[GitHub](https://github.com/) là một dịch vụ nổi tiếng cung cấp kho lưu trữ mã nguồn [Git](https://topdev.vn/blog/git-la-gi/) 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.

* Là công cụ giúp **quản lý source code** tổ chức theo **dạng dữ liệu phân tán**.
* Giúp **đồng bộ source code** của team lên 1 server.
* Hỗ trợ các thao tác **kiểm tra source code** trong quá trình làm việc (diff, check modifications, show history, merge source, …)

## 1 Một vài khái niệm của Git bạn cần nắm

* **git**: là prefix của các lệnh được sử dụng dưới CLI
* **branch**: được hiểu như là nhánh, thể hiện sự phân chia các version khi 2 version đó có sự sai khác nhất định và 2 version đều có sự khác nhau.
* **commit**: là một điểm trên cây công việc (Work Tree ) hay gọi là cây phát triển công việc
* **clone**: được gọi là nhân bản, hay thực hiện nhân bản. Sử dụng để clone các project, repository trên các hệ thống chạy trên cơ sở là git, ví dụ như: bitbucket, github, gitlab, cor(1 sản phẩm mã nguồn mở cho phép người dùng tự tạo git server cho riêng mình trên vps, server),… Việc clone này sẽ sao chép repository tại commit mình mong muốn, dùng để tiếp tục phát triển. Thao tác này sẽ tải toàn bộ mã nguồn, dữ liệu về máy tính của bạn.
* **folk**: Folk là thao tác thực hiện sao chép repository của chủ sở hữu khác về git account của mình. sử dụng và đối xử như 1 repository do mình tạo ra.
* **repository**: Kho quản lý dữ liệu, là nơi lưu trữ các dữ liệu, mã nguồn của project.
* **tag**: sử dụng để đánh dấu một commit khi bạn có quá nhiều commit tới mức không thể kiểm soát được.
* **remote**: sử dụng để điều khiển các nhánh từ một repository trên git server, đối xử với các nhánh trên remote tương tự như đối xử với các nhánh trên local
* **.gitignore:** file mặc định của git sử dụng để loại bỏ (ignore) các thư mục, file mà mình không muốn push lên git server

### **Quản lý source code dễ dàng**

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

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

### **Tracking sự thay đổi qua các version**

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

### **Github giúp cải thiện kỹ năng code, thậm chí là tracking bug**

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

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

### **Github là một kho tài nguyên tuyệt vời**

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

## 2. Tại sao bạn nên dùng Git?

### **1. Git theo dõi mọi sự thay đổi của các tập tin**

Thật vậy Git là VCS – Version Control System hệ thống quản lý phiên bản, việc theo    dõi mọi sự thay đổi của các tập tin là một trong những tính năng cơ bản nhất của Git

### **2. Theo dõi sự thay đổi của các người dùng trên các tập tin**

Khi bạn chia nhỏ một dự án thành nhiều task cho các thành viên khác nhau thực hiện, là một Leader bạn cần biết rõ ai là người Code những phần nào, sửa những phần nào… để đánh giá năng lực của từng người. Thật tuyệt vời khi Git đã xử lý toàn bộ những vấn đề ở trên phải không.

### **3. Phục hồi trạng thái của các tập tin vào một thời điểm nào đó**

Hãy tưởng tượng bạn Code khiến dự án bị lỗi không thể hoạt động, hoặc một thành viên nào đó lỡ tay xóa một vài tập tin trong dự án, bạn yên tâm Git sẽ đưa bạn về quá khứ để khôi phục lại những tập tin đó, thật tuyệt vời phải không.

### **4. Hỗ trợ làm việc Offline**

Đối với các VCS bạn cần phải có máy tính kết nối tới server để làm việc nhóm hoặc commit sự thay đổi của các tập tin, nhưng đối với Git bạn hoàn toàn có thể làm việc mà không có kết nối Internet, khi nào có kết nối mạng bạn chỉ cần đẩy lên server thế là xong.

