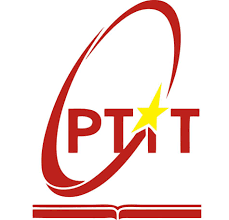
**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG  
CƠ SỞ THÀNH PHỐ HỒ CHÍ MINH**

**🙢🕮🕮🙠**



**NHẬP MÔN CÔNG NGHỆ PHẦN MỀM**

**ĐỀ TÀI**

**TÌM HIỂU VỀ GIT VÀ GITHUB**



**Giảng viên hướng dẫn : TS. Võ Xuân Thể**

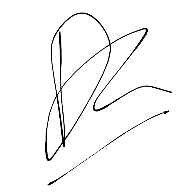
**Sinh viên thực hiện : Nguyễn Đức Khải**

**Lớp : D17CQCN02-N**

**MSSV : N17DCCN066**

**TP.HCM-2020**

<6C>.<06>.<N17DCCN066>

**Họ tên** :NGUYỄN ĐỨC KHẢI

**Ký tên** :

**Học viện**  :Công nghệ Bưu chính Viễn thông

**Môn**  :Nhập môn công nghệ phần mềm

**Tên Chủ đề (CNPM mới)** :Tìm hiểu về Git và GitHub

**Nguồn** :<https://github.com/>, <https://git-scm.com/>

1. **Giới thiệu**

TÌM HIỂU VỀ GIT VÀ GITHUB

1. **Lý do nghiên cứu**

Git và GitHub được sử dụng khá phổ biến trong giới lập trình viên, chắc hẳn bạn sẽ thấy nhiều đường dẫn kho mã nguồn có link từ GitHub.com. Nó rất tiện lợi và an toàn, đáp ứng tốt các nhu cầu làm việc nhóm. Có thể nói, biết sử dụng Git ở mức cơ bản là kỹ năng mà một lập trình viên bắt buộc phải có.

1. **Giới thiệu sơ lược**
   1. **Sơ lược về Hệ thống VCS**

Hệ thống quản lý phiên bản (Version control system - VCS) là một dạng phần mềm Quản lý mã nguồn (Source Code Management- SCM)

Nó gồm 3 chức năng chính:

* Ghi lại lịch sử: ghi nhận và lưu lại sự thay đổi của các file theo thời gian
* Chức năng phục hồi: cho phép khôi phục trạng thái (phiên bản) ở một thời điểm trước đó khi phát hiện lỗi trong quá trình làm.
* Chức năng đồng bộ: có thể theo dõi sự thay đổi của file theo thời gian, ai đã thay đổi, thay đổi vào lúc nào.

Có rất nhiều hệ thông VCS hiện nay như: Concurrent Versions System, Subversion, Git, Mercurial, …

Một số khái niệm thường dùng trong VCS

* Commit: Là thao tác ra lệnh cho VCS ghi lại những thay đổi mà tác giả thực hiện với các file mà VCS đang theo dõi. Ngoài ra còn bổ sung themcác chú thích từ tác giả cho commit này.
* Diff: dùng để phân tích sự khác nhau giữa 2 phiên bản nào đó . Phần khác nhau này có thể xuất ra thành một file gọi là “patch”.
* Applying patch: là việc chiết xuất những file sau sửa đổi bằng việc tổnghợp bản trước khi sửa đổi và bản batch.
  1. **Phân loại VCS theo cách ứng dụng**

VCS cục bộ:

* VCS sẽ tạo ra một Cơ sở dữ liệu lưu giữ những thay đổi của các file và giúp bạn phục hồi về một phiên bản cũ một cách dễ dàng. Hệ thống đó bạn có thể gọi là VCS cục bộ. Khi làm việc nhóm các developer thiết lập một hệ thống quản lý phiên bản trung tâm (CVCS) ở một máy chủ, ở đó chứa các phiên bản của file, các lập trình viên kết nối và lấy về sửa đổi rồi lại cập nhật lại.
* Tuy nhiên CVCS vẫn có nhược điểm như Server bị dừng thì không thể kết nối để các thành viên làm việc, Database ở Server bị lỗi (ví dụ do đĩa) thì khó phục hồi file đã mất.

VCS trung tâm

* Để khắc phục nhược điểm của CVCS, một loại VCS nữa được đề xuất đó là DVCS (Distributed Version Control Systems) - Hệ thống quản lý phiên bản phân tán, nó giống với CVCS là có một máy chủ ở đó có Database lưu giữ các phiên bản của file.
* Tuy nhiên khác biệt đó là các máy khách (các developer) kết nối vào thỉ nó không chỉ lấy file mà nó lấy luôn cả hệ thống Database. Điều này có nghĩa là khi Server bị ngắt, các máy khách vẫn làm việc bình thường trên Database ở máy trạm, sau đó chuyển (commit) lên Server sau, hoặc Database ở Server bị lỗi thì máy khách bất kỳ đều có thể phục hồi lại cho Server
  1. **Sơ lược về Git**

Git chính là một Hệ thống quản lý phiên bản phân tán, được phát triển vào năm 2005 bởi chính cha đẻ của hệ điều hành Linux –Linux Tovalds. Ban đầu Git được dành cho việc phát triển nhân Linux, tuy nhiên, hiện nay Git đã trở thành một trong các công cụ quản lý mã nguồn phổ biến và ưu việt nhất. Git hiện nay có khả năng chạy trên nhiều hệ điều hành khác nhau như: Linux, Windows, Mac OS, …

Kể từ khi ra đời vào năm 2005, Git đã phát triển và trưởng thành để trở nên dễ sử dụng hơn và vẫn giữ được các ưu điểm vốn có như tốc độ, hỗ trợ phân tán, … trở nên cực kì hiệu quả với các dự án lớn nhỏ.

* 1. **Sơ lược về GitHub**

Nếu Git là tên gọi của một mô hình hệ thống quản lý mã nguồn. Trong mô hình này, các máy tính có thể nhân bản (Clone) lại mã nguồn từ một kho chứa mã nguồn (repository) và GitHub chính là một hệ thống máy chủ dành riêng để chứa các kho mã nguồn Git dựa trên nền web cho các dự án phát triển phần mềm. GitHub có đầy đủ các tính năng của Git, ngoài ra nó còn bổ sung các tính năng để giúp những người phát triển có thể tương tác với nhau.

GitHub được phát hành và chạy chính thức vào tháng 4/2008 và được Microsoft mua lại vào tháng 6/2018. Tính đến tháng 3/2018, GitHub có hơn 25 triệu người dử dụng với hơn 80 triệu kho mã nguồn, khiến nó trở thành máy chủ chứa mã nguồn lớn trên thế giới.

GitHub hiện nay đã trở thành một yếu tố có sức ảnh hưởng trong cộng đồng phát triển mã nguồn mở. Thậm chí nhiều nhà phát triển đã bắt đầu xem nó là một sự thay thế cho sơ yếu lý lịch và một số nhà tuyển dụng yêu cầu các ứng viên cung cấp liên kết tài khoản GitHub trong hồ sơ xin việc để đánh giá năng lực.

GitHub có hai phiên bản: miễn phí và trả phí. Với phiên bản có phí thường được các doanh nghiệp sử dụng để tăng khả năng quản lý cũng như phân quyền bảo mật dự án. Còn lại thì phần lớn mọi người đều sử dụng GitHub với tài khoản miễn phí để lưu trữ source code.

1. **Cách thức hoạt động và các chức năng cơ bản của Git và GitHub**
   1. **Cách hoạt động của Git**

Điểm khác biệt của Git so với các VCS khác (như Subversion…) chính là “cách Git nghĩ về dữ liệu”. Phần lớn các hệ thống khác coi thông tin được lưu trữ như là một tập hợp các tập tin kèm theo các thay đổi được thực hiện trên mỗi file theo thời gian.



Git lưu dữ liệu dưới dạng một loạt "ảnh chụp" (snapshot) của một hệ thống tập tin, có nghĩa là mỗi khi bạn chuyển (commit) trạng thái hiện tại của dự án, về cơ bản Git tiến hành “chụp một bức ảnh” ghi lại nội dung hệ thống các tập tin thời điểm đó và tạo ra một tham chiếu đến ảnh chụp đó. Nếu các tập tin không có thay đổi thì Git sẽ không lưu trữ tập tin đó lần nữa mà chỉ có một liên kết đến tập tin đã lưu ở lần trước đó.



Git lưu dữ liệu của kho mã nguồn trong thư mục “.git”. Khi bạn thực hiện các thao tác trong Git, phần lớn các hành động đó được thêm vào cơ sở dữ liệu của Git. Giống như trong các VCS khác, bạn có thể mất hoặc làm rối tung giữ liệu mà bạn chưa lưu trữ, nhưng khi bạn đã lưu trữ nó thì ít khi bị mất các dữ liệu đó. Rất khó để yêu cầu hệ thống thực hiện một hành động nào đó mà không thể khôi phục lại được hoặc xóa dữ liệu đi dưới mọi hình thức, vì vậy mà bạn có thể thoải mái thử nghiệm mà không sợ phá hỏng mọi thứ.

* 1. **Các chức năng cơ bản của Git và GitHub**
     1. **Kho mã nguồn (repository)**

Kho mã nguồn được hiểu đơn giản là một kho chứa toàn bộ project bao gồm source code và lịch sử thay đổi cũng như nội dung thay đổi của từng file và từng cá nhân đóng góp cho project đó, gồm 2 loại là:

* Remote repository: Kho này dùng để chia sẻ cho nhiều người và được đặt trên server chuyên dụng. Ví dụ: GitHub, GitLab, Bitbucket, …
* Local repository: Kho này được đặt trên máy cá nhân và chỉ dành cho một người. Repository này sẽ được đồng bộ với remote repository bằng các lệnh của git.
  + 1. **Sao lưu (Backup)**

Bao gồm các thao tác lưu lại trạng thái hiện tại trên hệ thống, ghi nhận lại lịch sử các xử lý: thêm, xóa, cập nhật các file hay thư mục trên repository.

Khi chuyển (commit) trạng thái của dự án lên server, kho mã nguồn sẽ ghi lại sự khác biệt giữa các lần commit trước so với hiện tại. Chúng được ghi nối tiếp nhau theo thứ tự thời gian. Do đó, bạn có thể xem lại lịch sử thay đổi trong quá khứ dựa theo các commit trước đó.

Sau khi commit thì có thể đẩy lên (push) một branch nào đó ở remote repository hoặc một branch mới hoàn toàn lên remote repository.

Sau khi push lên thì các thành viên của team có thể thấy và đồng bộ code xuống máy local thông qua tính năng cập nhật (pull).

* + 1. **Phân nhánh (Branching)**

Branch là nhánh của repository, các nhánh này sẽ độc lập với nhau và phát triển một tính năng hoặc làm một nhiệm vụ nào đó không gây ảnh hưởng đến các nhánh khác.

Mỗi repository có thể tạo nhiều nhánh khác nhau, và chúng độc lập với nhau nên khi bạn thực hiện các thao tác thêm xóa, cập nhật với project trên nhánh này sẽ không ảnh hưởng đến các nhánh khác. Branch mới được tạo ra sẽ chứa toàn bộ trạng thái đã thực hiện trên project ngay thời điểm trước khi được tạo.

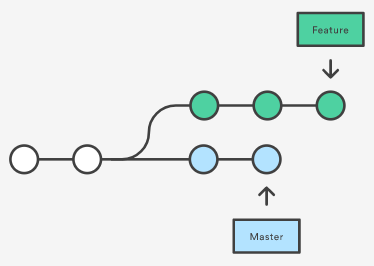
Branch mặc định là **master**. Có hai loại branch:

* Local branch: nhánh nằm trên máy tính cá nhân.
* Remote branch: nhánh nằm trên máy chủ từ xa.

