ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG THÁI NGUYÊN

KHOA CÔNG NGHỆ THÔNG TIN



BÀI TẬP LỚN

GIT VÀ GIT HUB

Môn: Công nghệ phần mềm

Khoa Công Nghệ Thông Tin

Giảng viên hưỡng dẫn: Phạm Thị Hương

Các thành viên trong nhóm: Lâm Văn Giáp, Nông Văn Quỳnh, Hoàng Minh Thao, Nông Đức Thịnh

**MỤC LỤC**

Contents

[chương I. TỔNG QUAN 5](#_Toc55396854)

[1. Tên đề tài: 5](#_Toc55396855)

[2. Tầm quan trọng của Git&GitHub 5](#_Toc55396856)

[3. Mục đích đê tài 5](#_Toc55396857)

[4. Lịch sử hình thành và phát triển, hiện trạng của Git&GitHub 5](#_Toc55396858)

[4.1 Lịch sử hình thành và phát triển của git 5](#_Toc55396859)

[4.2 Sự phát triển của GitHub 7](#_Toc55396860)

[5. Vai trò, ý nghĩa của Git&GitHub 7](#_Toc55396861)

[5.1 Đối với Git: 7](#_Toc55396862)

[5.2 Đối với GitHub 8](#_Toc55396863)

[chương II. NỘI DUNG CƠ SỞ LÝ THUYẾT 9](#_Toc55396864)

[1. Bài toán cụ thể 9](#_Toc55396865)

[2. Quản lý phiên bản phần mềm 11](#_Toc55396866)

[- Một số công cụ quản lý phiên bản 12](#_Toc55396867)

[3. Git là gì? 12](#_Toc55396868)

[3.1 cơ chế checkin check out trong Git 13](#_Toc55396869)

[4. GitHub là gì? 14](#_Toc55396870)

[3.1 Phạm vi 14](#_Toc55396871)

[3.2 Độ tin cậy và quản lý bảo mật trên Github 15](#_Toc55396872)

[3.3 Tính năng API của Github 16](#_Toc55396873)

[3.4 Ưu nhược điểm của GitHub 17](#_Toc55396874)

[5. So sánh SVN với GitHub và ViSual SourceSafe, CVS 17](#_Toc55396875)

[5.1 SVN là gì? 17](#_Toc55396876)

[5.2 ViSual SourceSafe là gì? 18](#_Toc55396877)

[5.3 Hệ thống phiên bản đồng thời (CVS) 18](#_Toc55396878)

[5.4 Ưu điểm của cơ chế làm việc trên nhiều nhánh 19](#_Toc55396879)

[chương III. DEMO CÀI ĐẶT VÀ SỬ DỤNG CƠ BẢN GITHUB 22](#_Toc55396880)

[1. cài đặt và thiết lập ban đầu 22](#_Toc55396881)

[2. Cách tạo một Repository 23](#_Toc55396882)

[2.1 Tạo loca Repository 23](#_Toc55396883)

[3. Tìm hiểu thêm về Commit và Staging Area 27](#_Toc55396884)

[3.1 Staging Area là gì? 27](#_Toc55396885)

[3.2 Commit là gì và nó hoạt động ra sao? 28](#_Toc55396886)

[3.3 Tìm hiểu thêm về trạng thái 29](#_Toc55396887)

[3.4 Untracked 29](#_Toc55396888)

[3.5 Tracked 30](#_Toc55396889)

[4. Branch – kỹ thuật phân nhánh 32](#_Toc55396890)

[4.1 Branch trong Git là gì? 32](#_Toc55396891)

[4.2 Head – con trỏ vị trí 33](#_Toc55396892)

[4.3 Cách tạo một branch 33](#_Toc55396893)

[4.4 chechout một branch 33](#_Toc55396894)

[4.5 Gộp dữ liệu từ một branch 34](#_Toc55396895)

[4.6 xóa branch 35](#_Toc55396896)

[4.7 Làm việc với remote branch 35](#_Toc55396897)

Giới thiệu:

Bài tập lớn môn: Công Nghệ Phần Mềm

Đề tài: Quản lý mã nguồn với công cụ Git & GitHub

1. Danh sách nhóm

|  |  |  |  |
| --- | --- | --- | --- |
| STT | Họ và tên | Lớp | Thực hiện nội dung |
| 1 | Lâm Văn Giáp | CNTT-K17C | Tổng hợp thông tin chương 2 |
| 2 | Nông Văn Quỳnh | CNTT-K17C | Tổng hợp thông tin chương 3 |
| 3 | Hoàng Minh Thao | CNTT-K17C | Tổng hợp thông tin chương 1 |
| 4 | Nông Đức Thịnh | CNTT-K17C | Làm slide |

1. TỔNG QUAN
   1. Tên đề tài:

Tìm hiều về quản lý mã nguồn với công cụ Git&GitHub

* 1. Tầm quan trọng của Git&GitHub

Trong quá trình phát triển phần mềm, chúng ta đều sẽ gặp phải một số vấn đề lớn như : Làm thế nào để quản lý được các phiên bản của quá trình quản lý phần mềm; Khách hang muốn sử dụng lại một phiên bản cũ thì sẽ làm như thế nào; Làm thế nào để quản lý mã nguồn chung cho cả nhóm; Làm thế nào để biết code này là của ai viết trong hệ thống;…. Để giải quyết được những vấn đề trên, phần mềm quản lí phiên bản ra đời và Git&GitHub là một trong số đó. Công cụ này có thể lưu trữ tất cả các phiên bản trong cùng một thư mục, và tất cả những thành viên tham gia dự án có thể truy cập và thực hiện những thao tác thay đổi phiên bản ở trong thư mục, có thể chia sẽ và sử dụng những thay đổi của phiên bản mà các thành viên khác đã cập nhật trước đó. Git&GitHub như một nền tảng lưu trữ online lớn nhất thế giới về các dự án nhiều người làm.

Hơn nữa việc quản lý mã nguồn tốt sẽ có nhiều lợi ích:

* Tối ưu hóa chi phí phát triển, bảo trì và hỗ trợ sau khi hoàn thành
* Cung cấp một môi trường phát triển, bảo trì, thử nghiệm một cách ổn định
* Nâng cao chất lượng, tiết kiệm thời gian phát triển
* Tạo ra tính rõ ràng, minh bạch, tránh được nhầm lẫn
  1. Mục đích đê tài

Tìm hiểu về cách sử dụng các chức năng của Git&GitHub. Các lệnh của GitHub được thực hiện như thế nào. Cách uplate một project lên GitHub hoặc là cách tải một project tử GitHub về máy tính cá nhân. Qua đó những thành viên trong dự án của thể trao đổi, trải nghiệm các phiên bản, sửa lỗi và cập nhật các phiên bản mới nhất mặc dù có khoảng cách về địa lý.

* 1. Lịch sử hình thành và phát triển, hiện trạng của Git&GitHub
     1. Lịch sử hình thành và phát triển của git

Chúng ta bắt đầu với nhân của Linux. 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 Distributed Version Control System (DVCS) - một hệ thống kiểm soát phiên bản phân tán độc quyền tên là BitKeeper. Vào năm 2005, nhân Linux và công ty thương mại phát triển BiKeeper bị phá vỡ, và công cụ đó không còn được cung cấp miễn phí nữa. Chính đều này đã thúc đẩy cộng đồng phát triển nhân Linux (chính xác hơn là Linus Torvalds, người sáng tạo ra Linux) phát triển công cụ riêng của 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” (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 hóa 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.

Dưới đây là một số phiên bản của Git qua từng giai đoạn

|  |  |  |  |
| --- | --- | --- | --- |
| Phiên bản | Ngày đầu phát hành | Phiên bản mới nhất | Ngày phát hành |
| 0.99 | 2005-07-11 | 0.99.9n | 2005-12-15 |
| 1.0 | 2005-12-21 | 1.0.13 | 2006-01-27 |
| 1.1 | 2006-01-08 | 1.1.6 | 2006-01-30 |
| 1.5 | 2007-02-14 | 1.5.6.6 | 2008-12-17 |
| 1.7 | 2010-02-13 | 1.7.12.4 | 2012-10-17 |
| 1.9 | 2014-02-14 | 1.9.5 | 2014-12-17 |
| 2.0 | 2014-05-28 | 2.0.5 | 2014-12-17 |
| 2.1 | 2014-08-16 | 2.1.4 | 2014-12-17 |
| 2.3 | 2015-02-05 | 2.3.10 | 2015-09-29 |
| 2.5 | 2015-07-27 | 2.5.5 | 2016-03-17 |
|  |  |  |  |
| 2.7 | 2015-10-04 | 2.7.4 | 2016-03-17 |
| 2.11 | 2016-11-29 | 2.11.4 | 2017-09-22 |
| 2.15 | 2017-10-30 | 2.15.3 | 2018-09-27 |
| 2.19 | 2018-09-10 | 2.19.2 | 2018-11-21 |
| 2.21 | 2019-02-24 | 2.21.0 | 2019-02-24 |

* + 1. Sự phát triển của GitHub

Sự phát triển của nền tảng GitHub bắt đầu ngày 19 tháng 10 năm 2007. Trang web được đưa ra vào tháng 4 năm 2008 do Tom Preton-Werner, Chris Wanstrath, và PJ Hyett thực hiện sau khi nó được hoàn thành một vài tháng trước đó, xem như giai đoạn beta.

Dự án trên GitHub cso thể được truy cập và thao tác sử dụng một giao diện và làm việc với tất cả các lệnh Git tiêu chuẩn. GitHub cũng cho phép người dùng đăng kí 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.

