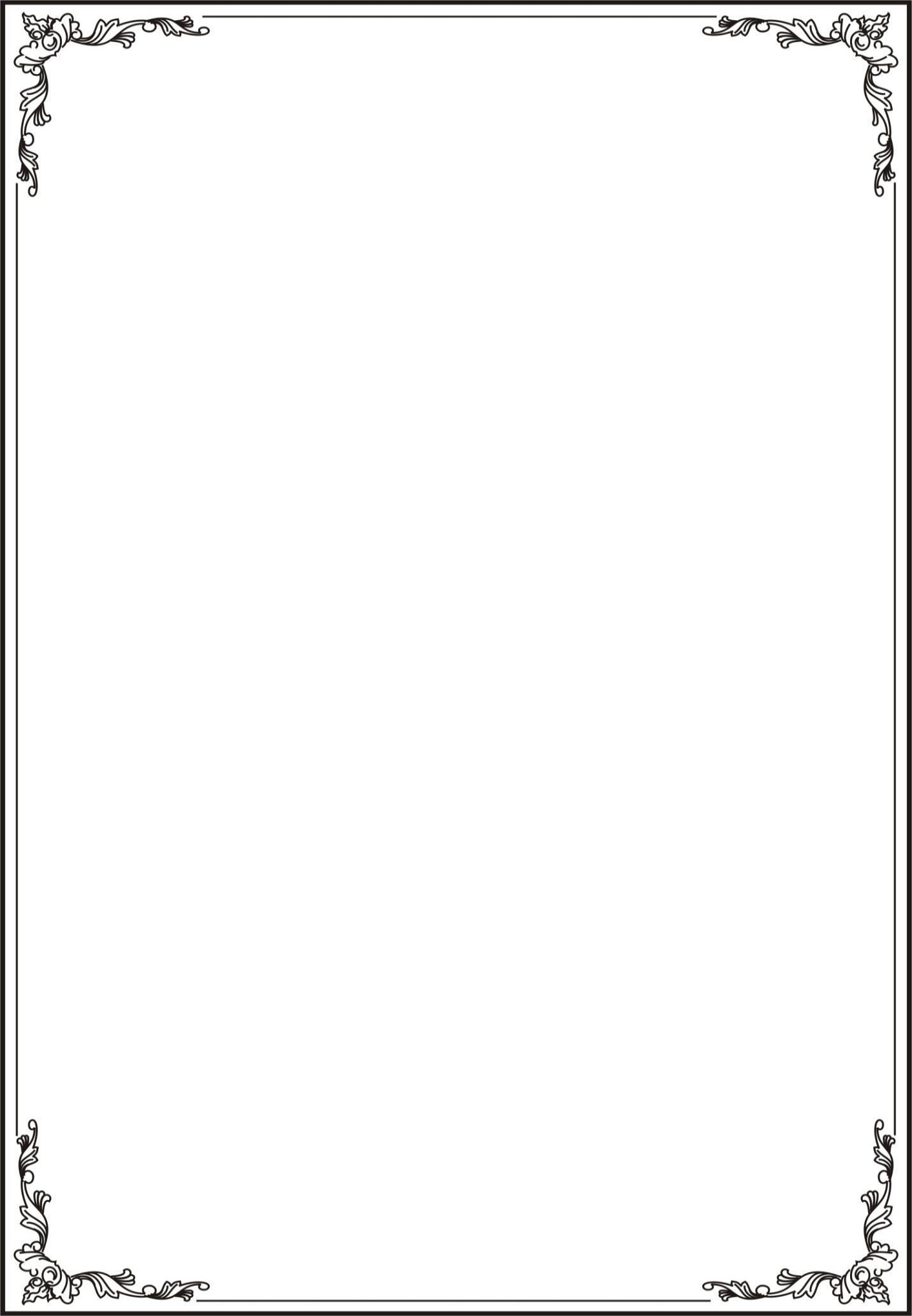
**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG**

**KHOA CÔNG NGHỆ THÔNG TIN**

\_\_\_\_\_\_\_\_\_\_\_\_\_🙠⬥🙢\_\_\_\_\_\_\_\_\_\_\_\_\_



**BÁO CÁO BÀI TẬP LỚN**

**ĐỀ TÀI:**

**Docker for DevOps and Developer**

Giảng viên hướng dẫn : **TS.Quách Xuân Trưởng**

**Sinh viên thực hiện: Nhóm 2**

***1. Nguyễn Duy Quang (nhóm trưởng)***

***2. Ngô Văn Trụ***

***3. Trần Như Quỳnh***

***4. Nông Quang Huy***

***5. Nguyễn Huy Hiệp***

**Lớp: *KTPM-K20E***

***Thái Nguyên ngày….tháng…., Năm 2024***

MỤC LỤC

[CHƯƠNG 1: TỔNG QUAN VỀ ĐỀ TÀI… 4](#_1fob9te)

[1.2 Lý do chọn đề tài 5](#_3znysh7)

[1.3 Phạm vi chủ đề 6](#_2et92p0)

[1.4 Mục tiêu đề tài 6](#_tyjcwt)

[1.5 Tổng quan về quy trinh DevOps 7](#_3dy6vkm)

[1.6 Tầm quan trọng … trong giai đoạn bảo trì phần mềm 8](#_1t3h5sf)

[CHƯƠNG 2 CÔNG CỤ ĐÓNG GÓI VÀ TRIỂN KHAI 9](#_4d34og8)

2.1. Đóng gói dự án

Các khái niệm cơ bản

Các công cụ đóng gói dự án

2.2. Triển khai dự án [9](#_2s8eyo1)

[Khái niệm cơ bản 9](#_17dp8vu)

Các công cụ quản lí dự án

[2.3 Tại sao dùng Docker cho Đóng gói và triển khai 13](#_2jxsxqh)

[17](#_1ci93xb)

[CHƯƠNG 3: TÌM HIỂU VỀ Docker 20](#_3whwml4)

[3.1 20](#_2bn6wsx)

[3.1.1 Sơ lược về Docker 20](#_qsh70q)

[3.1.2 Lịch sử Docker 21](#_3as4poj)

[3.1.3 Các khái niệm quan trọng của Docker 22](#_1pxezwc)

[3.1.4 Cách hoạt động của Docker 24](#_49x2ik5)

[3.1.5 Quy trình hệ thống dùng Docker 25](#_2p2csry)

[3.2 Công cụ kết hợp với Docker 26](#_147n2zr)

[3.2 Docker Compose 26](#_3o7alnk)

[3.3 Kubernetes 26](#_23ckvvd)

[3.2.2 Lịch Sử Git 26](#_ihv636)

[3.2.3 Đắc điểm của Git 27](#_32hioqz)

[3.2.4 Ưu nhược điểm của git 33](#_1hmsyys)

[3.3 Các khái niệm trong git và github 35](#_41mghml)

[3.4 Git Remote Server 38](#_2grqrue)

[3.4.1 Giao thức local 39](#_vx1227)

[3.4.2 Giao thức HTTP 40](#_3fwokq0)

[3.4.3 Giao thức SSH 41](#_1v1yuxt)

[3.4.4 Git Daemon 42](#_4f1mdlm)

[3.5 Cài đặt Git 43](#_2u6wntf)

[3.5.1 Trên Linux 43](#_19c6y18)

[3.5.2 Trên Windows 44](#_3tbugp1)

[3.5.3 Phiên bản GUI 51](#_28h4qwu)

[CHƯƠNG 4 TRIỂN KHAI DEMO 54](#_nmf14n)

[KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 64](#_37m2jsg)

[TÀI LIỆU THAM KHẢO 66](#_1mrcu09)

# CHƯƠNG 1: TỔNG QUAN VỀ ĐỀ TÀI

1.1 [Tầm quan trọng của Docker trong quy trình DevOps and Delevoper](#_46r0co2)

- Docker đóng vai trò quan trọng trong triển khai và phát hành dự án phần mềm. Dưới đây là một số điểm quan trọng về tầm quan trọng của Docker trong triển khai và phát hành:

+ Đóng gói chương trình với các phụ thuộc: Docker là công cụ cung cấp khả năng đóng gói và chạy chương trình dưới môi trường lỏng lẻo độc lập gọi là Container.

+ Phân phối chương trình nhanh và nhất quán: Docker cho phép nhà phát triển làm việc trong môi trường tiêu chuẩn hóa sử dụng container cục bộ, nó cung cấp các dịch vụ. Sử dụng container tốt cho quy trình làm việc CI/CD.

+ Thuận lợi triển khai và mở rộng quy mô: Nền tảng dựa trên container của Docker cho phép khối lượng công việc có tính linh động cao. Container của Docker có thể chạy trên laptop, máy vật lí hay máy ảo trên hệ thống dữ liệu, trên nhà cung cấp dịch vụ đám mây, hay trên hỗn hợp môi trường. Bản chất linh động và nhẹ nhàng của Docker cũng khiến nó trỏe nên dễ dàng năng động để quản lý khối lượng công việc, mở rộng hoặc loại bỏ các ứng dụng và dịch vụ theo nhu cầu kinh doanh, trong thời gian gần như thực.

+ Chạy nhiều khối lượng công việc hơn trên cùng phần cứng: Docker nhẹ mà nhanh. Nó cung cấp tính khả thi, giải pháp thay thế hiệu quả về mặt chi phí cho các máy ảo dựa trên bộ ảo hóa, nhờ đó bạn có thể sử dụng nhiều dung lnợn máy chủ hơn để đạt được mục tiêu kinh doanh của mình. Docker hoàn hảo cho môi trường mật độ cao và sự triển khai vừa và nhỏ khi bạn cần nhiều tài nguyên với ít tài nguyên hơn.

Why ?

## 1.2 Lý do chọn đề tài

- Việc chúng tôi chọn đề tài về “Docker cho vận hành - bảo trì và nhà phatr triển” có thể được lý giải bằng một số lý do sau:

* Khó khăn gặp phải tron quá trình triển khai:

+ Phổ biến và quan trọng: Docker phổ biến và quan trọng trong đội nhóm phát triển và kĩ sư DevOps. Nắm vững kiến thức về chúng là một yêu cầu cơ bản đối với bất kỳ nhà phát triển phần mềm nào.

+ Tính thực tiễn: Việc nghiên cứu và hiểu rõ về Docker mang lại lợi ích lớn cho việc phát triển phần mềm thực tế. Các kỹ thuật và công cụ liên quan đến Docker thường được áp dụng trong các dự án phần mềm thực tế, giúp tăng hiệu quả làm việc với khâu đóng gói và triển khai.

+ Cơ hội nghề nghiệp: Hiểu biết về Docker là một kỹ năng quan trọng cho các kĩ sư DevOps, cần thiết cho nhà phát triển phần mềm. Việc nắm vững kiến thức về chúng có thể tạo ra cơ hội nghề nghiệp tốt trong ngành công nghiệp phần mềm.

+ Hỗ trợ cộng đồng: Docker có một cộng đồng phát triển rất lớn và tích cực. Việc nghiên cứu và chia sẻ kiến thức về chúng có thể góp phần vào việc phát triển và thúc đẩy sự chia sẻ kiến thức trong cộng đồng.

+ Tính hệ thống và linh hoạt:. Việc hiểu biết về cách sử dụng chúng một cách hiệu quả có thể giúp tăng tính linh hoạt và hiệu quả của quá trình phát triển phần mềm.

## 1.3 Phạm vi chủ đề

- Giới thiệu và cơ bản về Docker: Điều này bao gồm giới thiệu về Docker là gì, lịch sử phát triển của nó, cách thức hoạt động, cùng với các khái niệm cơ bản như Dockerfile ,Image, Container và cách sử dụng GitHub để lưu trữ và quản lý mã nguồn.

- Cài đặt và cấu hình Docker: Hướng dẫn cài đặt Docker trên các hệ điều hành khác nhau và cách cấu hình Docker.

- Các lệnh cơ bản của Docker:

- Đóng gói với Docker: Hướng dẫn cách sử dụng file Dockercompose để tạo Image.

- Hợp tác và quản lý dự án trên DockerHub: Giới thiệu về các tính năng của DockerHub.

- Quản lý mã nguồn mở (Open Source) trên GitHub: Hướng dẫn về cách tham gia và đóng góp vào các dự án mã nguồn mở trên GitHub, cũng như việc sử dụng các tính năng của GitHub như Forks, Pull Requests, và Code Reviews.

- Tối ưu hóa quy trình làm việc với : Đề xuất các phương pháp và quy trình làm việc hiệu quả khi sử dụng Git và GitHub trong một dự án phần mềm.

- Tích hợp công cụ và dịch vụ với Git và GitHub: Hướng dẫn cách tích hợp Git và GitHub với các công cụ và dịch vụ khác như IDEs, CI/CD pipelines, và các dịch vụ lưu trữ cloud khác.

## 1.4 Mục tiêu đề tài

-  Hiểu biết sâu sắc về Docker: Mục tiêu chính là hiểu rõ về cách hoạt động của Docker, từ các khái niệm cơ bản đến các tính năng phức tạp như …

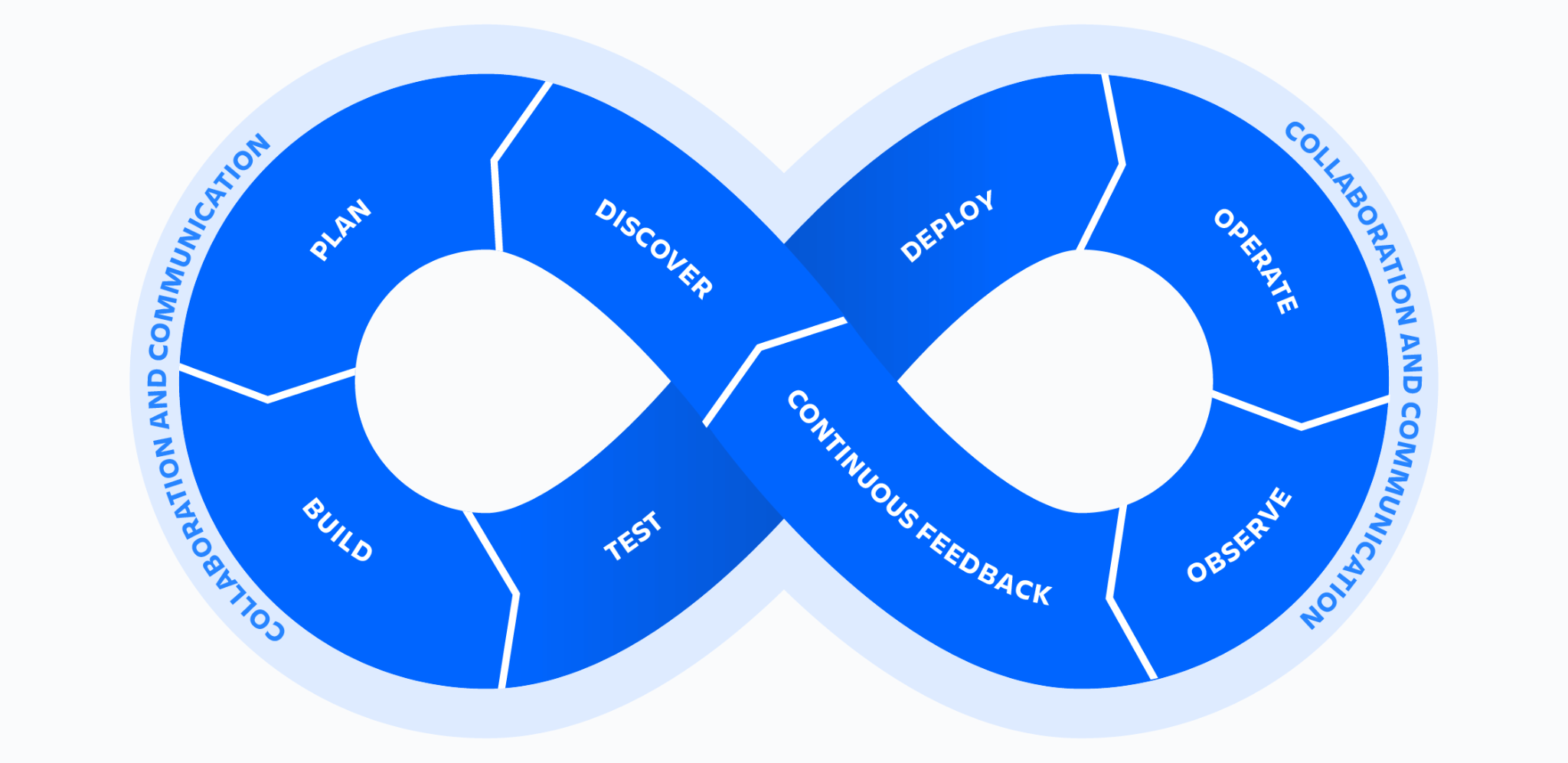
- Áp dụng Docker trong dự án thực tế: Mục tiêu này đặt ra là áp dụng kiến thức về Docker vào một dự án phần mềm thực tế, từ việc: … mới cho đến việc ….

