**Sử dụng git stash hiệu quả**

<https://kipalog.com/posts/Su-dung-git-stash-hieu-qua>

Tôi xin tổng hợp các cách dùng git stash tôi hay sử dụng

## Lưu lại thay đổi

Git stash được sử dụng khi muốn lưu lại các thay đổi **chưa commit**, thường rất hữu dụng khi bạn muốn đổi sang 1 branch khác mà lại đang làm dở ở branch hiện tại.

Muốn lưu toàn bộ nội dung công việc đang làm dở, bạn có thể sử dụng git stash như sau

$ git stash save *# or just "git stash"*

Khi này branch đã trở nên "sạch sẽ" và git status sẽ cho thấy bạn có thể chuyển sang branch tuỳ thích. Bạn có thể git stash **bao nhiêu lần tuỳ thích** và mỗi lần đó git sẽ lưu toàn bộ lần thay đổi đó như 1 phần tử trong 1 stack.

## Lấy lại thay đổi

Sau khi đã git stash 1 hoặc vài lần, bạn có thể xem lại danh sách các lần lưu thay đổi bằng câu lệnh

$ git stash list

stash@{0}: WIP on <branch-name>: <lastest commit>

stash@{1}: WIP on <branch-name>: <lastest commit>

stash@{2}: WIP on <branch-name>: <lastest commit>

Nếu muốn xem cả nội dung của từng thay đổi thì thêm option -p

$ git stash list -p

hoặc xem nội dung cụ thể hơn nữa của lần thay đổi thứ 1:

$ git stash show stash@{1}

Khi muốn apply lại thay đổi từ stash lần 1 bạn có thể

$ git stash apply stash@{1}

## Xoá các thay đổi không cần thiết

Đôi khi bạn muốn lấy lại thay đổi và xoá nội dung thay đổi lưu trong stack đi, khi đó bạn có thể

$ git stash apply stash@{1}

$ git stash drop stash@{1}

hoặc đơn giản hơn là

$ git stash pop stash@{1}

Thậm chí nếu muốn xoá *toàn bộ stack* thì có thể dùng clear

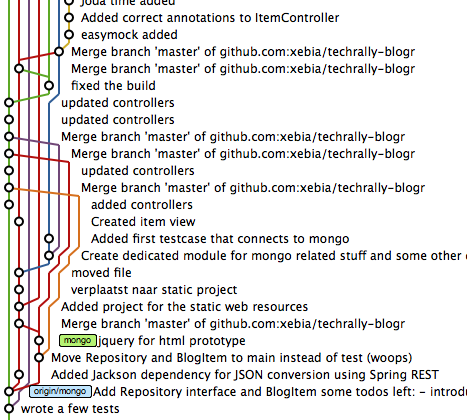
$ git stash clear

Bạn vốn làm việc một mình một cõi, “thầu nguyên con” dự án. Dù phải code sấp mặt nhưng cuộc đời vẫn thật êm xuôi: tốc độ thần thánh 500 dòng/giờ, commit code mới pặc pặc vào master, cuối ngày chỉ việc git push, tắt máy, rồi dắt gấu đi nhậu (hoặc lội suối băng đèo về nhà thay tã cho con).

Ngờ đâu sếp (hoặc co-founder) xót thương bạn vất vả, bèn tuyển ngay 2 em đào nhí vào trợ giúp. Và mọi chuyện bắt đầu phức tạp từ đây: code trong team bị chồng chéo và xung đột liên tục, branch master đang thẳng thớm đẹp xinh bỗng phân nhánh như điên, tốc độ code của bạn giảm trong khi tốc độ chửi thề ngày một tăng nhanh. Bạn không còn thời giờ dắt gấu đi nhậu, cũng không thể lội kịp về nhà thay tã cho con nữa. Cuộc đời bế tắc.

# REBASE –MERGE

<https://kipalog.com/posts/Cach-phan-nhanh-va-chia-viec-trong-nhom-voi-Git>

[[](https://res.cloudinary.com/duqeezi8j/image/upload/v1524414315/git-merge-hell-smaller_lxhizg.png)](https://res.cloudinary.com/duqeezi8j/image/upload/v1524414315/git-merge-hell-smaller_lxhizg.png)

Nhìn hình này bạn có thấy quen không (nguồn: Xebia.com)?

