**GIT BASH**

***\*Mục đích của file này là muốn sử dụng thành thạo git bash khi thấy ngta khá là chuyên nghiệp trong việc sử dụng git với git bash và cũng không muốn đóng kén phần mềm git bash đã quá lâu ☺***

**I. Git là gì?**

- Git là một hệ thống quản lý phiên bản phân tán (distributed version controler system).

*\*VSC (Version Control System): … khỏi để cập.*

- Tác dụng của Git:

+ Lưu lại các phiên bản khác nhau của mã nguồn dự án phần mềm.

+ Khôi phục lại mã nguồn từ một phiên bản bất kỳ.

+ Dễ dàng so sánh các phiên bản.

+ Phát hiện được người nào và phiên bản nào gây ra lỗi.

+ Khôi phục lại tập tin bị mất.

+ Dễ dàng thử nghiệm, dễ dàng mở rộng tính năng của sản phẩm mà không sợ làm ảnh hưởng tới phiên bản chỉnh của mã nguồn.

+ Giúp phối hợp thực hiện sự án trong một nhóm hiệu quả. (Thấy mỗi dòng này là có gì đó không màu mè và chính xác nhưng cũng chẳng có gì để nói).

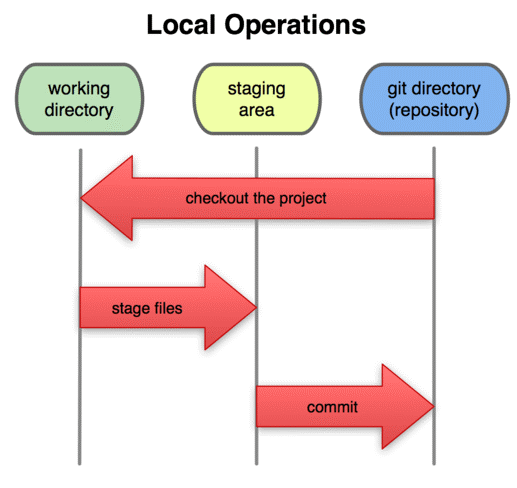
- Lợi ích của Git:

+ Git chứng minh mình là một developer thực sự. (Cái này là chép lại chứ thực sự với tôi nó không phải là lợi ích gì cả).

+ Github giúp cải thiện code skill: Giúp lập trình viên cải thiện kỹ năng code bằng cách theo dõi và so sánh thay đổi thường xuyên.

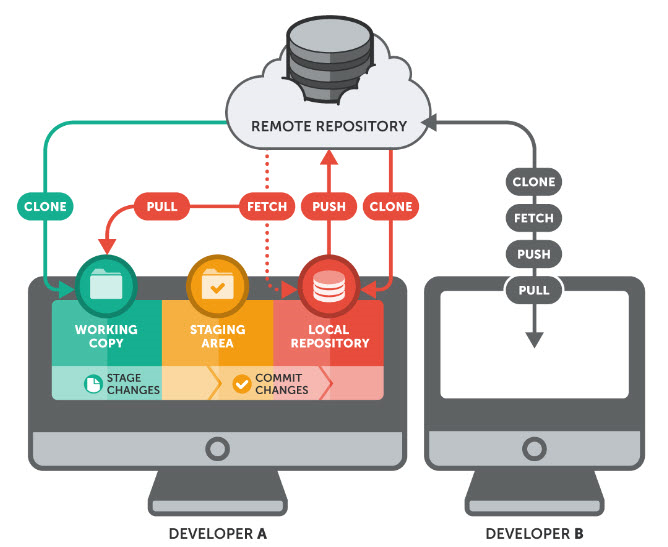
+ Phát triển nhiều kỹ năng mới ☺ (và thế là tôi sang một page khác để viết phần phía trên).

- Quy trình xử lý công việc trên Git thường là như sau:

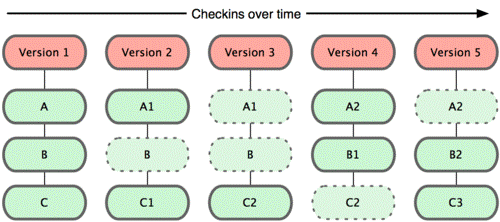


**II. Các khái niệm cơ bản:**

***1. Repository (kho lưu trữ):*** đây là nơi lưu trữ tất cả code và các tập tin, tài nguyên cần thiết). Có hai loại repository là local repository và remote repository.



***2. SNAPSHOT:*** cơ chế lưu trữ phiên bản của git là sau mỗi lần thực hiện lưu trạng thái sẽ tạo ra một SNAPSHOT lưu lại nội dung của tất cả các tập tin thư mục tại thời điểm lưu đó, sau đó nếu tập tin không có thay đổi thì Git sẽ không lưu lại tập tin đó mà chỉ tạo liên kết tới tập tin gốc đã tồn tại trước đó, giúp tiết kiệm không gian lưu trữ:



***3. Commit:***

- Commit là thao tác báo cho hệ thống biết bạn muốn lưu trữ lại trạng thái hiện hành.

- Mỗi commit được tạo ra gồm một số thông tin chính:

+ Tên và email của ngưởi tạo commit

+ Ngày giờ commit được tạo ra

+ Message: Mô tả cho sự các sửa đổi (Để sau này xem lại còn biết nó sửa vì mục đích gì)

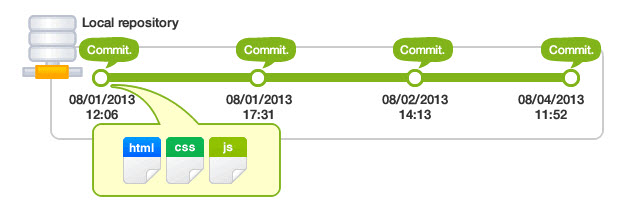
+ Một ID là một chuỗi SHA gồm 2 dạng: Full và Short.

Full: Chuỗi SHA đầy đủ

Short: Là 7 ký tự đầu tiên được cắt ra từ chuỗi Full

- Khi thực hiện commit, trong repos sẽ ghi lại sự khác biệt từ lần commit trước để với trạng thái hiện tại. Các commit ghi nối tiếp nhau theo thứ tự thời gian nên hoàn toàn có thể theo vết các commit và biết được lịch sử thay đổi trong quá khứ.

- Khi thực hiện commit thì hệ thống luôn yêu cầu nhập một commit message để tóm tắt lại những gì đã thực hiện tại lần lưu này.



***4. Clone:***

- Việc clone trong git sẽ tạo ra một bản sao của một repos có sẵn, có thể là một project đang tham gia hoặc một repos khác. Khi clone một project khác, nó sao chép các thiết lập và tạo ta một branch master trên máy của bạn.

- Có thể fork một project của người khác và có một branch master trên máy của mình chỉ khác là có thể thực hiện pull request để đóng góp thêm cho repos đó. Chủ sở hữu sẽ duyệt qua các đóng góp đó và có thể sẽ tiến hành merge nội dung đã chỉnh sửa của mình vào source gốc.

***5. Push:***

- Dùng để đưa local repos lên server.

***6. Fetch:***

- Khi thực hiện fetch, lệnh này truy cập vào repos trên server và kéo toàn bộ dữ liệu mình chưa có từ server về, sau đó có thể tích hợp vào branch bất kỳ lúc nào.

