Bản báo cáo

Nhóm 11 :

Trần Đoàn Thanh Sang (nhóm trưởng)

Đoàn Chí Tôn ( thành viên )

Đào Trọng Nghĩa (thành viên)

Bao gồm các thành phần sau:

1. Định nghĩa/ giới thiệu  
   GIT:  
   Git (/ɡɪt/, đọc là "Ghít") là phần mềm quản lý mã nguồn phân tán được phát triển bởi Linus Torvalds vào năm 2005, ban đầu dành cho việc phát triển nhân Linux. Hiện nay, Git trở thành một trong các phần mềm quản lý mã nguồn phổ biến nhất. Git là phần mềm mã nguồn mở được phân phối theo giấy phép công cộng GPL2.  
   Phần mềm hỗ trợ  
   Có nhiều website và phần mềm cho phép lưu trữ (miễn phí và có phí) Git:  
   BerliOS, GitHub, GitLab, Gitorious, Sourceforge, GNU Savannah, GitEnterprise  
   GITHUB:  
   GitHub là một dịch vụ cung cấp kho lưu trữ mã nguồn Git dựa trên nền web cho các dự án phát triển phần mềm. GitHub cung cấp cả phiên bản trả tiền lẫn miễn phí cho các tài khoản. Các dự án mã nguồn mở sẽ được cung cấp kho lưu trữ miễn phí. Tính đến tháng 4 năm 2016, GitHub có hơn 14 triệu người sử dụng với hơn 35 triệu kho mã nguồn, làm cho nó trở thành máy chủ chứa mã nguồn lớn trên thế giới.  
   Github 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 một liên kết đến tài khoản Github để đánh giá ứng viên.  
   Vào ngày 4 tháng 6 năm 2018, Microsoft đã thông báo việc đạt được thỏa thuận mua lại GitHub với giá 7,5 tỷ Đô la Mỹ. Ngày chính thức chuyển nhượng quyền sở hữu không được công bố.  
   Dịch Vụ:  
   Sự phát triển của nền tảng GitHub bắt đầu vào ngày 19 tháng 10 năm 2007. Trang web được đưa ra vào tháng 4 năm 2008 do Tom Preston-Werner, Chris Wanstrath, và PJ Hyett thực hiện sau khi nó đã được hoàn thành một vài tháng trước đó, xem như giai đoạn beta.  
   Dự án trên Github có thể được truy cập và thao tác sử dụng một giao diện dòng lệnh và làm việc với tất cả các lệnh Git tiêu chuẩn. Github cũng cho phép người dùng đăng ký và không đăng ký để duyệt kho công cộng trên trang web. Github cũng tạo ra nhiều client và plugin cho máy tính để bàn.  
   Trang web cung cấp các chức năng mạng xã hội như feed, theo dõi, wiki (sử dụng phần mềm Gollum Wiki) và đồ thị mạng xã hội để hiển thị cách các nhà phát triển làm việc trên kho lưu trữ.  
   Một người sử dụng phải tạo ra một tài khoản cá nhân để đóng góp nội dung lên Github, nhưng các kho mã nguồn công cộng có thể được duyệt và tải về với bất cứ ai. Với một người dùng đã đăng ký tài khoản, họ có thể thảo luận, quản lý, tạo ra các kho, đóng góp cho kho của người dùng khác, và xem xét thay đổi mã.  
   GitHub cũng có một dịch vụ khác: một trang web kiểu pastebin gọi là Gist, dùng để lưu trữ các đoạn mã; trong khi Github sẽ được cho lưu trữ các dự án lớn hơn. Một dịch vụ lưu trữ khác được gọi là Speaker Deck.  
   Các phần mềm chạy GitHub được viết bằng Ruby on Rails và Erlang bởi GitHub, Inc, phát triển Chris Wanstrath,[10] PJ Hyett, và Tom Preston-Werner.  
   Hệ thống kiểm soát phiên bản (VCS) - ghi lại thay đổi vào một tập tin hoặc thiết lập các tập tin theo thời gian để có thể nhớ lại các phiên bản cụ thể sau.  
   Phạm vi  
   GitHub chủ yếu được sử dụng để lưu trữ mã nguồn phần mềm, nhưng cũng thường được sử dụng với nhiều loại tập tin như Final Cut hoặc các tài liệu Word.  
   Ngoài mã nguồn, Github hỗ trợ các định dạng và các tính năng sau đây:  
   3D làm cho các tập tin mà có thể được xem trước bằng cách sử dụng tích hợp trình xem file STL mới hiển thị các tập tin trên một khung 3D.Người xem được hỗ trợ bởi WebGL và Three.js.  
   Nguồn gốc định dạng PSD của Photoshop có thể được xem trước và so với các phiên bản trước của cùng một tập tin.  
   Lồng nhiệm vụ danh sách  
   Tài liệu và Wiki  
   Các trang web nhỏ có thể được lưu trữ từ kho công cộng trên Github. Định dạng URL. Và có thể được tạo ra bằng cách bắt đầu một kho lưu trữ được định dạng như projectname.io  
   Code Snippets (bằng cách sử dụng tên miền phụ Gist)  
   Theo dõi vấn đề và tính năng yêu cầu  
   Trực quan của dữ liệu không gian địa lý  
   Biểu đồ Gantt  
   Gists  
   GitHub cũng vận hành một trang web kiểu pastebin có tên là Gist, dành cho các đoạn mã, trái ngược với GitHub, dành cho các dự án lớn hơn. Tom Preston-Werner đã giới thiệu tính năng này tại một hội nghị Ruby vào năm 2008.  
   Gist xây dựng dựa trên khái niệm đơn giản truyền thống về pastebin bằng cách thêm kiểm soát phiên bản cho các đoạn mã, dễ dàng phân nhánh và được mã hóa TLS cho các gists riêng tư. Bởi vì mỗi "gists" là kho lưu trữ Git của riêng nó, nhiều đoạn mã có thể được chứa trong một trang duy nhất và chúng có thể được đẩy và kéo (push and pull) bằng cách sử dụng Git.  
   Người dùng chưa đăng ký có thể tải lên Gists cho đến ngày 18 tháng 2 năm 2018, khi việc tải lên gists chỉ có sẵn cho những người dùng đã đăng nhập, được báo cáo là để giảm thiểu việc spam.  
   URL của gists sử dụng ID hệ thập lục phân và các chỉnh sửa đối với gists được ghi lại trong lịch sử sửa đổi, lịch sử này có thể hiển thị sự khác biệt về văn bản của ba mươi bản sửa đổi trên mỗi trang với tùy chọn giữa chế độ xem "phân tách" và "hợp nhất". Giống như kho lưu trữ, Gists có thể được phân nhánh, "gắn dấu sao", tức là được đánh dấu công khai và nhận xét. Số lượng các bản sửa đổi, số sao và phân nhánh được hiển thị trên trang chính.  
   Github dành cho doanh nghiệp  
   Github cho doanh nghiệp cũng hoạt động giống github.com, nhưng hỗ trợ phiên bản trả phí cho các doanh nghiệp muốn bảo vệ mã nguồn của mình, không công khai ra cộng đồng.  
   Việc làm  
   Một trong những nguồn thu nhập khác của Github là GitHub Jobs nơi sử dụng lao động có thể gửi lời mời làm việc với giá 450$/tháng. Nhân viên bán hàng của GitHub không được trả lương trên cơ sở hoa hồng.