* 1. Vai trò, ý nghĩa của Git&GitHub
     1. Đối với Git:
* Sắp xếp công việc tốt hơn. Nghĩa là bạn có thể tập trung giải quyết từng task mà không phải bận tâm những task liên quan
* Linh hoạt hơn khi phải làm cùng lúc nhiều task, bở vì bạn có thể cấu trúc công việc dễ dàng hơn
* Tự tin hơn khi thử nghiệm những ý tưởng mới. Vì bạn có thể tách biệt thử nghiệm với dự án chính, điều này giúp nâng cao chất lượng code cũng như tính sáng tạo.
* Git dễ sử dụng, an toàn, 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 kì 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 hơn trong việc triển khai sản phẩm
  + 1. Đối với GitHub

-Quản lý source code dễ dàng

Khi bạn tạo một repo, toàn bộ source code của repo đó được lưu trên GitHub. Tại đây, bạn có thể xem lại quá trình mình đã làm việc thông qua các comment sau mỗi lần commit. Và cái hay ở đây, là nhiều người có thể cùng làm một repo.

Lợi ích đầu tiên, chính là biết được ai đã commit và commit cái gì. Tiếp theo, source có thể phát triển theo nhiều nhánh. Nguyên tắc làm việc với các nhánh như thế này: Chúng ta có thể rẽ nhiều nhánh để phát triển project. Nhưng cuối cùng, phải merge lại vào nhánh MASTER để ra được project hoàn chỉnh.

-Tracking sự thay đổi qua các version

Khi có nhiều member cùng thực hiện một dự án thì khá là phức tạp để theo dõi revisons – ai thay đổi cái gì, lúc nào và mấy cái files đó được stored ở đâu. Đừng lo vì GitHub đã tính đến chuyện này, bằng cách luôn lưu lại những thay đổi đã push lên repository. Cũng tương tự với Microsoft Word hay Google Drive, chúng ta có một lịch sử phiên bản để phòng trường hợp các phiên bản trước đó bị mất hay không được lưu.

* Markdowm

Markdown là một cách định dạng text trên web. Chúng ta có thể chỉnh sửa cách hiển thị của document, format từ như dạng in đậm hay in nghiêng, thêm hình và tạo list những thứ ta có thể làm với markdown. Hầu hết, Markdown chỉ là đoạn text đơn thuần với những kí tự đặc biệt chèn vào, như # hay \*. Trong GitHub thì có thể sử dụng Markdown ở những nơi: Git, Comments tại Issues và Pull requets, các file có đuôi .md hay. markdown…

-GitHub giúp cải thiện kỹ năng code, thậm chí là tracking bug

Có hàng ngàn hàng vạn cách để học, học trên Github sẽ là một ý kiến không tồi trong thời đại này. Với hàng vạn open source projects, hàng trăm ngàn contributors, hàng tỉ commit mỗi ngày thì chỉ bằng việc xem. So sánh, học tập từ những thay đổi đó đã đem lại cho ta hàng tá điều hay để cải thiện kỹ năng code của bản thân.

“Bug tracking” là một tính năng được GitHub tích hợp vào để đơn giản hóa quá trình “tìm và diệt bọ”. Để hiểu được quy trình thì những gì ta cần làm là mở dashboard của từng project lên và filter các thông tin. Sau đó, các câu hỏi sẽ được hệ thống, sắp xếp theo mức độ phổ biến, thời gian update hay tương tại. Phần mềm này cũng có giao diện khá mượt nên luôn được xếp hạng cao trong cộng đồng IT dev.

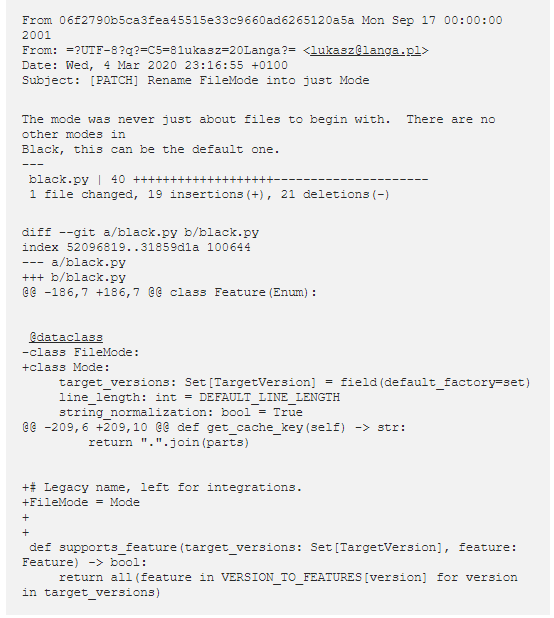
* Github là một kho tài nguyên tuyệt vời

Với chức năng Explore, chúng ta có thể theo dõi, tìm kiếm những open source projects theo đúng technology patter ưa thích. Github hỗ trợ code search không kể nó ở dưới dạng một project riêng biệt hay là website. Ngoài ra, nền tảng này cũng có SEO khá tốt nên người dùng có thể tìm kiếm bất kỳ code string nào được chia sẻ public.

1. NỘI DUNG CƠ SỞ LÝ THUYẾT
   1. Bài toán cụ thể

Khi viết code có rất nhiều lúc chúng ta sẽ muốn hoặc phải quay lại đoạn code mình đã viết tại một thời điểm trước đây. Một cách đơn giản, ta hoàn toàn có thể dùng tính năng undo của trình biên soạn(editor) mình đang dùng. Tuy nhiên, editor có thể không lưu lại lịch sử sau khi đóng, và cũng có thể bấm nhầm sang một phím khác thay vì ấn undo, khiến cho lịch sử trạng thái được editor lưu lại đa bị ghi đè lên khiến chúng ta không thể undo được nữa. Ngoài ra, khi nhiều người cùng làm việc trong một dự án, các thành viên cần tìm ra cách chia sẽ thay đổi của mình để đảm bảo rằng tất cả mọi người đang làm việc trên cùng một phiên bản của dự án. Trước đây, lập trình viên thường chia sẻ code của mình bằng cách xuất ra file pactch và gửi cho nhau qua email. Khi nhận được được một bản patch, các thành viên sẽ áp dụng nó vào trong phiên bản code hiện tại của mình trên máy tính cá nhân. Mỗi file patch chỉ biết đến hai trạng thái của dự án là trước và sau khi được trích suất ra. Như vậy, khi có nhiều file patch, ta cần phải biết thứ tự của chúng để có thể xây dựng lại chính xác một phiên bản của dự án tại bất kì thời điểm nào.

Ví dụ về file patch:



Ngoài ra, vì file patch cũng chỉ là bản file bình thường được lưu trong ổ cứng nên ta cần một cách để sao chúng. Khi có nhiều người cùng làm việc trên một file, ta cần dùng một cơ chế đồng bộ các file patch với nhau, vì rất có thể sẽ có những thay đổi chồng chéo lên nhau.

Để giải quyết các vấn đề nêu trên, ta cần đến một hệ thống quản lí các phiên bản. chính vì vậy, Git&GitHub được ra đời và là một trong những hệ thống quản lý phiên bản phổ biến nhất hiện nay.

* 1. Quản lý phiên bản phần mềm

Quản lý mã nguồn chủ yếu liên quan đến việc theo dõi các sửa đổi với mã. Các công cụ để quản lý mã nguồn đôi khi được gọi là “hệ thống quản lý mã nguồn” (SCMS – Source Control Management System), “hệ thống kiểm soát phiên bản” (VCS – Version Control System), “hệ thống kiểm soát sửa đổi” (RCS – Revision Control System) hoặc đơn giản là kho mã tùy thuộc và các tính năng mà chúng cung cấp hoặc cách chúng được sử dụng. Một kho lưu trữ thường đề cập đến một dự án. Các hệ thống quản lý phiên bản phải bao gồm các tính năng như xác thực truy cập, theo dõi lịch sử sửa đổi, các phiên bản.

Quản lý phiên bản còn được gọi là “kiểm soát phiên bản” hoặc “kiểm soát sửa đổi”, là phương tiện để theo dõi và kiểm soát hiệu quả các thay đổi đối với một tập các thực thể có liên quan. Kiểm soát phiên bản thường được sử sụng nhất để theo dõi và kiểm soát các thay đổi đối với mã nguồn. Nó là một công cụ rất quan trọng trong một chiến lược quản lý với vòng đời tổng thế của môt phần mềm. qua nhiều năm, nhiều hệ thống kiểm soát phiên bản khác nhau đã được phát triển, đặc biệt là trong các lĩnh vực quản lý phiên bản và quản lý tài liệu. Các hệ thống này có thể thương mại hoặc mở nguồn và thường được chạy như các ứng dụng độc lập. Hệ thống quản lý phiên bản có thể hoạt động như kho lưu trữ trung tâm (ví  
dụ như Git). Đối với những hệ thống này, người dùng cá nhân nhận được một bản  
sao hoàn chỉnh của một kho lưu trữ (ví dụ, sao chép nó) và lưu trữ nó cục bộ.  
Những thay đổi đó sau đó được đẩy / kéo vào kho gốc để sẵn sàng cho các thành  
viên khác trong nhóm. Môi trường cộng tác cũng đã được phát triển (ví dụ GForge,  
GitHub, GitLab, v.v.) xung quanh các hệ thống điều khiển phiên bản để tạo thuận  
lợi cho việc tiếp cận theo nhóm dựa trên việc quản lý vòng đời của các ứng dụng.  
Trong phát triển phần mềm mã nguồn mở, những người tham gia hầu hết đến  
từ những vùng địa lý khác nhau, do đó cần có các công cụ để hỗ trợ người tham gia  
trong việc phát triển mã nguồn. Vào đầu những năm 2000, Hệ thống phiên bản đồng thời (CVS) là một ví dụ nổi bật về một công cụ quản lý phiên bản đang được sử dụng trong các dự án phần mềm mã nguồn mở. CVS giúp quản lý các tập tin và mã số của một dự án khi một số người đang làm việc trên dự án cùng một lúc. CVS cho phép nhiều người cùng làm việc trên cùng một tệp. Điều này được thực hiện bằng cách di chuyển tệp vào thư mục của người dùng và sau đó hợp nhất các tệp khi người dùng hoàn tất. CVS cũng cho phép người dùng dễ dàng truy xuất phiên bản 12 trước của tệp. Vào giữa những năm 2000, hệ thống điều khiển sửa đổi  
Subversion (SVN) được tạo ra để thay thế CVS. Nó nhanh chóng đạt được nền tảng như một hệ thống quản lý phiên bản PMNM.

