Trước khi sử dụng Git thì bạn cần phải cấu hình hai thông số đó là username và email, nếu không bạn sẽ không thể thực hiện các thao tác như commit, clone, stash ...

**Để cấu hình username ta sử dụng lệnh sau:**

git config --global user.name "Tên của bạn"

**Để cấu hình email ta sử dụng lệnh sau:**

git config --global user.email "Email của bạn"

* **Những lệnh git thường sử dụng**

**Đi chuyển đến thư mục:**

cd duong\_dan\_toi\_thu\_muc

**Đổi ổ đĩa:**

cd /TenODia

**Lùi 1 cấp:**

cd ..

**Đến thư mục của User trên windows:**

cd ~

**Xem danh sách file và folder:**

ls duongdan

*nếu không truyền đường dẫn nó sẽ lấy đường dẫn hiện tại.*

**Tạo folder mới:**

mkdir name\_folder

**Xóa folder:**

rmdir name\_folder

**Xem đường dẫn thư mục hiện tại:**

pwd

**Tạo repository:**

git init

*nếu kết quả hiện ra là: initialized empty Git repository in .... thì đã thành công*

Đối với mỗi dự án sẽ được leader chia thành nhiều task nên ta phải sử dụng branch , branch hoạt động riêng lẻ nên không ảnh hưởng đến nhau.

Branch là những phân nhánh ghi lại những lịch sử, có thể có nhiều thay đổi đồng thời trên repository, giúp giải quyết nhiều nhiệm vụ cùng lúc.

**Làm việc với branch**

**Branch master:**

Khi tạo repository thì Git thiết lập branch mặc định là master, nghĩ là nó sẽ tự động tạo một branch master và mọi hoạt động lúc này đều trên branch master. Nó đóng vai trò cập nhập dữ liệu và đồng bộ với repository.

Giả sử có 10 tasks, thì ta tạo các branch khác và branch master chỉ để pull code tử remote branch master và merge với các branch còn lại. Mỗi branch làm việc sẽ được ghi lại lịch sử nên trong quá trình làm việc ta có thể rollback dựa vào index mà Git đã lưu.

**Tạo một branch:**

Tạo một file rồi commit lên branch master thì lúc này branch master sẽ được tạo.

.git là folder chứa history của Git.

**Add file vào history:**

git add demo.txt

**commit lên branch master, lúc này branch master mới được tạo:**

git commit -m”ghi chú”

**Lệnh tạo mới branch:**

git branch namebranch

**Xem danh sách branch:**

git branch

*nếu branch có dấu \* trước là branch hiện tại.*

**Chuyển đổi giữa các branch:**

git checkout namebranch

*nếu kết quả trả về là swiched to branch “namebranch” thì thực hiện chuyển đổi thành công.*

**Git commit branch:**

Để lưu lại những thay đổi vào branch thì ta sử dụng commit

git commit -m”message”

*Trước khi commit thì ta phải khai báo sẽ commit file nào.*

git add filename

git commit -m”message”

**Git merge branch:**

Để merge một branch bất kì vào branch hiện tại thì:

git merge branch name\_branch

**Xử lý xung đột:**

Giả sử ta làm việc trên 2 task1 và task2 và cùng sửa 1 file, lúc này merge task1 vào task2 sẽ bị xung đột(conflict), vì vậy ta phải sửa sung đột sau đó mới merge được vào.

$ git merge branch1

Auto-merging demo.txt

CONFLICT (content): Merge conflict in demo.txt

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

ta sửa file xung đột rồi commit lại.

**Git – xóa local branch**

Sau khi thực hiện xong nhiệm vụ, ta merge các branch vào master sau đó ta sẽ xóa các branch để tránh xung đột và quản lý mã nguồn tốt hơn.

git branch -d branch\_name

TH1: branch chưa merge vào branch master thì nó sẽ hiện cảnh báo. Xác nhận xóa:

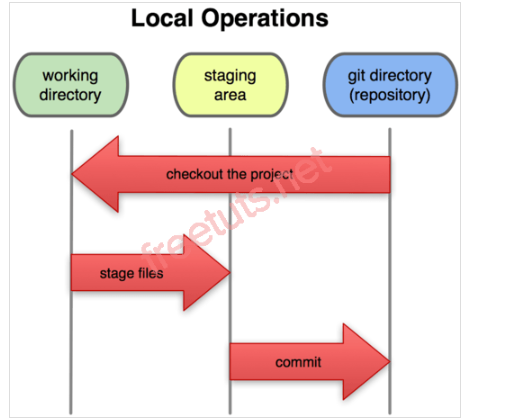
git branch -D branch\_name

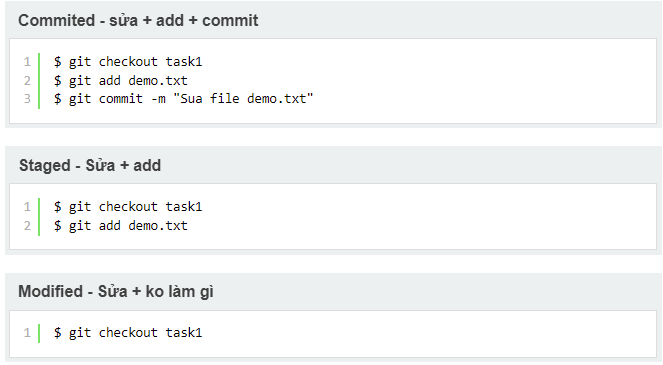
TH2: branch ở chế độ fully merged thì xóa bình thường.

**Làm việc với commit**

**Mỗi tập tin trong Git được quản lý dựa trên 3 trạng thái là: committed, modified và staged:**

* Committed: những dữ liệu đã được lưu một các an toàn trong cơ sở dữ liệu, commit thành công.
* Modified: thay đổi tập tin nhưng chưa commit vào cơ sở dữ liệu, tức là chưa sử dụng git add và git commit.
* Staged: đánh dấu sẽ commit phiên bản hiện tại của một tập tin đã chỉnh sửa trong lần commit sắp tới. Trạng thái này xảy ra khi git add nhưng chưa git commit.
* 3 trạng thái của một dự án sử dụng Git.
* Khu vực Git: là thư mục lưu trữ siêu dữ liệu(metadata) và cơ sở dữ liệu của dự án, thư mục này bị ẩn bới hdd windows nên phải bật hiện thị file ẩn. Khu vực này tiếp nhận và lưu trữ các commit từ stage area.
* Khu vực làm việc: nếu không sử dụng remote repository thì đây là bản sao của dự án, còn không thì đây là thư mục chính của dự án và branch master chính là bản chính, còn các branch mới tạo là bản sao.
* Khu vực tổ chức: đây là tập tin đơn giản nằm trong thư mục git, nó sẽ chứa thông tin về trạng thái của một file trong dự án.





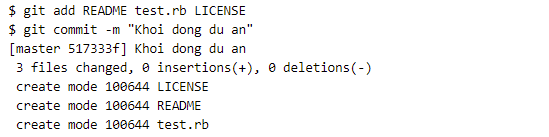
**Git object:**

* Khi ta sử dụng lệng $ git init thì một thư mục ẩn được tạo ra đó là .git, thư mục này sẽ chứa toàn bộ database và các thao tác của dự án, nó có một số thư mục con và một số file quan trọng gồm: HEAD, branches, config, description, hooks, index, info, object, refs.
* Cần chú ý đến thư mục object nó chứa toàn bộ database, nó có 4 object sau:
  + Tree: là directory
  + Blob: là file
  + Commit
  + Tag

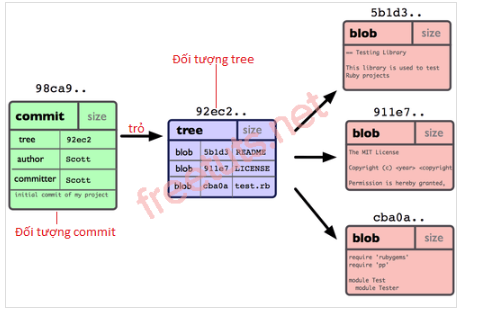
**Mô hình đối tượng 1 branch**

khi ta thực hiện một commit, Git sẽ tạo một đối tượng commit và có chứa các con trỏ tới ảnh của nội dung mà ta tổ chức tại stage, ngoài ra còn chứa thông tin tác giả, thông điệp, cũng có thể chứa các con trỏ khác trỏ tới các commit cha của commit đó. Commit đầu tiên không có cha. Commit thứ hai chở đi sẽ có một cha hoặc nhiều cha nếu nó được merge từ nhiều branch.

