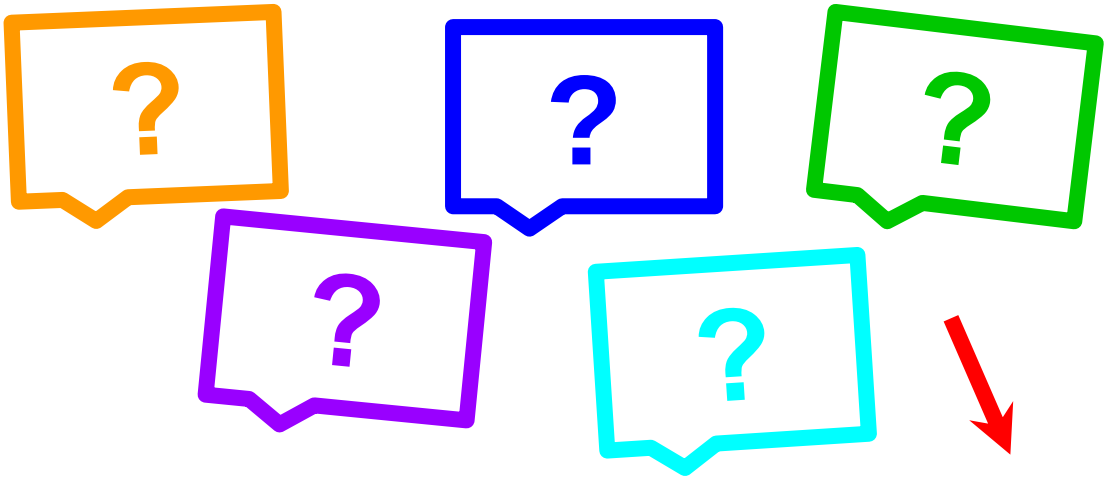
Git
Concepts and Workflows

Credits go to sasa.zivkov@sap.com, matthias.sohn@sap.com and christian.halstrick@sap.com

Nội dung

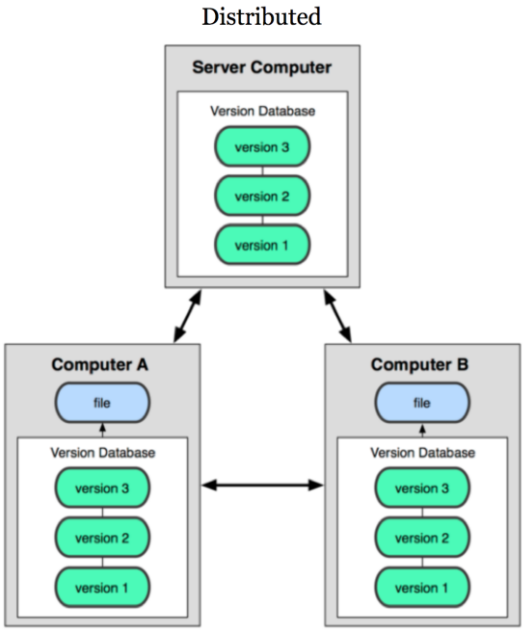
- Cấu trúc kho chứa của GIT
- Thay đổi nội dung
- Nhánh
- Clone + Fetch
- Merge, Rebase, Cherry-Pick
- Push
- Interactive Rebase

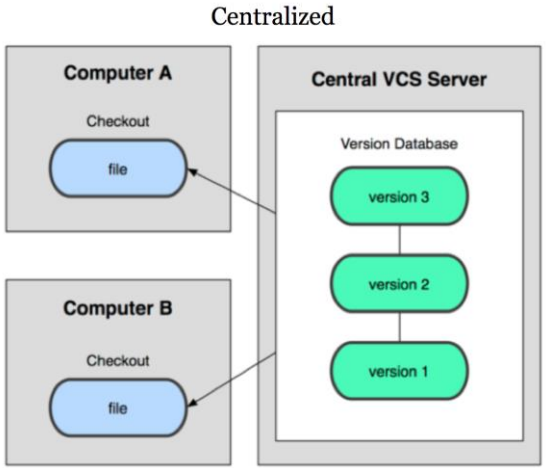
First question



Q: What's a distributed versioning system?

