# Chương 2: QUẢN LÝ MÃ NGUỒN VÀ ĐẢM BẢO CHẤT LƯỢNG MÃ NGUỒN

## 2.1 Quản lý mã nguồn

### 2.1.1 Giới thiệu chung về quản lý mã nguồn

Trong quá trình phát triển chương trình phần mềm, sau một thời gian mã nguồn chương trình sẽ ngày càng nhiều, rất khó để lập trình viên có thể kiểm soát được các chức năng đã thực hiện, cũng như quản lý tất cả mã nguồn đã viết ra. Đặc biệt đối với nhóm lập trình thì việc chia sẻ mã nguồn, quản lý công việc giữa các thành viên trong nhóm càng trở nên cấp thiết nhằm không bị chồng chéo công việc cũng như tăng tốc thực hiện dự án. Chính vì thế, việc áp dụng hệ thống quản lý phiên bản mã nguồn sẽ giúp giải quyết tất cả các vấn đề trên, giúp các thành viên trong dự án quản lý công việc, quản lý mã nguồn một cách hiệu quả.

Hệ thống quản lý phiên bản mã nguồn - Version Control System (VCS) là một hệ thống được sử dụng trong việc phát triển phần mềm, giúp lưu trữ và ghi nhận sự thay đổi của các tập tin, mã nguồn theo thời gian, nhờ đó hệ thống có thể theo dõi việc cập nhật sửa đổi mã nguồn của các thành viên trong suốt quá trình làm dự án.

Version control được hiểu như 1 công cụ giúp theo dõi sự thay đổi của toàn bộ cấu trúc chương trình, từ các file code cho đến các file hình ảnh, video. Với version control, ta có thể tạo ra nhiều phiên bản của các file ứng với từng thay đổi mà chính mình tạo ra. Các VCS sẽ lưu trữ tất cả các file trong toàn bộ dự án và ghi lại toàn bộ lịch sử thay đổi của file. Mỗi sự thay đổi được lưu lại sẽ được và thành một version (phiên bản).

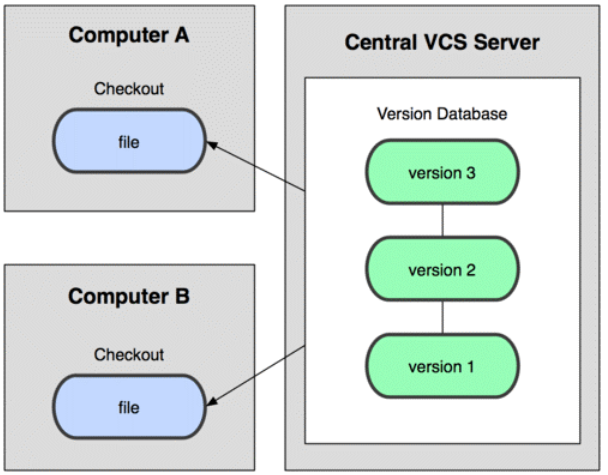
VCS nghĩa là hệ thống giúp lập trình viên có thể lưu trữ nhiều phiên bản khác nhau của một mã nguồn được nhân bản (clone) từ một kho chứa mã nguồn (repository), mỗi thay đổi vào mã nguồn trên local sẽ có thể ủy thác (commit) rồi đưa lên server nơi đặt kho chứa chính. Lập trình viên có thể xem lại danh sách các sự thay đổi của file như xem một dòng thời gian của các phiên bản. Mỗi phiên bản bao gồm: nội dung file bị thay đổi, ngày giờ sửa đổi, người thay đổi là ai, lý do thay đổi hay tên phiên bản

