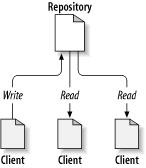
**CHƯƠN I: SUBVERSION**

**I. CÁC KHÁI NIỆM CƠ BẢN**

**1. Giới thiệu chung**

Subversion là một hệ thống tập trung chia sẻ thông tin. Tại cốt lõi của nó là một kho chứa (**Repository Server** ), đó là một cửa hàng trung tâm dữ liệu. Kho lưu trữ thông tin trong các hình thức của một *cây hệ thống tập tin*- Một hệ thống cấp bậc của các tập tin và thư mục. Số lượng bất kỳ các chương trình clientkết nối với kho lưu trữ, và sau đó đọc hoặc viết vào những tập tin này. Bằng cách ghi dữ liệu, chương trình client đưa thông tin có sẵn cho những người khác; bằng cách đọc dữ liệu, chương trình client nhận được thông tin từ những người khác.

  
**Hình 1.1. Một hệ thống Máy Khách/Máy Chủ điển hình**

Điều này nghe giống như định nghĩa của một máy chủ tập tin thông thường. Và quả thực, các kho lưu trữ làmột loại máy chủ tập tin, nhưng nó không giống loại bình thường của chúng ta. Những gì làm cho kho lưu trữ Subversion đặc biệt là nó nhớ mọi sự thay đổitừng được viết vào nó: mỗi lần thay đổi cho mỗi tập tin, và thậm chí thay đổi cây thư mục, chẳng hạn như việc bổ sung, xóa, và sắp xếp lại các tập tin và thư mục.

**2. Giải pháp Lock-Modify-Unlock**

Giả sử chúng ta có hai thành viên trong nhóm chỉnh sửa các tập tin kho lưu trữ tại cùng một thời gian. Nếu thành viên 1 lưu thay đổi của mình vào kho lưu trữ đầu tiên, có thể một vài khoảnh khắc sau đó thành viên 2 vô tình có thể ghi đè lên chúng với phiên bản mới của tập tin. Các thay đổi tập tin do thành viên 1 đã thực hiện sẽ không có mặt trong phiên bản mới hơn của tập tin do thành viên 2 chỉnh sửa, bởi vì thành viên 2 không thấy được những thay đổi mà thành viên 1 đã chỉnh sửa. Điều này chắc chắn là một tình huống mà chúng ta muốn tránh.

Nhiều phiên bản kiểm soát hệ thống sử dụng một mô hình Lock-Modify-Unlock (khoá-thay đổi-mở khóa) để giải quyết vấn đề này, đây là một giải pháp rất đơn giản. Trong hệ thống như vậy, kho lưu trữ chỉ cho phép một người thay đổi một tập tin tại một thời điểm. Đầu tiên thành viên thứ 1 phải khóa các tập tin trước khi có thể bắt đầu thay đổi nó. Khóa một tập tin là rất giống như mượn một cuốn sách từ thư viện, nếu thành viên thứ 1 đã khóa một tập tin, thì sau đó thành viên thứ 2 không thể thực hiện bất kỳ thay đổi với nó. Nếu thành viên thứ 2 cố gắng để khóa các tập tin, các kho lưu trữ sẽ từ chối yêu cầu. Tất cả những gì có thể làm là đọc các tập tin, và chờ đợi cho thành viên thứ 1 kết thúc những thay đổi của mình và mở khóa của mình. Sau khi thành viên thứ 1 mở các tập tin, hết lượt của mình, sau đó thành viên thứ 2 có thể thực hiện công việc của mình của mình bằng cách khóa và chỉnh sửa.

Vấn đề với mô hình Lock-Modify-Unlock là nó có một chút hạn chế, và thường trở thành một rào cản cho người sử dụng:

- Khóa có thể gây ra các vấn đề hành chính. Đôi khi thành viên nào đó sẽ khóa một tập tin và sau đó quên nó. Trong khi đó, thành viên khác vẫn đang chờ đợi để chỉnh sửa các tập tin. Và nếu thành viên đó đi nghỉ thì các thành viên khác phải nhờ một quản trị viên để mở khoá. Sẽ gây ra sự chậm trễ không cần thiết và lãng phí thời gian.

Khóa có thể gây ra tuần tự không cần thiết. Nếu thành viên thứ 1 muốn chỉnh sửa phần đầu của một tập tin văn bản, và thành viên thứ 2 chỉ đơn giản là muốn chỉnh sửa cuối cùng một tập tin. Những thay đổi này không chồng chéo lên nhau chút nào. Họ có thể dễ dàng chỉnh sửa các tập tin cùng một lúc, và không có hại lớn, giả sử những thay đổi sáp nhập với nhau đúng cách. Không cần cho họ phải thay phiên nhau trong tình huống này.

Khóa có thể tạo ra một cảm giác an toàn sai lầm. Ví dụ thành viên thứ 1 khóa và chỉnh sửa tập tin A, trong khi đồng thời thành viên thứ 2 khóa và chỉnh sửa các tập tin B. Nhưng giả sử rằng A và B phụ thuộc vào nhau, và các thay đổi được thực hiện cho từng ngữ nghĩa không tương thích. Đột nhiên A và B không làm việc cùng nhau nữa. Hệ thống khóa không có khả năng để ngăn chặn vấn đề nhưng nó vẫn đưa ra cho các thành viên cảm giác an toàn sai. Các thành viên sẽ nghĩ rằng các tập tin khóa, mỗi người đều thực hiện công việc một cách an toàn, cách biệt và do đó tạo ra những thay đổi không tương thích giữa các phiên bản.

**3. Giải pháp Copy-Modify-Merge**

Subversion, CVS và các hệ thống kiểm soát phiên bản khác sử dụng mô hình Copy-Modify-Merge (sao chép-sửa đổi-hợp nhất) như một giải pháp thay thế cho việc khóa. Trong mô hình này, mỗi khách hàng của người dùng đọc kho lưu trữ và tạo một bản sao làm việc cá nhân của tệp hoặc dự án. Người dùng sau đó làm việc song song, sửa đổi các bản sao riêng tư của họ. Cuối cùng, các bản sao riêng tư được hợp nhất với nhau thành một phiên bản cuối cùng mới. Hệ thống kiểm soát phiên bản thường hỗ trợ việc hợp nhất, nhưng cuối cùng con người chịu trách nhiệm làm cho việc hợp nhất diễn ra chính xác.

Ví dụ: Thành viên 1 và thành viên thứ 2 đều tạo ra các bản sao làm việc cùng một dự án, sao chép từ kho. Họ làm việc đồng thời, và thay đổi cùng một tập tin A trong bản sao của họ. Thành viên thứ 1 lưu thay đổi của mình vào kho lưu trữ đầu tiên. Khi thành viên thứ 2 cố gắng để lưu các thay đổi của anh ta, kho lưu trữ thông báo rằng tập tin của anh ta quá cũ . Nói cách khác, tập tin A đó trong kho lưu trữ bằng cách nào đó thay đổi kể từ khi anh ta mới sao chép nó. Vì vậy, thành viên thứ 2 yêu cầu với thành viên thứ 1 để hợp nhất bất kỳ thay đổi mới nhất từ ​​kho vào bản sao hoạt động của tập tin A. Nếu những thay đổi của thành viên thứ 1 không chồng chéo lên nhau với bản của anh ta, anh ta sẽ có cả bộ những thay đổi tích hợp, sau đó thành viên này lưu bản sao làm việc của mình trở lại vào kho.

Nhưng điều gì sẽ xảy ra nếu thay đổi của thành viên thứ 1 làm trùng với thay đổi của thành viên thứ 2? Tình trạng này được gọi là một xung đột , Và nó thường không phải là một vấn đề lớn. Khi thành viên thứ 2 yêu cầu thành viên thứ 1 hợp nhất các thay đổi kho lưu trữ mới nhất vào bản sao làm việc của mình, bản sao của tập tin A là bằng cách nào đó gắn cờ là trong trạng thái xung đột: anh ta sẽ có thể nhìn thấy cả hai bộ những thay đổi xung đột, và tự lựa chọn giữa chúng . Lưu ý rằng phần mềm có thể không tự động giải quyết xung đột, con người chỉ có khả năng hiểu biết và thực hiện những lựa chọn thông minh cần thiết. Khi thành viên thư 2 đã giải quyết thủ công những thay đổi chồng chéo (có lẽ bằng cách thảo luận về cuộc xung đột với thành viên thứ 1), anh ta có thể an toàn lưu các tập tin sau khi sáp nhập trở lại vào kho.

Mô hình sao chép, sửa đổi, sáp nhập có thể nghe một chút hỗn loạn, nhưng trong thực tế, nó chạy rất trơn tru. Người dùng có thể làm việc song song, không bao giờ chờ đợi nhau. Khi họ làm việc trên các tập tin tương tự, nó hóa ra rằng hầu hết các thay đổi đồng thời của họ không chồng chéo lên nhau ở tất cả các xung đột hiếm khi xảy ra. Và số lượng thời gian cần thiết để giải quyết xung đột là ít hơn so với thời gian đã mất bằng một hệ thống khóa.

Có một trường hợp nơi mô hình khóa-sửa đổi-mở khóa cho kết quả tốt hơn, đó là khi mà bạn có các tập tin không hợp nhất được. Ví dụ, nếu kho lưu trữ của bạn có chứa một số hình ảnh đồ họa, và hai người thay đổi hình ảnh tại cùng một thời gian, không có cách nào cho những thay đổi được sáp nhập với nhau. Một trong hai sự thay đổi sẽ bị mất.

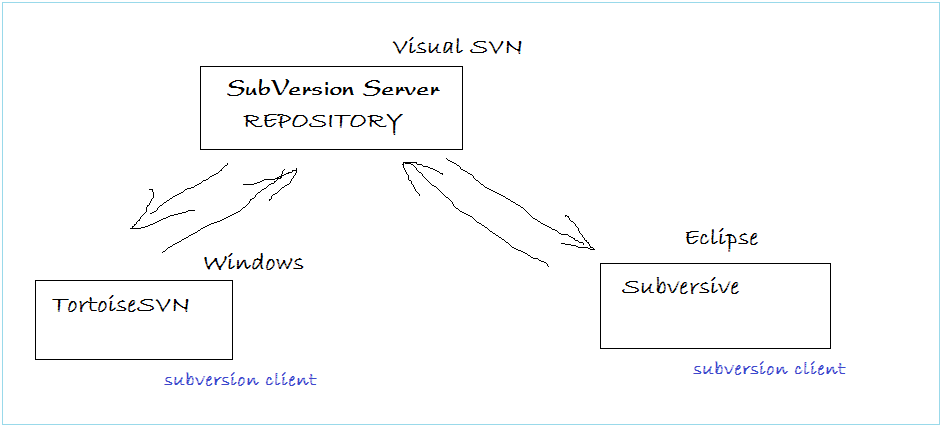
**II. HƯỚNG DẪN DOWNLOAD, CÀI ĐẶT VÀ LÀM VIỆC VỚI VISUAL SVN**

**1. Giới thiệu**

**Visual SVN** sử dụng như một **Repository Server** (Kho chứa). Dùng để lưu trữ các file dữ liệu chia sẻ giữa các thành viên trong nhóm. **Visual SVN** được cài trên 1 máy chủ, các thành viên trong nhóm sẽ cài đặt chương trình **client**(**subversion client**), có nhiều chương trình **Subversion client**, được nhiều hãng khác nhau phát triển và cơ bản là đều miễn phí.

*Đối với các lập trình viên****Java****bạn có thể cài****Subversive****vào****Eclipse****, bạn có thể dễ dàng chia sẻ và lấy dữ liệu project từ****SVN Repository.***

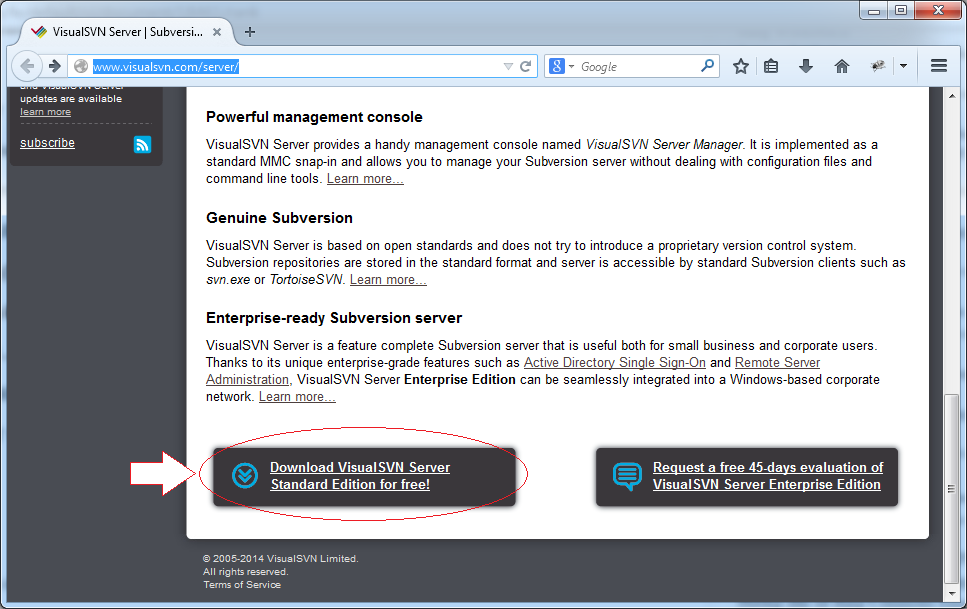
[Cài đặt Subversive cho Eclipse](https://openplanning.net/10111/cai-dat-subversive-cho-eclipse)

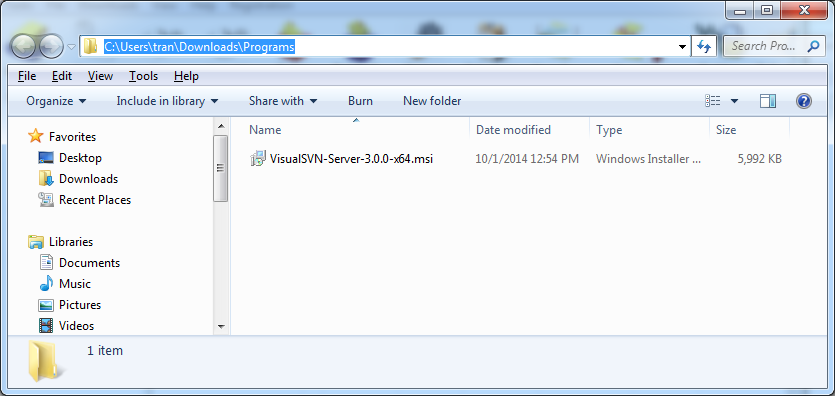


## 2. Download VisualSVN

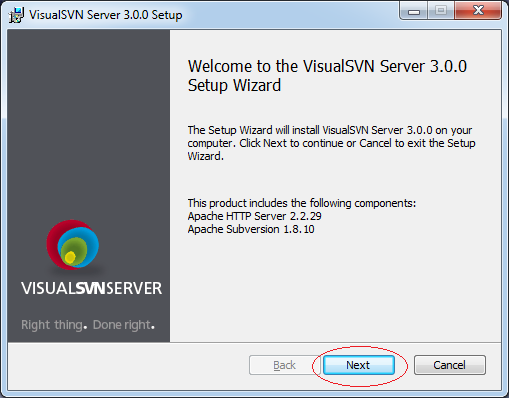
Truy cập vào URL:

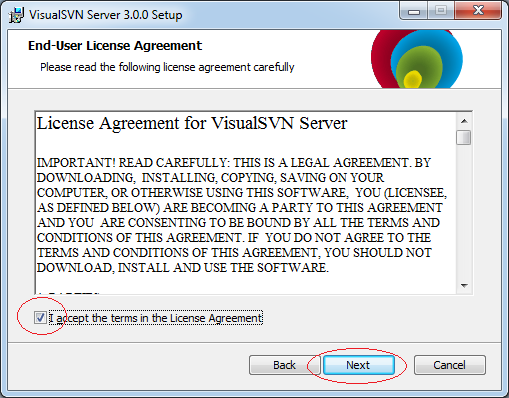
* <http://www.visualsvn.com/server/>

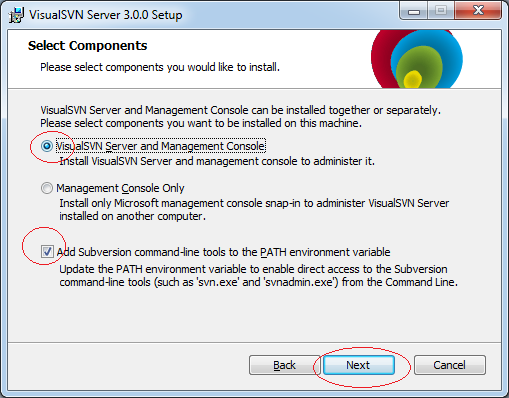




## 3. Cài đặt Visual SVN



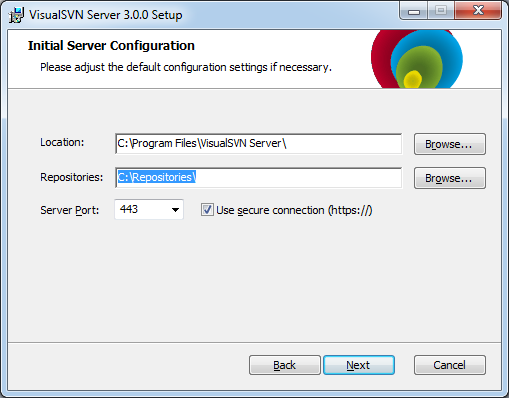




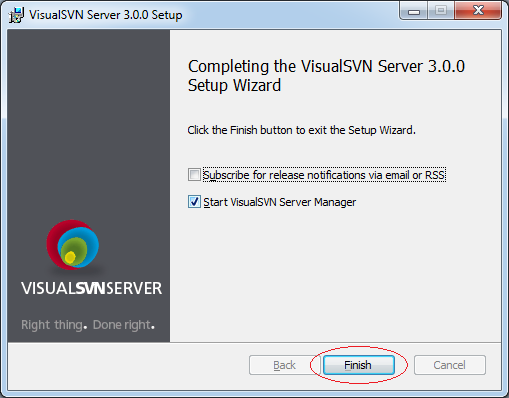


Chọn thư mục để cài đặt, chẳng hạn: **C:\Program Files\VisualSVN Server**

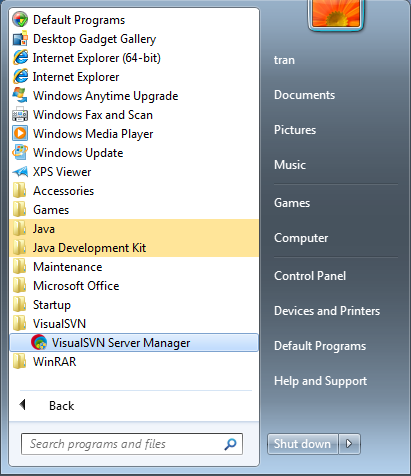
Chọn vị trí cho thư mục có chứa dữ liệu (Thông thường, bạn nên đặt vào 1 ổ cứng an toàn): C:\Repositories



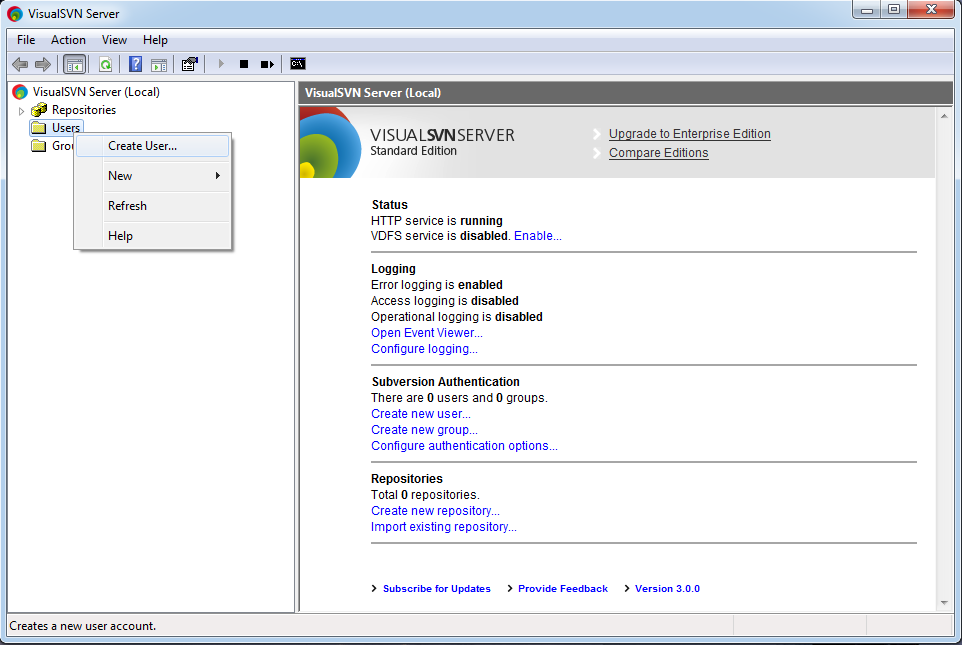
Nhấn **Finish**để hoàn thành.



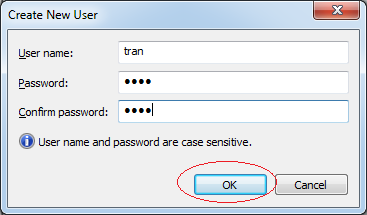
## 4. Quản lý và tạo Repository



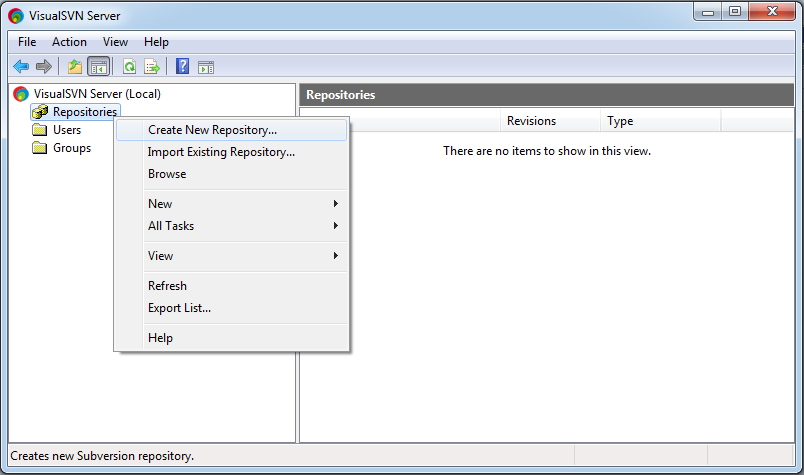
Đây là hình ảnh khi **Visual SVN** đã được chạy. Chúng ta tạo mới một **user**.

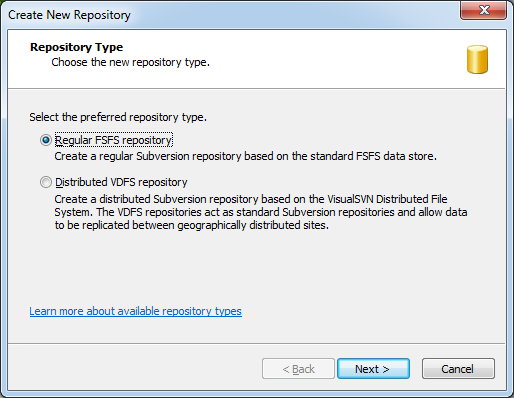


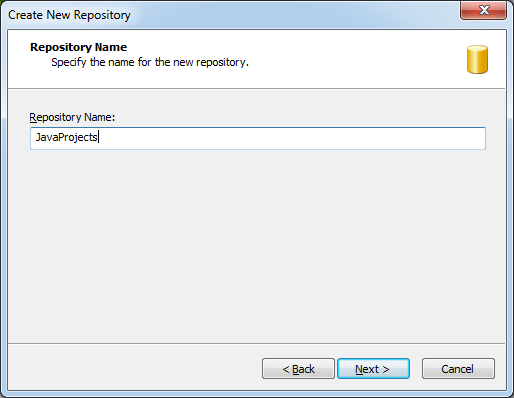
Nhập thông tin **username/password**



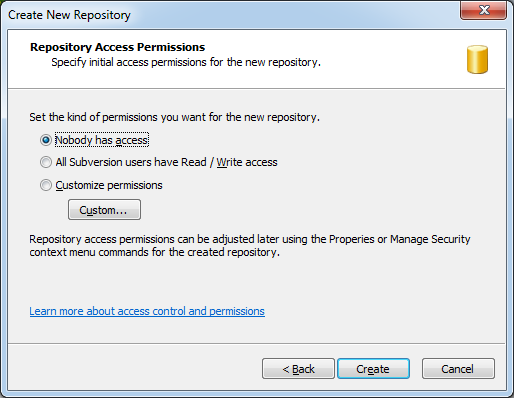
Bước tiếp theo chúng ta tạo mới một **Repository**với tên **JavaProjects**. Đây là một **Repository**rỗng chưa có dữ liệu, dữ liệu sẽ được đẩy lên **server**từ một **subversion client** (bởi một thành viên trong nhóm lập trình). Và được các thành viên khác lấy về. Các thành viên có thể sửa đổi dữ liệu và đẩy lên **Repository**.

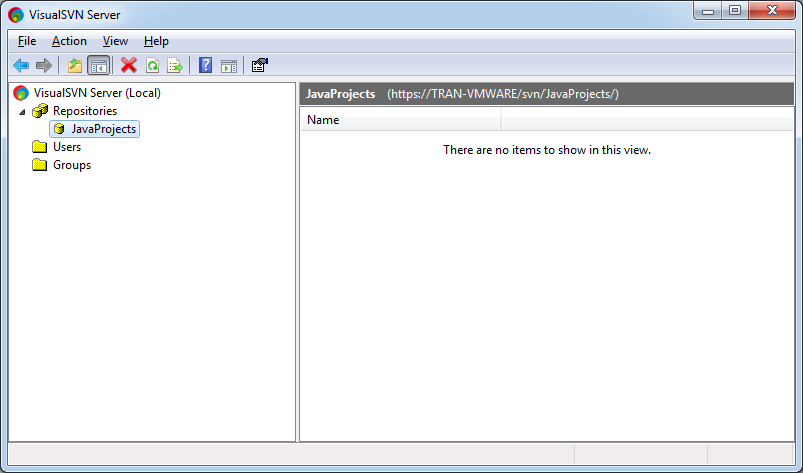




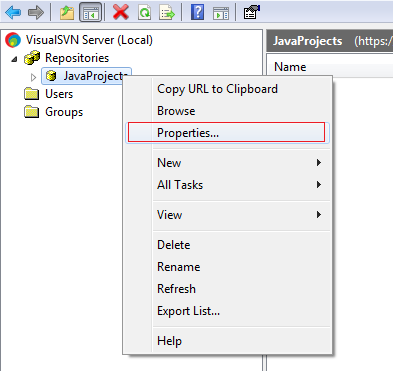


Mặc định không ai được phép truy cập vào **Repository** này, chúng ta sẽ phân quyền sau.

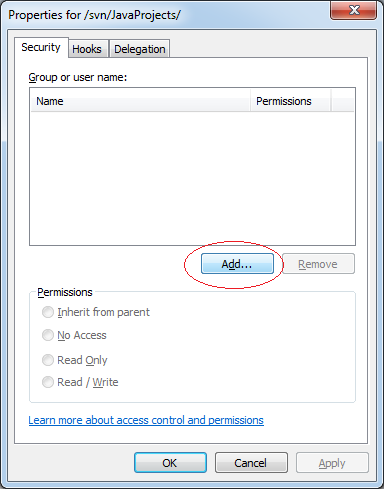


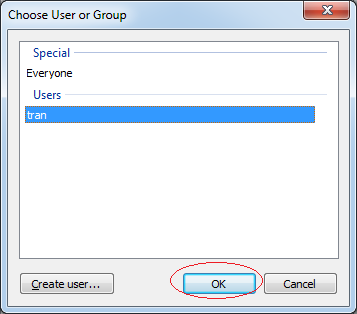


Nhấn phải chuột vào **Repository "JavaProjects"** và chọn **Properties**

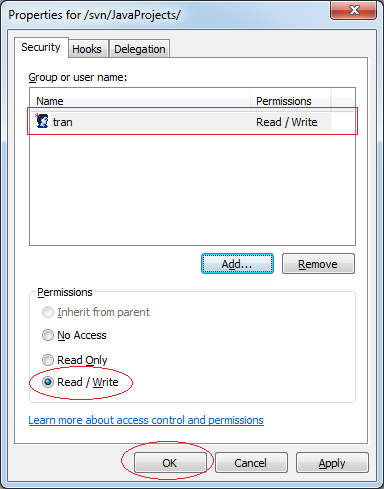


Nhấn **Add** để thêm **user** có quyền truy cập vào **Repository** này.

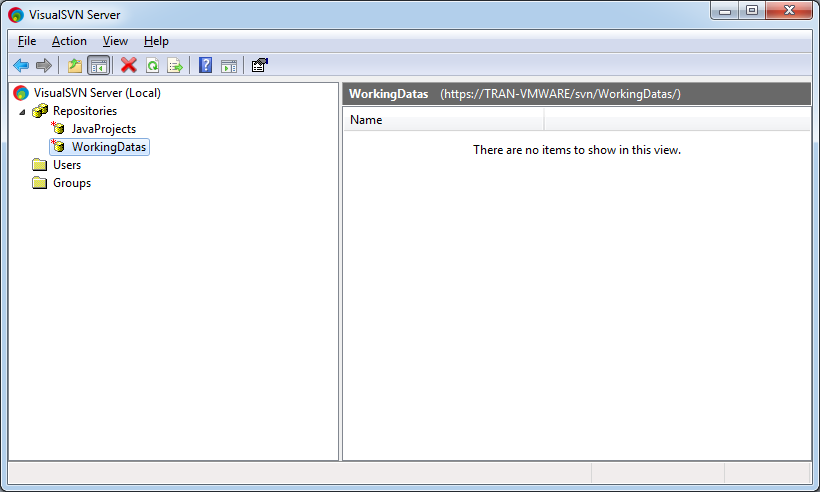




Lựa chọn quyền hạn của **User**.



Tương tự bạn có thể tạo thêm một **Repository**khác chẳng hạn ***WorkingDatas***

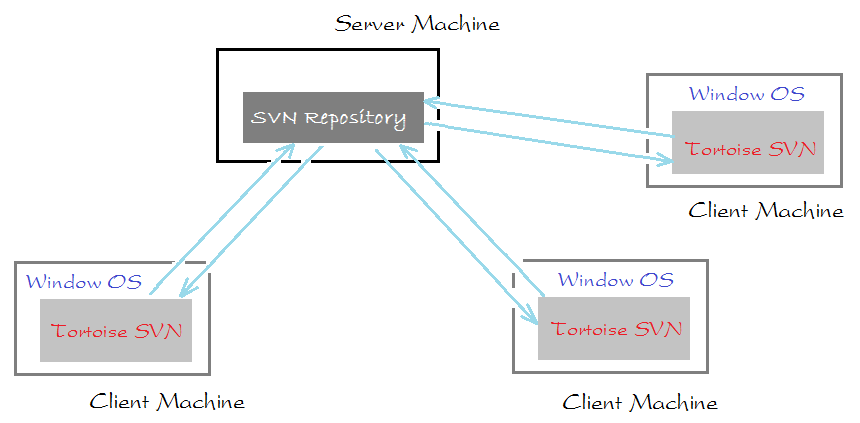


**II. HƯỚNG DẪN DOWNLOAD CÀI ĐẶT VÀ LÀM VIỆC VỚI TORTOISE SVN**

## 1. Tortoise SVN là gì?

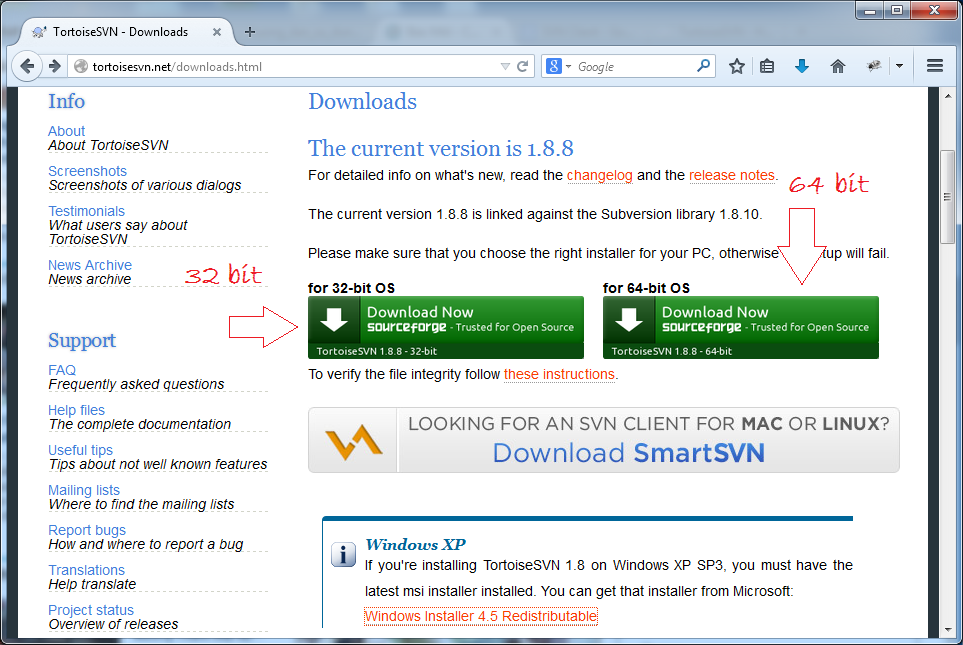
**Tortoise SVN** là một chương trình **SVN Client** nó được cài đặt tại các máy tính của các thành viên trong một nhóm, giúp dễ dàng chia sẻ dữ liệu lên **SVN Server** và lấy dữ liệu về từ **SVN Server**.

