GIT VÀ GITHUB CƠ BẢN

Source control là gì

Source control (một số nơi gọi là version control) là hệ thống quản lý mã nguồn giúp tăng hiệu suất công việc lên nhiều lần. Có rất nhiều hệ thống quản lý mã nguồn khác nhau, nổi tiếng nhất là Git, Mercurial, SVN và được áp dụng gần như trong toàn bộ các project thực tế hiện nay.

Lợi ích khi áp dụng source control trong project

Đặt một số tình huống thường gặp như sau khi các bạn làm project theo nhóm nhưng không áp dụng source control:

- Tình huống 1: project của team bao gồm 2 file là file A và file B và được lưu ở Google Drive. Một ngày nọ, bạn thêm tính năng vào file A, sau đó bạn upload file A mới lên Google Drive (gọi là file A phiên bản 2). Sau đó bạn phát hiện file A phiên bản 2 có lỗi và muốn quay lại phiên bản trước đó nhưng lúc này phiên bản trước đó đã không còn nữa rồi. Nếu như bạn cho rằng Google Drive có lưu lịch sử các phiên bản thì cũng nên nhớ rằng Google Drive chỉ lưu lịch sử 30 ngày thôi, giả sử bạn gặp lỗi vào ngày 31 thì sao?
- Tình huống 2: một ví dụ khác rất hay gặp đó là khi ghép code các phần của mọi người trong team với nhau. Giả sử project của team hiện tại đang có mỗi file A và được lưu ở Google Drive. Thành viên 1 viết tính năng mới vào file B và chỉnh vài nội dung trong file A rồi sau đó up file B và file A (phiên bản 2) lên Google Drive. Google Drive lúc này sẽ lưu file A (phiên bản 2) và file B. Thành viên 2 viết tính năng khác vào file C và cũng chỉnh nội dung file A phiên bản 1 (do cả 2 thành viên cùng phải làm chứ ở đây không có chuyện người này ngồi chờ người kia xong việc rồi mới làm nha). Nhưng khi upload file C và file A lên thì sẽ gặp tình trạng sau: thành viên 2 không biết trước đó file A đã bị chỉnh sửa bởi thành viên 1 rồi, nếu upload thẳng lên sẽ bị ghi đè nội dung chỉnh sửa của thành viên 1 gây ra lỗi. Lúc này thành viên

2 phải tốn công đi hỏi lại thành viên 1, cả 2 phải cùng ngồi bàn với nhau đã chỉnh những gì để tự tay ghép code thủ công từng đoạn code một rất tốn thời gian.

Những ví dụ trên các bạn có thể cho rằng nó khá nhỏ, nhưng nên nhớ ví dụ trên chỉ lấy project có 3 file và team chỉ có 2 người. Hãy tưởng tượng project có rất nhiều, hàng chục, hàng trăm file và team rất đông người thì những ví dụ trên sẽ trở nên phức tạp. Lấy tình huống 2 làm ví dụ: team có 10 người, mỗi người cùng lao vào chỉnh 1 file A, người này xóa tí code, người kia thêm tí code, sau đó 10 người cùng upload file A mới của họ lên Google Drive, lúc này để ghép code ổn định thì 10 người phải ngồi xem xét lại nên gắn code nào vào, nên bỏ code nào đi rồi đủ vấn đề phát sinh. Và, nếu đó là 10 người cùng chỉnh 10 file thì sao? Chắc các bạn cũng hình dung được mức độ mệt mỏi khi cả team ghép code rồi.

Nhưng, source control ra đời nhằm khắc phục những tình trạng trên. Với tình huống 1, source control (nhấn mạnh tên khác là version control), có thể quản lý được toàn bộ các phiên bản, toàn bộ những lần thay đổi của project. Cho dù bạn có thêm 1 ký tự vào file thì source control cũng lưu lại. Với mỗi lần bạn thay đổi 1 file nào đó, hay thêm 1 file, xóa 1 file, sau khi bạn xác nhận (commit) sự thay đổi mới thì source control sẽ lưu lại lịch sử của file trước đó, và cho dù file đó bị xóa thì nó vẫn lưu lại nội dung file đã bị xóa nhằm trường hợp sau này bạn có cần dùng lại. Với tình huống 2, khi thành viên 1 thay đổi file A rồi cập nhật thay đổi, thì khi thành viên 2 thay đổi file A và cập nhật thì lúc cập nhật source control sẽ kiểm tra xem có được phép cập nhật lên hệ thống chung hay không. Ví dụ file A có nội dung là “123”, thành viên 1 đổi file A **gốc** thành “1234” (thêm ký tự 4) và cập nhật, thành viên 2 đổi file A **gốc** thành “12” (bỏ ký tự 3) rồi cập nhật, thì ngay lúc thành viên B bấm cập nhật, source control nhận ra rằng có sự bất thường ở chỗ thành viên 1 thì muốn thêm còn thành viên 2 thì muốn xóa, source control sẽ ngay lập tức ngưng cập nhật, cảnh báo đụng độ (conflict) và lúc này thành viên 2 sẽ được source control hỏi để xác nhận cập nhật một cách an toàn như:

- Thằng 1 nó muốn thêm ký tự 4 nhưng mày lại không có thêm, mày có đồng ý cho nó thêm không?
- Mày xóa ký tự 3 nhưng thằng 1 nó lại không xóa, mày có muốn xóa không hay giữ lại?

Tác giả: Hứa Anh Minh – Trường Đại học Khoa Học Tự Nhiên Thành phố Hồ Chí Minh

- Hai thằng bây chỉnh mệt quá, giờ muốn giữ hết hay khỏi giữ cái nào (giữ nguyên bản gốc).

Source control sẽ đưa ra cho bạn các lựa chọn nhằm giúp đảm bảo an toàn trong việc ghép code và tránh các sai sót không cần thiết.

Các loại source control

Có 2 loại source control chính như sau:

- Centralized version control system: hệ thống quản lý mã nguồn tập trung. Tập trung ở đây có nghĩa là toàn bộ các chức năng và sức mạnh của source control tập trung ở một server duy nhất. Người dùng để có thể truy cập được đến source code thì cần phải có mạng để kết nối trực tiếp đến server của source control đang chứa mã nguồn, nói ngắn gọn là quyền lực tập trung ở một nguồn như chế độ phong kiến ở châu Á. Một trong những hệ thống source control nổi tiếng nhất áp dụng tư tưởng này đó là SVN.
- Distributed version control system: hệ thống quản lý mã nguồn phân tán. Trái với hệ thống tập trung thì ở hệ thống phân tán, sức mạnh của source control nằm ở mỗi PC của các thành viên trong nhóm chứa project. Mỗi thành viên sẽ clone project từ server về máy PC cá nhân, có thể thay đổi chỉnh sửa trên đó offline mà không cần internet, chỉ khi nào cần cập nhật lên hệ thống chính thì mới cần mạng để cập nhật lên mà thôi, tức là quyền lực bị chia đều cho nhiều nguồn như chế độ phong kiến ở châu Âu. Một trong những hệ thống source control nổi tiếng áp dụng tư tưởng này là Git.