### 2.1.2 Mô hình hoạt động

#### 2.1.2.1 Mô hình quản lý mã nguồn tập trung (Centralized version control – CVC )

Mô hình sử dụng kiến trúc Client-Server. Server đóng vai trò là một kho dữ liệu (Repository) duy nhất lưu trữ tất cả các tập tin, mã nguồn, thay đổi… Các client sẽ kết nối đến server để lấy phiên bản mới nhất của mã nguồn về máy (check out) và cập nhật tất cả các thay đổi lên server (commit). Điển hình cho mô hình này có thể kể tới CVS (Concurrent Versioning System), Prefoce hay SVN (Subversion)

Trong hệ thống kiểm soát phiên bản tập trung (CVCS), một máy chủ hoạt động như một kho lưu trữ chính lưu trữ mọi phiên bản mã. Sử dụng kiểm soát nguồn tập trung, mọi người dùng cam kết trực tiếp với nhánh chính, vì vậy mô hình hoạt động này thường hoạt động tốt cho các nhóm nhỏ, vì các thành viên trong nhóm có khả năng giao tiếp nhanh chóng để không có hai nhà phát triển nào muốn làm việc trên cùng một đoạn mã đồng thời. Giao tiếp và kết nối liên tục mạnh mẽ là rất quan trọng để đảm bảo quy trình làm việc tập trung thành công.

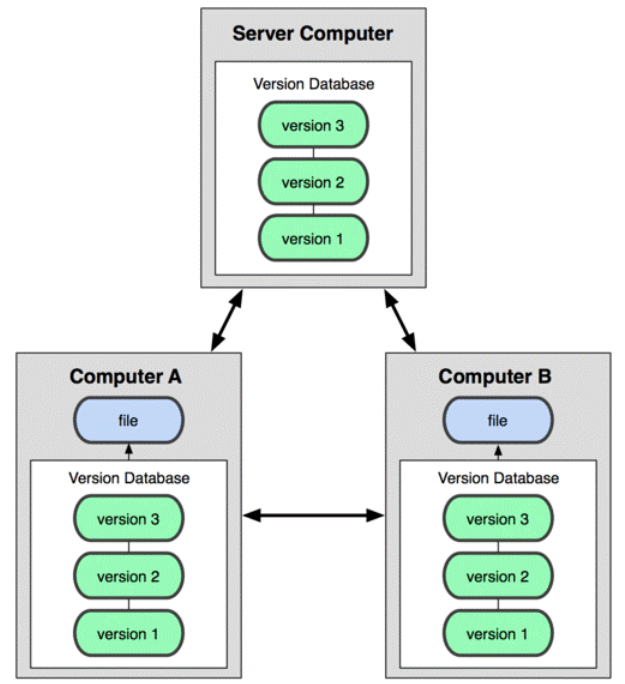


Các hệ thống kiểm soát mã nguồn nguồn tập trung, chẳng hạn như CVS, Perforce và SVN yêu cầu người dùng pull về phiên bản mới nhất từ máy chủ để tải xuống bản sao cục bộ trên máy của họ. Sau đó các lập trình viên viên sẽ push các commits đến máy chủ và giải quyết mọi xung đột hợp nhất trên kho lưu trữ chính.

Được triển khai dưới dạng client – server, quy trình làm việc tập trung cho phép khóa tệp để bất kỳ đoạn mã nào hiện đã được kiểm tra sẽ không thể truy cập được đối với người khác, đảm bảo rằng chỉ một nhà phát triển có thể đóng góp vào mã tại một thời điểm. Các thành viên trong nhóm sử dụng các nhánh để đóng góp vào kho lưu trữ trung tâm và máy chủ sẽ mở khóa các tệp sau khi hợp nhất.

