[CHƯƠNG 1: TỔNG QUAN 1](#_Toc22460254)

[1.1. Tổng quan 1](#_Toc22460255)

[1.1.1. Tổng quan về Git 1](#_Toc22460256)

[1.1.2. Tổng quan về Web Tin Tức Công Nghệ 1](#_Toc22460257)

[1.1.3. Tại sao phải quản lý mã nguồn 1](#_Toc22460258)

[1.2. Lý do chọn Git 1](#_Toc22460259)

[CHƯƠNG 2. CƠ SỞ LÝ THUYẾT 3](#_Toc22460260)

[2.1. Cơ sở lý thuyết 3](#_Toc22460261)

[2.1.1. Giới thiệu Git 3](#_Toc22460262)

[2.1.2. Giới thiệu Bitbucket 7](#_Toc22460263)

[2.2. Mô hình Phân nhánh 8](#_Toc22460264)

[2.2.1. Những nhánh chính 8](#_Toc22460265)

[2.2.2. Tạo release branch 13](#_Toc22460266)

[2.2.3. Kết thúc relase branch 14](#_Toc22460267)

[2.2.4. Tạo hotfix branch 16](#_Toc22460268)

[2.2.5. Kết thúc hotfix branch 16](#_Toc22460269)

[2.3. Danh sách các lệnh git sử dụng trong quá trình bảo trì 17](#_Toc22460270)

[2.3.1. git merge <branch> 17](#_Toc22460271)

[2.3.2. git pull origin <branch> 17](#_Toc22460272)

[2.3.3. git status git diff –stat 17](#_Toc22460273)

[2.3.4. git log 17](#_Toc22460274)

[2.3.5. git checkout c3d88eaa1aa4e4d5f 18](#_Toc22460275)

[2.4. Sử dụng phần mềm phpstrom quản lý souce code trên git và bảo trì dự án 18](#_Toc22460276)

[2.4.1. Giao diện chính của phần mềm 18](#_Toc22460277)

[2.4.2. Sử dụng PHPStorm trong bảo trì phần mềm 18](#_Toc22460278)

[CHƯƠNG 3: DEMO 21](#_Toc22460279)

[3.1. Giới thiệu dự án xây dựng website đọc tin tức công nghệ 21](#_Toc22460280)

[3.1.1. Tổng quan về dự án 21](#_Toc22460281)

[3.1.2. Các chức năng 21](#_Toc22460282)

[3.1.3. Đối tượng người dùng 21](#_Toc22460283)

[3.2. Kế hoạch phát hành các phiên bản 22](#_Toc22460284)

[3.2.1. Phát hành phiên bản 1.0 22](#_Toc22460285)

[3.2.2. Phát hành phiên bản 1.0.1 23](#_Toc22460286)

[3.2.3. Phát hành phiên bản 1.1 23](#_Toc22460287)

[3.2.4. Phát hành phiên bản 1.2 24](#_Toc22460288)

[3.2.5. Phát hành phiên bản 1.3 24](#_Toc22460289)

[3.2.6. Phát hành phiên bản 1.4 24](#_Toc22460290)

[3.2.7. Phát hành phiên bản 1.5 25](#_Toc22460291)

[3.2.8. Phát hành phiên bản 1.6 25](#_Toc22460292)

[3.2.9. Phát hành phiên bản 1.7 26](#_Toc22460293)

[3.2.10. Phát hành phiên bản 1.8 26](#_Toc22460294)

CHƯƠNG 1: TỔNG QUAN

* 1. Tổng quan
     1. Tổng quan về Git

Git là phần mềm quả lý mã nguồn phân tán (Distributed Version Control System). Được phát triển bởi Linus Torvalds và năm 2005. Bạn đầu dành cho việc phát triển nhân Linux. Hiện nay là một trong những hệ thống quản lý mã nguồn phân tán phổ biến nay.

* + 1. Tổng quan về Web Tin Tức Công Nghệ

Với thời đại công nghệ thông tin, mọi thông tin đều được đưa lên Internet. Việc muốn đưa thông tin nhanh chóng kịp thời tới người dùng thông qua Internet là việc thật sự cần thiết.

Nên chúng em đã xây dựng lên một trang web tin tức công nghệ. Nhằm đưa thông công nghệ mới nhất đến cho người dùng.

* + 1. Tại sao phải quản lý mã nguồn

Kiểm soát mã nguồn là một phần quan trong trong vòng đời phát triển phần mềm. Nó cho phép bạn theo dõi các thay đổi của mã nguồn và cũng như kiểm tra lịch sử thay đổi của mã nguồn hay khôi phục lại mã nguồn.

Với việc quản lý mã nguồn bạn có thế làm việc với nhóm hoặc cá nhân thực hiện thay đổi mà mà không bị cản trở bởi việc không đồng bộ mã nguồn.

Việc khắc phục mã nguồn cũng trở nên dễ dàng hơn khi chúng ta có thể xem lại toàn bộ các thay đổi và ai là người thay đổi.

Vì vậy việc quản lý mã nguồn hết sứ là cần thiết.