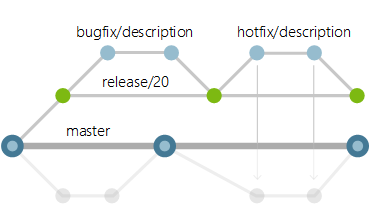
***7. Pull:***

- Lệnh này sẽ tự động lấy toàn bộ dữ liệu từ repos trên server và gộp vào branch hiện tại.

***8. Branch:***

*­*- Là một cơ chế để xử lý các thay đổi trong một kho lưu trữ duy nhất, đại diện cho luồng phát triển của repos, nó là một version riêng của code.

- Với branch, ta có thể tách riêng ra các tính năng của dự án và thử nghiệm các tính năng đó cũng như khắc phục, sửa lỗi nào đó của project. Khi mới bắt đầu một Repos sẽ có một branch chính là branch master. Đây là branch chính chứa toàn bộ mã nguồn ban đầu, và sau đó có thể tách nhánh để thực hiện các tính năng.



- Các thao tác như fetch, push hoặc pull đều được hiện trên nhánh hiện hành nên cần kiểm tra để tránh sai sót.

- Khi nhánh chưa được commit và có xung đột ở nhánh đó thì git sẽ không cho phép chuyển nhánh.

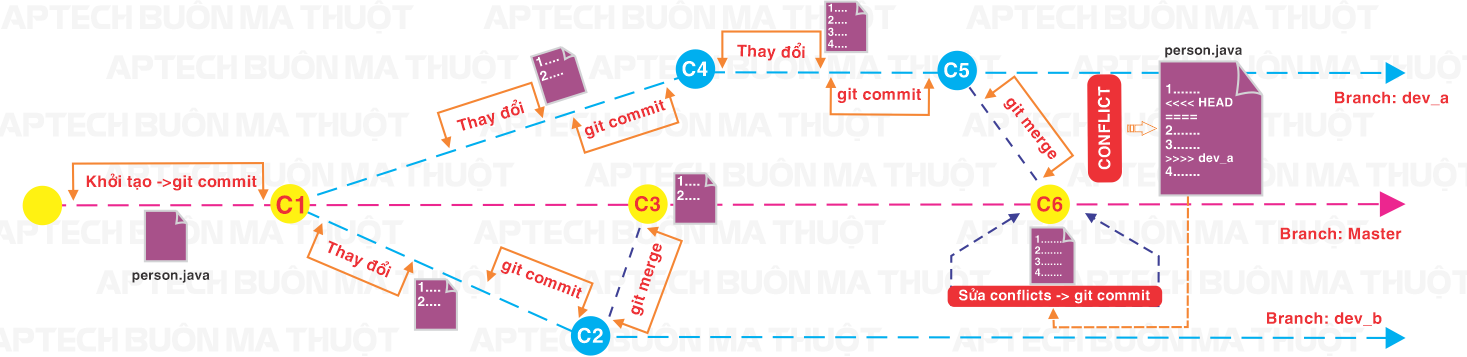
***9. Merge:*** *là thực hiện gộp code từ một nhánh vào một nhánh và khi gộp có thể xảy ra các xung đột (conflict).*

***10. Git Rebase:***

-

***10. Conflict:***

- Conflict là xung đột có thể xảy ra khi hai hay nhiều người cùng thay đổi một tệp. Conflig có thể xảy ra tại local repos hoặc remote repos. Khi cả hai cùng cố gắng thay đổi một file thì git sẽ không biết như thế nào là đúng, và khi đó git sẽ tạm dừng hợp nhất tại file đó.



- Xung đột chỉ xảy ra trong quá trình hợp nhất, những người không tham gia vào quá trình đổi file này sẽ không biết gì về xung đột đã xảy ra. Ai gây ra conflig thì người đó sửa.

- Khi có xung đột tại một vị trí nào đó trong một file, git sẽ thêm một số từ khoá trong file và trách nhiệm của dev là giải quyết chúng để có thể tiếp tục tiến hành merge.

- Các từ khoá này là conflict deviders gồm:

+ <<<<<<< HEAD: là phần nội dung tồn tại trong nhánh master hiện tại mà tham chiếu header đang trỏ tới.

+ =======: sau dòng này là nơi xảy ra conflict, thể hiện nội dung xung đột.

+ ">>>>>>> user": Là những gì mà user muốn hợp nhất với master.

***11. Fork và pull request:***

- Fork là tạo ra một bản copy của repository. Việc fork cho phép bạn dễ dàng chỉnh sửa thay đổi source code gốc mà không ảnh hưởng source gốc.

- Các quy trình cơ bản của fork:

1. Fork repository đó về tài khoản Github của mình.
2. Thực hiện fix bug hoặc thêm tính năng.
3. Gửi một Pull Request tới repository gốc. (đôi khi mình giữ lại xài chứ không cho ngta ☺)

- Khi fork sẽ tạo ra một master branch riêng tách rời với master brach của chủ sở hữu và có thể merge lên branch này.

- Pull request: là chức năng cho phép báo với chủ sỡ hữu code gốc về các tính năng

***12. Git Tag:***

C: copy of a file into a new one

D: deletion of a file

M: modification of the contents or mode of a file

R: renaming of a file

T: change in the type of the file

X: "unknown" change type (most probably a bug, please report it)

***13. Lưu ý khác:***

1. In Unix systems the end of a line is represented with a line feed (LF). In windows a line is represented with a carriage return (CR) and a line feed (LF) thus (CRLF). when you get code from git that was uploaded from a unix system they will only have an LF.

If you are a single developer working on a windows machine, and you don't care that git automatically replaces LFs to CRLFs, you can turn this warning off by typing the following in the git command line

git config core.autocrlf true

2. Các trạng thái commited, staged, modified:

- Commited = sửa + add + commit

$ git checkout task1

$ git add demo.txt

$ git commit -m "Sua file demo.txt"

- Staged: Sửa + Add

$ git checkout task1

$ git add demo.txt

(chưa add sẽ mang trạng thái untracked)

Untracked files:

(use "git add <file>..." to include in what will be committed)

NV.txt

- Modified: Sửa only:

$ git checkout task1

**II. Các lệnh trong Git:**

***1. Kiểm tra phiên bản:***

git --version

git version 2.23.0.windows.1

***2. Cấu hình username và gmail:***

- Trước khi sử dụng Git thì cần phải cấu hình hai tham số là username và email.

- Để cấu hình username, có thể làm như sau:

git config --global user.name “yourname”

- Để cấu hình gmail có thể làm như sau:

git config --globall user.email “yourmail”

***3. Các lệnh với thư mục:***

cd path

- Tới thư mục user:

cd ~

- Tạo mới một thư mục:

mkdir folder\_path

- Xoá thư mục hiện tại:

rmdir

- Xem đường dẫn tuyệt đối tới thư mục hiện tại:

pwd

- Xem danh sách file và thư mục trong đường dẫn hiện tại:

ls

- Tạo file:

> file\_name.txt

***4. Các lệnh với git Repo:***

***\*Thực hiện thiết lập repo (trước tiên cần cd tới folder chứa dự án):***

git init

=> Lúc này trong thư mục vừa mở sẽ có một thư mục bị hidden tên .git



Sau này nếu cd lại folder này thì sẽ ở nhánh của repo lưu lại trước đó.