### **5. Hỗ trợ rất tốt trong môi trường làm việc nhóm**

Còn điều gì tuyệt vời hơn khi mà bạn có thể chia nhỏ từng thành phần, từng công việc cho các thành viên trong nhóm mà không sợ bị rối hay lỗi Code.

### **6. Một phần trong yêu cầu tuyển dụng của các doanh nghiệp CNTT hiện nay**

Chà chà nếu bạn đang xin việc trong ngành lập trình, ắt hẳn bạn sẽ thấy nhan nhản đơn tuyển dụng yêu cầu phải nắm được Git, nếu biết chút ít hoặc Pro Git bạn sẽ có lợi thế hơn hẳn so với các ứng viên khác.

### **7. Cộng đồng hỗ trợ và sử dụng rộng lớn**

Hiện nay Git đã chiếm 74% thị phần VCS, với thị phần to lớn dẫn tới cộng đồng sử dụng Git cũng lớn theo, với việc cộng đồng hỗ trợ và sử dụng rộng lớn sẽ giúp bạn tìm kiếm được sự giúp đỡ dễ dàng hơn rất nhiều so với các VCS khác, bạn thì sao còn mình luôn lựa chọn những phần mềm hoặc hệ thống được người sử dụng để làm việc cho dễ.

## 3. Write a good commit

Giúp sau một thời gian quay lại dự án dựa vào commit tìm bug sẽ dễ hơn.

Tìm lại những gì đã làm dễ dàng hơn.

Tăng tốc quá trình review.

Lưu ý: message không quá 50 ký tự

git commit -m "[HÀNH ĐỘNG CHUNG] - [NỘI DUNG CHI TIẾT] - [LƯU Ý NẾU CÓ]"

## 4. What could cause code conflicts and produce a checklist of possible causes

* + Merge code
  + Xảy ra khi tích hợp các commit từ các nguồn khác
  + Pull, cherry pick, stash apply, rebase.

### Git cherry-pick

Là một cách để checkout 1 commit bất kỳ tại 1 branch được chỉ định về branch hiện tại. Hay chính là git cherry-pick sẽ bốc thay đổi của 1 commit trên 1 nhánh nào đó áp dụng vào nhánh hiện tại.

Lấy 1 commit từ 1 brach bỏ vào master

Lấy n commits từ 1 brach bỏ vào master

1 lần commit cho cả 2 branches\

### Git stash save | Lưu lại công việc đang làm dở

Bạn đang code file awesome.vue nhưng **chưa muốn commit** và đang muốn chuyển sang một branch khác, git sẽ yêu cầu bạn commit những thay đổi trước khi chuyển branch. Câu lệnh này cũng giống như Git stash nhưng bạn có thể thêm vào nhiều tùy chọn đi cùng với nó. Mình có thể coi Git stash == Git stash save (khi không có tùy chọn nào).

## 5. Khi nào cần dùng merge

Git Merge là một lệnh dùng để hợp nhất các nhánh độc lập thành một nhánh duy nhất trong Git.

Khi sử dụng lệnh hợp nhất trong Git, chỉ có nhánh hiện tại được cập nhật để phản ánh sự hợp nhất, còn nhánh đích sẽ không bị ảnh hưởng.

Dùng kết hợp với git checkout

## 6. Branch được tạo từ master

## 7. Working directory

File thư mục làm việc hiện tại ghi lại những thay đổi

File ở staging area mới được commit sau lệnh git add

Git repository lưu những thay đổi commit

## 8. Checkout – tên file và git reset Head

Git checkout – tên file có thể loại bỏ file khi chưa lên staging

Git reset head tên file có thể loại bỏ file ở staging area

## 9. Branching and merging

Git checkout -b branch có thể tạo 1 nhánh mới

Git merge A->B

Checkout sang B

## 10. get reset

-- Soft quay lại trạng thái staging area sau khi nhấn add

-- mixed quay lại trang thái ở working directory

-- reset mất luôn trạng thái thay đổi

## 11. Git revert

Bỏ commit cũ, nguy hiểm khi những commit khác xử lí trên cùng một file nên tạo commit mới sửa thay đổi đó