## Cách cài đặt trên window, linux( có thực hiện cài đặt) Cách cài GIT: Bước 1: Tải về

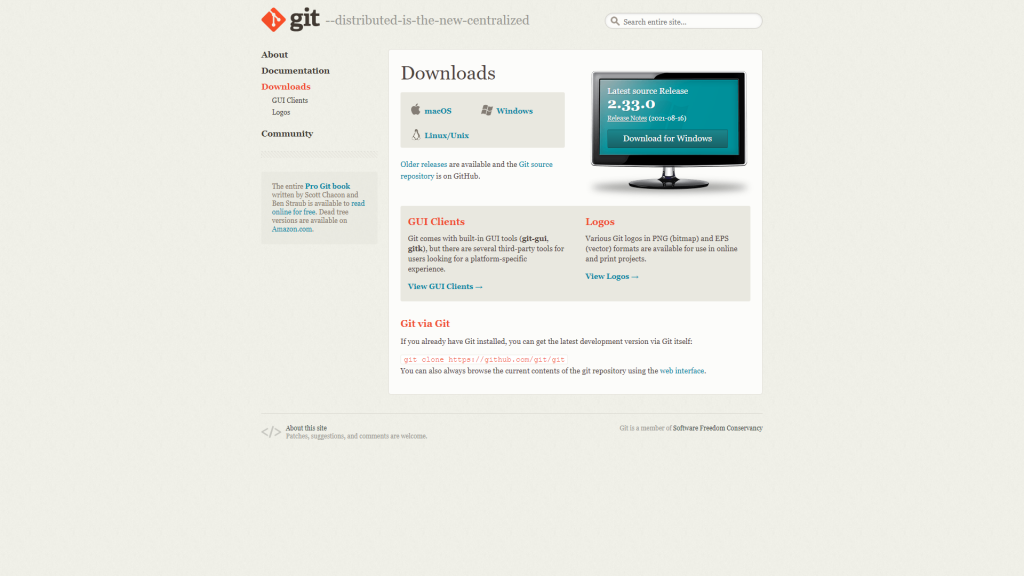
## Có nhiều cách để đến với trang chủ của Git, bạn có thể tìm kiếm trên Google và lựa chọn trong kết quả tìm kiếm hoặc đến trực tiếp trang chủ bằng cách gõ vào thanh điều hướng trình duyệt đường dẫn

[**https://git-scm.com/**](https://git-scm.com/)

Bấm vào nút **Download 2.23.0 for Windows** để tải tập tin cài đặt về.  
Lưu ý rằng con số ***2.23.0*** là phiên bản ***Git*** hiện tại được phát hành, nó có thể bị thay đổi bởi các phiên bản mới hơn. Xem thêm ảnh minh họa bên dưới.



Hệ thống trang tải của Git sẽ tự động gửi tập tin cài đặt phù hợp với hệ điều hành Windows hiện tại của bạn. Ngoài ra, bạn còn có thể tải thêm các phiên bản khác tùy ý. Xem thêm ảnh minh họa bên dưới.



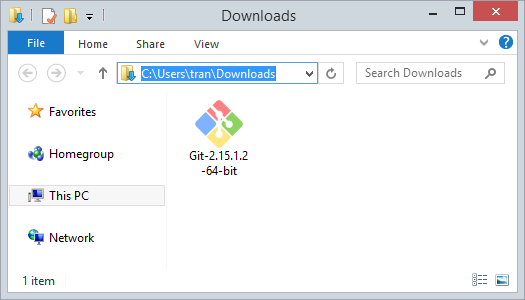
## Bước 2: Cài đặt

Sau quá trình chờ tải về, bạn sẽ có được một tệp thực thi trên máy có tên dạng  
**Git-2.23.0-64-bit.exe**, trong đó:

* **2.23.0** đã nói ở trên bước 1 rồi, là phiên bản của Git lúc đó.
* **64-bit** là kiến trúc mà hệ điều hành Windows của máy đang dùng để cài Git. Xem thêm ảnh minh họa bên dưới.

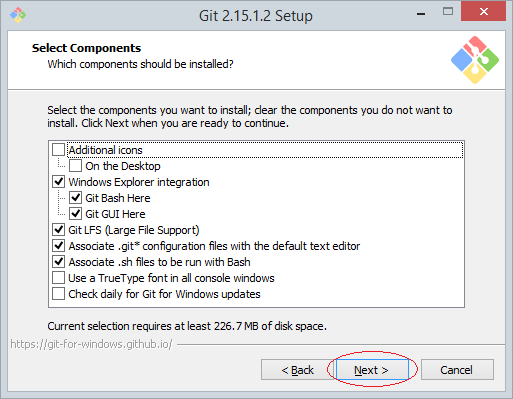
Thực thi tập tin đó bằng cách Bấm đôi chuột hoặc Bấm chuột phải lên tệp đó rồi chọn Open. Tùy bạn thôi!

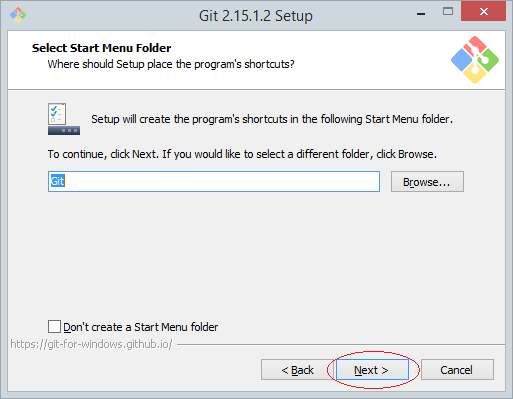
Nếu như máy của bạn có sử dụng tường lửa hoặc phân quyền để chặn các truy cập không khả tín thì bạn sẽ gặp thông báo Cảnh báo bảo mật, mình cứ chọn Run cho nó thực hiện cài đặt. Xem thêm hình ảnh minh họa bên dưới.