- Một số công cụ quản lý phiên bản  
Có nhiều công cụ quản lý phiên bản như:  
Github: Là nơi lưu trữ source code nổi tiếng thế giới thế giới, Github có chức năng như một nguồn resource phát triển độc lập. Các developer có thể xây dựng project, chia sẻ với cộng đồng và cộng đồng có thể sửa đổi.  
GitLab nó cũng tương tự như GitHub nhưng GitLab theo hướng kinh doanh- Launchpad: Là nhà của untity và nhiểu tiện ích linux phổ biến. Phần lớn các dự án lưu trữ tại launchpad phát triển và xây dựng cho cộng đồng Linux. Hệ thống phiên bản đồng thời (CVS). CVS đã xuất hiện từ những năm 80 và đã rất phổ biến với cả các nhà phát triển thương mại và nguồn mở. Mercurial bắt đầu gần cùng thời với Git và cũng là một công cụ kiểm soát sửa đổi pân tán. Ngoài ra còn có các công cụ khác như SVN, VisualSourceSafe.

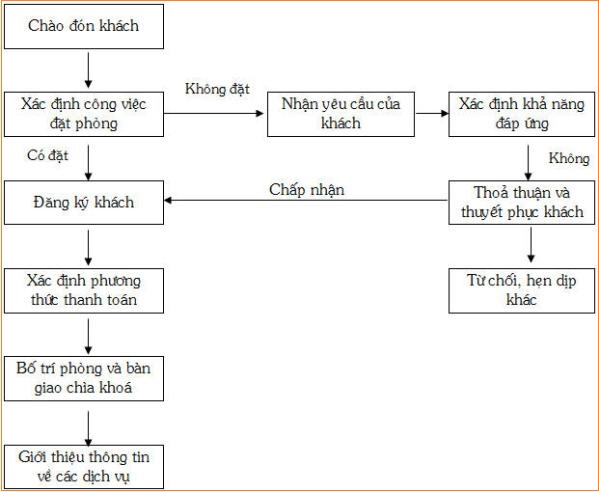
* 1. Git là gì?

Git là tên gọi của một Hệ thống quản lý phiên bản phân tán (Distributed  
Version Control System – DVCS) là một trong những hệ thống quản lý phiên bản  
phân tán phổ biến nhất hiện nay. DVCS nghĩa là hệ thống giúp mỗi máy tính có thể  
lưu trữ nhiều phiên bản khác nhau của một mã nguồn được nhân bản (clone) từ một kho chứa mã nguồn (repository), mỗi thay đổi vào mã nguồn trên máy tính sẽ có thể ủy thác (commit) rồi đưa lên máy chủ nơi đặt kho chứa chính. Và một máy tính khác (nếu họ có quyền truy cập) cũng có thể clone lại mã nguồn từ kho chứa hoặc clone lại một tập hợp các thay đổi mới nhất trên máy tính kia. Trong Git, thư mục làm việc trên máy tính gọi là Working Tree. Ngoài ra, có một cách hiểu khác về Git đơn giản hơn đó là nó sẽ giúp 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 chúng ta ở từng phiên bản, họ cũng có thể đối chiếu các thay đổi rồi gộp phiên bản của ta 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 tái sử dụng lại một ảnh chụp nào đó mà chúng ta 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. cơ chế checkin check out trong Git

Mỗi tập tin trong Git được quản lý dựa trên ba trạng thái: committed, Modified, và staged. Committed có nghĩa là dữ liệu được lưu trữ một cách an toàn trong cơ sở dữ liệu. Modified có nghĩa là đã thay đổi tập tin nhưng chưa commit vào cơ sở dữ liệu. Và Staged là chúng ta đánh dấu sẽ commit phiên bản hiện tại của một tập tin đã chỉnh sửa trong lần commit sắp tới. Điều này tạo ra ba phần riêng biệt của một dự án sử dụng Git: thư mục Git, thư mục làm việc, và khu vực tổ chức(stagingarea)

Cơ chế check in trong Git:



* 1. **GitHub là gì?**

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 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.

3.1 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 là http://projectname.github.io. 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.

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 $450/listing. Nhân viên  
bán hàng của GitHub không được trả lương trên cơ sở hoa hồng. Phổ biến: 24 Tháng Hai năm 2009, trong một cuộc nói chuyện tại Yahoo! trụ sở thành viên trong nhóm GitHub công bố trong một cuộc nói chuyện tại trụ sở trên Yahoo! rằng trong năm đầu tiên GitHub là trực tuyến, nó tích lũy 46.000 kho công cộng, 17.000 trong số họ trong tháng trước đó một mình. Vào thời điểm đó, khoảng 6.200 kho đã được chia hai ít nhất một lần và 4.600 sáp nhập, 05 tháng 7 năm 2009, một Blog Github bài thông báo họ đạt đến 100.000 người sử dụng nhãn hiệu, 27 tháng 7 năm 2009, Tom Preston-Werner thông báo rằng những con số này đã tăng lên 90.000 kho công cộng duy nhất, 12.000 đã được chia hai ít nhất một lần, với tổng số 135.000 kho. Vào tháng 7 năm 2010, GitHub thông báo rằng nó chứa 1 triệu kho. Vào tháng 4 năm 2011, GitHub thông báo rằng nó được lưu trữ 2 triệu kho. 16 Tháng 1 năm 2013, GitHub thông báo đã thông qua 3 triệu người sử dụng đánh dấu và sau đó được lưu trữ hơn 5 triệu kho. Tháng 7 năm 2012, Peter Levine, đối tác ở nhà đầu tư GitHub của Andreessen Horowitz, nói rằng GitHub đã được phát triển doanh thu 300% mỗi năm kể từ năm 2008 "có lợi nhuận suốt từ đó đến giờ".

3.2 Độ tin cậy và quản lý bảo mật trên Github

GitHub có hơn 800 dự án chuyên về bảo mật cung cấp cho các nhà quản trị  
CNTT và các chuyên gia an toàn thông tin đủ loại công cụ để phân tích phần mềm  
độc hại , kiểm tra tấn công xâm nhập, tầm soát máy tính và mạng, ứng phó sự cố,  
giám sát mạng, và nhiều việc khác Kiểm tra tấn công xâm nhập: Khi nói đến kiểm tra tấn công xâm nhập, không có lựa chọn nào tốt hơn Metasploit Framework của Rapid7. Thư viện các kiểu tấn công phong phú của nó có thể sử dụng để đánh giá mức độ an toàn của ứng dụng hoặc hệ thống mạng trước khi bị tin tặc kẻ tấn công. Metasploit có cấu trúc mô-đun linh hoạt cho từng loại thiết bị, dùng để kiểm tra máy tính, điện thoại di động, thiết bị định tuyến (router), chuyển mạch (switch), hệ thống điều khiển công nghiệp và các thiết bị nhúng. Metasploit có thể chạy trên nhiều nền tảng, bao gồm Windows, Linux, Mac, Android và iOS Phòng thủ toàn diện: CFSSL của CloudFlare là "con dao Thụy Sĩ" đa năng cho phép tạo chữ ký số, xác minh và đóng gói chứng chỉ TLS. Vừa là công cụ dòng lệnh vừa là máy chủ HTTP API, CFSSL cho phép nhà quản trị CNTT tạo công cụ TLS/PKI tùy chỉnh và cấp chứng chỉ số (CA) có thể sử dụng nhiều khóa chữ ký. CFSSL còn có tính năng quét TLS kiểm tra cấu hình máy chủ dò tìm lỗ hổng và chuyển gói tin để thiết lập cấu hình hay thu hồi chứng chỉ. Việc vô tình lộ dữ liệu nhạy cảm như các khóa và mật khẩu là vấn đề phổ biến trong phát triển phần mềm. Gitrob giúp các chuyên gia bảo mật quét kho mã nguồn của mình trên GitHub tìm các tập tin nhạy cảm. Tuy GitHub có sẵn chức năng dò tìm những thông tin này, nhưng Gitrob giúp cho công việc đơn giản hơn bằng cách lập danh sách tất cả kho chung và riêng trên GitHub, và dựa trên đó dò tìm các tên tập tin có thể chứa thông tin nhạy cảm. Gitrob lưu kết quả tìm kiếm vào một cơ sở dữ liệu PostgreSQL và hiển thị với một ứng dụng web đơn giản Giám sát mạng: Bro Security Network Monitor cho phép các chuyên gia bảo mật giám sát tất cả máy tính trên mạng (có thể can thiệp vào luồng dữ liệu mạng và kiểm tra các gói tin truyền trên mạng) và cho phép các nhà phân tích kiểm tra lớp ứng dụng. Ngôn ngữ kịch bản của Bro có thể dùng để tạo các chính sách giám sát cho website. Theo thông tin trên trang web của dự án  
(https://github.com/bro/bro), Bro được sử dụng nhiều trong môi trường khoa học  
như các trường đại học, viện nghiên cứu, và các trung tâm điện toán. OSSEC là hệ  
thống phát hiện xâm nhập dựa trên máy chủ có các tính năng theo dõi nhật ký hệ  
thống (log) và quản lý sự kiện và thông tin bảo mật (SIEM - security information  
and event management), có thể chạy trên nhiều nền tảng gồm Linux, Mac OS,  
Solaris, AIX, và Windows. Nó thường được dùng để phân tích log, kiểm tra sự toàn vẹn tập tin, giám sát chính sách, phát hiện rootkit, cảnh báo thời gian thực, … Bằng cách cấu hình OSSEC gửi cảnh báo khi có những thay đổi hệ thống tập tin trái phép hay hành vi độc hại chèn vào các nhật ký phần mềm, các tổ chức và doanh nghiệp có thể đảm bảo việc tuân thủ các chính sách bảo mật . Ứng phó sự cố và điều tra: Mozilla Defense Platform (MozDef) cung cấp cho các chuyên gia bảo mật nền tảng để giám sát, ứng phó và hợp tác đối phó với những sự cố bảo mật trong thời gian thực, cho phép tự động hóa việc xử lý sự cố MozDef sử dụng Elasticsearch, Meteor và MongoDB, mở rộng các tính năng SIEM truyền thống với các biểu đồ, hình ảnh trực quan. Đây là nền tảng hiện được sử dụng tại Mozilla. OS X Auditor phân tích các thành phần mở rộng của hệ thống, các thành phần của bên thứ ba, các tập tin tải về và các ứng dụng cài đặt trên hệ thống đang chạy (hoặc bản sao). Công cụ điều tra này trích xuất thông tin người dùng chẳng hạn như lịch sử và cookie trình duyệt, các tập tin tải về, dữ liệu đăng nhập, tài khoản mạng xã hội và email, kết nối Wi-Fi... và xác minh “uy tín” của từng thứ dựa trên nhiều nguồn. Nghiên cứu và dò tìm lỗ hổng: Công cụ phân tích malware tự động Cuckoo  
Sandbox có nguồn gốc từ một dự án năm 2010 trong chương trình Google Summer  
of Code (hỗ trợ phát triển các dự án mã nguồn mở). Cuckoo cho phép mổ xẻ các tập tin nghi ngờ và giám sát các hành vi có thể gây hậu quả trong một môi trường ảo cô lập, nó kết xuất bộ nhớ và phân tích dữ liệu (chẳng hạn như lần vết các lệnh gọi API) để xác định hành vi của một tập tin đáng ngờ thực hiện trên hệ thống. Jupyter (Jupyter Notebook) là ứng dụng web cho phép tạo và chia sẻ tài liệu (sổ tay) có chứa mã nguồn, ký hiệu, hình ảnh trực quan và văn bản chú giải. Dự án này không chuyên về bảo mật nhưng bất kỳ chuyên gia gia bảo mật nào cũng cần phải có. Có nhiều công cụ bổ sung cho nó, trong đó có Jupyterhub, một máy chủ nhiều người dùng.