* 1. Lý do chọn Git

Với việc Công nghệ phát triển mạnh mẽ, ngày càng có nhiều công cụ quản lý mã nguồn phát triển ra đời như:

* CodePlex là một công cụ được phát triển bởi Microsoft.
* Beanstalk
* CodeBase
* Launchpad
* Git

Việc nhiều có nhiều công cụ quả lý mã nguồn như vậy, chọn được một công cụ quản lý mã nguồn là rất khó. Sau khi tìm hiểu chúng em quyết định chọn Git là công cụ quả lý mã nguồn cho dự án của mình.

Vì Git có các tính năng:

* Dễ dàng xem các thay đổi, các chú thích, các bác cáo các lỗi
* Dễ dàng làm việc với mọi người
* Có thể tích hợp cách công cụ khác
* Có một số công cụ kiểm tra project
* Có phiên bản dành cho Windows, Linux, Mac OS
* Miễn phí mọi thứ, không có private repositories

CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

1. 1. Cơ sở lý thuyết
      1. Giới thiệu Git

* Khái niệm Git

Git là tên gọi của một Hệ thống quản lý phiên bản phân tán (Distributed Version Control System – DVCS) là một trong những hệ thống quản lý phiên bản phân tán phổ biến nhất hiện nay.

DVCS nghĩa là hệ thống giúp mỗi máy tính có thể lưu trữ nhiều phiên bản khác nhau của một mã nguồn được nhân bản (clone) từ một kho chứa mã nguồn (repository), mỗi thay đổi vào mã nguồn trên máy tính sẽ có thể ủy thác (commit) rồi đưa lên máy chủ nơi đặt kho chứa chính.

Với 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.

* **Repository (kho chứa)**

Git sử dụng Repository (kho chứa) nghĩa là nơi mà bạn sẽ lưu trữ mã nguồn. Repository có hai loại là Local Repository (Kho chứa trên máy cá nhân) và Remote Repository (Kho chứa trên một máy chủ từ xa).

Lệnh cơ bản để tạo kho chứa:

* Ta sử dụng lệnh “**git init**” để tạo kho chứa dưới Local.
* **Branch (nhánh)**

Khi làm việc với git chúng ta luôn có một nhánh chính gọi là “master”.

Nó là nhánh chứa toàn bộ mã nguồn chính trong repository và nếu muốn chỉnh sửa mã nguồn mà không muốn làm thay đổi mã nguồn chúng ta chuyển sang một nhánh làm việc khác mà không làm ảnh hưởng đến mã nguồn hiện có trên nhánh master.

Các lệnh với Branch (nhánh):

* Để tạo một nhánh chúng ta sử dụng lệnh: “**git branch tên\_branch**”
* Để kiểm tra ta sử dụng lệnh “**git branch**”
* Để chuyển sanh nhánh khác ta sử dụng lệnh “**git checkout tên\_branch**”
* Ta có thể tạo một nhánh mới từ lệnh checkout bằng cách: “**git checkout -b tên\_branch**”
* Mỗi khi thực hiện lệnh checkout vào branch nào đó thì toàn bộ mã nguồn trên working tree sẽ được đổi sang môi trường dành cho branch đang checkout.
* Chúng ta có thể xóa nhánh đi được bằng lệnh “git branch -d tên\_branch” sau khi đã merge vào nhanh master.
* **Git Clone**

Lệnh này sẽ sao chép toàn bộ dữ liệu trên repository và sao chép luôn các thiết lập về repository, tức là nó sẽ tự động tạo một master branch trên máy tính của bạn. Lệnh này chỉ nên sử dụng khi bạn cần tạo mới một Git mới trên máy tính với toàn bộ dữ liệu và thiết lập của một remote repository.

Để clone một repository có sẵn ở trên máy cục bộ, bạn hãy sử dụng dòng lệnh “**git clone tên\_đường dẫn”**

Nếu repository đó ở máy chủ khác thì bạn hãy gõ dòng lệnh sau: “**git clone tênusername@địachỉmáychủ:/đường-dẫn-đến/repository”**

* **Git pull**

Lệnh này sẽ tự động lấy toàn bộ dữ liệu từ remote repository và gộp vào cái branch hiện tại bạn đang làm việc.

Lệnh pull “**git pull**”

* **Git push**

Lệnh này sẽ đẩy toàn bộ dữ liệu ở staging area lên repository.

Thực push “**git push origin tên\_nhánh**”

* **Git Add**

Để thêm tệp vào staging area ta dùng lệnh “**git add tên\_file**”

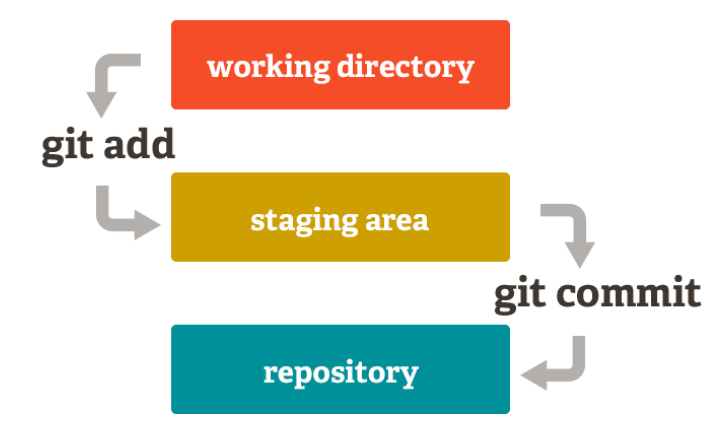
Ta có thể thêm tất cả các tệp bằng lệnh “**git add \*”**

* **Commit**

Git có 3 khu vực làm việc chính: Trên máy tính cá nhân, trên kho chứa mã nguồn(repository) và khu vực trung gian (staging area).