Đây là mô hình làm việc giữa **Tortoise SVN** và **SVN Repository**:



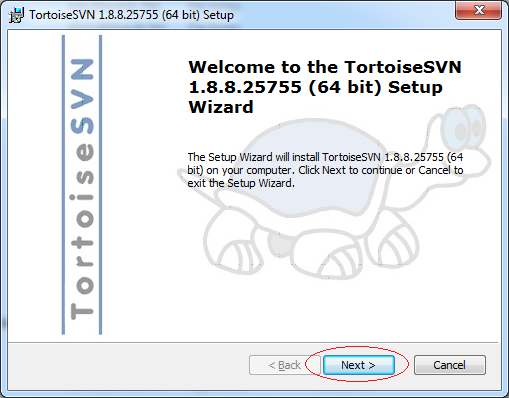
## 2. Download Tortoise SVN

Truy cập trang: <http://tortoisesvn.net/downloads.html>

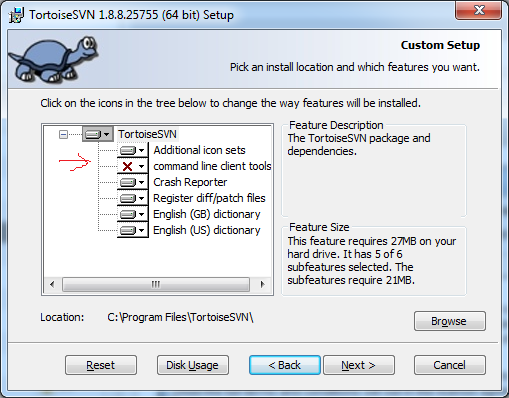


Tùy thuộc vào hệ điều hành của bạn là **64**hay **32 bit** để download cho đúng.

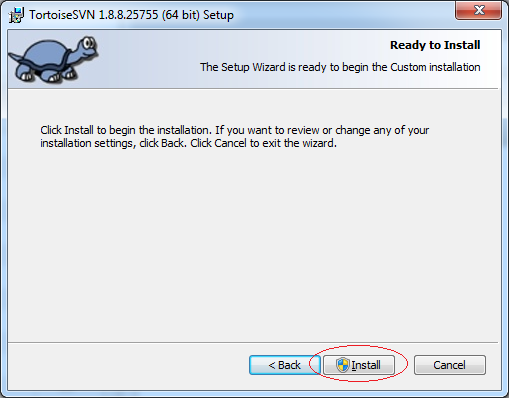
## 4. Cài đặt Tortoise SVN

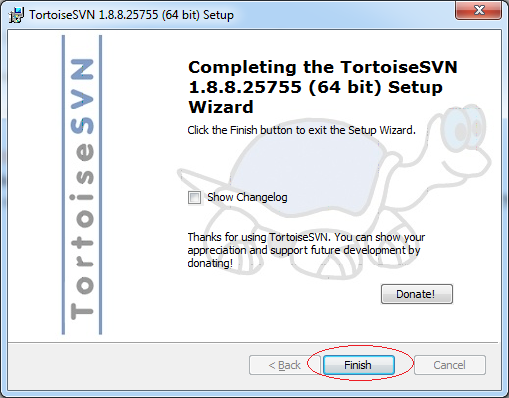












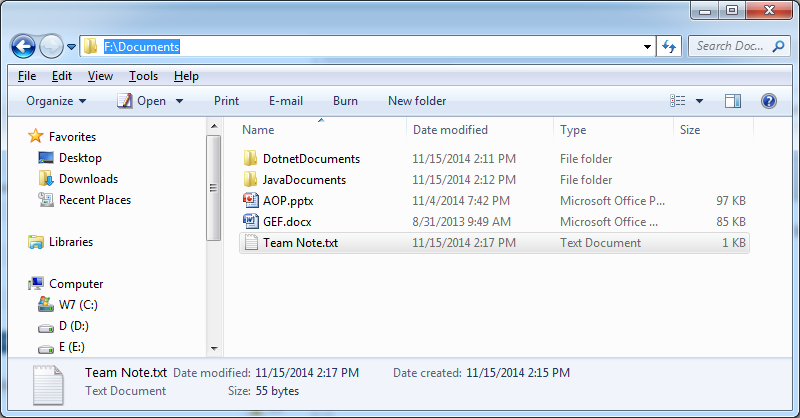
Việc cài đặt thành công, bạn cần phải khởi động lại máy tính.

## 5. Bắt đầu với Tortoise SVN

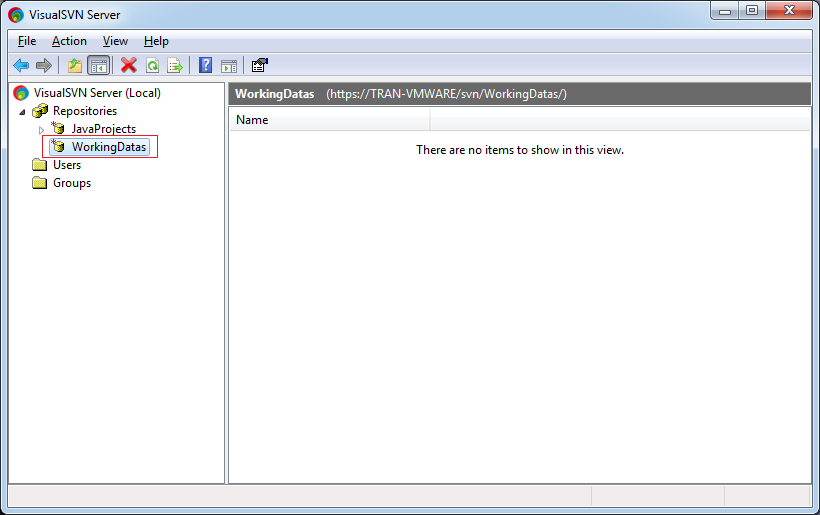
### 5.1. Import dữ liệu lên Repository

Trước hết cần một thành viên trong nhóm chia sẻ dữ liệu  ban đầu lên **SVN Repository**. Nếu không có dữ liệu gì thì có thể bỏ qua bước này.

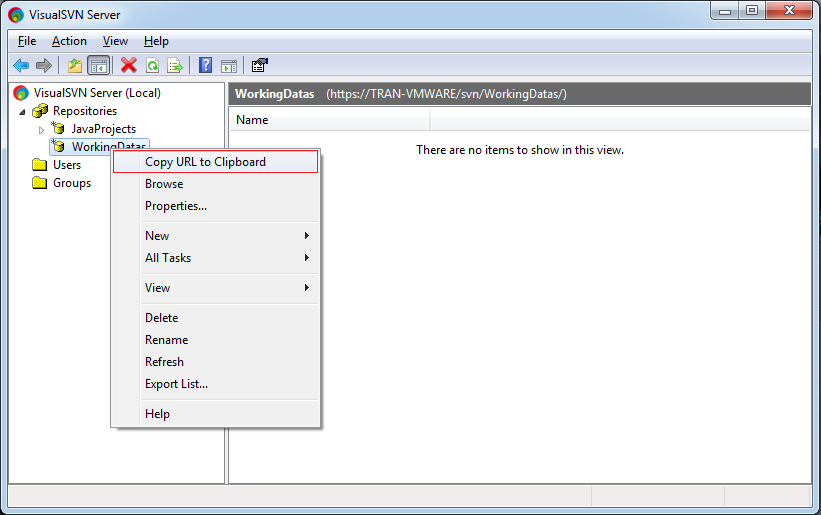
Giả sử tôi **import**(nhập khẩu) các file dữ liệu trong thư mục **Documents** vào **SVN Repository**:



Trên **Visual SVN** đã tạo sẵn một **Repository**có tên **WorkingDatas.**



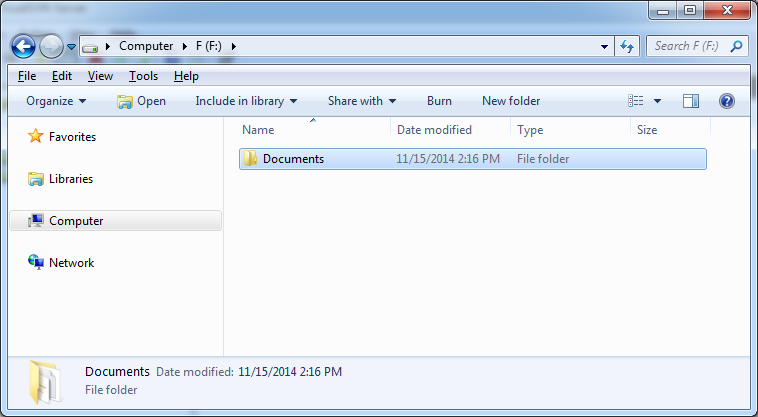
Copy URL:



URL mà bạn copy được có thể chứa tên của máy chủ SVN.

* https://TRAN-VMWARE/svn/WorkingDatas/

Bạn có thể thay đổi tên máy chủ bằng địa chỉ IP.



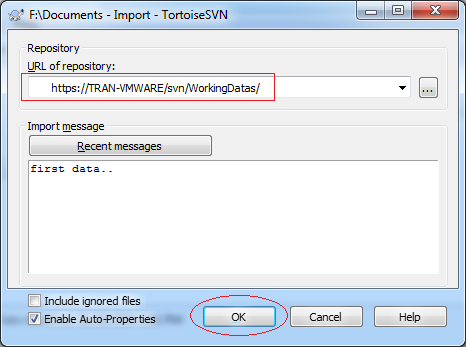
Nhấn phải chuột vào thư mục **Documents**, chọn:

* **TortoiseSVN/Import...**

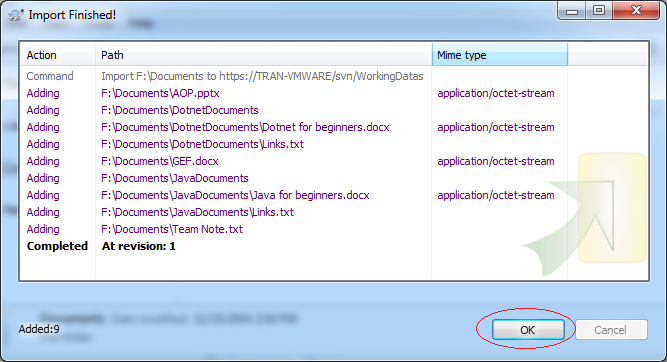


Nhập vào **URL**của **SVN Repository**.

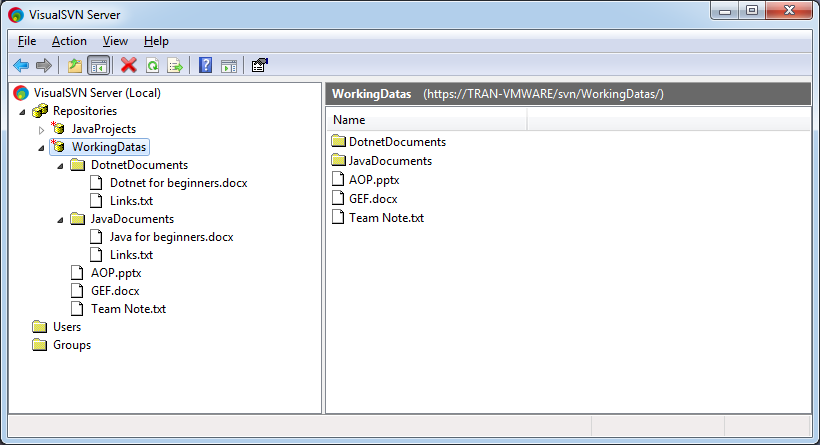
https://TRAN-VMWARE/svn/WorkingDatas/



Dữ liệu đã được **import**(nhập khẩu) vào **SVN Repository**.



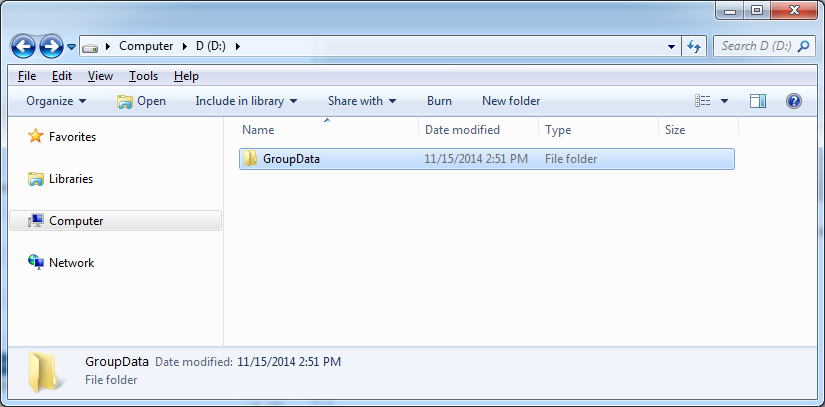
Nhìn trên **Visual SVN**:

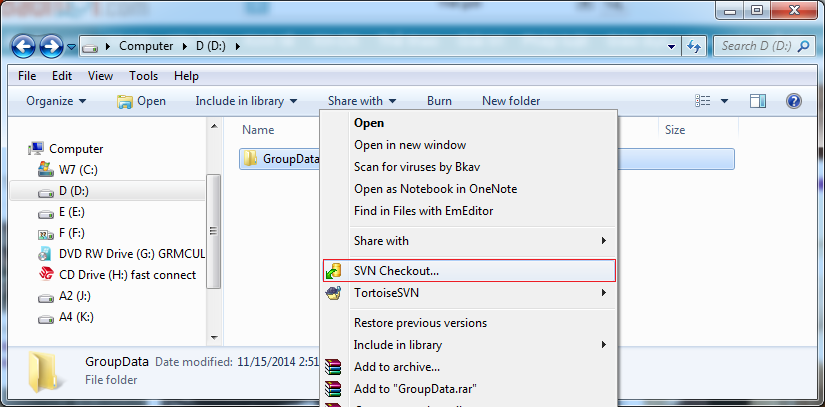


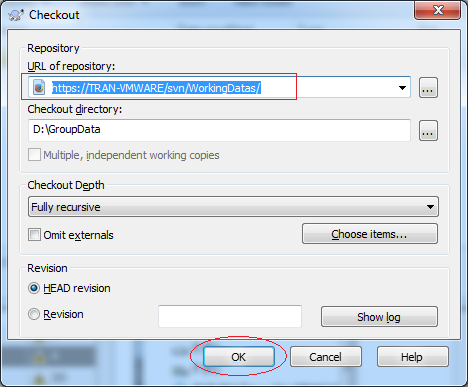
### 5.2. Checkout

Tất cả các thành viên nhóm sẽ **checkout**các dữ liệu từ trên **SVN Repository**về máy tính của mình, bao gồm cả người **import**dữ liệu ban đầu vào **SVN Repository**.

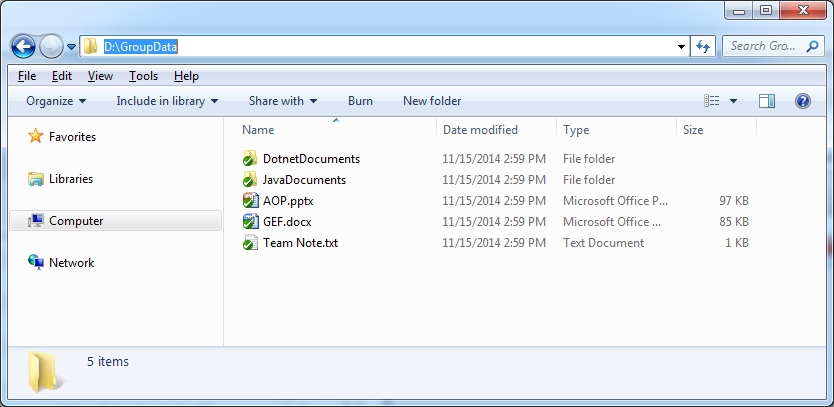
Tại các máy tính của các thành viên, tạo mới một thư mục rỗng nào đó, và **checkout**dữ liệu từ **SVN Repository** về đó. Thư mục này sẽ là thư mục làm việc của mỗi thành viên trên máy tính của mình.







