tài liệu tham khảo

<https://backlog.com/git-tutorial/vn/stepup/stepup1_3.html> (ưu tiên)

<https://sociss.edu.vn/lessons/tu-hoc-git-tu-co-ban-den-nang-cao-phan-2>

<https://medium.com/@trunghieuvn/nga%CC%80y-2-qua%CC%89n-ly%CC%81-source-code-va%CC%80-la%CC%80m-vi%C3%AA%CC%A3c-nho%CC%81m-v%C6%A1i-git-8778e7fec375>

# kiểm tra tra phiên bản git hiện tại

git --version

# cấu hình ban đầu

git config

git --help // xem ý nghĩa các câu lệnh trong git

git help config // cxem ý nghĩa các câu lệnh trên html

git config --global user.name = "tên tài khoản"

git config --global user.email = "tài khoản email đăng ký GitHub"

# cấu hình thư mục làm việc

git init // tọa file .git

git status // trạng thái : file thay đổi. thêm , xóa file, chưa, đã commit,..

git add \* // thêm thay đổi xác nhận

git commit -m "title of commit to repository" // commit file đã add

git commit -a -m "title of commit to repository" // gộp 2 lệnh add và commit lại thành 1

\* việc full giữa các nhánh không ảnh hưởng tới các nhánh, branch (local) ở trong máyme

# clone về máy tính

git clone <địa chỉ https>

# Tạo tài liệu github

## Tạo mới 1 remote repository trên github, sau đó đẩy dữ liệu lên

echo "# test1" >> README.md

git init

git add README.md

git commit -m "first commit"

git remote add origin https://github.com/vuminhhieu99/test1.git

git push -u origin master

## git remote add origin <địa chỉ https>

* có thể thay origin bằng bất kỳ tên nào khác
* tương ứng với nó là các lệnh
  + git push origin <tên nhánh>
  + git pull origin <tên nhán

## git clone <địa chỉ https>

--- lấu toàn bộ dữ liệu từ remote responsitory

! nếu bạn không phải là người đóng góp. hãy tạo 1 fork từ tài khoản của mình rồi kéo về từ đó

# Remote Branch

Như bạn đã biết ở trên, mỗi repository có các branch. Local repository của chúng lại tham chiếu tới nhiều repository khác thông qua remote. Dĩ nhiên các repository này sẽ có các branch bị trùng tên với nhau, chẳng hạn như master. Vậy làm sao phân biệt branch nào của repository nào?

Câu trả lời là Remote Branch. Các remote branch cũng là các branch bình thường, nhưng có prefix được git thêm vào chính là tên của remote.

## checkout branch hiện tại

git checkout <tên branch>

## tạo 1 brach mới trên cả local và remote

* tạo 1 branch trên local

Tạo một branch mới có tên là feature

# Cách 1, chỉ thực hiện tạo branch:

> git branch feature

# Cách 2, tạo mới branch và chuyển luôn sang branch đấy để code:

> git checkout -b feature/a

* tạo 1 branch trên remote repoditory
* commit xong push lên bằng câu lệnh: git push origin <tên branch>
  + nếu phía remote chưa tạo nó sẽ tự động tạo. VD:

$ git push origin branch2

Enumerating objects: 3, done.

Counting objects: 100% (3/3), done.

Delta compression using up to 4 threads

Compressing objects: 100% (2/2), done.

Writing objects: 100% (2/2), 255 bytes | 255.00 KiB/s, done.

Total 2 (delta 0), reused 0 (delta 0), pack-reused 0

remote:

remote: Create a pull request for 'branch2' on GitHub by visiting:

remote: https://github.com/vuminhhieu99/test1/pull/new/branch2

remote:

To https://github.com/vuminhhieu99/test1.git

\* [new branch] branch2 -> branch2

* + nếu đã có 1 nhánh remote trùng tên thì sẽ push lên đấy luôn-
* có thể cài git push --set-upstream origin branch2 để rút ngắn câu lệnh push thành : $ git push

Hoặc:

* tạo branch trên remote
* tạo branch local bằng cách: $ git checkout Hieu

khi đó sẽ có thông báo: Branch 'Hieu' set up to track remote branch 'Hieu' from 'origin'.