Giả sử có 3 tạp tin và ta tổ chức cả 3 tập tin đó vào stage để commitm, lúc này ta sẽ thực hiện lệnh commit như sau:



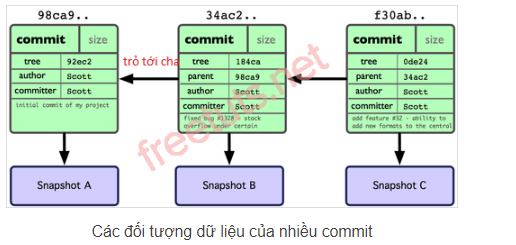
Lệnh commit sẽ băm cả 3 file và lưu chúng dưới dạng đối tượng tree, trong tree chứa tất cả các blob và mỗi blob sẽ trỏ đến file gốc của nó. Sau đó tạo một đối tượng commit chứa các thông tin như tác giả, email, thông điệp... và đặc biệt là là đối tượng commit này có một con trỏ trỏ tới đối tượng tree, vì thế ta có thể tái tạo lại history thông qua đối tượng commit này.



Dữ liệu kho chứa của một commit (snapshot).

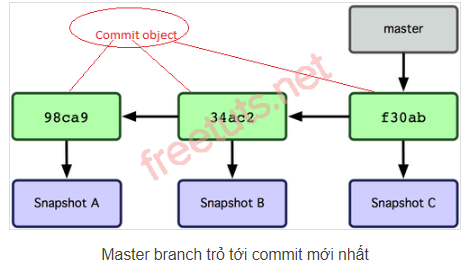
Như vậy tree và các blob gọi lại snapshot.

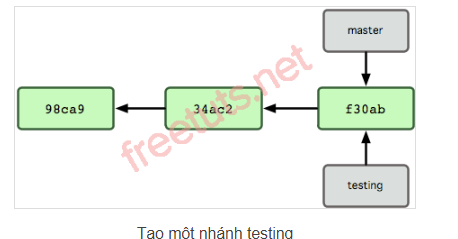
Nếu ta thay đổi dữ liệu 1 trong 3 file và tiếp tục lệnh commit thì lúc này ngoài các thông tin nhưu trên, đối tượng commit còn tạo thêm một con trỏ và trỏ tới đối tượng commit trước đó. Sau 2 commit thì lúc này mô hình kho chức sẽ là:



**Mô hình đối tượng nhiều branch:**

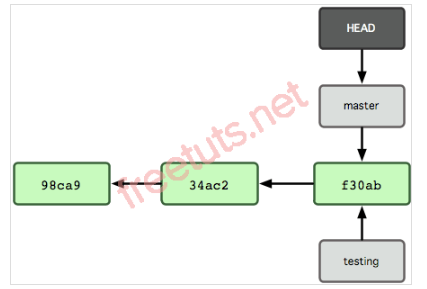
Khi tạo một nhánh(branch) thực chất là ta đang tạo một con trỏ và trỏ tới commit mới nhất tại thời điểm thực thi lệnh. Ta có tên nhánh mặc định là master branch. Khi chúng ta thực thi commit ở branch master thì mô hình ở dạng:



Khi ta tạo một nhánh mới tức là tạo một con trỏ và trỏ tới commit mới nhất. 

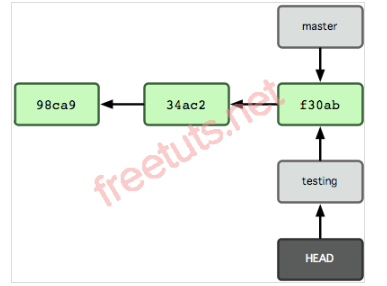
**Con trỏ HEAD**

Git có một con trỏ đặc biệt tên là HEAD, đây là con trỏ sẽ trỏ tới một branch nội bộ trong repository của bạn. Hình dưới đây là ta đang làm việc trên nhánh master vì con trỏ HEAD trỏ tới master.