3.3 Tính năng API của Github

Ngoài những tính năng tuyệt vời của hệ thống quản lý source phân tán GIT nói chung, Github còn hỗ trợ người dùng những tính năng quan trọng thông qua API sau: API to Update The Repository via HTTP: GitHub hỗ trợ người dùng có thể edit file source code từ web browser thông qua HTTP – POST; API to Access  
Compare Views: Tính năng này hỗ trợ người dùng review và so sánh code của dự  
án thông qua việc xem các commit, comments, các dòng khác nhau giữa 2 version  
của file code ... Tính năng này cũng thông qua HTTP - POST, người dùng có thể  
thực hiên trên web browser; API to Manage Service Hooks: GitHub hỗ trợ tính năng mở rộng post-receive hooks. Tính năng này cho phép người dùng đăng ký 1 URL của mình (như là một web hook) cho các respository. Bất cứ khi nào có người push source code của họ lên repository, GitHub sẽ thông báo cho bạn biết bằng cách POST thông tin (dạng JSON) về lần push đó đến URL mà bạn đã đăng ký trước đó.

3.4 Ưu nhược điểm của GitHub

- Ưu điểm

Git dễ cài đặt và 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). Giúp cải thiện kỹ năng lập trình bằng cách theo dõi và sửa đổi thường xuyên: Chúng ta 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 triển khai sản phẩm. Chứng minh mình là 1 lập trình viên thực thụ. Giúp học hỏi các kỹ năng mới.

-Nhược điểm

Tài khoản github là miễn phí, nhưng kho chứa riêng tư lại bị giới hạn nếu  
muốn dùng thêm phải trả phí. Phát hiện nhiều điểm yếu trên thuật toán SHA1 của  
Github việc này có thể dẫn đến bị mã hóa dữ liệu. Sử dụng github trên window hơi  
cồng kềnh. Hệ thống quản lý phiên bản buộc phải đánh dấu rõ ràng vào tập tin.  
Trong khi điều này đặc biệt phiền toái vì nó lại dính líu đến việc phải liên lạc với  
máy chủ trung tâm.

* 1. **So sánh SVN với GitHub và ViSual SourceSafe, CVS**
     1. SVN là gì?

SVN (viết tắt của Subversion là một hệ thống quản lý version control systemVCS). Nó là 1 hệ thống quản lý phiên bản tập trung. SVN Subversion là hệ thống  
quản lý phiên bản mạnh mẽ, hữu dụng, và linh hoạt. Tích hợp vào Windows  
explorer, mỗi khi cập nhật phải vào đúng thư mục rồi cập nhật, rất bất tiện (nếu  
muốn tích hợp vào trong Visual studio thì phải cài đặt thêm Visual SVN, phải  
tốn tiền mua). Không cho phép cài đặt nhiều loại "tiến trình công việc" (workflow).  
SVN có khả năng xử lý các cấu trúc lồng nhau của các dự án và các gói Java. Rất dễ bị xung đột (conflic) nếu sửa file mà không update trước, nếu đã bị conflic thì bạn phải mở chức so sánh 2 phiên bản rồi tiến hành sửa. SVN Subversion quản  
lý tập tin và thư mục theo thời gian. SVN Subversion giống như một hệ thống file  
server mà các client có thể download và upload file một cách bình thường. Điểm đặt biệt của SVN Subversion là nó lưu lại tất cả những gì thay đổi trên hệ thống file: file nào đã bị thay đổi lúc nào, thay đổi như thế nào, và ai đã thay đổi nó. SVN  
Subversion cũng cho phép recover lại những version cũ một cách chính xác. Các  
chức năng này giúp cho việc làm việc nhóm trở nên hiệu quả và an toàn hơn rất  
nhiều.

* + 1. ViSual SourceSafe là gì?

ViSual SourceSafe (viết tắt là VSS) là 1 phần mềm để quản lý mã nguồn  
bằng cách tạo thư viện ảo cho các tập tin trên máy tính. Được tích hợp sẵn vào  
Visual studio nên dùng rất thuận tiện. Sử dụng 2 cơ chế là check in 1 người và nhiều người (nếu 1 người check in thì người khác ko thể checkin, do đó sẽ không bị vấn đề conflic) VSS không có khả năng sử lý các cấu trúc lông nhau của các dự án và các gói Java. Không được phát triển từ lâu. Đối với nhóm phát triển nhỏ Microsoft, VSS là rất dễ dàng để thực hiện giải pháp, trong đó giải pháp thay thế là không thể kiểm soát VERSION. VSS không có chi phí, không cần máy chủ (trừ một file chia sẻ). Nó thường không làm việc, và một số cửa hàng sử dụng nó. Dễ duy trì phiên bản mới nhất của một tập tin, nhưng lịch sử thường bị hỏng.

5.3 Hệ thống phiên bản đồng thời (CVS)  
CVS đã xuất hiện từ những năm 80 và đã rất phổ biến với cả các nhà phát  
triển thương mại và nguồn mở. Nó được phát hành theo giấy phép GNU và sử dụng một hệ thống để cho phép người dùng kiểm tra các mã mà họ sẽ làm việc trên và kiểm tra các thay đổi của họ. Ban đầu, CVS đã xử lý xung đột giữa hai lập trình viên bằng cách chỉ cho phép phiên bản mã mới nhất được xử lý và cập nhật. Như vậy, đây là lần đầu tiên, hệ thống phục vụ đầu tiên mà người dùng phải xuất bản các thay đổi nhanh chóng để đảm bảo rằng những người dùng khác đã đánh bại họ.  
Bây giờ, CVS có thể xử lý các dự án phân nhánh để phần mềm được phát triển có thể phân chia thành các sản phẩm khác nhau với các tính năng độc đáo và sẽ được đối chiếu sau. Máy chủ CVS chạy trên các hệ thống giống Unix với phần mềm máy khách chạy trên nhiều hệ điều hành. Nó được coi là hệ thống kiểm soát phiên bản trưởng thành nhất vì nó đã được phát triển trong một thời gian dài như vậy và không nhận được nhiều yêu cầu cho các tính năng mới tại thời điểm này. Một dự án rẽ nhánh của CVS, CVSNT đã được tạo để chạy CVS trên các máy chủ Windows và hiện đang được tích cực phát triển để tăng chức năng.