Git sẽ kiển tra trong máy nếu chưa có branch thì tự động so trùng tên nhánh trên remote. Nếu trùng tên sẽ tự động tạo local và kết nối tới remote branch trùng tên

! origin đại diện cho url của remote repository nên bắt buộc phải có khi gõ

nhưng có thể thay bằng tên khác

! các nhánh tạo sau sẽ được kế thừa toàn bộ file từ nhánh trước.

! cách nhánh trước(nhánh gốc) không thể sát nhập vào các nhánh sau

branch2 no pull request -> branch3

! có thể pull từ branch bất kỳ nhưng khy push thì push cho đúng nhánh tương ứng trên remote(nhánh trùng tên) đừng cố thử lung tung \*\*\*

## Liệt kê đầy đủ thông tin các nhánh

$ git remote show origin

## liệt kê các nhánh local

git branch

## liệt kê tất cả các nhánh

git branch –a

## Liệt kê các remote urls

git remote –v

## Xóa 1 remote branch

git push --delete <repository> <tagname>

VD: git push --delete origin StateOne

## Xóa 1 local branch

git branch -d <branchname>

**Sự cố 16: Khi pull từ remote về thì có quá nhiều conflict nên tạm thời muốn trở lại như cũ**

Sau khi pull code về cập nhật nhưng thấy conflict nhiều như núi thì quả là muốn trở lại như cũ Gần giống như sự cố 15

# Lấy từ code mới remote

$ git pull origin master

# Phát sinh conflict

# Suy nghĩ lại thì trong pull(fetch + merge) muốn bỏ phần merge đi

$ git reset --hard ORIG\_HEAD

# đưa tài liệu lên git (nhớ là chỉ đưa lên được nhánh của mình mình>

git add data.txt

git commit -m "Update data.txt"

git push origin <tên branch đang checkout>

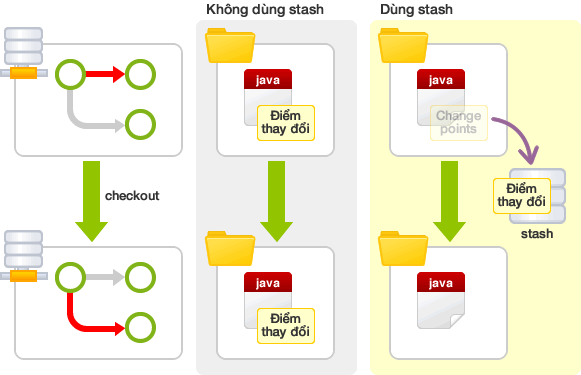
# **Stash**

**Khi những file được thêm mới hay nội dung thay đổi vẫn chưa commit vẫn còn lưu lại Index và Work tree, mà thực hiện checkout đến branch khác thì nội dung thay đổi đó sẽ di chuyển từ branch ban đầu đến branch chuyển đến.**

Tuy nhiên ở branch di chuyển đến, trường hợp có file giống như vậy đã được tiến hành thay đổi cái gì đó thì checkout sẽ thất bại. Khi xảy ra trường hợp này thì sẽ commit lại nội dung thay đổi 1 lần nữa hoặc là sử dụng stash để lưu tạm thời nội dung thay đổi, sau đó phải thực hiện commit.

Stash là khu vực ghi lại tạm thời nội dung thay đổi của file. Bằng việc sử dụng stash, trong work tree và index, những thay đổi chưa được commit có thể lưu lại tạm thời.

Những thay đổi lưu tạm này về sau có thể lấy ra và hiển thị trên branch ban đầu hay là phản ánh lên branch khác.