Git và GitHub là gì

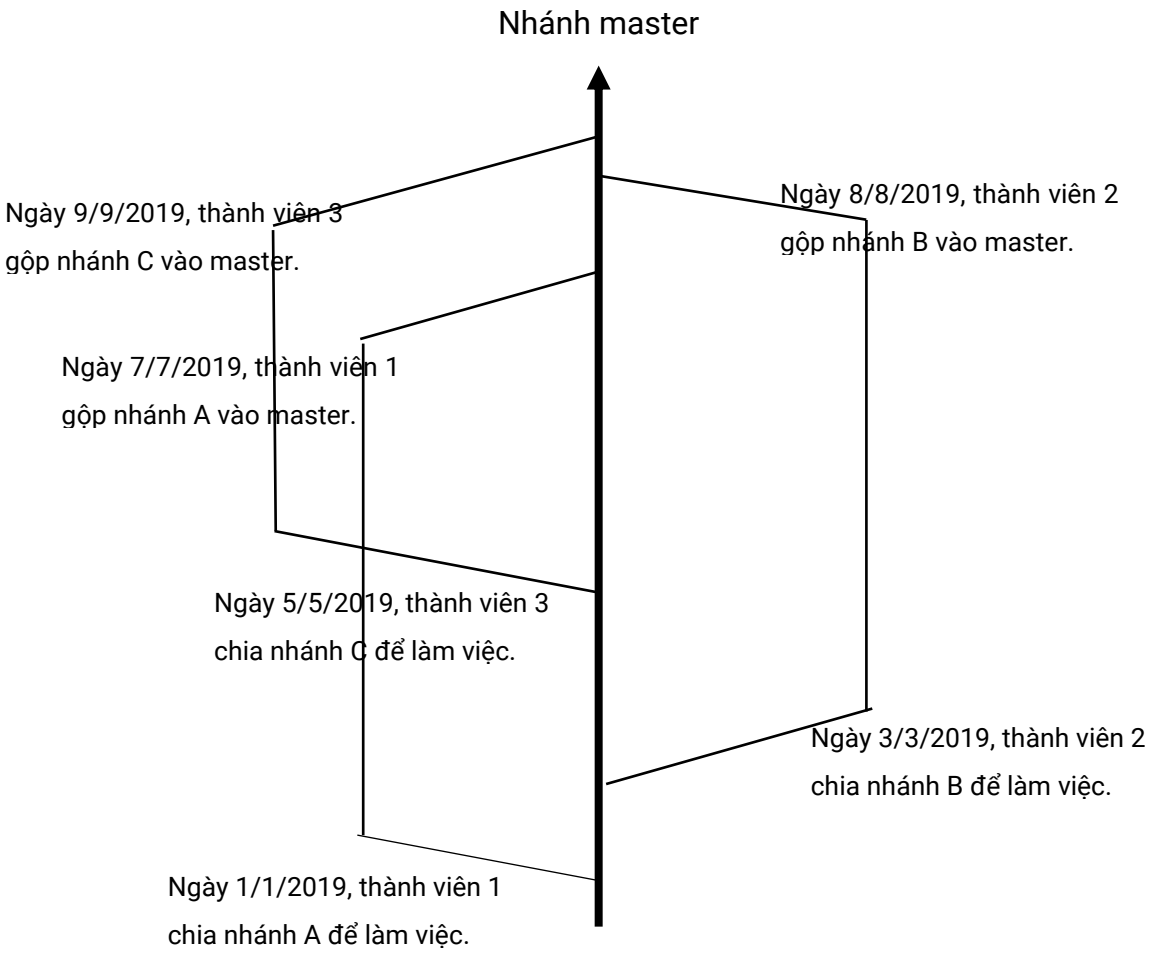
Git, như đã nói, là một trong những hệ thống source control mạnh nhất hiện nay (hay có thể nói là mạnh nhất cũng được). So với các hệ thống source control khác thì Git ra đời sau nhưng được hỗ trợ và phát triển rất mạnh cũng như cơ chế của Git dễ nắm bắt hơn nên được sử dụng rộng rãi hơn. Git sẽ được cài trên máy PC cá nhân của bạn và sẽ là source control quản lý cho các project trên máy bạn. Lúc này máy PC của bạn sẽ được

xem như một local storage cho các project và được quản lý bởi git. Các bạn có thể tạo project ở local rồi cho git quản lý hay dùng git để clone (tải xuống?) một project nào đó trên server về máy local rồi làm việc. Sau khi làm việc xong xuôi thì dùng git để cập nhật lên server chung nhằm lưu trữ lâu dài.

Còn GitHub là gì? Đừng nhầm giữa Git và GitHub, Git là source control dùng để quản lý project ở dưới local, tức là dưới máy bạn, còn GitHub là một giải pháp lưu trữ mã nguồn online, tức là có server và database riêng cho các bạn lưu code trên đó.

Để rõ ràng hơn thì mình sẽ lấy ví dụ về quy trình cụ thể khi làm project như sau: đầu tiên cả team cùng tạo chung một remote repository (repository là nơi lưu trữ và quản lý mã nguồn) trên GitHub. Sau đó ở dưới máy cá nhân, các bạn tạo project, cho Git quản lý nó dưới local (gọi là local repository). Mỗi thành viên sẽ thực hiện việc thay đổi code ở máy của họ, được quản lý bởi Git trên máy của họ. Sau đó, để ghép code với nhau khi hoàn tất, các thành viên sẽ dùng Git để đưa (push code) các đoạn code họ chỉnh sửa từ local repository (máy tính cá nhân của họ) lên cái remote repository (là database của GitHub). Lúc này, sau khi push code hết lên (các bạn sau này sẽ thường xuyên nghe chữ push code), các thành viên để cập nhật code mới của nhau tại máy họ (ở local repository, do thành viên 1 thay đổi trên máy của thành viên 1 rồi push lên GitHub thì ở máy thành viên 2 chưa có code do thành viên 1 thay đổi) thì họ sẽ thực hiện thao tác là kéo code về (pull code) để cập nhật.