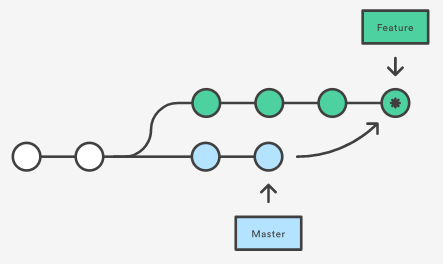
Một branch trên local có thể liên kết với 1 hoặc nhiều branch trên remote hoặc không branch nào cả. Git cho phép và khuyến khích bạn tạo nhiều branch tại máy tính cá nhân của bạn, việc tạo, sáp nhập và xóa bỏ những branch này diễn ra rất nhanh chóng. Việc chuyển đổi trạng thái code từ branch này sang branch khác hoàn toàn dễ dàng, nếu như chức năng mới của bạn bị lỗi, bạn hoàn toàn có thể xóa branch đó đi mà chẳng gây ảnh hưởng những chức năng khác.

* + 1. **Hợp nhất nhánh (Merge và Rebase)**

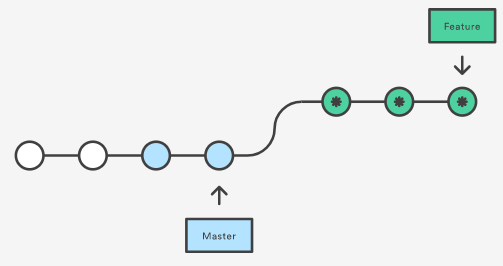
Là việc hợp nhất một nhánh phát triển hoặc hợp nhất lịch sử thay đổi vào nhánh khác. Sau khi viết chương trình để thực hiện chức năng mới ở branch nhỏ, tiến hành kiểm thử, nếu tất cả đều không lỗi thì có thể hợp nhất vào nhánh chính (thường là branch master).

Tuy có cùng chức năng là gộp nhánh nhưng cách hoạt động của merge và rebase lại khác với nhau, ta có thể so sánh để thấy sự khác biệt cơ bản đó qua mô tả như sau, Giả sử có 2 branchs cần merge như sau:

Sử dụng **merge** sẽ tạo ra một commit mới là kết hợp từ 2 commit cuối cùng của 2 nhánh cần gộp vào với nhau, log commit sẽ không bị thay đổi và thứ tự các commit sẽ được sắp xếp theo dòng thời gian tạo commit, kết quả như thế này:



Sử dụng **rebase** sẽ đưa toàn bộ commit của feature branch lên đầu, đồng nghĩa với việc sẽ làm thay đổi thời gian của lịch sử commit, kết quả như thế này:

Vậy kết luận khác biệt rõ ràng dễ thấy nhất là rebase sẽ đưa commit lên đầu branch sau khi gộp còn merge sẽ giữ lại lịch sử commit theo dòng thời gian.

* + 1. **Tạo bản sao kho mã nguồn (Fork và Clone)**

Mỗi người dùng đều có thể tạo một bản sao lưu đầy đủ dữ liệu trên máy chủ chính về máy tính cá nhân của mình, gọi hành động này gọi là clone.

Ngoài ra đối với các dự án phát triển thêm các tính năng mới (lớn) mà không muốn làm thay đổi code cũ trên GitHub (core), bạn sẽ dùng đển fork. Fork là hành động tạo một bản sao của repository gốc thành một repository của bạn. Việc fork một repository cho phép bạn dễ dàng chỉnh sửa, thay đổi source code mà không ảnh hưởng tới source gốc. Bạn cũng có thể cập nhật mã nguồn của mình vào dự án gốc (core) nếu được cho phép hoặc có thể cập nhập các thay đổi mới ở core về project của mình.

* + 1. **Các câu lệnh cơ bản trong Git**

1. **git config**

Tác dụng: Để thiết lập user name và email của bạn trong tập tin cấu hình (main configuration file).

Cách xài: Để kiểm tra tên và kiểu email trong cấu hình dùng **git config  -- global user.name** và **git config -- global user.email**. Để thiết lập email hoặc tên mới **git config  -- global user.name = “your username”** và **git config --global user.email = “youremail@email.com”**

1. **git init**

Tác dụng: Khởi tạo một repository, một dự án mới hoặc đã có.

Cách xài: vào thư mục của dự án và sử dụng **git init** trong thư mục đó.

1. **git clone**

Tác dụng: Nhân bản một mã nguồn hay dự án từ máy chủ về máy tính cá nhân.

Cách xài: **git clone <path>.**  Trong đó **path** là đường địa chỉ dẫn đến kho mã nguồn được chứa ở máy chủ.

1. **git status**

Tác dụng: Để kiểm tra trạng thái của những tập tin bạn đã thay đổi trong thư mục làm việc. Ví dụ: Tất cả các thay đổi cuối cùng từ lần commit cuối cùng.

Cách xài: sử dụng lệnh **git status** trong thư mục làm việc.

1. **git add**

Tác dụng: Thêm thay đổi vào stage/index trong thư mục làm việc.

Cách xài: sử dụng lệnh **git add <file>** để lưu thay đổi những file bạn muốnhoặc **git add .** để thêm mọi file đã thay đổi, thêm, xóa, …

Hãy nhớ là mỗi tập tin trong thư mục làm việc của bạn có thể ở một trong hai trạng thái: tracked hoặc untracked. Có thể hiểu rằng tập tin tracked là các tập tin đã có mặt trong ảnh (snapshot) trước; chúng có thể là unmodified, modified, hoặc staged. Tập tin untracked là các tập tin còn lại: bất kỳ tập tin nào trong thư mục làm việc của bạn mà không có ở ảnh (lần commit) trước hoặc không ở trong khu vực tổ chức (staging area).

1. **git commit**

Tác dụng: Lưu lại một “ảnh chụp” của các sự thay đổi trong thư mục làm việc. Các tập tin, thư mục được thay đổi đã phải nằm trong khu vực tổ chức. Mỗi lần commit nó sẽ được lưu lại lịch sử chỉnh sửa của code kèm theo tên và địa chỉ email của người commit. Ngoài ra trong Git bạn cũng có thể khôi phục lại tập tin trong lịch sử commit của nó để chia cho một branch khác, vì vậy bạn sẽ dễ dàng khôi phục lại các thay đổi trước đó.

Cách dùng: **git commit -m ”thông báo các nội dung thay đổi”**

1. **git push/git pull**

Tác dụng: đẩy những thay đổi đã commit lên kho mã nguồn trên máy chủ lên hoặc lấy về các thay đổi từ kho mã nguồn trên máy chủ.

Cách dùng: **git pull <name\_remote> <name\_branch>** và **git push <name\_remote> <name\_branch>**. Trong đó “name\_remote” là tên của kho mã nguồn trên máy chủ, “name\_branch” là tên của kho mã nguồn ở máy tính cá nhân.

1. **git checkout**

Tác dụng: Chuyển sang branch khác hoặc tạo mới một branch.

Cách dùng: **git checkout <branch\_name>** để chuyển sang nhánh khác hoặc **git checkout -b <branch\_name>** để tạo nhánh mới.

1. **git stash**

Tác dụng: Lưu tạm thời các thay đổi đang làm mà bạn chưa muốn commit ngay lập tức. Đây là một dạng bộ nhớ tạm dùng để lưu các thay đổi đang làm mà chưa commit, sau đó phục hồi code về nguyên trạng ban đầu trước khi có những thay đổi đó. Áp dụng khi bạn đang code dở tay một chức năng gì đó, nhưng chưa xong mà cần phải lấy code từ mã nguồn trên máy chủ về.

Cách dùng: **git stash** trong thư mục làm việc của bạn để lưu tất cả thay đổi vào stash. **git stash list** để xem toàn bộ các stash đã lưu. **git stash apply <stash>** để lấy thay đổi từ một stash bất kì. **git stash clear** để xóa toàn bộ stash hoặc **git stash pop** để lấy thay đổi ra và xóa stash đó đi.

1. **git merge và git rebase**

Tác dụng: Hợp nhất hai nhánh lại với nhau.

Cách dùng: Chuyển tới branch bạn muốn hợp nhất rồi dùng **git merge <branch\_name>** hoặc dùng **git rebase <branch\_name>**

1. **git reset**

Tác dụng: Hoàn nguyên trạng thái code về một commit nào đó hoặc hủy một thay đổi của tập tin.

Cách dùng: **git reset HEAD <file\_name>** dùng để hủy thay đổi của file. **git reset <commit\_id>** dùng để hoàn nguyên trạng tháu của git về vị trí commit này nhưng không thay đổi code hoặc sử dụng **git reset –hard <commit\_id>**  để hoàn nguyên trạng thái của cả git lẫn code về vị trí commit này.

1. **git remote**

Tác dụng: Để kiểm tra kho mã nguồn trên máy chủ hoặc bạn có thể thêm các kho mã nguồn trên máy chủ cho kho mã nguồn trên máy cá nhân.

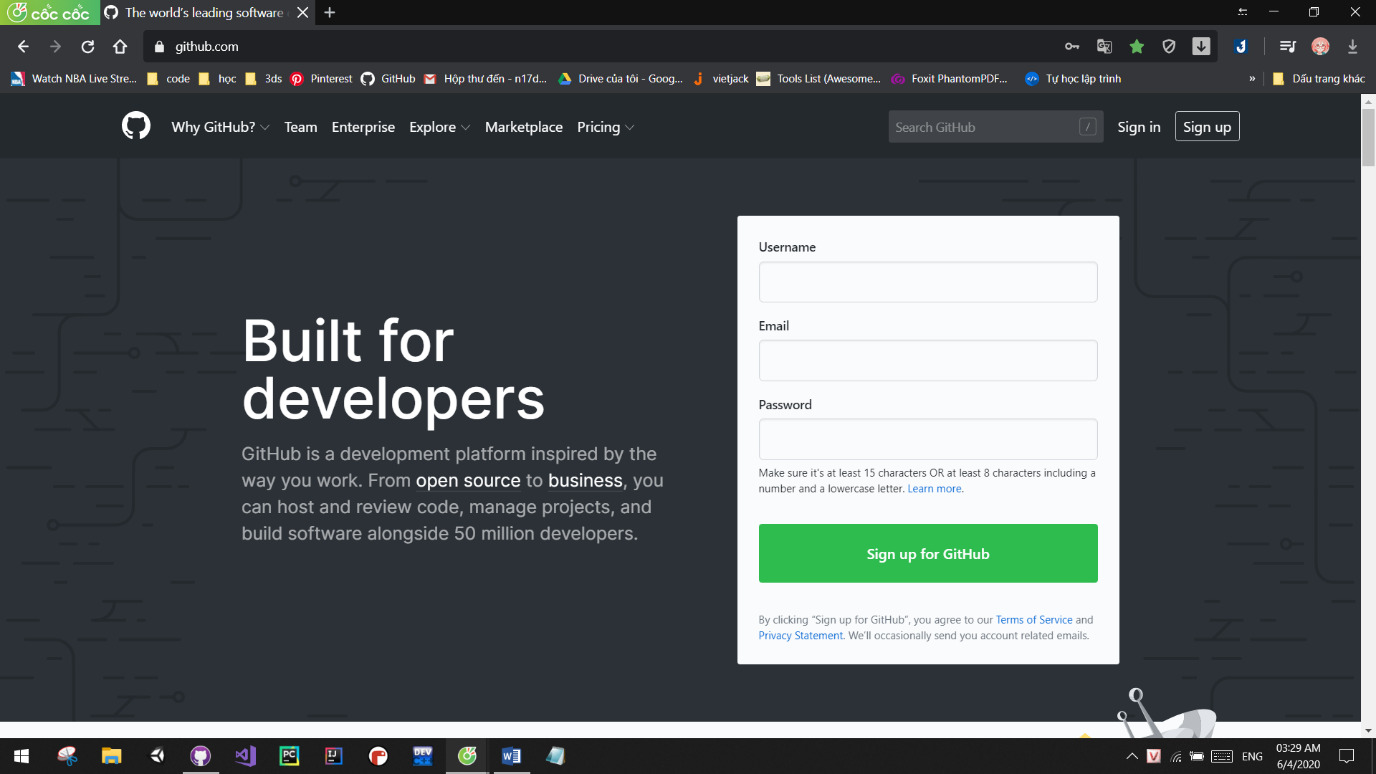
Cách dùng: **git remote** để kiểm tra và liệt kê. **git remote add <remote\_name> <remote\_address>** để thêm kho mã nguồn từ máy chủ về máy tính cá nhân, trong đó remote\_name và remote-address lần lượt là tên và địa chỉ của thêm kho mã nguồn từ máy chủ**.**