## 12. .Gitignore

Qua qua những thứ không cần commit nên igonre trước khi commit vì phải xử lí nhiều thứ

13. github

Lưu trữ lịch sử commit chung giữa các user thay vì phải gửi tới n-1 người khi có sự thay đổi và cần những ngừ khác biết

## 13. Git remote add origin

## 14 Git credential

Lưu lại thông tin đăng nhập không phải đăng nhập lại những lần sau

## 15. Git clone & pull

- Git clone copy nguyên bản repo trên github

- git pull

## 16. git merge

Git Merge là một lệnh dùng để hợp nhất các chi nhánh độc lập thành một nhánh duy nhất trong Git.

Khi sử dụng lệnh hợp nhất trong Git, chỉ có nhánh hiện tại được cập nhật để phản ánh sự hợp nhất, còn nhánh đích sẽ không bị ảnh hưởng.

## 16.Git rebase

git rebase sữa đổi một commit trước có thể giải quyết xung đột

## Git Pull là gì?

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

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

## 1. Lệnh git fetch là gì?

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

## 2. ****Sự khác biệt giữa git pull và git fetch là gì?****

Trong quá trình làm việc với git, nhiều engineer hay bị rối và nhầm lẫn hai lệnh git fetch và git pull bởi cả hai đều được sử dụng để tải về remote content. Tuy nhiên, git fetch được coi là phiên bản ‘an toàn’ hơn của git pull. Khi sử dụng, lệnh này sẽ tải xuống remote content mà không cập nhật trạng thái hoạt động của local repository. Từ đó, nội dung công việc hiện tại của bạn không bị ảnh hưởng. Ngược lại, lệnh git pull sẽ tải xuống remote content và ngay lập tức thực hiện [git merge](https://vn.got-it.ai/blog/git-merge-la-gi-nhung-kien-thuc-co-ban-ve-git-merge) để merge commit cho remote content mới. Nếu bạn có các thay đổi đang chờ xử lý, điều này có thể vô tình khiến local repository rơi vào trạng thái xung đột.

Ngoài ra, git fetch thường được dùng trong trường hợp có nhiều người làm việc trên cùng một nhánh. Còn git pull chỉ nên sử dụng khi có một người làm việc trên nhánh để hạn chế xung đột. Bạn chỉ nên dùng lệnh git pull trên một thư mục làm việc sạch (không có thay đổi đã cam kết).

## Git rebase và git merge

Ban sử dụng git rebase nếu như bạn muốn các sự thay đổi thuộc về branch của bạn luôn luôn là mới nhất. Và bạn có thể log một cách có hệ thống dễ nhìn, dễ tracking sau này.

Bạn sử dụng git merge nếu bạn muốn sắp xếp các commit theo mặc định. Bạn không biết về những gì mình làm gì trên branch đó thì dùng merge cho đảm bảo việc tracking sao này có thể tốn nhiều thời gian lần mò.

### Một số vấn đề cần lưu ý sau:

Git rebase thì nên dùng trên branch riêng, nó sẽ đẩy history commit của branch lên, history commit sẽ tách biệt hẳn với những commit từ branch khác, rất tiện cho quản lý các branch. Đặt biệt khi các bạn có các branch master / develop / hot-fix / features / release …

Cả rebase và merge sẽ conflict kinh khủng hơn nếu không update code thường xuyên chứ không phải chỉ có rebase như mọi người thường nói đâu nhé. Ví dụ: Nếu như master branch có time line hơn branch của bạn 1 tháng . Lúc đó hãy rebase hay merge branch của bạn và sẽ thấy conflict 2 cái có khác gì nhau!

 Git merge là làm cho git commit list dài ra áp dụng cho branch riêng thì không phù hợp vì khó trace log vì nhiều commit dài thòn không phải do bạn tạo ra!?. Nhất là trong 1 dự án dài hơi, việc nhìn lại log của vài tháng trước có thể sẽ là vấn đề trong bầu trời đầy sao chổi với bạn.