Ngoài ra, để tránh ảnh hưởng công việc của nhau trong team thì khi làm việc, mỗi thành viên sẽ tạo 1 nhánh (branch) mới trong Git chung, kiểu như là mỗi người mỗi việc, đường ai nấy đi, việc ai nấy làm ấy. Ai có việc gì mới cứ tạo nhánh mới, tạo đường đi mới ra, không bị đụng với đường đi của người khác. Sau khi hoàn tất phần việc của mình thì sẽ quay về nhánh chính để gặp lại mọi người ở đó (branch master – nhánh chính). Dưới đây là hình ảnh minh họa cho việc chia nhánh và gộp nhánh:



Như các bạn thấy, tư tưởng của Git rất là… OOP (ai làm việc nấy, không can thiệp lẫn nhau), và khi xong xuôi hết mọi việc thì mọi người sẽ cùng hợp nhất lại ở master (lúc ghép code). Ngoài ra thì Git còn có những tính năng rất mạnh mẽ khác nhưng ở giới hạn cơ bản đủ dùng thì chỉ dừng lại ở những tính năng sau:

- Tạo một project và quản lý bằng git (`init`).
- Kết nối Git với một remote repository nào đó (remote repository không nhất thiết phải là GitHub, có thể là GitLab, BitBucket tùy theo nhu cầu).
- Clone một project từ remote repository về local bằng Git (`clone`).
- Xem trạng thái hiện tại của project (`status`).
- Xem log (bản ghi) của Git cho project từ lúc mới tạo đến hiện tại (`log`).

Tác giả: Hứa Anh Minh – Trường Đại học Khoa Học Tự Nhiên Thành phố Hồ Chí Minh

- Thêm các file sau khi đã chỉnh nội dung (`add`).
- Xác nhận sự thay đổi (`commit`).
- Cập nhật sự thay đổi lên remote repository (`push`).
- Cập nhật mới code ở remote repo về local (`pull`).
- Tạo branch mới và chuyển giữa các branch trong Git (`checkout`).
- Ghép branch trong Git (dùng khi ghép code) (`merge`).

Một số lệnh cơ bản của Git

Yêu cầu cần phải có một số kỹ năng cơ bản dùng terminal (CMD, PowerShell, Bash,…) trước khi áp dụng những lệnh của Git vì ta sẽ dùng Git trên terminal. Những kỹ năng cơ bản cần có là:

- Tạo thư mục (`mkdir`).
- Xóa thư mục (`rmdir`).
- Tạo file (New-Item trên Windows, touch trên `Linux`).
- Hiện danh sách item trong thư mục (`ls`).
- Di chuyển giữa các thư mục hay phân vùng (`cd`).
- Copy item (`cp`).
- Xóa item (`rm`).

Lưu ý cuối cùng đó là toàn bộ các lệnh dưới đây sẽ được thực hiện trong thư mục của project do Git quản lý.

Tạo mới project và quản lý bằng Git

Tại thư mục gốc chứa project trên máy, ta sẽ dùng lệnh `git init` để khởi tạo Git quản lý project đó.

File .gitignore của Git là gì?

Ở các project sử dụng Git làm source control để quản lý, các bạn sẽ thường bắt gặp một file có tên khá lạ đó là .gitignore (git ignore). Trong project của ta thì không phải lúc nào

ta cũng cần push toàn bộ nội dung lên remote repository vì dung lượng rất nặng hay không cần thiết. File .gitignore được tạo ra nhằm mục đích liệt kê danh sách những nội dung ta muốn bỏ qua và khi ta xác nhận thay đổi hay push nội dung lên remote repository thì Git sẽ bỏ qua những nội dung được liệt kê trong .gitignore. Những nội dung đó có thể là một file nào đó, hay một thư mục nào đó chẳng hạn. Để tạo được file .gitignore thì ta sẽ gõ lệnh `New-Item .gitignore` (với Windows) hoặc `touch .gitignore` (với Linux) trong thư mục gốc của project.