Dữ liệu đã được lấy về.

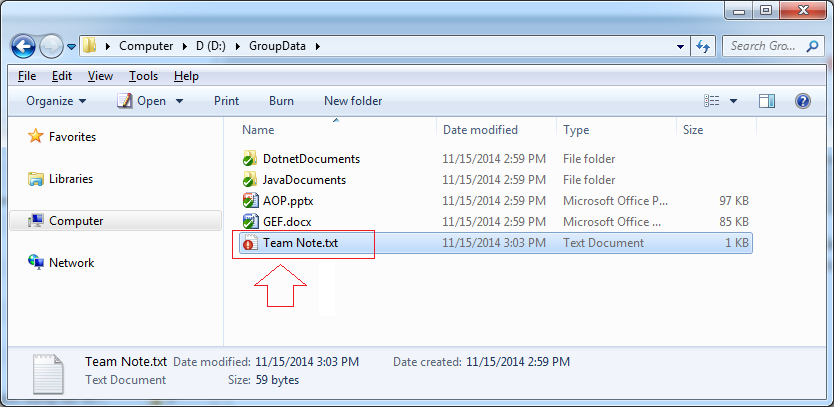


### 5.3. Commit và Update

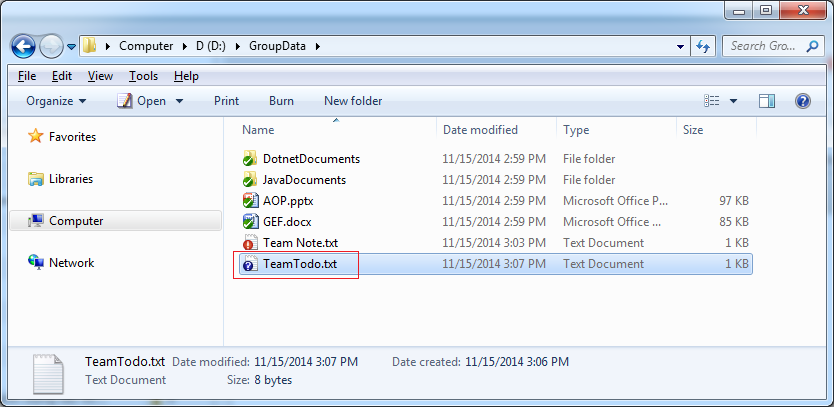
**Checkout**là hành động lấy toàn bộ dữ liệu được chia sẻ trên **Repository**xuống máy địa phương lần đầu tiên. Sau đó dữ liệu được sửa đổi tại địa phương. Việc đưa dữ liệu đã thay đổi lên **Repository**được gọi là **Commit**. Và việc cập nhập các dữ liệu đã thay đổi về là hành động **Update**.

Bây giờ chúng ta sẽ sửa (hoặc thêm) file trên các thư mục rồi **Commit**lên.

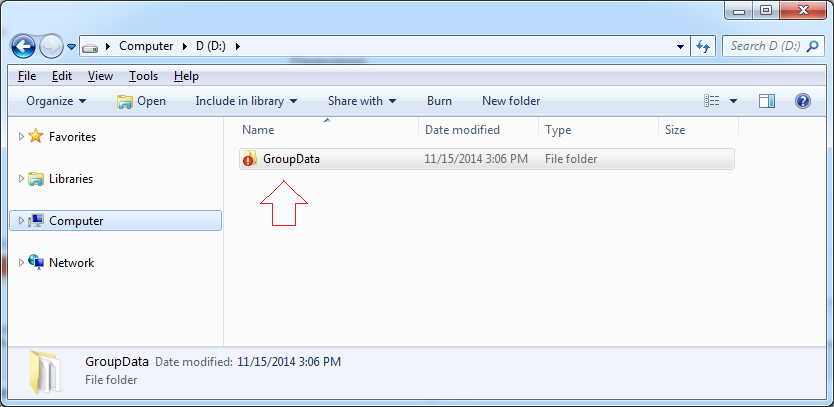
Ví dụ tôi thay đổi file ***TeamNote.txt*** và lưu lại, khi đó biểu tượng **SVN**của file này thay đổi, thông báo rằng file này đã bị sửa đổi.



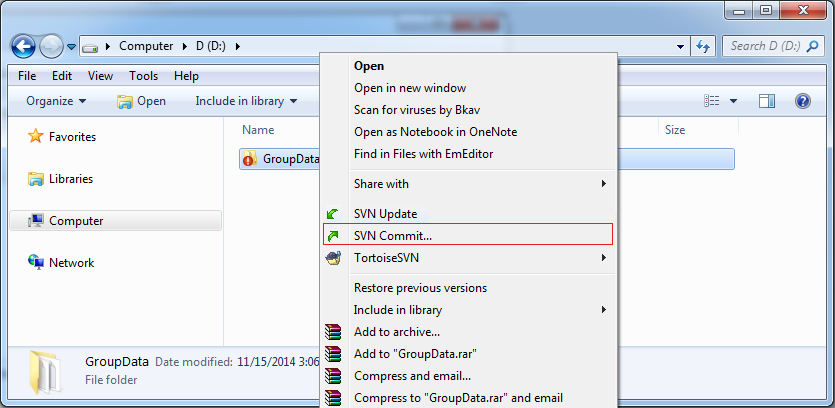
Tiếp theo tôi tạo mới một file ***TeamTodo.txt***. Biểu tượng của file mới được tạo ra có dấu ?, thông báo rằng file này mới, và chưa được quản lý bởi **SVN**.



Thư mục SVN chứa file có dữ liệu bị sửa đổi cũng thay đổi biểu tượng.

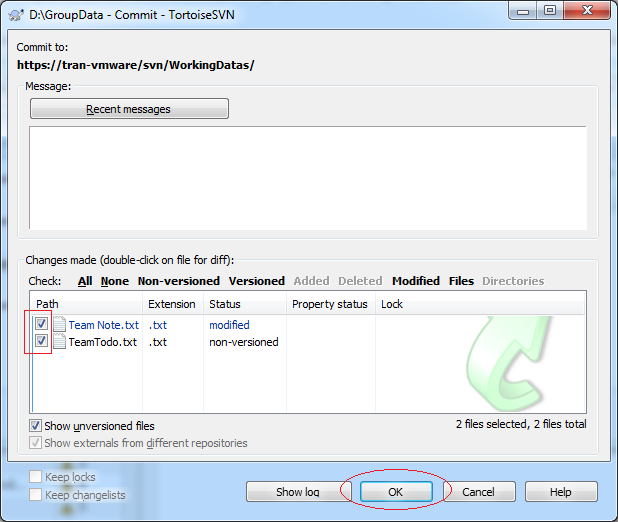


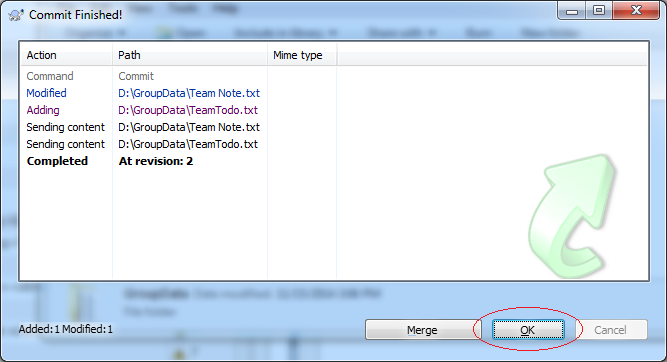
Để **Commit**dữ liệu lên **SVN Repository**, nhấn phải chuột vào file hoặc thư mục cần **Commit**, chọn **SVN Commit...**



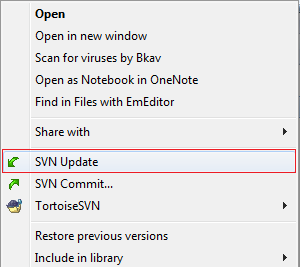
Check chọn các file cần **commit**trên danh sách.

* Mặc định **Tortoise SVN** tự động check vào các file có sửa đổi.





Update: Thành viên khác trong nhóm có thể **Update**(cập nhập) các dữ liệu mới nhất từ **SVN Repository**.



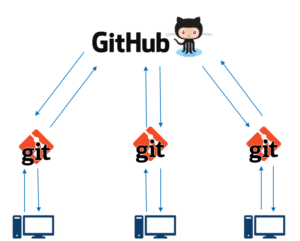
**CHƯƠNG II:GITHUB**

**I. GIỚI THIỆU**

Github là một nền tảng dựa trên web được sử dụng để kiểm soát phiên bản. Git đơn giản hóa quá trình làm việc với người khác và giúp dễ dàng hợp tác trong các dự án.

GitHub là một phần mềm kiểm soát phiên bản "VCS" được sử dụng nhiều. Nó rất hữu ích khi dự án của bạn có nhiều hơn một người cùng làm việc. Ví dụ, một nhóm phát triển phần mềm muốn xây dựng một trang web và mọi người phải cập nhật code của họ đồng thời trong khi làm việc với dự án. Trong trường hợp này, Github giúp họ xây dựng một kho lưu trữ tập trung, nơi mọi người có thể tải lên, chỉnh sửa và quản lý các đoạn code của dự án đó.

GitHub có nhiều lợi thế khác nhau nhưng nhiều người thường nghi ngờ tại sao không sử dụng dropbox hoặc bất kỳ hệ thống dựa trên điện toán đám mây nào? Để trả lời câu hỏi này, bạn hãy hình dung: Nếu có nhiều hơn hai nhà phát triển phần mềm đang làm việc trên cùng một file và họ muốn cập nhật đồng thời. Thật không may, người lưu tệp đầu tiên sẽ được ưu tiên hơn những người còn lại. Trong khi đó ở Github, điều này không xảy ra như vậy. Github có cơ chế ghi lại các thay đổi và phản ánh chúng theo một cách có tổ chức để tránh mọi sự xung đột chồng chéo hoặc trùng lắp giữa bất kỳ tệp nào được tải lên. Do đó, sử dụng kho lưu trữ tập trung GitHub, sẽ tránh được tất cả sự nhầm lẫn và làm việc trên cùng một tập tin hoặc dự án sẽ trở nên rất dễ dàng.



Nếu bạn nhìn vào hình ảnh bên trên này, thì **GitHub** là một kho lưu trữ trung tâm và **Git**là một công cụ cho phép bạn tạo một kho lưu trữ cục bộ tại máy tính của bạn. Bây giờ mọi người thường bị nhầm lẫn giữa git và GitHub nhưng thực tế nó rất khác nhau. Git là một công cụ kiểm soát phiên bản cho phép bạn thực hiện tất cả các loại hoạt động để lấy dữ liệu từ máy chủ trung tâm hoặc đẩy dữ liệu đến nó trong khi GitHub là một nền tảng lưu trữ cốt lõi để giúp lập trình viên cộng tác với nhau kiểm soát các phiên bản mã code. Hay nói cách khác, GitHub là một tập hợp nhiều kho lưu trữ được đặt trên một máy chủ từ xa, nó cho phép bạn lưu trữ kho của mình trên đó.

Dưới đây là những ưu điểm mà GitHub làm cho git trở nên đơn giản hơn:

- GitHub cung cấp cho bạn một giao diện trực quan (nền web) đẹp mắt giúp bạn theo dõi hoặc quản lý các dự án được kiểm soát phiên bản ở local trái ngược so với Git (bạn phải thao tác trên CLI)

- Khi bạn đăng ký GitHub, bạn sẽ được kết nối với các mạng xã hội bao gồm các tổ chức, cá nhân cũng đang làm việc cùng lĩnh vực với bạn giúp bạn có thể trao đổi với họ và theo dõi các thông tin mới nhất về các dự án code khác mà bạn quan tâm. Song song đó, Github còn giúp bạn xây dựng được một hồ sơ cá nhân mạnh mẽ.

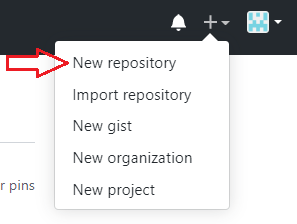
**II. LÀM VIỆC VỚI GITHUB**

**1. Tạo kho lưu trữ GitHub - Tạo Repository trên GitHub**

Một kho lưu trữ "Repository" là một không gian lưu trữ, nơi mà dự án của bạn sẽ được đẩy lên. Nó có thể là nằm ở local như một thư mục trên máy tính của bạn hoặc nó có thể là một không gian lưu trữ trên GitHub hoặc một máy chủ trực tuyến khác. Bạn có thể lưu trữ các tệp mã, tệp văn bản, hình ảnh hoặc bất kỳ loại tệp nào trong kho lưu trữ. Nếu như bạn cần một kho lưu trữ như GitHub để khi bạn thực hiện một số thay đổi và muốn tải chúng lên kho để lưu trữ và chia sẻ trực tuyến. Kho lưu trữ GitHub này hoạt động như kho lưu trữ từ xa của bạn. Các bước để tạo kho lưu trữ GitHub rất đơn giản, bạn có thể làm theo cách sau:

Đầu tiên hãytruy cập liên kết:  https://github.com/ . Điền vào biểu mẫu đăng ký và nhấp vào **Sign up for Github**để tạo cho mình 1 tài khoản Github.  
Đăng nhập vào tài khoản GitHub của bạn.

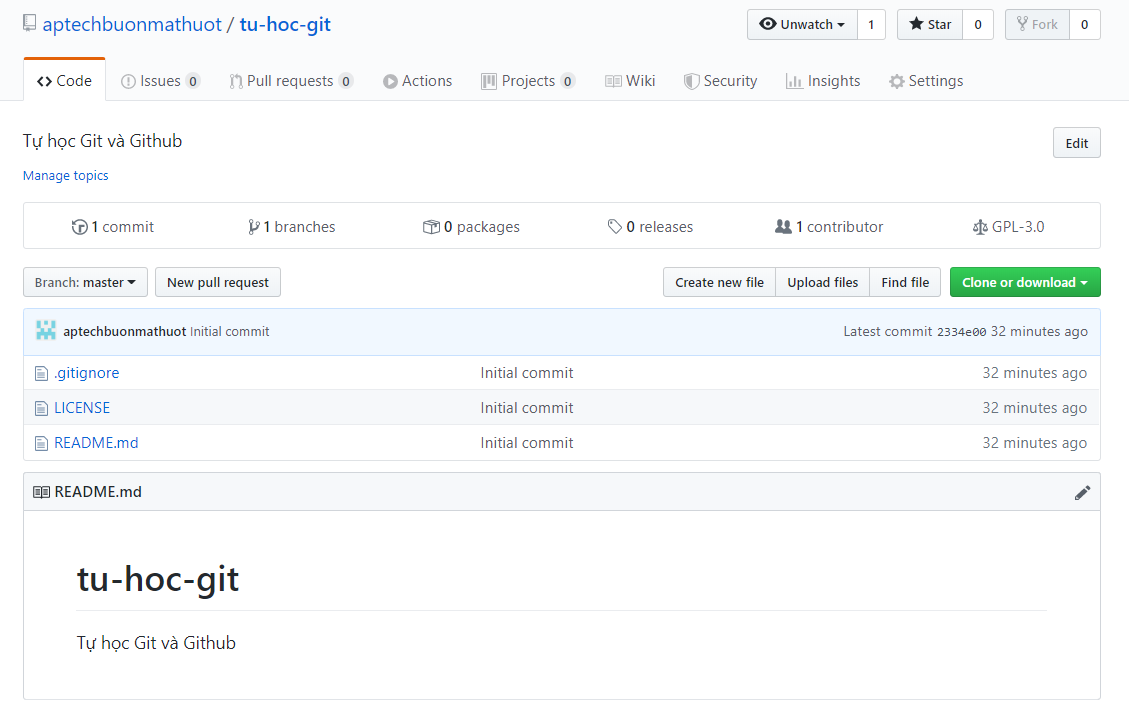
Sử dụng dấu "+" ngoài cùng bên phải để tạo một một Repository



Bây giờ, nếu bạn nhận thấy theo mặc định, kho lưu trữ GitHub là công khai "**Public"**, điều đó có nghĩa là bất kỳ ai cũng có thể xem nội dung của kho lưu trữ này. Còn đối với kho riêng "**Private**", bạn phải chỉ định ai thì họ có thể xem nội dung.

***Lưu ý:*** hãy khởi tạo kho lưu trữ của bạn bằng tệp**README.md** nếu như đây là kho lưu trữ hoàn toàn mới. Tệp này thường được sử dụng để lưu các mô tả về kho của bạn, các thay đổi của kho để giúp người xem có thể nắm được một cách khái quát và nhanh nhất về kho mà bạn chia sẻ. Khi bạn đánh dấu vào tùy chọn **Initialize this repository with a README**thì file README sẽ là tệp đầu tiên bên trong kho lưu trữ của bạn.