Hệ thống quản lý phiên bản phân tán





Phân tán có nghĩa là:

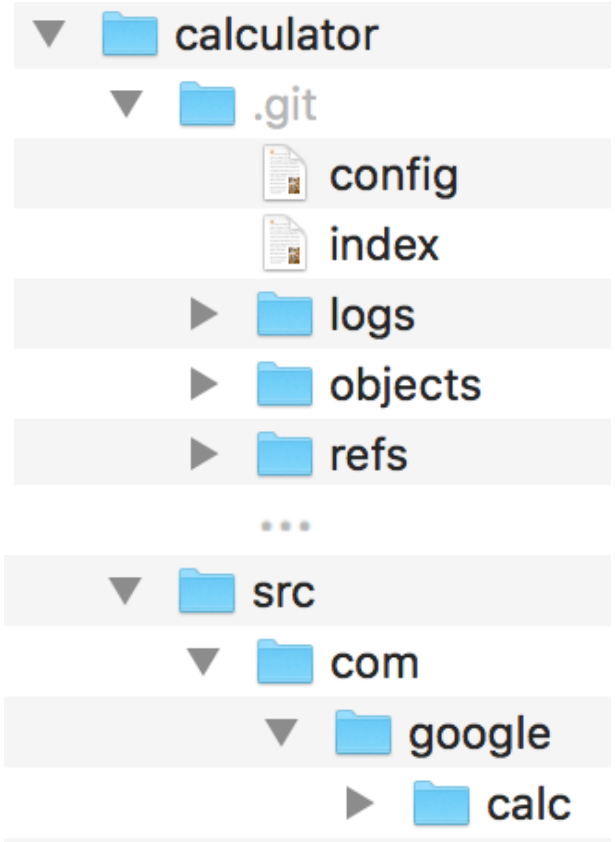
- Mỗi lập trình viên có một *kho chứa* riêng ở máy cá nhân
- Kho chứa chung không khác gì so với kho chứa của các máy cá nhân
- Dễ dàng sử dụng phiên bản offline
- Dễ dàng phân nhánh dự án
- Ví dụ: Git, Mercurial, Bazaar

GIT

- Phát hành năm 2005 để phục vụ cho lập trình Linux Kernel
- Được sử dụng cho Linux, Android, Eclipse
- Tích hợp trong Eclipse, Netbeans, XCode
- GitHub – host nổi tiếng nhất
- Được sử dụng ở Google, SAP, Qualcomm, Ericsson, Sony, Wikimedia, Intel, NVIDIA, Twitter, Garmin, etc.

git

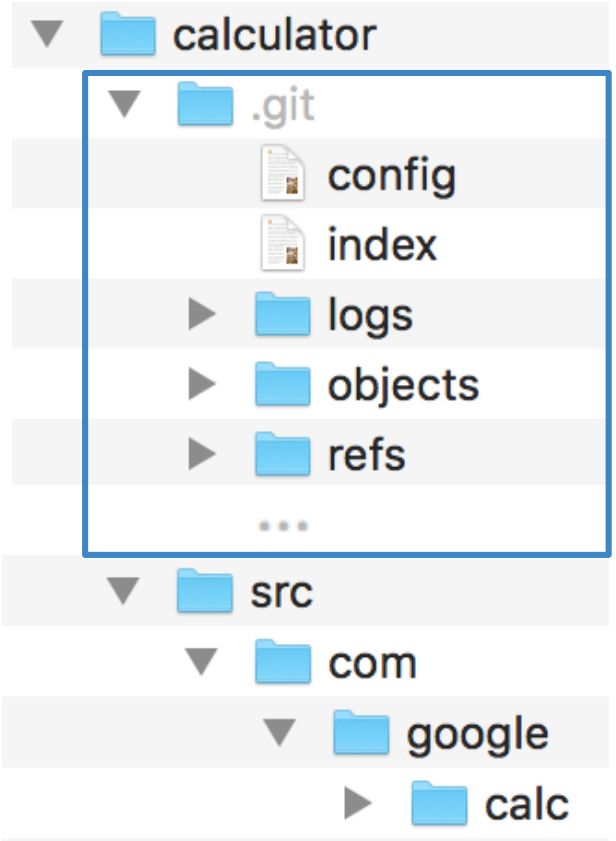
Cấu trúc kho chứa trong GIT



Một kho chứa (***Git repository***) tạo ra bằng 2 cách:

- `git init`
- `git clone` (giải thích ở phần sau)

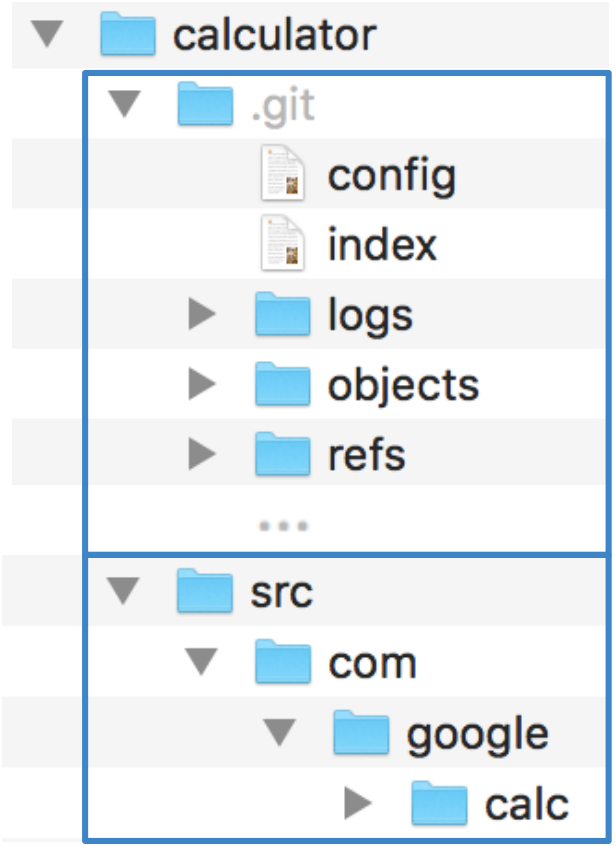
Cấu trúc kho chứa trong GIT



Thư mục ***.git*** là một kho chứa (***Git repository***)

- Thư mục ***.git*** chứa dữ liệu về tất cả các phiên bản.
- Hầu hết các file trong thư mục ***.git*** là có thể đọc hiểu được.

Cấu trúc kho chứa trong GIT

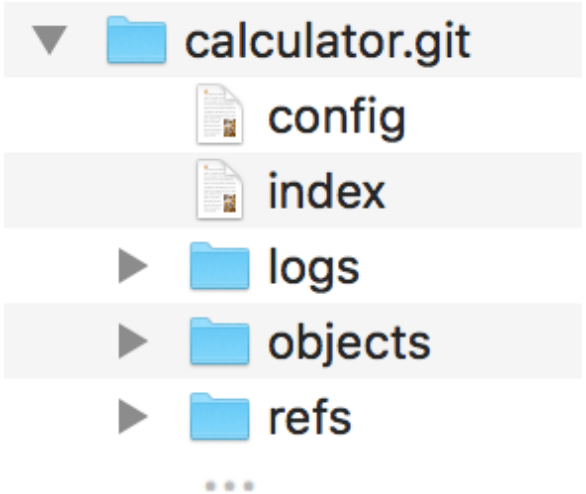


Thư mục ***.git*** là một kho chứa (***Git repository***)

Các file/thư mục đồng cấp với thư mục ***.git*** gọi là ***working tree***

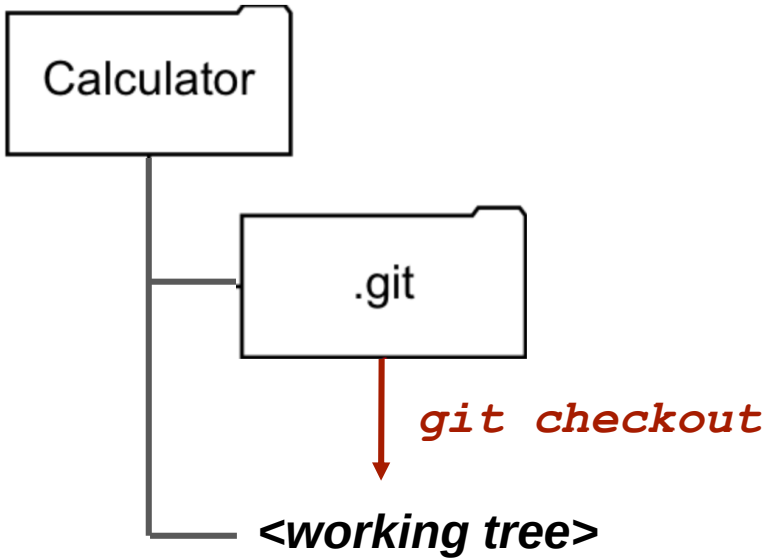
Một kho chứ có ít nhất một ***working tree***.