***\*Thêm file vào git:***

+ Tạo mới một file demo.txt

+ Gõ lệnh sau trong thư mục được chọn là git init: (sẽ không có thông báo gì nếu file là đúng).

git add demo.txt

*Có thể sử dụng add \* để thêm tất cả vào Git.*

***\*Commit thay đổi trong git: (sau khi có thay đổi trong project)***

git commit -m "Message cho lan commot hien tai"

=> Kết quả nếu thành công:

git commit -m "Test thoi nha Git server"

[master (root-commit) 7eca8fd] Test thoi nha Git server

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100644 demo.txt

- Lúc này trên Repo sẽ có 1 file trong nhánh master, có thể tạo ra một nhánh khác bằng lệnh:

git branch <branchname>

*Bài toán ví dụ:* tạo ra 2 nhánh với hai task 1 và 2, task 1 là thêm 1 file txt với tên file là Dong, task 2 là tạo ra một file txt với tên là NV:

=> Tạo 2 nhánh: (không có thông báo là tạo thành công)

git branch task\_1

git branch task\_2

- Xem danh sách các nhánh:

git branch

=> Kết quả: (có dấu \* là nhánh hiện tại)

git branch

\* master

task\_1

task\_2

***\*Chuyển nhánh:***

- Tiếp theo cần chuyển branch qua task\_1 để thực hiện công việc, với lệnh checkout:

git checkout task\_1

*Có thể vừa chuyển nhánh vừa tạo nhánh bằng lệnh:*

git checkout – b task\_3

*Ví dụ:*

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Courses (new\_branch)

$ git checkout -b created\_branch

Switched to a new branch 'created\_branch'

Kiểm tra lại nhánh hiện tại có là task\_1 không?

Cách 1: (xem trên thân)

Cách 2: git branch:

$ git branch

master

\* task\_1

task\_2

Nếu chuyển nhánh thành công sẽ có thông báo sau:

$ git checkout task\_1

Switched to branch 'task\_1'

Nếu chuyển nhánh không thành công do một số lý do như nhánh không tồn tại sẽ xuất hiện thông báo sau:

$ git checkout task\_4

error: pathspec 'task\_4' did not match any file(s) known to git

- Thực hiện chỉnh sửa file trong branch task\_1: Thêm file Dong.txt:



- Tiến hành commit nhánh 1:

+ Trước khi commit cần phải sử dụng lệnh git add <file\_name> để khai báo là sẽ commit file Dong.txt để có thể tracking được nó trong git:

$ git add Dong.txt

Nếu commit mà không add file vào thì sẽ xuất hiện thông báo tương tự như sau:

$ git commit -m "done task\_1"

On branch task\_1

Untracked files:

Dong.txt

nothing added to commit but untracked files present

+ Sau đó commit bằng lệnh commit, kết quả trả về như sau:

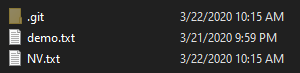
$ git commit -m "task\_1 done"

[task\_1 43e357c] task\_1 done

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100644 Dong.txt

- Tiến hành thay đổi và commit nhánh 2:



$ git add NV.txt

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/GitBashRepo (task\_2)

$ git commit -m "task\_2 done"

[task\_2 39caa6a] task\_2 done

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100644 NV.txt

- Tiến hành chuyển sang nhánh master và merge nhánh 1:

$ git checkout master

Switched to branch 'master'

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/GitBashRepo (master)

$ git merge task\_1

Updating 7eca8fd..43e357c

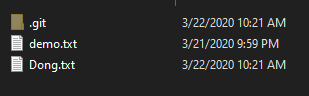
Fast-forward

Dong.txt | 0

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100644 Dong.txt

Lúc này file Dong.txt đã nằm ở nhánh master:

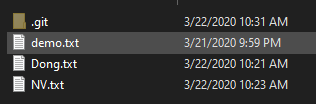


- Merge nhánh 2 vào master:

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/GitBashRepo (master)

$ git merge task\_2

=> Kết quả trong nhánh master:



=> Xong.

***\*Xử lý xung đột:***

- Tạo mới một Repos:

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access

$ mkdir Conflict

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access

$ cd Conflict/

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Conflict (master)

$ ls -a

./ ../ .git/

- Thêm một file Test.txt sau thêm vào git, commit và kiểm tra:

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Conflict (master)

$ > Test.txt

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Conflict (master)

$ git add Test.txt

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Conflict (master)

$ git commit -m "Init done"

[master (root-commit) 2fcee69] Init done

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100644 Test.txt

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Conflict (master)

$ git branch

\* master

- Thay đổi thông tin trong Test.txt trong master và commit:

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Conflict (master)

$ vim Test.txt



17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Conflict (master)

$ git commit -m "done"

On branch master

- Tạo hai nhánh br\_1 và br\_2:

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Conflict (master)

$ git branch br\_1

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Conflict (master)

$ git branch br\_2

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Conflict (master)

$ git branch

br\_1

br\_2

\* master

- Chuyển qua nhánh 1, chỉnh sửa nội dung trong file Test.txt và commit:

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Conflict (master)

$ git checkout br\_1

Switched to branch 'br\_1'

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Conflict (br\_1)

$ vim Test.txt

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Conflict (br\_1)

$ git commit -a -m "done"

warning: LF will be replaced by CRLF in Test.txt.

The file will have its original line endings in your working directory

[br\_1 8b05e19] done

1 file changed, 1 insertion(+), 1 deletion(-)

Nội dung file mới: 

- Chuyển qua nhánh 2, chỉnh sửa và commit:

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Conflict (br\_1)

$ git checkout br\_2

Switched to branch 'br\_2'

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Conflict (br\_2)

$ vim Test.txt

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Conflict (br\_2)

$ git commit -a -m "done"

[br\_2 a3035a6] done

1 file changed, 1 insertion(+), 1 deletion(-)

Nội dung file mới: 

- Merge br\_1 vào master:

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Conflict (br\_2)

$git checkout master

Switched to branch 'master'

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Conflict (master)

$ git merge br\_1

Updating 9c91609..8b05e19

Fast-forward

Test.txt | 2 +-

1 file changed, 1 insertion(+), 1 deletion(-)

=> Nội dung của Test.txt trong master sau khi merge:



- Merge br\_2 vào master => Xảy ra conflict do cùng sửa một file (được đề cập là Test.txt như bên dưới):

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Conflict (master)

$ git merge br\_2

Auto-merging Test.txt

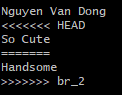
CONFLICT (content): Merge conflict in Test.txt

Automatic merge failed; fix conflicts and then commit the result.

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Conflict (master|MERGING)

$

- Mở Test.txt trong master lên:



- Xoá các dòng ký tự được thêm sau đó lưu lại và commit => merge hoàn tất và conflict được giải quyết.



17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Conflict (master|MERGING)

$ vim Test.txt

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Conflict (master|MERGING)

$ git commit -a -m "done"

[master 8cc860a] done

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Conflict (master)

$

***\*Xoá local branch:***

- Trạng thái fully merged: Khi một branch không phải master được tạo ra và có một số commit nhưng không được merge với master thì được gọi là not fully merged.