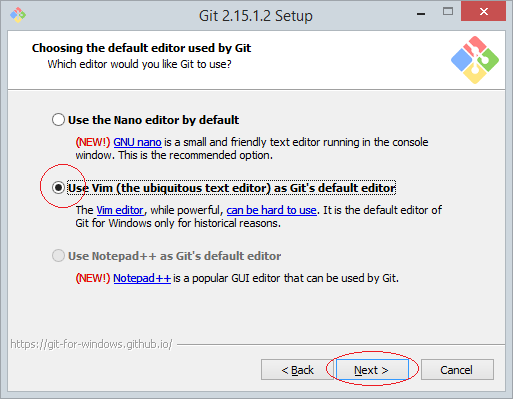


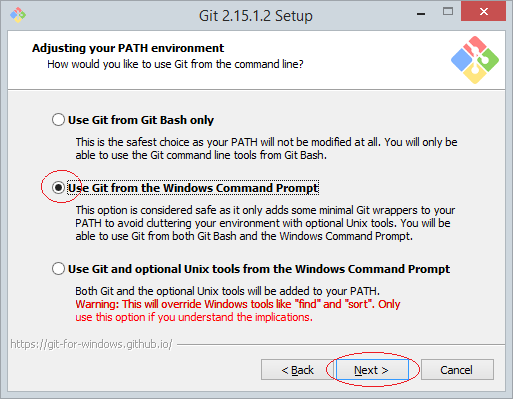
Cài đặt **Git** trên **Windows** rất đơn giản, về cơ bản bạn có thể chấp nhận các mặc định và nhấn **Next, Next …** cho tới khi hoàn thành.