Sau khi kho lưu trữ của chúng ta đã được tạo thành công. Nó sẽ trông giống như dưới đây:



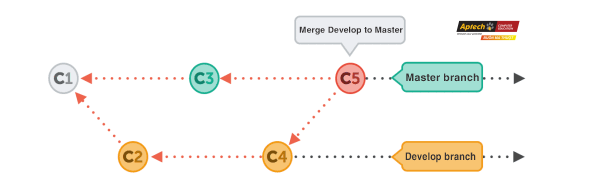
Như vậy, tôi đã chuẩn bị sẵn sàng cho mình một kho lưu trữ từ xa trên Github, bây giờ thì tôi có thể thực hiện các commit, pull, push và thực hiện tất cả các hoạt động khác với kho này. Bây giờ hãy tiếp tục tìm hiểu phân nhánh trong GitHub.

**2. Tạo chi nhánh "Branch" trên Github và quản lý các thay đổi trong chi nhánh**

**a) Chi nhánh "Branch" trong  Github là gì?**

Branch trong **git và github** dịch ra là chi nhánh, nó sẽ giúp bạn làm việc trên các phiên bản khác nhau của kho lưu trữ cùng một lúc. Giả sử bạn muốn thêm một tính năng mới (đang trong giai đoạn phát triển) và bạn lo ngại tính năng mới này chưa được phát triển ổn định, nếu mang nó vào phiên bản đã hoàn thiện chạy ổn định rồi thì nó có thể gây ra lỗi bug không đáng có cho dự án của mình. Các **Branches**của git lúc này giúp làm hạn chế các lỗi đó.

Chi nhánh cho phép bạn di chuyển qua lại giữa các trạng thái / phiên bản "states/versions" khác nhau của dự án. Để giải quyết vấn đề nêu trên, bạn có thể tạo một nhánh mới và kiểm tra tính năng mới trên đó mà không ảnh hưởng đến nhánh chính. Khi bạn đã thực hiện xong, bạn có thể hợp nhất "**merge**" các thay đổi từ nhánh mới sang nhánh chính. Ở đây, nhánh chính là nhánh **master**, có trong kho lưu trữ của bạn theo mặc định. Tham khảo hình ảnh dưới đây để hiểu rõ hơn:

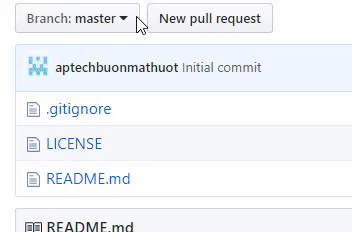


Như mô tả trong hình trên, có một nhánh master và một nhánh mới develop để thử nghiệm. Trong nhánh develop này, hai bộ thay đổi C2,C4 được thực hiện và sau khi hoàn thành, nó được hợp nhất trở lại nhánh chính master tại C5. Đây là cách phân nhánh hoạt động!

Để tạo một nhánh trong GitHub, hãy làm theo các bước dưới đây:

- Nhấp vào danh sách thả xuống "Branch: master"

- Ngay khi bạn nhấp vào chi nhánh, bạn có thể tìm thấy một chi nhánh hiện có hoặc bạn có thể tạo một chi nhánh mới. Trong trường hợp của tôi, tôi đang tạo một chi nhánh mới với tên là develop. Tham khảo ảnh chụp màn hình dưới đây để hiểu rõ hơn.

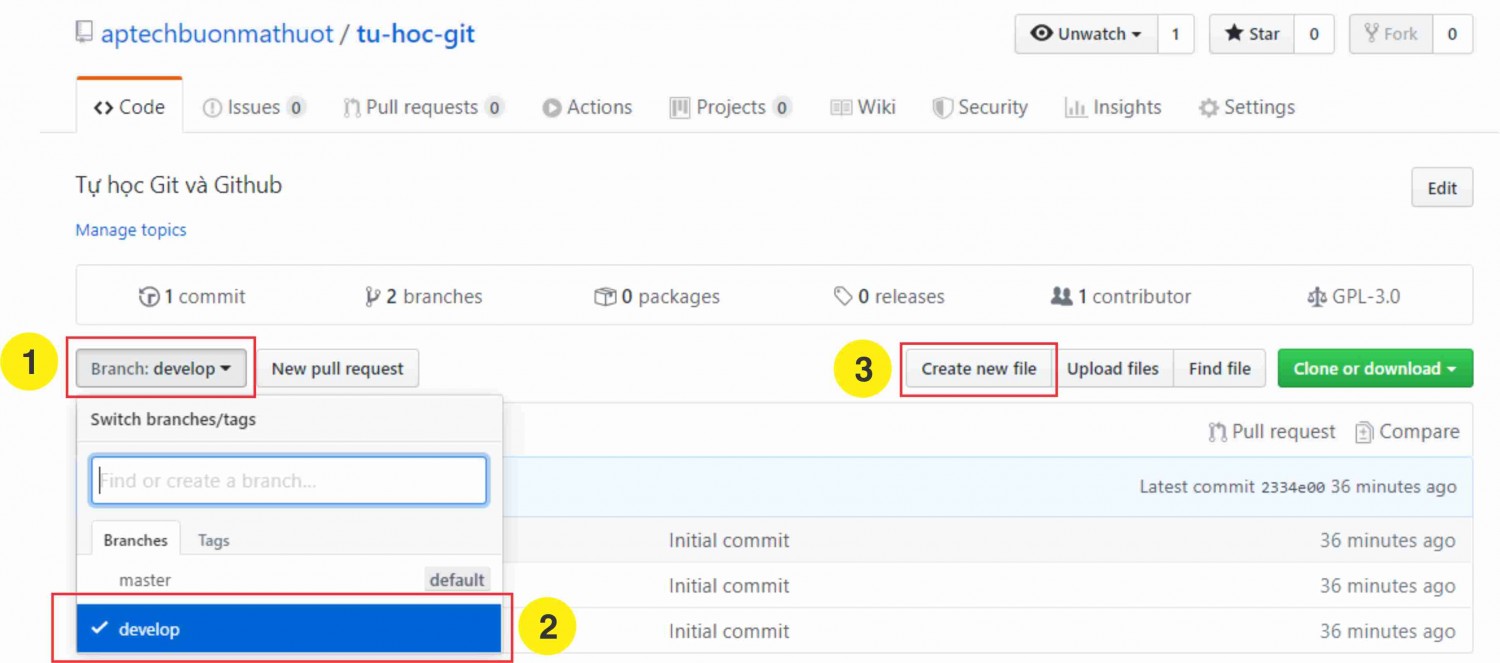


Khi bạn đã tạo một nhánh mới, bây giờ bạn có hai nhánh trong kho lưu trữ của mình tức là master (nhánh chính) và develop. Chi nhánh develop mới chỉ là bản sao của chi nhánh master, tại thời điểm bạn vừa tạo mới nếu bạn kiểm tra trên develop bạn sẽ thấy tất cả các file, commit của master. Bây giờ hãy tiếp tục thực hiện một số thay đổi trong nhánh develop mới xem thế nào nhé!

**b) Quản lý các thay đổi trong chi nhánh "Branches"**

**Lệnh commit:**Thao tác này giúp bạn lưu các thông tin về sự thay đổi của tệp chỉ khi bạn thực hiện các thao tác như thêm, xóa, sửa tập tin. Khi bạn **commit**một thay đổi của tập tin, bạn phải luôn cung cấp một thông điệp "**mesage**"  nó sẽ giúp ghi nhớ những thay đổi do bạn thực hiện. Đại loại như: "*Hôm nay, ngày 21/11/2019 lúc 11h30'20s tôi có thêm một dòng trong file index.html*" ^^ mặc dù thông báo này không bắt buộc nhưng nó luôn được khuyến nghị để nó có thể phân biệt các phiên bản khác nhau (mỗi thay đổi trên file sẽ tạo ra một phiên bản mới của file đó được hiểu như ảnh chụp snapshot) hoặc phân biệt các commit bạn đã thực hiện trong kho lưu trữ của mình. Những commit này duy trì lịch sử thay đổi, từ đó giúp những người cộng tác khác hiểu rõ hơn về tệp mà họ cùng đang thao tác trên đó. Bây giờ hãy thực hiện commit đầu tiên của chúng ta, làm theo các bước dưới đây:

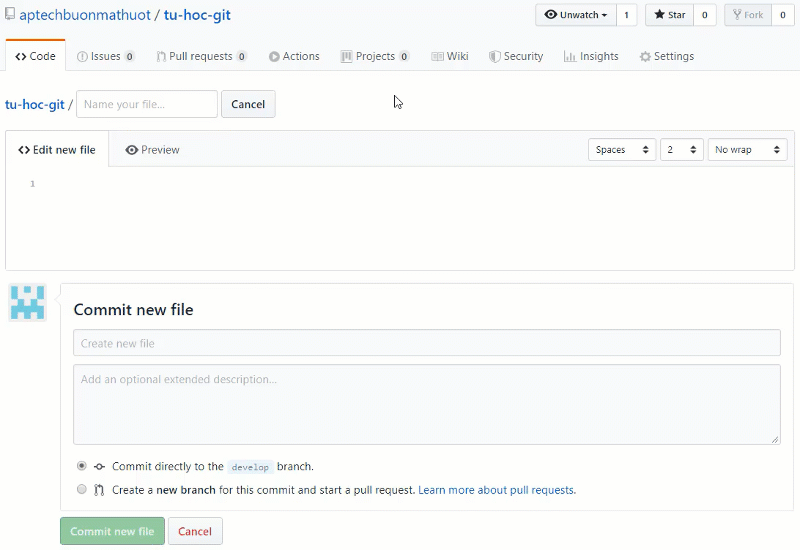
- Nhấp vào branch develop vừa tạo chọn **Create New file**



- Sau khi tạo mới một tập tin, ở đây tôi đặt tên cho tập tin là hello.txt kèm theo nội dung nhập ở phần **<>Edit new file**

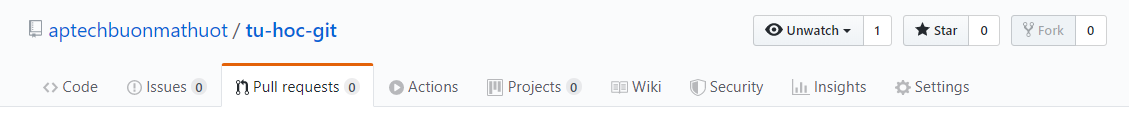
- Viết một commit để ghi lại những thay đổi của bạn với tiêu đề và phần mô tả nội dung commit

- Nhấp vào **Commit new file** để tạo mới 1 commit kèm theo .



**Lệnh pull:**Lệnh kéo "pull" là lệnh quan trọng nhất trong GitHub. Nó cho biết những thay đổi được thực hiện trong tệp và yêu cầu những người đóng góp khác xem nó và ra yêu cầu hợp nhất nó với nhánh chính master (**Pull request**) . Khi commit được thực hiện, bất kỳ ai cũng có thể pull tệp và có thể bắt đầu một cuộc thảo luận về nó. Khi tất cả đã xong, bạn có thể hợp nhất bằng cách merge vào master. Lệnh Pull so sánh các thay đổi được thực hiện trong tệp và nếu có bất kỳ xung đột nào, bạn có thể giải quyết thủ công. Bây giờ chúng ta hãy xem các bước khác nhau liên quan để tạo một Pull Request trong GitHub.

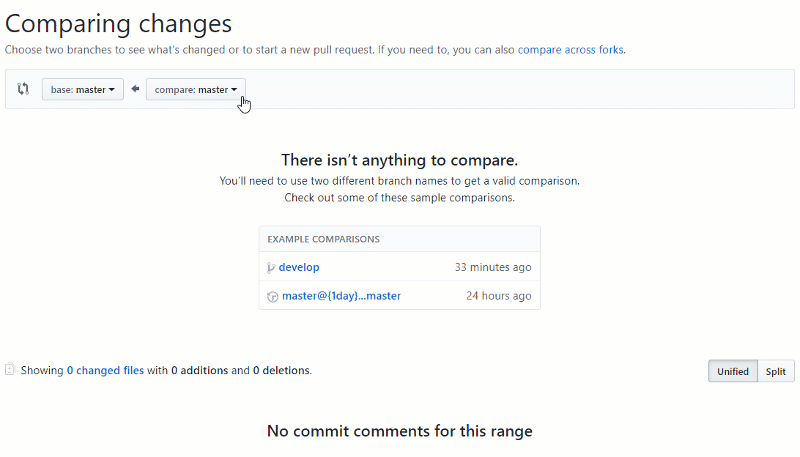
- Nhấp vào tab 'Pull Request'.



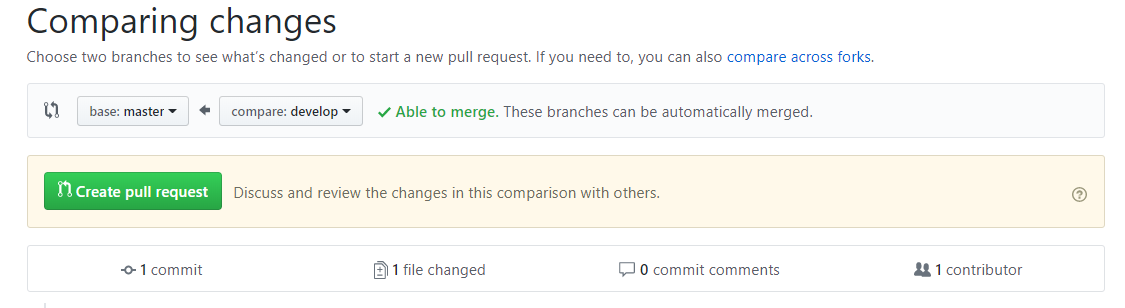
- Nhấp vào 'New pull request'.

Tab Pull request 2- Hướng dẫn sử dụng Github - Aptech Buôn Ma Thuột

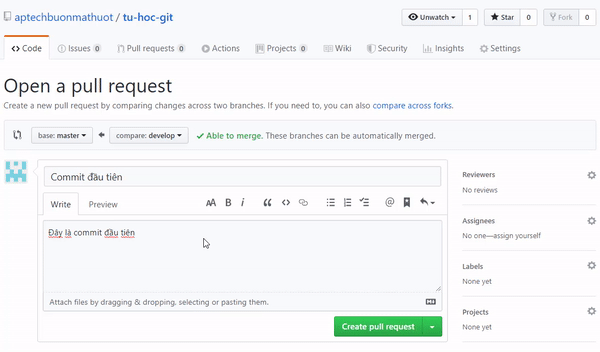
- Khi bạn nhấp vào yêu cầu kéo, chọn Branch và nhấp vào 'develop ' để xem các thay đổi giữa hai tệp có trong kho lưu trữ của chúng ta



- Nhấp vào **Create Pull request** sau khi đã kiểm tra và so sánh các thay đổi



Nhập bất kỳ tiêu đề, mô tả cho các thay đổi của bạn và nhấp vào **Create pull request**

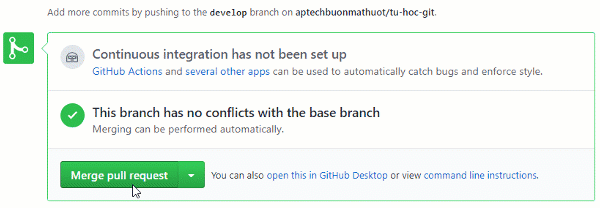


Sau khi tạo thành công Pull request, chúng ta hãy tiếp tục hợp nhất yêu cầu này.

**Lệnh Merge:**Đây làlệnh cuối cùng để hợp nhất các thay đổi trên nhánh Develop vào nhánh chính Master sau khi một Pull request đã được khởi tạo và các cộng tác viên đã xem xét, chấp thuận nội dung. Các bước sau đây để hợp nhất 2 nhánh này.

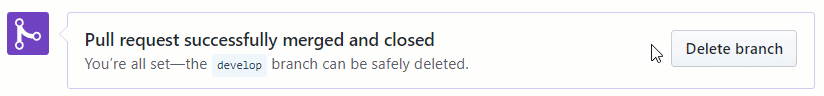
- Nhấp vào **Merge pull request** để hợp nhất các thay đổi vào nhánh chính.

- Nhấp vào Confirm merge để xác nhận hợp nhất, nếu hợp nhất thành công sẽ có thông báo.