Staging area là khi vực sẽ lưu trữ những thay đổi của bạn trên tập tin để nó có thể commit được, vì muốn commit được tệp tin của bạn phải nằm trong Staging area.

Commit nghĩa là một hành động để Git lưu lại một bản chụp của các sự thay đổi trong thư mục làm việc, và các tệp thư mục được thay đổi đã phải nằm trong Staging area.



Hình 2.1. 1 Mô tả vùng Staging area

Mỗi lần commit nó sẽ lưu lại lịch sử chỉnh sửa của mã nguồn kèm theo tên và địa chỉ email của người commit.

Ngoài ra bạn có thể khôi phục lại tập tin trong lịch sử commit của nó để chia cho một branch khác, điều này sẽ dễ dàng cho khôi phục lại các thay đổi trước đó.

Lệnh commit trong Git sẽ là: “**git commit -m "Lời nhắn"**”

Tệp tin trong git sẽ có 2 trạng thái chính là: **Tracked** và **Untracked**.

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.

Có thể đưa tệp tin từ **Untracked** 🡪 **Tracked** bằng câu lệnh: “**git add tên\_file**”

Hoặc đưa tệp tin từ **Tracked** về **Untracked** bằng câu lệnh: “**rm tên\_file**”

* **Git push (đẩy các thay đổi)**

Push là lệnh

* **Git Log**

Để xem lịch sử của các lần commit trước đó, bạn sử dụng lệnh “**git log”**.

* **Git Tag**

Trong Git có hai kiểu tag chính đó là:

Lightweight Tag: Các tag này chỉ đơn thuần là đánh dấu snapshot của commit.

Annotated Tag: Với tag này, bạn có thể đặt tiêu đề cho tag, và khi xem nó sẽ có thông tin về người tag, ngày tag ,….

Để tạo Lightweight Tag ta dùng lệnh “**git tag tên\_tag**”

Để xem thông tin cua lần commit có gắn tag này ta dùng lệnh “**git show tên\_tag**”

Để tạo Annotated Tag ta dùng lệnh “**git tag –a tên\_tag –m "lời nhắn"**”

Ta muốn đẩy các tag lên repository ta dùng lệnh “**git push --tags**”

Nhập tag vào nhánh ta dùng lệnh “**git checkout -b tên\_branch tên\_tag**”.

* **Ưu điểm**
* Git giúp sắp xếp công việc tốt hơn. Nghĩa là chúng ta có thế tập trung giải quyết công việc của từng task mà không phải bận tâm lo lắng đến những task liên quan.
* Linh hoạt hơn khi làm việc cùng nhiều task.
* Git hoạt động nhanh chóng
* Nhanh chóng bắt đầu một kho lưu trữ mới
* Git sử dụng rất dễ dàng
* Sao lưu ẩn
* An toàn cao
* Phân nhánh dễ dàng
* Không yêu cầu một phần cứng mạnh
* **Nhược điểm**
* Mất thời gian tìm hiểu và nhớ cách dòng lệnh
* Thiếu tích hợp IDE và Explorer
* Giao diện chưa rõ ràng
  + 1. Giới thiệu Bitbucket
* **Khái niệm**

Bitbucket là dịch vụ lưu trữ nền web được dùng cho các mã nguồn lập trình và phát triển những dự án lập trình có sử dụng hệ thống kiểm soát Git revision.

Bitbucket thuộc sở hữu của Atlassian, Bitbucket cung cấp cả hai giải pháp tài khoản miễn phí và tài khoản trả phí.

Điểm nổi trội của Bitbucket so với các dịch vụ khác như Github ... là Bitbucket cho phép tạo không giới hạn dự án riêng tư (private repositories).

Bitbucket được tích hợp với rất nhiều dịch vụ khác của Atlassian như Jira, HipChat, Confluence và Bamboo.

* **Lịch sử ra đời**

Bitbucket trước đây là một dự án startup độc lập, được thành lập bởi Jesper Nøhr. Vào ngày 29 tháng 9 năm 2010, Bitassuck bị Atlassian mua lại sau đó.

* Những tính năng chính của Bitbucket
* Pull requests with code review and comments
* Bitbucket Pipelines
* 2 step verification and required two step verification
* IP whitelisting
* Merge Checks
* Code search (Alpha)
* Git Large File Storage (LFS)
* Documentation, including automatically rendered README files in a variety of Markdown-like file formats
* Issue tracking
* Wikis
* Static sites hosted on Bitbucket Cloud: Static websites have the bitbucket.io domain in their URL
* Add-ons and integrations
* REST APIs to build third party applications which can use any development language
* Snippets that allow developers to share code segments or files
* Smart Mirroring
  1. Mô hình Phân nhánh
     1. Những nhánh chính

Kho chứa sẽ có 2 nhánh chính hoạt động mãi mãi:

* Master
* Develop

Nhánh master tại origin là nhánh quen thuộc với tất cả người dùng Git. Song song là nhánh develop



Origin/master được coi là nhánh chính với HEAD phản ánh trạng thái production-ready.