Thôi đừng vội lật bàn quýnh cả sếp lẫn đào. Hãy thử làm theo một số quy ước sau đây, đảm bảo team dev nho nhỏ xinh xinh của bạn sẽ không còn “giẫm chân” nhau nữa. Công việc xuôi buồm mát mái, tình đồng nghiệp sẽ lại thương mến thương nè.

***Ngoài lề:*** Bạn đừng bỏ qua bài viết [*Tăng hiệu suất sử dụng git với git alias*](https://ehkoo.com/bai-viet/tang-hieu-suat-su-dung-git-voi-git-alias)

## Vậy, vứn đề chính ở đây là gì?

Không có gì nghiêm trọng cả, chỉ là chúng ta chưa có một quy ước phân chia nhánh (branch) hợp lý thôi. So với các chương trình quản lý phiên bản khác như SVN, khả năng phân nhánh của Git phải nói là siêu nhẹ và cực kì dễ dàng. Do đó, bạn có thể chia dự án thành 2 nhánh chính:

* master
* dev

Nhánh master sẽ là nơi chứa phần code **ổn định nhất**, sẵn sàng để triển khai bất cứ lúc nào. Trong khi đó, nhánh dev ban đầu được tách ra từ master, và sẽ chứa phần code **mới nhất** được phát triển.

**Nói nhỏ:** Một số tài liệu sẽ đặt tên nhánh tách ra là develop. Tùy bạn chọn tên gì cũng được, nhưng theo Ehkoo thì gõ dev nhanh hơn gõ develop :p

**Nhắc bài chút xíu**  
Để tạo nhánh mới trong Git, bạn dùng lệnh:

git checkout -b <tên nhánh mới> [nhánh gốc]

Chẳng hạn, để tạo nhánh dev từ master, bạn gõ git checkout -b dev master. Nếu không cung cấp tham số [nhánh gốc], nhánh mới tạo sẽ dựa trên nhánh hiện tại bạn đang ở. Để xem nhánh hiện tại là nhánh nào, bạn có thể dùng lệnh git branch.

$ git branch

auth

\* auth-session

dev

graphql

master

Trong ví dụ trên thì nhánh hiện tại chính là auth-session.

## Phân chia công việc

Mỗi khi phát triển tính năng mới, bạn sẽ tạo một nhánh từ dev.

git checkout -b login dev

**Đặt tên nhánh là gì đây?**  
Tùy bạn thôi. Một số tài liệu sẽ khuyến khích bạn dùng tiền tố feature/<tên tính năng> để dễ phân biệt. Nhưng theo kinh nghiệm của Ehkoo, ngoài nhánh master, dev, và các nhánh fix-xxx, thì tất cả các nhánh còn lại đều có thể ngầm hiểu là nhánh chức năng.

Nhánh này dưới quyền cai quản của bạn, nên mặc sức muốn làm gì thì làm nhé. Hãy commit thường xuyên, dù chỉ là những thay đổi nhỏ nhất. Cũng đừng ngần ngại rằng commit nhỏ sẽ khiến git log khó theo dõi. Chúng ta sẽ có cách xử lý chúng sau.