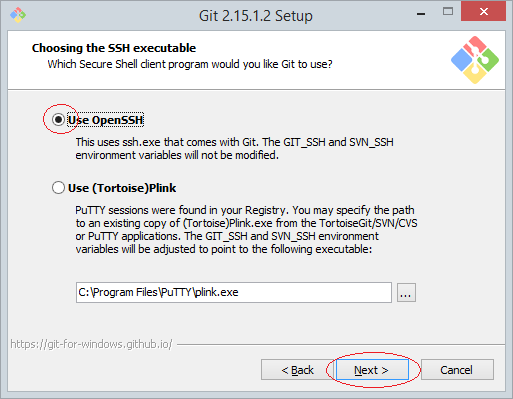


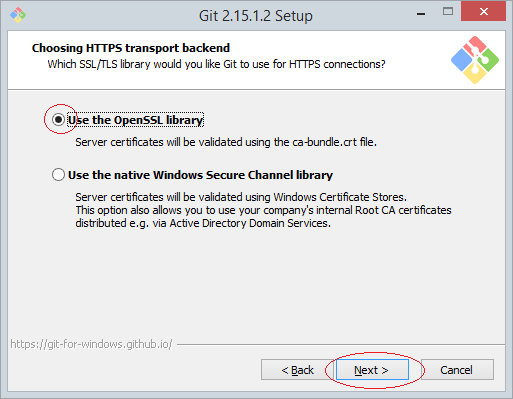


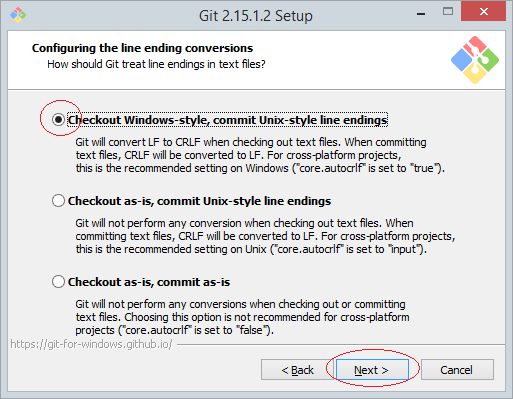


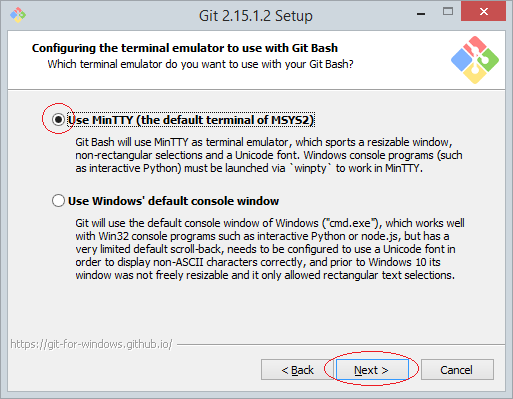


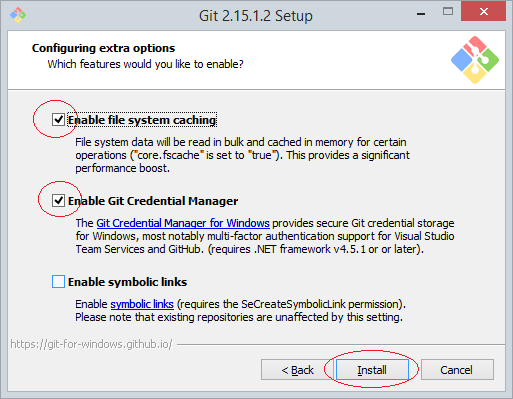


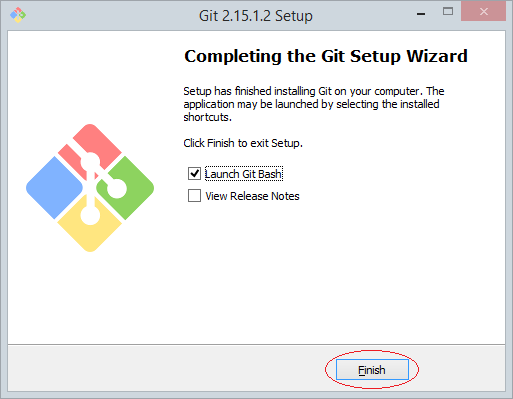




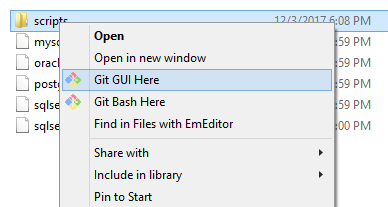


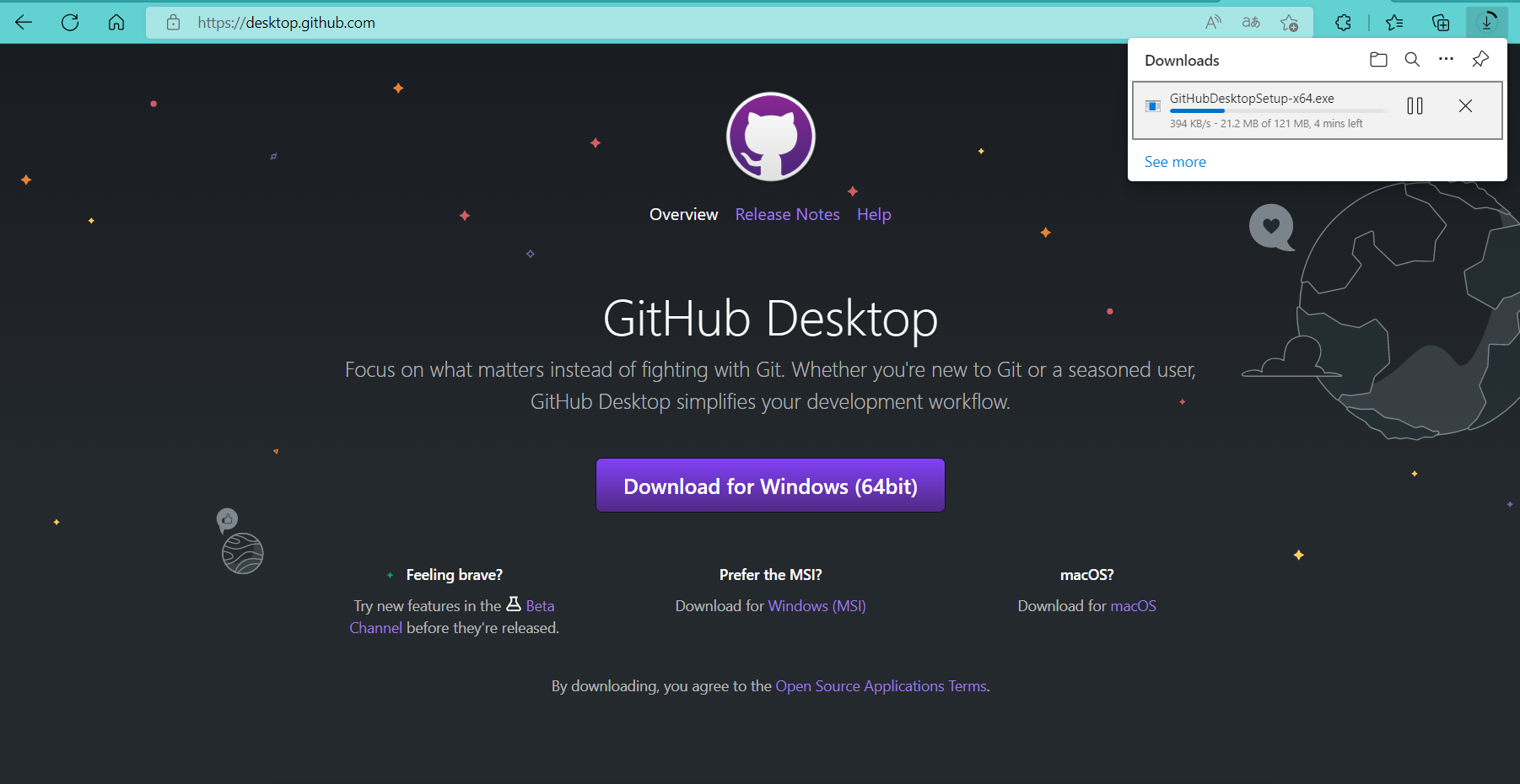
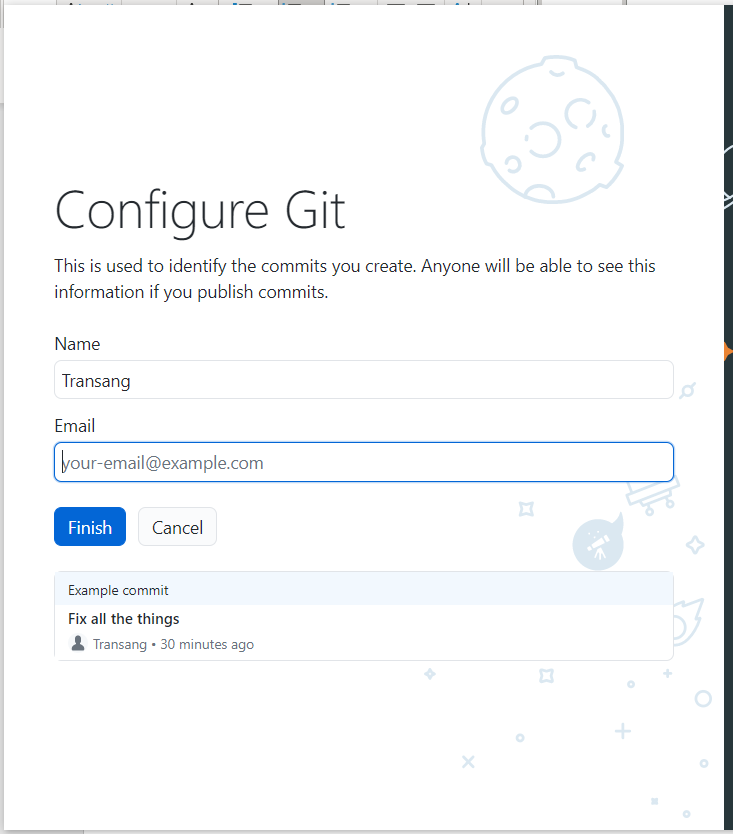






OK, Bạn vừa cài đặt xong **Git**.Sau khi cài đặt xong, nhấn phải chuột vào một thư mục bất kỳ, một **Context-Menu** sẽ hiển thị, bạn có thể nhìn thấy các **Menu-Item** của **Git**, điều này chứng tỏ rằng bạn đã cài đặt **Git** thành công.