Kết nối Git với một remote repository

Để làm được điều này thì yêu cầu remote repository đó phải là rỗng, tức là chưa có gì trong đó hết do nếu repository đó có nội dung rồi thì sẽ không dùng kết nối mà sẽ là sao chép bản sao (clone) về máy chúng ta. Để kết nối, ta sẽ dùng lệnh `git remote add <tên_remote> <url_của_remote_repository>`. Thông thường `<tên_remote>` sẽ được đặt là `origin`, nhưng bạn có thể chọn đặt tên khác tùy thích, nhưng cần phải đảm bảo nhất quán giữa các thành viên trong team. Ví dụ `<url_của_remote_repository>` của mình là https://github.com/HuaAnhMinh/c4w-17CLC2-1753070 thì mình sẽ add như sau: `git remote add origin https://github.com/HuaAnhMinh/c4w-17CLC2-1753070`.

Clone một remote repository về local repository

Như đã nói, nếu remote repository đã có nội dung rồi thì ta không thể gọi `git remote add…` được nữa mà sẽ sao chép nội dung nó về máy, bao gồm cả `<tên_remote>` luôn, nên sau khi thay đổi nội dung thì ta có thể dùng lại `<tên_remote>` ban đầu để cập nhật code chứ không cần tạo `<tên_remote>` mới. Cú pháp để clone sẽ là `git clone <url_của_remote_repository>`. Ví dụ nếu các bạn muốn clone repository c4w-17CLC2-1753070 trên GitHub của mình thì dùng `git clone https://github.com/HuaAnhMinh/c4w-17CLC2-1753070.git`.

Xem trạng thái của project

Để xem trạng thái hiện tại của project (có gì thay đổi chưa, thay đổi có được xác nhận chưa, có gì không ổn hay không,...) thì ta dùng lệnh `git status`. Git status là lệnh được khuyến khích hãy liên tục dùng nhiều nhất khi làm project vì nó giúp ta luôn cập nhật được tình hình nội dung trong project hiện tại.

Xem log của project

Với Git, mỗi lần ta xác nhận thay đổi nội dung của project thì Git sẽ lưu lại toàn bộ những lần xác nhận vào log để sau này ta có thể dễ dàng kiểm tra và chỉnh sửa nếu có lỗi. Lệnh để xem log là `git log`.

```
HuaAnhMinh@HUA-ANH-MINH MINGW64 /f/Projects/bigocoder_app (dev)
$ git log
commit 2d4b85acfc1a069e17524f36d3a22fb8972d80f4 (HEAD -> dev, origin/dev)
Merge: cbf1445 3dcb014
Author: Hua Anh Minh <huaanhminh0412@gmail.com>
Date:   Sat May 25 16:57:53 2019 +0700

    Merge branch 'dev' of https://bitbucket.org/bigoengineer/bigocoder_app into dev

commit cbf1445055fdbfc5208d02721d320a3c64cabb5c
Author: Hua Anh Minh <huaanhminh0412@gmail.com>
Date:   Sat May 25 16:57:32 2019 +0700

    Fix bug: unexpected error after register new account

commit 3dcb014181bac60cd7152b1294cfb6237d564d03
Author: kerbal <tranquockhanh997@gmail.com>
Date:   Fri May 24 21:30:51 2019 +0700

    fix pagination

commit e7fb842557c3885259287f45bf8e355b578ea070
Author: kerbal <tranquockhanh997@gmail.com>
Date:   Thu May 23 16:28:05 2019 +0700

    add toggle premium button
```

Như hình trên, mỗi lần commit (những dòng chữ xanh) thì git sẽ ghi lại vào git log và khi log ra sẽ được như thế.

Thêm nội dung đã được thay đổi (push)

Nội dung đã được thay đổi ở đây bao gồm mọi hành động mà bạn thay đổi, từ việc tạo file, chỉnh file có sẵn, xóa file, thêm thư mục, xóa thư mục,… đều được xem là thay đổi nội dung trong project và Git sẽ quản lý toàn bộ. Đầu tiên, nếu bạn thay đổi nội dung gì đó ở project mà chưa thêm vào Git, hãy thử gõ lệnh `git status` để xem trạng thái, ví dụ các bạn đổi file A, B mà chưa thêm thì khi xem trạng thái Git sẽ hiện ra dòng chữ đỏ cảnh báo là file A và B chưa được thêm vào. Để thêm nội dung đã được thay đổi, ta dùng một trong 2 cách sau (tùy theo tình huống mà áp dụng nhé):

