Git nâng cao

1 Thao tác với branch

Khi làm việc trong một dự án có nhiều người tham gia, chúng ta thường sẽ tạo nhiều nhánh để dễ dàng quản lý code hơn. Ví dụ: Người A code tính năng login ở nhánh `feature/Login` và người B code tính năng register ở nhánh `feature/Register`, 2 người không ảnh hưởng đến nhau khi code. Sau đó, hai người sẽ tiến hành gộp code của họ lại vào `master` khi họ code xong.

1.1 Tạo branch mới

Câu lệnh này sẽ tạo một nhánh mới dựa trên nhánh hiện tại

```
git branch TenNhanhMoi
```

2 câu lệnh dưới sẽ tạo một nhánh mới dựa trên nhánh hiện tại và chuyển sang nhánh mới luôn.

```
git checkout -b TenNhanhMoi
```

```
git switch -c TenNhanhMoi
```

`git switch` mới ra ở phiên bản Git 2.23. Các bạn dùng cái nào cũng được

1.2 List tất cả branch

Câu lệnh này sẽ cho bạn biết local repo của bạn hiện tại đang có bao nhiêu nhánh. Những nhánh mà câu lệnh `git branch` show ra là những nhánh mà bạn đã từng làm việc trên local. Có những nhánh đã có ở local mà bạn chưa làm việc (những nhánh này thường là do bạn pull hoặc fetch code về sẽ có) thì nó sẽ không hiện ở đây.

```
git branch
```

hoặc `git branch --list` đều như nhau

Nếu muốn thấy các remote branch

```
git branch -r
```

Nếu muốn thấy tất cả local và remote branch

```
git branch -a
```

1.3. Đổi tên branch

Nếu bạn tạo tên nhánh bị sai, bạn cũng có thể sửa lại tên nhánh hiện tại bằng cách

```
git branch -m TenMoi
```

Cách trên chỉ áp dụng khi nhánh hiện tại của bạn là nhánh sai và bạn muốn đổi tên nhánh hiện tại. Trong trường hợp bạn muốn đổi tên nhánh nào đó không nhất thiết là nhánh hiện tại thì bạn phải ghi rõ tên nhánh cũ và nhánh mới như dưới đây

```
git branch -m TenNhanhCu TenNhanhMoi
```

Các bạn đổi tên branch thì chỉ đổi được ở dưới local repo, nếu branch đó đã xuất hiện ở trên remote repo thì khi bạn push nó sẽ tạo một branch mới trên remote repo

1.4 Chuyển branch

Nếu bạn đơn giản chỉ muốn chuyển sang một nhánh trên local repo, 2 câu lệnh dưới đây đều được

```
git switch TenNhanh
```

```
git checkout TenNhanh
```

> **Lưu ý**: Nhánh phải có trên local repo thì mới chuyển được nhé, nếu nhánh đó có trên remote repo mà local chưa có thì bạn phải tiến hành cập nhật repo của bạn lại bằng câu lệnh `git fetch`.

1.5 Push một branch

Nếu local branch của bạn không tồn tại trên remote, chạy câu lệnh sau

```
git push -u origin localBranch
```

Hoặc câu lệnh này, nó sẽ giúp các bạn không cần gõ chính xác tên local branch

```
git push -u origin HEAD
```

> Head là tham chiếu đến đầu danh sách branch hiện tại

1.6 Xóa branch

Xóa branch ở local

```
git branch -D localBranchName
```

Xóa branch ở remote

```
git push origin --delete remoteBranchName
```

Hoặc bạn có thể dùng cú pháp rút gọn

```
git push origin :remoteBranchName
```

Nếu bạn gặp lỗi dưới đây thì có thể ai đó đã xóa branch

```
error: unable to delete 'branchName': remote ref does not exist
error: failed to push some refs to 'git@repository_name'
```

Khi ai đó đã xóa một branch trên remote nhưng khi bạn gõ `git branch -r` vẫn show ra origin branch đó thì bạn cần thực hiện đồng bộ hóa bằng câu lệnh dưới đây.

```
git fetch -p
```

`-p` nghĩa là **"prune"**. Sau khi fetch, những branch không còn trên remote cũng sẽ xóa khỏi local repo của bạn.

