

***Github***

**Hà nội,28/10/2018**

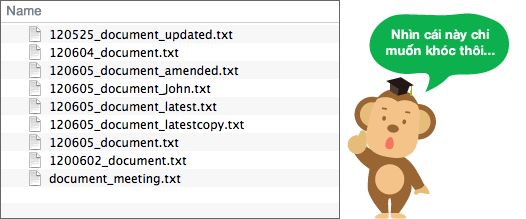
**Họ tên: Bùi Công Quyền**

**MSSV:SE05625-SE1218**

**I.Tìm hiểu về git,github**

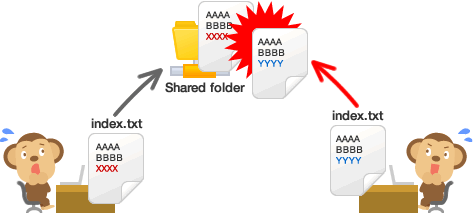
Khi muốn đưa về trạng thái trước khi chỉnh sửa của file thì mọi người sẽ làm như thế nào?

Cách đơn giản nhất đó là sao chép lại file trước khi chỉnh sửa. Trường hợp dùng phương pháp này thì sẽ phải thường xuyên thực hiện việc thêm ngày đã thay đổi vào tên thư mục hay file. Tuy nhiên, việc tự mình sao chép file mỗi lần chỉnh sửa thì sẽ rất vất vả, và cũng dễ xảy ra nhầm lẫn.



Hơn nữa, trường hợp đã đặt tên một cách vô trật tự như hình trên thì sẽ không thể phân biệt được file nào là bản mới nhất. Thêm nữa, với file đang chia sẻ và làm việc trong nhóm thì cũng sẽ có gắn tên của người chỉnh sửa. Nhưng lại không thể dễ dàng biết được đã thực hiện thay đổi như thế nào.

Ngoài ra, trường hợp file đang chia sẻ trong nhóm, nếu có hai người chỉnh sửa cùng lúc, thì sẽ xảy ra trường hợp xóa mất nội dung thay đổi của người đã chỉnh sửa trước đó.



Và để giải quyết những vấn đề này thì các hệ thống quản lý phiên bản như Git được tạo ra.

1. ***Thế nào là git github***

**1.1:*Định nghĩa về git:* 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**. Đại loại là như vậy.

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

*Mô hình hoạt động của DVCS*

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.

***1.2:Định nghĩa github***:GitHub là một dịch vụ lưu trữ trên web dành cho các dự án có sử dụng hệ thống kiểm soát Git revision. GitHub cung cấp chức năng social networking như là feeds, followers và network graph để các Developer học hỏi kinh nghiệm làm việc thông qua lịch sử commit. Nếu comment dùng để mô tả chức năng của đoạn code, thì commit message trên Git dùng để mô tả hành động mà Dev vừa thực hiện trên code. **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

## 1.3: 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 những 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 như từng làm với SVN, nó sẽ chiếm space ở server cũng như làm mất thời gian của bạn vào việc tương tác kết nối với server, vậy tại sao không commit nó lên repository local nhỉ, vừa nhanh thao tác lại không mất space của server.

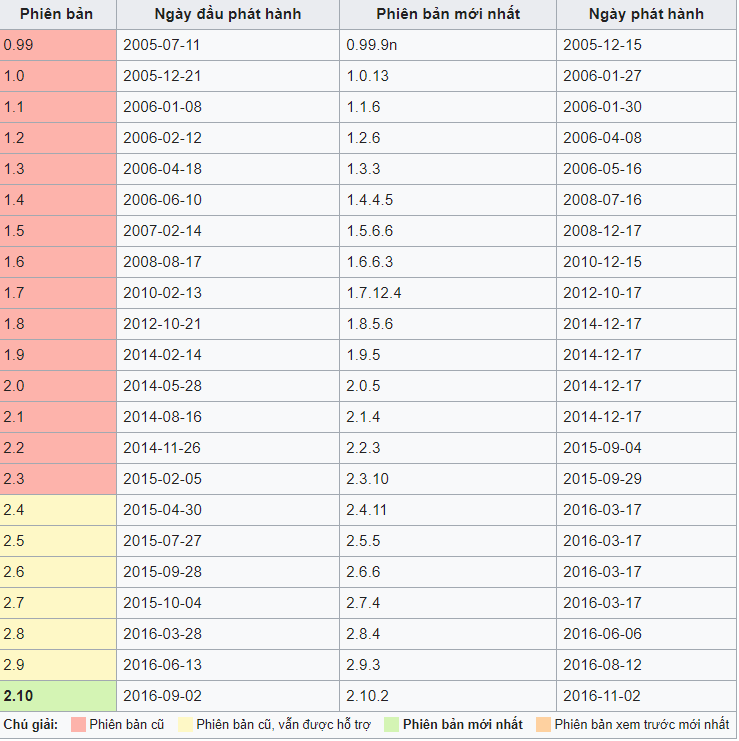
1. ***Lịch sử phát triển:***

Cũng như nhiều thứ tuyệt vời khác trong cuộc sống, Git ra đời từ một chút của sự huỷ diệt/phá sản/kết thúc có tính sáng tạo và sự tranh cãi nảy lửa. Nhân của Linux là một dự án phần mềm mã nguồn mở của một phạm vi khá lớn. Trong phần lớn thời gian bảo trì của nhân Linux (1991-2002), các thay đổi của phần mềm được truyền đi dưới dạng các bản vá và các tập tin lưu trữ. Vào năm 2002, dự án nhân Linux bắt đầu sử dụng một DVCS độc quyền có tên là BitKeeper.

Vào năm 2005, sự hợp tác giữa cộng đồng phát triển nhân Linux và công ty thương mại phát triển BitKeeper bị phá vỡ, và công cụ đó không còn được cung cấp miễn phí nữa. Chính điều này đã thúc đẩy cộng đồng phát triển Linux (chính xác hơn là Linus Torvalds, người sáng lập ra Linux) phát triển công cụ của riêng họ dựa trên những bài học từ việc sử dụng BitKeeper. Một số mục tiêu của hệ thống mới được vạch ra như sau:

* Nhanh
* Thiết kế đơn giản
* Hỗ trợ tốt cho "phát triển phi tuyến tính" (non-linear development) - (hàng ngàn nhánh song song)
* Phân tán toàn diện
* Có khả năng xử lý các dự án lớn giống như nhân Linux một cách hiệu quả (về mặt tốc độ và khối lượng dữ liệu)

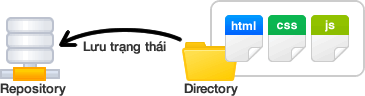
Kể từ khi ra đời năm 2005, Git đã tiến hoá và phát triển toàn diện để dễ dàng sử dụng hơn, tuy thế các tiêu chí ban đầu vẫn được đảm bảo. Nó nhanh một cách đáng kinh ngạc, vô cùng hiệu quả với các dự án lớn, và một hệ thống phân nhánh không thể tin được cho phát triển phi tuyến tính



1. ***Tìm hiểu vê git***

# *3.1:* Repository quản lý lịch sử

Repository là nơi sẽ ghi lại trạng thái của thư mục và file. Trạng thái được lưu lại đang được chứa như là lịch sử thay đổi của nội dung. Bằng việc đặt thư mục muốn quản lý lịch sử thay đổi dưới sự quản lý của repository, có thể ghi chép lại lịch sử thay đổi của thư mục và file trong thư mục đó.