Cách cài GITHUB:  
B1: tải GitHub từ trang web desktop.github.com   
B2:Nhập tên và Email  


1. Staging và commits(init, Add, commit,...)

Staging Area

Staging Area nghĩa là một khu vực mà nó sẽ được chuẩn bị cho quá trình commit. Trước hết, bạn cần phải hiểu rằng trong các hệ thống quản lý phiên bản (Version Control System) thì các dữ liệu sẽ được lưu trữ ở hai nơi, một là thư mục bạn đang làm việc trên máy tính (working tree, mình không nhắc lại nữa đâu) và một là kho chứa mã nguồn (repository) sau khi bạn đã thực hiện thay đổi (ví dụ như kho chứa trên Github).

Nhưng với Git thì nó có thêm một lựa chọn nữa đó là có thêm một khu vực trung gian gọi là Staging Area và đây chính là một lợi thế lớn của Git. Staging Area nghĩa là khu vực sẽ lưu trữ những thay đổi của bạn trên tập tin để nó có thể được commit, vì muốn commit tập tin nào thì tập tin đó phải nằm trong Staging Area. Một tập tin khi nằm trong Staging Area sẽ có trạng thái là Stagged.

Và để đưa một tập tin vào Staging Area thì bạn sẽ cần phải sử dụng lệnh git add tên\_file mà mình đã có ví dụ ở phần trước.

Commit và hoạt động của Commit

Hiểu đơn giản hơn, commit nghĩa là một hành động để Git lưu lại một bản chụp (snapshot) của các sự thay đổi trong thư mục làm việc, và các tập tin và thư mục được thay đổi đã phải nằm trong Staging Area. Mỗi lần commit nó sẽ được lưu lại lịch sử chỉnh sửa của mã nguồn 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 phân nhánh (branch) khác, đây là mấu chốt của việc bạn sẽ dễ dàng khôi phục lại các thay đổi trước đó mà mình có giới thiệu qua ở phần giới thiệu serie này.

Và tất nhiên, lệnh commit trong Git sẽ là git commit -m "Lời nhắn".

Và nếu bạn muốn đưa tập tin lên repository thì bạn phải commit nó trước rồi sau đó lệnh git push origin master sẽ có nhiệm vụ đưa toàn bộ các tập tin đã được commit lên repository.

Điều kiện để commit một tập tin

Nếu bạn muốn commit một tập tin đó, bạn sẽ cần phải đưa tập tin đó vào trạng thái tracked bằng lệnh git add tên\_file. Trong git có hai loại trạng thái chính đó là Tracked và Untracked, cụ thể:

Tracked – Là tập tin đã được đánh dấu theo dõi trong Git để bạn làm việc với nó. Và trạng thái Tracked nó sẽ có thêm các trạng thái phụ khác là Unmodified (chưa chỉnh sửa gì), Modified (đã chỉnh sửa) và Staged (đã sẵn sàng để commit).

Untracked – Là tập tin còn lại mà bạn sẽ không muốn làm việc với nó trong Git.

Nhưng bạn phải nên biết rằng nếu tập tin đó đã được Tracked nhưng đang rơi vào trạng thái (Modified) thì nó vẫn sẽ không thể commit được mà bạn phải đưa nó về Staged cũng bằng lệnh git add.

1. Checkout, Revert, Reset, Rm,...

Checkout:

Chúng ta có thể thực hiện nhiều thao tác bằng lệnh git checkout như chuyển sang một chi nhánh cụ thể, tạo một nhánh mới, checkout một chi nhánh từ xa, v.v. Các lệnh git branch và git checkout có thể dùng làm điều này.

Bạn có thể trình bày cách xem danh sách các nhánh có sẵn bằng cách thực hiện lệnh git branch và chuyển sang một nhánh được chỉ định.

Cú pháp:

git checkout <branchname>

Tạo và chuyển đổi nhánh

Các lệnh git checkout cho phép bạn tạo và chuyển sang một nhánh mới. Bạn không chỉ có thể tạo một nhánh mới mà còn có thể chuyển đổi nó đồng thời bằng một lệnh duy nhất. Tùy chọn git checkout -b là một cờ tiện lợi thực hiện thao tác run git branch <new-branch> trước khi chạy git checkout <new-branch>.

Cú pháp:

git checkout -b <branchname>

Checkout nhánh từ xa

Git cho phép bạn checkout một nhánh từ xa bằng lệnh git checkout. Đó là một cách để một lập trình viên tiếp cận công việc của đồng nghiệp hoặc cộng tác viên để xem xét và cộng tác. Mỗi kho lưu trữ từ xa chứa tập hợp các nhánh của riêng nó. Vì vậy, để kiểm tra một nhánh từ xa, trước tiên bạn phải tìm nạp(fetch) nội dung của nhánh.

git fetch --all

Trong các phiên bản mới nhất của Git, bạn có thể checkout nhánh từ xa giống như nhánh cục bộ.

Cú pháp:

git checkout <remotebranch>

Các phiên bản trước đó của Git yêu cầu tạo một nhánh mới dựa trên điều khiển từ xa. Trong các phiên bản trước, lệnh dưới đây được sử dụng để kiểm tra(checkout) nhánh từ xa.

git checkout <remotebranch> origin / <remotebranch>

Checkout dùng để chuyển nhánh

Giả sử đang ở nhánh nào đó, muốn chuyển sang nhánh master thì thực hiện lệnh:

git checkout master

Lúc này nhánh master hoạt động, và thư mục làm việc là các file tương ứng với nhánh này.

Phục hồi file từ phiên bản cũ

Giả sử có file index.html, muốn phục hồi nó về phiên bản ở commit có mã hash là HASH, thì thực hiện:

git checkout HASH index.html

Nếu bạn muốn phục hồi nội dung từ index (staging nếu có, nếu không từ commit cuối) thì đơn giản là:

git checkout index.html

Phục hồi nhiều file, ví dụ \*.html từ index (staging nếu có, nếu không từ commit cuối)

git checkout -- \*.html

Có thể thực hiện với tất cả các file bằng

git checkout -- .