- Xoá bằng lệnh sau:

git branch -d branch\_name

*Trong đó branch\_name chính là tên của branch mà bạn muốn xóa. Lưu ý là bạn không thể xóa branch mà bạn đang làm việc, ví dụ bạn đang làm việc trên branch client thì bạn không thể xóa branch client, thay vào đó ban phải checkout sang branch khác để xóa branch client.*

*Khi chạy lệnh trên sẽ có hai trường hợp xảy ra.*

**Trường hợp 1**: Nếu branch đang ở trạng thái fully merged thì bạn xóa bình thường.

**Trường hợp 2**: Nếu branch đang ở trạng thái not fully merged thì git sẽ cảnh báo, lúc này ban phải sử dụng lệnh sau để xóa.​

git branch -D branch\_name

***\*Lệnh commit --amend:***

- Trong một số trường hợp bạn commit nhưng bị quên add một số file nào đó và bạn không muốn tạo ra một commit mới thì có thể sử dụng lệnh commit kết hợp tham số --amend để gộp các file đó và bổ sung vào commit cuối cùng, vì vậy không tạo ra commit mới.

git commit --amend

git commit --amend -m "Message mô tả thay đổi"

git commit --amend --author "Nguyen Huu Kim <nguyen.huu.kim@framgia.com>"

- Nếu thực hiện lệnh này ngay sau commit cuối cùng mà không thay đổi gì thêm thì SNAPSHOT cuối cùng cũng không thay đổi gì, message có thể sẽ thay đổi và ghi đè lên trước đó. (Lưu ý là add file mới xong rồi mới commit)

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Conflict (master)

$ git add Test\_2.txt

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Conflict (master)

$ git commit --amend

[master 955b775] done but add more file

Date: Sun Mar 22 14:03:10 2020 +0700

***\*Loại bỏ tập tin đã lưu vào Stage:***

- Stage (Update + Add *nói bên trên rồi*):

- Ví dụ:

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Conflict (master)

$ > DongNV.txt

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Conflict (master)

$ > NV.txt

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Conflict (master)

$git add DongNV.txt

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Conflict (master)

$ git add NV.txt

- Có thể xem trạng thái hiện tại:

git status

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Conflict (master)

git status

On branch master

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

new file: DongNV.txt

new file: NV.txt

Changes not staged for commit:

(use "git add/rm <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

deleted: Test.txt

deleted: Test\_2.txt

- Theo hướng dẫn này thì để unstage một file mới thêm cần lệnh:

use "git restore --staged <file>

Unstage NV.txt:

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Conflict (master)

$ git restore --staged NV.txt

- Xem lại status:

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Conflict (master)

$ git status

On branch master

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

new file: DongNV.txt

Changes not staged for commit:

(use "git add/rm <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

deleted: Test.txt

deleted: Test\_2.txt

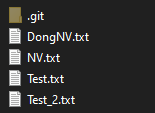
Untracked files:

(use "git add <file>..." to include in what will be committed)

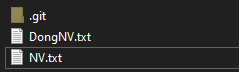
NV.txt

***\*Xoá một file ra khỏi branch:***

- Trước tiên xem cây thư mục master hiện tại:



- Thực hiện xoá trực tiếp hai file trên master:



- Sau đó tiến hành commit:

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Conflict (master)

$git commit -m "delete"

On branch master

Changes not staged for commit:

(use "git add/rm <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

deleted: Test.txt

deleted: Test\_2.txt

Untracked files:

(use "git add <file>..." to include in what will be committed)

NV.txt

no changes added to commit (use "git add" and/or "git commit -a")

- Lúc này thì việc xoá các file này không được staged lại, cần thực hiện một trong 2 hành động là update các file cần commit hoặc huỷ sự xoá của 2 file này (trong đối tượng commit và tree trước đó thì vẫn còn hai file này và có thể khôi phục lại).

- Giả sử, không thực hiện một trong hai thao tác trên mà chuyển qua nhánh khác xong quay lại và xem cây thư mục:

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Conflict (master)

$ git checkout br\_1

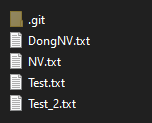
Switched to branch 'br\_1'

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Conflict (br\_1)

$ git checkout master

Switched to branch 'master'

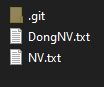
*Cây thư mục sau khi quay lại:*



*Do hai file kia chưa staged mặc dù đã commit branch này. Như vậy sau khi xoá, cần phải staged sự bị xoá của chúng trong git để git có thể biết và việc xoá lúc này mới có tác dụng nếu không đối tượng commit hiện tại sẽ lấy các file trước đó đã staged (hai file ta đã xoá) và cho vào đối tượng commit hiện tại. (xoá cũng tính là staged)*

- Giờ mới là là xoá thực sự:

+ Xoá hai file trong cây thư mục:



+ Khai báo việc xoá hai file này cho git bằng lệnh git rm <tên\_file>

git rm <tên\_file>

Khai báo xoá Test.txt:

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Conflict (master)

$ git rm Test.txt

rm 'Test.txt'

Xem lại status:

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Conflict (master)

$ git status

On branch master

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

deleted: Test.txt

Changes not staged for commit:

(use "git add/rm <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

deleted: Test\_2.txt

Untracked files:

(use "git add <file>..." to include in what will be committed)

NV.txt

=> Lúc này chỉ còn lại file Test\_2.txt là chưa staged trạng thái delete:

Khai báo xoá cho Test\_2.txt:

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Conflict (master)

$git rm Test\_2.txt

rm 'Test\_2.txt'

+ Tiến hành commit: (sẽ thấy có thông báo vễ sự xoá file)

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Conflict (master)

$ git commit -m "done"

[master 61475f7] done

2 files changed, 3 deletions(-)

delete mode 100644 Test.txt

delete mode 100644 Test\_2.txt

Đổi qua nhánh khác, sau đó quay lại:

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Conflict (master)

$git checkout br\_1

Switched to branch 'br\_1'

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Conflict (br\_1)

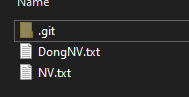
$ git checkout master

Switched to branch 'master'

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Conflict (master)

$

Xem lại cây thư mục:



=> Như vậy thì hai file kia đã bị xoá hoàn toàn.

***Như vậy xoá và sau đó rm một file cũng làm cho file đó ở trạng thái staged (in the Recycle Bin)***

*\*Các lệnh git stash: là lệnh dùng để thiết lập trạng thái ban đầu cho tất cả các file trong thư mục làm việc. Trạng thái ban đầu ở đây chính là nội dung dữ liệu ban đầu của file (*so với commit cuối cùng hoặc pull từ remote repository*):*

- Lệnh git stash sẽ có tác dụng với ***tất cả dữ liệu đang hoạt động trong working directory với điều kiện là dữ liệu đó đã được đưa vào trạng thái Staged hoặc đã từng được committed.***

- Cú pháp cơ bản: git stash

*- Nếu file bạn mới tạo và chưa add thì chỉ cần xoá hết nội dung là được, vì nó chưa được staged mà không cần dùng lệnh.*

- Nếu file đã staged hoặc được committed thì dùng lệnh để đưa nó về trạng thái ban đầu.

Ví dụ:

+ Tạo một Repos và một file:

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access

$ mkdir StashTest

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access

$ cd StashTest/

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/StashTest

$ git init

Initialized empty Git repository in E:/Dual Access/StashTest/.git/

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/StashTest (master)

$ > Test.txt

+ Tiến hành thay đổi nội dung và commit:

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/StashTest (master)

$ vim Test.txt



17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/StashTest (master)

$ git add Test.txt

warning: LF will be replaced by CRLF in Test.txt.