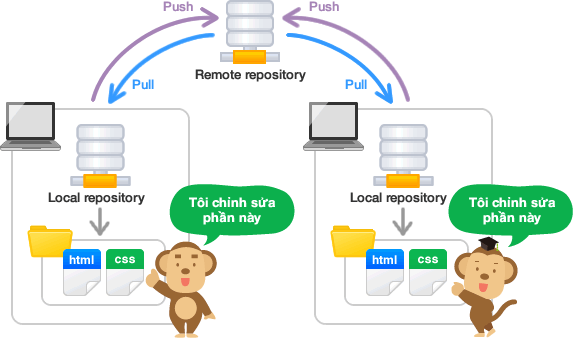


**Remote repository và local repository**

Đầu tiên, repository của Git được phân thành 2 loại là remote repository và local repository.

* 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 của bản thân mình, dành cho một người dùng sử dụng.

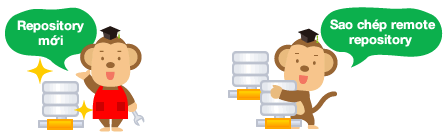
Do repository phân thành 2 loại là local và remote nên với những công việc bình thường thì có thể sử dụng local repository và thực hiện trên toàn bộ máy sẵn có. Khi muốn công khai nội dung công việc mà bản thân đã làm trên local repository, thì sẽ upload lên remote repository rồi công khai. Thêm nữa, thông qua remote repository cũng có thể lấy về nội dung công việc của người khác.



**Tạo repository**

Có 2 cách tạo local repository mà mình có sẵn.

Cách thứ nhất là tạo repository hoàn toàn mới, cách thứ hai là sao chép remote repository rồi tạo.



Trang tiếp theo sẽ trình bày

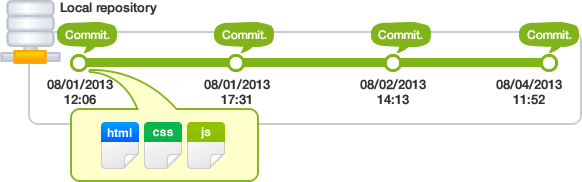
**3.2: commit**

# Commit sẽ ghi lại thay đổi

Để ghi lại việc thêm/ thay đổi file hay thư mục vào repository thì sẽ thực hiện thao tác gọi là Commit.

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.

Commit này đang được chứa tại repository trong trạng thái đã nối tiếp với nhau theo thứ tự thời gian như trong hình dưới đây. Bằng việc lần theo commit này từ trạng thái mới nhất thì có thể hiểu được lịch sử thay đổi trong quá khứ hoặc nội dung thay đổi đó.



Các commit này, được đặt tên bởi 40 ký tự alphabet không trùng nhau đã được tính toán từ thông tin commit. Bằng việc chỉ định tên này, có thể chỉ định commit từ trong repository.

TTip

Những thay đổi mang ý nghĩa khác nhau chẳng hạn như thêm chức năng hay sửa lỗi thì hãy cố gắng chia ra rồi commit. Để sau này khi xem lịch sử và tìm kiếm một nội dung thay đổi định sẵn sẽ dễ dàng hơn.

Khi thực hiện commit có yêu cầu nhập giải thích commit (commit message). Vì commit message là bắt buộc nên nếu để trống mà thực hiện thì commit sẽ thất bại.

TTip

Vì phần giải thích đó sẽ trở thành thông tin quan trọng khi bản thân xem lại lịch sử sau đó hay trường hợp tìm kiếm nội dung thay đổi mà người khác commit, nên hãy chú ý để viết giải thích sao cho dễ dàng hiểu được nội dung thay đổi. Viết giải thích theo hình thức tiêu chuẩn trên Git như sau:

Dòng thứ 1: Tóm tắt nội dung thay đổi trong commit

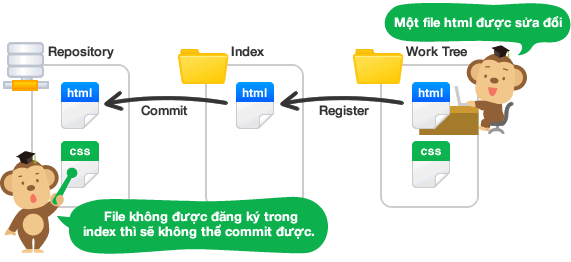
Dòng thứ 2: Dòng trống

Dòng thứ 3 trở đi: Lý do đã thay đổi

# 3.3: 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.

Và trên Git, giữa repository và working tree tồn tại một nơi gọi là index. Index là nơi để chuẩn bị cho việc commit lên repository.



Trên Git, khi đã thực hiện commit thì trạng thái sẽ không được ghi trực tiếp trong repository từ working tree, mà sẽ ghi trạng thái đã được thiết lập của index được xây dựng ở giữa đó. Vì thế, để ghi lại trạng thái của file bằng commit thì trước hết cần đăng ký file trong index.

Với việc chèn index vào giữa như thế này, có thể thực hiện commit không bao gồm những file không cần thiết trong working tree, hay có thể đăng ký chỉ một phần thay đổi của file trong index rồi commit.

### 4: Tại sao nên sử dụng Git?

Có rất nhiều lợi thế để bạn nên sử dụng Git trong việc lập trình ngay từ hôm nay, bất kể là lập trình cái gì đi chăng nữa.

* Git dễ sử dụng, an toàn và nhanh chóng.
* Có thể giúp quy trình làm việc code theo nhóm đơn giản hơn rất nhiều bằng việc kết hợp các phân nhánh (branch).
* Bạn có thể làm việc ở bất cứ đâu vì chỉ cần clone mã nguồn từ kho chứa hoặc clone một phiên bản thay đổi nào đó từ kho chứa, hoặc một nhánh nào đó từ kho chứa.
* Dễ dàng trong việc deployment sản phẩm.
* Và nhiều hơn thế nữa.