- Tối ưu hóa quy trình làm việc: Mục tiêu này nhấn mạnh vào việc tìm hiểu và triển khai các quy trình làm việc hiệu quả nhất khi sử dụng Docker, bao gồm các best practices trong việc ….

- Thúc đẩy hợp tác và sự linh hoạt: Mục tiêu này là tạo điều kiện thuận lợi cho việc hợp tác và sự linh hoạt trong việc phát triển phần mềm, bằng cách sử dụng các tính năng….

- Nâng cao kỹ năng cá nhân và cộng đồng: Mục tiêu cuối cùng là nâng cao kỹ năng của các thành viên trong nhóm nghiên cứu về việc sử dụng Docker, và đóng góp vào việc chia sẻ kiến thức và kinh nghiệm với cộng đồng thông qua các dự án mã nguồn mở hoặc bài viết hướng dẫn.

## 1.5 Tổng quan về quy trinh DevOps



DevOps .. là tập hợp các phương pháp, công cụ và triết lý văn hóa mà tự động hóa và tích hợp các quy trình giữa nhóm phát triển phần mềm và nhóm vận hành. Nó nhấn mạnh việc trao quyền cho nhóm, giao tiếp và cộng tác giữa các nhóm cũng như tự động hóa công nghệ.

Vòng đời devOps: Vì tính chất liên tục của DevOps, Người thực hành sửa dụng vòng lặp vô cực để hiển thị các giai đoạn của vòng đời DevÓp liên quan với nhau ntn. Mặc dù có vẻ như diễn ra tuần tự nhưng vòng lặp tượng trưng cho nhu cầu hợp tác liên tục và cải tiến lặp đi lặp lại trong toàn bộ vòng đời:

Discover: Xây dựng phần mềm là một môn thể thao đồng đội. Để chuẩn bị cho sprint sắp tới, các nhóm phải hội thảo để khám phá, sắp xếp và ưu tiên các ý tưởng. Ý tưởng phải phù hợp với mục tiêu chiến lược và mang lại tác động cho khách hàng. Agile có thể giúp hướng dẫn các nhóm DevOps.

Plan: Các nhóm DevOps nên áp dụng các phương pháp thực hành linh hoạt để cải thiện tốc độ và chất lượng. Agile là một cách tiếp cận lặp đi lặp lại để quản lý dự án và phát triển phần mềm, giúp các nhóm chia công việc thành những phần nhỏ hơn để mang lại giá trị gia tăng.

Build: Git là một hệ thống kiểm soát phiên bản mã nguồn mở và miễn phí. Nó cung cấp hỗ trợ tuyệt vời cho việc phân nhánh, hợp nhất và viết lại lịch sử kho lưu trữ, dẫn đến nhiều quy trình làm việc và công cụ sáng tạo và mạnh mẽ cho quy trình xây dựng phát triển.

Test: Tích hợp liên tục (CI) cho phép nhiều nhà phát triển đóng góp vào một kho lưu trữ chung duy nhất. Khi các thay đổi mã được hợp nhất, các thử nghiệm tự động được chạy để đảm bảo tính chính xác trước khi tích hợp. Việc hợp nhất và thử nghiệm mã thường giúp các nhóm phát triển có được sự đảm bảo về chất lượng và khả năng dự đoán của mã sau khi triển khai.

Deploy: Triển khai Triển khai liên tục (CD) cho phép các nhóm phát hành các tính năng thường xuyên vào sản xuất theo cách tự động. Các nhóm cũng có tùy chọn triển khai với cờ tính năng, cung cấp mã mới cho người dùng một cách đều đặn và có phương pháp thay vì tất cả cùng một lúc. Cách tiếp cận này cải thiện tốc độ, năng suất và tính bền vững của các nhóm phát triển phần mềm.

Operate: Quản lý việc cung cấp dịch vụ CNTT từ đầu đến cuối cho khách hàng. Điều này bao gồm các hoạt động liên quan đến thiết kế, triển khai, cấu hình, triển khai và bảo trì tất cả cơ sở hạ tầng CNTT hỗ trợ các dịch vụ của tổ chức.

Observe: Nhanh chóng xác định và giải quyết các vấn đề ảnh hưởng đến thời gian hoạt động, tốc độ và chức năng của sản phẩm. Tự động thông báo cho nhóm của bạn về những thay đổi, hành động có rủi ro cao hoặc lỗi để bạn có thể tiếp tục duy trì dịch vụ.

Continuous feedback: Các nhóm DevOps nên đánh giá từng bản phát hành và tạo báo cáo để cải thiện các bản phát hành trong tương lai. Bằng cách thu thập phản hồi liên tục, các nhóm có thể cải thiện quy trình của mình và kết hợp phản hồi của khách hàng để cải thiện bản phát hành tiếp theo.

## 1.6 Tầm quan trọng của giai đoạn triển khai trong DevOps

Trong giai đoạn DevOps, triển khai là rất quan trọng vì các lý do sau:

* cho phép các nhóm nhanh chóng cung cấp các tính năng và cải tiến mới cho người dùng, giảm thời gian đưa sản phẩm và bản cập nhật mới ra thị trường;
* cho phép các nhóm nhận phản hồi từ người dùng và các bên liên quan nhanh hơn, cho phép họ thực hiện các điều chỉnh cần thiết cho phần mềm và cải thiện chất lượng của phần mềm;
* thường có phạm vi nhỏ hơn, giảm nguy cơ đưa các lỗi hoặc sự cố lớn vào phần mềm. Điều này cho phép các nhóm phát hiện và giải quyết các vấn đề sớm hơn trong quá trình phát triển, trước khi chúng trở thành các vấn đề lớn hơn;
* yêu cầu sự hợp tác chặt chẽ giữa các nhóm phát triển, thử nghiệm và vận hành, thúc đẩy giao tiếp và hợp tác tốt hơn giữa các phòng ban;
* tạo ra văn hóa cải tiến liên tục, khuyến khích các nhóm liên tục lặp lại và cải thiện phần mềm dựa trên phản hồi từ người dùng và các bên liên quan.

1.7 Các khó khăn trong giai đoạn đóng gói và triển khai

Cài đặt trên máy chủ, m áy khách

Cài đặt trên các h ệ điều hành khác

* Lựa chọn dịch vụ triển khai
* Cấu hình: Thiết lập sản phẩm mới đòi hỏi phải cấu hình các thông số cẩn thận để phù hợp với các yêu cầu của sản phẩm.
* Cài đặt và liên k ết giữa các thành phần

# CHƯƠNG 2: CÔNG CỤ ĐÓNG GÓI VÀ TRIỂN KHAI CHƯƠNG TRÌNH

## 2.1. Đóng gói dự án

Khái niệm cơ bản về đóng gói ( các phụ thuộc, các phiên bản, các lỗi khi đóng gói dự án,..)

Công cụ đóng gói dự án (...

### 2.2. Triển khai dự án

Khái niệm cơ bản (ác lỗi gặp phải khi triển khai, công cụ triển khai, lưu trữ,... )

Công cụ triển khai dự án(...

## 2.4 Lý do chọn Docker tròn giai đoạn đóng gói và triển khai

# CHƯƠNG 3: TÌM HIỂU VỀ Docker

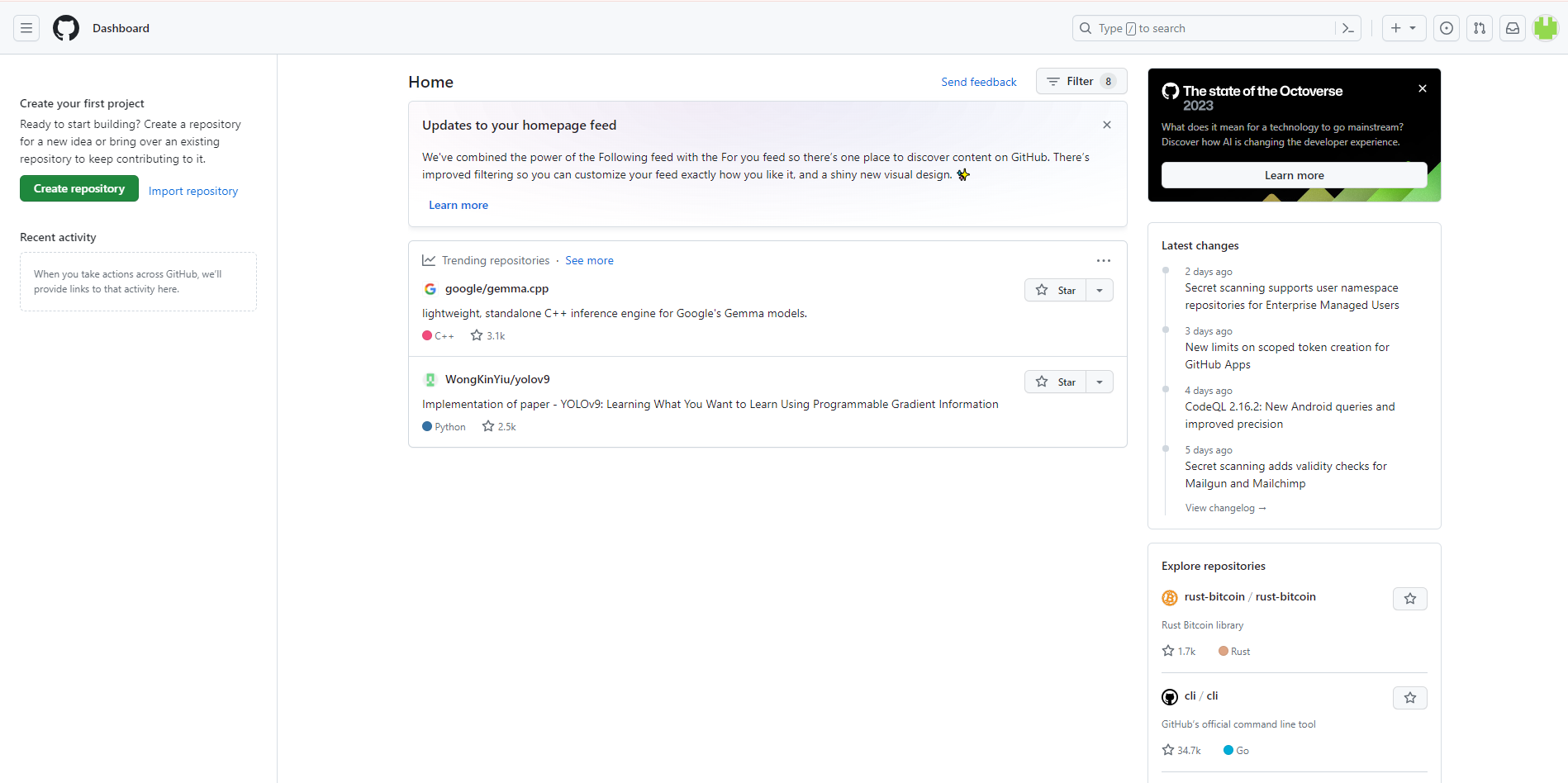
## 3.1

### 3.1.1 Sơ lược về Docker

Docker là nền tảng cung cấp cho các công cụ, service để các developers, adminsystems có thể phát triển, thực thi, chạy các ứng dụng với containers. Hay nói một cách khác nó là một nền tảng để cung cấp cách để building, deploy và run các ứng dụng một cách dễ dàng trên nền tảng ảo hóa - "Build once, run anywhere". Hay nói một cách dễ hiểu như sau: Khi chúng ta muốn chạy app thì chúng ta phải thiết lập môi trường chạy cho nó. Thay vì chúng ta sẽ đi cài môi trường chạy cho nó thì chúng ta sẽ chạy docker.

Ứng dụng Docker chạy trong vùng chứa (container) có thể được sử dụng trên bất kỳ hệ thống nào: máy tính xách tay của nhà phát triển, hệ thống trên cơ sở hoặc trong hệ thống đám mây. Và là một công cụ tạo môi trường được "đóng gói" (còn gọi là Container) trên máy tính mà không làm tác động tới môi trường hiện tại của máy, môi trường trong Docker sẽ chạy độc lập.