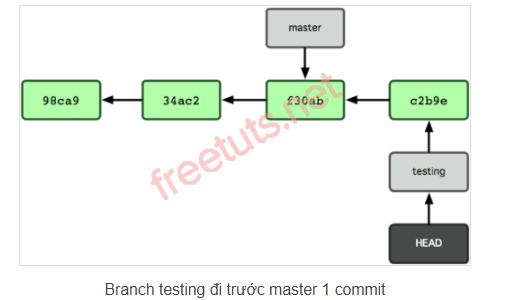


Giả sử chuyển sang branch testing với lệnh checkout

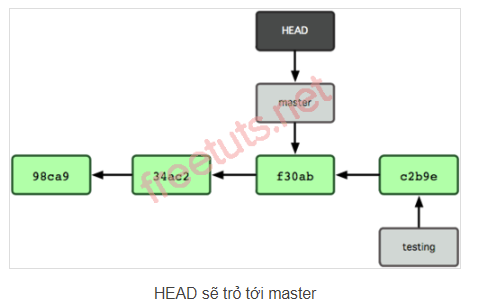
Lúc này HEAD sẽ trỏ tới branch testing



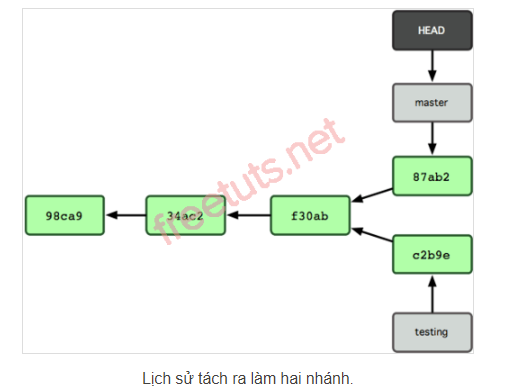
Khi ta thực hiện 1 lệnh commit tại branch testing, thì lúc này:



Bây giờ lại chuyển sang branch master:



Nếu ta thực hiện commit tại master thì lịch sử sẽ bị tách ra.



Vậy ta có thể thay đổi lịch sử dựa vào địa chỉ của các đối tượng commit.

**Lệnh git commit –amend:**

$ git commit –amend

Nếu ta thực hiện lệnh này ngay sau commit cuối cùng mà không thay đổi gì thêm thì ảnh của commit cuối cùng đó k có gì thay đổi, chỉ nội dung message thay đổi.

$ git commit -m”bo xung”

$ git add “myfile.html”

$ git commit –amend

**Loại bỏ tập tin đưa vào stage:**

khi add một file nào đó vào stage sau đó muốn loại bỏ nó thì ta sử dụng lệnh reset HEAD.

VD: tạo 2 file và sử dụng git add để đưa vào stage nhưng rồi nhận thấy nó không cần thiết, ta muốn loại bỏ nó khỏi stage.

Trước tiên ta sử dụng lệnh git status để kiểm tra trạng thái.

$ git status

$ git reset HEAD name\_file

**Lệnh git stash – dùng cho tất cả dữ liệu**

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

$ git stash

VD: giả sử ta đang code file demo.html sau đó gặp lỗi và không thể sửa, ta muốn reset nó về nội dung ban đầu.

TH1: file mới tạo, chưa đưa vào stage(chưa add)

* Trường hợp này chỉ cần xóa hết nội dung là được, bới file chưa được tổ chức vào stage.

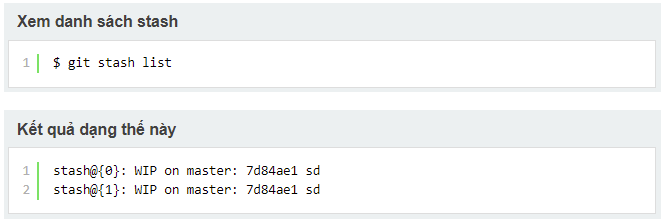
TH2: file đã được đưa vào stage hoặc tồn tại trước đó.

* Trường hợp này chạy lệnh git stash để đưa nội dung file về trạng thái ban đầu.
* Lệnh full sẽ là git stash save.

**Các tham số của git stash**

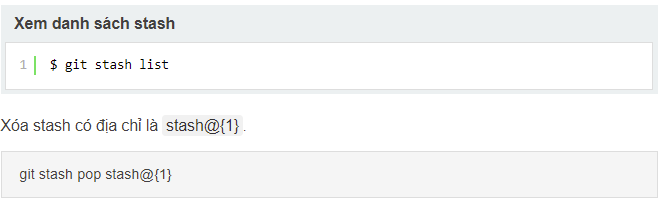
**1, git stash list:**

* Xem danh sách bao nhiêu lệnh stash đã dùng.
* Giả sử ta stash nhiều làn và muốn xem danh sách và địa chỉ của nó.