Nếu bạn là một lập trình viên thì Git là một hệ thống bạn cần phải biết cách sử dụng, ít nhất là ngay từ bây giờ.

### 5. Công cụ chuẩn bị

Để sử dụng Git thì hầu như không cần chuẩn bị công cụ gì, ngoài việc mình khuyến khích các bạn nên thực hành trên hệ điều hành Linux như Ubuntu, Mac,…bạn có thể dùng phần mềm VMWare Workstation tạo ra một máy ảo và cài Ubuntu Desktop vào. Tuy nhiên nếu bạn vẫn muốn sử dụng Git trên Windows thì cũng được thôi nhưng do Git là một hệ thống phát triển trên nền Linux nên bạn dùng Linux sẽ tiện hơn.

# II.Cách sử dụng github

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:

[Những việc làm hấp dẫn](https://topdev.vn/)



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.

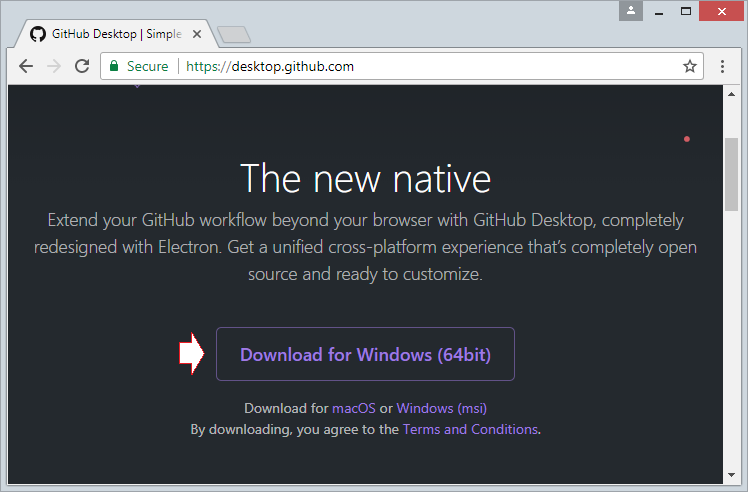


4- Download & cài đặt GitHub Desktop

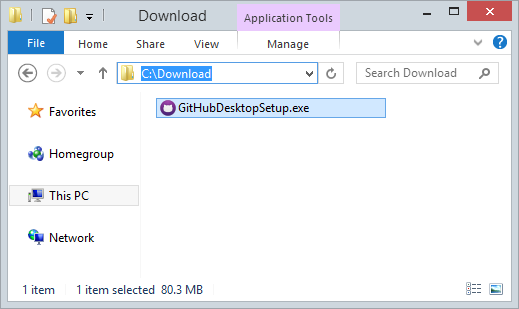
**GitHub Desktop** về bản chất là một công cụ trực quan cho phép bạn quản lý **Local Repository** (Kho chứa địa phương) trên máy tính của bạn.

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

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

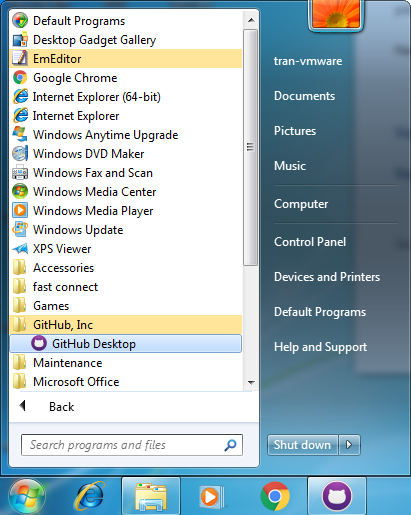


Kết quả download:

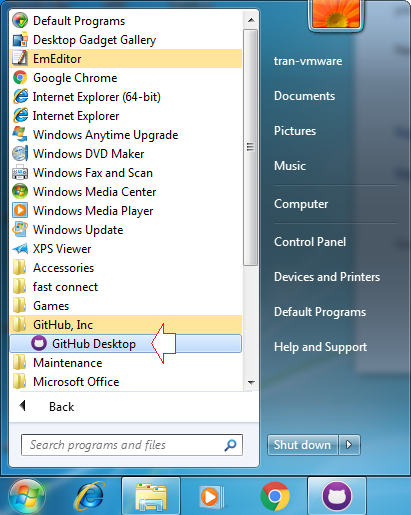


Sau khi download xong, bạn cần cài đặt **GitHub Desktop** vào máy tính:

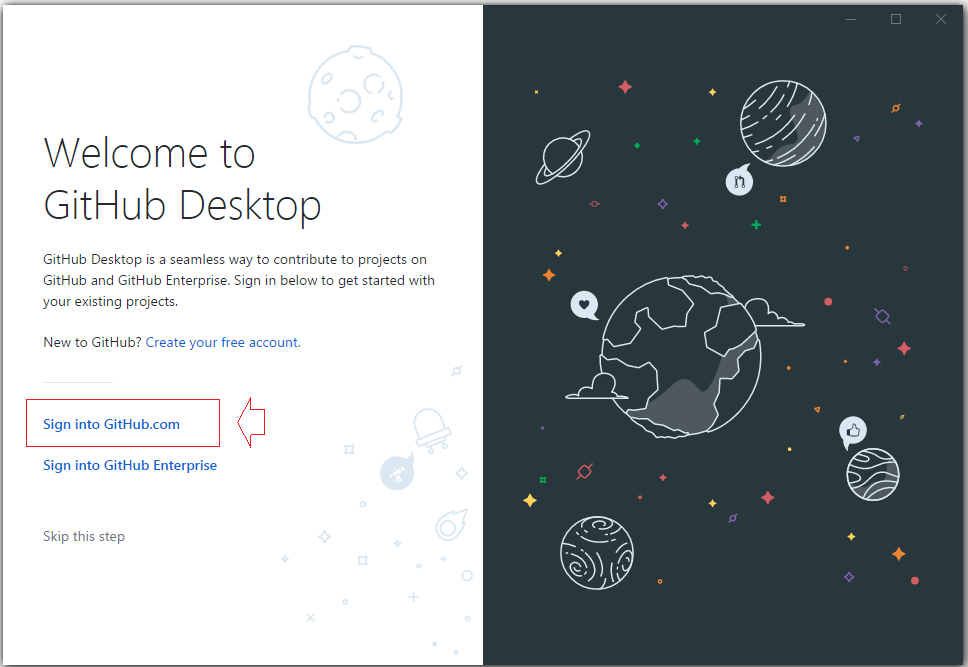
**GitHub Desktop**đã được cài đặt thành công.