Docker có thể làm việc trên nhiều nền tảng như Linux, Microsoft Windows và Apple OS X



### 3.1.2 Lịch sử Docker

**--Lich sử docker**

**Docker là một nền tảng mã nguồn mở được phát triển để tự động hóa việc triển khai ứng dụng trong các container. Dưới đây là một cái nhìn tổng quan về lịch sử của Docker:**

**1. Khởi nguồn (2010-2013)**

* **2010: Solomon Hykes bắt đầu phát triển một dự án có tên "dotCloud", một dịch vụ Platform as a Service (PaaS). Đây là nền tảng gốc của Docker.**
* **2013: Docker chính thức được ra mắt vào tháng 3. Nó được công bố tại hội nghị PyCon và nhanh chóng thu hút sự chú ý từ cộng đồng lập trình viên.**

**2. Phát triển nhanh chóng (2013-2014)**

* **Docker trở thành một dự án mã nguồn mở, thu hút sự đóng góp từ nhiều lập trình viên.**
* **Các tính năng như Docker Hub (nơi lưu trữ và chia sẻ các container) được giới thiệu.**
* **Docker đã chứng minh được sự hiệu quả trong việc triển khai, mở rộng và quản lý ứng dụng.**

**3. Tăng trưởng và phổ biến (2014-2015)**

* **Docker trở thành một phần quan trọng trong DevOps và kiến trúc microservices.**
* **Nhiều công ty lớn bắt đầu sử dụng Docker để cải thiện quy trình phát triển và triển khai ứng dụng.**
* **Docker Inc. được thành lập để phát triển và hỗ trợ Docker.**

**4. Tiến hóa và cộng đồng (2016-2019)**

* **Các công cụ như Docker Compose, Docker Swarm và Kubernetes (mặc dù không phải của Docker) bắt đầu phát triển mạnh mẽ để quản lý container.**
* **Docker tiếp tục cập nhật và cải tiến, với các phiên bản mới mang lại nhiều tính năng và cải thiện hiệu suất**

****

### 3.1.3 Các khái niệm quan trọng của Docker

Docker là một nền tảng mã nguồn mở giúp phát triển, vận chuyển và chạy ứng dụng trong các container. Dưới đây là một số khái niệm quan trọng của Docker:

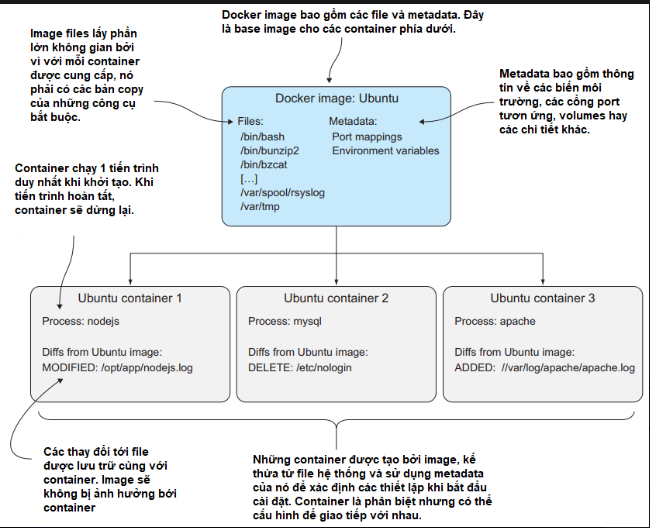
### 1. Container

Container là một đơn vị tiêu chuẩn hóa mà trong đó các ứng dụng và các phụ thuộc của chúng được đóng gói. Container chạy trên cùng một hệ điều hành nhưng tách biệt về môi trường, giúp ứng dụng chạy nhất quán trên nhiều môi trường khác nhau.

### 2. Image

Image là một mẫu (template) không thay đổi dùng để tạo ra container. Mỗi image chứa tất cả các tệp cần thiết để chạy ứng dụng, bao gồm mã nguồn, thư viện và các tệp cấu hình.

Nãy giờ chúng ta cứ đá qua đá lại 2 khái niệm này mà chưa hiểu rõ lắm, giờ chúng ta sẽ tìm hiểu qua chút về nó.

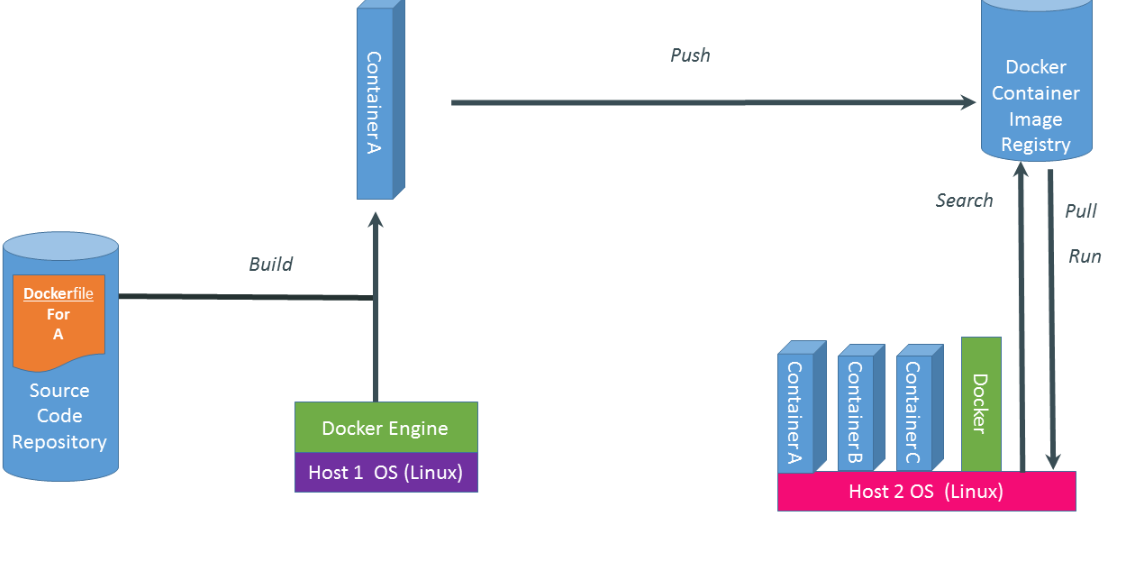


Hình vẽ giải thích khá rõ về Image và Container.

Nếu bạn đã quen với hướng đối tượng, chúng ta có thể hiểu images như những class còn cointaner là những object. Tương tự như việc các object được khởi tạo từ những class, container cũng được khởi tạo từ image. Bạn có thể tạo ra nhiều container từ 1 image, và chúng đều bị cô lập như cách các object khởi tạo. Bất kỳ thay đổi nào trên object, nó sẽ không ảnh hưởng đến định nghĩa và function của class.

### 3.1.4 Cách hoạt động của Docker

Docker sử dụng kiến ​​trúc client-server. Docker client sẽ liên lạc với các Docker daemon, các Docker daemon sẽ thực hiện các tác vụ build, run và phân phối các Docker container. Cả Docker client và Docker daemon đều có thể chạy trên cùng 1 máy hoặc có thể kết nối loại Dockẻ client điều khiển các docker daemon như hình dưới đây. Docker client và daemon giao tiếp với nhau thông qua socker hoặc RESTful API.



### a, Xây dựng.

Đầu tiên chúng ta sẽ tạo một dockerfile, trong dockerfile này chính là mã của chúng ta.

Dockerfile này sẽ được Build tại một máy tính đã cài đặt Docker Engine.

Sau khi xây dựng ta sẽ thu được Container, trong Container này chứa thư viện và ứng dụng của chúng ta.

### b, Đẩy.

Sau khi có được Container, chúng tôi thực hiện đẩy Container này lên đám mây và lưu trữ ở đó.

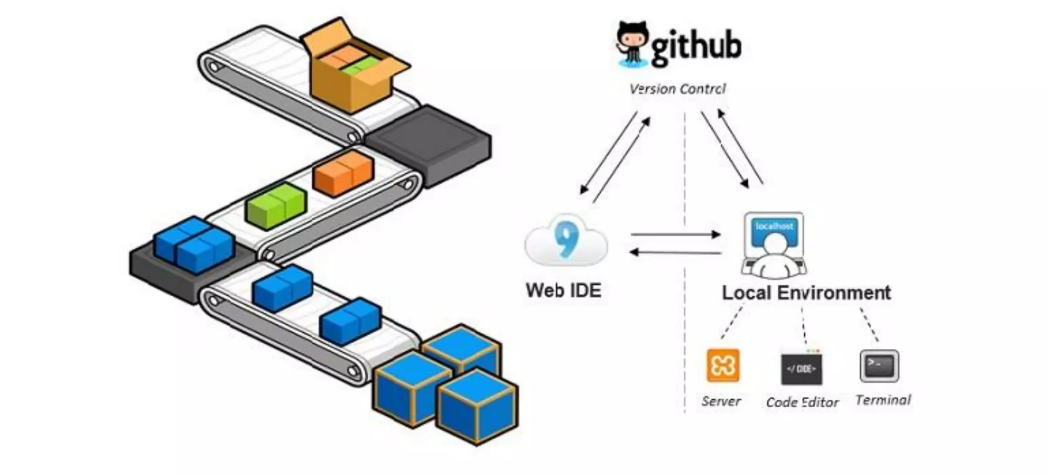
Việc đẩy này có thể thực thi qua môi trường mạng Internet.

### c, Kéo, Chạy

Giả sử một máy tính muốn sử dụng Container mà chúng tôi đã đẩy lên đám mây (máy đã cài đặt Docker Engine) thì bắt buộc máy phải thực hiện Kéo container này về máy. Sau đó thực hiện Run Container này.

Đó chính là quy trình 3 bước mô tả hoạt động của một hệ thống sử dụng Docker. Rất đơn giản và rõ ràng.

### 3.1.5 Quy trình hệ thống dùng Docker



Quy tắc làm việc trên Github diễn ra trên hai nền tâng đó là **Local Workflow và Server Workflow**. Người dùng thay đổi Source Code ở Local rồi xác nhận sự thay đổi đó ở Server. Bản xác nhận phía Server phải là một phiên bản có chứa tính năng hoàn chinh hoặc có thể chạy được. Việc xác nhận các đoạn Code chưa hoàn chinh sẽ làm anh hưong thành viên khác khi họ sử dụng chung kho lưu trữ.

Từ kho luu trữ này, người dùng có thể tạo các bản Build cho dự án gốc bằng cách gui các Source Code có sự thay đổi lên đó. Với vấn đề bảo mật và kiểm soát quyèn truy cập, mỗi người dùng khi sử dụng kho lưu trữ của Github phải cung cấp mã chứng nhận, có thể sử dụng một cặp khóa trong giao thức SSH, hệ thống sẽ so sánh SSH Key o Local và trên Server tương ứng với Account nào đã được chính nguời dùng đó đăng kí.

#### 3.1.5.1 Thao tác với Repo ở Local

Các thao tác dòng lệnh diễn ra ở Local hầu như đều là các dòng lệnh cơ bản của Git, trong đó, hai dòng lệnh nổi trội thường được nhiều đối tượng người dùng sử dụng là git add và git commit:

**-git add**: thêm tập tin đã được thay đỗi vào Stage

**-git commit**: các tập tin trong Stage sẽ được đưa qua Repo của Local

#### 3.1.5.2 Thao tác với Repo ở Server

Khi nguời dùng hoàn tất tất cả quy trình, tạo ra một phiên bản ổn định của dụ án ở Local, họ có thể quyết định cập nhật tập tin dự án đó lên Repo tại Server vói ba câu lenh sau là chu yếu:

**- push**: push các thay đổi từ Repo Local đén Repo Server

**- fetch**: cập nhật thay đổi từ Repo Server vè Repo Local

**- pull/rebase**: sao chép Source Code từr Server vè Local Workspace

## 3.2 Công cụ kết hợp với Docker

### 3.2.1 Docker Compose

3.2.2 DockerHub.

### 3. 2.3 Kubernetes

3.2.4 Jenkins

3.3. Các câu lệnh của Docker

Git là một đại diện tiêu biểu của hệ thống quản lý phiên bản phân tán DVCS mã nguồn mở, một trong những dạng hệ thống quản lý phiên bản phân tán phổ biến nhất hiện nay

Git có những đặc điểm sau:

+ Có hướng tiếp cận mới so với các hệ thống Source Control khác như SVN hay CVS truyển thống

+ Có nhiệm vụ theo dõi những thay đổi, chỉnh sửa trong Source Code của người dùng vào mọi thời điểm và đồng bộ những Source Code do họ chỉnh sửa lên Server cùng đồng nghiệp.

Ngoài ra, Git chạy được trên nhiều hệ điều hành khác nhau như Linux, Windows, MacOS...

### 3.2.2 Lịch Sử Git

Được xem là một sáng kiến hay và ra đời trong thời điểm xảy ra tranh chấp, git ra đời để thay thế **BitKeeper** vốn được dùng để chứa hàng ngàn dòng Code, bản vá và các tập tin của nhân Linux. Do mối quan hệ giữa nhóm lập trình viên phát triển nhân linux và công ty thương mại phát triển BitKeeper bị đổ bể dẫn tới bản thân BitKeeper bị thu hồi. Điều này dẫn tới cộng đồng phát triển nhân linux, bao gồm cả người tạo ra nó là **Linus Torvards**, tự phát triển một công cụ dựa trên những kiến thức được tiếp thu từ việc sử dụng BitKeeper. Git ra đời cùng với những yêu cầu được đề ra như sau:

-Tốc độ

-Thiết kế đơn giản

-Phân tán toàn diện

-Phù hợp cho cả dự án lớn và nhỏ

-Hỗ trợ mạnh mẽ cho hướng phát triển không tuyến tính

Git sau đó phát triển mạnh mẽ, ảnh hưởng đến nhiều dự án lớn, và có cả một nhánh dành cho phát triển phi tuyến tính. Nhóm em xin tóm tắt quá trình hình thành Git qua các ý sau:

- Git ra đời vào năm 2005

- Tác giả là **Linus Torvald**, đồng viết ra Linux kernel

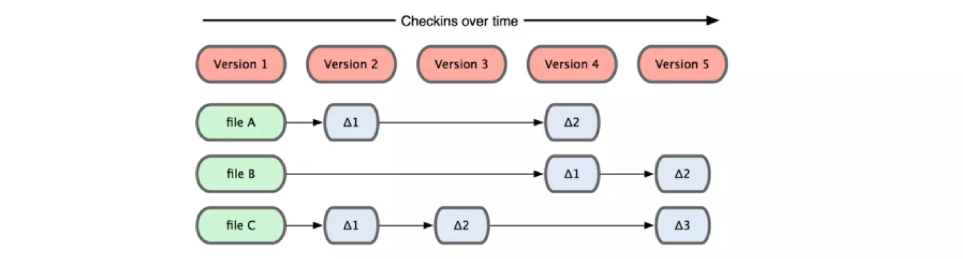
-Git vốn được phát triển để quản lý Source Code của Linux

-Git trở thành phần mềm quản lý mã nguồn mở phổ biến và được phân phối theo giấy phép công cộng GPL2

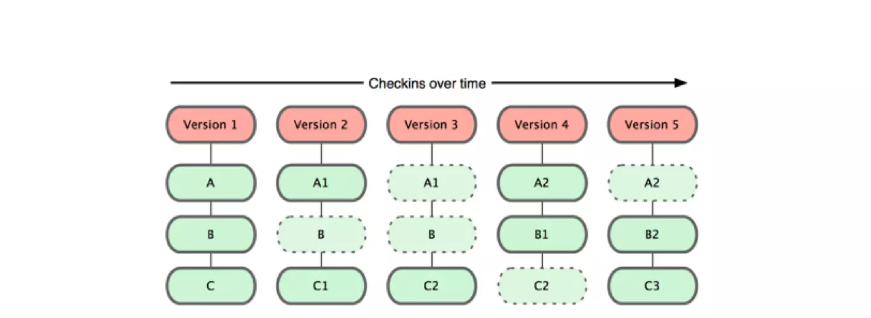
### 3.2.3 Đắc điểm của Git

#### 3.2.3.1 Tương tác dữ liệu với Git

Đối với các VCS còn lại, git có hướng suy nghĩ khác biệt về dữ liệu. Hầu hết các hệ thống sẽ chứa thông tin như một danh sách các file được thay đổi, chỉ khi nào có một thao tác lưu lại “commit” thì chỉ duy nhất file thực thi lệnh đó mới được lưu trên server. Người dùng sẽ không thể thực hiện commit toàn bộ phiên bản trạng thái hiện tại, hệ thống sẽ ánh xạ đến lần lưu trữ file trước.