* ***Ưu điểm của mô hình quản lý mã nguồn tập trung***
* Hoạt động tốt với các tệp nhị phân: Các tệp nhị phân, ví dụ các file dạng image hoặc văn bản, yêu cầu dung lượng lớn, vì vậy các nhà phát triển phần mềm chuyển sang hệ thống kiểm soát phiên bản tập trung để lưu trữ dữ liệu này. Với một máy chủ tập trung, các nhóm có thể kéo một vài dòng mã mà không cần lưu toàn bộ lịch sử trên máy cục bộ của họ. Người dùng của các hệ thống phân tán phải tải xuống toàn bộ dự án, điều này chiếm thời gian và không gian và ngăn họ làm sai khác. Nếu một nhóm làm việc với các tệp nhị phân thường xuyên, một hệ thống tập trung cung cấp cách tiếp cận hiệu quả nhất cho các nhà phát triển.
* Cung cấp tầm nhìn đầy đủ cho dự án: Với vị trí tập trung, mọi thành viên trong nhóm đều có toàn bộ khả năng hiển thị mã nào hiện đang được thực hiện và những thay đổi nào được thực hiện. Điều giúp các nhóm phát triển phần mềm hiểu được trạng thái của một dự án và cung cấp nền tảng cho sự cộng tác, vì các nhà phát triển chia sẻ công việc trong máy chủ trung tâm.
* Dễ hiểu và dễ sử dụng: Hệ thống quản lý tập trung dễ dàng sử dụng hơn phi tập trung, chính vì thế nó dễ dàng và phù hợp với lập trình viên ở mọi trình độ khác nhau. Điều này cũng giúp ít xảy ra các xung đột trong việc merge các mã nguồn lại với nhau.
* ***Nhược điểm của mô hình quản lý mã nguồn tập trung***
* Rủi do về dữ liệu: Do chỉ có một máy chủ lưu trữ dữ liệu và mọi thay đổi đều commit lên máy chủ nên khi xảy ra sự cố với máy chủ tập trung hoặc với máy tính cá nhân rất dễ bị mất dữ liệu. Trường hợp xấu nhất hoàn toàn có thể xảy ra đó là mất toàn bộ mã nguồn của dự án.
* Tốc độ chậm hơn và có thể ảnh hưởng tới tiến độ của dự án: Tốc độ vận hành của các nhóm phát triển phần mềm có ảnh hưởng trực tiếp đến tốc độ họ có thể gửi các tính năng và mang lại giá trị kinh doanh. Nếu các nhóm chậm phát triển, quá trình lặp lại và đổi mới bị đình trệ và các nhà phát triển có thể trở nên thất vọng với việc mất bao lâu để thấy những thay đổi của họ trong ứng dụng.
* Sự kết hợp với cùng vấn đề bị giảm: CVCS phù hợp với những dự án nhỏ và ít người. Bởi mỗi lập trình viên sẽ thực hiện một module của dự án và các lập trình viên khác không thể cùng thực hiện ( không thể cùng push). Điều này cũng ảnh hường nhiều tới tốc độ phát triển của dự án.

#### 2.1.2.2 Mô hình quản lý mã nguồn phân tán (Distributed version control – DVC)



Hệ thống kiểm soát phiên bản phân tán (DVCS) là một loại kiểm soát phiên bản trong đó cơ sở source code hoàn chỉnh - bao gồm cả lịch sử phiên bản đầy đủ của nó - được sao chép trên mọi máy tính của nhà phát triển.

Hệ thống kiểm soát phiên bản phân tán (Distributed Version Control System - DVCS) mang một bản sao cục bộ của kho lưu trữ hoàn chỉnh đến máy tính của mọi thành viên trong nhóm để họ có thể thực hiện Commit, phân nhánh và hợp nhất cục bộ. Máy chủ không phải lưu trữ tệp vật lý cho mỗi nhánh - nó chỉ cần sự khác biệt giữa mỗi commit. So với kiểm soát phiên bản tập trung, DVCS phân nhánh và hợp nhất quản lý tự động, tăng tốc hầu hết các hoạt động, cải thiện khả năng làm việc ngoại tuyến cho lập trình viên và không dựa vào một vị trí duy nhất để sao lưu mã nguồn.