Ưu điểm:

* Đã được sử dụng trong nhiều năm và được coi là công nghệ  
  trưởng thành

Nhược điểm:

* Di chuyển hoặc đổi tên tập tin không bao gồm cập nhật phiên bản  
  Rủi ro bảo mật từ các liên kết tượng trưng đến các tập tin  
  Không được thiết kế để phân nhánh
  + 1. Ưu điểm của cơ chế làm việc trên nhiều nhánh

Hai nhánh (branch) đi suốt chiều dài phát triển code là Master và Develop. Trong đó Nhánh chính là nhánh Master, nhánh này luôn đảm bảo rằng code được lưu trữ trên đó là phiên bản chính thức mới nhất đang được phát hành. Nhánh Develop là nhánh được cập nhật liên tục các đóng góp của tất cả mọi người ở mọi thời điểm. Nhánh này sẽ tiếp nhận các đóng góp của mọi người gửi đến thông qua việc tiếp nhận Pull Request cũng như tiếp nhận việc nhập các nhánh khác vào (integration branch). Khi nhánh Develop đạt độ chín mùi, nó sẽ được nhập vào nhánh Master đồng thời được dán nhãn phiên bản và quá trình phát hành phiên bản được tiến hành. Ngoài 2 nhánh trên, kho code còn có các nhánh hỗ trợ phát triển, và nó chỉ tồn tại trong một giai đoạn nào đó của dự án nhằm phục vụ những mục đích nhất định, gồm có các nhánh sau: Feature branches; Các nhánh tính năng Release branches; Các nhánh hỗ trợ phát hành phiên bản; Hotfix branches ; Các nhánh hỗ trợ vá lỗi nhanh. Để đánh dấu các mốc quan trọng (ví dụ các phiên bản được phát hành), ta sử dụng các tag. Các tag này sẽ giúp truy cập nhanh đến kho code tại thời điểm diễn ra dấu mốc sự kiện được đánh dấu; Nhánh cho phép ta thoải mái thử nghiệm các ý tưởng mới với dự án. Với nhánh, ta có thể thay đổi, vọc phá mà không sợ gây ảnh hưởng đến dự án. Sau đó, nếu ưng ý với kết quả, ta có thể hợp nhất (merge) nó với dự án. Còn nếu không, ta có thể bỏ cả nhánh và dự án không hề bị ảnh hưởng. Nếu bây giờ chạy lệnh git status, ta sẽ thấy ở dòng trên cùng cho biết ta đang ở nhánh master. Đây là nhánh mặc định của Git. Để xem danh sách các nhánh hiện hành, ta dùng lệnh: “-git branch”. Một danh sách các nhánh sẽ hiện ra. Dấu sao đánh dấu nhánh mà ta đang ở. Để tạo một nhánh mới, ta thêm tên của nhánh vào sau lệnh git branch: “-git branch branch\_name”. Sau khi tạo nhánh mới, ta phải chuyển qua nhánh đó mới có thể sử dụng được: “- git checkout branch\_name”. Giờ đây, ta có thể làm việc trên nhánh mới vừa tạo. Những commit thực hiện ở nhánh mới hoàn toàn không ảnh hưởng gì đến nhánh chính (gọi là master) cho đến khi ta ra lệnh hợp nhất (merge). Để quay về nhánh chính, ta dùng lệnh: “-git checkout master”. Ta cũng có thể kết hợp cả 2 lệnh tạo nhánh và chuyển qua nhánh đó trong cùng một dòng: “-git checkout –b branch\_name”. Ngoài ra, để xem toàn bộ commit của tất cả các nhánh thay vì chỉ một nhánh như ta đã biết, ta dùng lệnh: “-git log—all”. Để thêm tên nhánh vào trong danh sách kết quả cho dễ xem, ta thêm vào –decorate: “-git log –all decorate”.

Một số lệnh trên Git

* Git init: Nếu như bạn muốn theo dõi một dự án cũ trong Git, bạn cần ở trong thư mục của dự án đó. Lệnh này sẽ tạo một thư mục mới có tên .git, thư mục này chứa tất cả các tập tin cần thiết cho kho chứa. Tác dụng: Khởi tạo 1 git repository 1 project mới hoặc đã có. Cách dùng: ***git init*** trong thư mục gốc của dự án
* Git clone: Tác dụng: Copy 1 git repository từ remote source. Cách dùng: ***git clone <:clone git url:>***
* Git add: Tác dụng: Thêm thay đổi đến stage/index trong thư mục làm việc. Cách dùng: ***git add***
* Git status: Tác dụng: Để check trạng thái của những file bạn đã thay đổi trong thư mục làm việc. VD: Tất cả các thay đổi cuối cùng từ lần commit cuối cùng. Cách dùng: ***git status*** trong thư mục làm việc
* Git commit: Tác dụng: commit nghĩa là một action để Git lưu lại một snapshot của các sự thay đổi trong thư mục làm việc. Và các tập tin, thư mục được thay đổi đã phải nằm trong Staging Area. 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 ”Đây là message, bạn dùng để note những thay đổi để sau này dễ dò lại”***
* Git pull: Tác dụng: Push hoặc Pull các thay đổi đến remote. Nếu bạn đã added và committed các thay đổi và bạn muốn đẩy nó lên hoặc remote của bạn đã update và bạn apply tất cả thay đổi đó trên code của mình. Cách dùng: ***gitpull <:remote:> <:branch:>*** and ***git push <:remote:> <:branch:>***
* Git branch: Tác dụng: liệt kê tất cả các branch (nhánh).Cách dùng: ***git branch*** hoặc ***git branch -a***
* Git config: Tác dụng: Để set user name và email của bạn trong main configuration file.  
  Cách dùng: Để kiểm tra tên và kiểu email trong cấu hình dùng ***git config -- global user.name*** và ***git config -- global user.email***. Để set email hoặc tên mới ***git config -- global user.name = “Hải Nguyễn”*** và ***git config -- global user.email =***[***lamthao@gmail.com***](mailto:lamthao@gmail.com)
* Git add: Tác dụng: Thêm thay đổi đến stage/index trong thư mục làm việc. Cách dùng: ***git add***
* Git checkout: Tác dụng: Chuyển sang branch khác. Cách dùng: ***git checkout <: branch:>*** hoặc ***\*\* \_ git checkout -b <: branch:>*** nếu bạn muốn tạo và chuyển sang một chi nhánh mới.
* Git stash: Tác dụng: Lưu thay đổi mà bạn không muốn commit ngay lập tức. Cách dùng: ***git stash*** trong thư mục làm việc của bạn
* Git merge: Tác dụng: Merge 2 branch lại với nhau. Cách dùng: Chuyển tới branch bạn muốn merge rồi dùng ***git merge <:branch\_ban\_muon\_merge:>***
* Git remote: Tác dụng: Để check remote/source bạn có hoặc add thêm remote. Cách dùng: ***git remote*** để kiểm tra và liệt kê. Và ***git remote add <: remote\_url:>*** để thêm.

1. DEMO CÀI ĐẶT VÀ SỬ DỤNG CƠ BẢN GITHUB
   1. cài đặt và thiết lập ban đầu

Nếu bạn dùng Windows thì có thể tải file .exe cài đặt Git tại địa chỉ [http://git-scm.com/download/win](https://git-scm.com/download/win). Khi cài bạn có thể để nguyên tùy chọn mặc định mà không cần tùy chỉnh gì thêm nếu bạn chưa hiểu về nó.

Sau khi cài đặt Git vào Windows, bạn sẽ cần mở ứng dụng **Git Bash** lên để bắt đầu sử dụng các dòng lệnh của Git.

Đối với Mac OS , bạn có thể sử dụng file installer tải tại địa chỉ [http://git-scm.com/download/mac](https://git-scm.com/download/mac) để cài đặt.

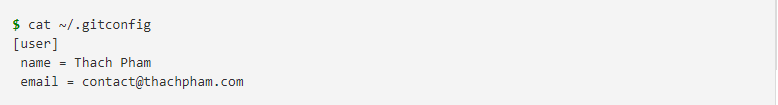
Nếu bạn đang sử dụng hệ điều hành Ubuntu/Debian thì có thể sử dụng lệnh sau để cài Git.



Hoặc lệnh sau để cài trên CentOS/Fedora/RHEL.



Thiết lập chứng thực cá nhân: Sau khi cài Git xong, việc đầu tiên bạn nên làm là khai báo tên và địa chỉ email vào trong file cấu hình của Git trên máy. Để làm điều này bạn sẽ cần sử dụng hai lệnh sau đây để thiết lập tên và email.

  
Sau khi thiết lập xong, bạn có thể kiểm tra thông tin chứng thực trên user của bạn bằng cách xem tập tin ~/.gitconfig (nhắc lại rằng dấu ~ nghĩa là thư mục gốc của user).  
  
 Như vậy là bạn đã xong bước ban đầu đó là cài đặt Git và thiết lập tên và email của mình vào Git để bắt đầu làm việc.

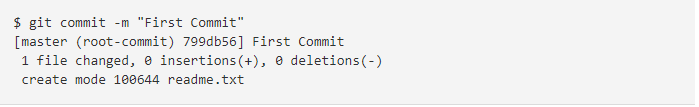
* 1. Cách tạo một Repository

**Repository** (kho chứa) nghĩa là nơi mà bạn sẽ lưu trữ mã nguồn và một người khác có thể sao chép (clone) lại mã nguồn đó nhằm làm việc. Repository có hai loại là Local Repository (Kho chứa trên máy cá nhân) và Remote Repository (Kho chứa trên một máy chủ từ xa).

* + 1. Tạo loca Repository

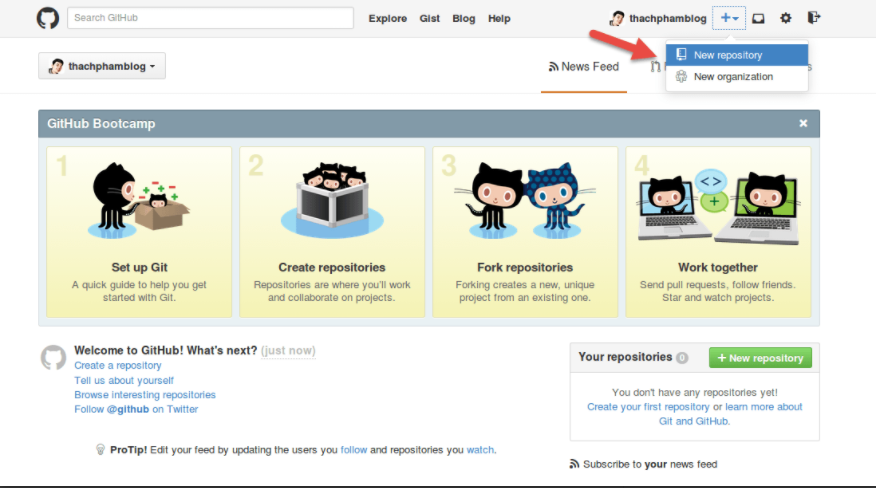
Trước hết, để tạo một repository thì bạn cần truy cập vào thư mục của mã nguồn với lệnh cd, sau đó sử dụng lệnh git init để khởi tạo repository trong thư mục đó. Ở ví dụ này, chúng ta sẽ tạo tạo ra một thư mục mới để chứa code sau này và khởi tạo repository cho nó, dùng lệnh git init tên\_folder để nó tự khởi tạo thư mục.  
  