1.7 Push code mà không cần origin

Nếu bạn để ý thì khi ở nhánh `master`, bạn push code lên orgin thì bạn chỉ cần `git push` là xong, không cần `git push origin master`. Nhưng nếu bạn ở nhánh khác ví dụ nhánh `feature` thì khi bạn push code lên `origin feature` thì phải `git push orgin feature` khá dài dòng và phải nhớ tên nhánh mệt mỏi.

Cách cải thiện khá đơn giản, bạn chỉ cần thêm option `-u` câu lệnh push là được, chỉ cần làm 1 lần, lần sau không cần làm với nhánh đó nữa.

```
git push -u origin feature
```

`-u` là short-hand của `--set-upstream`. Nó cho git biết rằng hãy tự kết nối nhánh `local feature` với `origin feature`

Làm 1 lần thôi, lần sau chỉ cần ở nhánh `feature` và nhấn `git push` là code tự lên.

> Tip: Nếu muốn xem những local branch đã kết nối với remote branch thì chỉ cần gõ `cat .git/config`

git merge

Gộp các commit lại của 2 nhánh với nhau dựa trên thời gian các commit, sau khi gộp thì sẽ tạo ra một commit gộp sau cùng.

```
git merge branchTarget
```

Chúng ta có 3 nhánh:

- main
- Alpha: checkout từ nhánh main và thêm 2 commit D,E
- Beta: checkout từ nhánh main và thêm 2 commit là F,G

Thứ tự các commit theo chiều ngang được sắp xếp theo thời gian

```
   (main)
A--B-C---D--E (Alpha)
      \
       -F-----G (Beta)
```

Nếu ở nhánh Beta chúng ta muốn có những commit của nhánh Alpha (hay còn gọi là có code) thì ta sẽ tiến hành merge Alpha vào Beta nên ta thực hiện lệnh

```
git checkout Beta
git merge Alpha
```

Sơ đồ commit sẽ như thế này, các commit của Alpha sẽ xuất hiện trong Beta và được sắp xếp theo thứ tự thời gian. Ta sẽ thấy sự xuất hiện của một commit merge nữa phía sau cùng, commit merge này chứa tất cả những sự thay đổi của nhánh Beta sau khi merge nhánh Alpha vào.

```
   (main)
A--B-C---D--E (Alpha)
      \       \
       -F----G-M (Beta)
```

Bây giờ nhánh Beta đã có đầy đủ code của nhánh Alpha.

Nếu trong quá trình merge có bị conflict thì thao tác theo thứ tự

1. Tìm file conflict trong tab source và fix nó.
2. Sau khi fix hết các file bị conflict rồi thì dùng `git add .` hoặc add từng file
3. Tiến hành thêm commit cho những file vừa fix conflict bằng câu lệnh `git commit --no-edit`, nếu muốn edit commit thì `git merge --continue`, còn nếu muốn tự viết commit thì cứ `git commit -m 'thông điệp'` như thường
4. push hết lên với câu lệnh `git push`

Nếu merge xong rồi mới nhận ra mình không cần merge nữa thì có thể dùng `git reset --hard <commit-của bạn-trước-khi-merge>`. Cách này áp dụng cho cả đã push code hay chưa push đều được.

> Tips: `git pull` là sự kết hợp giữa `git fetch` và `git merge`. Trong thực tế thì mình dùng `git pull` chứ ít khi dùng `git merge` vì mình luôn muốn fetch data mới nhất từ origin về rồi mới merge.
>
> Bạn cũng có thể tạo merge request (một số chỗ gọi là pull request) bằng git server như Github thay vì dùng câu lệnh. Những merge request này được tạo ra để phục vụ việc team review và approve code.

git rebase

Lấy tất cả các commit của `branchTarget` làm lại base (gọi là re-base) cho mình, các commit của mình mà khác so với `branchTarget` sẽ bị thay đổi hash và thêm vào sau cùng.

```
git rebase branchTarget
```

```
   (main)
A-B-C-----D (feature)
     \
      --E----F (feature/Login)
```

```
git checkout feature/Login
git rebase feature
```

```
   (main)(feature)
A-B-C-----D--E'----F' (feature/Login)
```

Nếu những commit của branch `feature/Login` và `feature` đã xuất hiện trên remote thì có thể sau khi rebase `feature` vào `feature/Login` thì bạn sẽ gặp một hiện tượng đó là git sẽ báo bạn kiểu như là **"Current branch feature/Login is 2 commit behind, 3 commit ahead origin/feature/Login"**.