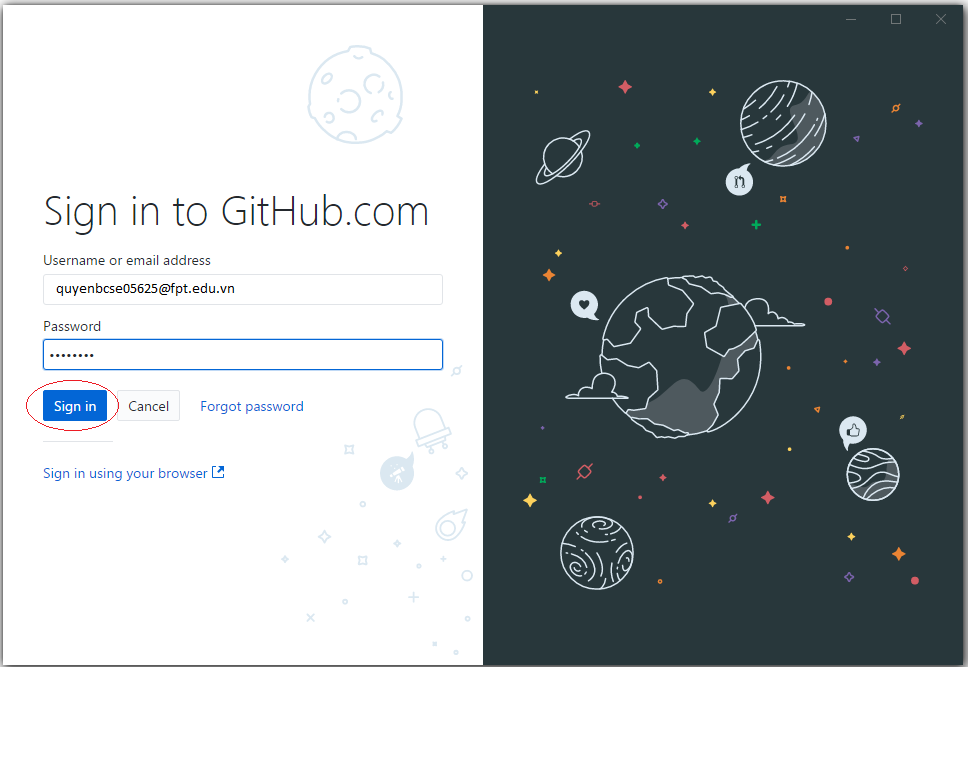


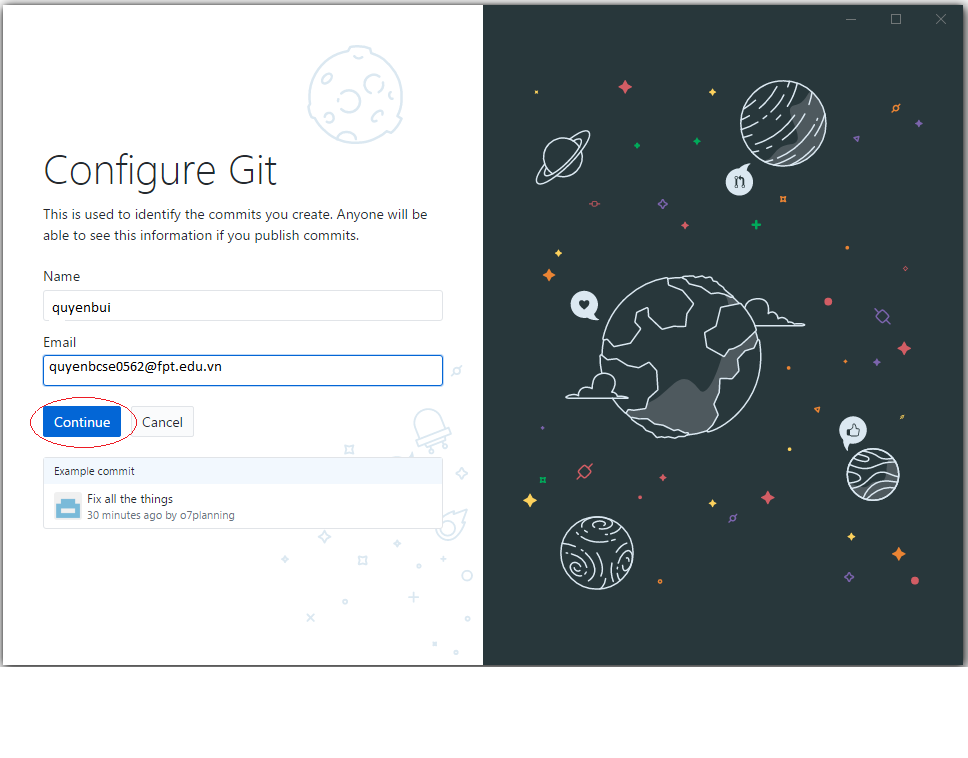
5- Chạy GitHub Desktop

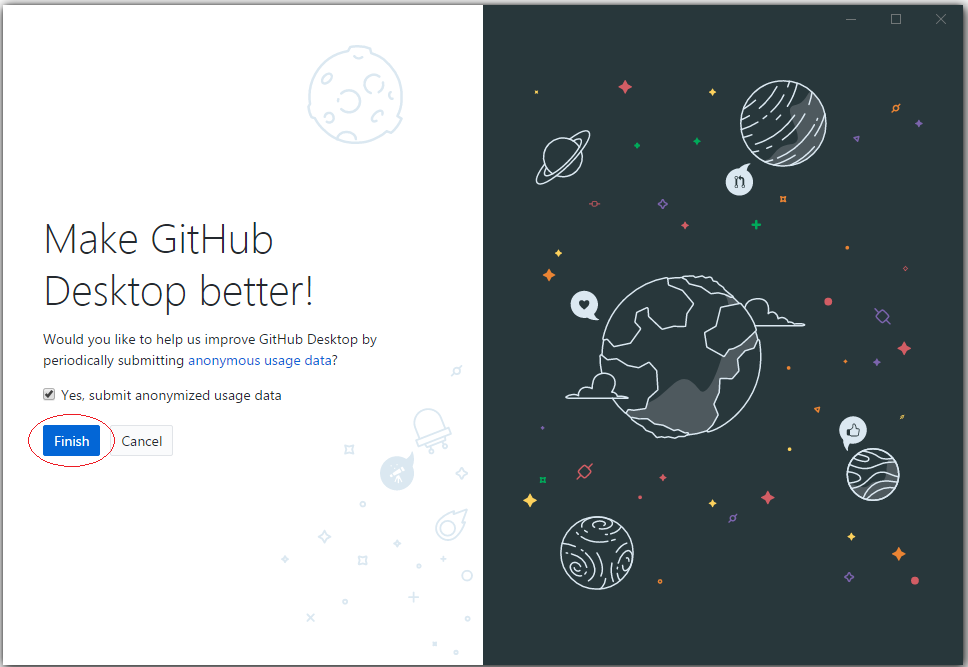


Đăng nhập trên **GitHub Desktop** để kết nối vào tài khoản **GitHub**của bạn.