Origin/develop được coi là nhánh chính với HEAD phản ảnh trạng thái thay đổi mới nhất trong quá trình phát triển, chuẩn bị cho release tiếp theo.

Khi source code bên develop đạt đến một mức độ ổn định nào đó và sẵn sàng để release thì sẽ được merge sang bên master và đánh dấu với release number.

Như vậy, theo định nghĩa về nhánh master, chúng ta mặc định hiểu rằng khi có thay đổi được merge và master thì tức là sẽ có một phiên bản production mới được release. Nhờ đó chúng ta có thể sử dụng script để tự động build lên production server mỗi khi có commit ở master.

* Những nhánh phụ

Bên cạnh hai nhánh chính master và develop, mô hình còn có thêm rất nhiều nhưng nhánh phụ khác để giúp các thành viên trong nhóm có thể phát triển song song, dễ dàng tracking theo features, chuẩn bị cho release hoặc fix nhanh các vấn đề production. Khác với hai nhánh chính kia, các nhánh phụ này chỉ tồn tại trong một khoảng thời gian ngắn rồi sẽ bị xóa đi.

* Feature branches
* Release branches
* Hotfix branches

Phía trên là các loại nhánh khác nhau hay được sử dụng. Mỗi loại nhánh đều có nhiệm vụ riêng, cách xử lý riêng.

Về mặt kỹ thuật, không có nhánh là đặc biệt so với các nhánh khác cả. Tất cả chỉ là các nhánh thông thường, chúng chỉ được phân loại bằng cách chúng ta sử dụng ra sao.

* Feature branches
* Tách từ: develop
* Merge vào: develop
* Naming convetion: tự do, ngoại trừ master, develop, release-\*, hotfix-\*



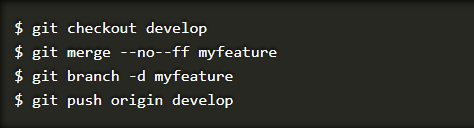
Feature branches được sử dụng để phát triển các feature mới phục vụ cho release sau này. Khi bắt đầu phát triển một chức năng, có thể chưa rõ được thời điểm chức năng đó tích hợp vào hệ thống và release. Feature branch sẽ tồn tại trong quá trình chức năng được phát triển, cuối cùng sẽ được merge lại vào develop (khi quyết định lần release tới bao gồm chức năng đó) hoặc bọ bỏ đi (khi thấy chức năng không cần thiết).

Về cơ bản thì feature branches chỉ tồn tại ở kho của developers chứ không phải ỏ orgin.

Tạo feature branch



Merge vào develop



--no—ff giúp thao tác merge luôn tạo ra một commit mới, ngay cả khi có thể merge theo fast-forward. Flag này giúp chúng ta không bị mất thông tin liên quan đến lịch sử commit của feature branch.



Ở trường hợp bên phải, không thể nhận biết được những commit nào phát triển cùng chức năng nếu không đọc log message của từng commit. Khi đấy, nếu muốn revert lại cả feature (phải revert nhiều commit liên quan) thì thực rất khó khăn và đó là lý do –no—ff được sử dụng.

Đương nhiên, nó tạo ra thêm vài commit, nhưng chả có vấn gì cả.

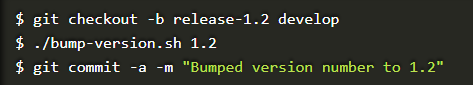
* Release branches
* Tách từ: develop
* Merge vào: develop và master
* Naming convention: release-\*

Release branches được sử dụng để chuẩn bị cho release bản production mới. Tất cả các công việc cuối cùng trước khi release sẽ được thực hiện ở đây, ngoài ra còn có thể fix nốt các bug lẻ tẻ, chuẩn bị meta-data (version number, build dates, etc…). Nhờ việc tách nhánh ra khỏi develop, chúng ta có thể tiếp tục phát triển các features cho đợt release khác một cách bình thường.

Thời điểm được lựa chọn để tách nhánh từ develop là khi develop phản ánh được trạng thái mong muốn cho việc release mới. Ít nhất lúc đó tất cả các features dành cho đợt release phải được merge vào develop rồi. Những features nhắm đến các lần release sau thì chưa được merge vào, phải đợi sau khi tách nhánh.

Chúng ta sẽ tiến hành đánh version theo rule của dự án ngay sau khi tạo release branch.

* + 1. Tạo release branch

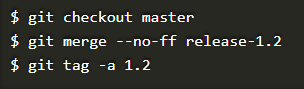


Ở ví dụ trên, bump-version tượng trưng cho một script thay đổi một vài file trong source code để phản ánh version mới. Sau khi tạo branch mới và chuyển sang branch đó, chúng ta thực hiện nâng version rồi commit thao tác đó.

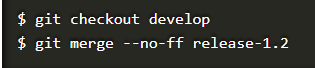
Branch mới này sẽ tồn tại cho đến khi việc release được thực hiện gọn ghẽ. Trong khoảng thời gian đó, có thể thực hiện fix bug ở branch này, tuy nhiên nghiêm cấm việc bổ sung feature mới lên đó. Tốt nhất nếu có feature mới thì hãy merge vào develop và đợi đợt release sau.

* + 1. Kết thúc relase branch

Khi source code trên release branch sẵn sàng để release, đầu tiền phải merge vào master, sau đó phải được merge lại vào develop để những lần release sau cũng chưa những thay đổi như lần này.