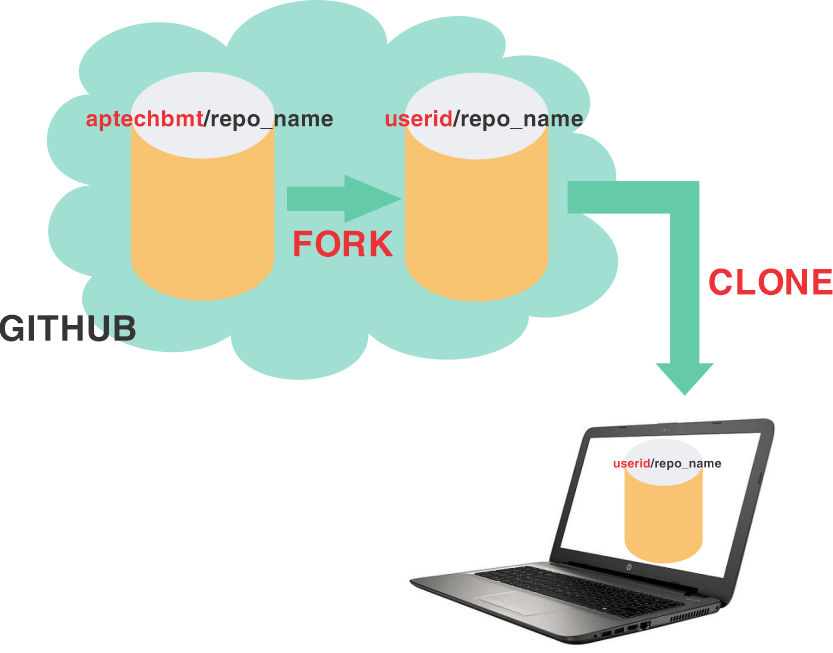
**Lưu ý**: Hợp nhất chỉ diễn ra khi không xảy ra xung đột conflict. Github sẽ kiểm tra conflict, nếu xảy ra xung đột bạn phải giải quyết vấn đề này trước khi lệnh merge được thực hiện.

Sau khi merge thành công vào nhánh master, bạn có thể xóa nhánh develop bằng cách nhấn vào **Delete branch**

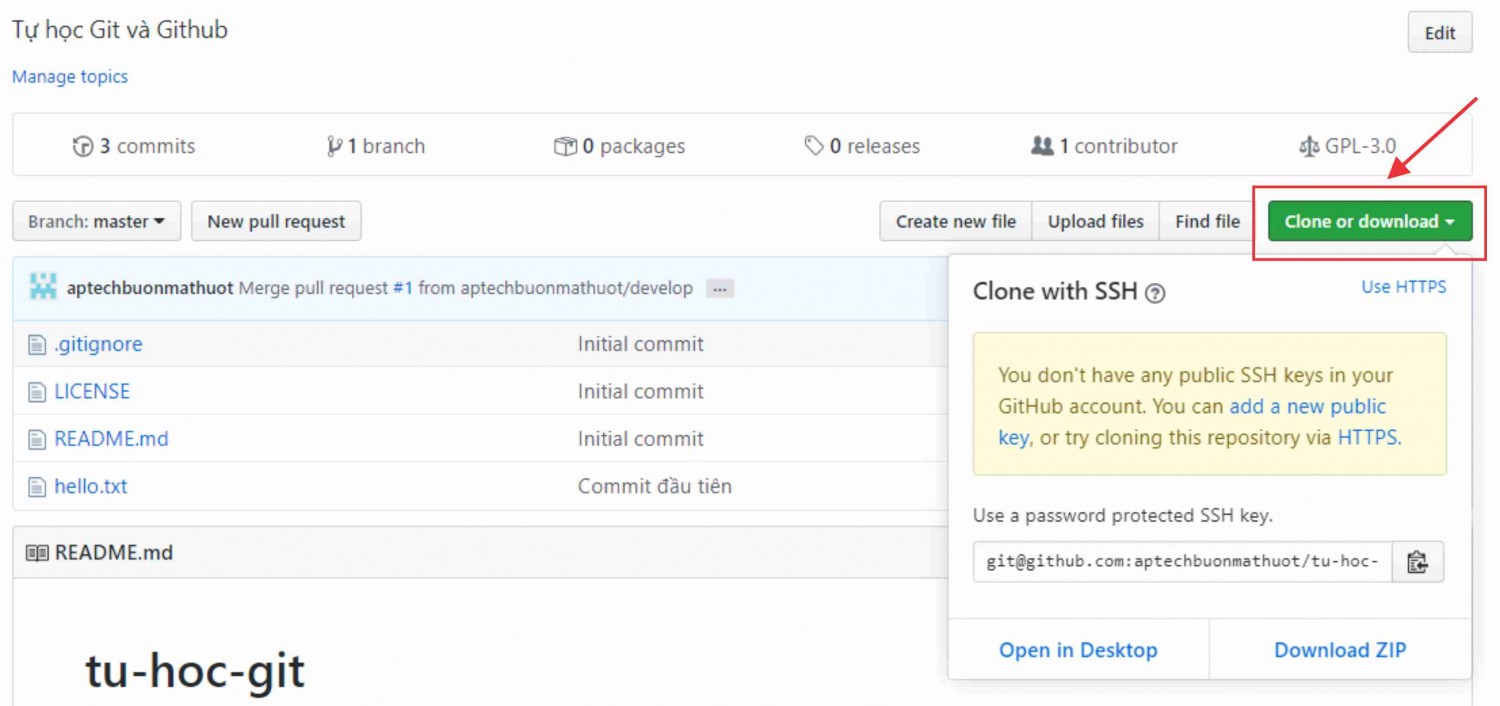


**3. Nhân bản Clone và Fork trong Github**

Trước khi đi vào từng phần, bạn hãy nhìn qua sơ đồ này để hình dung được cách thức hoạt động của Clone và Fork trên Github. Bạn có thể hiểu là Fork thao tác sao chép các kho lưu trữ "repository" từ tài khoản này đến tài khoản khác trên Github, còn Clone thì giúp chúng ta tải xuống kho lưu trữ về máy tính ở local.



Nhân bản **Clone:**Trước khi tôi thực sự nói về việc nhân bản một kho lưu trữ GitHub, thì việc đầu tiên bạn cần phải hiểu lý do tại sao chúng ta cần sao chép một kho lưu trữ? Câu trả lời rất đơn giản! Đó là để tái sử dụng code, giả sử bạn muốn sử dụng một số mã code có trong kho lưu trữ công cộng (tài khoản Github khác), bạn có thể sao chép trực tiếp nội dung bằng cách nhân bản hoặc tải xuống xuống máy tính của mình.



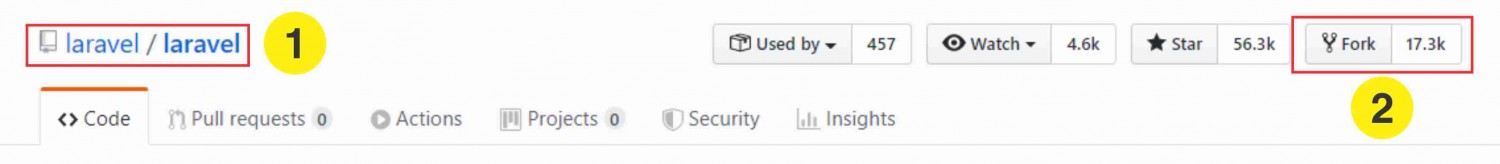
Rẽ nhánh kho lưu trữ với **Fork:**Đầu tiên, chúng ta hãy nói về lý do tại sao chúng ta cần rẽ nhánh. Giả sử, bạn cần một số mã có trong kho lưu trữ công khai, để đẩy nó vào kho lưu trữ và tài khoản GitHub của bạn. Muốn làm điều này chúng ta cần rẽ nhánh "Fork" một kho lưu trữ sang tài khoản của mình đúng không nào?

Nhưng trước khi chúng ta bắt đầu với forking, có một số điểm quan trọng mà bạn nên luôn luôn ghi nhớ.

- Những thay đổi được thực hiện đối với kho lưu trữ ban đầu sẽ được phản ánh trở lại kho lưu trữ rẽ nhánh.

- Nếu bạn thực hiện thay đổi trong kho lưu trữ rẽ nhánh, nó sẽ không được phản ánh đến kho lưu trữ ban đầu cho đến khi và trừ khi bạn thực hiện yêu cầu kéo.

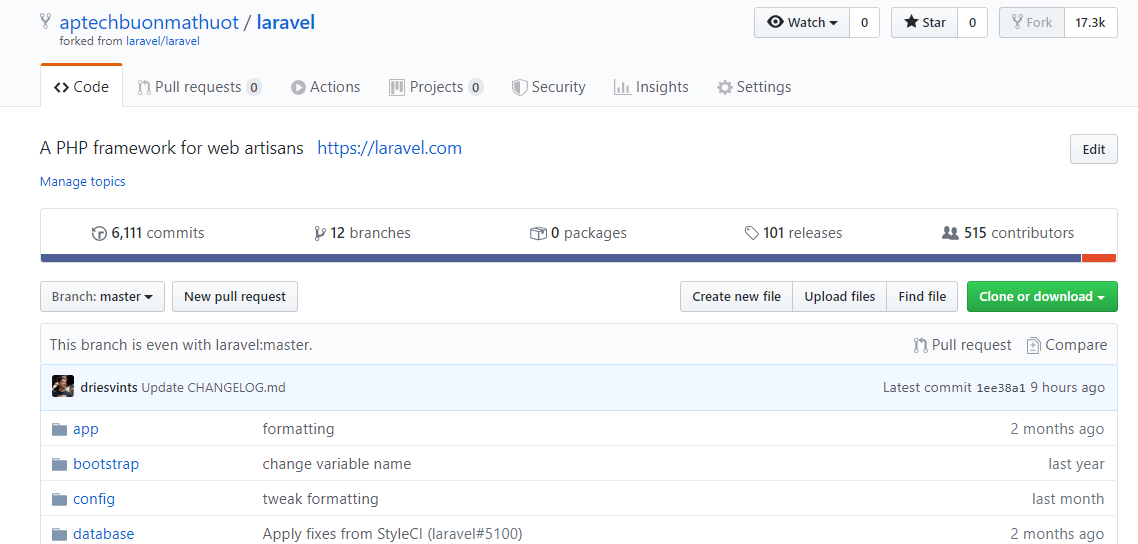
Bây giờ hãy xem làm thế nào bạn có thể **Fork**một kho lưu trữ. Đối với điều đó, hãy làm theo các bước dưới đây:



- Khám phá và tìm kiếm các kho lưu trữ công cộng mà bạn cần Fork. Ví dụ: Tôi sẽ sử dụng kho lưu trữ mã nguồn Laravel để làm.

- Ngay khi bạn nhấp vào trên Fork, bạn sẽ mất một chút thời gian để phân nhánh kho lưu trữ. Sau khi hoàn thành, bạn sẽ nhận thấy tên kho lưu trữ nằm trong tài khoản của bạn.

- Kết quả sau khi Fork kho lưu trữ của Laravel về tài khoản Github của tôi. Như bạn thấy đường dẫn đã thay đổi từ *laravel/laravel* thành *aptechbuonmathuot/laravel.*



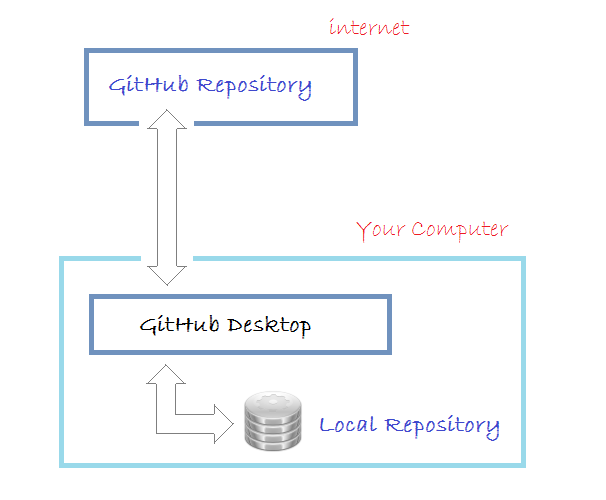
**III. LÀM VIỆC VỚI GITHUB DESKTOP**

## 1. Nguyên tắc hoạt động

Để làm việc với **GitHub**bạn cần:

1. Đăng ký một tài khoản **GitHub**và tạo một **Repository** (**GitHub Repository**).
2. Cài đặt **GitHub Desktop**, một công cụ trực quan quản lý **Local Repository** (Kho chứa dữ liệu địa phương).
3. Cấu hình để có thể đồng bộ hóa dữ liệu bằng **GitHub Desktop** lên **Repository server**.

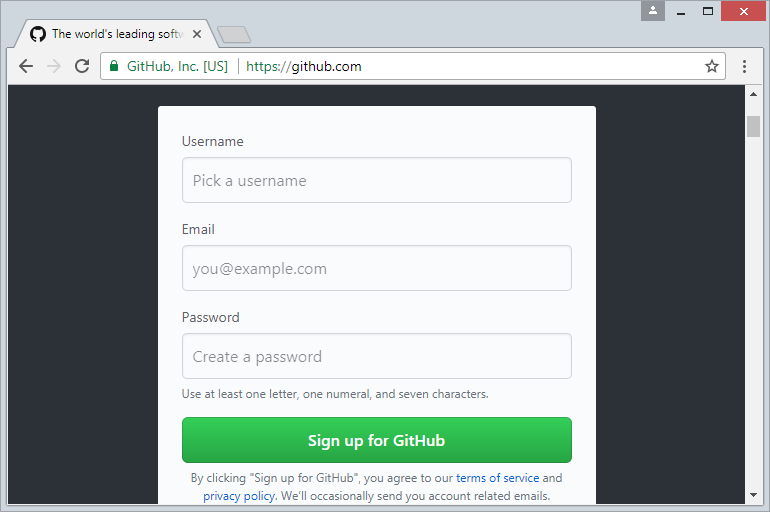
Hãy xem hình minh họa dưới đây:



## 2. Đăng ký tài khoản GitHub

Trước hết bạn cần phải đăng ký miễn phí một tài khoản **GitHub**. Bạn có thể vào trang chủ của **GitHub**tại:

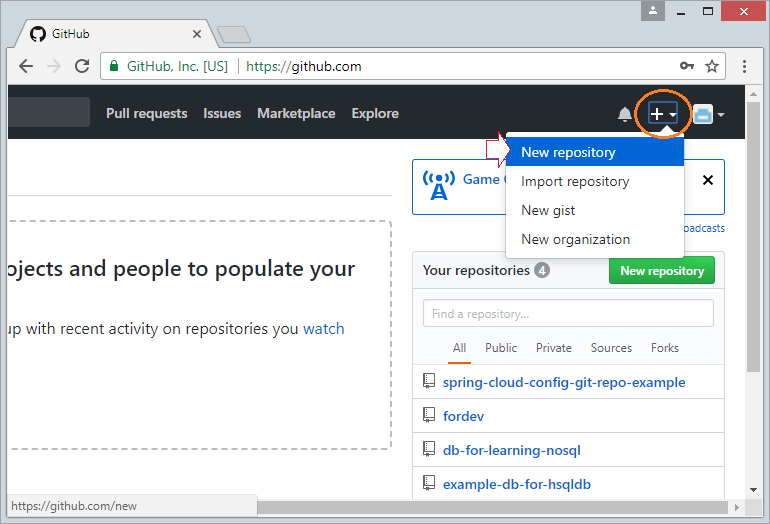
* [https://github.com](https://github.com/)

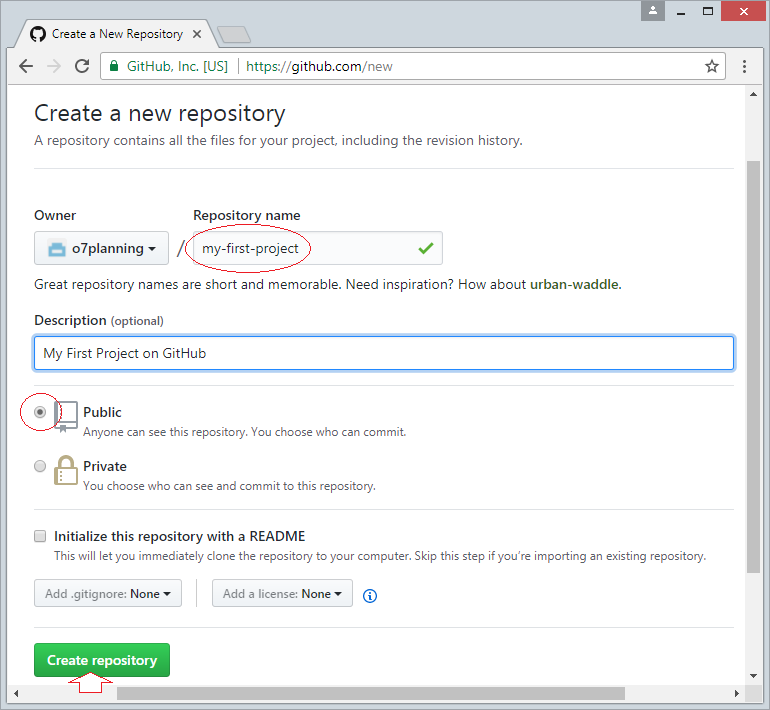


Việc đăng ký một tài khoản là đơn giản, bạn chỉ cần nhập **username/password** và địa chỉ **email**. Sau khi đăng ký xong bạn cần vào Email kích hoạt tài khoản.