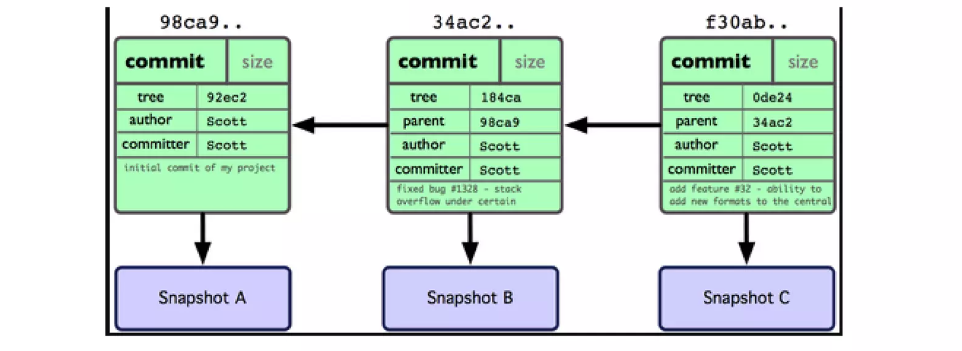


Git xem dữ liệu như là một chuỗi **“snapshot”** của một hệ thống tập tin thu nhỏ. Mỗi lần người dùng **"commit"** hay lưu trạng thái của dự án, git sẽ chụp ảnh hết tất cả trạng thái phiên bản tập tin hiện tại và chứa nó như một tham chiếu đến snapshot đóKhi file không thay đổi, Git sẽ tham chiếu đến lần lưu thông tin trước đó



#### 3.2.3.2 Cơ chế Snapshot của Git

Git lưu dữ liệu dưới dạng một ảnh chụp “Snapshot”, mỗi khi người dùng tiến hành lưu lại “commit” thì Git chụp lại hệ thống các file đó và lưu giữ một tham chiếu đến nó chỉ khi file có thay đổi, nếu không Git sẽ tham chiếu đến lần lưu trước đó.



#### 3.2.3.3 Thao tác Local của Git

Mọi thao tác như thêm mới, xem lại lịch sử... với Git đều diễn ra tại cục bộ do mỗi máy trạm đều có một cơ sở dữ liệu Git. Ngoài ra, người dùng muốn đưa file dữ liệu lên một điểm lưu trữ khác hoặc muốn lấy file cập nhật của người khác thì lúc đó mới cần đến vai trò của server.

Điều này giúp Git cải thiện tốc độ với việc sử dụng tài nguyên không đáng kể, bởi vì người dùng đã có toàn bộ dự án tại repo cục bộ của họ.

Nó có ích cho người dùng khi họ Offline hay Off VPN, họ vẫn làm việc thoải mái và chỉ cần đợi trong điều kiện có mạng có thể up file lên. So với các hệ thống tập trung, họ sẽ không thể **commit, push** hay làm gì khi không kết nối tới cơ sở dữ liệu của server

#### 3.2.3.4 Định dạng đối tượng trong Git

Git có một mục **object** được lưu trữ dùng để chứa file gốc và tất cả những tin nhắn log, chủ nhân file, ngày tạo và thông tin khác cần để tạo nên một sửa đổi hay nhánh cho dự án. Có 3 thành phần quan trọng của **object** cần chú ý: type, size, content.

**-Size:** đơn giản là kích cỡ của nội dung

**-Content**: dựa vào type (loại) của đối tượng

**-Type:** loại đối tượng được chứa trong đường dẫn .git/objects, gồm 4 loại khác nhau, bao gồm:

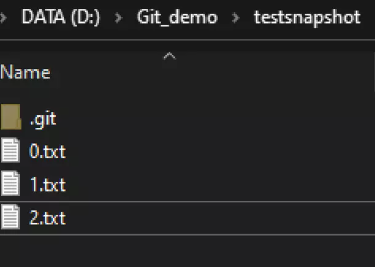
**+ blob**: dùng để lưu file dữ liệu – là tập tin nhị phân của mỗi phiên bản file, đây còn là nguyên nhân dẫn đến sự gia tăng dung lượng kho chứa

**+ tree**: đơn giản là một đường dẫn

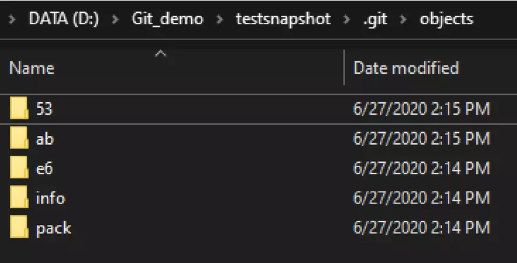
**+ commit**: giữ metadata cho mỗi lần file thay đổi trong kho lưu trữ, bao gồm chủ sở hữu, người gửi, dữ liệu commit, thông điệp nhật ký

**+ tag**: gán một tên có ý nghĩa mà người dùng đọc được cho một đối tượng cụ thể khi sử dụng commit

Có một khái niệm liên quan đến cơ chế **Snapshot** của Git là **loose object format** (định dạng đối tượng trong suốt của git). Để thấy được thư mục đối tượng, người quản trị tạo 3 file lần lượt là 0.txt, 1.txt, 2.txt và sử dụng các lệnh commit, trong folder sẽ xuất hiện file **.git.**

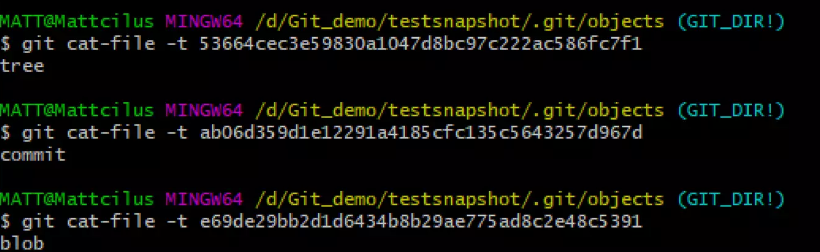


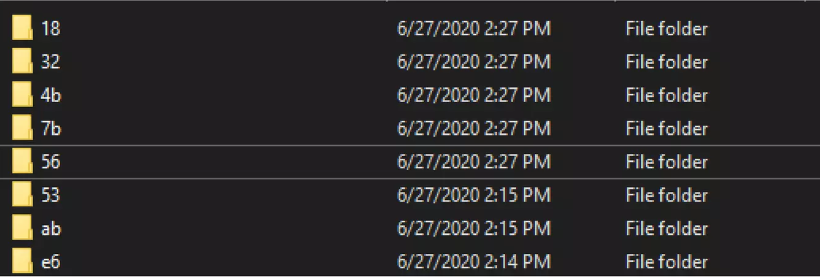
Git tạo 3 thư muck trong đường dẫn .git/objects lần lượt là 53,ab,e6



Trong mỗi thư mục sẽ có chứa một file mã băm gồm 40 kí tự với 2 kí tự là tên gọi của thự mục đường dẫn, 38 ki tự còn lại được dùng như tên file, người dùng gõ lệnh **"git cat-file [-option] [ten\_file]"**, option được chọn ở đây là **[-t]**

\*Lưu ý: ten\_file phải bao gồm đủ 40 kí tự, tức là bao gồm cả 2 kí tự tên thư mục đường dẫn

Chỉnh sửa nội dung tệp tin,Git sẽ tạo thêm 5 tập tin trên những tập tin cũ

Thông qua câu lệnh **git cat-file [-t]**, các thư mục băm với 2 kí tự đầu sẽ được hiển thị rõ ràng nội dung cũng như tính năng tương tự

Bất cứ khi nào người dùng commit, git đều chụp lại vào ổ đĩa giữa phiên bản cũ và phiên bản mới trong cùng một file, kể cả ta chỉ thay đổi một kí tự nội dung file.

#### 3.2.3.5 Tính toàn vẹn của Git

Mọi thứ trong Git đều được kiểm tra **"checksummed”** trước khi được lưu trữ và được tham chiếu bởi **Checksum** này. Chức năng này cho Git biết mọi sự thay đổi trong một file hay một đường dẫn, nó bảo đảm tính toàn vẹn giúp người dùng không thể mất thông tin trong trường hợp trao đổi dữ liệu hay file lỗi không thể nhận ra. Một sự thật là Git đã lưu cơ sở dữ liệu không phải bởi tên file mà là giá trị hàm băm của chính nội dung file đó.

Hàm băm Git sử dụng là **SHA - 1**, gồm 40 kí tự được tạo ra theo nội dung của file có độ dài 40 ký tự, được tích hợp ở tầng thấp nhất của Git và là một triết lý không thể thiếu của nó. Nó còn được xem là một **“commit id”.** Một ví dụ điển hình của hàm băm **SHA - 1**, ta sẽ thực hiện các lệnh sau:

**"git show -s --format=%h” với kí tự 'H' là xem phiên bản đầy đủ**

**"git show -s --format=%h" với kí tự 'h' là xem phiên bản rút gọn của hàm băm**

****

#### 3.2.3.6 Tính dễ dàng trong Git

Git đơn giản chỉ thêm dữ liệu vào cơ sở dữ liệu Git, khó làm cho hệ thống làm một việc gì đó mà không phục hồi lại được cũng như không thể xóa hoàn toàn dữ liệu khỏi hệ thống. Điều này khiến Git trở nên thú vị đối với những người dùng thích trải nghiệm mà không sợ những mối nguy làm hại hệ thống, vì Git chứa dữ liệu đó và giúp người dùng phục hồi nó.

### 3.2.4 Ưu nhược điểm của git

Dựa vào những đặc tính và sự phổ biến của Git đối với người dùng, nhất là lập trình viên, ta sẽ đưa ra những lợi ích khi sử dụng Git. Tuy nhiên, một công cụ hay phần mềm nào cũng đều có lợi và hại. Vậy Git có nhược điểm gì và nó có độ ảnh hưởng như thế nào tới công việc mà người dùng vẫn bỏ qua và sử dụng Git?



Để biết được hai điểm này, ta sẽ phải tham khảo thông qua những đánh giá của người dùng Git trên các diễn đàn, trang báo mạng do họ đã trải nghiệm sản phẩm, có thể là ít hoặc lâu dài, họ dành thời gian cho Git, nắm rõ được những tiện lợi của Git, đồng thời biết đến một vài bất tiện của nó. Tổng hợp, nhóm em sẽ tổng hợp những đánh giá trên website: <https://www.trustradius.com/> để thấy được những ưu/nhược điểm của Git.

#### 3.2.4.1 Ưu điểm

Đa phần người dùng đều đánh giá, trong vô số công cụ thì Git lại là một “ông vua” thuộc mô hình DVCS. Họ đã khẳng định việc đó dựa vào những ưu điểm của Git, điều đã giúp nó trở thành công cụ phổ biến nhất hiện nay:

- Mô hình phân tán – làm việc tại Local, không công khai, tốc độ phản hồi nhanh chóng, làm việc offline, truy cập mọi nơi, backup

- Chia nhánh và gộp dễ dàng – có thể làm việc trên một nhánh riêng, tiết kiệm, nhanh và không chiếm nhiều dung lượng, khi xong ý tưởng riêng có thể gộp lại với nhánh chính - - - Quy trình làm việc linh hoạt – cho phép người dùng tạo một một quy trình làm việc riêng “Workflow” phù hợp, từ một quy trình chính phân ra các quy trình phụ

- Đảm bảo toàn vẹn dữ liệu – quản lý dự án tuyệt vời bằng việc sử dụng cây hàm băm SHA-1, dữ liệu bị gián đoạn có thể bị phát hiện

- Tích hợp trong nhiều công cụ khác

- Mã nguồn mở, giúp tiết kiệm chi phí cho chủ doanh nghiệp

- Cho phép một dự án có nhiều thành viên, tổ chức cùng code, testing kể cả ở cách xa nhau

- Nhiều tài liệu và quy trình, cộng đồng hỗ trợ

- Có giao diện dòng lệnh trực quan, dễ học và mở rộng khả năng, có thể triển khai dự án chỉ với một vài câu lệnh cơ bản, có nhiều tool dòng lệnh rất mạnh hỗ trợ người dùng cuối

#### 3.2.4.2 Nhược điểm

Bên cạnh những ưu điểm vượt trội trên, Git cũng tồn tại những vấn đề bất cập gây khó chịu cho người dùng. Đó chính là những nhược điểm sau đây:

- Không có cách nào để tránh tải toàn bộ lịch sử commit của một dự án, nếu dự án có nhiều Module và Package thì đây là một vấn đề lớn

- Hàm băm SHA – 1 trong kho Git xuất hiện nhiều điểm yếu do sự va chạm giá - trị băm, một máy tính thông thường có thể âm thầm phá hủy giá trị băm này

- Quá trình học sử dụng Git khá phức tạp, vài câu lệnh có Option không trực quan đòi hỏi người sử dụng phải có hiểu biết về quá trình hoạt động bên trong của Git, nhiều câu lệnh và cú pháp không đồng nhất ở vài cấp bậc

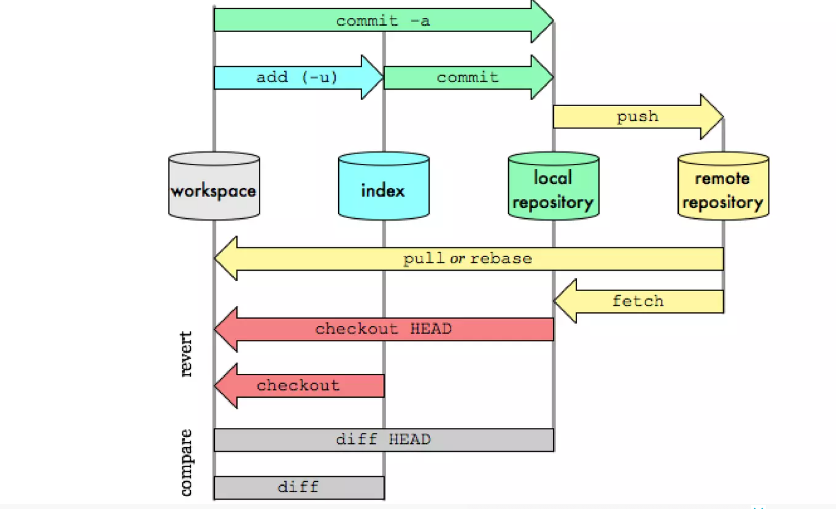
-Nếu dự án không có file dạng văn bản thì cập nhật thường xuyên sẽ làm Git giải quyết cồng kềnh và chậm chạp

- Thiếu nhiều công cụ hỗ trợ giao diện cho Git Client

## 3.3 Các khái niệm trong git và github

Trong Git và Github tồn tại nhiều thuật ngữ đòi hỏi người sử dụng, đặc biệt là những người dùng mới, phải làm quen, đọc nhiều tài liệu, tham khảo các trang mạng mới biết được ý nghĩa và công dụng của nó. Nhóm em sẽ trình bày những thuật ngữ cần quan tâm trong bài như sau:

**Repository:** viết tắt là Repo, thường được xem là một dự án, một kho chứa chính cho toàn bộ mã nguồn lệnh, tập tin, thư mục có nội dung cần cho dự án trong hệ thống quản lý Version. Có thể xem nó là một đối tượng cơ sở dữ liệu chứa tất cả mọi thứ từ phiên bản, các phiên ủy thác, lịch sử thay đổi của nội dung và không giới hạn cho người chia sẻ và sao chép. Git chia Repo thành 2 loại mô tả thông qua hình sau:



**-Local Repository:** thao tác trên máy tính cá nhân người dùng hoặc server phát triển ở vùng Local, dành cho người dùng sử dụng không cần Internet

**- Remote Repository:** (tên gọi mặc định là Origin) nằm ngoài, thao tác thông qua Local, thao tác nhiều người dùng. Khi muốn chia sẻ nội dung công việc đã làm ở Local thì ta sẽ tải lên Remote Repo để công khai nội dung công việc đó.