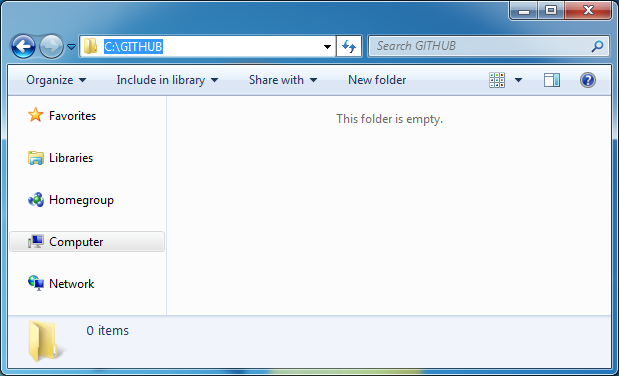
Cho tới lúc này trên máy tính của bạn chưa có một **Local Repository** nào.

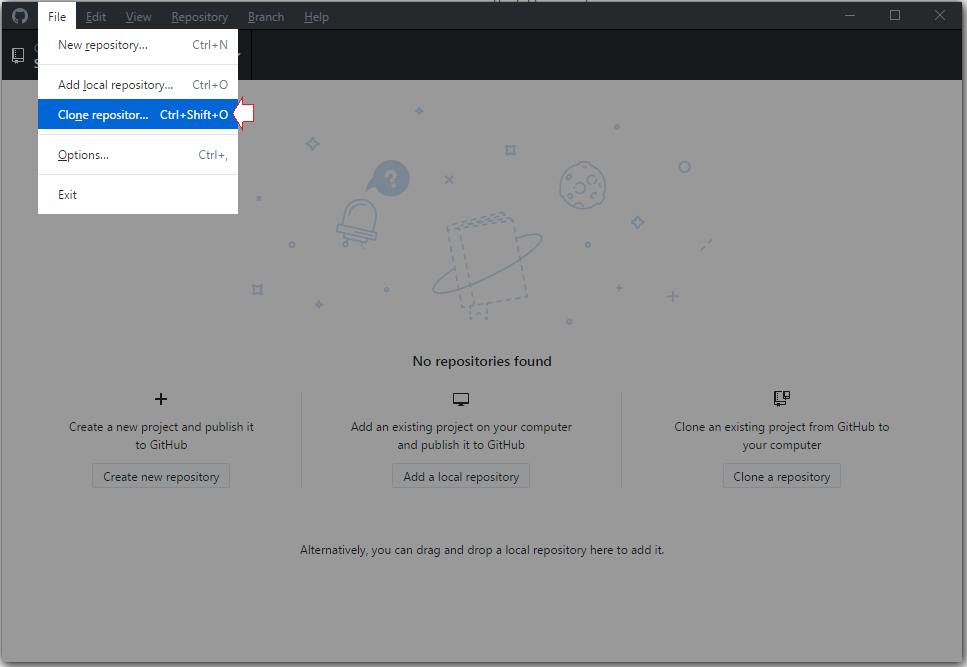
6- Kết nối GitHub và GitHub Desktop

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

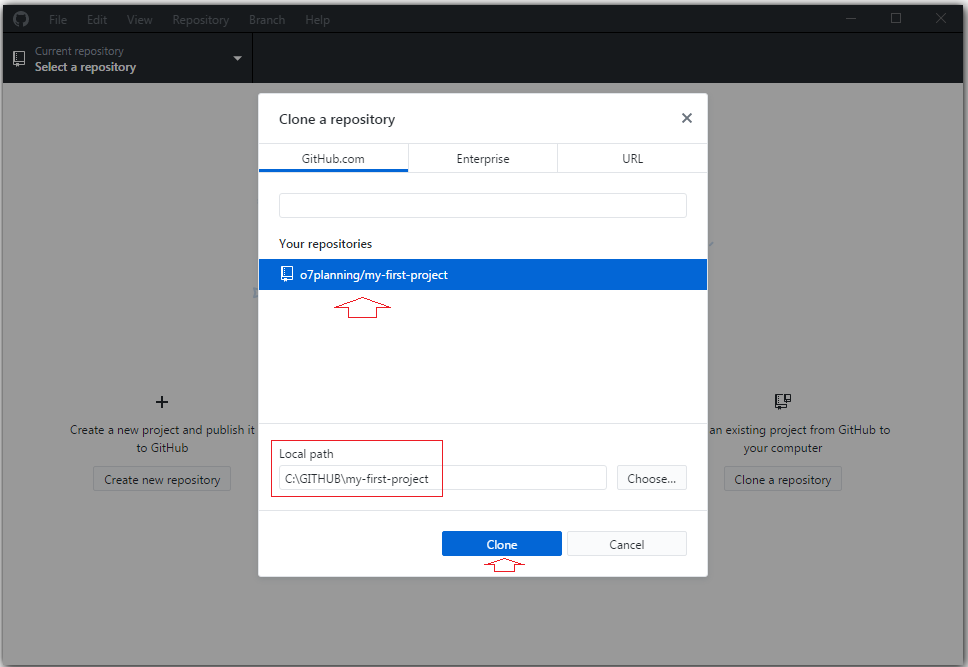
Chẳng hạn:

* **C:/GITHUB**

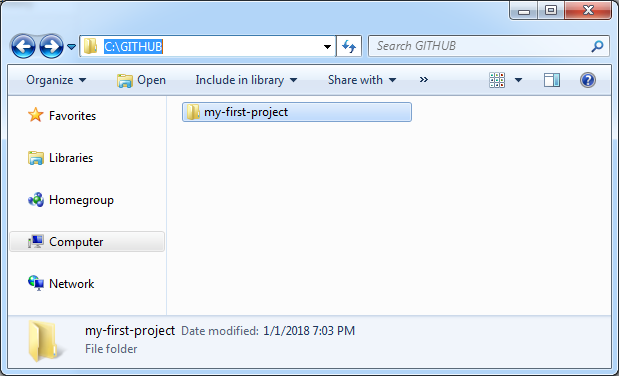


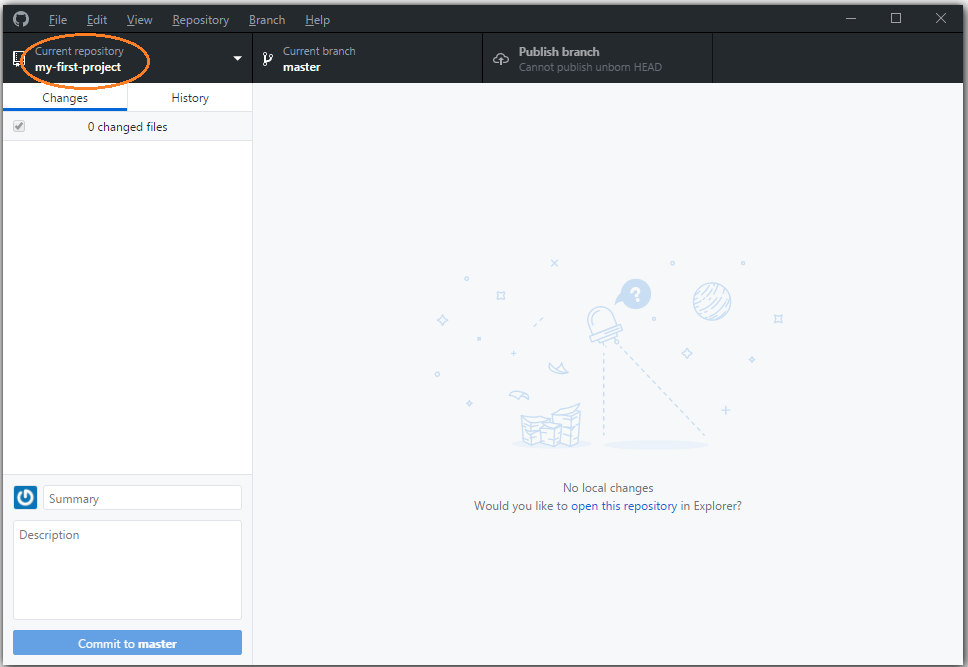


Trên **GitHub Desktop**, lựa chọn một **Repository**bạn đã tạo trên **GitHub** để **clone**(Tạo bản sao chép) thành một bản ở máy tính địa phương của bạn.

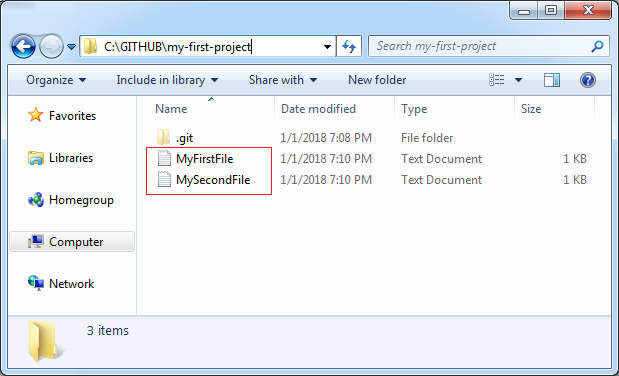


Lúc này trên **GitHub Desktop** bạn sẽ thấy một **Local Repository** đã được tạo ra.

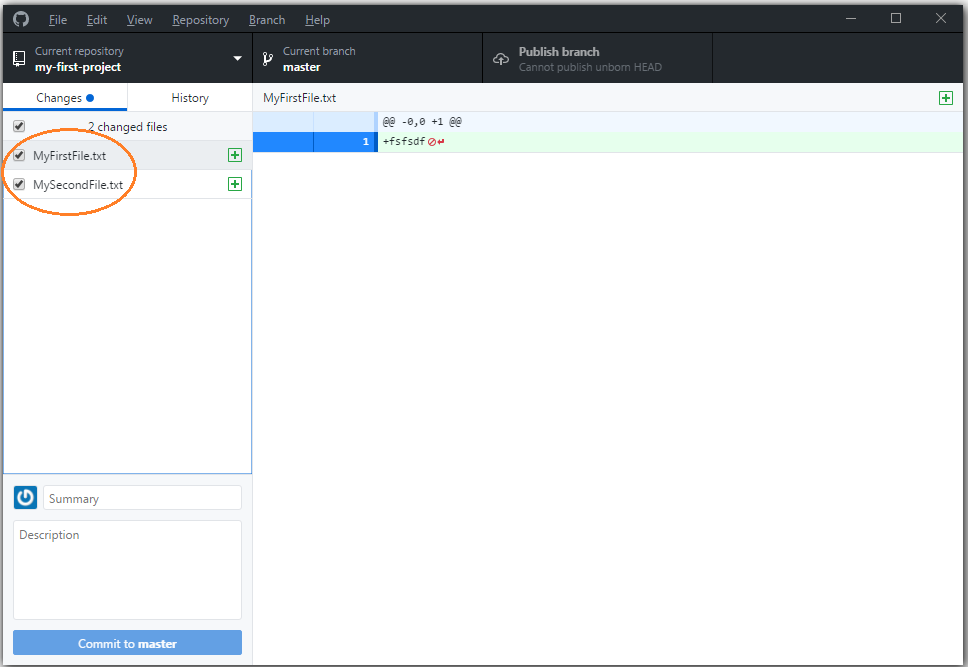




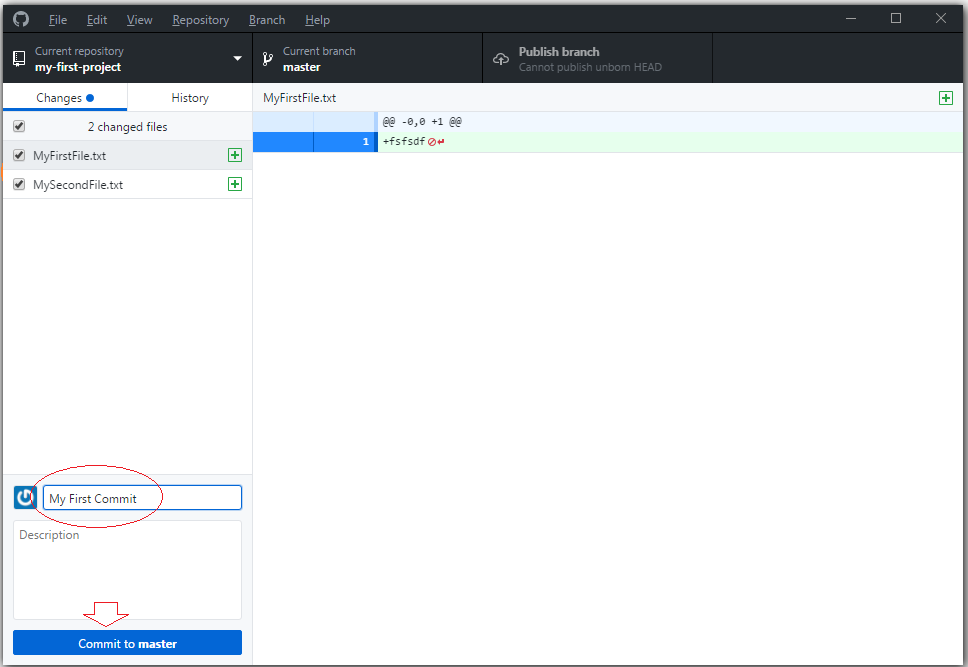
Copy một vài file dữ liệu của bạn vào **Local Repository**:

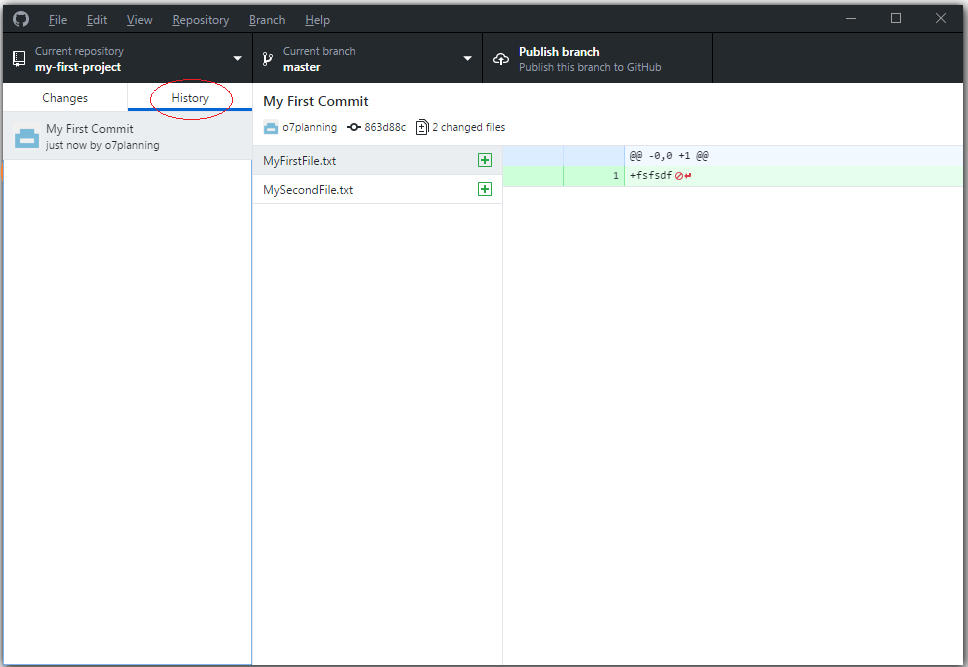


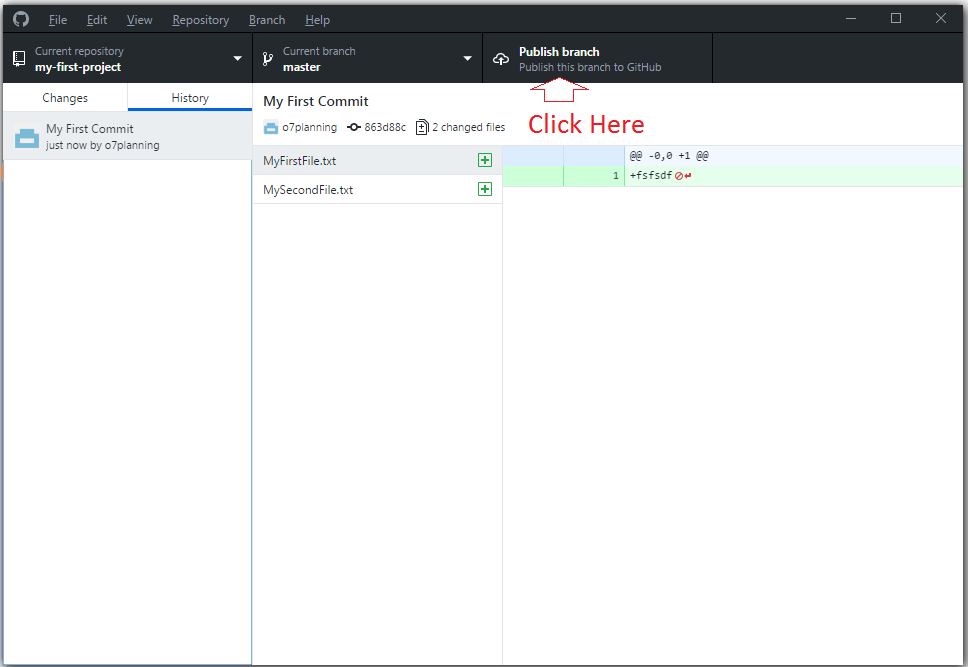
**GitHub Desktop** ngay lập tức nhận biết được các thay đổi tại **Local Repository**.



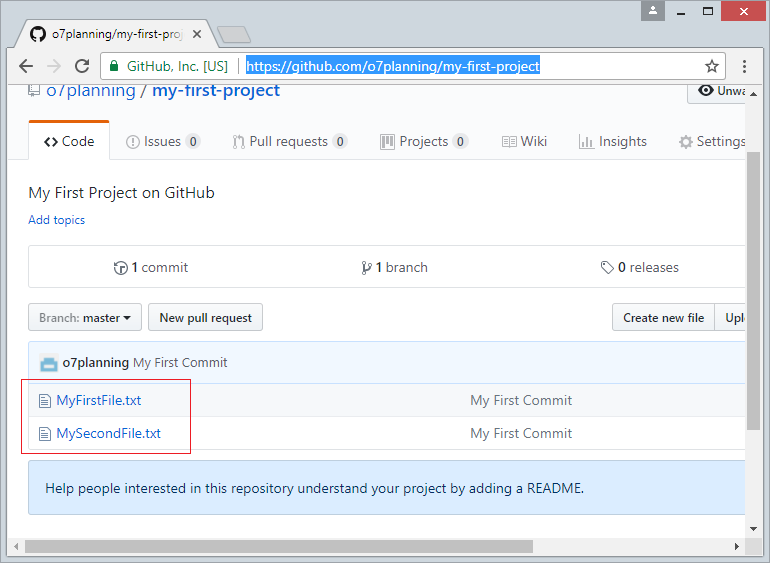
Nhập thông tin ghi chú (Comment) và nhấn **Commit**dữ liệu.