The file will have its original line endings in your working directory

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/StashTest (master)

$ git commit -m "inti done"

[master (root-commit) 556666e] inti done

1 file changed, 1 insertion(+)

create mode 100644 Test.txt

+ Thử lệnh git stash:

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/StashTest (master)

$ git stash

No local changes to save

Kiểm tra lại nội dung file thì không có gì thay đổi. ☺

+ Thay đổi nội dung trong Test.txt và Staged nó:

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/StashTest (master)

$ vim Test.txt

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/StashTest (master)

$ git add Test.txt

warning: LF will be replaced by CRLF in Test.txt.

The file will have its original line endings in your working directory

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/StashTest (master)

$ git status

On branch master

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

modified: Test.txt



+ Test lệnh => nội dung file lúc này là Nguyen Van Dong

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/StashTest (master)

$ git stash

Saved working directory and index state WIP on master: 556666e inti done

+ Thử đổi và commit thay vì chỉ staged:

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/StashTest (master)

$ vim Test.txt

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/StashTest (master)

$ git add Test.txt

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/StashTest (master)

$ git commit -m "test stash"

[master 9020f61] test stash

+ Test lệnh:

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/StashTest (master)

$git stash

No local changes to save

+ Kiểm tra xem file bị reset nội dung nhưng có bị unstaged không => Không bị unstaged mà chỉ bị reset nội dung.

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/StashTest (master)

$ git status

On branch master

nothing to commit, working tree clean

=> Stash chỉ có tác dụng với các file staged nhưng chưa commit.

- Một số lệnh liên quan tới git stash:

+ git stash list: Xem các lần stash và thay đổi của nó:

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/StashTest (master)

$ git stash list

stash@{0}: WIP on master: 556666e inti done

stash@{1}: WIP on master: 556666e inti done

+ git stash pop và git stash drop: bó tay ☺

+ git stash show: xe dữ liệu thay đổi trong các lần stash:

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/StashTest (master)

$ git stash show

Test.txt | 2 +-

1 file changed, 1 insertion(+), 1 deletion(-)

Nó có một tham số đó là địa chỉ của stash muốn xem $ git stash show <stash>. Nếu bạn không truyền <stash> thì mặc định nó sẽ lấy stash mới nhất.

+ git stash clear: xóa tất cả stash ra khỏi history. Khi đó ds không còn gì hết:

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/StashTest (master)

$ git stash clear

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/StashTest (master)

$ git stash list

+ …

***\*Lệnh rebase:***

- Gộp nhiều commit thành 1 commit:

***\*Xem lại danh sách n commit gần đây:***

# Cú pháp: git log -<n>

> git log -5

# Hiển thị mỗi commit trên một dòng

> git log --oneline -5

- Ví dụ:

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Courses (new\_branch)

$ git log

commit 669f501b9e230d6890fdd231f37b50401b96862e (HEAD -> new\_branch, courses/new\_branch)

Author: 17520350@gm.uit.edu.vn <33110542+dongnvsince1999@users.noreply.github.com>

Date: Mon Mar 23 01:21:37 2020 +0700

update git bash

commit 20452477991db632957fc5e832b541df85c2464b

Author: 17520350@gm.uit.edu.vn <33110542+dongnvsince1999@users.noreply.github.com>

Date: Mon Mar 23 01:13:19 2020 +0700

not thin

commit 32d738c590b3fb3016d4049afef28f7123e9738e (courses/master)

Author: 17520350@gm.uit.edu.vn <33110542+dongnvsince1999@users.noreply.github.com>

Date: Mon Mar 23 00:49:53 2020 +0700

init courses

***\*So sánh các thay đổi của code trước khi commit:***

git diff

- Ví dụ:

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Courses (new\_branch)

$ git diff

**diff --git a/Git Bash.docx b/Git Bash.docx**

**index 02590bb..0770f75 100644**

**--- a/Git Bash.docx**

**+++ b/Git Bash.docx**

@@ -2,6 +2,7 @@

\*Mục đích của file này là muốn sử dụng thành thạo git bash khi thấy ngta khá là chuyên nghiệp trong việc sử dụng git với git bash và cũng không muốn đóng kén phần mềm git bash đã quá lâu

I. Git là gì?

- Git là một hệ thống quản lý phiên bản phân tán (distributed version controler system).

+\*VSC (Version Control System): ... khỏi để cập.

- Tác dụng của Git:

+ Lưu lại các phiên bản khác nhau của mã nguồn dự án phần mềm.

+ Khôi phục lại mã nguồn từ một phiên bản bất kỳ.

@@ -23,6 +24,13 @@ II. Các khái niệm cơ bản:

3. Commit:

- Commit là thao tác báo cho hệ thống biết bạn muốn lưu trữ lại trạng thái hiện hành.

+- Mỗi commit được tạo ra gồm một số thông tin chính:

+ + Tên và email của ngưởi tạo commit

+ + Ngày giờ commit được tạo ra

+ + Message: Mô tả cho sự các sửa đổi (Để sau này xem lại còn biết nó sửa vì mục đích gì)

+ + Một ID là một chuỗi SHA gồm 2 dạng: Full và Short.

+ Full: Chuỗi SHA đầy đủ

+ Short: Là 7 ký tự đầu tiên được cắt ra từ chuỗi Full

- Khi thực hiện commit, trong repos sẽ ghi lại sự khác biệt từ lần commit trước để với trạng thái hiện tại. Các commit ghi nối tiếp nhau theo thứ tự thời gian nên hoàn toàn có thể theo vết các commit và biết được lịch sử thay đổi trong quá khứ.

- Khi thực hiện commit thì hệ thống luôn yêu cầu nhập một commit message để tóm tắt lại những gì đã thực hiện tại lần lưu này.

@@ -343,6 +351,11 @@ git branch -D branch\_name

\*Lệnh commit --amend:

- Trong một số trường hợp bạn commit nhưng bị quên add một số file nào đó và bạn không muốn tạo ra một commit mới thì có thể sử dụng lệnh commit kết hợp tham số --amend để gộp các file đó và bổ sung vào commit cuối cùng, vì vậy không tạo ra commit mới.

git commit --amend

+

+git commit --amend -m "Message mô tả thay đổi"

+

+git commit --amend --author "Nguyen Huu Kim <nguyen.huu.kim@framgia.com>"

+

- Nếu thực hiện lệnh này ngay sau commit cuối cùng mà không thay đổi gì thêm thì SNAPSHOT cuối cùng cũng không thay đổi gì, message có thể sẽ thay đổi và ghi đè lên trước đó. (Lưu ý là add file mới xong rồi mới commit)

:

*\*Commit với message nhiều dòng:*

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Courses (new\_branch)

$ git commit -m 'First line

> Second line'

[new\_branch 16f053a] First line Second line

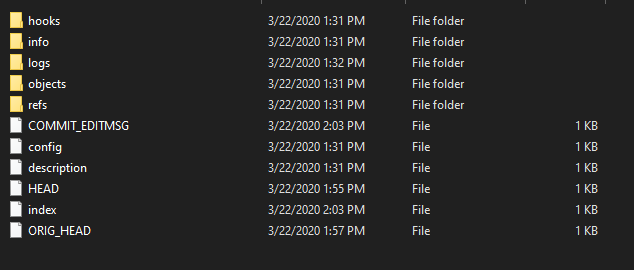
1 file changed, 0 insertions(+), 0 deletions(-)

-

**III. Git Objects:**

***\*Git Object là gì?***

- Khi chạy lệnh git init, thì một thư mục ẩn được tạo ra, thư mục này chứa toàn bộ db và các thao tác quan trọng của project, gồm một số thư mục con và file quan trọng: HEAD, branches/, config, description, hooks/ index, info/, objects/, refs/



- Trong đó thư mục object khá quan trọng vì nó chứa toàn bộ db, gồm có 4 object như sau:

* **tree**: là directory
* **blob**: là file
* **commit**
* **tag**

- Mỗi object được định danh bởi một Unique ID có chiều dai 40 ký tự dựa vào thuật toán ban SHA – 1: Ex: 73b41f9f027614a0417edc7cca83ea5065eb36

***1. Mô hình đối tượng một branch:***

- Khi thực hiện commit, Git tạo ra một đối tượng commit và có chứa các con trỏ tới nội dung của stage, ngoài ra còn có thể chứa thông tin tác giả, thông điệp.