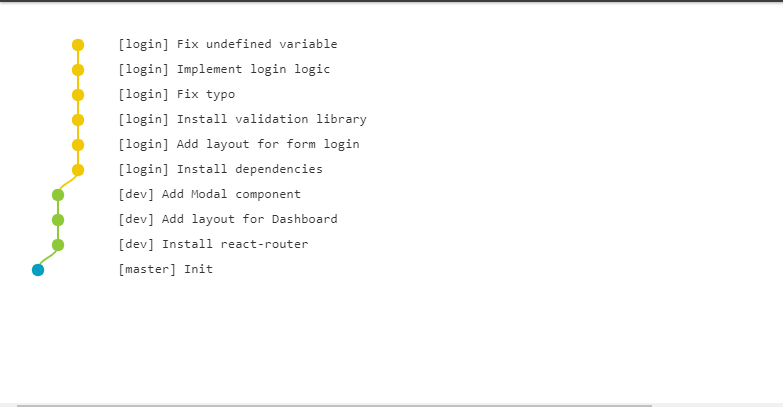
**Câu hỏi: Ê, lỡ như có hai hay nhiều người cùng làm chung một tính năng thì sao?**  
Nếu vậy, bạn có thể tiếp tục chia nhỏ hơn nữa, để đảm bảo mỗi người làm việc trên một nhánh độc lập. Cũng theo kinh nghiệm riêng của Ehkoo, thì một tính năng to sẽ có nhiều nhất là 2-3 người cùng phát triển. Nếu vượt quá con số này, thì nên xem lại định nghĩa và cách phân chia việc cho tính năng đó.

## Chuẩn bị merge vào dev

Sau khi code hoàn tất và tất cả unit tests đã chạy thành công, giờ là lúc bạn merge/gửi code để review tính năng mới vào dev. Thông thường, sẽ có 2 trường hợp xảy ra:

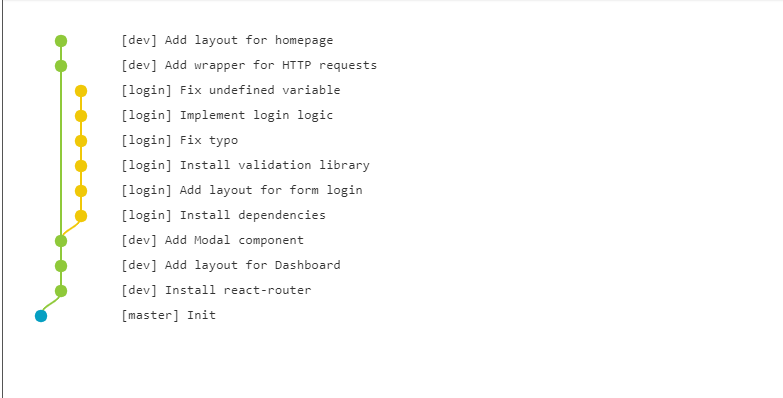
**Trường hợp 1: Không có gì mới trong dev**

Giả sử lúc đó Git history của dự án giống như thế này:

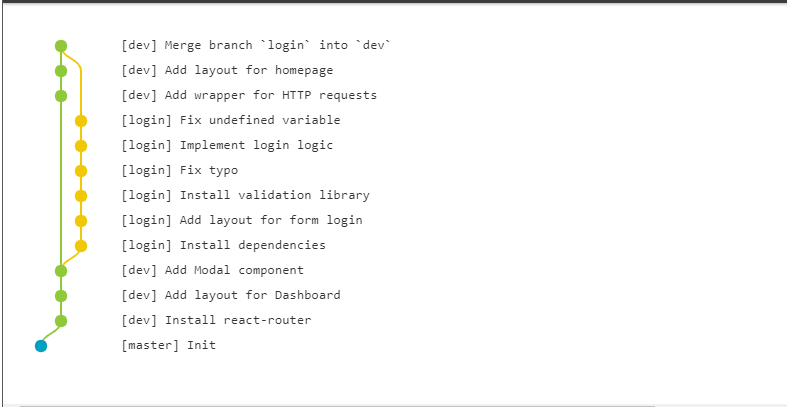


Như bạn thấy, nhánh login màu vàng được rẽ ra từ nhánh dev màu xanh, và trong nhánh dev không có code gì mới. Đây là trường hợp lý tưởng, đảm bảo khi merge vào dev chúng ta sẽ không bị xung đột code.

**Trường hợp 2: Có commits mới trong nhánh dev**



Trong trường hợp này, branch dev (màu xanh) đang có 2 commits phía trước branch login (màu vàng). Nếu trong 2 commits đó có chứa thay đổi liên quan đến dev, chẳng hạn như package.json, thì khả năng cao là sẽ xảy ra xung đột khi merge trực tiếp login vào. Mà dù có may mắn không xảy ra xung đột code, thì merge vào cũng sẽ làm history xấu đi.