Cấu trúc kho chứa trong GIT



Một kho chứ không có working tree gọi là ***bare repository***.

Checkout

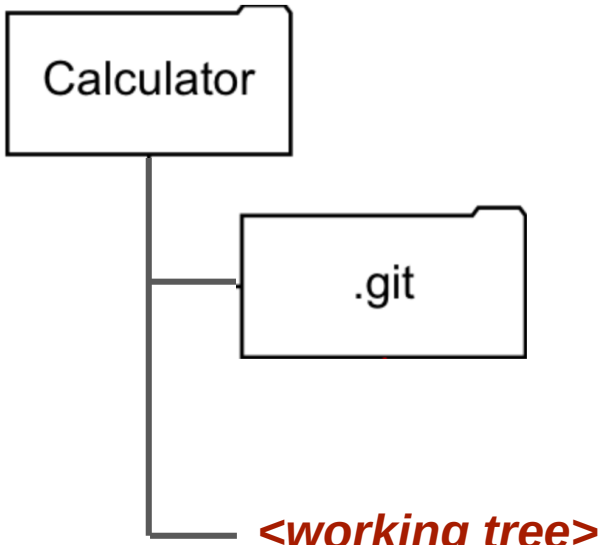


Checkout:

- Là thao tác lấy ra ***working tree*** nội dung của ***commit*** bạn muốn làm việc.

Commit là gì?

Thay đổi nội dung của Working tree



Bạn hãy thử thay đổi nội dung của working tree:

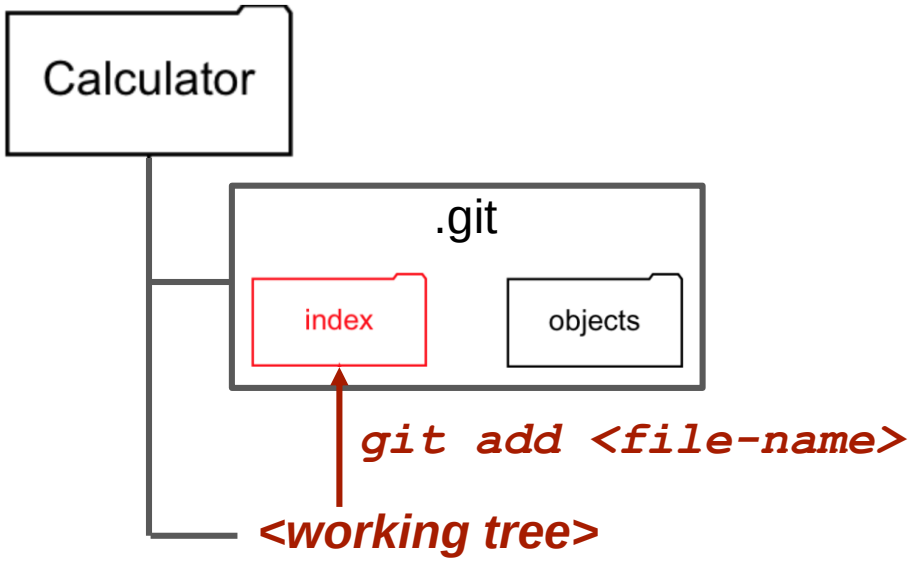
- Thêm, sửa hoặc xóa file.
- Nếu muốn thông báo cho git các thay đổi mà bạn muốn hoặc không muốn *commit*:

```
git add <file>
git rm <file>
```

commit là gì?