1. **git branch**

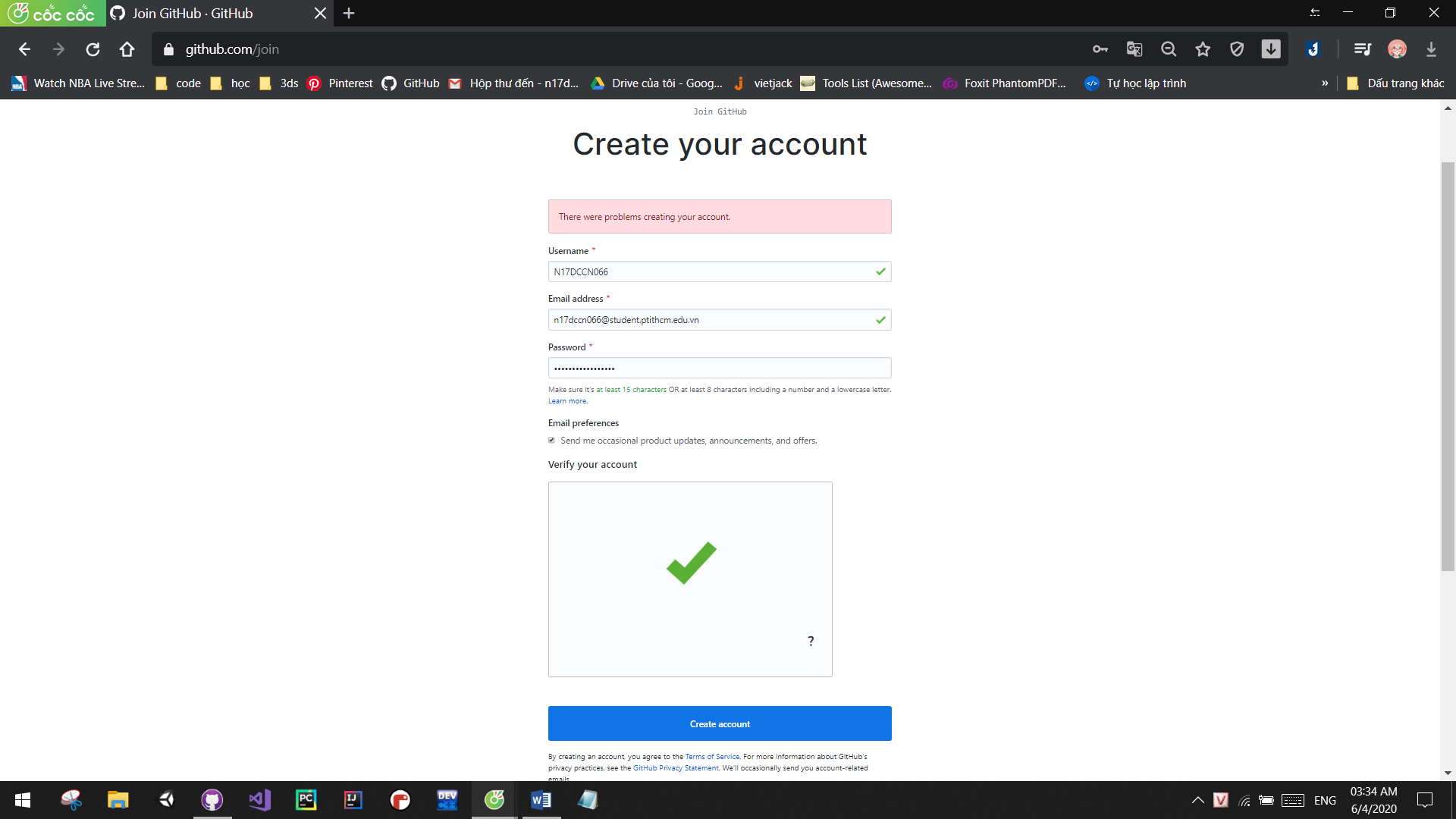
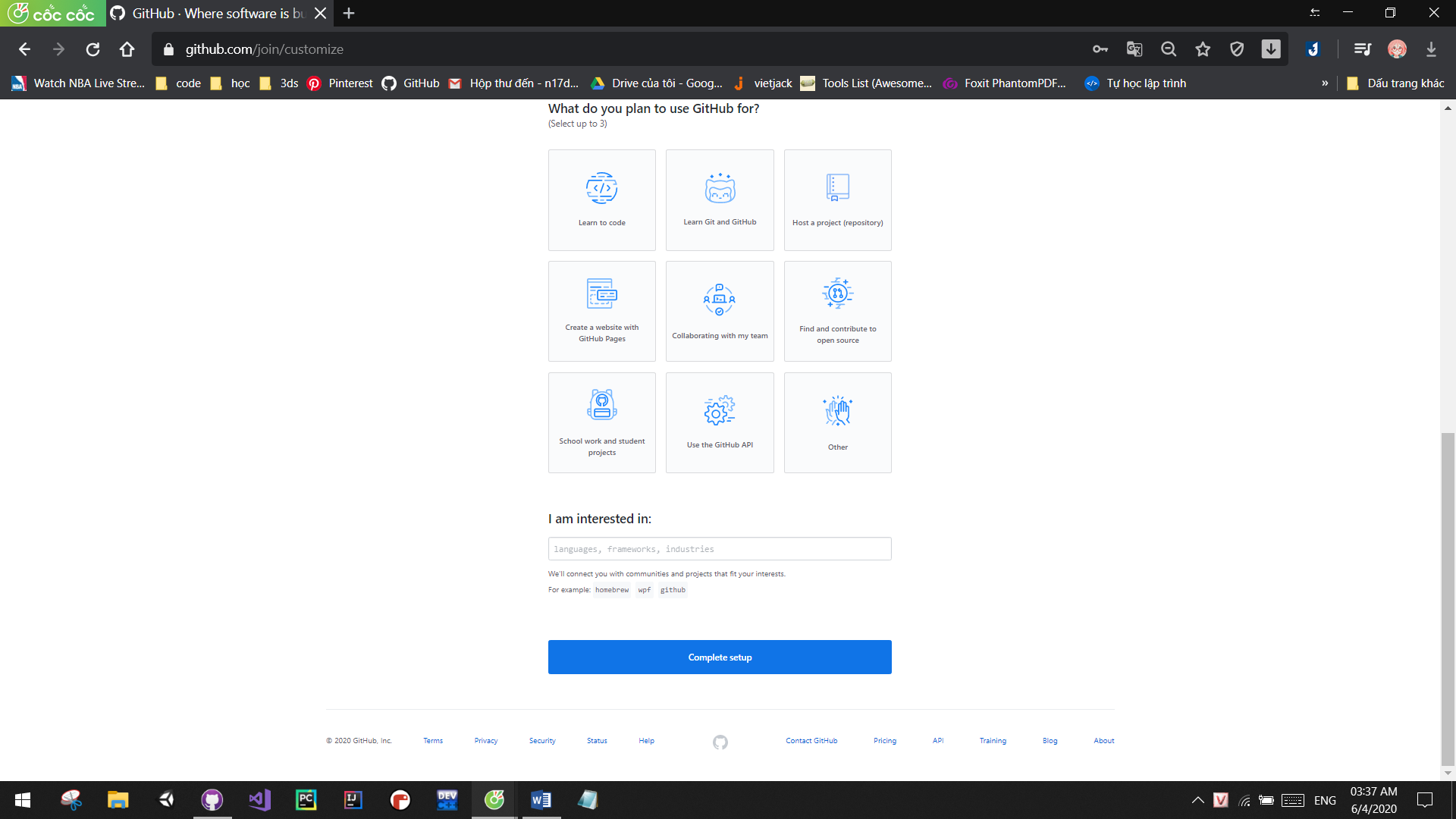
Tác dụng: các công việc có thể thực hiện với nhánh như: liệt kê, tạo mới, xóa, đổi tên,….

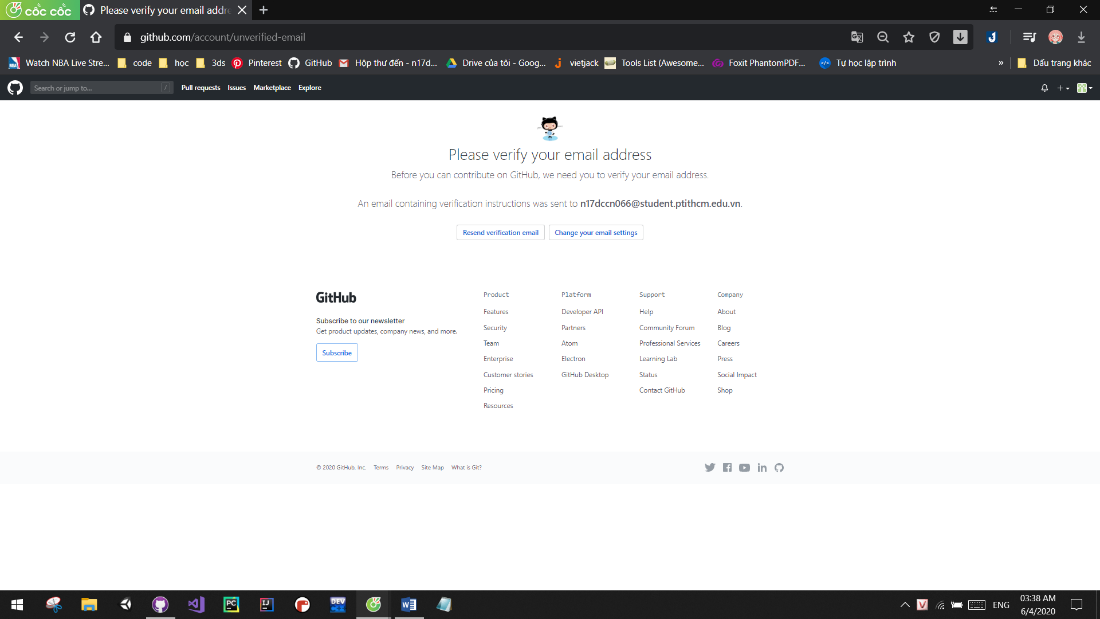
Cách dùng: **git branch** để liệt kê các nhánh, **git branch –v** để liệt kê các nhánh và lần commit cuối. **git branch <branch\_name>** để tạo nhánh mới. **git branch -d <branch\_name>** đê xóa nhánh. **git branch -m <oldBranch\_name> <newBranch\_name>** để đổi tên nhánh.