Giải thích:

- `feature/Login` ở local chỉ giống với `origin/feature/Login` đoạn `A-B-C` thôi => `feature/Login` đứng sau 2 commit của `origin/feature/Login` là `E-F`. Ngoài ra `feature/Login` có 3 commit trước so với `origin/feature/Login` là `D--E'----F'`

- Git sẽ yêu cầu bạn pull code từ origin `feature/Login` về trước khi push lên. Mọi thao tác làm thay đổi lịch sử Git trên origin như xóa commit, đổi mã hash commit thì Git đều không cho push mà yêu cầu pull về trước. Nếu pull code về thì ta lại có commit `E-F` khá thừa thải vì đã có `E'-F'` rồi, để tránh hiện tượng này thì bạn có thể push force `git push -f` luôn.

Git rebase sẽ làm Git Graph của bạn trông đẹp, dễ nhìn, đỡ phải phân tách nhánh rồi gộp như bên rebase. Nhưng cái gì cũng có cái giá của nó. Bạn phải sử dụng `git push -f` sau cùng, và câu lệnh này **rất nguy hiểm** nếu bạn đang thực hiện trên một branch nhiều người dùng.

Ở trên mình đang thực hiện trên branch `feature/Login`, mình rebase `feature` vào. Nếu mình làm người lại là rebase `feature/Login` vào `feature` thì rất nguy hiểm.

Hãy cẩn thận với git rebase

Không nên rebase những nhánh khác vào nhánh tổng (nhánh có nhiều người làm gốc để tạo nhánh con, nhánh được chia sẻ bởi nhiều người). Ví dụ bạn đang ở nhánh `master`, bạn muốn có code của nhánh `feature/Login` và bạn thực hiện

```
git checkout master
git rebase feature/Login
```

Lúc này thì base lịch sử commit của nhánh `master` sẽ bị **thay thế**. Nó sẽ lấy base là nhánh `feature/Login` và đặt những commit mới của `master` ra sau cùng, điều này không tốt chút nào với một nhánh tổng. Việc **thay thế** base lịch sử commit của `master` sẽ gây khó khăn trong việc revert tính năng Login, bạn phải `git revert` tất cả commit của tính năng Login từng cái một.

Khi rebase thì phải push force, mà push force rất nguy hiểm, nhất là trên nhánh tổng. Vì nó sẽ overide code của người khác đã commit trên nhánh tổng đó.

Nhưng nó sẽ là an toàn nếu bạn đang ở nhánh `feature/Login` (đây là nhánh chỉ có mỗi 1 mình bạn đảm nhận code) và bạn muốn có code mới nhất của nhánh `master`, bạn thực hiện

```
git checkout feature/Login
git rebase master
```

Ưu điểm rebase:

- Tạo một đường thẳng commit sạch sẽ, dễ nhìn
- Nếu làm trên nhánh cá nhân thì sẽ phù hợp

Khuyết điểm

- Nếu rebase trên branch chung thì sẽ rất nguy hiểm vì có thể làm mất code người khác

- Dùng push force có thể làm mất code nếu không để ý
- Muốn undo rebase thì không phải chuyện đơn giản. Nếu bạn rebase trên máy bạn thì bạn có thể undo bằng cách sử dụng `git reflog` và `git reset` nhưng nếu bạn clear local repo hoặc bạn phải thực hiện undo rebase trên máy khác thì bạn phải dùng `git revert` để hoàn tác từng commit.

Cách an toàn nhất trong mọi trường hợp là cứ dùng `git merge` đi, mặc dù merge nó sẽ tạo ra thêm 1 commit merge và lịch sử commit của branch đó sẽ bị thêm khá nhiều commit xen kẻ của người khác làm cho bạn rối nhưng ít ra nó sẽ không làm cho bạn bị mất code.

git fetch

Phổ biến nhất là chỉ cần gõ `git fetch`, nó sẽ load về local tất cả những branch, commit hiện có.

git remote

Lệnh `git remote` cho phép tạo, xem, xóa kết nối với các repo.