Q: Câu lệnh **`git add`** *sẽ làm gì?*

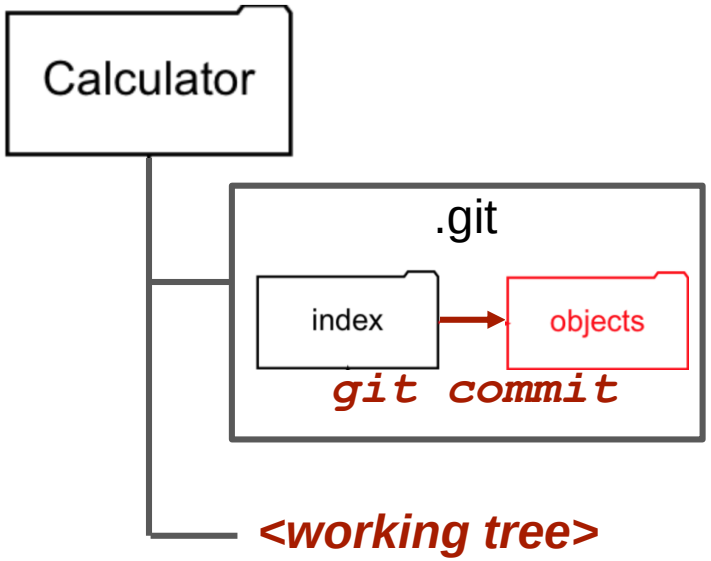
Thay đổi nội dung của Working tree



Trong kho chứa, thư mục ***Index*** (hay còn gọi là ***Staging Area***) là nơi chuẩn bị nội dung cho commit:

- `git add` và `git rm` để cập nhật nội dung của ***index***
- Stage single hunks: `git add -p <file>`
- Unstage files: `git reset HEAD <file>`