Khi bạn trở về hẳn một commit có mã HASH nào đó bằng lệnh:

git checkout HASH

Thì lúc này con trỏ HEAD sẽ chuyển đến commit này, và Git ở chế độ head detached, bạn làm việc trên một nhánh tạm thời

Nếu có thực hiện các commit trên nhánh này và cần lưu lại thì cuối cùng tạo nhánh mới bằng lệnh

git switch -c ten-nhanh-moi

Sử dụng git switch

Lệnh này dùng để chuyển nhánh và có thể tạo nhánh mới, ví dụ chuyển về nhánh master

git switch master

Tạo nhánh mới, kích hoạt nhánh bắt đầu từ một commit có mã HASH

git switch -c ten-nhanh HASH

Hoặc tạo nhanh từ commit cuối

git switch -c ten-nhanh

Chuyển về làm việc tại nhánh tạm thời bắt đầu từ commit có mã HASH

git switch --detach HASH

Sử dụng git restore

Lệnh git restore để phục hồi các file của thư mục làm việc.

Để phục hồi tất cả các file dùng lệnh:

git restore .

Cách sử dụng giống như git checkout cho trường hợp phục hồi

Revert:

Trong Git, thuật ngữ hoàn lại, hoàn lại, hồi lại(revert) được sử dụng để hoàn lại một số thay đổi. Lệnh git revert được sử dụng để áp dụng thao tác hoàn lại. Nó là một loại lệnh hoàn tác. Tuy nhiên, nó không phải là một thay thế huỷ bỏ truyền thống. Nó không xóa bất kỳ dữ liệu nào trong quá trình này, thay vào đó, nó sẽ tạo ra sự thay đổi với tác dụng ngược lại, đó là hoàn lại commit đã chỉ định nào đó. Nói chung, git revert là một commit.

* Tùy chọn trong Git revert:

Git hoàn lại cho phép một số thao tác bổ sung như chỉnh sửa, không chỉnh sửa, dọn dẹp và hơn thế nữa. Chúng ta cùng tìm hiểu ngắn gọn các tùy chọn này:

<commit>: Tùy chọn commit được sử dụng để hoàn lại một commit. Để hoàn lại một commit, chúng ta cần id tham chiếu commit. Lệnh git log có thể truy cập nó.

git revert < commit-ish >

<–edit>: Nó được sử dụng để chỉnh sửa tin nhắn commit trước khi hoàn lại commit. Nó là một tùy chọn mặc định trong lệnh git revert.

git revert -e < commit-ish >

-m parent-number / – mainline parent-number: nó được sử dụng để hoàn lại việc hợp nhất(merge). Nói chung, chúng ta không thể hoàn lại một hợp nhất(merge) vì chúng ta không biết mặt nào của hợp nhất(merge) nên được coi là nhánh chính để merge. Chúng ta có thể chỉ định parent-number và cho phép hoàn lại để đảo ngược thay đổi liên quan đến parent đã chỉ định.

-n / –no edit: Tùy chọn này sẽ không mở trình soạn thảo văn bản. Nó sẽ trực tiếp hoàn lại commit cuối cùng.

git revert -n < commit-ish >

–cleanup = <mode>: Tùy chọn xóa sạch cách khoảng trắng và nhận xét khỏi tin nhắn.

-n / – no-commit: Nói chung, đây là lệnh hoàn lại commit theo mặc định. Tùy chọn không commit sẽ không tự động commit. Ngoài ra, nếu tùy chọn này được sử dụng, chỉ mục(index) của bạn không phải khớp với commit HEAD.

- Git hoàn lại về commit trước đó

git log

* Git Revert Merge

Trong Git, hợp nhất(Merge) cũng là một commit có ít nhất hai 2 nhanh gọp lại. Nó kết nối các nhánh và code để tạo ra một dự án hoàn chỉnh.

Hợp nhất trong Git là một commit có ít nhất 2 nhanh gọp lại. Nó tập hợp nhiều dòng phát triển. Trong luồng công việc mà các tính năng được phát triển trong các nhánh và sau đó được hợp nhất thành một dòng chính, các commit hợp nhất thường sẽ từ 2 nhánh nào đó.

* Cách hoàn lại hợp nhất

Thông thường, hoàn lại một hợp nhất được coi là một quá trình phức tạp. Nó có thể phức tạp nếu không được thực hiện đúng cách. Chúng ta sẽ hoàn lại thao tác hợp nhất với sự trợ giúp của lệnh git revert. Mặc dù một số lệnh khác như git reset có thể thực hiện được. Chúng ta sẽ tìm hiểu cách hoàn lại một hợp nhất.

Git reset

Git reset sinh ra để ta sửa chữa những lỗi lầm đó.

git reset rất mạnh, nó có thể cứu rỗi những lỗi lầm, nhưng nếu bị dùng sai cách, nó có thể trở thành thảm họa. Nên hãy nắm thật chắc nó nhé.

Cấu trúc của một câu lệnh git reset:

git reset [--soft | --mixed | --hard | --merge | --keep] <commit>

Về cơ bản ta hay sử dụng 3 lệnh sau là chủ yếu:

git reset <commit>

git reset --soft <commit>

git reset --hard <commit>

trong đó git reset <commit> thực chất chính là sử dụng tham số mặc định –mixed

Rm:

1. Lệnh git rm

Lệnh git rm được sử dụng để xóa các file khỏi cây làm việc và chỉ mục.

Nếu chúng ta muốn xóa file khỏi kho lưu trữ của mình. Sau đó, nó có thể được thực hiện bằng lệnh git rm. Hãy lấy một file tin newfile.txt để kiểm tra lệnh rm. Lệnh git rm sẽ được vận hành như sau:

git rm <Tên file>

Lệnh trên sẽ xóa file khỏi Git và kho lưu trữ. Lệnh git rm loại bỏ file không chỉ khỏi kho lưu trữ mà còn khỏi vùng dàn. Nếu chúng ta kiểm tra trạng thái của kho lưu trữ, thì nó sẽ hiển thị là đã bị xóa.

Tùy chọn được lưu trong bộ nhớ cache được sử dụng trong trường hợp này. Nó chỉ định rằng hoạt động loại bỏ sẽ chỉ hoạt động trên chỉ mục dàn dựng chứ không phải trên kho lưu trữ. Lệnh git rm với tùy chọn được lưu trong bộ nhớ cache sẽ được sử dụng như:

git rm --cached <tên file>

Lệnh trên sẽ xóa một file khỏi hệ thống kiểm soát phiên bản. file đã xóa sẽ vẫn còn trong kho lưu trữ. Bằng cách nào đó, lệnh này sẽ hoạt động như lệnh rm.