 Ở đoạn trên, nó hiển thị dòng thông báo đã khởi tạo một kho Git trống tại đường dẫn như trên. Lưu ý rằng thư mục ẩn .git/ là nơi nó sẽ chứa các thiết lập về Git cũng như lưu lại toàn bộ thông tin về kho chứa, bạn **không cần đụng chạm gì vào thư mục**.git/ này.

Nếu kho chứa của bạn đã có sẵn mã nguồn thì bạn cần **phải đưa các tập tin về trạng thái Tracked** nhằm có thể làm việc được với Git. Để làm việc này, bạn sẽ cần sử dụng lệnh git add tên\_file, có thể sử dụng dấu \* để gom toàn bộ. Sau đó có thể sử dụng lệnh git status để xem danh sách các tập tin đã được tracked.  

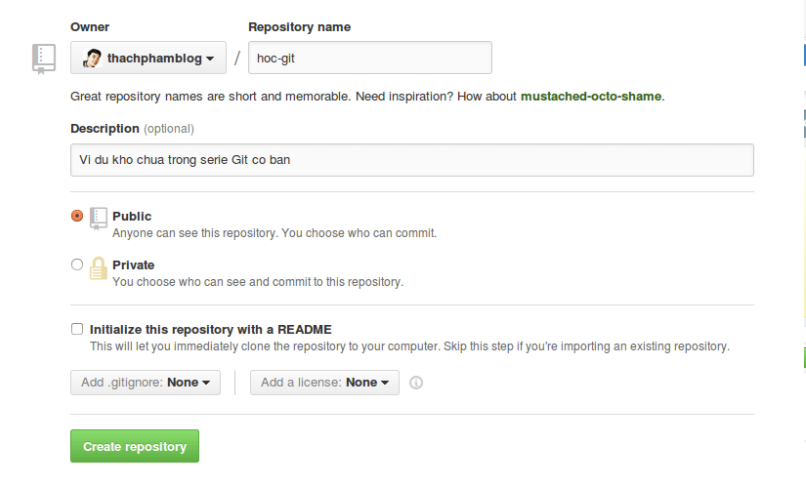

Và sau khi tập tin đã được đưa vào trạng thái tracked và nếu một tập tin đã tracked thì nó phải được đưa vào lại Staging Area (giải thích ở bài sau) cũng bằng lệnh git add thì bạn mới có thể tiến hành ủy thác (**commit**) nhằm lưu lại bản chụp các thay đổi. Lệnh commit sẽ có cấu trúc git commit -m "Lời nhắn", lúc này tất cả các tập đang trong trạng thái tracked (file mới) hoặc một tập tin đã được tracked nhưng có một sự thay đổi mới thì sẽ được commit.  


Bây giờ thì bạn đã hoàn thành việc commit lần đầu tiên các tập tin mà bạn đã đưa vào kho. Tóm lại là tới đây bạn đã có một kho chứa mã nguồn Git trên máy của bạn.  
  
 2.2. Tạo Repository trên GitHub và làm việc

Trước tiên bạn cần đăng nhập vào [Github](https://github.com/" \t "_blank), sau đó ấn vào dấu + trên menu và chọn New repository.



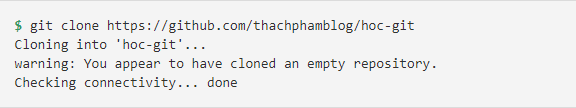
Bạn sẽ cần đặt tên cho kho chứa của bạn. Bạn có thể chọn loại kho chứa là **Public** (ai cũng có thể clone) và **Private** (chỉ có những người được cấp quyền mới có thể clone).



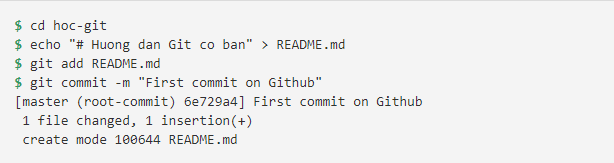
*Điền thông tin khi tạo Repository trên GitHub*

Khi tạo xong nó sẽ dẫn bạn tới trang hướng dẫn làm việc với kho chứa vừa tạo. Và kho chứa của bạn bây giờ sẽ có địa chỉ là https://github.com/$user-name/$repository, ví dụ <https://github.com/thachphamblog/hoc-git>

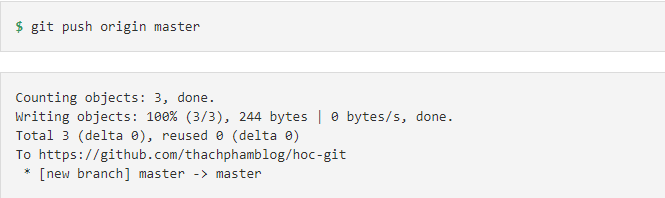
Việc của bạn bây giờ là hãy clone cái kho chứa này về máy của mình bằng lệnh git clone địa\_chỉ.



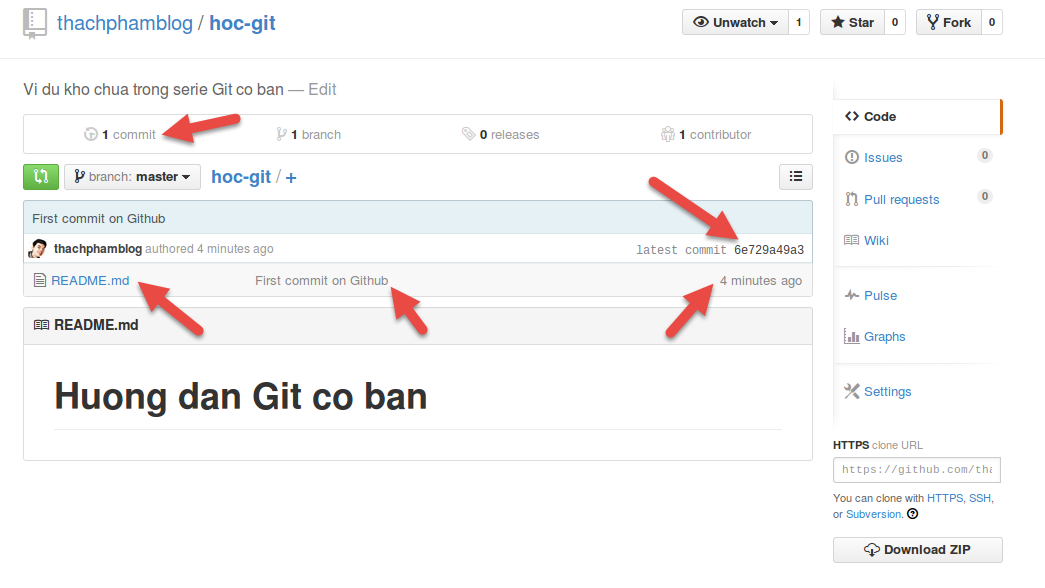
Bây giờ hãy truy cập vào thư mục working tree (thư mục vừa clone repository về) và thử tạo ra một file tên là *README.md*, sau đó dùng lệnh git add để đưa file này vào Staging Area.



Tuy nhiên sau khi commit xong, tập tin đã được commit sẽ vẫn không thể xuất hiện trong kho chứa trên Github mà bạn phải làm thêm một việc nữa đó là dùng lệnh git push để đẩy các tập tin đã được commit lên Github. Lưu ý rằng bạn sẽ cần nhập tài khoản và mật khẩu Github.



origin nghĩa là tên remote và master là tên branch, thích kỹ hơn ở bài riêng. Bây giờ bạn có thể kiểm tra kho chứa của bạn trên Github rồi.

[](https://thachpham.com/wp-content/uploads/2015/04/github-create-repository-03.png)

Kết quả sau khi push mã nguồn đã được commit lên Github

Có thể bạn sẽ thấy mỗi khi push lên Github thì nó hiển thị ra thông báo quá dài dòng, bạn có thể cho ẩn các thông báo đi mà chỉ hiển thị dòng khai báo username và password trên Github thì có thể thiết lập với lệnh này.



* 1. Tìm hiểu thêm về Commit và Staging Area
     1. Staging Area là gì?

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 và một là kho chứa mã nguồn (repository) sau khi bạn đã thực hiện thay đổi (ví dụ như kho chứa trên [Github](https://github.com/" \t "_blank" \o "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**.

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

*Mô hình giải thích cách hoạt động của Staging Area.*

Và để đưa một tập tin vào Staging Area thì bạn sẽ cần phải sử dụng lệnh git add tên\_file.

* + 1. Commit là gì và nó hoạt động ra sao?

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.

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.