Do đó, chúng ta sẽ cần sửa lại history của nhánh login bằng cách dùng git rebase.

#### git rebase là gì?

git rebase sẽ đem những commits bên trong nhánh login và áp dụng lại vào sau commit mới nhất trong nhánh dev. Cú pháp của lệnh này là:

git rebase <tên nhánh muốn áp dụng lại>

Trong trường hợp hiện tại, chúng ta sẽ chạy những lệnh sau:

*# Cập nhật repo hiện tại, đồng thời lấy về commits mới nhất của `dev`*

git pull

*# Chuyển qua nhánh `login`, có thể bỏ qua bước này nếu bạn chắc chắn*

*# mình đang ở `login`*

git checkout login

*# Tiến hành rebase*

git rebase dev

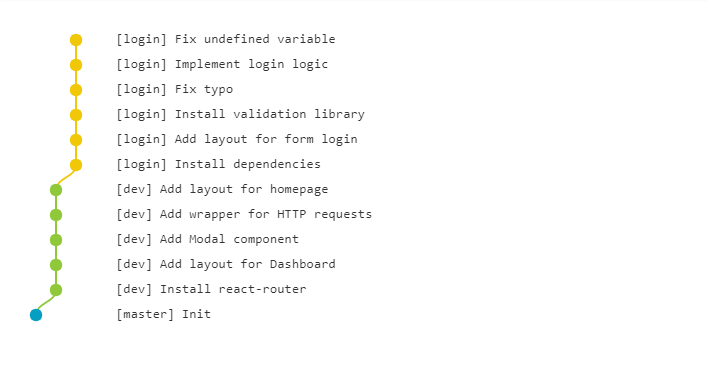
Nếu xảy ra xung đột code, bạn có thể phát hiện và giải quyết chúng sớm. Nguyên tắc chung là không sửa code của người khác, và chỉ kết hợp thêm những gì bạn làm. Việc thực hiện rebase tại nhánh chức năng do bạn phụ trách giúp giảm thiểu khả năng mất code, vì bạn là người hiểu rõ nhất phần code bạn viết.

Sau khi giải quyết hết các xung đột trong code, bạn chạy git rebase --continue để tiếp tục tiến trình rebase. Bạn cũng có thể chạy git rebase --abort để hủy bỏ rebase và đưa nhánh login về lại trạng thái ban đầu.

**Mách nhỏ:**  
Một cách giúp cho việc giải quyết xung đột trong code dễ dàng hơn là dùng git mergetool. Có rất nhiều công cụ hỗ trợ, và [Meld](http://meldmerge.org/) là một trong số đó.

Nếu chưa quen rebase, bạn có thể tạo một branch mới từ login, ví dụ: git checkout -b test login, và tiến hành rebase trên branch này. Sau khi chắc chắn là mọi thứ ổn thỏa, bạn có thể quay lại và tiến hành rebase cho login.

Khi rebase xong, mong là history của bạn trông sẽ giống như thế này:



Bạn thấy quen không? Chính là trường hợp 1 đã nói ở trên đó.

#### rebase interactively

Ở phần trên, chúng ta có băn khoăn là commit thường xuyên dễ tạo ra nhiều commit nhỏ đôi khi không cần thiết. Bạn có thể dùng git rebase để dọn dẹp chúng bằng cách thêm tham số -i (interactively) như sau:

*# Chắc chắn rằng bạn đang ở nhánh `login`*

git checkout login

*# Rebase lên dev interactively*

git rebase dev -i

Bạn sẽ được chuyển đến một giao diện tương tự như bên dưới.

pick ff80e85 A way to organize routes per module

pick 67cf18d Try Netlify Functions

pick 5546901 Add Dashboard view

pick 2a66ae3 Change layout

pick 58755b4 Add Books module, 404 page.

pick fd79cb9 Refactor. Reduce inline styling.

pick c671f60 Restyling 404 page.