* Hoàn lại lệnh git rm

Việc thực thi lệnh git rm không phải là vĩnh viễn; nó có thể được hoàn nguyên sau khi thực thi. Những thay đổi này không thể được duy trì cho đến khi một commit mới được thực hiện trên kho lưu trữ. Chúng ta có thể hoàn tác lệnh git rm. Có một số cách để làm như vậy. Cách thông thường và dễ hiểu nhất là lệnh git reset. Lệnh git reset sẽ được sử dụng như sau:

git reset HEAD

Hoặc chúng ta cũng có thể sử dụng:

git reset --hard

Lệnh trên sẽ đặt lại vị trí của Head. Vì vậy, nó sẽ có được vị trí của điểm trước đó của nó.

1. Log, Diff, Status

Log:

Lệnh git log giúp bạn xem lại thông tin lịch sự commit, nhằm giám sát sự thay đổi của dự án. Lệnh git log có nhiều tham số để xuất ra, định dạng các thông tin hiện thị theo cách mong muốn. Bạn có thể định dạng cách các thông tin mỗi commit được in ra khi xem, cũng như có thể lọc thông tin nào đó.

Mặc đinh thi hành git log nó liệt kê các commit theo thứ tự từ mới nhất đến cũ nhất, mỗi commit có các thông tin gồm: mã hash của commit, dòng thông báo, người tạo commit và ngày tạo commit

git log

Khi số lượng log nhiều, nó hiện thị trước một trang log, sau đó có dấu nhắc chờ lệnh để bạn điều hướng, tìm kiếm ... Để có trợ giúp về các lệnh này hãy nhấn h tại dấu nhắc lệnh

Một số phím chức năng bạn có thể nhập đề điều hướng và tìm kiếm trong log như:

return - dòng tiếp theo

w - trang tiếp

spacebar - trang trước

q - thoát

?pattern - tìm kiếm, với pattern là mẫu tìm kiếm (keyword)

/pattern - giống ?pattern

n - đến vị trí tìm kiếm phía dưới

N - đến kết quả tìm kiếm phía trước

Diff:

Lệnh git diff hiện thị thông tin thay đổi giữa thư mục làm việc và vùng index (staging) hoặc với commit cũ, thông tin thay đổi giữa index(staging) và commit, thông tin thay đổi giữa hai nhánh ...

Mặc định thi hành lệnh như sau:

git diff

Nó hiện thị thông tin tùy ngữ cảnh như sau:

Thông tin khác nhau giữa thư mục làm việc và commit cuối khi mà vùng index (staging) không có dữ liệu gì

Thông tin thay đổi giữa index và commit cuối nếu vùng index có dữ liệu

Status:

Lệnh git status

Lệnh git status hiện thị thông tin khác nhau (do thêm mới, xóa đi, sửa đổi các file) giữa các file trong các trường hợp:

* Khác nhau giữa các file trong vùng staging (chỉ mục) và commit tại con trỏ HEAD (Thường HEAD ở vị trí commit cuối):
* Khác nhau giữa các file trong thư mục làm việc và trong staging
* Khác nhau giữa thư mục làm việc và những file chưa được giám sát bởi Git

Cú pháp

git status

1. Branch

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

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

1. Merge

Trong Git, hợp nhất(merge) là một thủ tục để kết nối lịch sử đã phân nhánh. Nó kết hợp hai hoặc nhiều lịch sử với nhau. Lệnh git merge tạo điều kiện cho bạn lấy dữ liệu được tạo bởi git branch và tích hợp chúng vào một nhánh duy nhất. Git merge sẽ liên kết một loạt các commit thành một lịch sử thống nhất. Nói chung, hợp nhất git được sử dụng để kết hợp hai nhánh.

Nó được sử dụng để duy trì các dòng phát triển riêng biệt; ở một số giai đoạn, bạn muốn hợp nhất các thay đổi trong một nhánh. Điều cần thiết là phải hiểu cách hợp nhất hoạt động trong Git.

Trong hình trên, có hai nhánh master và feature . Chúng ta có thể thấy rằng chúng ta đã thực hiện một số commit trong cả nhánh feature và nhánh chính, và hợp nhất chúng. Nó hoạt động như một con trỏ. Nó sẽ tìm thấy một commit cơ sở chung giữa các nhánh. Khi Git tìm thấy một commit cơ sở được chia sẻ, nó sẽ tạo một “commit hợp nhất” mới. Nó kết hợp các thay đổi của từng chuỗi commit hợp nhất được xếp hàng đợi.

1. Merge & conflict

Git là công cụ làm việc nhóm hiệu quả, nó kiểm soát và quản lý rất tốt các đóng góp cho nội dung dự án giữa nhiều tác giả (thường là nhà phát triển). Đôi khi nhiều nhà phát triển có thể cố gắng chỉnh sửa cùng một nội dung. Nếu nhà phát triển A cố gắng chỉnh sửa mã mà Nhà phát triển B đang chỉnh sửa thì có thể xảy ra xung đột conflict.

1. Git Rebase

Rebasing là một quy trình để áp dụng lại các commit trên một nhánh nào đó. Nó được sử dụng để áp dụng một chuỗi các commit từ các nhánh khác nhau thành một commit cuối cùng. Nó là một thay thế của lệnh hợp nhất git. Nó là một quá trình hợp nhất tuyến tính.

Trong Git, thuật ngữ rebase được gọi là quá trình di chuyển hoặc kết hợp một chuỗi các commit thành một commit cơ sở mới. Rebasing rất có lợi và nó hình dung quá trình trong môi trường của quy trình làm việc phân nhánh tính năng.

Khi bạn thực hiện một số commit trên một nhánh tính năng (nhánh thử nghiệm) và một số trong nhánh chính. Bạn có thể rebase lại bất kỳ nhánh nào trong số này. Sử dụng lệnh git log để theo dõi các thay đổi (lịch sử commit). Kiểm tra nhánh mong muốn mà bạn muốn rebase lại. Bây giờ thực hiện lệnh rebase như sau:

Cú pháp:

git rebase < tên nhánh >

Nếu có một số xung đột trong nhánh, hãy giải quyết chúng và thực hiện các lệnh dưới đây để tiếp tục thay đổi:

git status

Nó được sử dụng để kiểm tra trạng thái,

git rebase --continue

Lệnh trên được sử dụng để tiếp tục với các thay đổi bạn đã thực hiện. Nếu bạn muốn bỏ qua thay đổi, bạn có thể bỏ qua như sau:

git rebase --skip

Khi hoàn thành việc phục hồi. Đẩy kho lưu trữ về nguồn gốc(origin). Hãy xem xét ví dụ dưới đây để hiểu lệnh git merge.