Có 2 hệ thống Remote được sử dụng nhiều là Github và Bitbucket

**Working tree**: (viết tắt là Tree) được xem là nhánh, chỉ mục làm việc hiện tại, nơi người dùng chỉnh sửa, tạo file mới.

**Index**: nơi bảo trì trạng thái sau khi chỉnh sửa trên Working Tree, là đối tượng nằm giữa Repo và Tree, đưa tập tin thay đổi commit lên Repo.

**Branch:** phân nhánh và ghi chép lịch sử, có 3 loại nhánh:

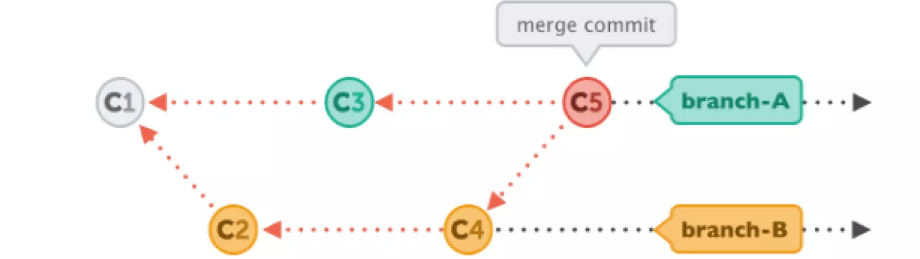
**- Local Branch**: Branch quản lý trong Local Repository

**- Remote Branch**: Branch trong Remote Repository

**-Remote Tracking Branch**: Branch Tung Local Repository theo dõi Remote Branch

**Checkout**: triển khai Branch ở Repo lên Tree, chuyển từ một nhánh sang một nhánh khác.

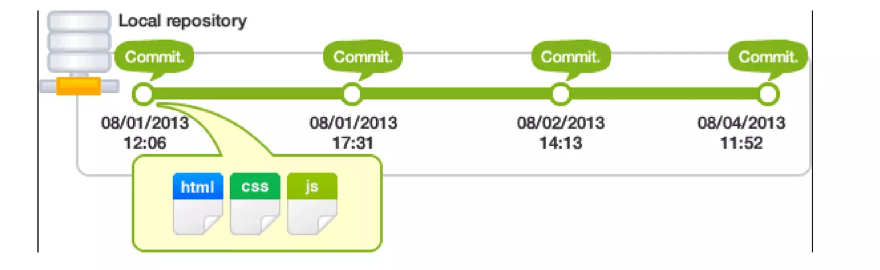
**Merge:** trộn 2 hay nhiều Commit từ một nhánh khác vào nhánh hiện tại.



**Conflict**: dây là trường hợp có 2 sự thây đổi Code cùng lúc làm máy tính không xác nhận đuợc Code “cần”. Lúc này, lập trình viên phải chinh tay thủ công, xóa bỏ dòng Code không cần thiết, một số IDE có hỗ trợ Extension cho phép ta chọn dòng Code ta cần xóa bỏ.

**Tag:** tên được gắn thêm vào file để dễ dàng tham chiếu

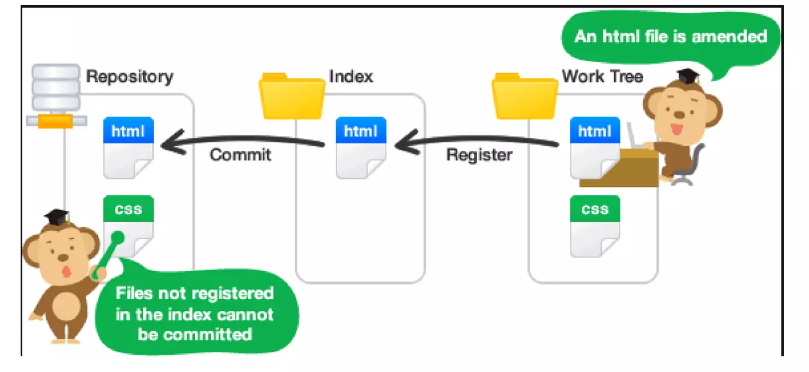
**Commit**: dây là những hành động thêm/thay đổi file hay thư mục vào Repo. Repository lúc này sẽ tao ra commit hoặc revision) để ghi lại khác biệt từ trạng thái commit lần trước với trạng thái hiện tại.Các commit được chứa tại repo nối tiếp nhau theo thứ tự thời gian



\*Trong commit sẽ có 2 trạng thái cần lưu ý:

**- Tracked**: tập tin được đánh dấu theo dõi trong Git để người dùng làm việc, gồm thêm các trạng thái phụ khác là Unmodified (chua chinh sửa), Modified (da chinh sửa) và Staged (đã sẵn sàng để commit)

**- Untracked**: tập tin còn lại mà người dùng không muốn làm việc vói nó trong Git **Staging**: bất cứ khi nào người dùng thực hiện commit thì Git sẽ tìm kiem file trong khu vực tổ chức (Staging Area) và chi những file trong khu vực này mới được xem xét dể commit.

**Revision:** giá trị hash được tạo mỗi lần commit, do Git quản lý theo thế hệ.

**HEAD:** là một điểm con trỏ, trỏ vào Commit mới nhất trong nhánh.

\*Ngoài ra, Git còn hỗ trợ thêm các câu lệnh sau cho người dùng tương tác với hệ thống Git.

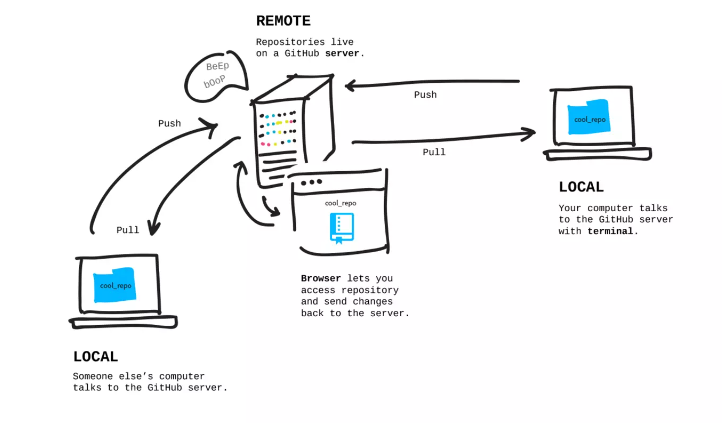
**Clone:** đây là hành động tải xuống bản sao của một Repository Local, quá trinh này càn kết nối Internet khi các Repository Instance đang được đồng bộ từ các Remote Repository.

**Fetch**: tải về (fetch) dữ liệu của toàn bộ các Branch trên URL nhưng không thực hien merge các thay đoi vào Local.

**Pull/Pull request**: tài vè (fetch) dữ liệu từ một Branch duy nhất từ Remote server và sau đó merge các thay đổi từ Remote này về Local Repository (đồng bộ từ Git server tói máy khách).

**Push**: thao tác đẩy các thay đổi bản sao chép từ Repository Local lên Remote Repository, sử dụng để lưu thay đổi vĩnh viễn trong Repo Git, tương tự như commit.

## 3.4 Git Remote Server



Ở mục tiêu đề này nhóm em muốn giới thiệu tổng quát về các giao thức được sử dụng đối với Git trên server cũng như công dụng của server đối với Git. Git server đơn giản đóng vai trò làm một máy chủ cải đặt Git, cho phép tạo Repository. Đơn cử là Github, Gitlab... sau đó cho phép người dùng "clone” nội dung về Local và có thể cập nhật lên lại Remote. Nó được xem như một Repo đùng giữa để nhiều người dung, thành viên dự án cùng đăng nhập vào, kể cả đang offline hay online, thực hiện push và pull di liệu qua nó.

Hầu hết các thao tác làm việc của gịt đều xuất hiện ở local (máy client) với local repository, chỉ khi nào người dùng có nhu cầu làm việc nhóm hay đơn giản muốn làm ở nhiều máy khác nhau thì họ sẽ sử dụng đến remote repository hay còn gọi là repository ở phía server.

Sử dụng Git như một server sẽ dễ kiểm soát cũng như phân quyền dễ dàng để quản lý các thành viên, khách hàng trong một dự án, luôn đảm bảo mức bảo mật cho dự án. Về phía các giao thức sử dụng hỗ trợ phía server, người dùng có thể dùng nhiều giao thức khác biệt để truyền dữ liệu: Local, HTTP, SSH và Git.

### 3.4.1 Giao thức local

Là giao thức cơ bản nhất trong số các giao thức Git trên server. Nó tạo một remote repository ở một đường dẫn khác trong cùng một host và dành cho nhóm người dùng muốn đăng nhập vào một hệ thống chia sẻ tập tin như NFS mount, hay như trưởng hợp mọi người cùng đăng nhập vào một máy tính.

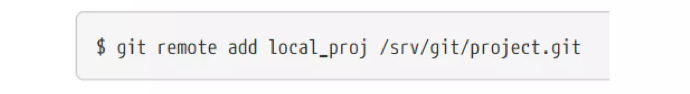
Người dùng có thể thực hiện các lệnh clone, push, pull từ local repository, một trường hợp clone từ local repo của người dùng.



Người dùng gỗ thẳng đường dẫn của chính file đó để ít ảnh hưởng đến tốc độ của git.



Để thêm Local Repo vào một dự án đang thực hiện, người dùng gõ lệnh như sau:



Với local proj là tên Local Repo

**Ưu điểm**: dễ dàng, chỉ cần truy cập một file đã tồn tại được phân quyền và có hỗ trợ mạng internet, dễ dàng chuyển giao công việc cho người bên ngoài nhóm vào làm việc.

**Nhược điểm**: thiết lập truy cập chia sẻ khó khăn đối với môi trường có nhiều nơi cần truy cập, chia sẻ phân vùng không phải cách nhanh nhất, mọi người dùng đều có quyền shell cao nhất nên có thể chỉnh sửa file bên trong hay làm gián đoạn repo.

### 3.4.2 Giao thức HTTP

Git giao tiếp với HTTP qua 2 phương thức, cũ nhất sẽ là Dump HTTP và mới nhất là Smart HTTP.

#### 3.4.2.1 Smart HTTP

Giao thức này hoạt động giống như SSH hay những giao thức của git, có cống HTTPs chuẩn và dùng nhiều phương pháp HTTP chứng thực khác nhau. Sau đây là một vài đặc điểm:

-Dễ dàng cho người dùng truy cập bằng phương pháp chứng thực

username/password hơn là thiết lập key cho phiên SSH

-Là cách để sử dụng gít phổ biến nhất hiện nay, có thể phục vụ cho cả giao thức git://

-Có thể đẩy lên bằng phương pháp chứng thực và mã hóa như giao thức SSH, nếu repo yêu cầu chứng thực, server sẽ yêu cầu quyền username/password de truy cập

-Phục vụ cả hai phương thức trên chỉ bằng một URL

#### 3.4.2.2 Kết xuất HTTP

Nếu server không phản hồi với giao thức Smart HTTP, Git Client sẽ cố gắng sử dụng lại giao thức đơn giản Dump HTTP. Giao thức xem bare git repo như là một tập tin thông thường trên máy chủ web. Giao thức rất dễ cài đặt, người dùng chỉ cần đặt bare git repo trong một thư mục HTTP root và một hook cập nhật cụ thể. Bất cứ thành viên nào có quyền dưới web server đều có thể clone Repository.

Để cấp quyền cho Repo bởi giao thức HTTP, chỉ cần vài lệnh sau:



Sau đó, để chạy hook cập nhật, người dùng chỉ cần chạy một câu lệnh thích hợp là (git-update-server-info) để khiến git fetch và cloning công việc ngay lập tức, chỉ xảy ra khi người dùng muốn push tới repo đỏ hoặc muốn clone thứ gì đó như câu lệnh sau:



**Ưu điểm:**

-Đơn giản vì chỉ cần một URL. cho mọi loại truy cập và có server điều khiển chứng thực khi cần

-Có thể chứng thực thông qua username/password giúp người dùng không phải qua thao tác tại key giữ bí mật và đưa key công khai lên server một cách rắc rối Nhanh và hiệu quả

**Nhược điểm:**

-Thiết lập có thể trở nên khó khăn ở vài server so với SSH

-Việc cung cấp thông tin đăng nhập đôi khi lại trở nên khó khăn, mặc dù có vài công cụ hỗ trợ đăng nhập với caching

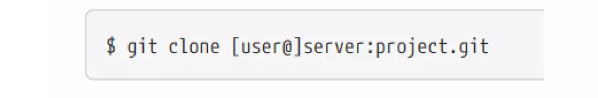
### 3.4.3 Giao thức SSH

Giao thức vận chuyển nổi tiếng của git khi tự hosting chính nó là SSH. Giao thức này được cài đặt ở hầu hết server. Nó đồng thời cũng là một giao thức chứng thực mạng và bởi tính phổ cập của nó nên rất dễ dàng thực hiện thiết lập và sử dụng.