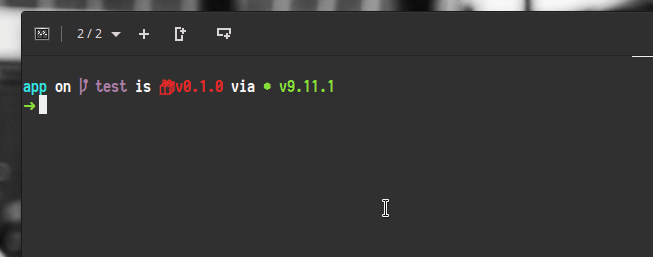
pick 33ef874 Basic layout for book management page.

pick 49c423a Clean up UI a bit

pick 3aa2840 Init

Theo lý thuyết, rebase sẽ đem từng commit và áp dụng lại theo thứ tự từ trên xuống dưới. Bởi vậy, bạn có thể thoải mái sắp xếp lại thứ tự của các commits trên. Bạn để ý lệnh pick ở phía trước mỗi commit. Lệnh này sẽ báo cho git biết hành động bạn muốn làm với commit, trong trường hợp này là áp dụng lại commit. Ngoài pick (p), chúng ta còn có:

* reword (r): áp dụng lại commit, và sửa commit message
* edit (e): áp dụng commit, nhưng dừng quá trình rebase lại để sửa code
* squash (s): kết hợp commit hiện tại vào commit trước nó
* fixup (f): giống như squash nhưng bỏ đi commit message
* exec (x): chạy một lệnh shell nào đó
* drop (d): bỏ, không sử dụng commit này

[[](https://res.cloudinary.com/duqeezi8j/image/upload/v1524470951/Peek_2018-04-23_11-06_zkqjpt.gif)](https://res.cloudinary.com/duqeezi8j/image/upload/v1524470951/Peek_2018-04-23_11-06_zkqjpt.gif)

Bạn có thể dùng các chữ viết tắt cho các commands

Bằng cách dùng rebase interactively, chúng ta có thêm nhiều quyền để quản lý và sửa đổi commits theo ý mình, làm cho history sạch đẹp hơn.

## Merge vào dev

Sau khi dọn dẹp nhánh login sạch đẹp, chúng ta có thể merge nhánh này vào dev. Thông thường, bạn -- dev cứng nhất team -- sẽ là người tiến hành kiểm tra và merge. Bạn có thể chọn hai cách tiếp cận:

#### merge

Bạn có thể merge trực tiếp vào dev như thế này:

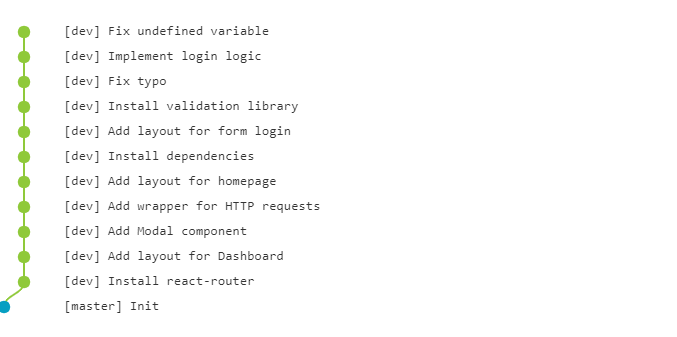
*# Chuyển qua nhánh `dev`*

git checkout dev

*# Merge `login` vào `dev`*

git merge login

Kết quả sẽ là:



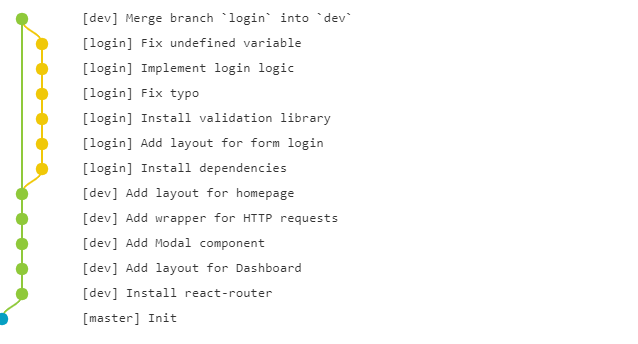
Cách thức này gọi là merge fast-forward. Tất cả commits của login đã được kết hợp vào dev. Boom! login biến mất khỏi thế gian như chưa hề tồn tại. SAD!