Khi client lấy mã nguồn phiên bản mới nhất (clone) từ server về máy thì hệ thống sẽ sao chép luôn cả kho mã nguồn (repository), vì thế máy client lúc này sẽ đóng vai trò là kho mã nguồn (repository), mỗi kho con sẽ đồng bộ các thay đổi đến kho trên server chính. Ưu điểm của mô hình này là sự linh hoạt trong việc cập nhật lại mã nguồn trên server, client không cần lúc nào cũng phải duy trì kết nối đến server.

Git, Mercurial và Bazaar là một trong số những hệ thống quản lý mã nguồn phân tán phổ biến nhất.

* ***Ưu điểm của mô hình quản lý mã nguồn phân tán***
* Cung cấp quy trình làm việc và hệ thống phân cấp mạnh mẽ, các lập trình viên triển push các thay đổi mã vào kho lưu trữ của riêng họ và những người bảo trì thiết lập quy trình xem xét mã để đảm bảo chỉ có mã chất lượng hợp nhất vào kho lưu trữ chính.
* Đảm bảo sự phát triển ổn định của dự án, nhiều lập trình viên có thể tham gia đóng góp phát triển. Hệ thống tỏ ra cực kì hiệu quả với các dự án từ nhỏ tới lớn
* Các bản sao lưu đáng tin cậy: Mỗi máy tính khi pull project về đều có bản sao lưu ở trên chính máy tính đó. Có thể thực hiện các hoạt động commit ngay cả khi offline.
* Hợp nhất nhanh chóng và phân nhánh linh hoạt. Hỗ trợ rất nhanh việc hợp nhất mã nguồn, phân nhánh công việc cho từng team, từng thành viên làm việc.
* Phản hồi nhanh tróng và ít xảy ra xung đột mã nguồn
* Linh hoạt trong làm việc ngoại tuyến.
* **Nhược điểm của mô hình quản lý mã nguồn phân tán**
* Với dự án có lượng tài nguyên lớn ( các file ảnh, text lớn ) thì cần nhiều dung lượng lưu trữ hơn.
* Nếu dự án lớn, có nhiều tệp và file thời gian merge, lưu trữ có thể mất thời gian lớn
* Vẫn cần có một máy chủ trung tâm để kiểm soát lịch sử của các phiên bản trong dự án với nhiều thành viên
* Hoàn toàn có thể xảy ra conflict trong khi hợp nhất mã nguồn
* Cần phải trả phí hoặc duy trì máy chủ làm repository

### 2.1.3 Lợi ích của việc quản lý mã nguồn

* ***Lưu trữ mã nguồn tập trung***

Lợi ích chính của hệ thống quản lý mã nguồn là tạo ra môi trường lưu trữ tập tin, mã nguồn. Tuy nhiên không như cách lưu trữ truyền thống, hệ thống quản lý mã nguồn sẽ lưu trữ tập trung và giải quyết được các vấn đề dư thừa dữ liệu cũng như tìm kiếm nội dung lưu trữ.

Như ở ví dụ trên mã nguồn chương trình được lưu trữ trong nhiều tập tin nén khác nhau. Số lượng, dung lượng tập tin lưu trữ sẽ tăng trưởng theo thời gian thực hiện dự án.

Việc lưu trữ truyền thống như ví dụ trên sẽ làm dữ liệu trở nên dư thừa, tốn dung lượng lưu trữ cùng một tập tin giống nhau nhiều lần. Nhiều phiên bản khác nhau được đặt tên không theo qui cách cũng sẽ gây nhiều khó khăn trong việc tìm kiếm, xác định phiên bản, và không thể xác định được có thay đổi gì trong mã nguồn.

* ***Tạo môi trường làm việc nhóm***

Hệ thống quản lý mã nguồn sẽ tạo ra môi trường làm việc tập trung cho các thành viên tham gia dự án. Với cách làm truyền thống, mỗi người trong dự án làm việc độc lập lập, phải sao chép một bộ mã nguồn, phải thường xuyên liên hệ thông báo cho nhau mình đang chỉnh sửa phần nào trong mã nguồn, rất dễ xảy ra nhầm lẫn, sai sót, chỉnh sửa trùng hoặc xóa phần chỉnh sửa của nhau.