Để liệt kê các liên kết

```
git remote
```

> Khi lấy một remote reop bằng câu lệnh `git clone`, mặc định repo tải về có liên kết với tên `origin` chứa địa chỉ tham chiếu đến remote repo nó tải về

Hiển thị thông tin chi tiết hơn, có thêm đường dẫn đến remote Repo

```
git remote -v
```

Tạo một liên kết

```
git remote add TenRemote url
```

Xóa một địa chỉ remote

```
git remote rm TenRemote
```

Đổi tên địa chỉ remote

```
git remote rename TenCu TenMoi
```

Xem thông tin về Remote

```
git remote show origin
```

Các kỹ thuật hoàn tác thay đổi với Git

Hoàn tác những file local change về trạng thái ban đầu

Dùng với terminal

1. Chạy `git status` bạn sẽ thấy những file bị change (hoặc cũng có thể coi trong VS Code)
2. Chạy câu lệnh dưới, thay thế `filename.html` thành file path của bạn

   ```
   git checkout filename.html
   ```

   Hoặc chúng ta có lệnh `git restore` chuyên để làm việc với các file trong working tree. Nếu không truyền gì thì mặc định là `-W` hoặc `--worktree`

   ```
   git restore filename.html
   ```

   ```
   git restore filename.html -W
   ```

   ```
   git restore filename.html --worktree
   ```

Dùng với VS Code

Vào Source Control, hover vào file và click vào icon "Discard Change" là được.

Hoàn tác những file ở khu vực Staging về khu vực làm việc

Ngược lại với `git add` thì `git reset` sẽ đưa một hoặc nhiều file ở khu vực **Staging** trở về khu vực code.

Dùng Terminal

Khôi phục 1 file từ staging về khu vực code

```
git reset index.html
```

Khôi phục nhiều file

```
git reset index.html app.js
```

Khôi phục tất cả các file.

```
git reset .
```

Tương tự với `git reset` ta có `git restore -S` hoặc `git restore --staged`

Dùng VS code

Vào Source Control, hover vào file và click vào icon dấu "-" "Unstage changes" là được.

Hoàn tác những file nào đó từ những commit trong quá khứ

Bạn chỉ cần nhớ những công việc liên quan đến hoàn tác file thì `git restore` luôn rất mạnh. Để hoàn tác file nào đó thuộc commit nào đó thì chỉ cần tìm mã hash commit đó rồi chạy command

```
git restore --source=3c13cc8 TenFile
```

Câu lệnh trên sẽ đưa file đó về trạng thái Changes, một trạng thái an toàn nhất.

Áp dụng với VS Code

Với UI VS code thì đơn giản hơn nhiều, bạn vào "Source Control", bạn có thể tìm file đó trong khu vực COMMITS hoặc tìm lịch sử chỉnh sửa file đó trong FILE HISTORY. Tại đây bạn có thể xem được các file đó đã thay đổi như thế nào trong quá khứ. Sau đó click chuột phải vào file đó chọn "Restore (Checkout)" nó sẽ đưa file bạn về trạng thái Staging

Hoàn tác commit bằng reset (đã push hay chưa đều được)

Nếu bạn có những local commit mà bạn không thích, chúng chưa được push thì bạn có thể reset để commit lại tốt hơn.

Dùng terminal

1. Mỗi commit có một mã hash khác nhau. Bạn cần tìm mã hash của cái commit mà bạn thấy ổn cuối cùng (cái mà bạn muốn hoàn tác trở về, Ví dụ bạn có commit là `A`, `B`. Bạn muốn hoàn tác commit `A` thì bạn phải trở về ngay tại commit `B`). Ở đây có một số cách tìm như
    - Trong commit history của Github
    - Trong terminal bằng cách gõ câu lệnh `git log --oneline`
    - Trong VS code
2. Khi bạn biết cái commit ổn cuối cùng rồi thì chỉ cần chạy câu lệnh dưới (thay thế `2f5451f` bằng commit hash của bạn)

    Bây giờ sẽ có một số option cho bạn, những option dưới đây sẽ xóa các commit và đưa những file thay đổi về những trạng thái khác nhau, mình sẽ sắp xếp theo thứ tự độ an toàn giảm dần

- Đưa những file thay đổi về trạng thái **Changes**. **Đây là option an toàn nhất**

```
git reset 2f5451f
```

hoặc thêm `--mixed`. Default `git reset` không thêm là mode `--mixed`

```
git reset --mixed 2f5451f
```

- Đưa những file thay đổi về trạng thái **Staging**

```
git reset --soft 2f5451f
```

- Đưa những thay đổi về tình trạng như chưa có gì thay đổi (nguy hiểm nhất vì nó sẽ làm mất code của bạn).

```
git reset --hard 2f5451f
```

- Giống `--hard` ngoại trừ việc nó sẽ giữ lại những thay đổi của những file (đã từng commit trong quá khứ) mà bạn đang edit dỡ chưa add tại **Changes**.