Lợi ích dễ thấy nhất của merge fast-forward là giúp cho history của bạn thẳng thớm gọn gàng, còn bất lợi là bạn không phân biệt được commits nào là của nhánh tính năng, cũng như thời điểm merge diễn ra. Trong trường hợp nhánh tính năng có quá nhiều commits nhỏ và dư thừa, chẳng hạn như những commits sửa lỗi chính tả, cập nhật thư viện..., history của bạn sẽ bị nhiễu.

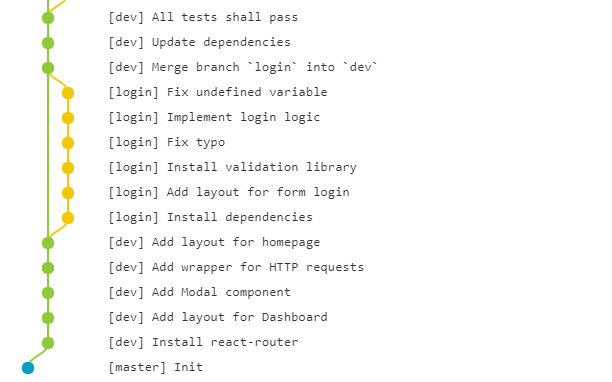
Bên cạnh đó, chúng ta cũng có cách merge non-fast-forward:

git merge login --no-ff

Và đây là kết quả:

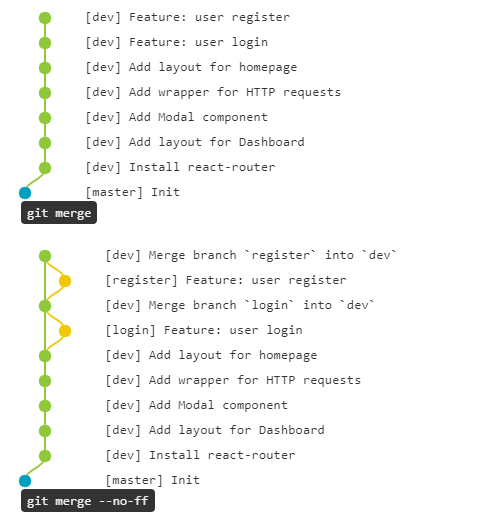


Như bạn thấy, một commit mới được tạo ra, giúp bạn dễ dàng nhận biết thời điểm nhánh login được merge vào. Khi dự án phát triển dần theo thời gian, history của dev sẽ như thế này.



#### rebase, squash và merge

Ngoài cách merge các commits của nhánh tính năng vào dev, bạn có thể rebase và squash tất cả commits lại làm một, sau đó tiến hành merge. Cách làm này giúp cho dev luôn ở trạng thái gọn gàng nhất, không chứa commit dư thừa. Trong trường hợp lý tưởng, history của dev sẽ giống như sau:



Để cách làm này phát huy tối đa hiệu quả, yêu cầu commit message phải được viết thật rõ ràng và chi tiết.

## Merge vào master

Yay! Sau một thời gian quằn quại, cuối cùng team của bạn đã ra được sản phẩm tương đối ổn. Giờ là lúc merge vào master và triển khai lên server.

Lúc này, cũng như khi merge vào dev, bạn có thể chọn merge (fast-forward hoặc non-fast-forward) hay rebase, squash và merge, nhưng theo kinh nghiệm của Ehkoo, merge --no-ff sẽ là lựa chọn tốt nhất, giúp cho master và dev luôn song song với nhau.