****

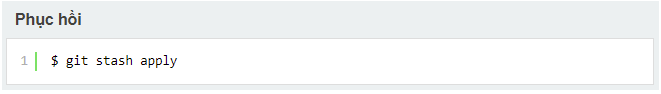
**2, git stash pop:**

* Xóa một stash khỏi lịch sử.
* Lệnh này sẽ không thực thi nếu xảy ra xung đột, lúc này cần xử lý thủ công bằng lệnh git stash drop để thay thế.
* Lệnh này cần có thêm 1 tham số đó là git stah drop stash
* Nếu không truyền tham số nó sẽ lấy stash đầu tiên stash@{0}.

****

**3, git stash apply:**

* **­**phục hồi lệnh stash, không xóa khỏi history.
* Nếu ta muốn khôi phục tất cả các lệnh stash, tức là lấy lại nội dung trước khi chạy lệnh stash thì dùng tham số apply.



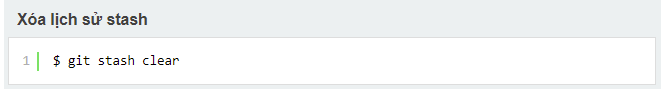
**4, git stash show:**

* Xem chi tiết của một stash, so sánh nó với dữ liệu ban đầu.
* Nếu tga muốn xem chi tiết dữ liệu lúc chạy lệnh stash so với thời điểm ban đầu thì sử dụng tham số show.

****

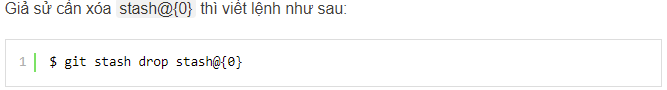
**5, git stash clear:**

* Nếu muốn xóa tất cả các stash ra khỏi history thì ta sử dụng tham số clear.



**6, git stash drop:**

* Nếu muốn xóa một stash thôi thay vì xóa tất cả thì ta sử dụng lệnh git stash drop <stash>
* Nếu ta không truyền tham số thì nó sẽ xóa tash cuối cùng.

****

**7, git stash save:**

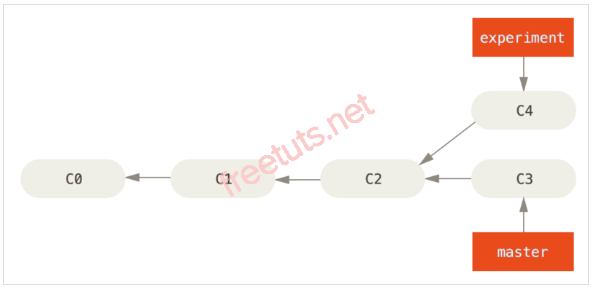
* Lệnh git stash create dùng để tạo một stash nhưng không lưu trữ vào thư mục refs mà thay thế vào đó trả về tên object. Lệnh này ít sử dụng.

**8, git stash store:**

* Nếu ta lỡ sử dụng stash save nhưng sau muốn lưu nó vào thư mực refs thì sử dụng tham số store.

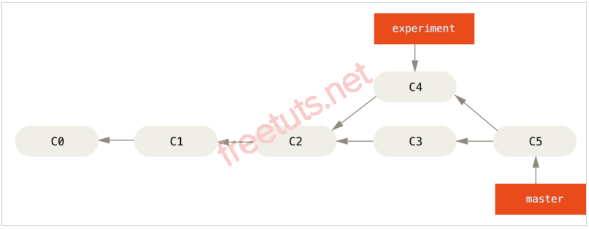
**Git rebase**

* Nó so sánh và lưu những thay đổi hai nhánh vào một file tạm sau đó thực hiện fast-forward merge.
  + Branch đang làm việc: là branch mà ta đang làm việc
  + Branch cần trộn: là branch muốn trộn vào branch hiện tại
  + Branch cần rebase: là branch muốn rebase vào branch hiện tại
* Giả sử có hai nhánh master và experiment, trong đó master là branch hiện tại.

****

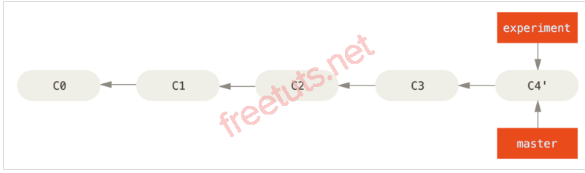
**Nếu merge:**

* Nếu sử dụng merge để trộn 2 branch với nhau thì nó sẽ thực hiện trực tiếp giữa 2 commit mới nhất (c3, c4) và commit cha chung gần nhất đó là c2, kết quả là một commit c5 nằm trên branch đang làm việc và commit c5 chính là thành quả của phép trộn.