- `git add .` : (có dấu . ở sau add nha) dùng để thêm toàn bộ nội dung đã thay đổi. Ưu điểm lệnh này là ngắn gọn và nhanh chóng, gõ 1 lệnh ngắn mà thêm toàn bộ nội dung. Nhược điểm là giả sử có file B bạn thay đổi nhưng không muốn thêm vào thì khi gọi `git add .` nó sẽ kệ bạn có muốn hay không và thêm file B vào luôn. Lệnh này thường dùng khi số lượng nội dung cần thêm là lớn, như khi bạn thay đổi hàng chục file thì thường gọi `git add .` cho nhanh.
- `git add <đường_dẫn_tới_file_1> <đường_dẫn_tới_file_2>`: ngược lại ở trên, dùng git add này các bạn sẽ phải gõ đường dẫn tới từng file hay thư mục bạn muốn thêm vào. Ưu điểm là các bạn có thể thêm vào chính xác nhưng file mà các bạn mong muốn, nhược điểm là tốn thời gian gõ lệnh. Lệnh này thường dùng khi số lượng nội dung cần thêm là ít, chỉ vài file.

Hủy việc thêm nội dung đã thay đổi

Giả sử bạn lỡ tay gọi `git add .` rồi Git thêm file mà bạn không muốn thêm thì sao? Lúc này ta dùng `git reset <tên_file>` để Git bỏ file đã được thêm ra. Có thể kết hợp `git add .` và `git reset` ở tình huống như sau: bạn thay đổi 1000 file nhưng bạn chỉ muốn thêm 999 file. Lúc này không ai lại đi git add từng file cả mà ta sẽ dùng `git add .` để thêm 1000 file rồi dùng `git reset` để gỡ file không muốn thêm ra.

Xác nhận thay đổi (commit)

Sau khi đã thêm file vào bằng git add, các bạn gõ `git status` thì Git sẽ hiện các dòng màu cam cảnh báo nhẹ là file đã được thêm nhưng chưa được xác nhận thay đổi. Để xác nhận thay đổi cập nhật với những nội dung đã được thêm vào (tức là chỉ có thể xác nhận với những file đã được git add vào), ta dùng `git commit -m "<mô tả ngắn gọn cho lần xác nhận này>"`. Ví dụ file A mình có bug là X, mình sửa bug X, sau đó gọi `git add ./A` để thêm nội dung file A vào Git, rồi để xác nhận mình sẽ gọi `git commit -m "Sua bug X trong file A"`. Có một số lưu ý sau về commit:

- Nên đặt mô tả của commit cho tường minh và ngắn gọn về những gì bạn đã thay đổi. Đừng đặt kiểu như "Fix bug" sẽ khá mơ hồ.
- Mỗi commit sẽ được Git cấp cho 1 mã hash duy nhất và các mã hash này hoàn toàn không giống nhau. Ta sẽ dựa vào các mã hash này để phân biệt giữa các commit với nhau và nếu ta muốn hoàn toàn lại project về phiên bản trước đó thì ta sẽ dùng mã hash tương ứng của lần commit muốn quay về.
- Mỗi khi commit thì nội dung của commit đó sẽ được ghi trong log của git.

Cập nhật sự thay đổi lên remote repository (push)

Trước khi dùng lệnh này, các bạn hãy dùng git status để kiểm tra xem nhánh (branch) hiện tại đang đứng là nhánh nào vì có nhiều trường hợp đang đứng ở nhánh A mà đi push nhầm sang nhánh khác thì sẽ gây ảnh hưởng không nhỏ và để kiểm tra tên remote hiện tại là tên gì. Lưu ý khác đó là lệnh push này chỉ áp dụng push đối với những nội dung đã vừa được commit, những file được thay đổi nhưng chưa được add hay chưa được commit thì sẽ không được push. Cú pháp lệnh push như sau: `git push <tên_remote> <nhánh_hiện_tại>`. Một số nơi các bạn thấy sẽ có thêm `-u` là `git push -u <tên_remote> <nhánh_hiện_tại>`. Việc thêm -u sẽ giúp git ghi nhớ `<tên_remote>` và `<nhánh_hiện_tại>` ngay lúc đó và về sau nếu như bạn vẫn push với `<tên_remote>` đó và `<nhánh_hiện_tại>` đó luôn thì chỉ cần gõ mỗi git push là được. Việc push code có một lưu ý khá quan trọng, đó là pull code ở mục sau.