## Hotfix

Hôm nay, thứ 6, ngày 1X. Bạn chạy npm run build rồi rsync code ở master lên server. Mọi thứ hoàn toàn bình thường. Bạn vào website, click vài cái. "Ngon, chạy rồi", bạn thầm nghĩ, "đi nhậu thoy!" Nhưng vừa vươn vai định gọi điện cho gấu, thì "ó e ò e", chuông điện thoại reng, số máy của sếp. "Hí hí, chắc được thưởng nóng chăng?" Bạn bắt máy, và nghe giọng sếp âu yếm GẦM ở đầu dây: "LỖI RỒI MÁÁÁ!!!"

Xin đừng trụy tim. Hãy hít một hơi thật sâu, rồi bình tĩnh tạo một branch mới từ master, fix-xxx chẳng hạn. Nhờ lắng nghe tiếng sếp gầm, bạn đã mau chóng mò ra lỗi trong đống code (vì bạn là dev cứng mà hihi). Bạn khẽ rủa thầm ku đào nhí viết code không kĩ, tự rủa nhẹ bản thân vì review sót. Nhưng thôi kệ, fix nhanh rồi còn về, kẻo gấu xé xác T^T.

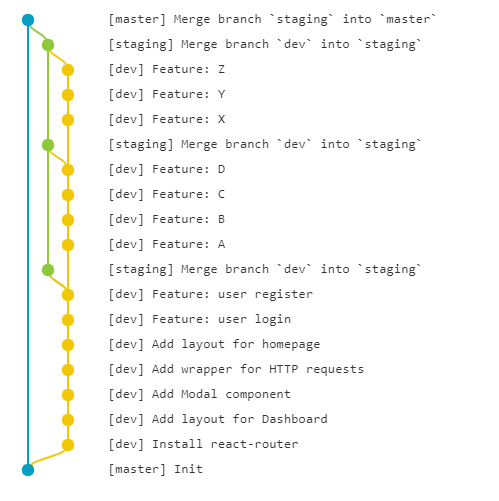
Bạn bèn merge --no-ff nhánh fix-xxx vào cả hai nhánh master và dev. Bằng cách này, phần sửa lỗi sẽ xuất hiện ở cả hai branches, giúp history không bị rẽ nhánh bất ngờ.

Bạn push, và chuông điện thoại lại vang lên...

## Vài vấn đề linh tinh khác

#### Có cần nhánh staging không?

Trong một số dự án, ngoài dev, còn có một đội ngũ "thần bí" được gọi là QA/QC. Họ được sinh ra trong team là để bới lỗi của anh em nhà dev, nên quan hệ đôi bên không được tình thương mến thương cho lắm. Dầu vậy, họ vẫn cần một nhánh riêng có tên gọi staging. Nhánh này sẽ chứa phần code ở giữa mastervà dev. staging được tách ra từ dev, có nhiều tính năng hơn master, và tương đối ổn định để có thể merge vào master.



Tùy vào tình hình cụ thể của team mà bạn quyết định có cần staging hay không.

#### Viết commit message như thế nào cho chuẩn?

Nếu bạn theo chuẩn rebase, squash và merge thì chuyện viết commit message tốt rất quan trọng, vì nó sẽ là tài liệu để mô tả toàn bộ một tính năng. Nhưng nên viết thế nào? Có một vài gợi ý cho bạn đây:

* Dòng đầu tiên không dài quá 80 chữ, luôn bắt đầu bằng động từ ở thì hiện tại, ngắn gọn súc tích, ví dụ: Add module Authentication. Bạn có thể chọn thêm tiền tố nếu cần thiết, chẳng hạn: Feature: Add module Authentication hay Fix: unable to get location params from URL
* Bỏ trống hai dòng
* Sau đó mô tả chi tiết về tính năng đang làm, những điểm cần lưu ý, phần nào của tính năng cần được cải thiện...
* Khuyến khích bạn kèm theo chữ ký signature khi commit bằng git commit –s

Một ví dụ

Feature: Add module Authentication

Signed-off-by: Long Dep Trai <long@ehkoo.com>

This module allows users to register/login into our website using

AWS Cognito account. Added routes:

\* /auth/register

\* /auth/login

Users after registration will receive a SMS to confirm their account.

TODO:

\* Implement social identities

\* Add Logout feature

\* Add Forgot password feature