Hệ thống quản lý mã nguồn sẽ khắc phục tất cả các vấn đề tồn tại với cách làm truyền thống; tạo ra môi trường để mọi thành viên trong nhóm có thể tương tác, phối hợp với nhau, tránh việc thực hiện công việc trùng lặp, chồng chéo lẫn nhau. Người quản lý có thể quản lý mã nguồn, điều hành, tổng hợp công việc của nhóm. Lập trình viên tham gia các nhóm làm dự án có thể quản lý mã nguồn của mình, chia sẻ cũng như tiếp nhận mã nguồn từ đồng nghiệp.

* ***Lưu vết các thay đổi trong mã nguồn***

Không chỉ lưu trữ tập tin, mã nguồn, hệ thống còn cho phép lưu trữ các thay đổi trong quá trình thực hiện dự án. Các thay đổi được lưu lại không chỉ là tạo mới, xóa tập tin mà còn nội dung bị thay đổi trong tập tin mã nguồn.

Mỗi lần có các thay đổi trong mã nguồn dự án, hệ thống quản lý mã nguồn sẽ cung cấp thông tin về mục đích thay đổi, phần nào của mã nguồn bị thay đổi, người cập nhật… Các lần thay đổi sẽ được lưu trữ trong suốt quá trình làm dự án, cho phép quay ngược lại phiên bản trước đó nếu có phát sinh lỗi hoặc các vấn đề không mong muốn trong mã nguồn.

Có được một kho dữ liệu lịch sử, lưu vết các lần thay đổi trong mã nguồn sẽ giúp người quản lý theo dõi tiến trình làm dự án, giúp lập trình viên dễ tiếp cận, dễ tìm hiểu về mã nguồn khi được giao thực hiện dự án. Tất cả lịch sử thay đổi đều cố thể được đưa vào Hệ thống quản lý dự án như DevOps, JIRA… để quản lý tiến độ dự án cũng như thực hiện các phân tích khác.

### 2.1.4 Git & Github

#### 2.1.4.1 Tổng quan về Git

Git là một hệ thống quản lý phiên bản phân tán (Distributed Version Control System – DVCS) được phát triển bởi Linus Torvalds vào năm 2005, nó là một trong những hệ thống quản lý phiên bản phân tán phổ biến nhất hiện nay. Git cung cấp cho mỗi lập trình viên kho lưu trữ (repository) riêng chứa toàn bộ lịch sử thay đổi.

***Phân nhánh và hợp nhất (Branching and Merging)***

Git cho phép và khuyến khích lập trình viên làm việc tại các nhánh cục bộ có thể hoàn toàn độc lập với nhau. Việc tạo, hợp nhất và xóa các dòng phát triển đó chỉ mất vài giây. Điều này cho phép lập trình viên:

* Triển khai ý tưởng mới mà không sợ ảnh hưởng tới mã nguồn cũ trước đó
* Các branch được phân chia vai trò rõ ràng ( 1 nhánh chính productions và các nhánh phụ dùng để phát triển
* Dễ dàng thử nhiệm các tính năng, nếu không phù hợp, không hoạt động chỉ cần xóa bỏ chúng

***Nhỏ và nhanh (Small and Fast)***

Hầu như tất cả các hoạt động trên Git đều được thực hiện cục bộ, mang lại lợi thế về tốc độ rất lớn cho các hệ thống tập trung liên tục phải giao tiếp với máy chủ ở đâu đó.

Git được xây dựng để hoạt động trên nhân Linux nên rất hiệu quả trong việc xử lý luồng dữ liệu lớn. Git được viết bằng ngôn ngữ C, giảm chi phí thời gian chạy liên quan đến các ngôn ngữ cấp cao hơn.