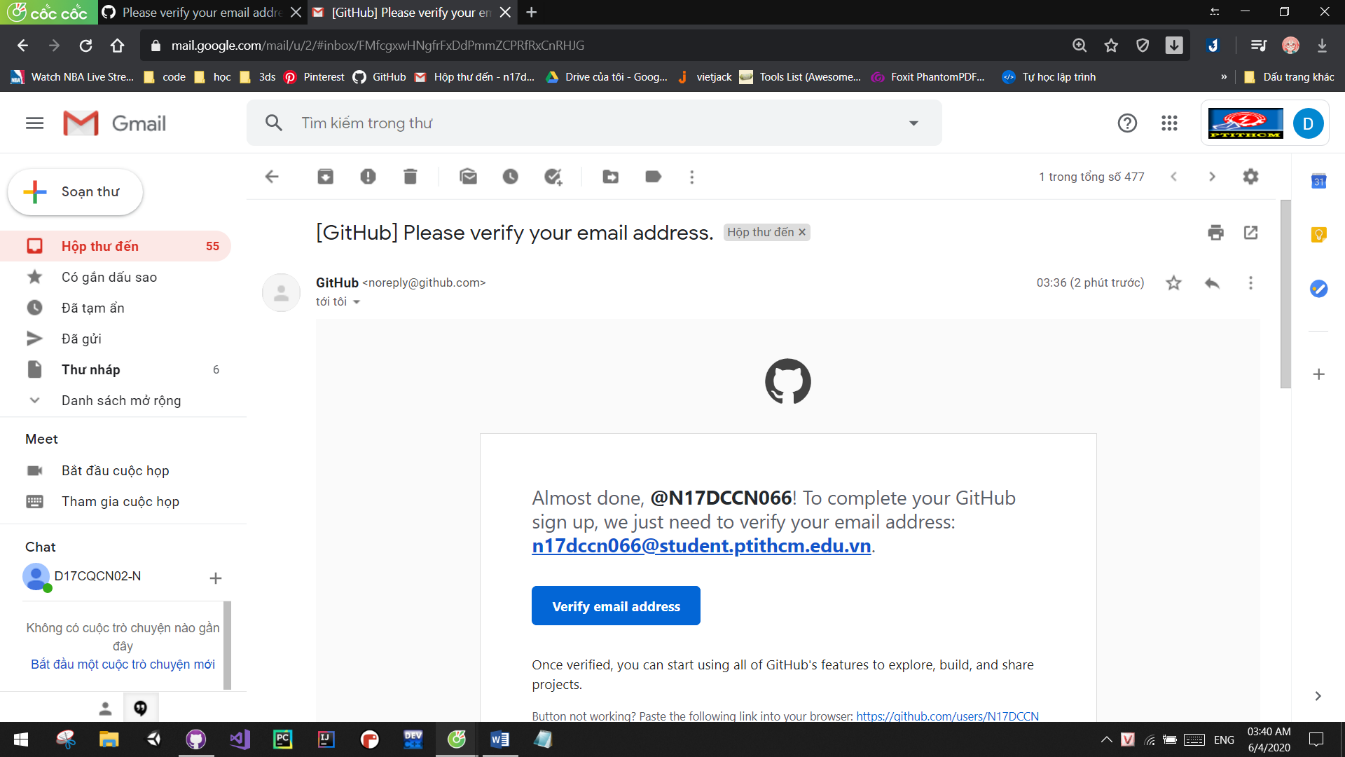
1. **Hướng dẫn sử dụng và thực hiện các chức năng cơ bản trên Git&GitHub**
   1. **Tạo tài khoản GitHub**

Bước 1: vào trang https://github.com/ nhấn vào **Sign up** ở góc phía bên phải bên trên hoặc nhấn vào **Sign up for GitHub** ở mục đăng nhập

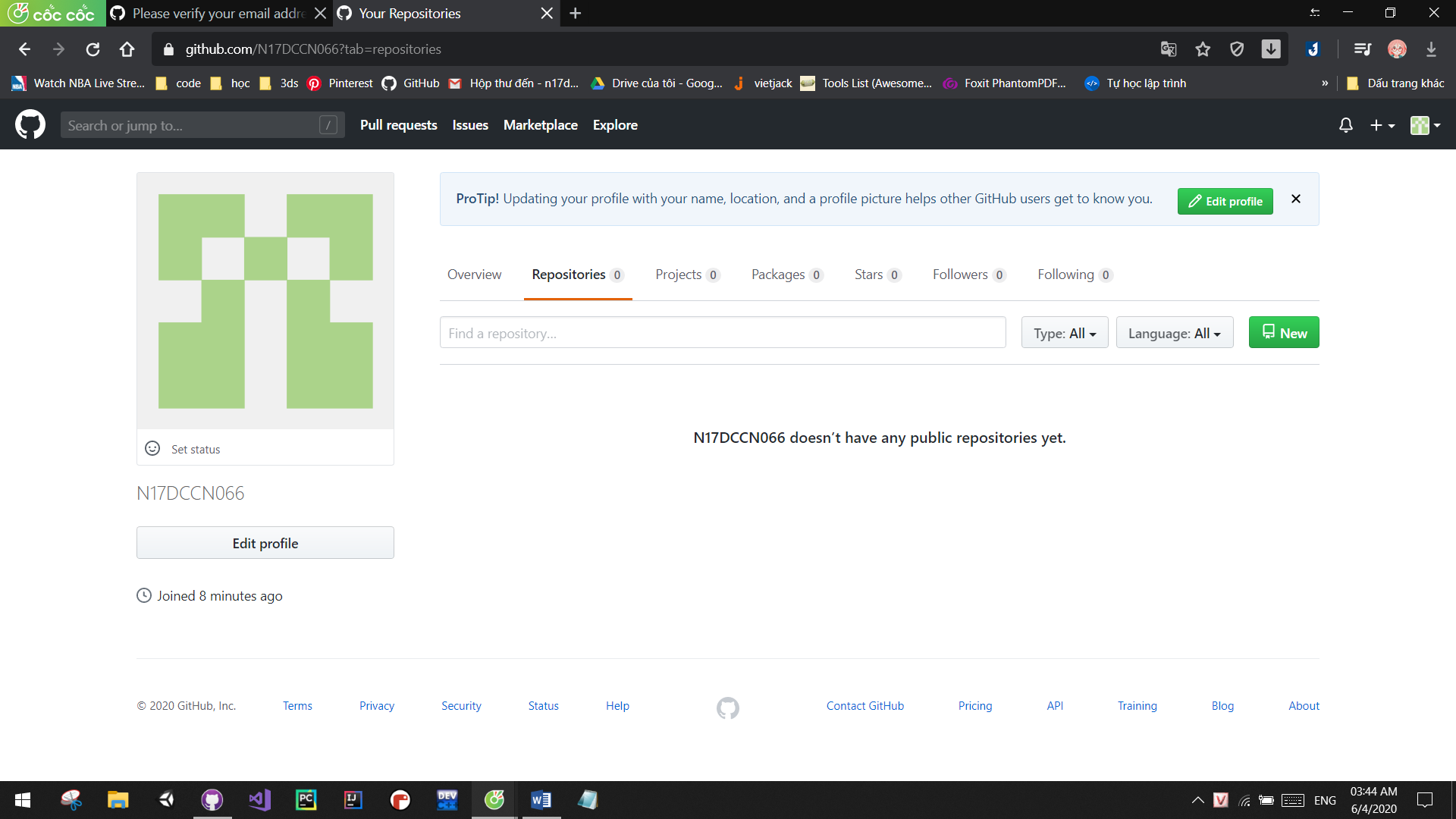
Bước 2: Nhập Usernamem email, password và bấm vào nút Create account để tiến hành tạo tài khoản mới

Bước 3: tiến hành các bước xác minh tài khoản. một email xác minh sẽ được gửi vào email đăng kí của bạn. bấm vào nút **Verify email address** để xác minh tài khoản



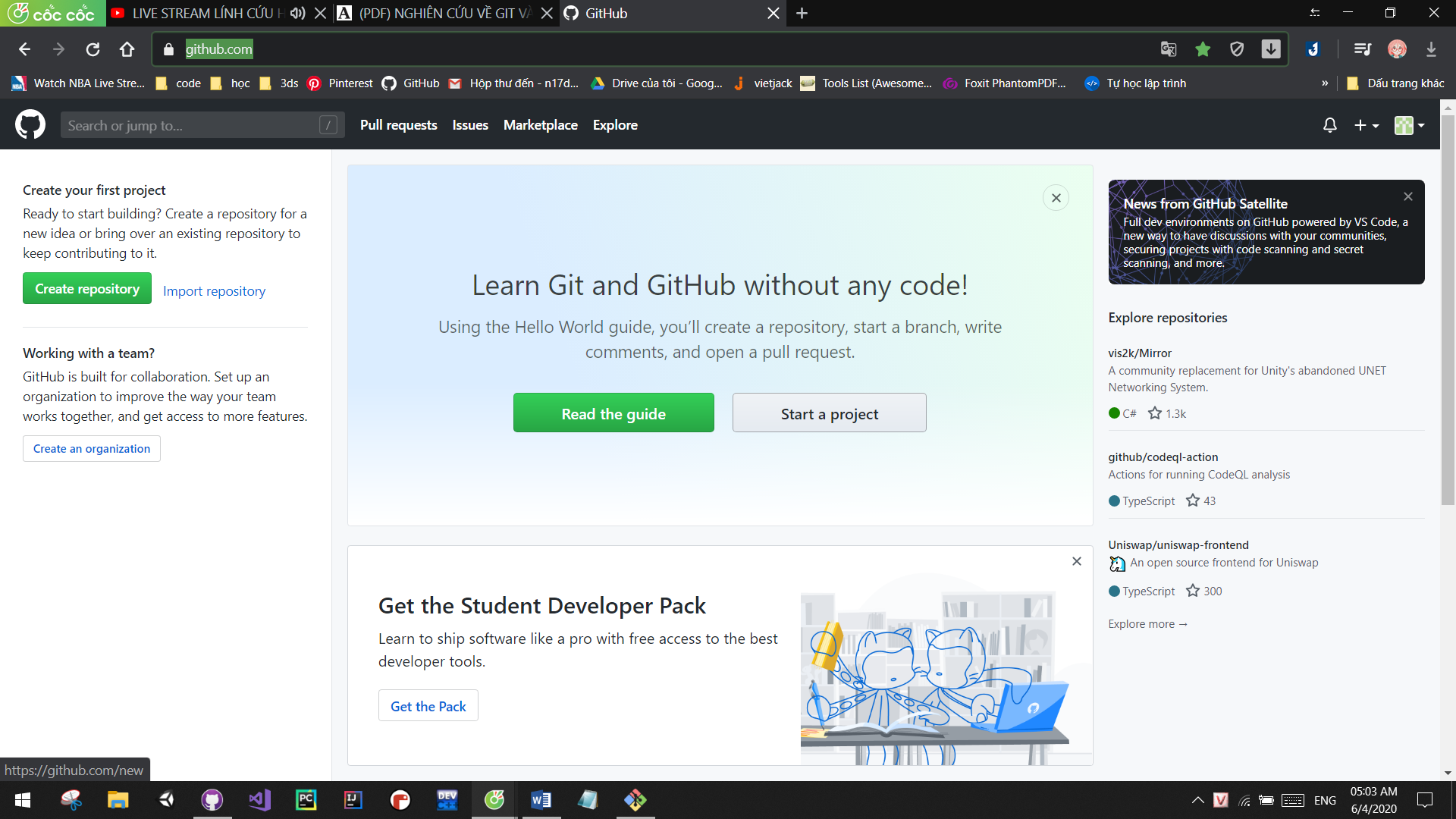


Hoàn thành việc tạo tài khoản:



* 1. **Cài đặt và sử dụng GitHub trên windows OS**
     1. **Sử dụng GitHub cơ bản**
        1. **Tạo Repository**

