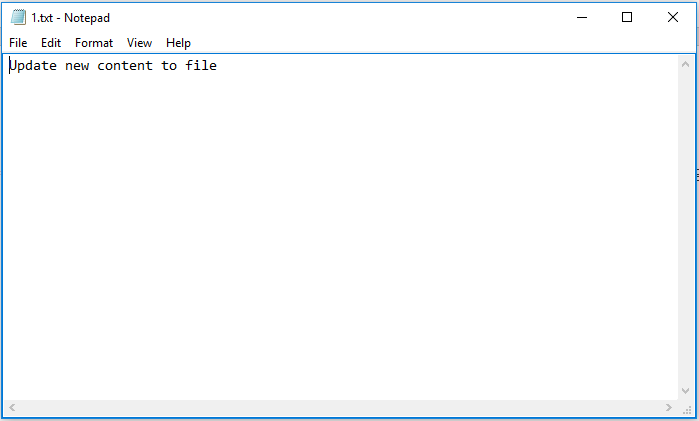
**Xử lý xung đột khi gộp nhánh.**

Ta thấy file xung đột ở đây là file 1.txt và sau khi mở file 1.txt ra chúng ta thấy như sau:

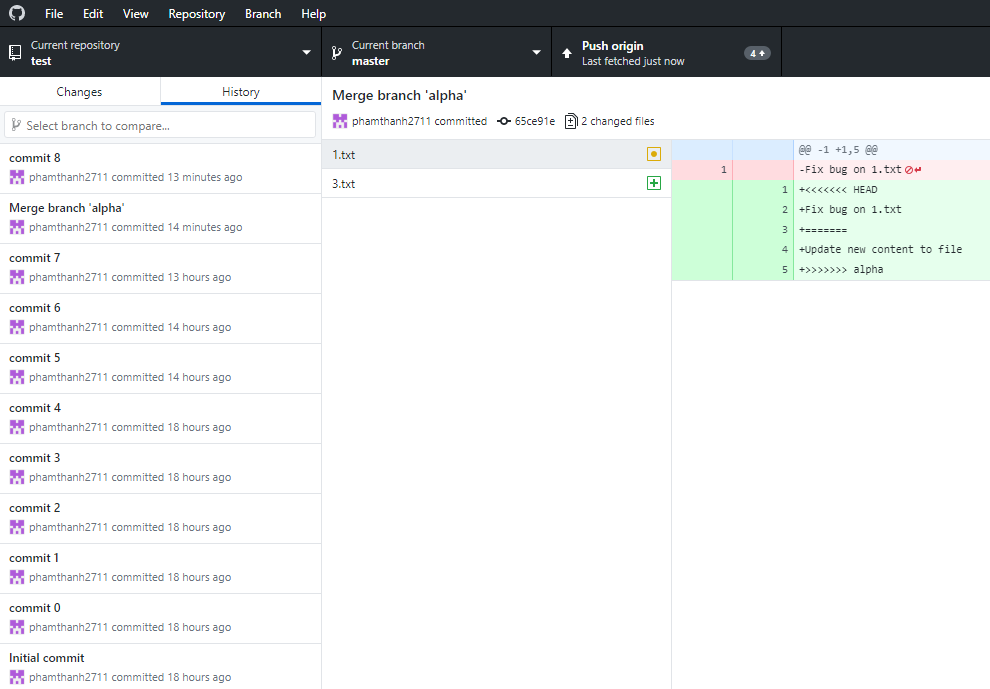


Bạn thấy nội dung xung đột nó được đánh dấu bên trong các ký hiệu. Nội dung nằm giữa <<<<<<< HEAD và ======= là nội dung có từ master, còn nội dung nằm giữa

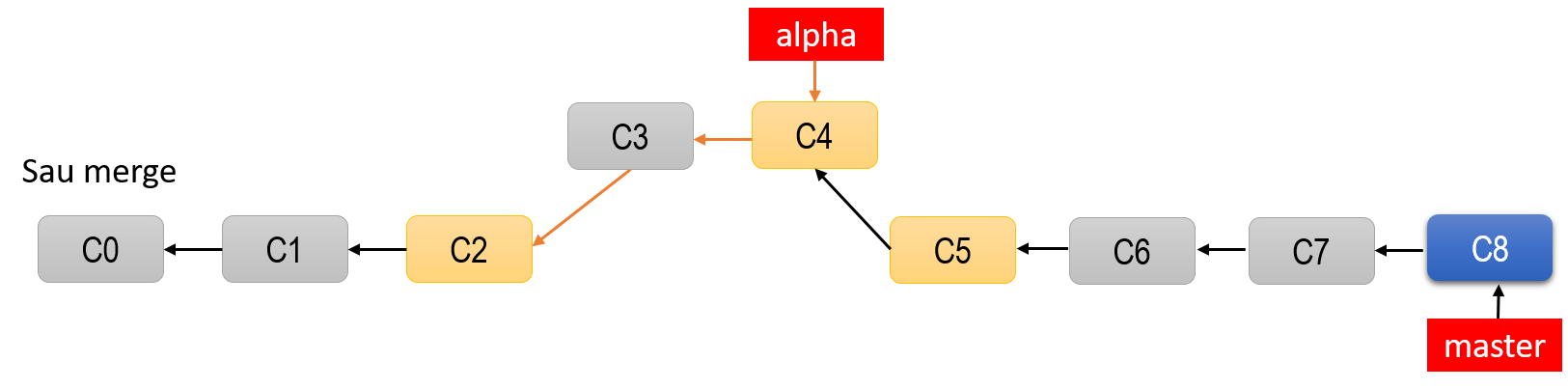
======= và >>>>>>> alpha là nội dung từ alpha. Giữa hai nội dung đó lấy nội dung nào thì giữ, còn lại thì xóa hết. Ví dụ file trên giữ lại nội dung như sau (nội dung từ alpha)



Sau khi xử lý xong xung đột trong file 1.txt ta tạo commit để hoàn tất việc gộp nhánh



Hình ảnh repo sau khi gộp nhánh sẽ như sau:



## **4, Quản lý nhóm làm việc trên github**

### **4.1, Git flow ?**

Git Flow được đưa ra nhằm cải thiện quá trình làm việc cùng Git. Thực chất, đấy là cách chia nhánh và merge nhánh vào khi hoàn thành một tập hợp tính năng hoặc fix.

Git Flow làm việc dựa trên merge, chứ không phải thao tác *cherry-pick* thường thấy.

Git Flow đưa ra các quy ước để triển khai công việc. Mục đích là các nhóm công việc triển khai song song nhưng không ảnh hưởng tới nhau. Các  môi trường development, staging và production tách biệt giúp quá trình kiểm thử (QA), trả lại feedback và xử lý các issue được gọn gàng và thống nhất hơn nhiều.

Ý tưởng của nó thực ra là duy trì các nhánh branch không đổi, tức không xoá (có tính cố định) trong suốt dòng đời sản phẩm. Branch master sẽ luôn là branch chính áp dụng cho production, trong khi các branch hotfix, features hay develop cung cấp các bản để phục vụ QA và hoàn thiện trước khi được đẩy lên master.

Khác với cách thông thường tạo ra nhiều vấn đề xảy ra ngay trên production, thứ mà chúng ta hay gọi là “rút kinh nghiệm từ những sai lầm thực tiễn”, Git Flow đẩy quá trình QA vào một phần bắt buộc cho cả developer và team QA và yêu cầu sự hoàn thiện cao hơn về chất lượng đầu ra.