***Là một hệ thống phân tán***

Phân tán là một trong những tính năng nổi bật nhất nhất của bất kỳ VCS tán nào, bao gồm Git, là nó được phân phối. Thay vì chỉ làm việc với một phần của dự án thì Git cho phép người dùng clone toàn bộ dự án về và thao tác trên đó.

***Tính toàn vẹn dữ liệu***

Mô hình dữ liệu mà Git sử dụng đảm bảo tính toàn vẹn của mật mã cho từng bit trong dự án của người dùng. Mọi tệp và commit đều được tổng kiểm tra và truy xuất bởi tổng kiểm tra của nó khi kiểm tra lại. Không thể lấy bất cứ thứ gì ra khỏi Git ngoài các bit chính xác mà người dùng đã đưa vào. Ngoài ra nhờ có các bản sao lưu dữ liệu local, dự án đảm bảo không bị mất mát dữ liệu.

***Git Free và Open Source***

Git là mã nguồn mở và hoàn toàn miễn phí đối với tất cả người dùng.

***Cơ chế hoạt động của git:***

* Git sẽ quản lý dữ liệu bằng tập hợp các "ảnh" (snapshot) của một hệ thống tập tin.
* Mỗi lần người dùng lưu lại trạng thái hiện tại của dự án, về cơ bản Git "chụp một bức ảnh" ghi lại nội dung của tất cả các tập tin tại thời điểm đó và tạo ra một tham chiếu tới "ảnh" đó.
* Để hiệu quả hơn, nếu như tập tin không có sự thay đổi nào, Git không lưu trữ tập tin đó lại một lần nữa mà chỉ tạo một liên kết tới tập tin gốc đã tồn tại trước đó.

#### 2.1.4.2 Một số thành phần của Git

* **Repository**

Trong hệ thống quản lý phiên bản, một repository là một cấu trúc dữ liệu giúp lưu trữ metadata cho một tập hợp các file hoặc cấu trúc thư mục. Đây chính là nơi chứa tất cả mã nguồn cho một dự án được quản lý bởi Git. Mỗi repo sẽ có hai cấu trúc dữ liệu chính đó là Object store và Index được lưu trữ ẩn trong thư mục .git

Có lại loại repository đó là local repository và remote repository.

* Local repository: Là repo được cài đặt trên máy tính của lập trình viên, repo này sẽ đồng bộ hóa với remote repo bằng các lệnh của git.
* Remote repository: Là repo được cài đặt trên server chuyên dụng, điển hình hiện nay là Github, Gitlab, Bitbucket
* **Branch**

Branch là cái dùng để phân nhánh và ghi lại luồng của lịch sử. Branch đã phân nhánh sẽ không ảnh hưởng đến branch khác nên có thể tiến hành nhiều thay đổi đồng thời trong cùng 1 repository. Các hoạt động trên mỗi branch sẽ không ảnh hưởng lên các branch khác nên có thể tiến hành nhiều thay đổi đồng thời trên một repository, giúp giải quyết được nhiều nhiệm vụ cùng lúc.

* **Commit**

Git Commit là một lệnh cam kết trong Git, được dùng để lưu lại những thay đổi trong repository bằng các ảnh chụp nhanh (snapshot). Có thể hiểu Commit giúp trình viên đánh dấu, lưu lại lại các thay đổi của mình với mã nguồn.

#### 2.1.4.3 Ưu điểm của Git

***Vì sao nên sử dụng git?***

Các dự án thực tế thường có nhiều lập trình viên làm việc song song. Vì vậy, một hệ thống kiểm soát phiên bản như Git là cần thiết để đảm bảo không có xung đột code giữa các lập trình viên.

Ngoài ra, các yêu cầu trong các dự án như vậy thay đổi thường xuyên. Vì vậy, một hệ thống kiểm soát phiên bản cho phép các nhà phát triển revert và quay lại phiên bản cũ hơn của code.

Cuối cùng, đôi khi một số dự án đang được chạy song song liên quan đến cùng một cơ sở code. Trong trường hợp như vậy, khái niệm phân nhánh trong Git là rất quan trọng.