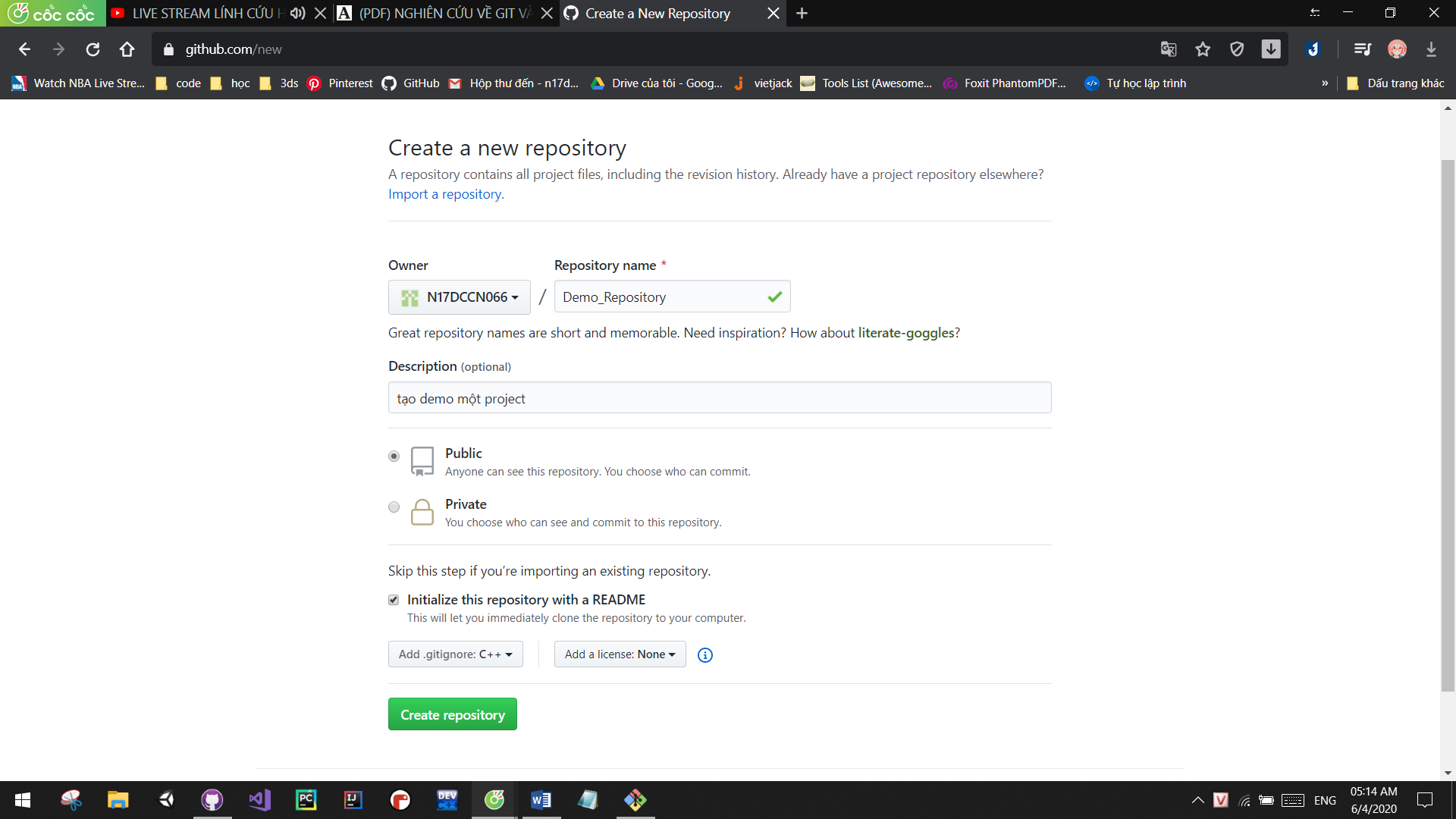
Bước 1: vào trang <https://github.com/>. Bấm vào Start a project



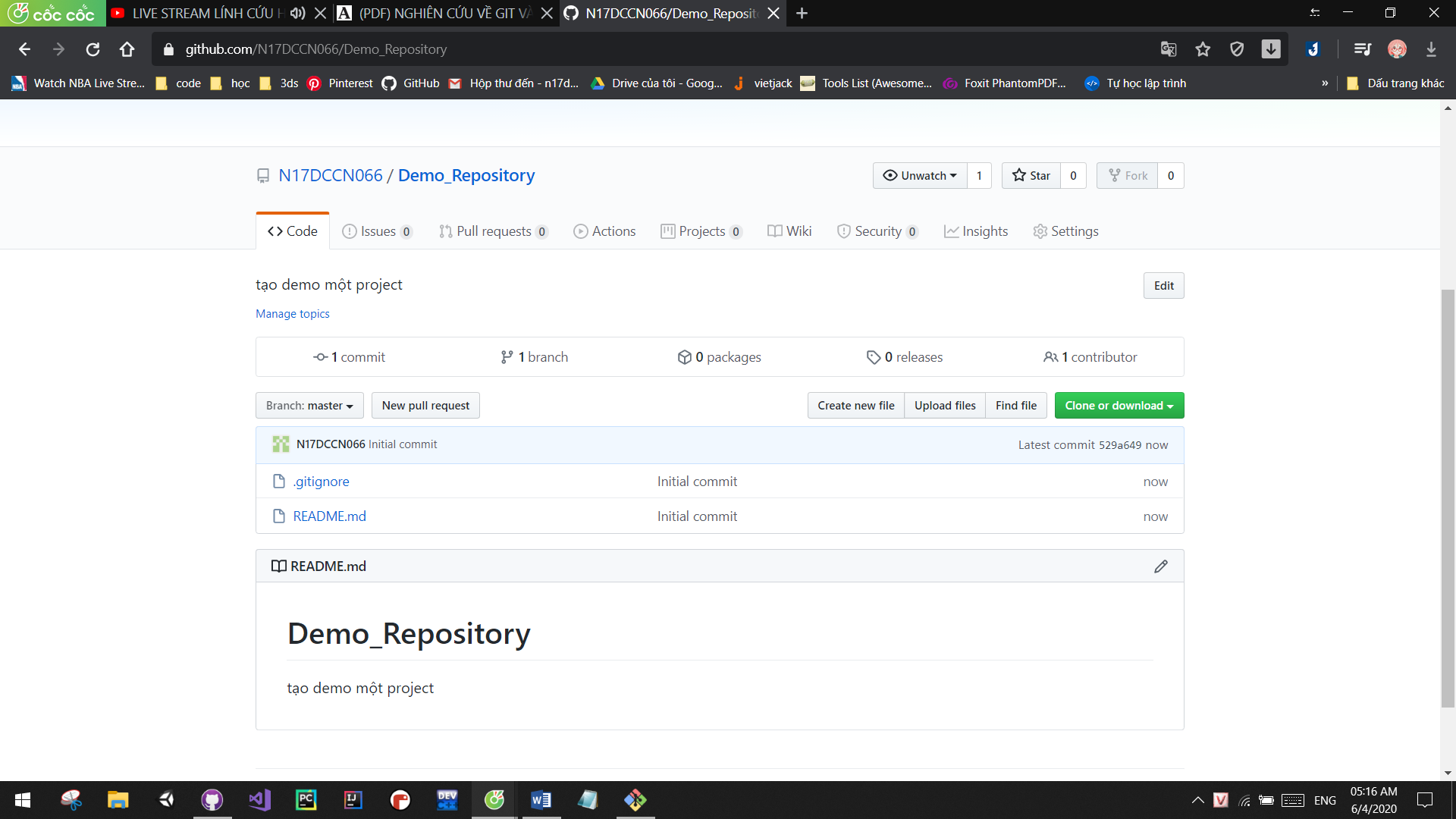
Bước 2: Nhập tên Repositoty và nhấn nút “Create Repository”. Ngoài ra cũng có thể thêm mô tả cho repository (Cái này chỉ là lựa chọn, không bắt buộc phải có).

Trong đó, bạn lưu ý 2 options sau:

* Theo mặc định thì repository để là public. Tức là ai cũng có thể xem được repository này của bạn. Nếu dự án của bạn chưa muốn công khai mà chỉ muốn quản lý nội bộ thì chọn Private.
* Bạn có thêm một README file để giới thiệu repository kèm với một file .gitignore. Github đã có sẵn template .gitignore cho bạn, cứ chọn một template phù hợp với mã nguồn dự án là được.



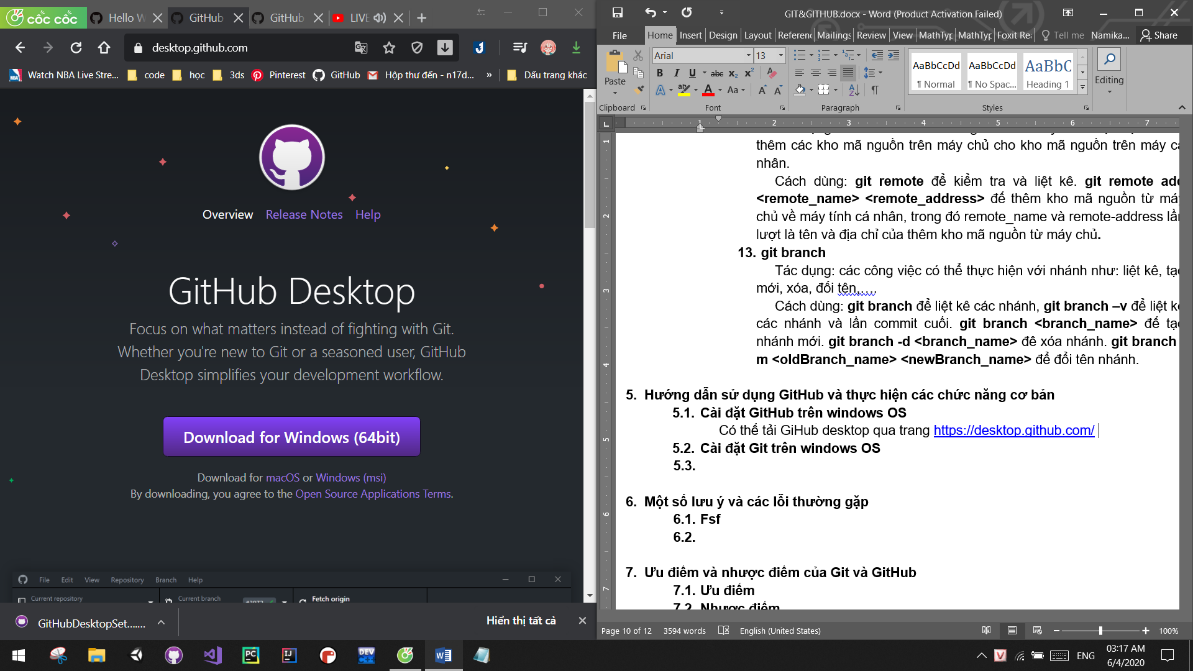
Khi tạo xong, repo sẽ như sau:



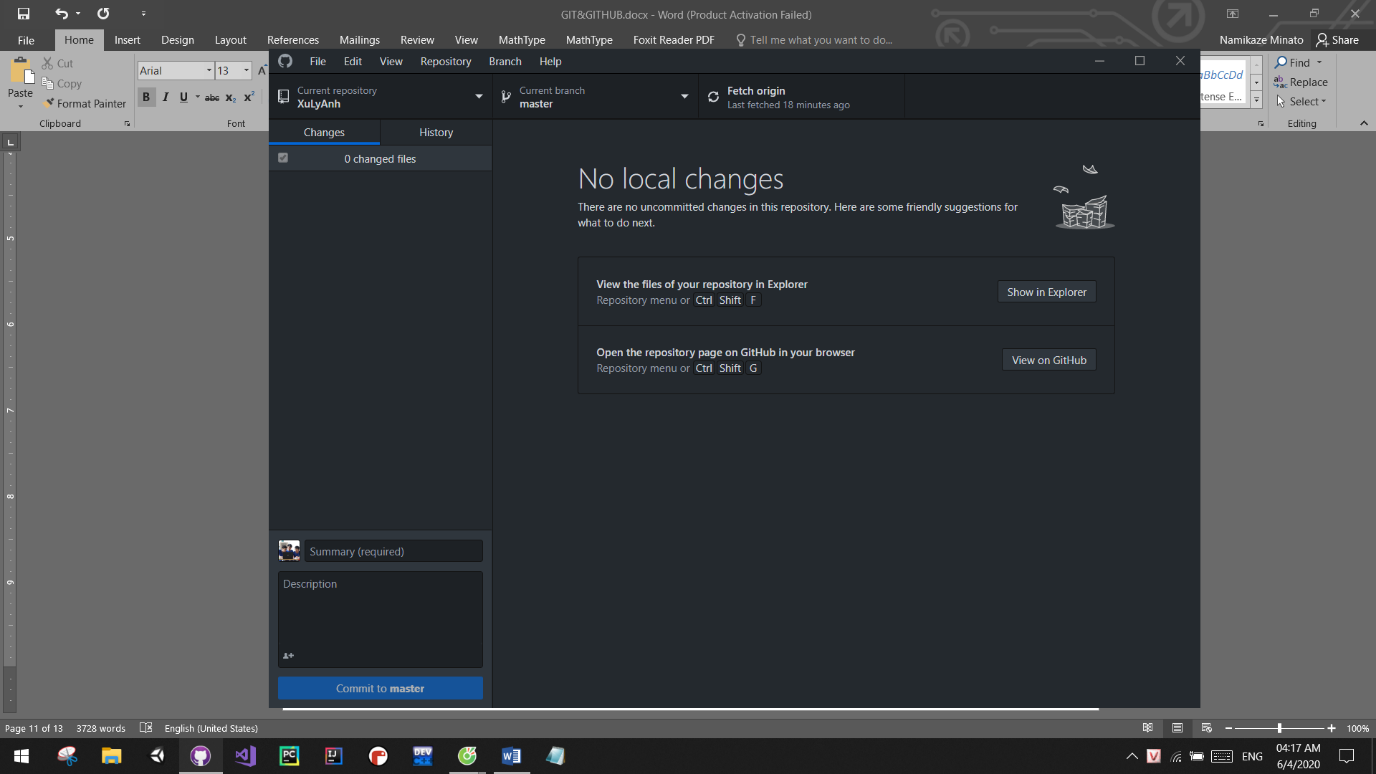
Khi đã có repository, có thể thực hiện các chức năng clone, pull, push… source code của mình lên đó.