Để clone Git Repository thông qua SSH, ta chỉ cần sử dụng URL đặc trưng ssh:// như sau:



Người dùng có thể sử dụng củ pháp ngắn gọn hơn như sau:



**Ưu điểm:**

-Giao thức phổ biết, dễ dàng cài đặt

-SSH daemon lại phổ biến, được nhiều nhà quản trị mạng có kinh nghiệm sử dụng

-Được cái sẵn trên nhiều hệ điều hành phân phối và có công cụ để quản lý

-Truy cập bảo mật, mọi dữ liệu đều được mã hóa và đã được chứng thực

-Ảnh hưởng tích cực đến các giao thức khác như HTTPs, Gít và các giao thức Local

**Nhược điểm:**

-Không hỗ trợ truy cập ẩn danh vào Git repo

-Nếu hệ thống sử dụng SSH, người dùng phải có cái tính năng SSH truy cập vào, không phù hợp với những dự án mã nguồn mở

-Khi cài đặt SSH trong hệ thống, chỉ có thể sử dụng nó khi push dữ liệu và phải xải công cụ, giao thức khác để fetch dữ liệu về

### 3.4.4 Git Daemon

Được đính kèm với các package cùng git, nó lắng nghe ở cổng mặc định là 9418 cung cấp một dịch vụ giống như giao thức SSH nhưng không có chứng thực. Muốn sử dụng, người dùng phải tạo một file git-daemon-export-ok bới vì daemon sẽ không phục vụ khi không có file trong đó, tuy nhiên nó lại không được bảo mật. Bất cứ ai tìm thấy địa chỉ URI. project của bạn đều có thể push vào đó.

**Ưu điểm:**

-Là giao thức internet nhanh nhất

-Phù hợp cho nhu cầu có một dự án public hay một dự án lớn không cần phương pháp xác thực

-Sử dụng chung phương pháp vận chuyển dữ liệu giống SSH nhưng không mã hỏa và xác thực

**Nhược điểm:**

-Yếu tố không xác thực

-Khó cấu hình thiết lập, yêu cầu phải chạy trên đúng daemon chính nó nên phải cải dặt xinetd hay systemd

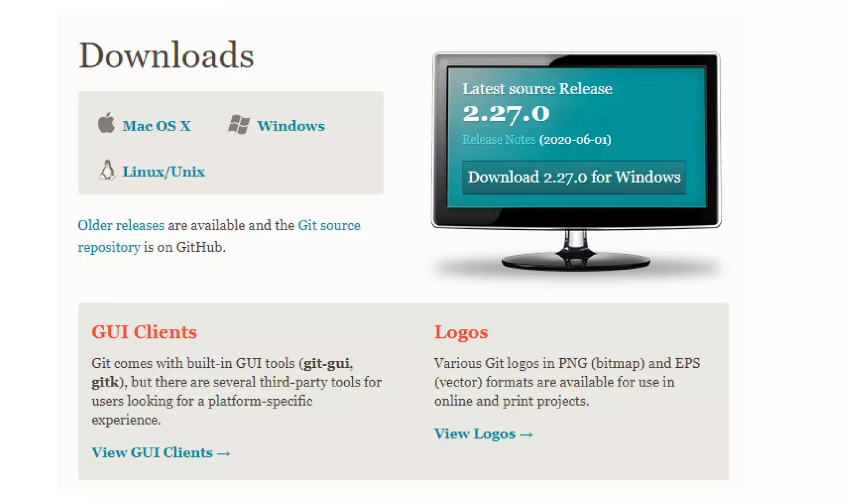
-Phải chỉnh tường lửa mở cổng mặc định 9418, do không phải là port tiêu chuẩn nên thường bị chặn

## 3.5 Cài đặt Docker

**Các môi trường Git hỗ trợ**

Git hỗ trợ nhiều nền tảng, có cả 3 nền tảng phổ biến với người dùng: Windows, Linux, MacOS. Do không có thiết bị chạy hệ điều hành MacOS, nhóm xin thực hiện demo các bước cài đặt trên 2 nền tảng còn lại là Windows và Linux.

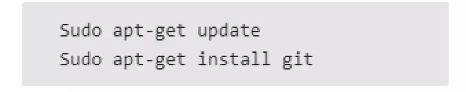
Người dùng tải file cài đặt hoặc clone Source Code từ trang web chính thức của Git cho 3 nền tảng qua đường dẫn sau: <https://git-scm.com/downloads>



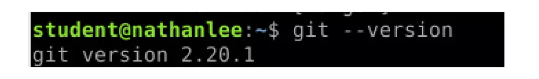
### 3.5.1 Trên Linux

Hệ điều hành Linux hỗ trợ Git rất tốt, đặc biệt là với công cụ Terminal mặc định của Linux, giúp Git phát huy được tối đa sức mạnh vốn có. Git sẽ có nhiều cách cải khác nhau dựa trên từng Distro từ nhân Linux. Người dùng cần truy cập vào trang chủ của Git, làm theo hướng dẫn với đường dẫn sau: <https://git-scm.com/download/linux>

**Bước 1:** Mở terminal và gỗ 2 lệnh sau



**Bước 2:** Gõ lệnh "git -version' để kiểm tra đã cài đặt thành công



### 3.5.2 Trên Windows

Windows vốn là hệ điều hành quen thuộc với đại đa số người dùng. Việc Git hỗ trợ cho hệ điều hành quốc dân này tạo nhiều điều kiện đa dạng cho người dùng, nhất là lập trình viên chọn lựa làm việc trên hệ điều hành họ thích. Để cải Git trên Windows, người dùng cải công cụ giả lập Terminal.

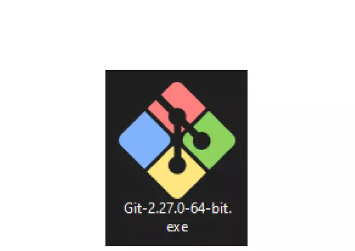
**Bước 1:** Truy cập vào đường dẫn <https://gitforwindows.org/>



Hoặc tải từ trang chủ: <https://git-scm.com/download/win>

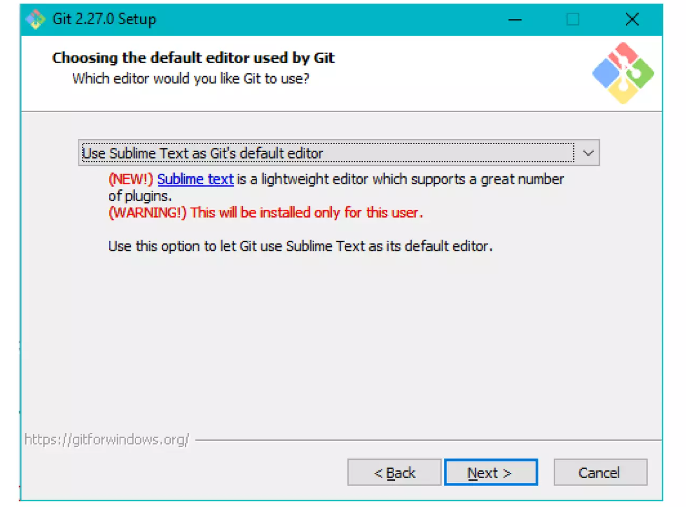


**Bước 2:** Tải xuống một file \*.exe, nhấn chuột vào file để tiến hành cài đặt

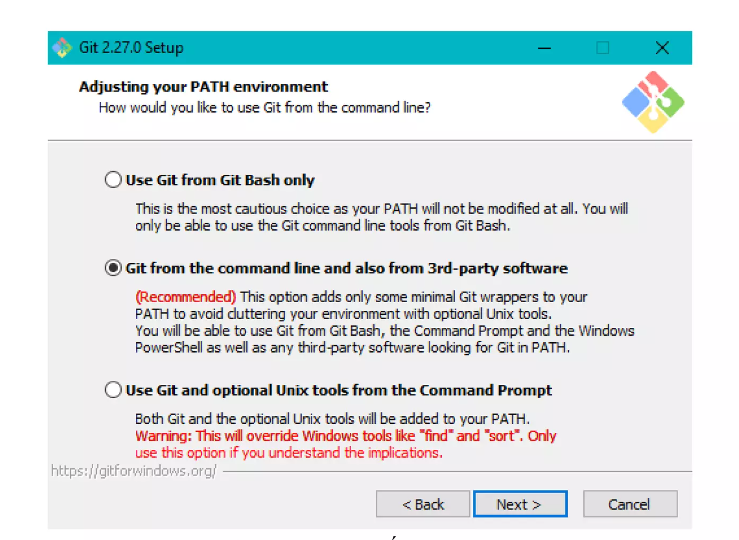


**Bước 3:** Cài đặt bình thường như các phần mềm khác có hỗ trợ giao diện cho Windows, dựa vào yêu cầu của mỗi người dùng mà chọn lựa cho phù hợp

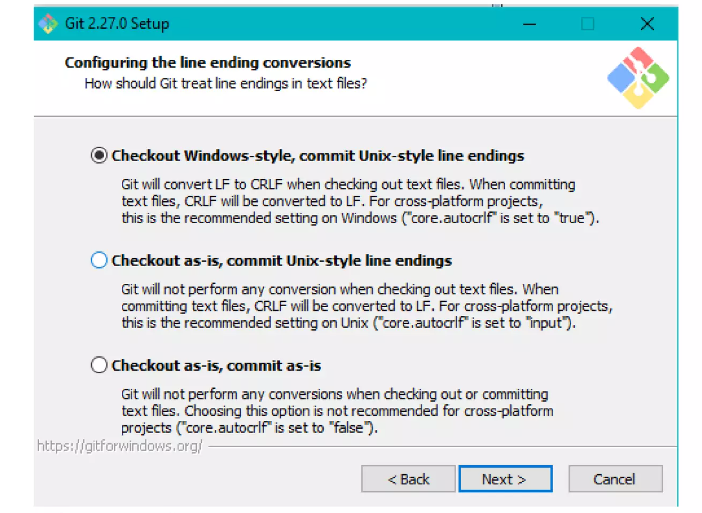
**Bước 4:** Bước tiếp theo là cài git editor cho windows, có thể là các text editor như vim, notepad++, sublime text....



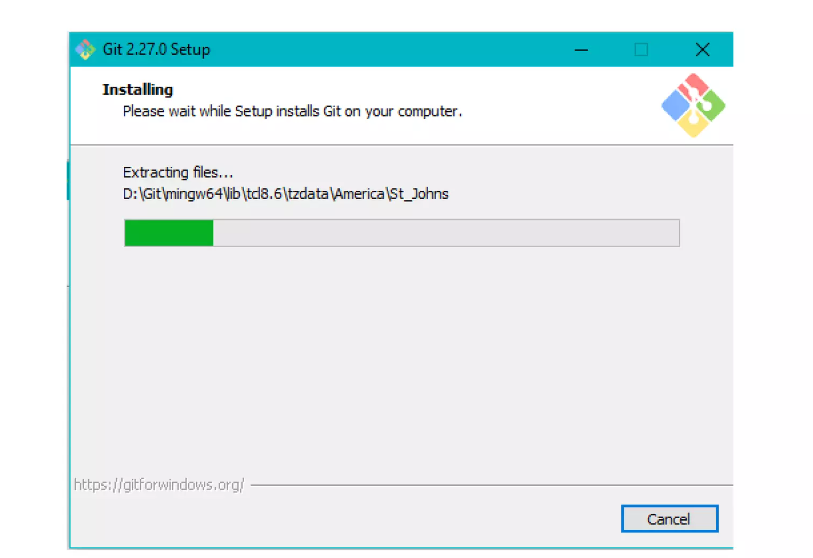
**Bước 5:** Tiếp theo, lựa PATH môi trường thích hợp với nhu cầu



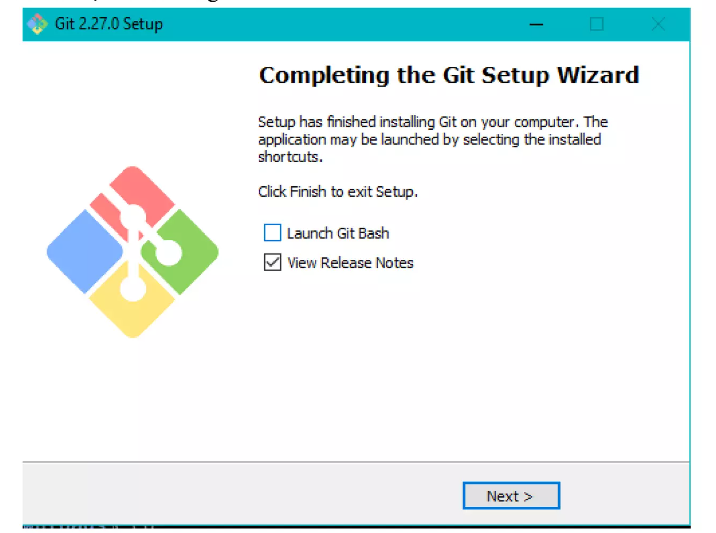
**Bước 6:** Chọn phong cách dòng lệnh như thế nào



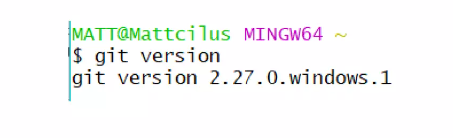
**Bước 7:** Bấm install để cài đặt



**Bước 8:** Cài đặt thành công



**Bước 9:** Gõ lệnh git version để kiểm tra phiên bản cũng như Git đã cài đặt thành công



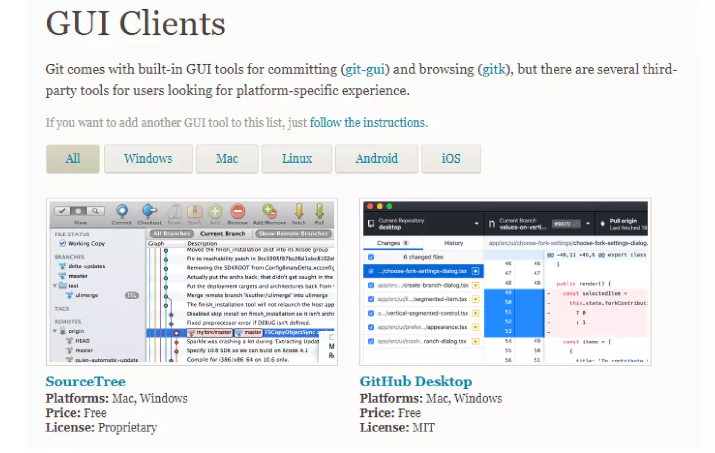
Sau khi cài đặt, do tùy chọn ở bước 5, công cụ Git sẽ xuất hiện hai tệp chạy chương trình, mỗi tệp ứng với một môi trường. Tệp git-bash sẽ dành cho người dùng quen thuộc với hệ thống kiến trúc hệ điều hành Unix/Linux. Tệp git-cmd còn lại phù hợp cho những người dùng quen thuộc với MS – Dos, vốn là các lệnh được tích hợp trong Command Prompt (CMD) của hệ điều hành Windows.