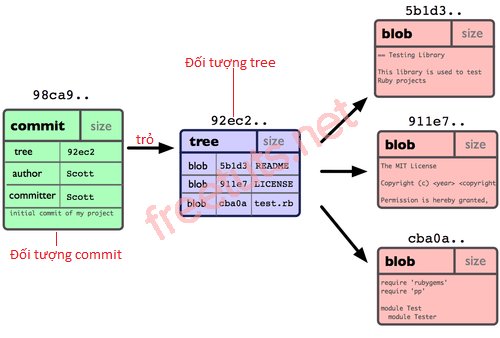
- Commit đầu tiên sẽ không có cha, các commit tiếp theo sẽ có thể có 1 hay nhiều cha.

- Ví dụ:

Giả sử có 3 tập tin và bạn tổ chức cả ba tập tin vào stage để commit (*README test.rb LICENSE*), lúc này bạn sẽ thực hiện lệnh commit như sau:

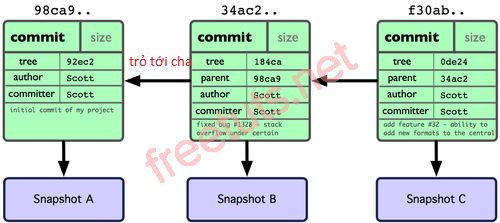
|  |  |
| --- | --- |
| . | $ git add README test.rb LICENSE  $ git commit -m "Khoi dong du an"  [master 517333f] Khoi dong du an   3 files changed, 0 insertions(+), 0 deletions(-)   create mode 100644 LICENSE   create mode 100644 README   create mode 100644 test.rb |

- Lệnh **commit** sẽ băm cả 3 file và ***lưu chúng dưới dạng*** **đối tượng tree** (thư mục), ***trong tree sẽ chứa tất cả các blob*** (tức là 3 file trên) ***và mỗi blod sẽ trỏ đên file gốc của nó.*** ***Sau đó nó tạo một*** **đối tượng commit** chứa các thông tin metadata như author, email, message ... ***và đặc biệt là đối tượng commit này có một con trỏ trỏ tới*** **đối tượng tree**, vì vậy ta có thể tái tạo lại history thông qua đối tượng commit này.



Như vậy đối tượng tree và các blob ta gọi là **snapshot**.

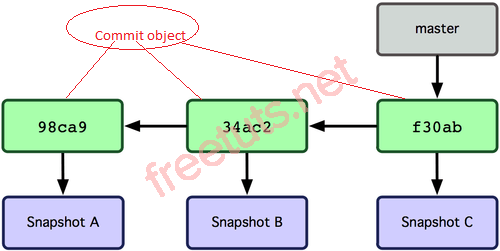
- Nếu bạn thay đổi dữ liệu cho một trong ba file và tiếp tục thực hiện một lệnh **commit** thì lúc này ngoài các thông tin như trên, ***đối tượng commit còn tạo thêm một con trỏ và trỏ tới đối tượng commit trước đó***. Sau 2 commit thì lúc này mô hình kho chứa sẽ như sau:



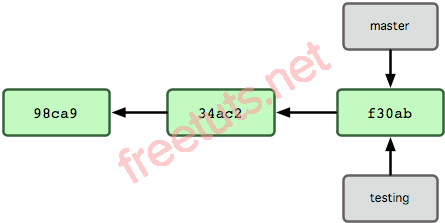
***2. Mô hình đối tượng nhiều branch:***

***­***- Khi bạn ***tạo một nhánh (*branch*) thực chất là bạn đang tạo một con trỏ và trỏ tới commit mới nhất tại thời điểm thực thi lệnh***. Chúng ta có tên nhánh mặc định là **Master Branch.** Như ở ví dụ trên thì chúng ta đang thực thi các commit tại branch master nên mô hình lúc này dạng như sau:

*Commit trên một nhánh master.*



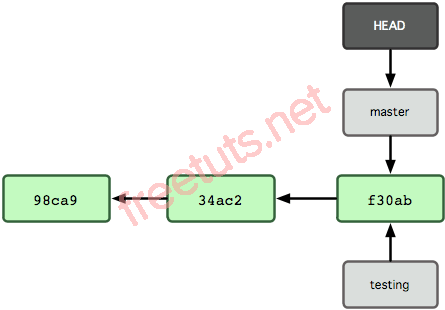
- Khi tạo một nhánh mới thì mô hình sẽ như sau (ở đây là tạo một nhánh testing trỏ tới commit có địa chỉ là f30ab.



***3. Con trỏ HEAD:***

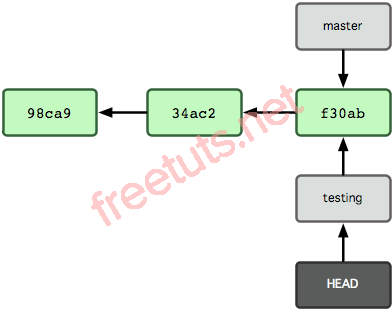
- Khi làm việc trên nhiều branch thì để biết được đang làm việc trên branch nào, Git có một con trỏ đặc biệt gọi là **con trỏ HEAD.**

- Khi đang làm việc trên master thì con trỏ HEAD sẽ trỏ tới Master:



- Nếu chuyển qua testing thì mô hình sẽ có dạng như sau:

$ git checkout testing

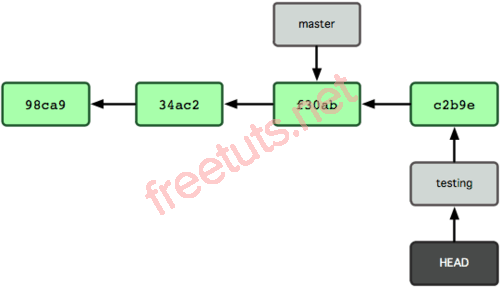


- Nếu thực hiện một commit tại branch testing thì mô hình có dạng như sau:

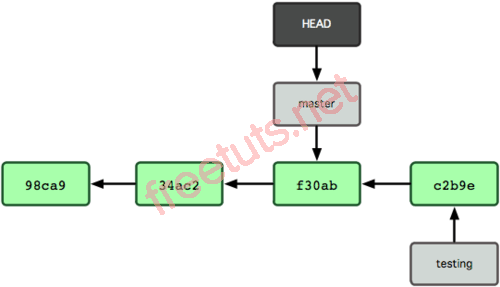
*Code để commit một chỉnh sửa*

$ vim test.rb

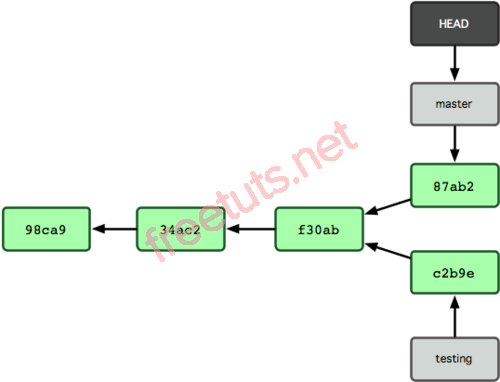
$ git commit -a -m 'made a change'



- Chuyển qua master và mô hình sẽ như sau:



- Nếu thực hiện commit tại master thì lịch sử sẽ bị tách ra:



**V. Làm việc với Remote Repository:**

- Để làm việc với Remote Repository thì cần **phải có server** và cài đặt Git Remote trên đó, điều này rất ít bạn có điều kiện làm, may ra công ty lớn thì mới có. Tuy nhiên hiện nay có một số server cung cấp dịch vụ miễn phí (có phí) như **github, gitlab** ... nên bạn có thể sử dụng dịch vụ này của họ để học và quản lý dự án của riêng mình.

- Bước đầu tiên bạn cần phải thêm một remote repository vào local repository, điều này có nghĩa là phiên bản tại local sẽ là nơi làm việc của bạn và phiên bản trên remote chính là nơi lưu trữ mã nguồn chính của dự án, tại remote các thành viên sẽ thực hiện các thao tác **pull** và **push** để đưa code và lấy code về local.

***\*Ví dụ trong các phần tiếp theo lấy server trên Github.***

***1. Thêm Remote Repos:***

- Cú pháp thêm remote repos vào local repos:

git remote add [shortname] [url]

Trong đó:

shortname: là tên mình muốn đặt cho remote repo.

url: là đường dẫn tới remote repos.

- Ví dụ: (mang Courses lên Github ☺):

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access

$ cd Courses/

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access

$ cd git init

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Courses (master)

$ git add \*

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Courses (master)

$ git status

On branch master

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Courses (master)

$ git commit -m "init Courses"

[master (root-commit) cd5fe88] init Courses

114 files changed, 520 insertions(+)

- Tạo một repos trên github có link bên dưới:

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Courses (master)

$ git remote add courses https://github.com/dongnvsince1999/Courses.git

=> Đã thêm remote repos xong.

*Trường hợp đặt tên remote đã tồn tại thì sẽ bị thông báo lỗi là* "fatal: remote origin already exists."*Để giải quyết vấn đề này bạn có thể thực hiện 3 thao tác sau:*

1. *Chọn một cái tên khác.*
2. *Đổi tên của remote đã tồn tại đó sang tên khác.*
3. [*Xóa remote*](https://freetuts.net/git-xoa-remote-repository-1100.html) *đã tồn tại đó.*

- Kiểm tra remote đã thêm: git remote

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Courses (master)

$ git remote

courses

- Kiểm tra chi tiết remote đã thêm: git remote -v

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Courses (master)

$ git remote -v

courses https://github.com/dongnvsince1999/Courses.git (fetch)

courses https://github.com/dongnvsince1999/Courses.git (push)

- Thay đổi link của remote repos: (để đây chứ không test)

$ git remote set-url courses https://github.com/dongnvsince1999/courses\_repo.git

***2. Xoá Remote Repos:***

- Trường hợp bạn đã thêm một remote repo nhưng sau đó bạn nhận thấy đã thêm nhầm link, hoặc bạn muốn xóa nó để thêm một remote khác thì có thể sử dụng lệnh xóa remote của git.

- Cú pháp cơ bản: git remote rm remote\_name

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Courses (master)

$ git remote rm courses

- Kiểm tra lại xem repos đang có remote nào không:

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Courses (master)

$ git remote

=> không hiển thị gì là xoá được ☺. Giờ add lại ☺

*Nếu bạn xóa một remote không tồn tại thì sẽ xuất hiện lỗi* "fatal: No such remote: remote\_name". *Ví dụ, khi xóa remote courses nhưng remote này không hề tồn tại nên sẽ nhận được thông báo lỗi là* "fatal: No such remote: courses". Xem hình bên dưới:

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Courses (master)

$ git remote rm courses

fatal: No such remote: 'courses'

***3. Đổi tên Remote Repos:***

git remote rename old\_name new\_name

***4.Push data lên remote:***

*\*Đẩy mới một branch lên lêm remote:*

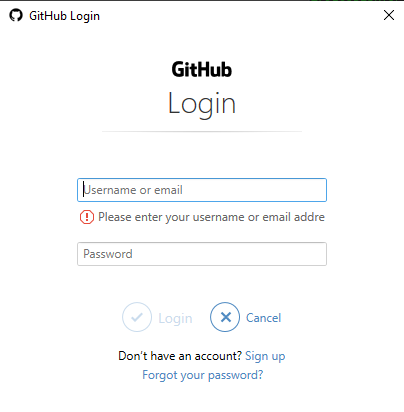
git push --set-upstream [remote] [branch]

- Nếu đang khởi động dự án thì công việc đầu tiên là thêm repository và đẩy branch từ local lên remote.

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Courses (master)

$ git push --set-upstream courses master

- Khi nhập lệnh này thì git bash sẽ yêu cầu nhập thông tin username và password (của github).



- Kết quả sau khi thực hiện lệnh thêm branch master vào Remote trên github:

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Courses (master)

$ git push --set-upstream courses master

Enumerating objects: 127, done.

Counting objects: 100% (127/127), done.

Delta compression using up to 4 threads

Compressing objects: 100% (115/115), done.

Writing objects: 100% (127/127), 160.97 MiB | 1.96 MiB/s, done.

Total 127 (delta 5), reused 0 (delta 0)

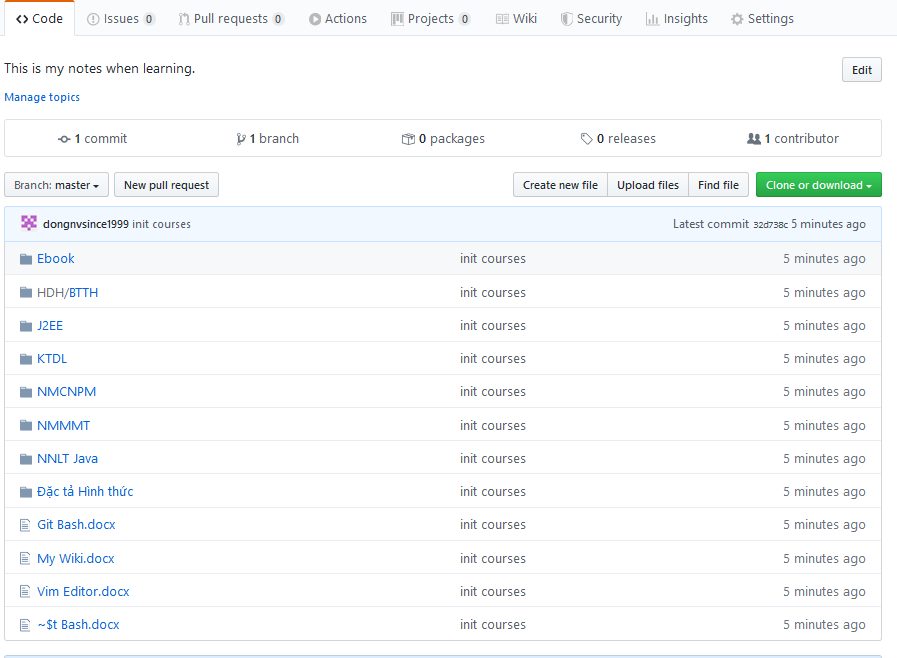
remote: Resolving deltas: 100% (5/5), done.