***Các ưu điểm của Git:***

* Dễ sử dụng, thao tác nhanh, gọn, lẹ và rất an toàn.
* Sễ dàng kết hợp các phân nhánh (branch), có thể giúp quy trình làm việc code theo nhóm đơn giản hơn rất nhiều.
* Chỉ cần clone mã nguồn từ kho chứa hoặc clone một phiên bản thay đổi nào đó từ kho chứa, hoặc một nhánh nào đó từ kho chứa là bạn có thể làm việc ở mọi lúc mọi nơi.
* Deployment sản phẩm của bạn một cách không thể nào dễ dàng hơn.
* Sắp xếp công việc dễ dàng và đơn giản hơn
* Linh hoạt hơn khi sử dụng một lúc nhiều task
* Git cho phép lập trình viên hoạt động offline
* Git hoàn toàn miễn phí

Cách lưu trữ thông minh khiến việc merge nhanh và chính xác, tối ưu được việc quản lý phiên bản

#### 2.1.4.4 Github

GitHub là một dịch vụ cung cấp kho lưu trữ trực tuyến (Remote Repository) mã nguồn Git dựa trên nền web cho các dự án phát triển phần mềm. GitHub cung cấp cả phiên bản trả tiền lẫn miễn phí cho các tài khoản.

GitHub là một dịch vụ nổi tiếng cung cấp kho lưu trữ mã nguồn Git cho các dự án phần mềm. Github có đầy đủ những tính năng của Git, ngoài ra nó còn bổ sung những tính năng về social để các developer tương tác với nhau.

**Một số ưu điểm của Github**

* Dịch vụ miễn phí, mặc dù nó cũng có các dịch vụ trả phí.
* Tìm kiếm rất nhanh trong cấu trúc repos.
* Cộng đồng lớn và dễ dàng tìm thấy sự giúp đỡ.
* Cung cấp các công cụ thiết thực để hợp tác và tích hợp tốt với Git.
* Dễ dàng tích hợp với các dịch vụ của bên thứ ba khác.
* Nó cũng hoạt động với TFS, HG và SVN.

**Nhược điểm của Github**

Github không phải là hoàn toàn mở.

* Github có giới hạn về dung lượng, vì bạn không thể vượt quá 100MB trong một tệp duy nhất, trong khi kho lưu trữ được giới hạn ở 1GB trong phiên bản miễn phí.

#### 2.1.4.5 Các bước cơ bản để triển khai một dự án với Git và github

**Bước 1:** Tạo mới một repository trên Github dùng làm remote Repository, sau đó lấy thông tin kết nối tới repository.

**Bước 2:** Tạo Local repository với git bằng câu lệnh **git init**

**Bước 3:** Thêm toàn bộ các tài nguyên của dự án vào git với câu lệnh **git add .**

**Bước 4:** Thực hiện commit các file vừa thêm ở bước trên bằng câu lệnh **git commit – m “message”**

**Bước 5:** Thêm remote bằng câu lệnh **git remote add** để có thể push lên remote repository đã tạo ở bước 1.  
Ví dụ: **git remote add origin https://github.com/user/repository.git**

**Bước 6:** Push commit từ local repository lên remote repository với câu lệnh: **git git push origin [name\_of\_your\_new\_branch]**

**Một số câu lệnh thường dùng khác trong quá trình sử dụng Git:**

git merge [branch\_name] : Dùng để merge branch hiện tại với branch\_name

git checkout [branch\_name] : Chuyển tới branch\_name

git log : Kiểm tra các commit đã commit trước đó

git pull : Lấy về thông tin từ remote và cập nhật vào các nhánh của local repo.

git fetch: Lệnh git fetch tải về dữ liệu từ Remote Repo nhưng không thực hiện merge ngay vào cách nhánh của local repository

git clone [repository\_link] : clone toàn repository remote về local