### 3.5.3 Phiên bản GUI

Ngoài ra, người dùng còn có thể tải về phiên bản Graphical User Interface (GUI) dành cho Git, hỗ trợ trên cả 3 nền tảng cung cấp giao diện để người dùng thao tác dễ dàng, giúp họ không gặp khó khăn với vấn đề học cách sử dụng dòng lệnh Sẽ có rất nhiều phiên bản GUI bao gồm miễn phí, trả phí, tích hợp tính năng khác để phục vụ tốt hơn. Truy cập đường dẫn sau để tải về phiên bản giao diện của Git:

<https://gitscm.com/download/gui/windows>



**3.6 Tính năng quản lý được hỗ trợ trong github**

Các tính năng quản lý dự án trong GitHub giúp tổ chức và theo dõi quá trình phát triển của dự án phần mềm một cách hiệu quả, tạo điều kiện thuận lợi cho sự hợp tác và tiến triển của nhóm.

**1. Projects (Dự án):**

Tạo Dự án: GitHub Projects là một công cụ quản lý dự án tích hợp, cho phép bạn tạo các bảng Kanban để tổ chức và theo dõi công việc.

Tùy chỉnh Cột và Thẻ: Bạn có thể tạo cột tùy chỉnh để phản ánh quy trình làm việc của dự án và thêm các thẻ đại diện cho các công việc.

Kéo và Thả Công việc: Di chuyển công việc giữa các cột bằng cách kéo và thả thẻ, giúp cập nhật trạng thái công việc một cách dễ dàng.

**2. Issues (Vấn đề):**

Quản lý Vấn đề: GitHub Issues là một hệ thống theo dõi công việc, vấn đề và yêu cầu tính năng trong dự án của bạn.

Gán Nhãn và Milestones: Gán nhãn và milestones cho các vấn đề để phân loại và theo dõi công việc theo các tiêu chí cụ thể.

Thảo luận và Phản hồi: Các thành viên nhóm có thể thảo luận và phản hồi trực tiếp trên các vấn đề, giúp tăng cường sự tương tác và hỗ trợ quy trình làm việc.

**3. Pull Requests (Yêu cầu kéo):**

Tạo và Xem xét Pull Requests: GitHub cho phép người dùng tạo pull request để đề xuất các thay đổi vào mã nguồn chính và cho phép các thành viên khác xem xét và đánh giá.

Gán Người Review và Assignee: Gán người review để yêu cầu xem xét và gán assignee để phân công công việc liên quan đến pull request.

Tích hợp CI/CD: Pull requests có thể được kết hợp với các dịch vụ CI/CD để tự động kiểm tra và triển khai mã nguồn.

**4. Wiki:**

Tài liệu Dự án: GitHub Wiki cung cấp một không gian để tạo và quản lý tài liệu dự án, bao gồm hướng dẫn sử dụng, tài liệu kỹ thuật và các thông tin khác liên quan.

Chia sẻ Kiến thức: Cung cấp nền tảng để chia sẻ kiến thức và thông tin về dự án với các thành viên trong nhóm và cộng đồng người dùng.

**5. Insights (Thông tin):**

Thống kê và Báo cáo: GitHub Insights cung cấp các báo cáo và thống kê về hoạt động và tiến độ của dự án, bao gồm cả thông tin về sự thay đổi của mã nguồn, cấp độ tham gia của các thành viên và nhiều hơn nữa.

Phân tích Dữ liệu: Đánh giá hiệu suất và sự tham gia của các thành viên trong nhóm dựa trên dữ liệu từ các hoạt động trên GitHub.

# CHƯƠNG 4 TRIỂN KHAI DEMO

**4.1 Mô hình và Kỹ thuật phân nhánh được sử dụng.**

**4.1.1 Mô hình phân nhánh GitFlow**

Mô hình phân nhánh Gitflow là một phương pháp quản lý mã nguồn dựa trên Git, được phát triển bởi Vincent Driessen. Mô hình này cung cấp một cách cấu trúc hóa và quản lý quy trình phát triển phần mềm, bao gồm các quy tắc và quy trình rõ ràng cho việc tạo, quản lý và hợp nhất các nhánh. Dưới đây là một sơ đồ và mô tả chi tiết về các nhánh và quy trình trong mô hình phân nhánh Gitflow:

**1. Nhánh Master:**

Nhánh master là nhánh chính của dự án, chứa mã nguồn ổn định đã được kiểm tra và sẵn sàng triển khai.

Mọi thay đổi từ nhánh release và hotfix được hợp nhất vào nhánh này sau khi kiểm tra và hoàn thành.

**2. Nhánh Develop:**

Nhánh develop là nơi phát triển chính của dự án, chứa mã nguồn mới nhất và các tính năng đang được phát triển.

Mọi tính năng mới được phát triển từ nhánh này.

**3. Nhánh Feature:**

Mỗi tính năng mới hoặc thay đổi lớn được phát triển trên một nhánh tính năng riêng.

Nhánh tính năng được tạo từ develop và sau khi hoàn thành, được hợp nhất trở lại vào develop.

**4. Nhánh Release:**

Khi chuẩn bị cho một bản phát hành mới, một nhánh release được tạo từ develop.

Trên nhánh release, các công việc như kiểm tra, sửa lỗi và chuẩn bị cho bản phát hành được thực hiện.

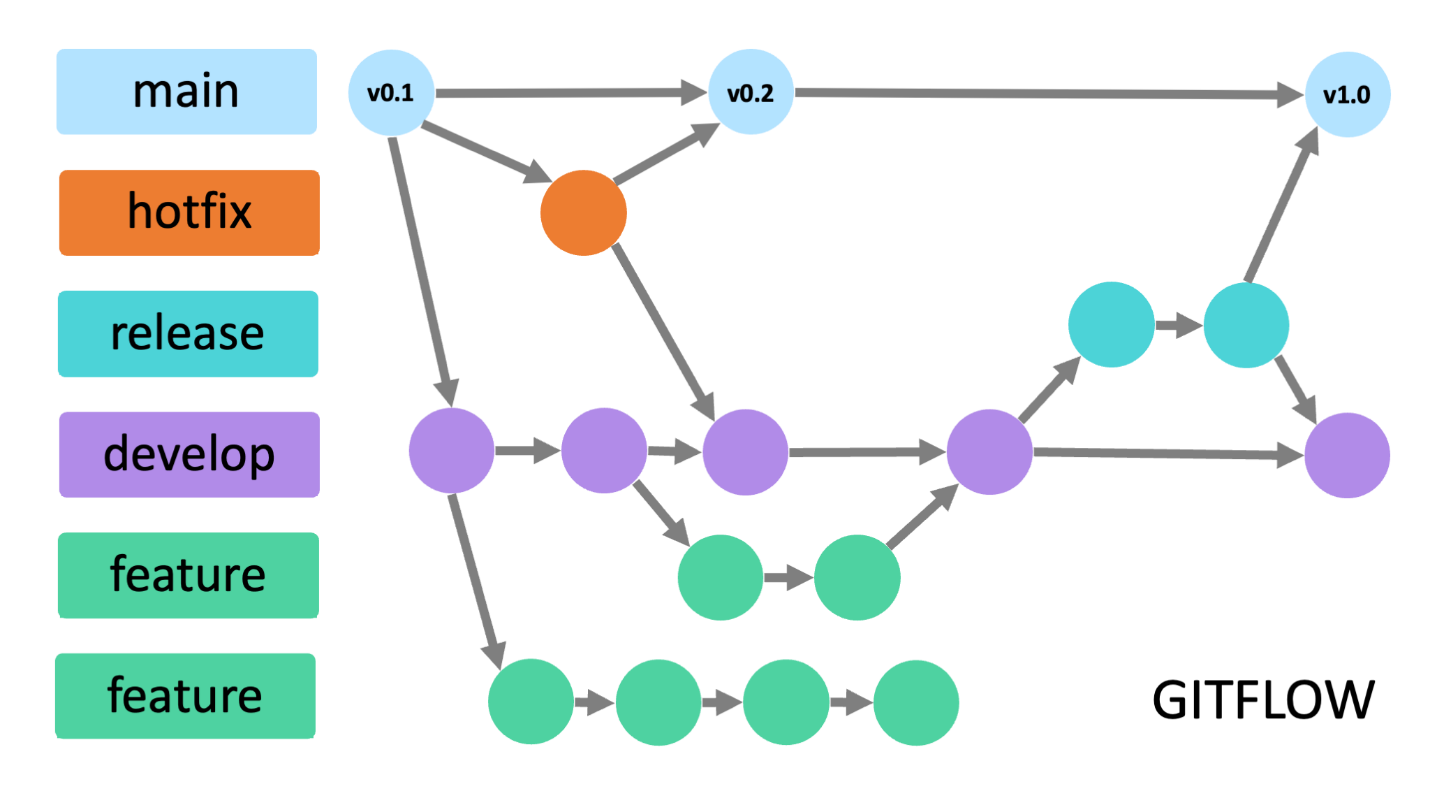
Sau khi hoàn thành, nhánh release được hợp nhất vào cả master và develop.

**5. Nhánh Hotfix:**

Để xử lý các lỗi gấp, một nhánh hotfix được tạo từ nhánh master.

Trên nhánh hotfix, các sửa đổi cần thiết được thực hiện và sau đó hợp nhất vào cả master và develop.

**Sơ đồ mô hình GitFlow :**



**Lợi Ích:**

Quản lý Mã nguồn Tốt hơn: Mô hình Gitflow cung cấp cách cấu trúc và tổ chức mã nguồn rõ ràng, giúp quản lý dự án hiệu quả hơn.

Tách Biệt Công việc: Các nhánh riêng biệt cho tính năng, bản phát hành và sửa lỗi giúp tách biệt các công việc và giảm xung đột.

Kiểm soát Quy trình Phát triển: Mỗi bước trong quy trình phát triển (phát triển, kiểm tra, phát hành, sửa lỗi) được xác định rõ ràng và kiểm soát chặt chẽ.

**4.1.2 Chi tiết kỹ thuật phân nhánh GitFlow**

**1. Tạo Nhánh (Branching):**

Tạo Nhánh: Sử dụng lệnh git branch <tên\_nhánh> để tạo một nhánh mới từ nhánh hiện tại. Ví dụ: git branch feature/new-feature.

Chuyển Nhánh: Sử dụng lệnh git checkout <tên\_nhánh> để chuyển đổi giữa các nhánh.

Ví dụ: git checkout feature/new-feature.

Tạo và Chuyển Nhánh: Bạn cũng có thể sử dụng lệnh git checkout -b <tên\_nhánh> để tạo và chuyển đến một nhánh mới trong một lệnh duy nhất.

**2. Làm việc trên Nhánh (Working on Branch):**

Thêm và Commit: Thực hiện các thay đổi trong mã nguồn của bạn trên nhánh của bạn. Sử dụng git add để thêm các tệp tin đã thay đổi và git commit để lưu các thay đổi đó vào lịch sử commit.

Push Nhánh: Khi bạn đã thực hiện các commit trên nhánh của mình, bạn có thể đẩy nhánh đó lên repository từ xa bằng cách sử dụng git push origin <tên\_nhánh>.

**3. Hợp Nhất (Merging) và Tạo Pull Request:**

Hợp Nhất Nhánh: Khi bạn đã hoàn thành công việc trên nhánh của mình, bạn có thể hợp nhất nhánh đó vào nhánh chính (ví dụ: master) bằng cách sử dụng git merge.

Tạo Pull Request (PR): Trên GitHub, bạn có thể tạo một Pull Request từ nhánh của mình vào nhánh chính của dự án. Điều này cho phép các thành viên khác trong nhóm xem xét và đánh giá công việc của bạn trước khi hợp nhất vào nhánh chính.

**4. Xử lý xung đột (Handling Conflicts):**

Xung đột: Nếu có xung đột giữa các thay đổi trên nhánh của bạn và nhánh mục tiêu, Git sẽ báo lỗi. Bạn cần giải quyết xung đột này bằng cách chỉnh sửa tệp tin và sau đó thêm và commit lại các thay đổi.

**5. Xóa Nhánh (Deleting Branch):**

Xóa Nhánh Cục Bộ: Sử dụng lệnh git branch -d <tên\_nhánh> để xóa một nhánh cục bộ sau khi đã hợp nhất.

Xóa Nhánh Từ Xa: Trên GitHub, sau khi đã hợp nhất một Pull Request, bạn có thể chọn xóa nhánh từ xa mà Pull Request đó đến từ.

**4.2 Áp dụng mô hình GitFlow vào dự án**

**1. Khởi Tạo Repository và Thiết Lập Cấu Trúc Nhánh**

**1.1. Khởi Tạo Repository:**

* Tạo repository trên GitHub cho dự án của bạn.
* Sao chép URL của repository để sử dụng trong việc clone dự án xuống máy tính cá nhân hoặc máy chủ.

**1.2. Clone Repository:**

* Sử dụng câu lệnh git clone <repository\_url> để sao chép repository xuống máy tính cá nhân.

**1.3. Thiết Lập Cấu Trúc Nhánh:**

* Tạo nhánh master: Nhánh chứa mã nguồn ổn định, sẵn sàng cho việc triển khai.
* Tạo nhánh develop: Nhánh phát triển chứa mã nguồn mới nhất, được tích hợp từ các nhánh tính năng.
* Sử dụng câu lệnh git checkout -b develop để tạo và chuyển sang nhánh develop.

**2. Phát Triển Tính Năng**

**2.1. Tạo Nhánh Tính Năng:**

* Tạo một nhánh mới từ nhánh develop cho mỗi tính năng cụ thể bằng câu lệnh git checkout -b feature/<feature\_name>.

**2.2. Phát Triển Tính Năng:**

* Phát triển tính năng trên nhánh tính năng vừa tạo.
* Thực hiện commit thường xuyên và đồng bộ hóa với nhánh develop bằng câu lệnh git push origin feature/<feature\_name>.

**2.3. Hoàn Thiện Tính Năng:**

* Khi tính năng hoàn thành, thực hiện kiểm tra và kiểm thử kỹ lưỡng.

**3. Chuẩn Bị Cho Phát Hành**

**3.1. Tạo Nhánh Release:**

* Tạo một nhánh từ develop gọi là release/\* bằng câu lệnh git checkout -b release/<version\_number>.

**3.2. Chuẩn Bị Phiên Bản:**

* Tiến hành kiểm tra tổng thể, cập nhật tài liệu, và chuẩn bị cho việc phát hành.

**4. Phát Hành Phiên Bản**

**4.1. Merge Nhánh Release vào Master và Develop:**

* Merge nhánh release/\* vào master và develop bằng các câu lệnh git merge release/<version\_number>.

**4.2. Triển Khai Phiên Bản:**

* Triển khai phiên bản đã phát hành lên môi trường sản phẩm.

**4.3. Đặt Tag Cho Phiên Bản:**

* Đặt tag cho phiên bản phát hành trên nhánh master bằng câu lệnh git tag -a <version\_number> -m "Release <version\_number>".