Khi muốn chuyển sang một branch khác, hãy hết sức lưu ý rằng bạn cần phải tạo thói quen commit lại các thay đổi trước khi chuyển bằng lệnh:

> git commit -m "My message for note"

git stash list: xem danh sách cách stash

git stash save: lưu lại những gì đã làm vào stash để chuyển nhánh mà không muốn commit

git stash apply: Lấy ra thằn stash được thêm vào cuối cùng theo cấu trúc stack(vào trước ra sau)

Câu lệnh tương đương : git stash apply stash@{0}

git stash apply stash@{<index>}: lấy ra stash thứ <index>

**Git stash pop : lấy gần nhấy và xóa luôn trong stack**

**Git stash clear: xóa toàn bộ stash**

git stash drop: xóa stash gần nhấy

git stash drop stash@{<index>}: chỉ định xóa stash có <index>

**Branch**

branch trong máy độc lập với branch trên web

có thể sáp nhập trước trong máy rồi chỉ đưa master lên web hoặc đưa lên web theo từng nhánh rồi sau đó sát nhập ở trên web

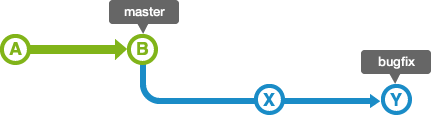
**Tích hợp branches**

Branch chủ đề đã hoàn thành công việc, cuối cùng sẽ được tích hợp vào branch tích hợp. Việc tích hợp branch thì có hai phương pháp, phương pháp sử dụng merge và phương pháp sử dụng rebase. Dựa vào việc sử dụng phương pháp nào mà lịch sử của branch sau khi tích hợp sẽ có sự khác biệt lớn.

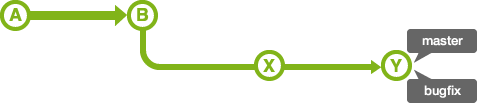
**Merge**

Khi sử dụng merge, có thể tổng hợp nhiều luồng lịch sử.

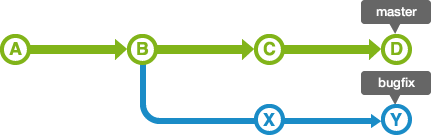
Ví dụ, có branch bugfix phân nhánh ra từ branch master như sơ đồ bên dưới.



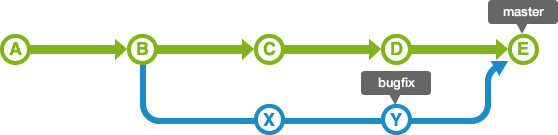
Khi merge branch bugfix này vào branch master, nếu trạng thái branch master không có gì thay đổi thì có thể thực hiện merge hết sức đơn giản. Vì lịch sử của branch bugfix sẽ bao gồm tất cả lịch sử của branch master, nên branch master chỉ có việc di chuyển đơn giản là có thể lấy được nội dung của branch bugfix. Và, người ta gọi merge như thế này là merge fast-forward (chuyển tiếp nhanh).



Nhưng, cũng có trường hợp lịch sử của branch master vẫn đang tiến triển sau khi phân branch bugfix. Trường hợp này thì cần thiết phải tổng hợp nội dung thay đổi của branch master và nội dung thay đổi của branch bugfix thành một.

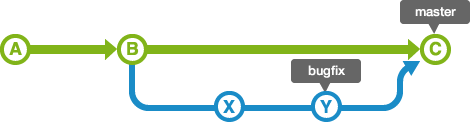


Vì thế, merge commit đã lấy thay đổi của cả hai branch sẽ được tạo ra. Đầu branch master sẽ di chuyển đến commit đó.



NNote

Khi thực hiện merge, bằng việc chỉ định lựa chọn merge non fast-forward, cho dù là trường hợp có thể merge fast-forward thì vẫn có thể tạo ra merge commit mới rồi kết hợp lại.



Khi tiến hành non fast-forward, vì branch vẫn còn nguyên như vậy, nên việc chỉ định công việc đã tiến hành tại branch đó sẽ trở nên đơn giản.

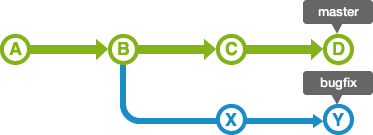
Giả sử như trên là bạn muốn merge nhánh feature-homepage vào master thì từ nhánh master bạn chạy lệnh sau

git checkout master

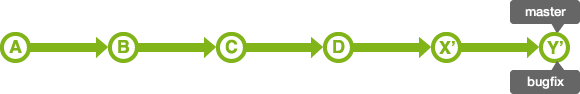
git merge feature-homepage

**Rebase**

Giống với ví dụ merge, sẽ có branch bugfix phân nhánh từ branch master như sơ đồ bên dưới.