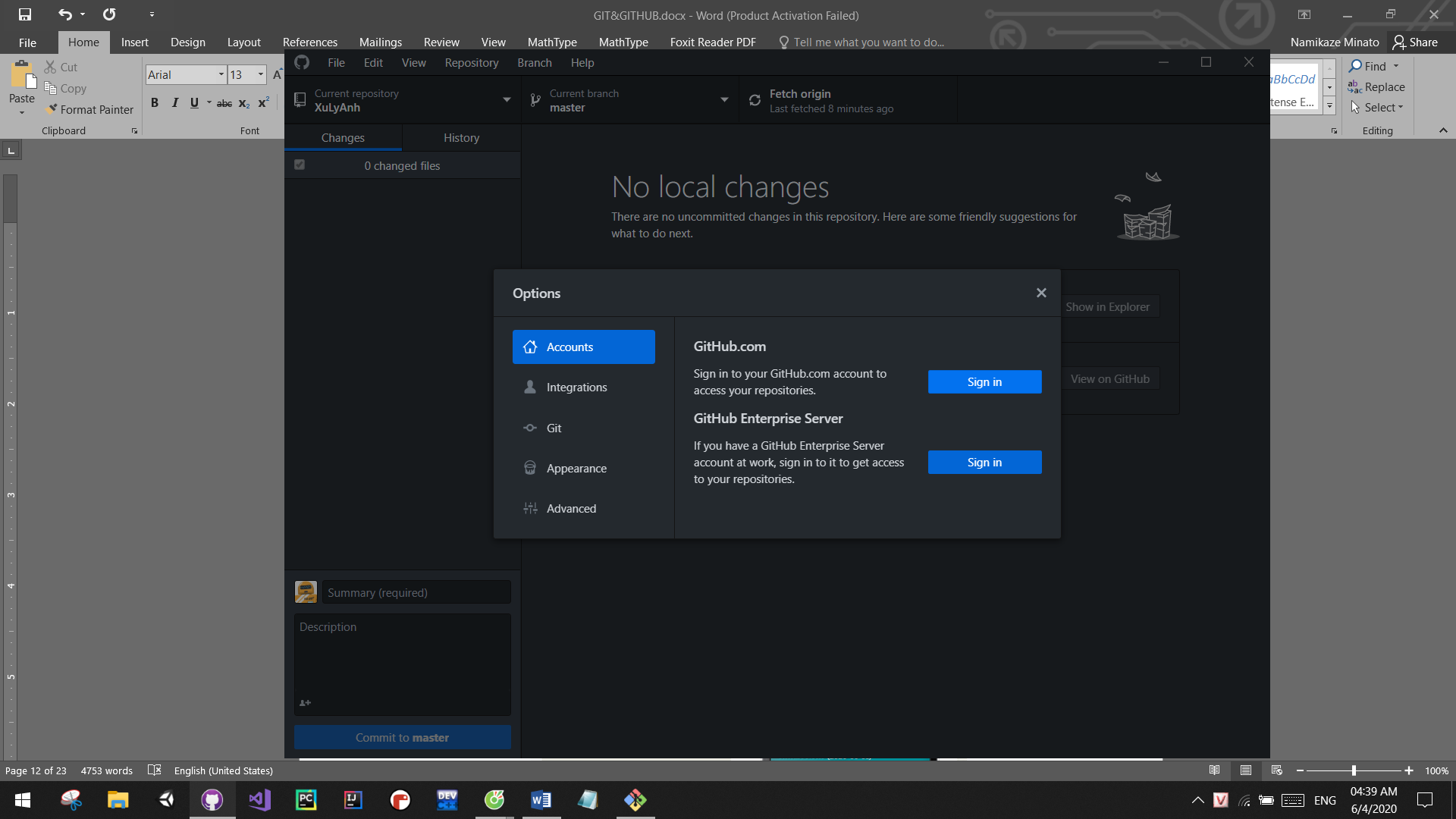
* + - 1. **Tạo branch trên Github**
      2. **Commit Command**
      3. **Pull Command**
      4. **Merge command**
      5. **Cloning dự án từ Github**
    1. **Sử dụng GitHub Desktop**

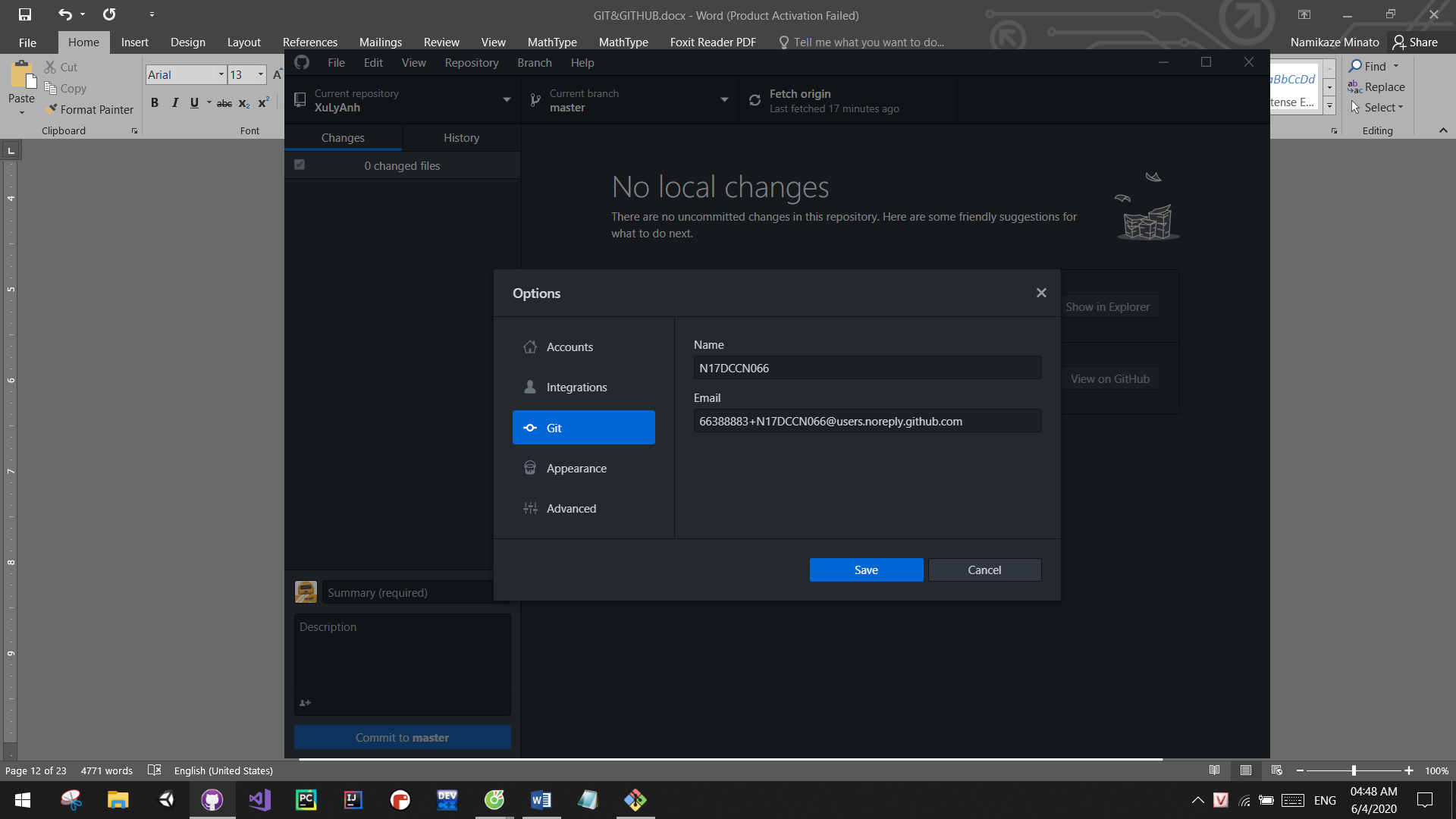
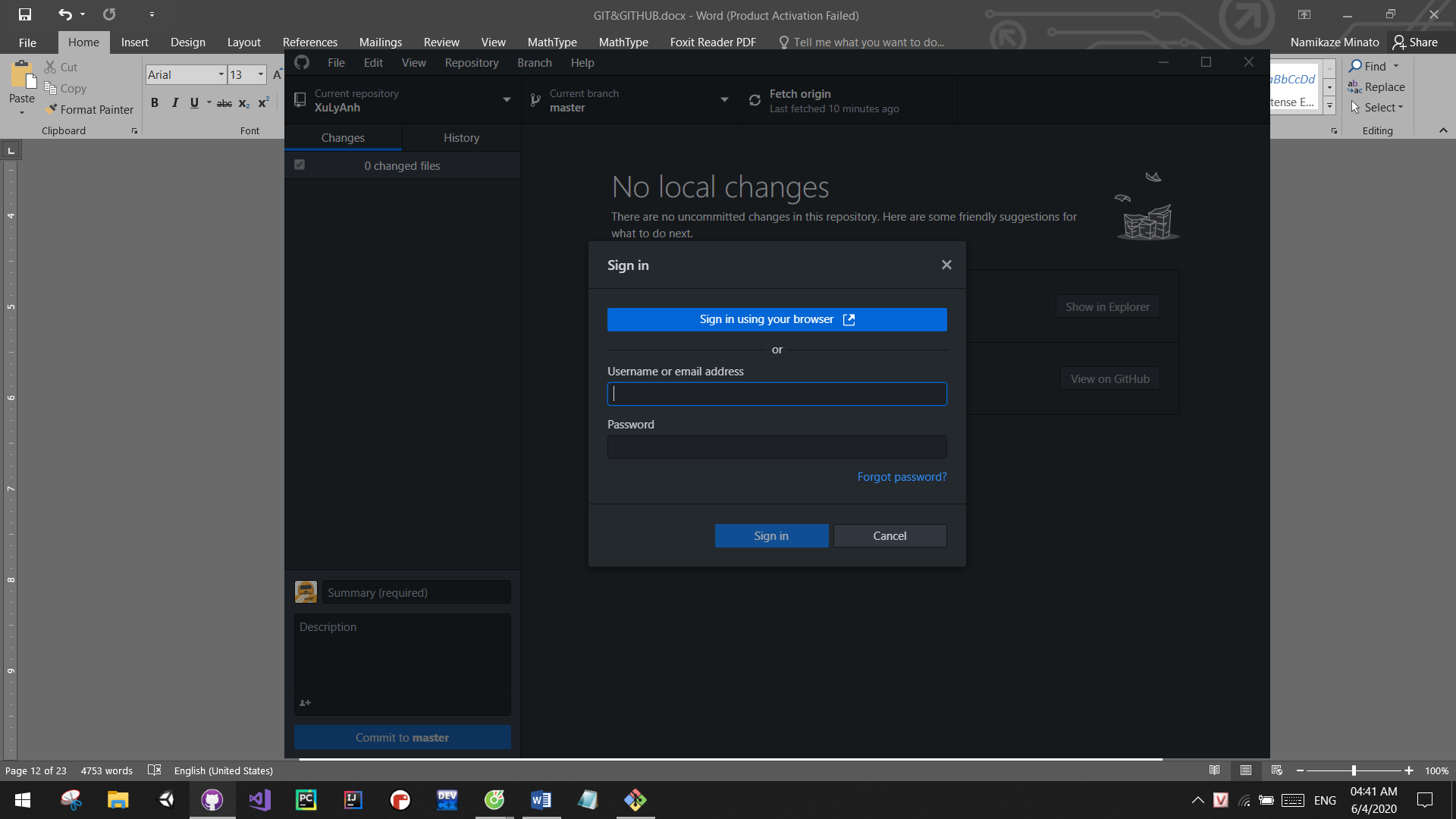
**GitHub Desktop** là một công cụ trực quan cho phép quản lý Local Repository (Kho chứa địa phương) trên máy tính cá nhân.