To https://github.com/dongnvsince1999/Courses.git

\* [new branch] master -> master

Branch 'master' set up to track remote branch 'master' from 'courses'.

- Trên github lúc này sẽ có một Repos như sau (gồm 1 commit và 1 branch):



- Tạo một branch mới ở local và chỉnh sửa sau đó commit:

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Courses (master)

$ git branch new\_branch

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Courses (new\_branch)

$ git add \*

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Courses (new\_branch)

$ git commit -m "not thin"

[new\_branch 2045247] not thin

1 file changed, 0 insertions(+), 0 deletions(-)

- Trên Github vẫn không có branch này, đưa lên bằng lệnh:

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Courses (new\_branch)

$ git push --set-upstream courses new\_branch

Enumerating objects: 5, done.

Counting objects: 100% (5/5), done.

Delta compression using up to 4 threads

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

Writing objects: 100% (3/3), 62.37 KiB | 1.69 MiB/s, done.

Total 3 (delta 2), reused 0 (delta 0)

remote: Resolving deltas: 100% (2/2), completed with 2 local objects.

remote:

remote: Create a pull request for 'new\_branch' on GitHub by visiting:

remote: https://github.com/dongnvsince1999/Courses/pull/new/new\_branch

remote:

To https://github.com/dongnvsince1999/Courses.git

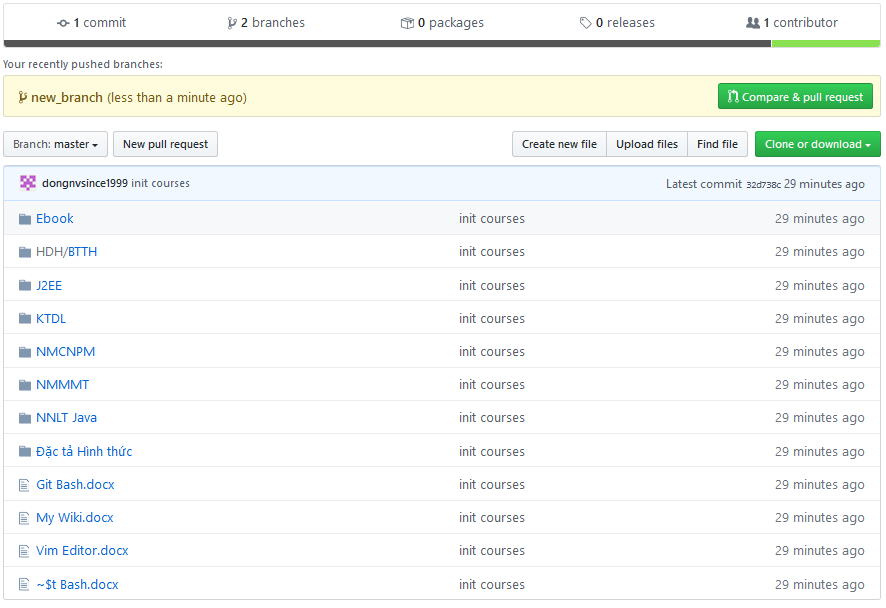
\* [new branch] new\_branch -> new\_branch

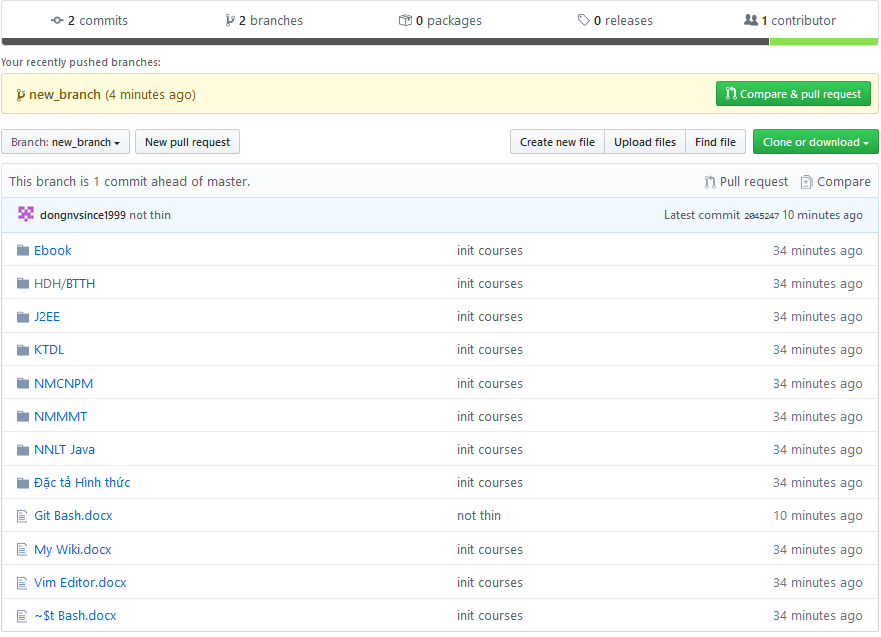
Branch 'new\_branch' set up to track remote branch 'new\_branch' from 'courses'.

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Courses (new\_branch)

$

- Lúc nay trên github đã có branch là new branch:

**



- Thay đổi và commit new\_branch:

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Courses (new\_branch)

$ git add \*

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Courses (new\_branch)

$ git commit -m "update git bash"

[new\_branch 669f501] update git bash

1 file changed, 0 insertions(+), 0 deletions(-)

- Trên Github vẫn không xuất hiện commit này, muốn đẩy commit lên thì thực hiện như phần bên dưới.

*\*Đẩy commit lên remote:*

- Sau khi thiết lập với lệnh git push --set-upstream thì trên remote đã có một branch và toàn bộ mã nguồn, vì vậy các thao tác tiếp theo bạn chỉ cần sử dụng lệnh push bình thường. Mỗi lần push sẽ được yêu cầu mật khẩu.

git push

17520@DESKTOP-60R8H26 MINGW64 /e/Dual Access/Courses (new\_branch)

$ git push

Enumerating objects: 5, done.

Counting objects: 100% (5/5), done.

Delta compression using up to 4 threads

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

Writing objects: 100% (3/3), 62.12 KiB | 1.51 MiB/s, done.

Total 3 (delta 2), reused 0 (delta 0)

remote: Resolving deltas: 100% (2/2), completed with 2 local objects.

To https://github.com/dongnvsince1999/Courses.git

2045247..669f501 new\_branch -> new\_branch

- Lúc này new\_branch đã có 3 commit:



*\*Lưu thông tin đăng nhập:*

- Chạy lệnh $ git config credential.helper store

- Thực hiện một thao tác push, lần này git sẽ bắt bạn đăng nhập, nhưng kể từ đó bạn sẽ không cần phải đăng nhập khi thao tác với remote repo.

**V. Các lệnh tương tác giữa Remote Repos và Local Repos:**

***1. Tạo ra một Local Repos từ Server Repos:***

git clone repository\_url

-