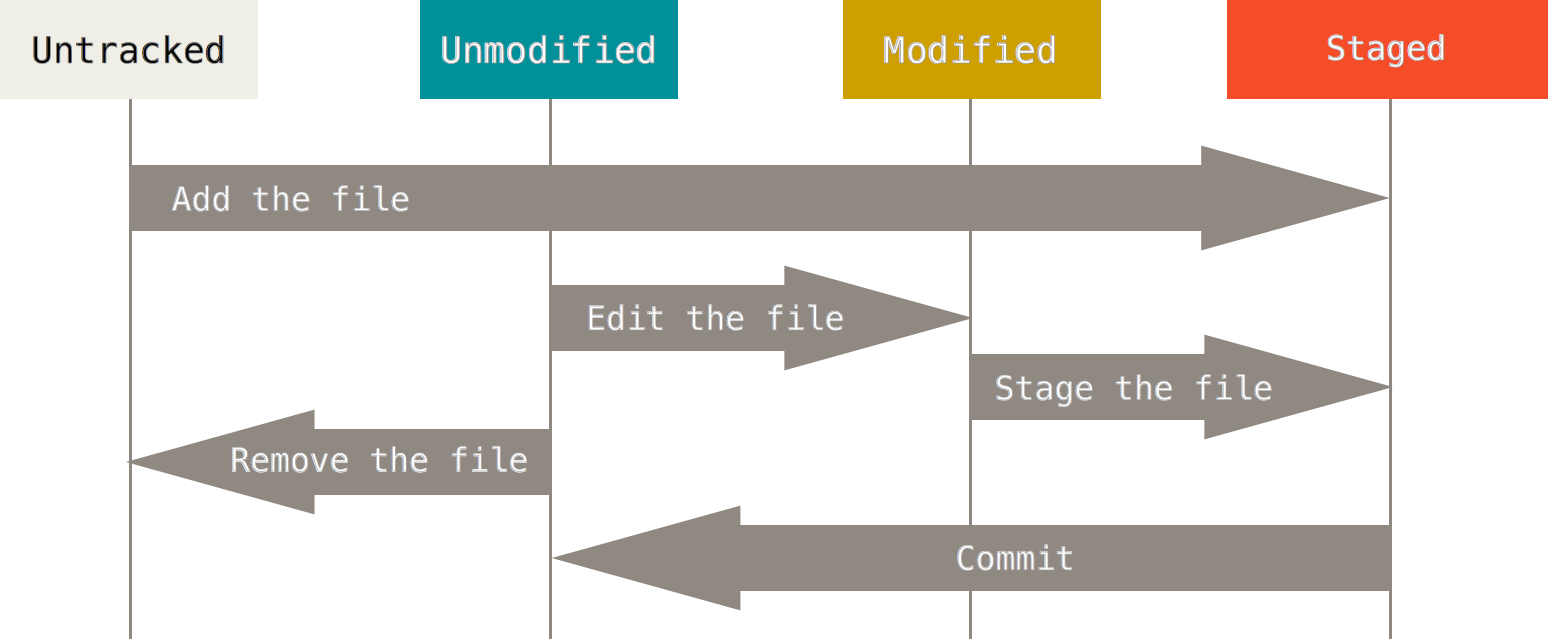
Điều kiện để Commit một tập tin:

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.

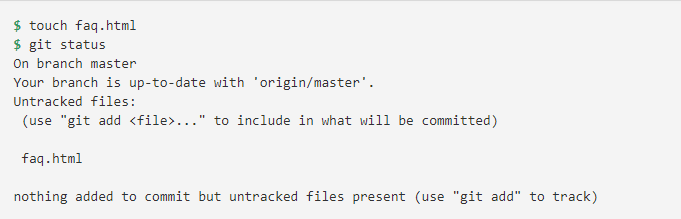
* + 1. Tìm hiểu thêm về trạng thái

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

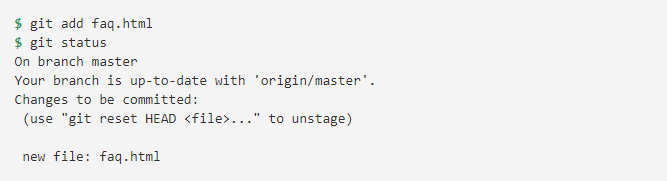
*Vòng đợi trạng thái của các tập tin*

* + 1. Untracked

Nếu bạn tạo ra hoặc thêm vào một tập tin mới vào trong thư mục làm việc của bạn thì nó sẽ ở trạng thái Untracked. Bây giờ thử tạo ra một tập tin mới tên là faq.html, sau đó dùng lệnh git status để xem trạng thái của Git trong thư mục làm việc.



Bây giờ bạn sẽ thấy nó đã liệt kê ra tên tập tin đang ở trạng thái Untracked. Để đưa nó về Tracked bạn sẽ sử dụng lệnh git add và xem lại trạng thái của nó.

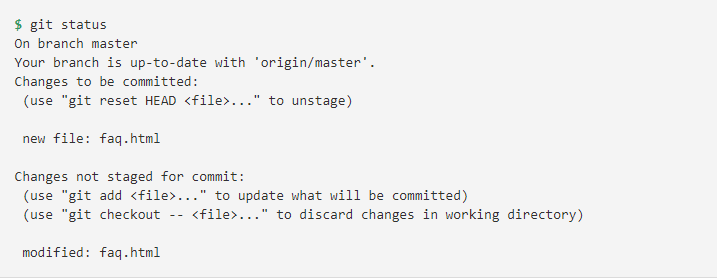


Bây giờ bạn thấy, tập tin faq.html đã được đưa về trạng thái Staged và nó có thể được commit. Tại sao? Vì bạn phải biết rằng nếu một tập tin ở trạng thái Untracked mà được đưa về Tracked thì nó sẽ nằm ở trạng thái Staged luôn, trừ khi bạn thay đổi nội dung tập tin này thì nó sẽ đưa về trạng thái Modified và nó không thể commit trừ khi bạn gõ lệnh git add cho nó.

* + 1. Tracked

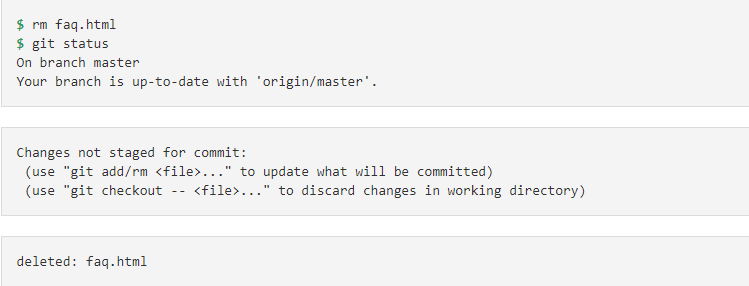
Một khi một tập tin đã được đưa về Tracked thì nó sẽ có thể thay đổi giữa 3 trạng thái khác nhau là **Modified**, **Unmodified** và **Staged**.

Trước hết bây giờ chúng ta đã có một tập tin mới đã được đưa về Staged với lệnh git add như ví dụ trên. Bây giờ tiến hành thay đổi nội dung của tập tin faq.html này và xem kết quả của lệnh git status.



Có một điều kỳ lạ ở tập tin faq.html? Đó là nó được hiển thị ở hai trạng thái Staged (có thể commit) và Modified (không thể commit) hay còn gọi là Unstaged. Sở dĩ có sự kỳ lạ đó ở đây là bởi vì trước đó bạn đã tạo ra tập tin faq.html và đưa về Tracked thì nó cũng đã được đưa về Staged để có thể commit. Tuy nhiên sau đó bạn lại chỉnh sửa nội dung của nó nên nó đã có một phiên bản khác nằm ở trạng thái Modified (không thể commit). Nếu bây giờ bạn gõ lệnh git commit để ủy thác nó thì bản chụp của tập tin faq.html ở lần cuối cùng bạn gõ lệnh git add sẽ được commit lên chứ nó không chứa các nội dung mà bạn vừa thêm vào. Và để nó có thể commit tập tin faq.html đã được chỉnh sửa thì bạn phải gõ lại lệnh git add faq.html lần nữa.

Trong Git, bạn có thể đưa một tập tin từ Tracked về Untracked với lệnh rm tên\_file. Lệnh rm sẽ giúp bạn đưa tập tin về trạng thái Untracked nhưng không xóa hẳn trong ổ cứng.



Còn nếu bạn muốn xóa nó luôn thì dùng lệnh git rm -f tên\_file

* 1. Branch – kỹ thuật phân nhánh

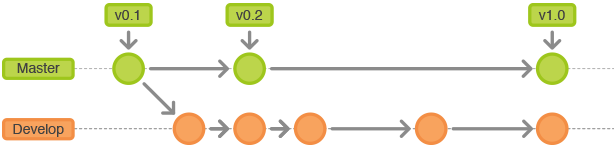
Trong khi làm việc với Git, bạn đã quá quen thuộc với việc chỉnh sửa mã nguồn, sau đó là commit mỗi khi chỉnh sửa xong và push lên remote repository nếu cần thiết. Nhưng bây giờ có một ví dụ đặt ra là muốn **tạo một phiên bản thử nghiệm với mã nguồn đang làm việc trong working tree hiện tại mà không gây ảnh hưởng đến các code hiện tại**. Vậy thì làm cách nào? Không lẽ clone một repository từ chính cái working tree hiện tại rồi sửa đổi hay sao? Như thế rất là mất công, mà lại không tối ưu và không thể đồng bộ hóa hoặc rất khó khăn để đồng bộ hóa.

Mà trong Git, chúng ta sẽ sử dụng một giải pháp khác tuyệt vời hơn, dễ dàng hơn gọi là **phân nhánh** (branching), mà cụ thể là phân nhánh cái gì? Đó là phân nhánh trong working tree hiện tại bạn đang làm việc đấy, và mỗi nhánh chúng ta sẽ gọi nó là một **branch**.

* + 1. Branch trong Git là gì?

Khi bắt đầu khởi tạo một repository hoặc clone một repository, bạn sẽ có một nhánh (branch) chính tên là master (bạn có thể hiểu master là một cái thân cây). Đây là branch mà sẽ chứa toàn bộ các mã nguồn chính trong repository.

Đó là lý do tại sao, ở các phần trước khi chúng ta push hoặc pull hay làm một số việc khác thì lại có tham số master trong câu lệnh, đó nghĩa là chúng ta đang thực hiện thao tác trên branch master.