Vậy là source code đã được release lên master, và được tag để tiện sau này tham chiếu.



Bây giờ thì việc release đã hoàn thành và chúng ta không cần đến branch này nữa.



* Hotfix branches
* Tách từ: master
* Merge vào: develop và master
* Name convention: hotfix-\*

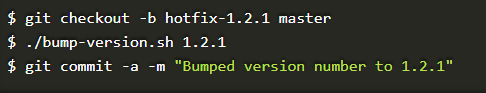


Hotfix branches cũng giống như release branches ở chỗ được sử dụng để chuẩn bị cho việc release production mới, chỉ khác ở chỗ là không có plan từ trước. Khi có một bug nghiêm trọng trên bản production cần được giải quyết ngay lập tức, một hotfix branch sẽ được tách ra từ master và được đánh version để nhận biết.

Ưu điểm của việc tách nhánh này ở chỗ các thanh viên trong nhóm khác có thể tiếp tục công việc ở develop trong khi những người khác có thể tập trung vào fix bug của production.

* + 1. Tạo hotfix branch

Hotfix branch được tạo ra từ master. Ví dụ, hiện tại version 1.2 là phiên bản của production đang chạy và xuất hiện lỗi nghiêm trọng. Tuy nhiên source code trên develop vẫn chưa ổn định, vì thế chúng ta phải tách nhánh hotfix và tiến hành sửa lỗi.



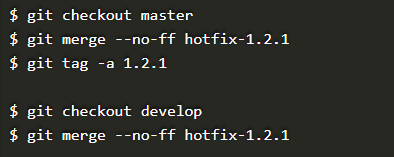
Sau khi tách nhánh phải tiến hành nâng version luôn.

Sau khi sửa lỗi, thực hiện commit.



* + 1. Kết thúc hotfix branch

Sau khi kết thúc sửa lỗi, những thay đổi phải được merge lại vào master, đồng thời cũng phải merge vào develop để ngăn lỗi xảy ra ở những lần release sau.



Tuy nhiên, có một điểm cần lưu ý rằng: khi đang tồn tại một release branch thì cần phải merge hotfix vào release branch đó, thay cho delvelop. Khi release branch được merge vào develop thì cuối cùng những thay đổi trong hotfix cũng được merge vào develop, nên không có vấn đề gì cả. Trừ khi thự sự công việc ở develop cần phần hotfix ngay lập tức và không thể đợi release branch được merge, thì cần phải cẩn thận merge hotfix vào develop.

Cuối cùng, không cần đến branch này nữa



* 1. Danh sách các lệnh git sử dụng trong quá trình bảo trì
     1. git merge <branch>

Nếu bạn đã thực hiện một loạt các thay đổi cho một branch, giả dụ tên branch là “develop”, và bạn muốn hợp nhất branch đó vào branch chính - master của bạn, bạn sử dụng câu lệnh git merge <tên branch>. Bạn sẽ checkout nhánh master, rồi chạy lệnh git merge develop để hợp nhất nhánh develop vào nhánh master.

* + 1. git pull origin <branch>

Nếu bạn làm việc với nhiều người, bạn sẽ gặp trường hợp trên GitHub đã được cập nhật rồi nhưng máy bạn lại không có những thay đổi đó. Khi đó, bạn có thể sử dụng lệnh git pull origin <tên branch> để kéo những thay đổi mới nhất về.

* + 1. git status git diff –stat

Nếu bạn muốn biết những file nào đã bị thay đổi và những file nào được theo dõi, bạn có thể sử dụng lệnh git status. Nếu bạn muốn biết những file đã được thay đổi *bao nhiêu*, bạn có thể sử dụng lệnh git diff để biết được mỗi file đã thay đổi bao nhiêu dòng code.