Bước 1: truy cập trang <https://desktop.github.com/> để tải tập tin cài đặt.

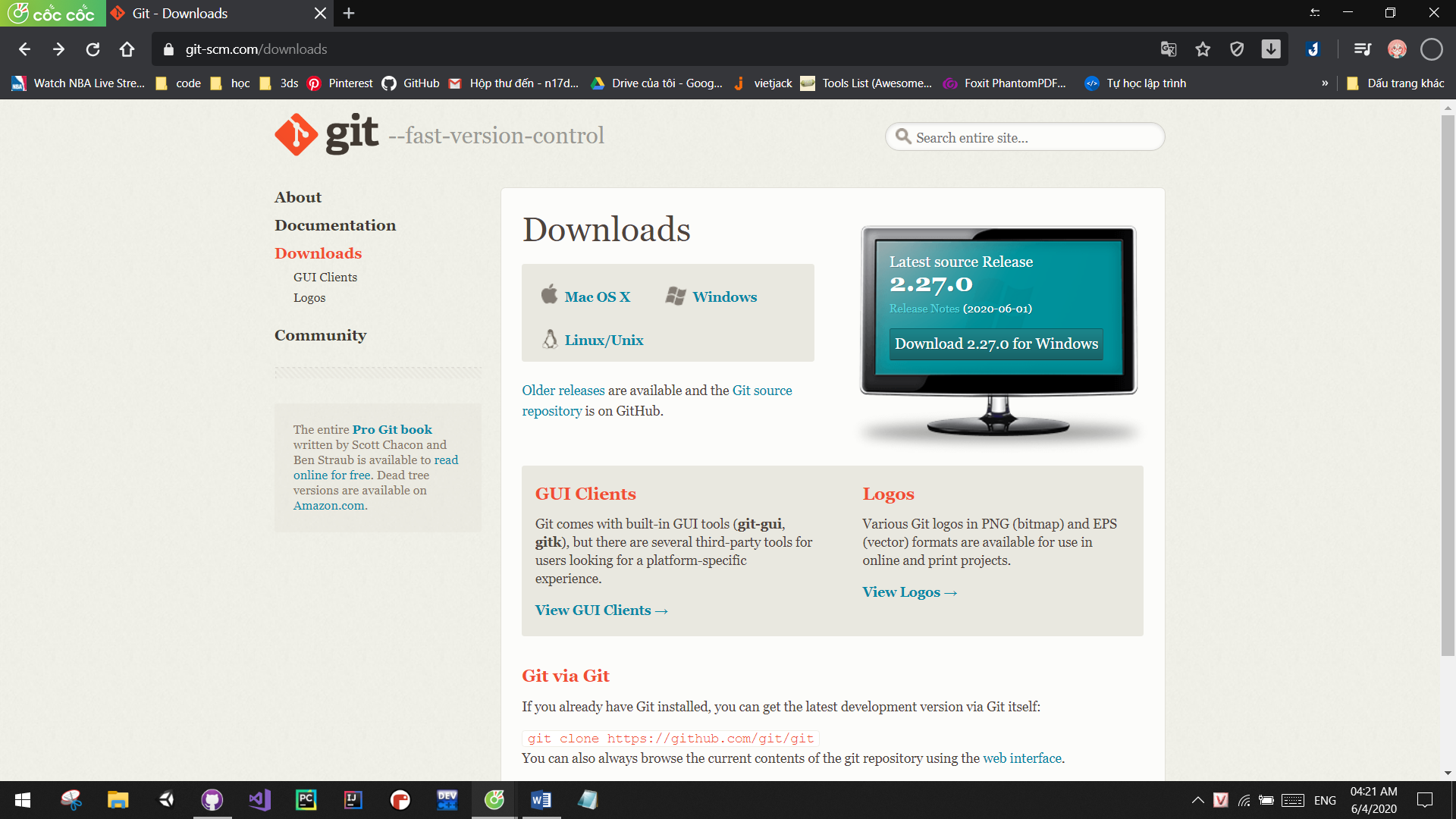
Bước 2: Sau khi tải, tiến hành cài đặt tập tin **GitHubDesktopSetup.exe** vào máy. Giao diện cơ bản như sau:



Bước 3: vào File -> Options, ở mục Account tiến hành đăng nhập

Bước 4: Sau khi đăng nhập, GitHub Desktop cũng sẽ tự tạo một Username và email cho git

* 1. **Cài đặt Git trên windows OS**

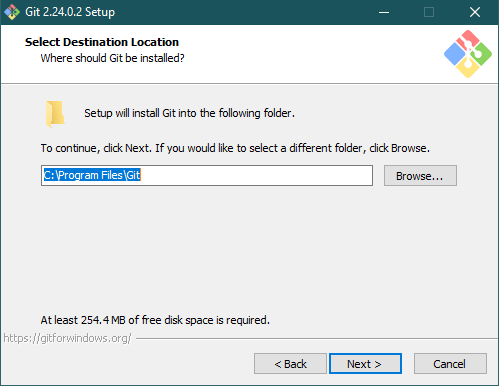
Bước 1: vào trang và chọn mục Downloads, chọn **Downloads 2.27.0 for Windows**

Bước 2: chạy tập tin tải về (ở đây là file **Git-2.27.0-64-bit.exe)**

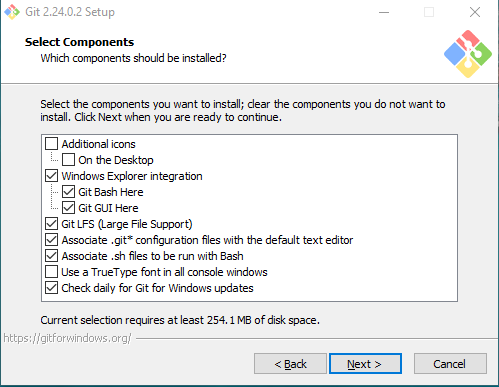
**Bước 3:**Sau khi bạn nhấn nút **Run**và cửa sổ đồng ý với giấy phép hiện ra



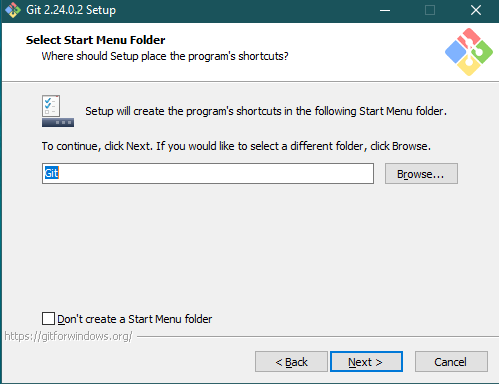
Bấm **Next**để lựa chọn nơi cài đặt Git trên máy tính



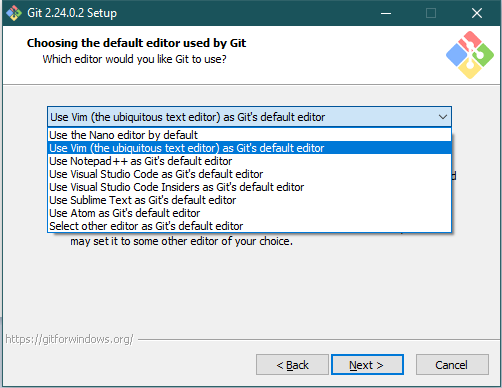
Bấm **Next**, bạn sẽ tìm thấy lời nhắc cửa sổ để chọn các thành phần sẽ được cài đặt. Sau khi bạn đã lựa chọn các thành phần mong muốn của mình, nhấp **Next**.

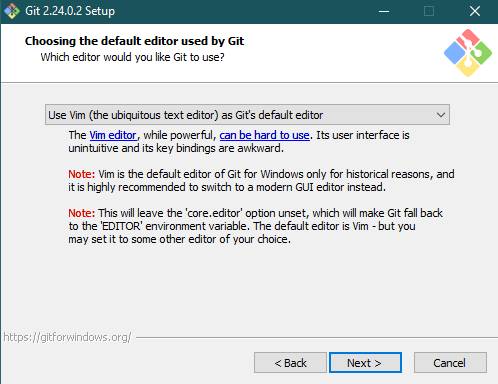


***Lưu ý:*** Để mặc định theo nhà sản xuất phần mềm nếu bạn chưa thực sự hiểu rõ về các lựa chọn của mình   
  
**Bước 4:**Cửa sổ nhắc tiếp theo sẽ cho phép bạn tạo đường dẫn trên Start Menu

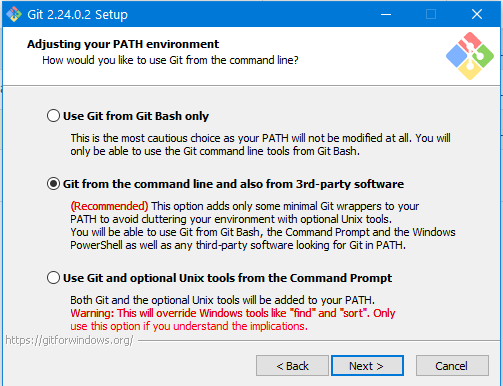


Bấm **Next**để được dẫn tới màn hình lựa chọn công cụ chỉnh sửa "editor", đây là công cụ sẽ giúp các bạn thực hiện việc biên tập các tập tin trong repository sau này. Git cung cấp cho chúng ta đến 8 tùy chọn editor như Nano, Viam (mặc định được lựa chọn), Notepad ++, Visual Studio Code, Sublime Text, Atom,.... nếu lựa chọn một trong số editor được liệt kê thì hãy chắc chắn máy của bạn đã cài đặt sẵn editor đó nhé. Ở đây, tôi để mặc định **Vim.**Bạn có thể xem thêm [Vim là gì? Hướng dẫn sử dụng Vim cơ bản!](http://aptechbmt.edu.vn/hoc-lap-trinh/thu-thuat-code/vim-la-gi-cac-lenh-thuong-hay-su-dung-nhat-trong-vim-9.html)

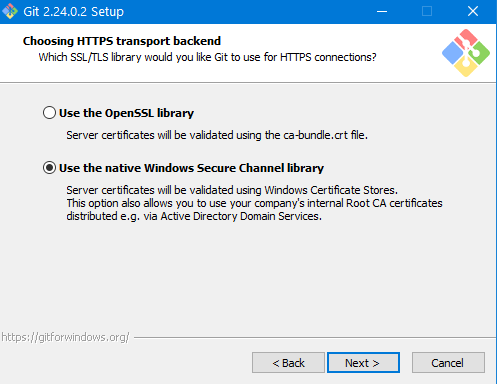




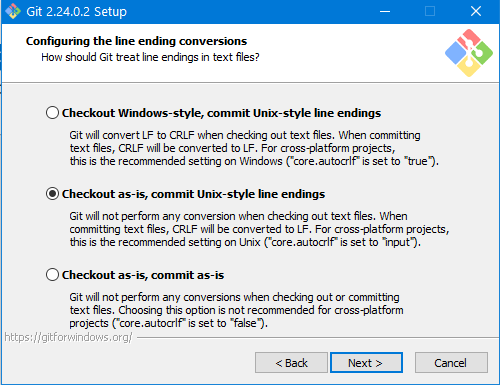
Bấm **Next**, để tới màn hình cài đặt môi trường cho Git. Đây là nơi bạn quyết định cách bạn muốn sử dụng Git. Bạn có thể chọn bất kỳ một trong ba tùy chọn theo nhu cầu của bạn. Nhưng đối với người mới bắt đầu học Git, tôi khuyên bạn nên sử dụng tùy chọn thứ 2: "Git from command line and also from 3rd-party software", đối với tùy chọn này thì ngoài Git Bash (CLI của Git) được cung cấp mặc định thì bạn có thể thao tác với Git thông qua CMD hoặc Powershell của Windows. Bấm **Next**để tiếp tục.