1. Git Squash

Là một người đóng góp có trách nhiệm cho Git, cần phải làm cho quá trình cộng tác hiệu quả và có ý nghĩa. Git cho phép một số công cụ cộng tác mạnh mẽ theo nhiều cách khác nhau. Git squash là một trong những công cụ mạnh mẽ giúp hỗ trợ cộng tác hiệu quả và ít đau đớn hơn.

squash không phải là lệnh; thay vào đó, nó là một trong nhiều tùy chọn có sẵn cho bạn để giảm giá tương tác git. squash cho phép chúng ta viết lại lịch sử. Giả sử chúng ta đã thực hiện nhiều commit trong quá trình làm việc của dự án, việc dồn tất cả các commit thành một commit lớn là lựa chọn đúng đắn hơn là thúc ép. Chúng ta hãy hiểu làm thế nào để squash hai commit.

Bước 1: Kiểm tra lịch sử commit

Để kiểm tra lịch sử commit, hãy chạy lệnh dưới đây:

git log --oneline

Lệnh đã cho sẽ hiển thị lịch sử trong một dòng. Chúng ta có thể theo dõi lịch sử và chọn các commit mà chúng ta muốn xóa.

Bước 1: Kiểm tra lịch sử commit

Để kiểm tra lịch sử commit, hãy chạy lệnh dưới đây:

git log --oneline

Lệnh đã cho sẽ hiển thị lịch sử trong một dòng. Chúng ta có thể theo dõi lịch sử và chọn các commit mà chúng ta muốn xóa.

Các commit trước đó được hiển thị ở đầu trình chỉnh sửa. Nếu chúng ta muốn hợp nhất chúng thành một commit duy nhất, thì chúng ta phải thay thế từ pick bằng squash trên đầu trình soạn thảo. Để viết trên trình chỉnh sửa, nhấn nút ‘ i ‘ để vào chế độ chèn . Sau khi chỉnh sửa tài liệu, nhấn : wq để lưu và thoát khỏi trình chỉnh sửa .

Bước 3: Cập nhật các commit

Khi nhấn phím enter , một cửa sổ mới của trình soạn thảo văn bản sẽ được mở để xác nhận commit. Chúng ta có thể chỉnh sửa thông báo commit trên màn hình này.

Khi chúng ta thoát khỏi trình chỉnh sửa, nó sẽ hiển thị mô tả về các bản cập nhật.

Kết quả ở trên liệt kê mô tả các thay đổi đã được thực hiện trên kho lưu trữ. Bây giờ, các commit đã bị bóp nghẹt. Kiểm tra lịch sử commit để xác nhận với sự trợ giúp của nhật ký git.

Bước 4: Đẩy commit bị bóp

Bây giờ, chúng ta có thể đẩy commit bị bóp nghẹt này trên máy chủ từ xa. Để đẩy commit bị bóp nghẹt này, hãy chạy lệnh dưới đây:

git push origin master

Hoặc là

git push -f origin master

Lệnh trên sẽ đẩy các thay đổi trên máy chủ từ xa. Chúng ta có thể kiểm tra commit này trên kho lưu trữ từ xa của chúng ta.

1. Git Fetch

Lệnh git fetch là một câu lệnh của git, được sử dụng để tải xuống các nội dung từ Remote repository mà không làm thay đổi trạng thái của Local repository (các dữ liệu như commit, các file, refs). Hiểu một cách đơn giản, khi bạn thực hiện lệnh git fetch, git sẽ thu thập và lưu trữ những thay đổi mới từ các branch của Remote repository về máy tính của bạn, nhưng không hợp nhất chúng với Local repository. Với git fetch, bạn có thể theo dõi các commit người khác đã cập nhật lên server, đồng thời nắm bắt được những thông tin khác nhau giữa remote và local.

Git fetch các câu lệnh

git fetch <remote>

Tìm nạp tất cả các branch từ kho lưu trữ. Điều này cũng tải xuống tất cả các commit và tệp được yêu cầu từ kho lưu trữ khác.

git fetch <remote> <branch>

Tương tự như lệnh trên, nhưng chỉ tìm nạp những branch được chỉ định.

git fetch --all

Tìm nạp tất cả các remote đã đăng ký và các branch của chúng:

git fetch --dry-run

Tùy chọn –dry-run sẽ thực hiện chạy thử lệnh. Nó sẽ xuất ra các ví dụ về các hành động mà nó sẽ thực hiện trong quá trình tìm nạp nhưng không áp dụng chúng.

Đồng bộ hoá nguồn gốc với git fetch

Ví dụ sau hướng dẫn quy trình đồng bộ hóa local repository của bạn với branch chính của kho lưu trữ trung tâm.

git fetch origin

1. Git Pull

Git Pull là một lệnh dùng để tải xuống dữ liệu từ một Remote repository và cập nhật Local repository phù hợp với dữ liệu đó. Nói cách khác, Git Pull là lệnh hợp nhất các thay đổi từ Remote repository vào Local repository.

Về bản chất, Git Pull chính là sự kết hợp của 2 lệnh Git Fetch và Git Merge. Giai đoạn đầu, Git Pull sẽ thực thi lệnh Git Fetch ở phạm vi nhánh cục bộ mà HEAD được trỏ đến. Khi dữ liệu được tải xuống, Git Pull sẽ bắt đầu quy trình hợp nhất như Git Merge. Một merge commit mới sẽ được tạo và HEAD cũng được cập nhật để trỏ đến merge commit đó.

Đẩy lên server lần đầu tiên

Nếu là lần đầu tiên đẩy Local Repo lên Remote Repo mới khởi tạo thì cần tạo ra một theo dõi kết nối, upstream giữa local và remote, vậy hãy dùng tham số -u. Ví dụ đẩy lên remote có tên origin và tạo upstream cho nhánh master

git push -u origin master

Đẩy lên server

Sau khi có upstream, mỗi lần cần đẩy dữ liệu lên remote của nhánh master, chỉ việc thực hiện lệnh

1. Git Push

Đẩy lên server tất cả các nhánh

Đẩy tất cả các nhánh ở local lên server có tên origin:

git push origin --all

Xóa một nhánh trên remote

Ví dụ khi cần xóa nhánh beta, trên remote có tên origin

git push origin --delete beta

Ghi đè nhánh với --force

Có thể ghi đè toàn bộ một nhánh ở remote bởi một nhánh ở master, dùng lệnh này cận thận

Ví dụ, khi ghi đè toàn nhánh master ở remote, giống với master của local

git push --force origin master