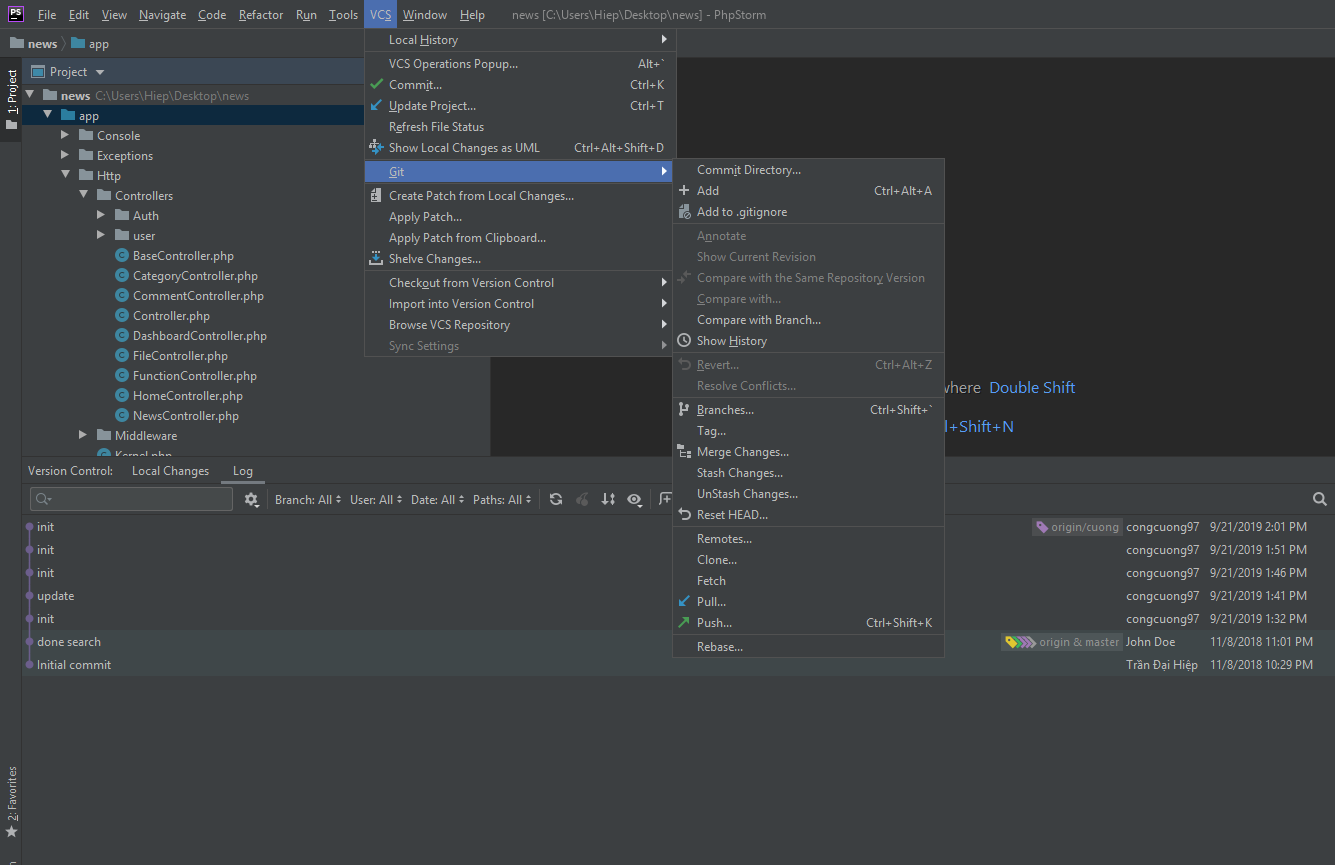
* + 1. git log

 Lệnh git log giúp bạn biết được lịch sử commit. Bạn sẽ muốn biết lịch sử commit của bạn. Các commit của bạn sẽ có một tin nhắn và một hash - dãy các số và chữ theo thứ tự ngẫu nhiên, ví dụ giống như thế này c3d882aa1aa4e3d5f18b3890132670fbeac912f7

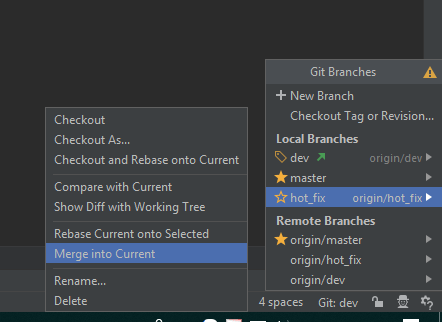
* + 1. git checkout c3d88eaa1aa4e4d5f

Giả dụ bạn push một commit phá app của bạn. Thay vì bạn ngồi fix bug thì bạn quay lại một commit trước và thử lại. Nếu bạn muốn quay ngược lại và checkout app của bạn từ các lần commit trước, bạn có thể làm nó trực tiếp bằng cách sử dụng dãy hash đó. Bằng cách này, app của bạn tách rời với phiên bản hiện tại (bởi bạn đang chỉnh sửa một bản ghi chép cũ, chứ không phải bản ghi chép hiện tại.

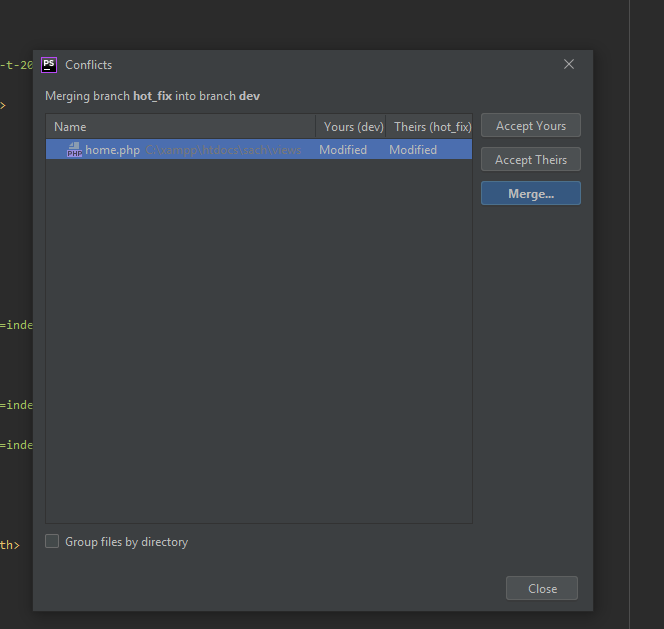
* 1. Sử dụng phần mềm phpstrom quản lý souce code trên git và bảo trì dự án
     1. Giao diện chính của phần mềm