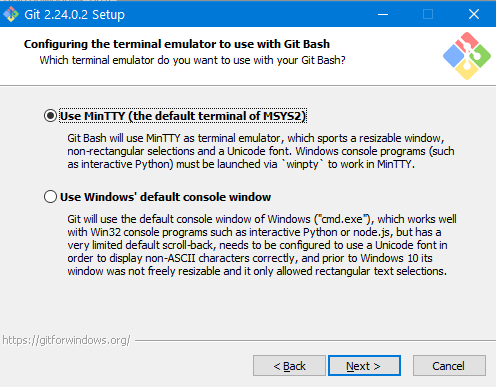
**Bước 5:**Bước tiếp theo là chọn các tính năng mã hóa kết nối cho Git của bạn. Nếu bạn là sinh viên hoặc nhà phát triển mới mà chưa rành về SSL/TLS là gì thì hãy chọn tùy chọn thứ 2 nhé. Bấm **Next**để tiếp tục



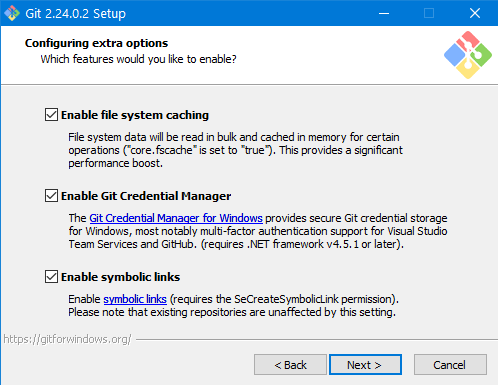
Màn hình tiếp theo nhắc nhở bạn tùy chọn định dạng cần chuyển đổi sau khi kết thúc thao tác dòng lệnh với Git. Chọn tùy chọn 2 để phù hợp với tiêu chuẩn kết thúc dòng Linux sẵn có của GIT.  Hiểu nôm na là với lựa chọn này, người dùng Windows sẽ có các tệp văn bản được chuyển đổi từ các kết thúc dòng kiểu Windows ( ) sang các kết thúc dòng kiểu Unix ( ) khi chúng được thêm vào kho lưu trữ. Điều này đảm bảo hơn về cơ chế lưu trữ trong Git khi bạn muốn xem lại Log hoặc Status. Bấm **Next**để tiếp tục.



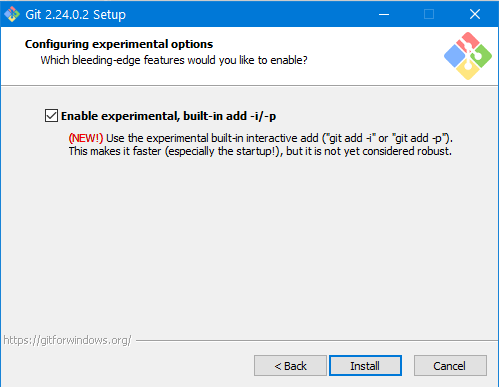
**Bước 6:**Màn hình cấu hình công cụ giả lập dòng lệnh để sử dụng Git Bash. Bạn có thể chọn 1 trong 2, nhưng cá nhân tôi sử dụng MinTTY. Bởi lẽ, Mintty là một phần phần mềm mô phỏng giao diện dòng lệnh mặc định của Git tương đối tốt, nó cung cấp các tính năng bổ sung như kéo và thả, toàn màn hình, sao chép và dán và hỗ trợ theme giúp chúng ta thao tác thuận lợi hơn với dòng lệnh. Bấm **Next**để tiếp tục



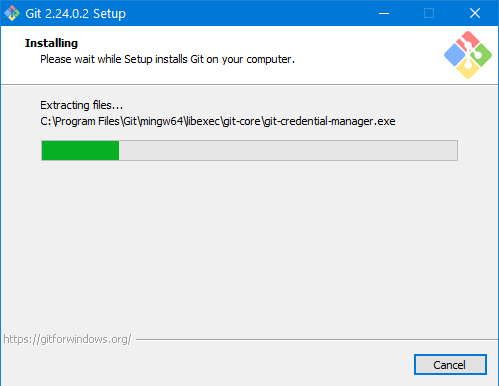
Bạn nhận được ba tùy chọn và bạn có thể chọn bất kỳ tùy chọn nào trong số chúng, tất cả chúng hoặc không có tùy chọn nào theo nhu cầu của bạn. Hãy để tôi nói cho bạn biết những tính năng này là gì:



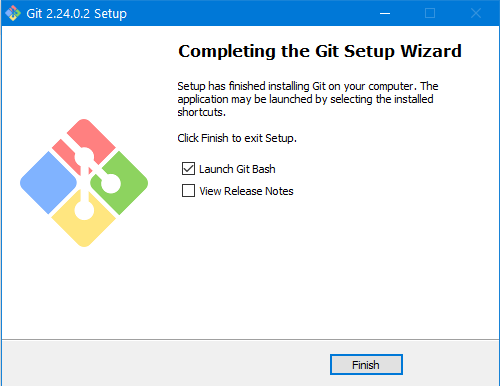
Đầu tiên là tùy chọn**Enable file system caching:** Bộ nhớ đệm sẽ được bật thông qua Trình quản lý bộ đệm, nó sẽ hoạt động liên tục trong khi Windows đang chạy. Dữ liệu tệp trong bộ đệm tệp hệ thống được ghi vào đĩa theo các khoảng thời gian được xác định bởi hệ điều hành và bộ nhớ được sử dụng trước đó bởi dữ liệu tệp đó sẽ được giải phóng.  
  
Tùy chọn thứ hai là **Enable Git Credential Manager:** Các chứng chỉ "Credential" quản lý Git cho Windows (GCM) là một helper chứng nhận trung gian cho Git. Nó lưu trữ an toàn thông tin đăng nhập của bạn trong Windows CM để bạn chỉ cần nhập chúng một lần cho mỗi kho lưu trữ từ xa mà bạn truy cập. Tất cả các lệnh Git trong tương lai sẽ sử dụng lại các thông tin hiện có. Bật tùy chọn này nếu bạn chắc chắn máy tính/tài khoản máy tính của bạn được bảo mật an toàn nhé :-)  
  
Tùy chọn thứ ba là **Enable symbolic links:** Tùy chọn này giúp bạn có thể tạo các liên kết tượng trưng bằng các phím tắt nâng cao. Bạn có thể tạo các liên kết tượng trưng cho từng tệp hoặc thư mục riêng lẻ và các liên kết này sẽ xuất hiện giống như chúng được lưu trữ trong thư mục có liên kết tượng trưng.  
  
Bấm **Next**để tiếp tục  
  
**Bước 7:**Nếu bạn muốn thử nghiệm các tính năng mới của Github, bạn có thể chọn thử nghiệm, dựng sẵn. Bạn có thể xem sự khác biệt mã giữa các phiên bản khác nhau cho (các) tệp.



Sau khi đã hoàn thành tất cả các thiết lập bên trên. Chọn **Install**để cài đặt



Chọn **Launch Git Bash** và nhấp vào **Finish**để kết thúc quá trình cài đặt.



Điều này sẽ khởi chạy Git Bash trên màn hình của bạn, hãy thử gõ "git --version" để kiểm tra phiên bản Git vừa cài đặt thành công vào máy tính. Nếu xuất hiện như bên dưới, chúc mừng bạn đã cài đặt thành công. 



1. **Một số lưu ý và các lỗi thường gặp**
   1. **Fsf**
2. **Ưu điểm và nhược điểm của Git và GitHub**
   1. **Ưu điểm**
   2. **Nhược điểm**
3. **Kết luận**
4. **Kế hoạc thực hiện đề tài**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **KẾ HOẠCH THỰC HIỆN ĐỀ TÀI NCKH** | | | | | | |
| Đề tài: | Tìm hiểu về Git và GitHub | | | | | |
| Người thực hiện: | | Nguyễn Đức Khải | |  | N17DCCN066 | | |
| Thời gian thực hiện: 5 tuần | | từ ngày 14/10/2019 | |  | đến ngày 03/11/2019 | | |
|  |  |  |  |  |  |  | |
| STT | Nội dung công việc | W1 | W2 | W3 | W4 | W5 | |
| 1 | Đề xuất ý tưởng |  |  |  |  |  | |
| 2 | Hình thành mục tiêu |  |  |  |  |  | |
| 3 | Đặt vấn đề |  |  |  |  |  | |
| 4 | Tìm và nghiên cứu tài liệu |  |  |  |  |  | |
| 5 | Giải quyết vấn đề |  |  |  |  |  | |
| 6 | Hoàn thành đề tài |  |  |  |  |  | |

1. **Tài liệu tham khảo**

* Holly Kathleen Hall, 27 Cath. U. J. L. & Tech 51, 2018. **“DEEPFAKE VIDEOS: WHEN SEEING ISN’T BELIEVING”.**
* Marc Jonathan Blitz, 71 Okla. L. Rev. 59, 2018. **“Lies, Line Drawing, and (Deep) Fake News”.**
* Nicholas Gardiner, Western Australian Academy of Performing Arts Edith Cowan University, 2019. **“Facial re-enactment, speech synthesis and the rise of the Deepfake”.**
* [1] Deepfake – Wikipedia tiếng Việt, <https://vi.wikipedia.org/wiki/Deepfake>, truy cập ngày 26/10/2019.
* [2] Công nghệ ‘deepfake’ là gì và nó có thể gây nguy hiểm thế nào?, <http://baovanhoa.vn/nhip-song-so/artmid/2065/articleid/22945/cong-nghe-%E2%80%98deepfake-la-gi-va-no-co-the-gay-nguy-hiem-the-nao>, truy cập ngày 26/10/2019.
* [3] Deepfake bóng ma mới của thế giới Internet, <https://news.zing.vn/deepfake-bong-ma-moi-cua-the-gioi-internet-post906004.html>, truy cập ngày 28/10/2019.
* [4] Deepfake – “Mặt tối” khó kiểm soát của thế giới công nghệ, <https://bnews.vn/deepfake-mat-toi-kho-kiem-soat-cua-the-gioi-cong-nghe/128328.html>, truy cập ngày 28/10/2019.
* [5] Deepfake – công nghệ khởi đầu cho những video khiêu dâm, dung tục giả mạo, <https://vnreview.vn/goc-nhin-vnreview/-/view_content/content/2890542/deepfake-cong-nghe-khoi-dau-cho-nhung-video-khieu-dam-dung-tuc-gia-mao>, truy cập ngày 30/10/2019.
* [6] Deepfake: Mặt tốt, mặt xấu và những khía cạnh đen tối, <https://viettimes.vn/deepfake-mat-tot-mat-xau-va-nhung-khia-canh-den-toi-169975.html>, truy cập ngày 30/10/2019.
* [7] Deepfake có thể là vấn đề lớn trong cuộc bầu cử Mỹ 2020, <http://special.vietnamplus.vn/deepfake-bau-cu-my>, truy cập ngày 30/10/2019.
* [8] Bóng ma Deepfake trỗi dậy: phát hiện 14.678 video ghép mặt người nổi tiếng, 96% số này có nội dung khiêu dâm, <http://genk.vn/bong-ma-deepfake-troi-day-phat-hien-14678-video-ghep-mat-nguoi-noi-tieng-96-so-nay-co-noi-dung-khieu-dam-20191010101515249.chn>, truy cập ngày 01/11/2019.
* [9] Deepfake, thứ vũ khí nguy hiểm bậc nhất sinh ra trong thời đại số, <http://cafebiz.vn/deepfake-thu-vu-khi-nguy-hiem-bac-nhat-sinh-ra-trong-thoi-dai-so-20190629181132437.chn>, truy cập ngày 02/11/2019.