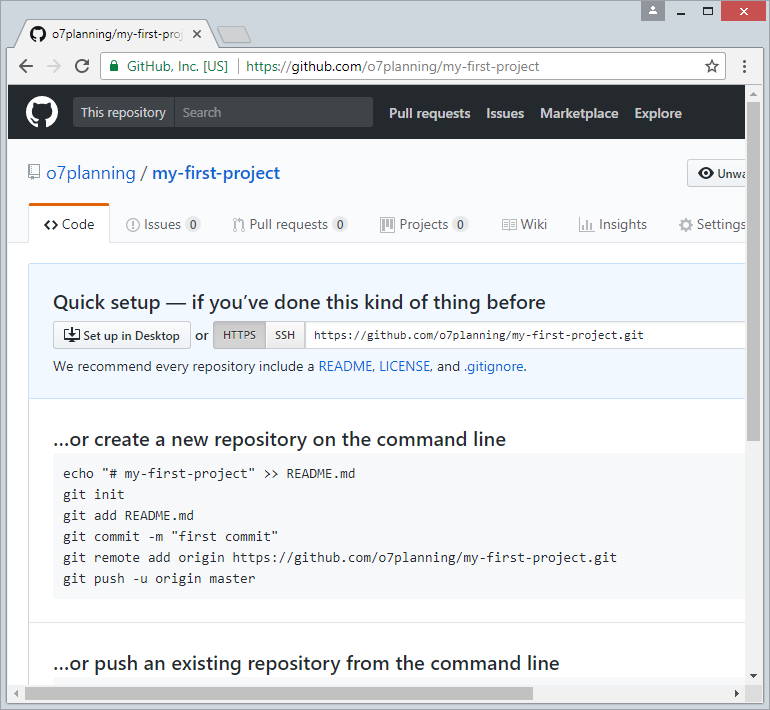
## 3. Tạo GitHub Repository

Sau khi đăng ký xong tài khoản **GitHub**, và đăng nhập vào. Bạn có thể tạo một **GitHub Repository**.



****

**Repository**đã được tạo ra.

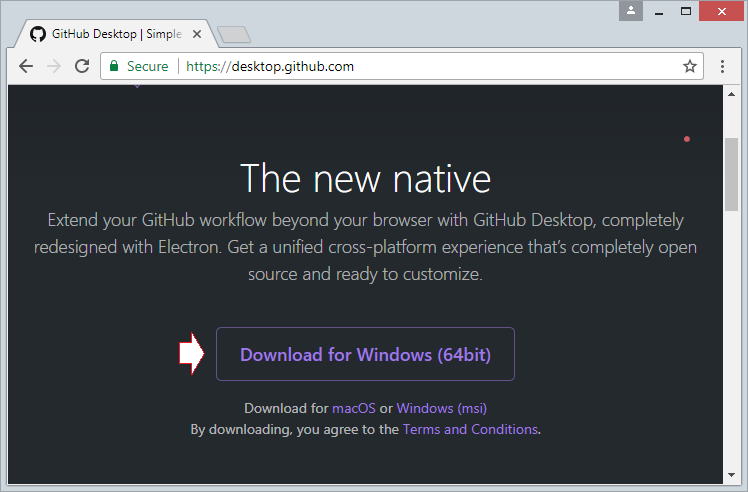


## 4. Download & cài đặt GitHub Desktop

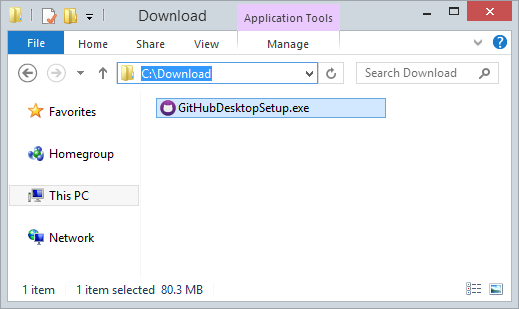
**GitHub Desktop** về bản chất là một công cụ trực quan cho phép bạn quản lý **Local Repository** (Kho chứa địa phương) trên máy tính của bạn.

Để download **GitHub Desktop** bạn vào địa chỉ:

* <https://desktop.github.com/>

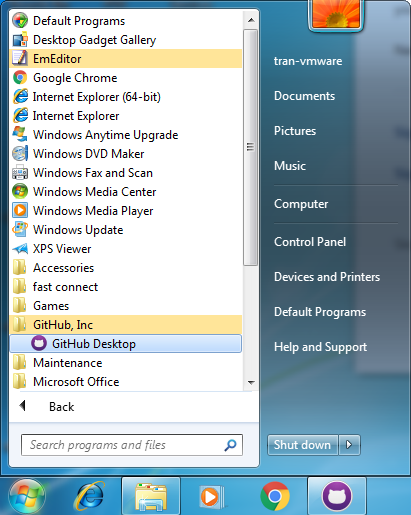


Kết quả download:



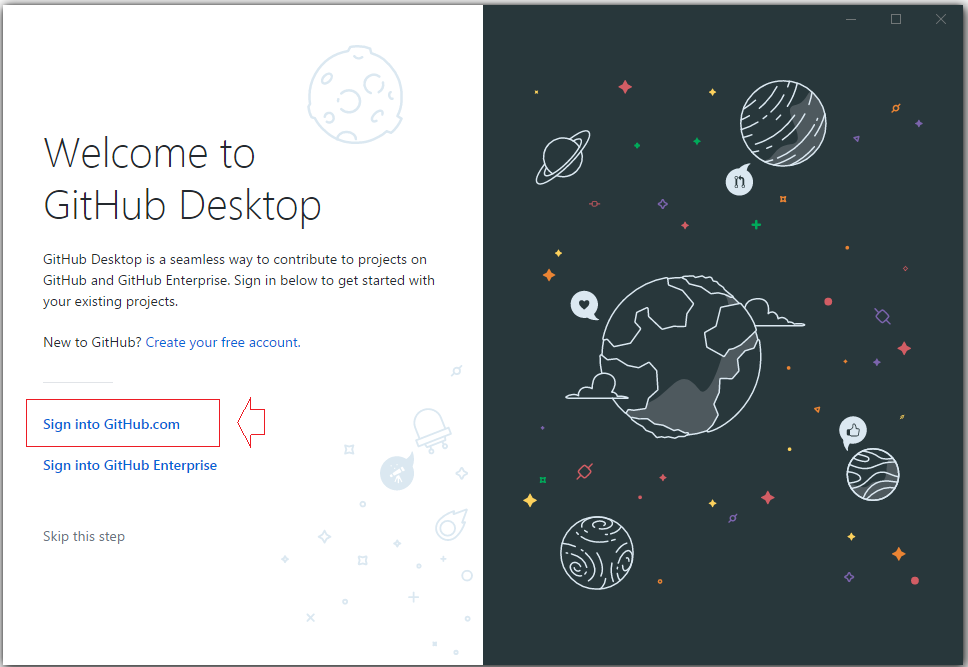
Sau khi download xong, bạn cần cài đặt **GitHub Desktop** vào máy tính:

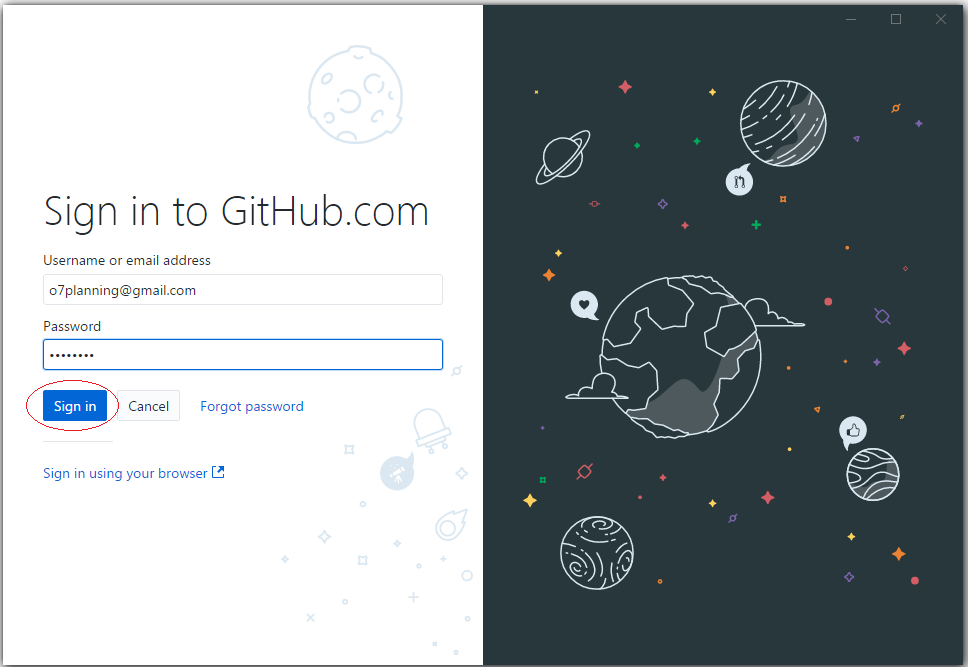
**GitHub Desktop**đã được cài đặt thành công.

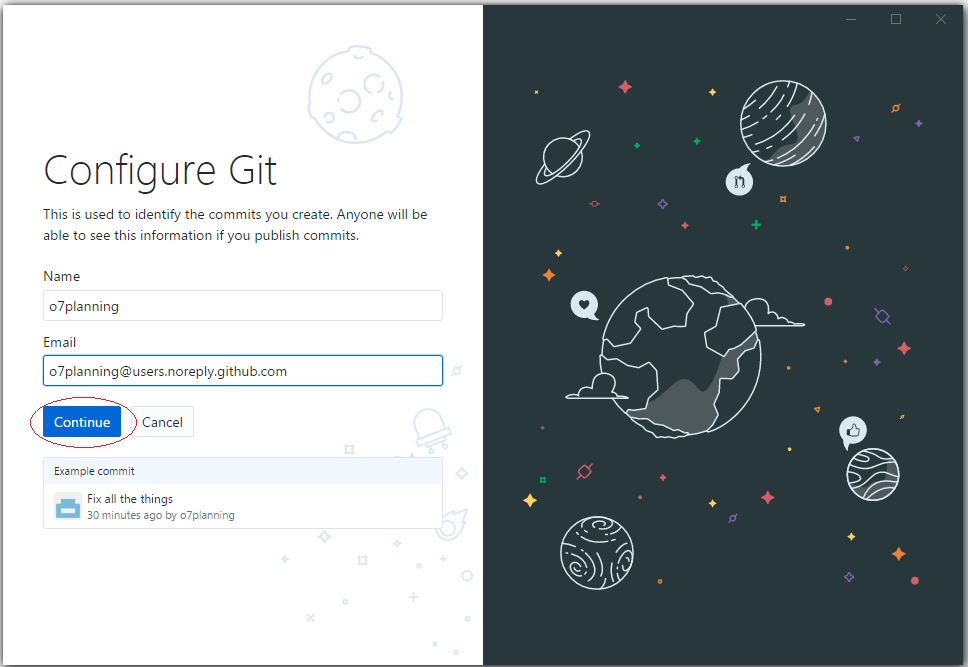


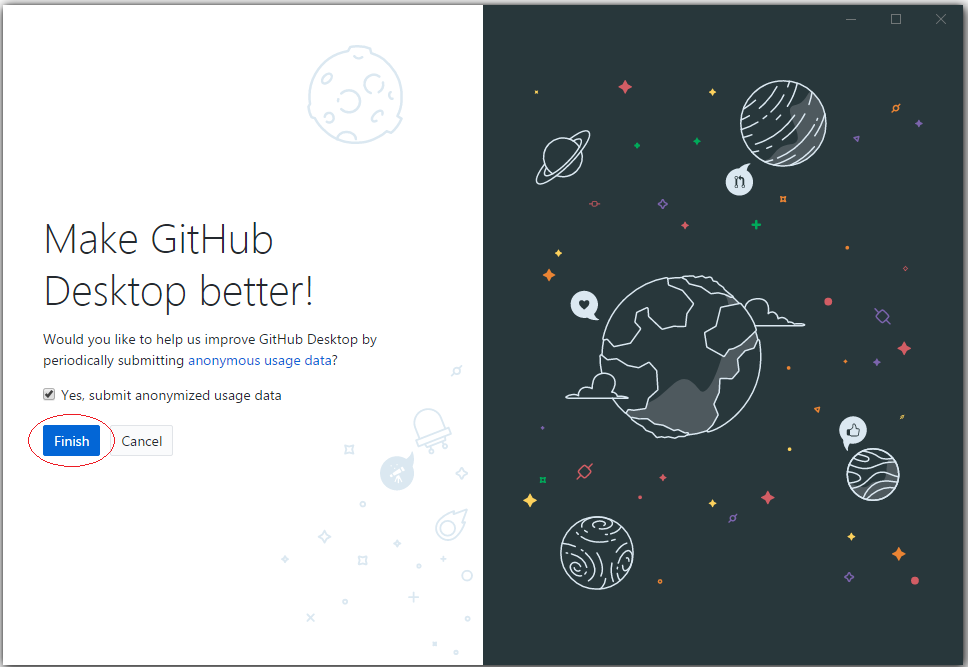
## 5. Chạy GitHub Desktop

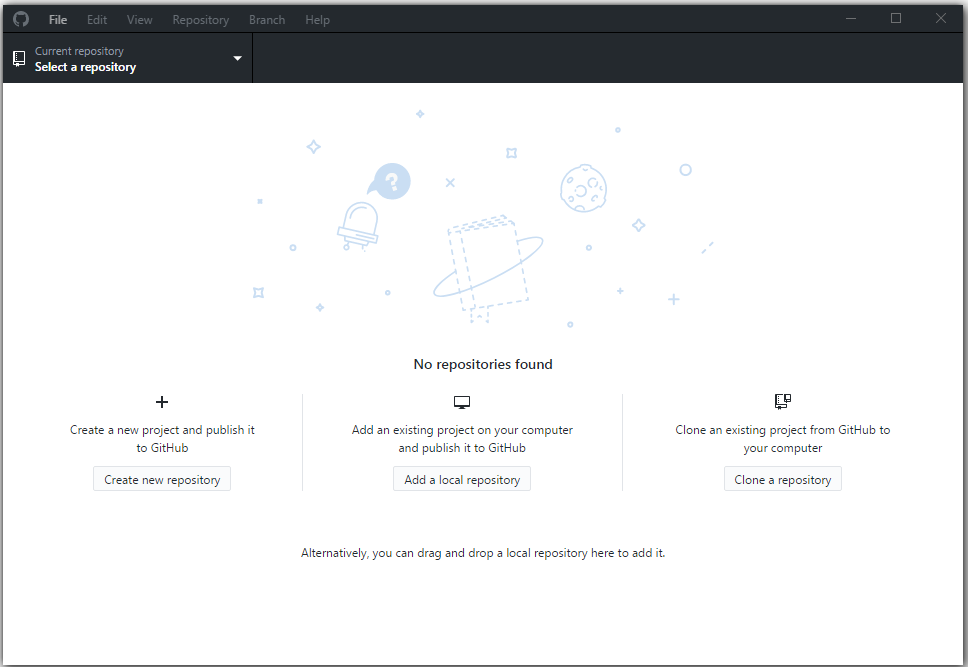
Đăng nhập trên **GitHub Desktop** để kết nối vào tài khoản **GitHub**của bạn.











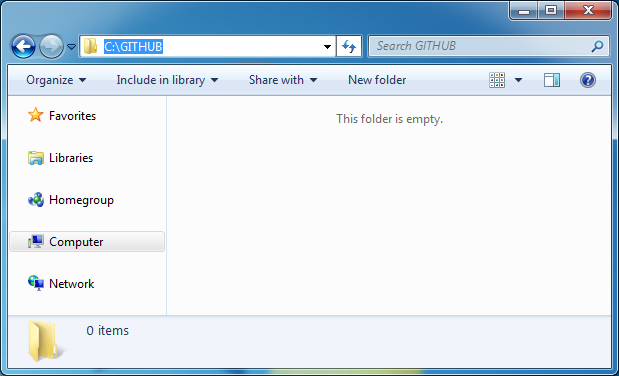
Cho tới lúc này trên máy tính của bạn chưa có một **Local Repository** nào.