Ví dụ mình đang edit lại file `index.html` đã tồn tại từ trước, và hiện tại nó đang xuất hiện ở khu vực **Changes**, mình thực hiện reset với `--merge` thì file này nó vẫn được giữ nguyên những thay đổi ở khu vực **Changes**. `--hard` thì nó xóa luôn những thay đổi đó nên nếu không cẩn thận có thể làm mất code.

```
git reset --merge 2f5451f
```

3. Dùng `git push -f` để push những thay đổi

Dùng VS code (Phải cài Git Lens)

Ở đây thì có 1 vài cách

- Mở Source Control lên, Vào khu vực COMMITS, hover chuộc vào commit muốn hoàn tác và click vào icon Undo Commit là được. Nó sẽ đưa những thay đổi về khu vực Staging. **Cách này chỉ có tác dụng với commit gần nhất**
- Mở terminal Git Bash trong terminal của VS Code. Gõ `git log --oneline` rồi Ctrl + Click vào mã hash commit mà muốn trở về. Tiếp theo chọn "Reset branch to commit". Sau đó chọn option reset tương tự như trên terminal

Lưu ý khi dùng `git reset`

Nếu bạn reset về những commit thuộc branch khác thì bạn có thể bị mất những commit trên branch của bạn. Vậy nên hãy cẩn thận, áp dụng **Git Graph** để cho chắc ăn.

Hoàn tác commit bằng revert

1. Chạy `git status` và chắc rằng working tree của bạn clean
2. Mỗi commit có một mã hash, tìm mã hash đó
3. Khi bạn tìm thấy mã hash rồi thì chạy câu lệnh dưới (thay thế `2f5451f` bằng mã hash của bạn)

   ```
   git revert 2f5451f --no-edit
   ```

   Note: `--no-edit` option ngăn git hỏi bạn edit một commit message. Nếu bạn không thêm option đó, nó sẽ mở 1 file bằng Code Editor của bạn để cho bạn edit lại cái message mặc định khi revert, bạn edit lại message rồi close file đi là được. (Nếu nó là VIM editor thì để thoát khỏi VIM, ấn `:` trong command mode, sau đó nhấn `q` để thoát, và cuối cùng nhấn **Return** (Mac) hoặc **Enter** (Windows))
4. Thao tác này sẽ tạo một commit mới đối lập với commit hiện có, hoàn tác các file về trạng thái trước đó như thể nó chưa bao giờ được thay đổi.
5. Nếu bạn làm việc với một remote repo, bạn cần push những thay đổi đó bằng `git push`

Lưu ý dùng `git revert -m 1 <merge-commit>` để revert commit merge.

Mỗi commit đều có commit cha (commit trước nó). Những commit thường thì sẽ có 1 cha, nhưng merge commit thì có nhiều hơn 1 cha. `-m 1` là chọn commit cha đầu tiên (commit cha đầu tiên lúc nào cũng là commit trên branch của hiện tại của bạn, hay còn gọi là branch được merge). Bạn có thể xem những commit cha bằng cách gõ `git log` và những commit cha sẽ được liệt kê trong mục `Merge` (hoặc xem trong **Git Graph**)

Gộp commit đã push (hoặc chưa push đều được)

Để gộp các commit đã push thì chúng ta dùng rebase interactive với squash. Với squash thì nó sẽ neo commit của bạn vào commit phía trước.

Ví dụ mình muốn gộp 3 commit gần nhất lại thành 1

Đầu tiên mình sẽ kiểm tra `git log --oneline`

```
8c6fb49 (HEAD -> bugfix, origin/bugfix) 1
44fa9a0 (origin/main, origin/HEAD, main) tm3
9bd70cc (origin/t3, t3) t3
16bc7f3 update main
6804903 (origin/t2, t2) t2 app.js
3e233a4 t2
```

Tiếp theo mình sẽ chạy câu lệnh

```
git rebase -i HEAD~3
```

Lúc này nó sẽ xuất hiện 1 file để bạn edit như thế này. Có thể bạn sẽ thấy thứ tự 3 commit gần đây hơi ngược so với khi dùng `git log`, nhưng không sau, kệ nó đi. Bạn chỉ cần biết thứ tự của nó theo chiều nào là được rồi.