Cập nhật code mới từ remote repository về local (pull)

Push và Pull như hai anh em luôn song hành với nhau, vì các bạn sẽ gặp rắc rối về đụng độ (conflict) nếu như quên pull code mới trước khi push. Ví dụ thành viên 1 push file mới tên là B lên remote repository. Thành viên 2 đã xong công việc, lúc này muốn push code của mình lên, nhưng trong máy local thành viên 2 chưa có file B do thành viên 1 push lên. Lúc này nếu thành viên 2 cố gắng push lên sẽ bị báo lỗi ngay lập tức là conflict như sau:

```
Error: thất bại khi push nội dung mới lên remote repository. Gợi ý: việc push bị từ chối vì remote repository có một số nội dung thay đổi mà bạn chưa cập nhật trên máy bạn, hãy dùng git pull… trước khi push lần nữa.
```

Ngoài ra git cũng có thể đưa thêm 1 lựa chọn "khá dễ thương" khác là gõ lệnh git push `fast-forwards`. Đừng dại mà thử nhé, `fast-forwards` sẽ bỏ toàn bộ conflict, tức là nó xóa toàn bộ các file trên remote repository gây ra lỗi (cụ thể là xóa file B) và đẩy code của ta lên. Có đợt mình lỡ dùng và xóa một nửa các file trong project của team đấy, ngầu chưa.

Lúc này, ta sẽ dùng git pull để cập nhật code về local của ta, cú pháp là: `git pull <tên_remote> <branch_hiện_tại>`. Ví dụ như `git pull origin master`.

Lúc này, khi gọi pull thì git tiến hành kéo code từ remote về và cố gắng gộp code với local của ta. Nếu mọi chuyện diễn ra suôn sẻ thì quá trình này sẽ thành công và lúc đó các bạn chỉ cần gọi git push… để push code lên là được. Nếu mọi chuyện không suôn sẻ thì… các bạn ăn thêm cú conflict nữa chứ sao. Conflict lần này thường là do tình huống 2 đã đề cập: nhiều thành viên cùng chỉnh sửa nội dung 1 file, ví dụ file A có nội dung "abc" và thành viên 1 chỉnh thành "abcd" và push lên thành công. Đến lượt của ta, ta chỉnh "abc" thành "ab", push lên bị conflict, pull về thì lúc này file A trên remote đang là "abcd", file A trong máy local của ta là "ab", git bị bối rối và không biết nên lấy phiên bản file A nào, lúc này nếu bạn có text editor xịn thì ngay trong file A khi mở ra nó sẽ hiện cho các bạn những lựa chọn sau:

- Lấy phiên bản file A từ remote.
- Lấy phiên bản file A từ local.
- Lấy cả 2 (thường nếu chọn cách này thì bạn nên xem kỹ sau khi gộp vì có thể nó không gộp như ta mong muốn).

Tạo và chuyển branch trong Git (checkout)

Để chuyển từ branch A sang branch B (B đã có) trong Git, ta dùng lệnh `git checkout B` khi đang đứng ở branch A. Để chuyển từ branch A sang branch B (B chưa có và ta muốn tạo mới B), ta đứng từ branch A và dùng lệnh `git checkout -b B` (có thêm `-b`). Lưu ý rằng trước khi chuyển sang branch khác thì ta phải commit toàn bộ những nội dung đã thay đổi ở branch hiện tại và trong Git thì branch mặc định khi mới tạo Git và là branch chính đó là branch master.

Ghép branch trong Git (merge)

Lệnh merge trong Git là một lệnh có phần hơi nâng cao một chút vì nó có nhiều loại cú pháp. Trong bài này mình sẽ hướng dẫn merge cơ bản nhất là merge từ branch này sang branch khác. Giả sử các bạn đang ở branch master, các bạn muốn merge branch A vào master thì sẽ gọi `git merge A`. Trong nhiều trường hợp các bạn sẽ gặp lỗi merge conflict, đụng độ khi gộp code. Thực chất conflict khi merge và khi pull thường giống nhau nên các bạn có thể xử lý như cách xử lý khi pull.

Kết

Phía trên là những gì cơ bản nhất mà lập trình viên nào cũng cần phải biết về source control và Git. Tất nhiên, những gì ở trên vẫn chưa đủ và thiếu rất nhiêu so với mức cần, vì thế để cải thiện và nâng cao khả năng dùng Git thì các bạn có thể tải ebook Pro Git và đọc thêm ở đây https://git-scm.com/book/en/v2