**Nếu rebase:**

* Với rebase thì lại khác, nó sẽ tạo một phiên bản mới (tạm thời) mang tên c3’ và trộn nó vào c4, phương pháp này gọi là rebase.
* Quy trình thực hiện: nó sẽ thực hiện bằng cách tìm tới commit cha gần nhất của hai nhánh(tức là nhánh đang làm việc so với nhánh cần rebase và lưu vào một file tạm) cuối cùng sẽ thực hiện fast-forward merge.



**Làm việc với remote**

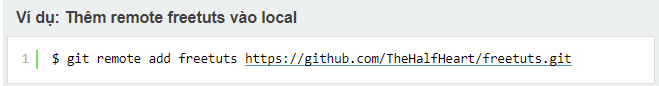
* Để làm việc với remote thì ta phải có sever và cài đặt remote trên đó. Có một số sever cung cấp dịch vụ free như github, gitlab...
* Bước đầu cần phải thêm một remote repository vào local repository, điều này nghĩa là phiên bản tại local sẽ là nơi làm việc các thành viên sẽ thực hiện các thao tác pull và push để đưa code và lấy code về local.

**1, Add remote repository**

* Để thêm một remote repo vào local repo:

git remote add short\_name URL

* + Shortname: là tên muốn đặt cho remote repo
  + url: là đường dẫn trỏ đến repo, phần đuôi là URL sẽ .git



* nếu tên remote tồn tại thì sẽ thông báo lỗi là “fatal: remote origin already exists”.

**2, Xem remote repo đã thêm**

* Để xem remote đã thêm thì ta sử dụng

git remote

hoặc

git remote -v

**3, Thay đổi linh của remote repo**

* giả sử ta gõ nhầm link của remote repo nên cần thay đổi lại link, lúc này ta sử dụng lệnh
* $ git remote set-url <shortname> <link>

**4, Xóa remote repo**

$ git remote rm remote\_name

* Nếu remote không tồn tại thì sẽ xuất hiện lỗi”fatal: no such remote: remote \_name”

**5, Đổi tên remote repo**

$ git remote rename old\_name new\_name

* Nếu remote không tồn tại: lỗi”fatal: no such remote” vậy nên để đổi tên ta nên kiểm tra lại tên cũ bằng lệnh git remote -v.
* Tên đã tồn tại: “fatal: remote [remote] already exists”.

**Git – push data lên remote repo**

* Nếu bạn là leader và khởi động cho dự án thì công việc đầu tiên là chọn công nghệ sử dụng cho dự án, sau đó tạo bộ khung và đẩy lên remote repo. Còn nếu bạn là thành viên của một dự án thì việc đầu tiên là lấy mã nguồn trên remote về local để thực hiện công việc, sau đó commit lên remote.

**1, Đẩy một branch lên remote:**

* Nếu khởi động dự án thì công việc đầu tiên là thêm remote repository, sau đó thực hiện đẩy branch tại local lên remote:

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

*Remote: là tên của remote tại bước thêm remote*

*Branch: là branch muốn đẩy lên, thường thì là branch master.*

khi thực hiện lệnh thì thì git bash sẽ yêu cầu nhập thông tin như username và password, đây là thông tin đăng nhập vào remote sever.

**2, Đẩy commit với lệnh git push**

* Sau khi git push –set-upstream thì trên remote đã có một branch và toàn bộ mã nguồn, vì vậy các thao tác tiếp theo chỉ cần sử dụng lệnh push bình thường. Mỗi lần push nó sẽ học password.

$ git push

* Nếu đang ở local branch master thì khi push lên cũng ở branch master. Nếu tạo branch mới và push lên remote thì:

$ git push –set-upstream [remote] branch\_name

Vd: tạo branch test và thực hiện đẩy nó lên remote repo

**3, Lưu thông tin đăng nhập**

**B1:**

$ git config credential.helper store

**B2:** thực hiện thao tác push, lần này git bắt đăng nhập, kể từ đó không cần phải đăng nhập nữa.

*khi ta chạy lệnh trên, git sẽ mở file store và ghi mật khẩu, vì vậy khi thực hiện thao tác đến remote thì git sẽ lấy mật khẩu ở đó thay vào.*