Bây giờ mình sẽ gộp commit có message là `1` và `tmp3` vào t3. Lưu ý: Bạn không thể gộp `t3` và `tm3` vào `1` được bởi vì `1` đứng phía sau `t3` và `tm3`. Tức là chỉ được gộp những commit phía sau vào commit phía trước thôi.

```
pick 9bd70cc t3
pick 44fa9a0 tm3
pick 8c6fb49 1

# Rebase 16bc7f3..8c6fb49 onto 16bc7f3 (3 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup [-C | -c] <commit> = like "squash" but keep only the previous
#                    commit's log message, unless -C is used, in which case
#                    keep only this commit's message; -c is same as -C but
#                    opens the editor
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
# .       create a merge commit using the original merge commit's
# .       message (or the oneline, if no original merge commit was
# .       specified); use -c <commit> to reword the commit message
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
```

Mình sẽ edit lại như thế này và lưu lại và tắt đi, sửa `pick` thành `s` (hoặc `squash` cũng được)

```
pick 9bd70cc t3
s 44fa9a0 tm3
s 8c6fb49 1

# Rebase 16bc7f3..8c6fb49 onto 16bc7f3 (3 commands)
```

```
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup [-C | -c] <commit> = like "squash" but keep only the previous
#                    commit's log message, unless -C is used, in which case
#                    keep only this commit's message; -c is same as -C but
#                    opens the editor
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
# .       create a merge commit using the original merge commit's
# .       message (or the oneline, if no original merge commit was
# .       specified); use -c <commit> to reword the commit message
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
```

Lúc này nó sẽ xuất hiện 1 file nữa để mình edit

```
# This is a combination of 3 commits.
# This is the 1st commit message:

t3

# This is the commit message #2:

tm3

# This is the commit message #3:

1

2

3

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Sun Jul 24 17:30:29 2022 +0700
#
# interactive rebase in progress; onto 16bc7f3
```

```
# Last commands done (3 commands done):
#    squash 44fa9a0 tm3
#    squash 8c6fb49 1
# No commands remaining.
# You are currently rebasing branch 'bugfix' on '16bc7f3'.
#
# Changes to be committed:
#   modified:   readme.md
#   new file:   t3.md
#   new file:   tm3.md
#
```

Mình sẽ edit lại cái commit message `t3` thành `gộp 3 commit lại` và lưu, đóng file.

```
# This is a combination of 3 commits.
# This is the 1st commit message:

gộp 3 commit lại

# This is the commit message #2:

tm3

# This is the commit message #3:

1

2

3

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Sun Jul 24 17:30:29 2022 +0700
#
# interactive rebase in progress; onto 16bc7f3
# Last commands done (3 commands done):
#    squash 44fa9a0 tm3
#    squash 8c6fb49 1
# No commands remaining.
# You are currently rebasing branch 'bugfix' on '16bc7f3'.
#
# Changes to be committed:
#   modified:   readme.md
#   new file:   t3.md
#   new file:   tm3.md
#
```

Bây giờ chỉ cần `git push -f` là xong.

Và tất nhiên sau khi rebase thì hash của commit sẽ bị thay đổi.

```
d88d8e9 (HEAD -> bugfix, origin/bugfix) gộp 3 commit lại
16bc7f3 update main
6804903 (origin/t2, t2) t2 app.js
3e233a4 t2
```

Sửa message một commit đã được push

Chúng ta cũng dùng rebase interactive nhưng thay vì `squash` thì chúng ta chọn `reword` hoặc `r`

> Sửa một message của commit sẽ làm thay đổi hash của commit đó và tất cả những commit phía sau nó. Sau khi sửa thì phải push force

Thêm thay đổi vào commit cuối cùng

Thêm file vào staging

```
git add .
```

Chạy câu lệnh

```
git commit --amend
```

Nó sẽ xuất hiện 1 file để bạn có thể edit lại commit message mới nhất, sau đó lưu và tắt file đó đi.

Lệnh `--amend` sẽ làm thay đổi mã hash của commit mới nhất, vì thế sau đó bạn phải dùng `git push -f` để push lên.

Sự khác nhau giữa `git reset`, `git restore`, `git revert`

- git reset chuyên reset con trỏ HEAD về một trạng thái nhất định
- git restore chuyên khôi phục những file trong working tree
- git revert chuyên khôi phục những commit những sẽ tạo ra một commit revert.