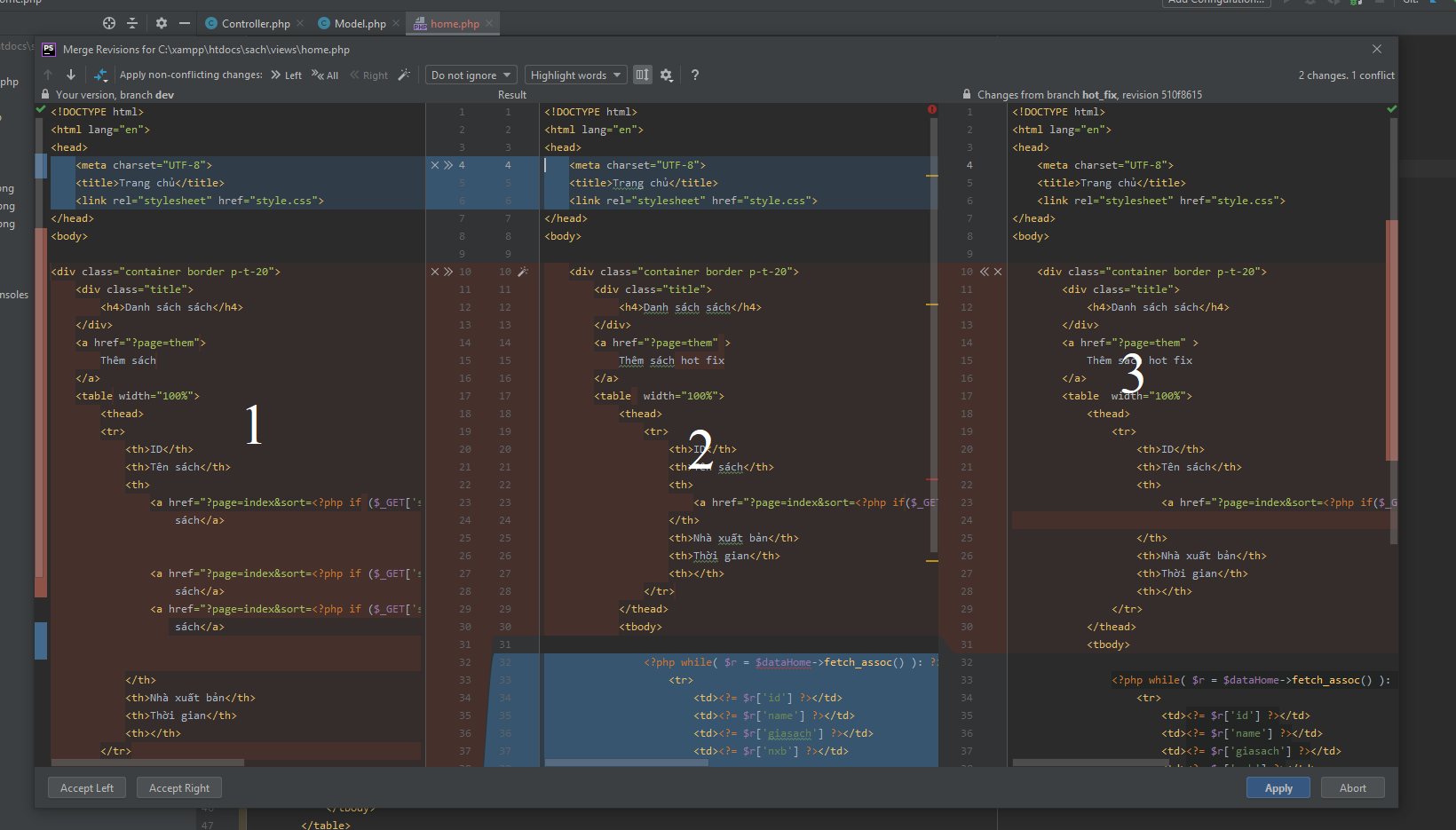
* + 1. Sử dụng PHPStorm trong bảo trì phần mềm
* Chọn nhánh cần gộp mà chọn merge into curent



* Phần mềm sẽ hiển thị các file có sự thay đổi



* Ấn vào file sẽ hiển thị các dòng thanh đổi



*Giao diện gộp code*

* Giao diện gộp code
* Phần số “1” phần phần code của nhánh “dev”
* Phần số “3” là phần của nhánh hot\_fix
* Phần số “2” sẽ là kết quả khi gộp code hoàn thành
* Trên phần mềm bạn có thể thấy “X” đại diện cho việc không gộp code ở nhánh đó, dấu “>>” ý nghĩa là gộp code 
* Sau khi xem xét code và gộp code hoàn tất bạn chọn “apply” để hoàn tất gộp file
* Phần mềm cũng hỗ trợ bạn chọn theo gộp theo bên trái hết hoặc theo nhánh bên phải hết

CHƯƠNG 3: DEMO

* 1. Giới thiệu dự án xây dựng website đọc tin tức công nghệ
     1. Tổng quan về dự án

Website tin tức công nghệ được xây dựng để phục vụ cho người dùng yêu thích và đam mê công nghệ ở nhiều lĩnh vực: thông tin, y tế, xã hội…

Website gồm 2 trang chính:

* Trang quản trị: giúp người quản trị có thể viết bài mới đưa lên website hay kiểm soát bình luận của người dùng…
* Trang người dùng: hiển thị tất cả các bài viết công nghệ để cho phép người dùng có thể đọc một cách dễ dàng.