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



# 5:1 vài câu lệnh git hữu ích

### Git config

git config --global user.name "John Doe"

git config --global user.email "john@example.com"

--global được sử dụng để áp dụng cho tất cả các projects. Nếu bạn ko sử dụng --global thì settings sẽ chỉ dùng cho riêng project đó.

### Giúp Git bỏ qua file modes

cd project/

git config core.filemode false

Câu lệnh trên hữu dụng khi chúng ta không cần quan tâm đến quyền truy cập files (ví dụ như khi sử dụng Windows).

### Liệt kê những settings đang sử dụng

git config --list

### Khởi tạo Git repo cho code có sẵn

cd existing-project/

git init

### Clone một remote repo

git clone https://github.com/user/repository.git

Câu lệnh trên sẽ tạo một thư mục mới có tên giống trên của repo.

### Clone một remote repo tại thư mục hiện tại

git clone https://github.com/user/repository.git .

### Xem thông tin trợ giúp cho một câu lệnh git

git help clone

### Update và merge branch hiện tại với một remote repo

cd repo/

git pull origin master

với origin là remote repo, master là remote branch.

Nếu bạn không muốn merge những thay đổi của bạn, hãy sử dụng git fetch

### Liệt kê các remote urls

git remote -v

### Thay đổi origin url

git remote set-url origin https://github.com/repo.git

### Thêm remote repo

git remote add remote-name https://github.com/user/repo.git

### Xem thay đổi (chưa đc add) của những file hiện tại

git diff

### Xem thay đổi (đã được add, chưa commit)

git diff --cached

### Xem thay đổi giữa local mà master

git diff origin/master

### Xem thay đổi giữa hai commits

git diff COMMIT1\_ID COMMIT2\_ID

### Xem những files thay đổi giữa hai commits

git diff --name-only COMMIT1\_ID COMMIT2\_ID

### Xem những files thay đổi tại một commit bất kỳ

git diff-tree -no-commit-id --name-only -r COMMIT\_ID

hoặc

git show --pretty="format:" --name-only COMMIT\_ID

### Xem thay đổi trước khi push

git diff --cached origin/master

### Xem thông tin cụ thể của một commit

git show COMMIT\_ID

### Kiểm tra status của working tree

git status

### Tạo vài thay đổi, rồi commit

git add changed\_file.txt

git add folder-with-changed-files/

git commit -m "Commiting changes"

### Đổi tên/Di chuyển/Xoá files

git rm removeme.txt tmp/crap.txt

git mv file oldname.txt file\_newname.txt

git commit -m "deleteing 2 files, renaming 1"

### Đổi message của commit cuối

git commit --amend -m "New commit mesage"

### Push local commits sang nhánh remote

git push origin master

### Xem commit history

git log

### Xem commit history cho hai commits gần nhất

git log -2

### Xem commit history cho hai commits gần nhất, bao gồm cả thay đổi

git log -p -2

### Xem commit history dưới dạng một dòng

git log --pretty=oneline

### Revert một commit rồi push

git revert COMMIT\_ID

git push origin master

### Revert đến thời điểm trước một commit

git reset COMMIT\_ID

git reset --soft HEAD@{1}

git commit -m "Revert to COMMIT\_ID"

git reset --hard

### Undo commit gần nhất, vẫn giữ thay đổi ở local

git reset --soft HEAD~1

### Undo commit gần nhất, không giữ thay đổi ở local

git reset --hard HEAD~1

### Undo commit gần nhất, vẫn giữ thay đổi ở index

git reset --mixed HEAD~1

hoặc

git reset HEAD~1

### Undo commits chưa push

git reset origin/master

### Reset về trạng thái của remote

git fetch origin

git reset --hard origin/master

### Xem các nhánh local

git branch

### Xem tất cả các nhánh

git branch -a

### Tạo một patch

git diff > patch-issue-1.patch

### Thêm một file rồi tạo patch

git add newfile

git diff --staged > patch-issue-2.patch

### Thêm một file, thay đổi rồi tạo patch

git add newfile

git diff HEAD > patch-issue-2.patch

### Tạo patch từ một commit

git format-patch COMMIT\_ID

### Tạo patch từ hai commit cuối

git format-patch HEAD~2

### Tạo patch từ tất cả những commits chưa push

git format-patch origin/master

### Tạo patch chứa dữ liệu nhị phân

git format-patch --binary --full-index origin/master

### Apply một patch

git apply -v patch-name.patch

### Apply một patch được tạo bằng format-patch

git am patch1.patch

### Tạo một tag

git tag 7.x-1.3

### Push một tag

git push origin 7.x-1.3

### Tạo một nhánh

git checkout master

git branch new-branch-name

Lưu ý với hai câu lệnh trên thì chúng ta chưa chuyển sang nhánh mới, mà vẫn ở nhánh master. Phải sử dụng thêm git checkout new-branch-name để chuyển nhánh.

Ngoài ra có thể tạo nhánh mới và chuyển sang luôn bằng 1 câu git checkout -b new-branch-name

### Chuyển nhánh

git checkout new-branch-name

### Xem commit history so với branch hiện tại

git cherry -v master

master ở đây là branch mà bạn muốn so sánh

### Merge commit từ branch khác

git checkout master

git merge branch-name

Ở đây chúng ta merge các commits của branch-name vào master.

### Merge branch mà không commit

git merge branch-name --no-commit --no-ff

### Xem thay đổi giữa state hiện tại và một branch

git diff branch-name

### Xem thay đổi trong một file, giữa state hiện tại và một branch

git diff branch-name path/to/file

### Xoá branch

git branch -d branch-name

### Push lên một branch

git push origin branch-name

### Lấy tất cả các branches

git fetch orgin

### Lấy thư mục root

git rev-parse --show-toplevel

### Xoá các file bị xoá ở local trên repo

git rm $(git ls-files --deleted)

### Xoá toàn bộ các files chưa đc track

git clean -f

xoá cả folder:

git clean -f -d

xem các file trước khi xoá:

git clean -n -f -d

### Unstage các files

git reset HEAD file.txt

### Xem tag gần nhất

git describe --tags `git rev-list --tag --max-count=1`

### Liệt kê các nhánh theo trình tự sử dụng gần nhất

git for-each-ref --sort=-committerdate refs/heads/ | head

### Tar cả project, ngoại trừ thư mục .git

cd ..

tar cJf project.tar.xz project/ --exclude-vcs

### Tar tất cả các files bị thay đổi ở local

git diff --name-only | xargs tar -cf project.tar -T -

### Tìm conflict

grep -H -r "<<<" \*

grep -H -r ">>>" \*

grep -H -r '^=======$' \*

### Apply một patch không sử dụng git

patch < file.patch