3 thằng này được quy định là thế nhưng trong số chúng có những tính năng mà thằng khác vẫn làm được. Vậy nên dùng quen thằng nào thì dùng thằng đó, không cần quá cứng nhắc.

Git stash

Dùng khi muốn lưu lại các thay đổi chưa commit, rất hữu dụng khi đang làm dỡ branch hiện tại mà lại muốn chuyển sang branch khác.

Để lưu toàn bộ thay đổi thì dùng

```
git stash
```

Để show toàn bộ thay đổi

```
git stash list
```

Kết quả sẽ như thế này

```
stash@{0}: WIP on Register: 8373b84 Hoàn thành register page
stash@{1}: WIP on Register: 812sd81 Tạo file a.md
```

Để show chi tiết toàn bộ thay đổi

```
git stash list -p
```

Để xem chi tiết stash vị trí thứ 0

```
git stash show stash@{0} -p
```

Để apply lại thay đổi stash 0

```
git stash apply stash@{0}
```

Để xóa một stash

```
git stash drop stash@{1}
```

Để xóa toàn bộ stash

```
git stash clear
```

Khi nào Git sẽ bắt chúng ta pull code về trước khi push lên

Khi origin branch có những commit mà local branch chúng ta không có. Điều này sảy ra khi

- Mã hash commit bị thay đổi

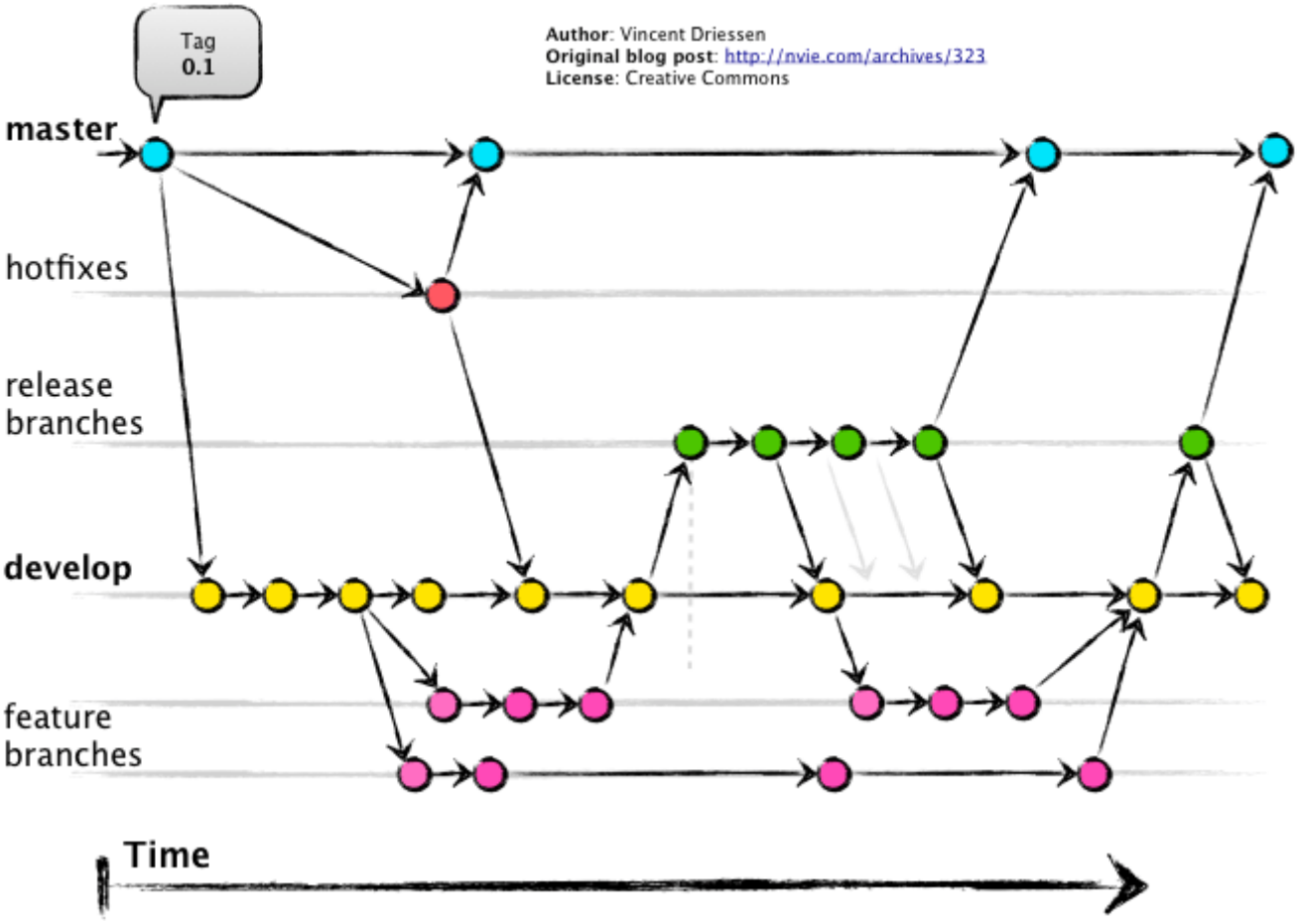
- Chúng ta đã xóa những commit trên local branch