Website được xây dựng dựa trên framework Laravel.

* + 1. Các chức năng
* Người dùng:
* Chức năng đọc tin tức.
* Chức năng tìm kiếm tin tức.
* Chức năng bình luận
* Người quản trị:
* Chức năng đăng nhập.
* Chức năng quản lý tin tức.
* Chức năng quản lý file.
* Chức năng quản lý bình luận.
  + 1. Đối tượng người dùng

Website được xây dựng phục vụ cho 2 đối tượng: người dùng truy cập vào website và người quản trị:

* Người dùng truy cập vào website là đối tượng chính mà đội ngũ phát triển cần hướng tới nhằm mục đích duy trì được sự tương tác của ngươi dùng đối với website.

Website có phát triển mạnh được hay không để nhờ vào sự hài lòng của người dùng truy cập.

* 1. Kế hoạch phát hành các phiên bản
     1. Phát hành phiên bản 1.0
* Chức năng cơ bản
* Khởi tạo dự án
* Phân chia route
* Tạo cơ sở dữ liệu
* Xây dựng giao diện html
* Lịch trình
* Release: 10/9/2019
* Nội dung chính
* Khởi tạo project laravel mới
* Thực hiện phân chia route cho người quản trị và người sử dụng
* Tạo migration trong laravel
* Khởi chạy migration để tạo ra databse
* Tạo các file view giao diện html cho trang người quản trị và người sử dụng
* Kế hoạch bảo trì và sửa mã
* Các chức năng trên được chia cho các thành viên trong team, khi các chức năng được xây dựng xong sẽ tiến hành merge code của các thành viên để tạo ra bản hoàn chỉnh.
* Thực hiện test các chức năng sau khi merge nếu có lỗi lập tức bảo trì sửa mã
* Sau khi các chức năng hoàn chỉnh sẽ tiến hành push lên nhánh chính master
  + 1. Phát hành phiên bản 1.0.1
* Chức năng cơ bản
* Tối ưu hóa giao diện
* Thực hiện sửa lỗi và cải thiện database
* Lịch trình
* Release: 12/9/2019
* Nội dung chính
* Chỉnh sửa lại một số trường dữ liệu trong file migration
* Phân tách giao diện ra các phần riêng: header, footer…
  + 1. Phát hành phiên bản 1.1
* Chức năng cơ bản
* Tạo file model.
* Thiết lập relationship giữa các model.
* Tạo modul đăng nhập auth.
* Lịch trình:
* Release: 13/9/2019
* Nội dung chính:
* Tạo các file model ứng với từng bảng trong database.
* Tạo relationship để truy xuất dữ liệu của các đối tượng mối quan hệ: 1 – n, n – 1, n – n.
* Chạy lệnh php artisan auth để tự động tạo ra modul đăng nhập.
* Tùy chỉnh thông tin đăng nhập là username và password.
  + 1. Phát hành phiên bản 1.2
* Chức năng cơ bản
* Xây dựng chức năng quản lý thể loại
* Lịch trình
* Release: 14/9/2019
* Nội dung chính
* Tạo controller category.
* Tạo giao diện quản lý thể loại.
* Xây dựng các hàm thực hiện các chức năng: thêm, sửa, xóa thể loại.
  + 1. Phát hành phiên bản 1.3
* Chức năng cơ bản
* Xây dựng chức năng quản lý bài viết
* Lịch trình
* Release: 15/9/2019
* Nội dung chính
* Tạo controller news.
* Tạo giao diện quản lý bài viết.
* Xây dựng các hàm thực hiện các chức năng: thêm, sửa, xóa bài viết.
  + 1. Phát hành phiên bản 1.4
* Chức năng cơ bản
* Xây dựng chức năng quản lý bình luận
* Lịch trình
* Release: 16/9/2019
* Nội dung chính
* Tạo controller comment.
* Tạo giao diện quản lý bình luận.
* Xây dựng các hàm thực hiện các chức năng: thêm, sửa, xóa bình luận.
  + 1. Phát hành phiên bản 1.5
* Chức năng cơ bản
* Xây dựng chức năng quản lý file
* Lịch trình
* Release: 17/9/2019
* Nội dung chính
* Tạo controller comment.
* Tạo giao diện quản lý bình luận.
* Xây dựng các hàm thực hiện các chức năng: thêm, sửa, xóa bình luận.
  + 1. Phát hành phiên bản 1.6
* Chức năng cơ bản
* Xây dựng chức năng hiển thị bài viết
* Lịch trình
* Release: 18/9/2019
* Nội dung chính
* Tạo controller home.
* Đổ dữ liệu ra trang chủ.
  + 1. Phát hành phiên bản 1.7
* Chức năng cơ bản
* Xây dựng chức năng tìm kiếm
* Lịch trình
* Release: 19/9/2019
* Nội dung chính
* Tạo controller search.
* Tạo giao diện hiển thị kết quả tìm kiếm
* Xây dựng các hàm thực hiện các chức năng tìm kiếm
  + 1. Phát hành phiên bản 1.8
* Chức năng cơ bản
* Xây dựng chức năng bình luận
* Lịch trình
* Release: 20/9/2019
* Nội dung chính
* Tạo controller comment.
* Xử lý sự kiến nút bình luận.
* Xây dựng các hàm thực hiện các chức năng bình luận bài viết