**5. Sửa Lỗi và Bảo Trì**

**5.1. Tạo Nhánh Hotfix:**

* Nếu có sự cố hoặc lỗi phát sinh, tạo một nhánh từ master gọi là hotfix/\* bằng câu lệnh git checkout -b hotfix/<fix\_name>.

**5.2. Sửa Lỗi:**

* Sửa lỗi trong nhánh hotfix/\* và sau đó merge vào cả master và develop để đồng bộ hóa.

**6. Quản Lý Thay Đổi**

* Đảm bảo sử dụng message commit có ý nghĩa và theo chuẩn của Gitflow.
* Sử dụng các công cụ quản lý mã nguồn như GitHub để theo dõi và kiểm soát quy trình làm việc theo Gitflow.

**4.3 Kế hoạch bảo trì phần mềm**

Mục tiêu:

Mục tiêu của kế hoạch này là duy trì tính ổn định và cải thiện chức năng của phần mềm qua các phiên bản bảo trì liên tục. Các bản cập nhật sẽ tập trung vào việc sửa lỗi, cải thiện tính năng và hiệu suất, đồng thời đáp ứng nhu cầu của người dùng.

Thành viên nhóm:

+ Nhóm phát triển Giao diện(FontEnd): Tuấn, Diệp, Hội, Dương, Vũ, Tuyến.

+ Nhóm phát triển BackEnd: Tuấn.

Phiên bản 1.0 (Tháng 3, 2024)

Mô tả:

Cập nhật giao diện người dùng để tăng tính tương tác và trải nghiệm người dùng.

Sửa lỗi liên quan đến hiệu suất và tính năng không hoạt động đúng cách.

Chi tiết công việc:

Phát triển giao diện người dùng mới:

Mô tả: Cải thiện giao diện người dùng để tăng tính tương tác và thu hút người dùng.

Phân công: Nhóm phát triển giao diện - Amanda và John.

Thời gian: Dự kiến 2 tuần.

Nhánh: feature/ui-update.

Sửa lỗi hiệu suất:

Mô tả: Tìm kiếm và sửa lỗi liên quan đến hiệu suất của ứng dụng để tăng tốc độ và đáp ứng.

Phân công: Nhóm phát triển backend - David và Emily.

Thời gian: Dự kiến 1 tuần.

Nhánh: hotfix/performance-issue.

*Phiên bản 1.1 (Tháng 4, 2024)*

· **Mô tả:**

· Cập nhật giao diện người dùng để tăng tính tương tác và trải nghiệm người dùng.

· Sửa lỗi liên quan đến hiệu suất và tính năng không hoạt động đúng cách.

· **Chi tiết công việc:**

· Phát triển giao diện người dùng mới:

· Mô tả: Cải thiện giao diện người dùng để tăng tính tương tác và thu hút người dùng.

· Phân công: Nhóm phát triển giao diện - Amanda và John.

· Thời gian: Dự kiến 2 tuần.

· Nhánh: feature/ui-update.

· Sửa lỗi hiệu suất:

· Mô tả: Tìm kiếm và sửa lỗi liên quan đến hiệu suất của ứng dụng để tăng tốc độ và đáp ứng.

· Phân công: Nhóm phát triển backend - David và Emily.

· Thời gian: Dự kiến 1 tuần.

· Nhánh: hotfix/performance-issue.

*Phiên bản 1.2 (Tháng 6, 2024)*

· **Mô tả:**

· Thêm tính năng đăng nhập bằng các tài khoản mạng xã hội (Facebook, Google).

· Cải thiện tính năng tìm kiếm.

· **Chi tiết công việc:**

· Phát triển tính năng đăng nhập bằng mạng xã hội:

· Mô tả: Thêm tính năng cho phép người dùng đăng nhập bằng tài khoản mạng xã hội để tăng tính tiện ích.

· Phân công: Nhóm phát triển backend - David và Emily.

· Thời gian: Dự kiến 2 tuần.

· Nhánh: feature/social-login.

· Cải thiện tính năng tìm kiếm:

· Mô tả: Tối ưu hóa tính năng tìm kiếm để cung cấp kết quả chính xác và nhanh chóng hơn cho người dùng.

· Phân công: Nhóm phát triển frontend - Amanda và John.

· Thời gian: Dự kiến 1 tuần.

· Nhánh: feature/search-improvement.

*Phiên bản 1.3 (Tháng 8, 2024)*

· **Mô tả:**

· Tối ưu hóa dữ liệu và cơ sở dữ liệu để tăng hiệu suất.

· Sửa lỗi và cải thiện tính ổn định.

· **Chi tiết công việc:**

· Tối ưu hóa cơ sở dữ liệu:

· Mô tả: Kiểm tra và tối ưu hóa cơ sở dữ liệu để tăng hiệu suất và sự ổn định của hệ thống.

· Phân công: Nhóm DevOps - Michael và Sarah.

· Thời gian: Dự kiến 2 tuần.

· Nhánh: feature/database-optimization.

· Kiểm tra và sửa lỗi tự động:

· Mô tả: Thiết lập hệ thống kiểm tra tự động để phát hiện và sửa lỗi tự động.

· Phân công: Nhóm kiểm thử - Daniel và Lisa.

· Thời gian: Dự kiến 1 tuần.

· Nhánh: feature/automated-testing.

Tổng Kết

Kế hoạch bảo trì phần mềm được thiết kế để cải thiện và bổ sung các tính năng của trang web giới thiệu việc làm một cách có tổ chức và hiệu quả.

Công việc đã được phân công cho từng nhóm nhân viên với sự phân chia công việc rõ ràng, đảm bảo tiến độ và chất lượng của từng phiên bản.

Mỗi tính năng mới và sửa đổi được thực hiện trên các nhánh riêng biệt để quản lý mã nguồn dễ dàng và giảm thiểu xung đột.

**4.3 Thống kê, theo dõi quy trình bảo trì qua lịch sử commit**

Báo cáo Tổng kết Quá trình Bảo trì - Website Mua sắm Trực tuyến "Shoppe"

Thời gian: Tháng 3 - Tháng 4, 2024

Tổng quan:

Trong khoảng thời gian bảo trì này, chúng tôi đã tiến hành nhiều sửa đổi và cải thiện trên trang web mua sắm trực tuyến "Shoppe". Dưới đây là một tóm tắt về những gì đã được thực hiện:

Các Thay Đổi:

Tính năng mới:

Tổng số tính năng mới được thêm vào: 3

Mô tả tóm tắt về các tính năng mới:

Tính năng "Đánh giá và Nhận xét Sản phẩm": Cho phép người dùng đánh giá và viết nhận xét về sản phẩm mà họ đã mua.

Tính năng "Gợi ý Mua Hàng Dựa trên Lịch Sử": Gợi ý sản phẩm dựa trên lịch sử mua hàng của người dùng.

Tính năng "Thanh toán bằng Ví Điện Tử": Hỗ trợ thanh toán nhanh chóng và thuận tiện bằng ví điện tử.

Sửa lỗi:

Tổng số lỗi đã sửa: 10

Mô tả tóm tắt về các lỗi đã sửa: Bao gồm sửa lỗi hiển thị sản phẩm không đồng nhất, lỗi tương tác với các nút chức năng, và sửa lỗi liên quan đến thanh toán.

Cải thiện hiệu suất:

Tổng số cải thiện về hiệu suất: 5

Mô tả tóm tắt về các cải thiện hiệu suất: Bao gồm tối ưu hóa tốc độ tải trang, cải thiện thời gian phản hồi và tăng cường bảo mật.

Số lượng Commit:

Tổng số lượng commit: 50

Phân loại commit:

Tính năng mới: 20 commit

Sửa lỗi: 20 commit

Cải thiện hiệu suất: 10 commit

Đánh giá Tổng quan:

Quá trình bảo trì đã đạt được một số kết quả tích cực, bao gồm việc thêm tính năng mới, sửa lỗi và cải thiện hiệu suất. Số lượng commit cho thấy sự hoạt động tích cực và cam kết của nhóm trong việc nâng cấp và bảo trì trang web mua sắm trực tuyến "Shoppe".

Đề Xuất Tiếp Theo:

Tiếp tục theo dõi và xử lý các lỗi và yêu cầu từ người dùng.

Tiếp tục phát triển tính năng mới để cải thiện trải nghiệm người dùng.

Tổ chức các phiên bản bảo trì định kỳ để duy trì tính ổn định và an toàn của trang web.

**4.4 Danh sách các lệnh git được sử dụng**

Dưới đây là một danh sách các lệnh Git mà bạn có thể sử dụng trong quá trình demo để thực hiện các tác vụ cơ bản như tạo nhánh, hợp nhất các nhánh, xem lịch sử commit và xem trạng thái hiện tại của dự án:

* git init: Khởi tạo một kho chứa Git mới trong thư mục hiện tại.
* git clone <url>: Sao chép một kho chứa Git từ một URL.
* git status: Xem trạng thái hiện tại của thư mục làm việc và các tệp tin đã thay đổi.
* git add <tên\_tệp>: Thêm các thay đổi trong tệp tin cụ thể vào chỉ mục chuẩn bị.
* git commit -m "Thông điệp commit": Tạo một commit mới với các thay đổi trong chỉ mục chuẩn bị và một thông điệp mô tả.
* git branch: Liệt kê tất cả các nhánh có trong kho chứa.
* git checkout -b <tên\_nhánh>: Tạo và chuyển đến một nhánh mới.
* git checkout <tên\_nhánh>: Chuyển đến một nhánh tồn tại.
* git merge <tên\_nhánh>: Hợp nhất nhánh hiện tại với nhánh đã chỉ định.
* git log: Hiển thị lịch sử commit trên nhánh hiện tại.
* git show <commit\_id>: Hiển thị thông tin chi tiết về một commit cụ thể.
* git pull: Cập nhật các thay đổi từ remote repository về local repository.
* git push: Đẩy các thay đổi từ local repository lên remote repository.
* git remote -v: Hiển thị thông tin về các remote repository được liên kết với local repository.
* git fetch: Lấy tất cả các thay đổi từ remote repository mà không thực hiện hợp nhất.
* git tag: Quản lý các tag (nhãn) trên các commit.

**4.5 Kết quả**

# KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Git và GitHub là hai công cụ mạnh mẽ giúp chúng em quản lý và theo dõi mã nguồn trong dự án phần mềm của mình một cách hiệu quả. Bằng cách sử dụng các tính năng như quản lý phiên bản, hợp nhất và theo dõi vấn đề, chúng em đã có thể cải thiện quá trình phát triển và hợp tác trong dự án của mình.

Sử dụng Git và GitHub trong dự án của chúng em đã đem lại nhiều lợi ích đáng kể và kết quả tích cực:

* Quản Lý Phiên Bản Hiệu Quả: Sử dụng Git cho phép chúng em quản lý lịch sử thay đổi của mã nguồn một cách linh hoạt và an toàn. Điều này giúp chúng em dễ dàng theo dõi sự thay đổi trong mã nguồn, quay lại phiên bản trước đó, và giải quyết xung đột một cách dễ dàng.
* Hợp Nhất Tính Năng Một Cách Cẩn Thận: Sử dụng GitHub, các pull request được sử dụng để kiểm tra và xác nhận tính năng mới trước khi hợp nhất vào nhánh chính. Quy trình này giúp đảm bảo tính ổn định và chất lượng của mã nguồn.
* Theo Dõi Và Giải Quyết Vấn Đề: Sử dụng tính năng theo dõi vấn đề trên GitHub giúp chúng em ghi lại và giải quyết các vấn đề hoặc yêu cầu tính năng một cách có tổ chức. Điều này tăng cường sự linh hoạt và phản hồi trong quá trình phát triển.

**Kết Quả Đạt Được**

Sử dụng Git và GitHub đã mang lại những kết quả tích cực cho dự án của chúng em:

* Tăng Cường Hợp Tác Trong Nhóm: Quy trình làm việc phân tán trên Git và GitHub đã tăng cường sự hợp tác và giao tiếp giữa các thành viên trong nhóm phát triển. Điều này đã giúp cải thiện hiệu suất làm việc và chất lượng mã nguồn.
* Kiểm Soát Và Quản Lý Mã Nguồn Hiệu Quả: Sử dụng các tính năng của Git và GitHub, chúng em đã có thể duy trì một mã nguồn ổn định và dễ quản lý. Cơ chế kiểm soát phiên bản và quản lý pull request giúp đảm bảo rằng chỉ những thay đổi chất lượng được hợp nhất vào mã nguồn chính.
* Tăng Hiệu Suất Phát Triển: Quy trình làm việc linh hoạt và hiệu quả trên Git và GitHub đã giúp tăng cường hiệu suất phát triển của dự án. Việc giảm thiểu xung đột và tối ưu hóa quy trình làm việc đã dẫn đến việc hoàn thành các tính năng nhanh chóng hơn và giảm thiểu lỗi phát sinh.

**Hướng Phát Triển**

Dựa trên kết quả và kinh nghiệm tích lũy từ việc sử dụng Git và GitHub trong dự án hiện tại, chúng em đề xuất một số hướng phát triển tiếp theo:

* Tối Ưu Hóa Quy Trình Làm Việc: Tiếp tục đánh giá và tối ưu hóa quy trình làm việc trên Git và GitHub để đảm bảo sự linh hoạt và hiệu quả cao nhất.
* Đào Sâu Vào Các Tính Năng Của GitHub: Nắm vững các tính năng nâng cao của GitHub như Projects, Wiki, và Actions để tối ưu hóa quy trình làm việc và tăng cường hiệu suất.
* Học Hỏi Và Chia Sẻ Kinh Nghiệm: Tổ chức các buổi workshop hoặc buổi đào tạo để chia sẻ kinh nghiệm sử dụng Git và GitHub với các thành viên mới trong nhóm và tổ chức.
* Mở Rộng Sử Dụng Cho Các Dự Án Khác: Áp dụng kiến thức và kinh nghiệm từ dự án hiện tại để mở rộng việc sử dụng Git và GitHub cho các dự án khác trong tổ chức, từ đó tăng cường sự phát triển và hiệu suất làm việc.

# TÀI LIỆU THAM KHẢO

[1] Giáo Trình Và Slide Môn Vận Hành và Bảo Trì Phần Mềm – Phạm Thị Thương

[2] <https://git-scm.com/doc>

[3] <https://rogerdudler.github.io/git-guide/>

[4] <https://chat.openai.com/>

[5] <https://www.youtube.com/watch?v=1EmOFN0oyks>

[6]<https://memart.vn/tin-tuc/blog/tim-hieu-github-la-gi-cach-su-dung-de-quan-ly-ma-nguon-va-cac-du-an-phat-trien-vi-cb.html>

[7] <https://vietjack.com/git/tai_lieu_tham_khao_ve_git.jsp>

[8] <https://freetuts.net/hoc-git>

[9] <https://codelearn.io/sharing/git-github-tu-co-ban-den-nang-cao-p1>