Nếu chúng ta tin rằng local branch của chúng ta là đúng, origin branch hãy cập nhật đúng theo local branch thì cứ dùng `git push -f` (nhưng phải để ý branch, không nên force trên branch tổng)

Có một option an toàn hơn `--force` là `--force-with-lease`. `--force-with-lease` chỉ cho phép overide những commit của bạn thôi, nếu có những commit của người khác thì nó sẽ không cho push.

Git Flow

Git flow là quy trình làm việc với Git, cách thiết kế các nhánh khác nhau để merge lại sao cho hạn chế những lỗi hay conflict không đáng có.



- **master**: Là nhánh default, đây là nơi chứa source chính thứ khi release sản phẩm ra cộng đồng.
- **hotfixes**: Nhánh này được checkout từ **master**, để sửa nhanh những lỗi của nhánh **master**. Sau khi sửa xong thì sẽ merge lại vào **master**. Bởi vì chúng ta không nên thay đổi code trực tiếp trên nhánh **master**, rất nguy hiểm đến sản phẩm đang chạy production.
- **release**: Nhánh này là nơi các tính năng đã hoàn thành và đang trong quá trình test, sau khi test xong thì sẽ được merge vào **master**
- **develop**: Được checkout từ **master**, nhánh này là nơi chứa tất cả lịch sử commit của dự án. Khi dev đã hoàn thành xong một chức năng trên nhánh feature thì sẽ được merge vào nhánh **develop**, Leader sẽ review và sẽ tiến hành merge **develop** vào release để cho tester test tính năng đó.

- **feature**: Được checkout từ **develop**, **feature** ở đây có thể là **feature/login**, **feature/register**, **feature/cart**. Sau khi hoàn thành một feature sẽ tiến hành merge vào **develop**.

Trên đây là một git flow cơ bản, tùy dự án sẽ có một số branch khác phục vụ những môi trường khác nhau như **release-qc** cho tester test, **release-beta** cho người dùng test.

Nhưng nguyên tắc để dùng Git an toàn

1. Merge branch nhỏ vào branch tổng thành **hãy tạo merge request**. Ví dụ merge **feature/Login** vào **develop**
    - Giúp cho Leader và các thành viên trong team có thể review code bạn và comment những đoạn code không hợp lý
    - Lưu lại lịch sử code của bạn (bao gồm cả commit), phục vụ việc kiểm tra sau này nếu dự án có vấn đề
    - Là nơi đánh giá code của các thành viên trong team, mọi người có thể học hỏi cách code lẫn nhau. Sau này nếu công ty có đánh giá nhân viên thì có thể xem lại cái merge request này để xem chất lượng code của nhân viên đó.
2. **Xử lý conflict code**
    - Để hạn chế conflict thì hãy chia nhỏ file, chia nhỏ code ra.
    - Thường xuyên cập nhật local branch để mới nhất so với remote branch bằng cách `git pull`. Ví dụ mình code ở branch **feature/Login**, để hạn chế việc xử lý conflict khi mình tiến hành merge **feature/Login** vào **develop** thì trong quá trình mình code, mình thường hay `git pull origin develop` để code của mình luôn mới.
    - Khi xuất hiện conflict thì hãy lựa chọn các option một cách cẩn thận, nên xem xét kêu gọi những người liên quan đến conflict để cùng giải quyết.