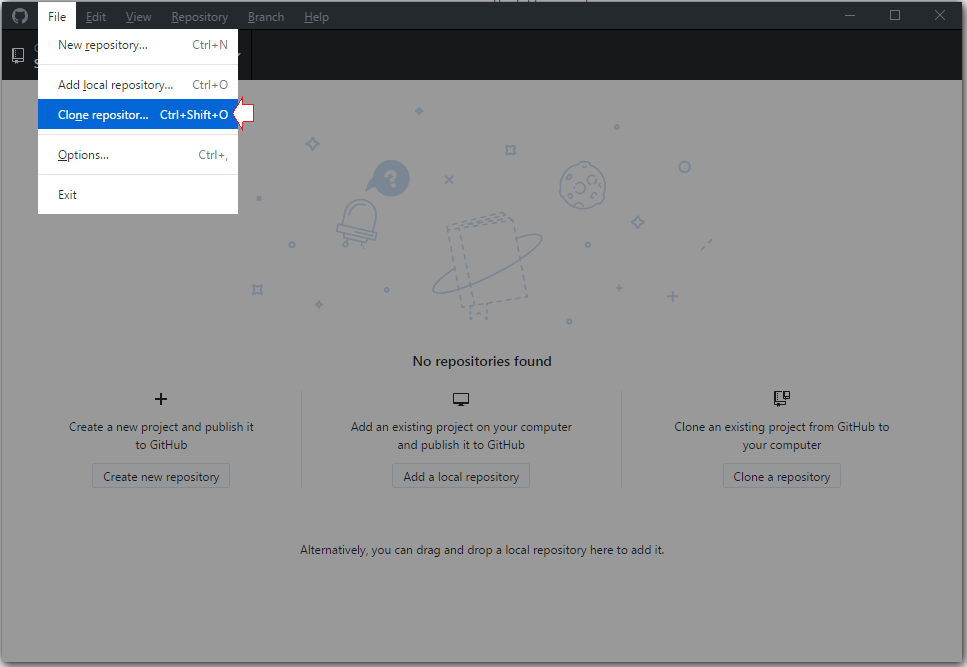
## 6. Kết nối GitHub và GitHub Desktop

Trước hết cần chọn một thư mục rỗng để làm vị trí chứa dữ liệu địa phương.

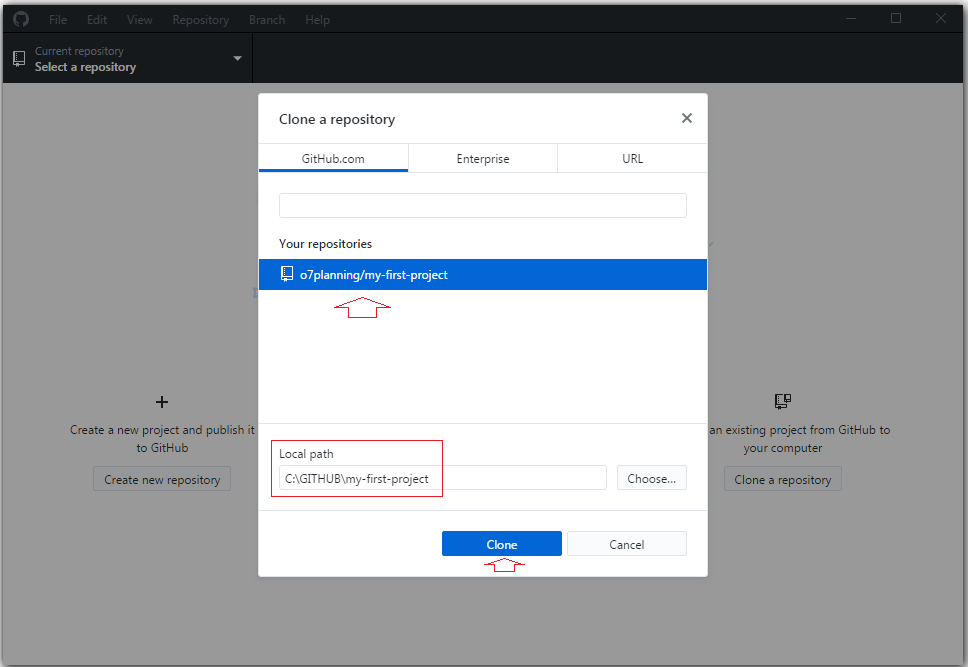
Chẳng hạn:

* **C:/GITHUB**

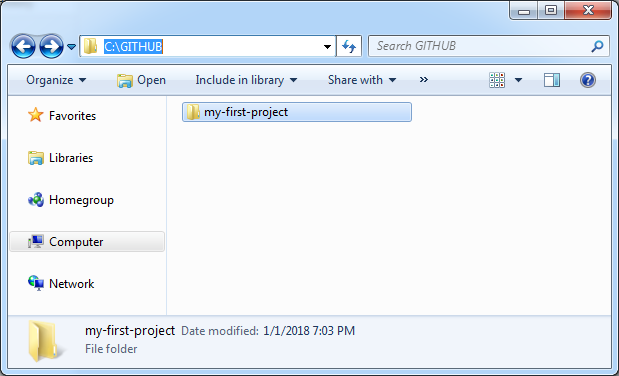


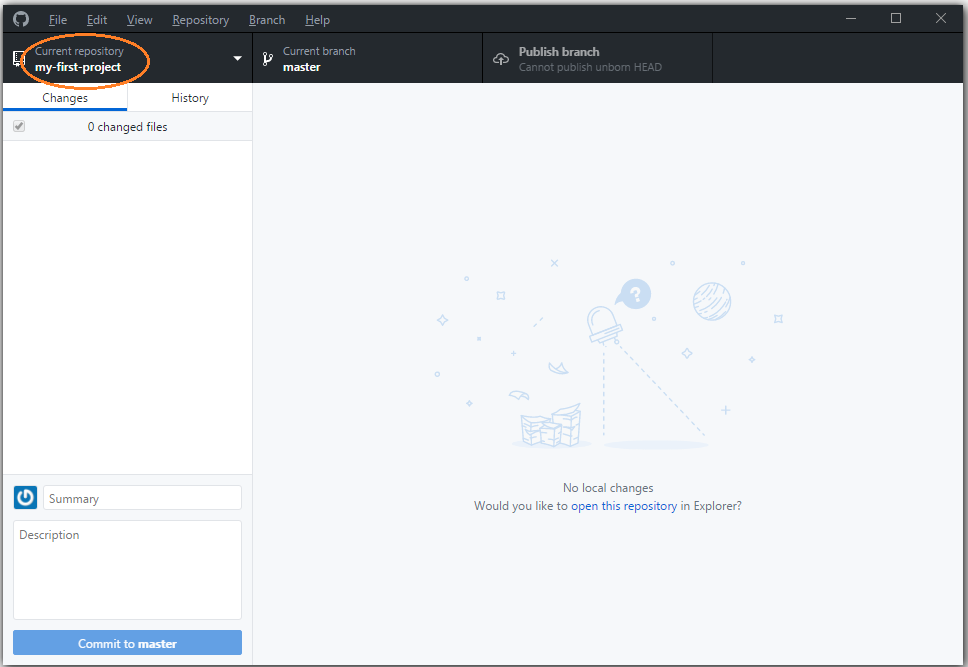


Trên **GitHub Desktop**, lựa chọn một **Repository**bạn đã tạo trên **GitHub** để **clone**(Tạo bản sao chép) thành một bản ở máy tính địa phương của bạn.

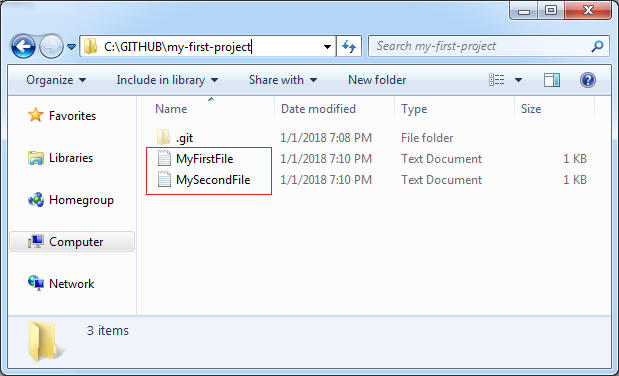


Lúc này trên **GitHub Desktop** bạn sẽ thấy một **Local Repository** đã được tạo ra.

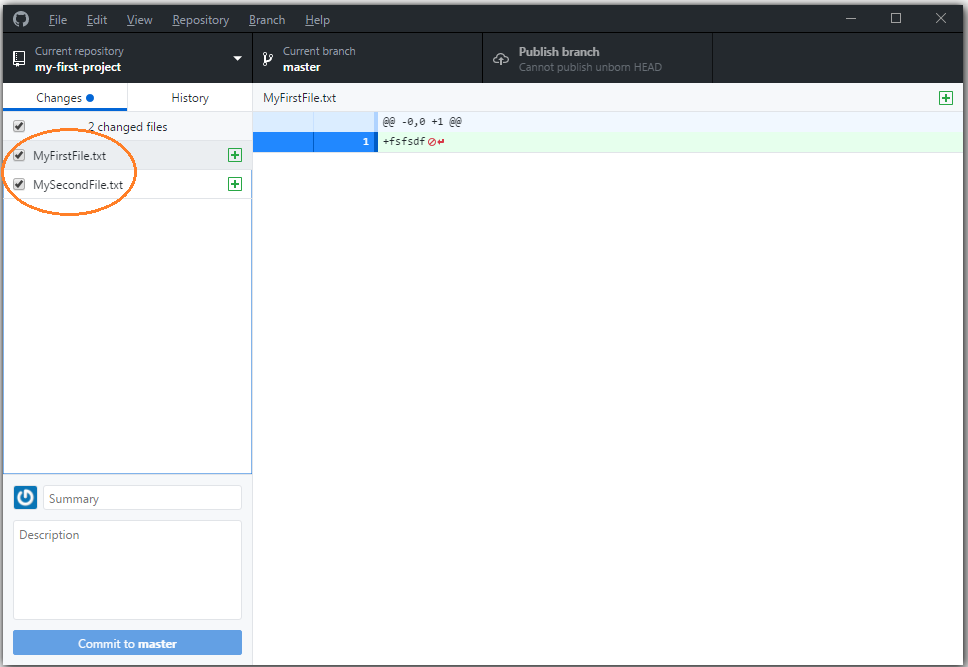




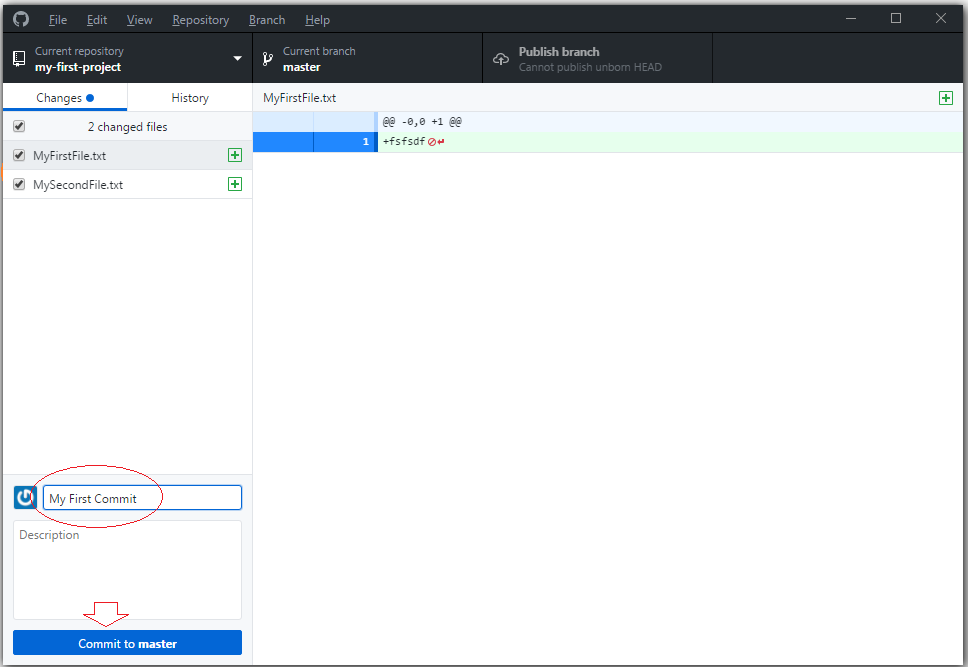
Copy một vài file dữ liệu của bạn vào **Local Repository**:

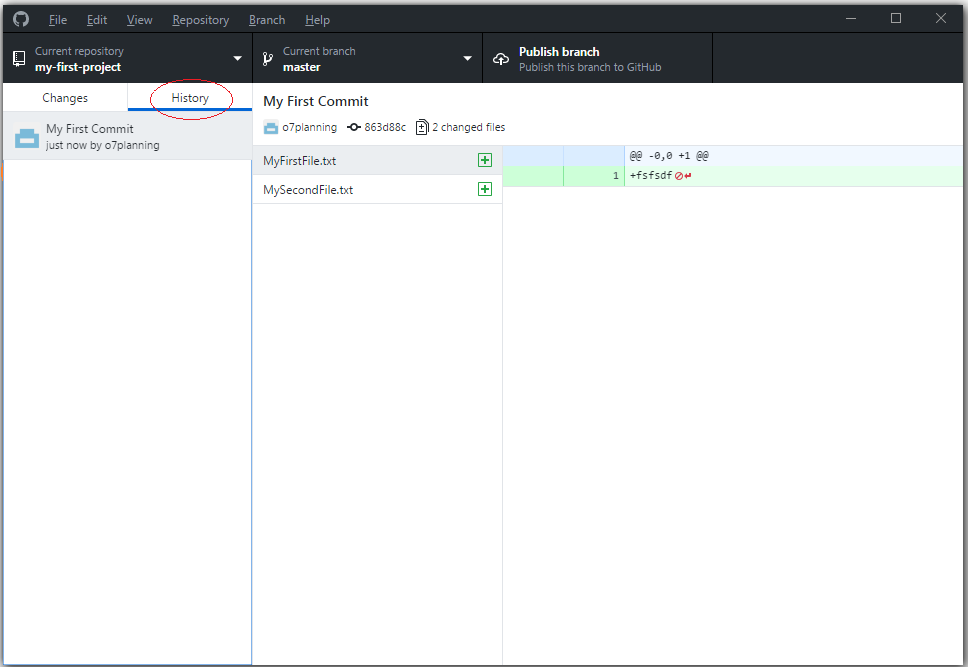


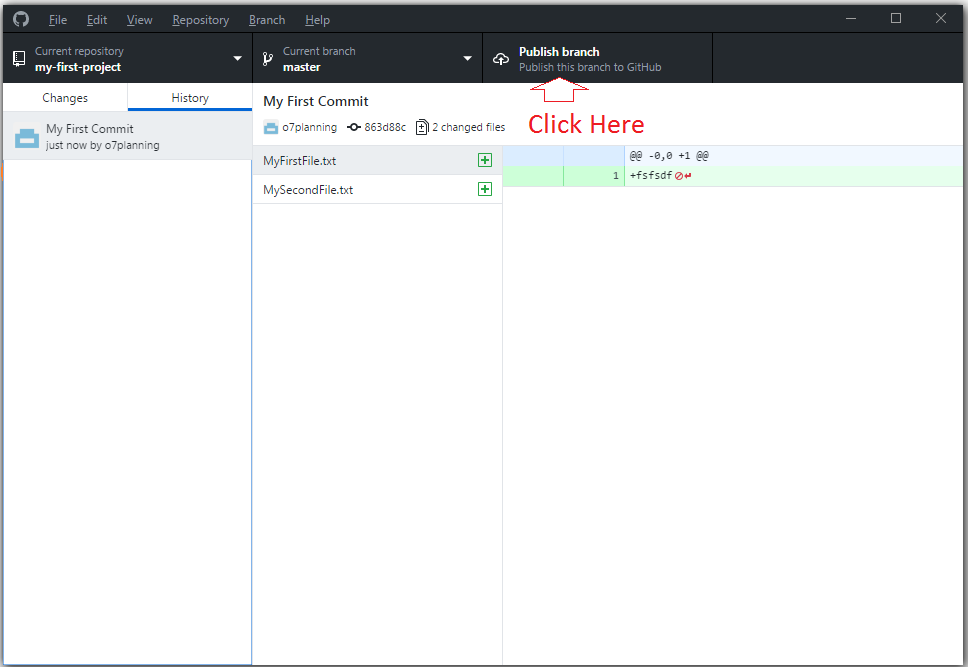
**GitHub Desktop** ngay lập tức nhận biết được các thay đổi tại **Local Repository**.



Nhập thông tin ghi chú (Comment) và nhấn **Commit**dữ liệu.







Các file dữ liệu bạn có thể nhìn thấy trên **Server**.