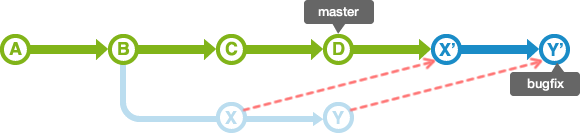


Trường hợp sử dụng rebase ở đây rồi tiến hành tích hợp branch thì lịch sử sẽ giống như sơ đồ tiếp theo. Bây giờ, sẽ trình bày đơn giản trình tự rebase như thế nào.



Trước hết, khi rebase branch bugfix vào branch master, lịch sử branch bugfix sẽ được thay đổi đính kèm sau branch master. Cho nên, lịch sử sẽ thành 1 đường như trong sơ đồ.

Khi này sẽ có trường hợp phát sinh xung đột tại commit di chuyển X và Y. Lúc đó tại từng commit cần thiết phải chỉnh sửa lại những chỗ phát sinh xung đột.



Nếu chỉ rebase không thôi thì vị trí đầu master vẫn cứ như vậy. Vì thế, merge branch bugfix từ branch master rồi di chuyển master đến phần đầu của bugfix.



Note

Cả merge và rebase đều là tích hợp lịch sử lại với nhau nhưng đặc trưng thì khác nhau.

* **Merge**  
  Lịch sử nội dung thay đổi vẫn còn lại nhưng sẽ trở nên phức tạp hơn.
* **Rebase**  
  Lịch sử sẽ trở nên đơn giản nhưng nội dung thay đổi từ commit ban đầu sẽ bị thay đổi. Cho nên cũng có trường hợp rơi vào tình trạng commit gốc không hoạt động.

Merge và Rebase thì tùy theo phương châm áp dụng của nhóm mà chia ra sử dụng.

Ví dụ, nếu áp dụng để hợp nhất hóa lịch sử, thì sẽ phân ra sử dụng như sau:

* Trường hợp đưa code mới nhất của branch global vào branch local –cùng tên thì sử dụng rebase.
* Trường hợp đưa branch chủ đề vào branch tích hợp, thì trước hết hãy rebase rồi merge.
* pull = fetch + merge
* pull --rebase = fetch + rebase

config rebase:

1-> fix config

2-> git add . (in git)

3-> git rebase --continue (in git)

4-> change name commit (nomal add ‘ in last in name)

5 -> clonse file have “change name commit”

* lưu ý khi rebase có thể bị dư code do chính mình viết

vd: khi rebase vào nhánh feature-5 ta bị dư dòng :

add feature-05

là do dòng thứ nhất fix từ commit thứ nhất: khi đó là

add feature-06

add feature-06 l1

add feature-05

khi fix commit thứ 2 ta có như sau

head<<<<<<<<<<<<<<<<<<<<<<<

add feature-06

add feature-06 l1

add feature-05

feture>>>>>>>>>>>>>>>>>>>>>>>>>

add feature-05

add feature-05 l1

add feature-05

add feature-05 l1

* **rebase là bạn đang viết lại lịch sử**

[Kế tiếp](https://backlog.com/git-tutorial/vn/stepup/stepup1_5.html)

# Sự khác nhau giữaclone, fetch và pull

Có thể bây giờ bạn đã biết được 3 lệnh để lấy dữ liệu về từ repository đó là git clone, git fetch và git pull. Nhưng cả ba loại đều là lấy dữ liệu, thế sự khác nhau của nó là gì?

**git clone**

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

**git pull**

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

**git fetch**

Lệnh này sẽ lấy toàn bộ dữ liệu từ remote repository nhưng sẽ cho phép bạn gộp thủ công vào một branch nào đó trên thư mục Git ở máy tính.

Dung git ignore để không add folde node\_modul trên remote git