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

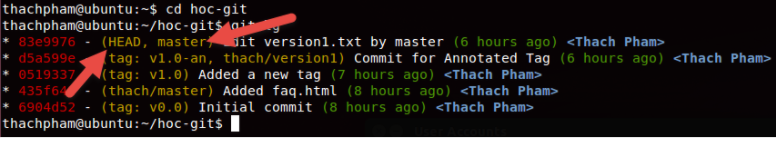
*Nhánh Master là nhánh chính mặc định trong working tree. Bạn có thể tạo ra một (hoặc nhiều nhánh mới) với tên là Develop chẳng hạn.*

Bây giờ nếu bạn muốn tạo một sự thay đổi nào đó mà có thể trực tiếp sử dụng trên working tree hoặc commit, push lên repository mà không ảnh hưởng đến branch master thì sẽ cần tạo ra một branch mới với tên là develop chẳng hạn. Và từ đó mỗi khi bạn thực hiện lệnh checkout vào branch nào đó thì toàn bộ mã nguồn trên working tree của bạn sẽ được đổi sang môi trường dành cho branch đang checkout.

Một ví dụ đơn giản, bên branch master bạn tạo một tập tin *master.html* rồi commit lên, lúc này khi bạn qua cái branch develop (đã được tạo trước đó) thì cái tập tin *master.html* kia sẽ không có vì nó đã được commit bởi branchmaster, không liên quan gì tớidevelop. Tương tự, các thay đổi của bạn bên develop cũng sẽ không ảnh hưởng gì tới bên master cả. Điều này có một cái thú vị là nó tương tác trực tiếp trên máy tính của bạn, ví dụ tập tin index.html của master có nội dung khác và index.html bên develop thì khi bạn dùng lệnh checkout là nội dung trên máy tính nó tự đổi tương ứng, đó là lý do bạn có thể test nhiều phiên bản trên máy tính mà không cần đổi thư mục, chỉ cần checkout cái branch.

* + 1. Head – con trỏ vị trí

Trước khi nói tiếp về branch thì chúng ta nói qua về HEAD. Trong Git, từ khóa HEAD sẽ tượng trưng cho con trỏ chỉ cho bạn biết bạn đang nằm ở đâu.



Như ảnh trên thì có nghĩa là mình đang ở branchmaster.

* + 1. Cách tạo một branch

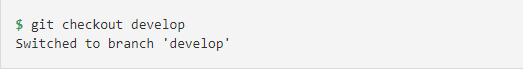
Trước tiên bạn có thể xem toàn bộ các branch mà bạn đang có trong working tree bằng lệnh git branch. Sau đó nếu muốn tạo thêm branch, chỉ cần gõ lệnh git branch tên\_brand. Ví dụ mình cần tạo branchdevelop



Bây giờ có thể gõ lại lệnh git branch một lần nữa để xem sẽ thấy brand tên develop xuất hiện.

* + 1. chechout một branch

Checkout ở đây nghĩa là truy cập kiểm tra mã nguồn trong branch đó để làm việc đấy. Để làm việc này, bạn sử dụng lệnh git checkout tên\_branch.



Lúc này chúng ta đã đổi sang branch develop rồi, để kiểm tra chắc ăn ta có thể gõ các lệnh kiểm tra HEAD ở trên.

Bây giờ sẽ làm việc trong branch mới chuyển hay nói đúng hơn là đang làm việc ở chỗ mà cái HEAD đang trỏ tới. Để chuyển về branch chính thì gõ git checkout master.

Bây giờ ta thử tạo một tập tin nào đó, sau đó commit ở branch develop rồi chuyển về branch master sẽ thấy những gì đã làm ở branch develop hoàn toàn vô nghĩa ởmaster. Dưới đây là ví dụ về việc ở branch master không có tập tin develop.html được tạo ra từ branch develop.



Tương tự với việc sửa tập tin hay bất kỳ làm việc gì khác nó cũng chỉ áp dụng thay đổi ở branch đang trỏ tới.

* + 1. Gộp dữ liệu từ một branch

Nếu mỗi branch nó nằm riêng như vậy thì bạn muốn sử dụng các thay đổi ở một branch nào đó cho master thì sao? À, chúng ta có thể sử dụng lệnh git merge để chuyển dữ liệu từ một branch nào đó về branch mà bạn đang trỏ đến. Lưu ý là ở branch cần chuyển về đã phải được commit. Ví dụ mình cần chuyển dữ liệu từ branch develop về master thì sẽ làm lần lượt các lệnh sau:

****

* + 1. xóa branch

Nếu không cần dùng tới branch nào nữa thì có thể xóa với lệnh git branch -d tên\_branch. Lưu ý là cái branch muốn xóa phải được gộp dữ liệu (merge) về master.

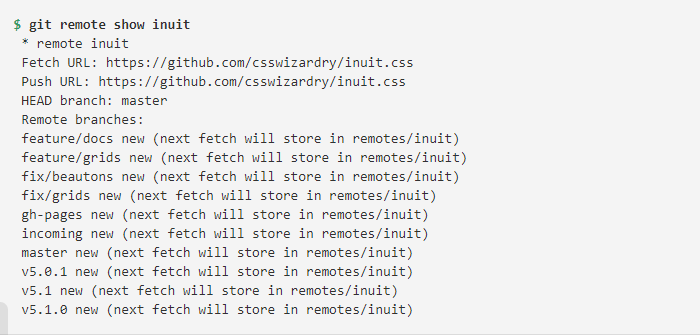


* + 1. Làm việc với remote branch

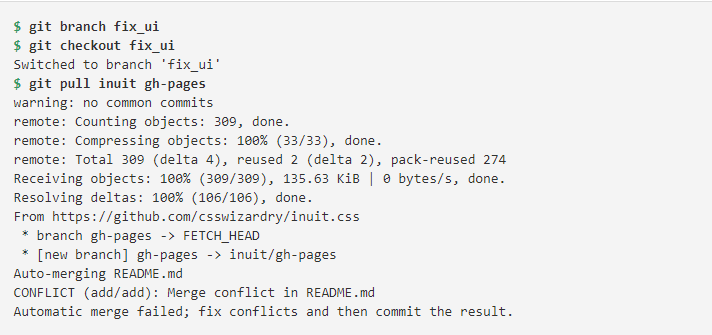
Quay lại với bài remote repository, bây giờ hãy tạo thêm một remote mới từ địa chỉ https://github.com/csswizardry/inuit.css và đặt tên cho remote này là inuit vào working tree.



Bây giờ ta có thể xem toàn bộ branch của cái remote inuit mới thêm vào bằng lệnh git remote show inuit.

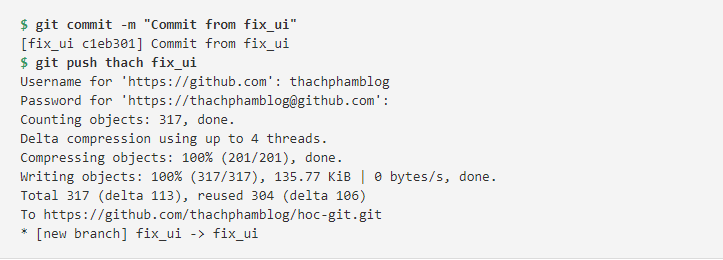


Bây giờ có thể chọn một cái remote branche cần fetch dữ liệu về. Ví dụ bây giờ tạo một branch mới cho working tree của mình tên làfix\_ui, sau đó nạp dữ liệu trong branch gh-pages của inuit thì mình sẽ lần lượt làm như sau.

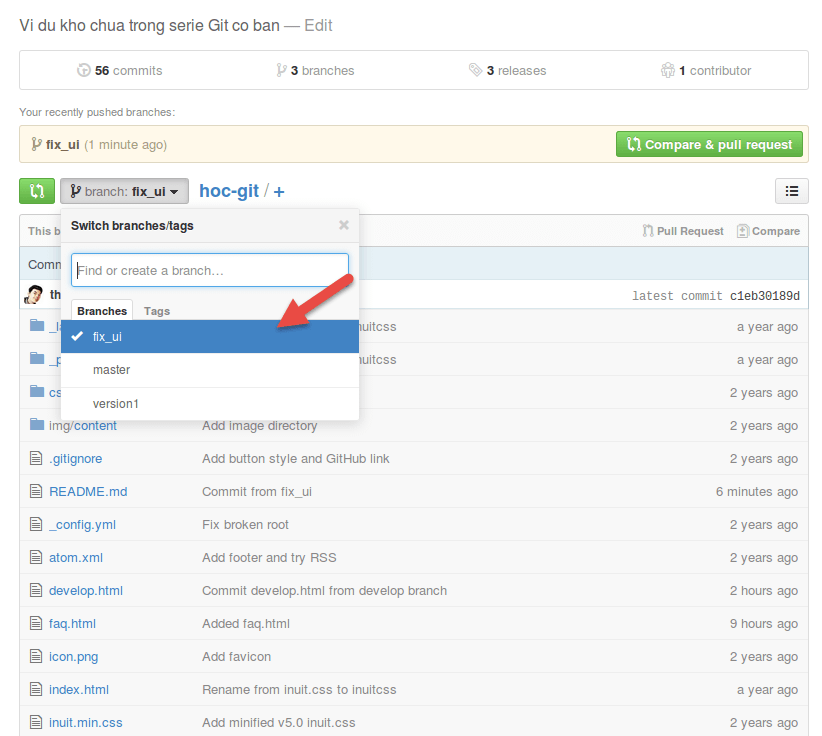


Có thể nó sẽ xảy ra lỗi conflict khi gộp file README.md cũng không quan trọng lắm nên có thể bỏ qua bằng cách gõ lệnh git add README.md để track file này, vì bây giờ gõ lệnh ls ra thì đã có các file từ inuit.

Bây giờ có thể commit nó và push nó lên repository:



Thach nghĩa là tên remote của repository mà mình cần push lên và fix\_ui là tên branch mình cần push. Và đây là kết quả sau khi push lên repository.



Chúng ta cũng có thể qua một branch khác và sử dụng lệnh git merge để gộp dữ liệu của branch này về.

Trên đây là hưỡng dẫn cài đặt và một số những hưỡng dẫn cơ bản của Git.

TÀI LIỆU THAM KHẢO