Thay đổi nội dung của Working tree

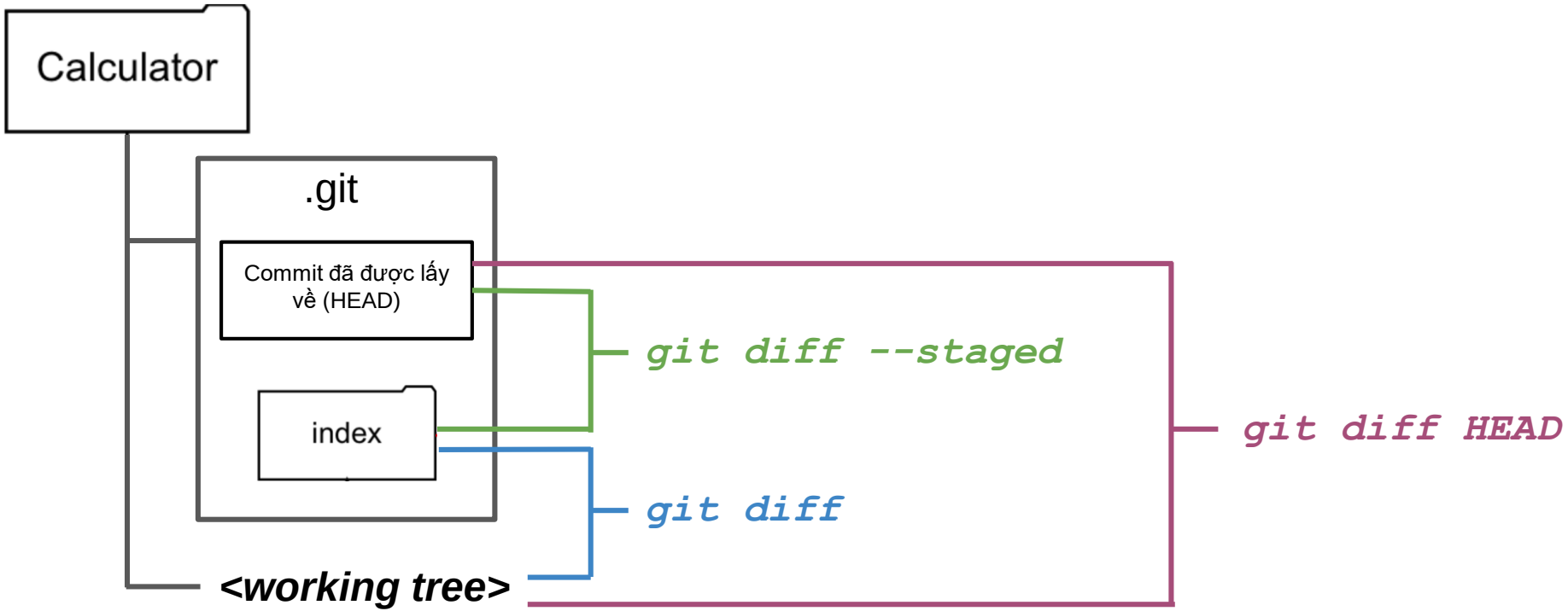


- `git commit` commits những thay đổi đã được ***staged*** (***index***)
- Có thể có những thay đổi trong ***working tree*** nhưng không được ***commit***.

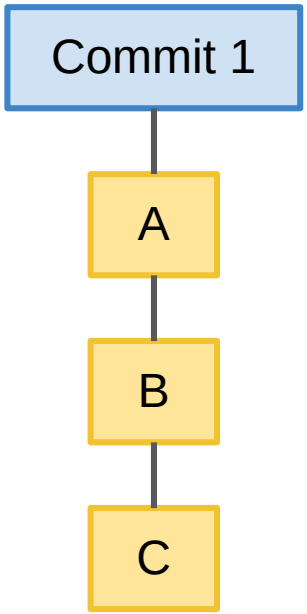
Xem lại sự thay đổi

- 3 trạng thái:
  - commit đã được lấy về (***`HEAD`***)
  - ***index*** (Những thay đổi đã được ghi nhận)
  - ***working tree*** (Thay đổi chưa được ghi nhận)
- `git status` hiển thị những sự thay đổi
  - Giữa ***index*** và commit đã được lấy về (***`HEAD`***)
  - Giữa ***working tree*** và ***index***
- `git diff` hiển thị các sự thay đổi trong từng file
  - Chi tiết trong slide tiếp theo

Xem lại sự thay đổi

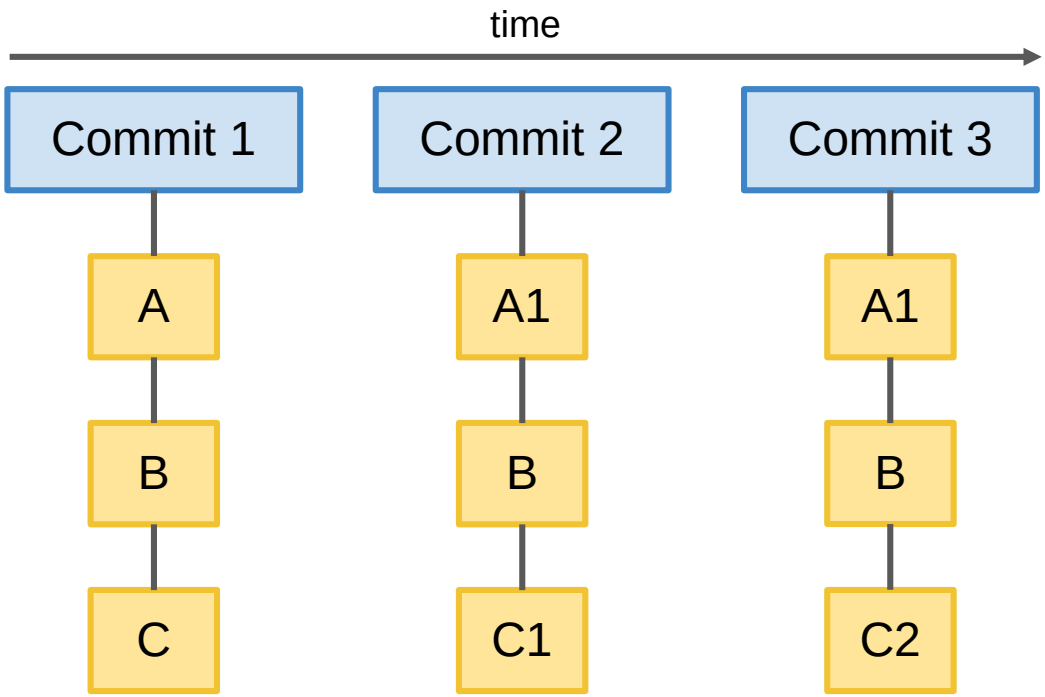


Commits



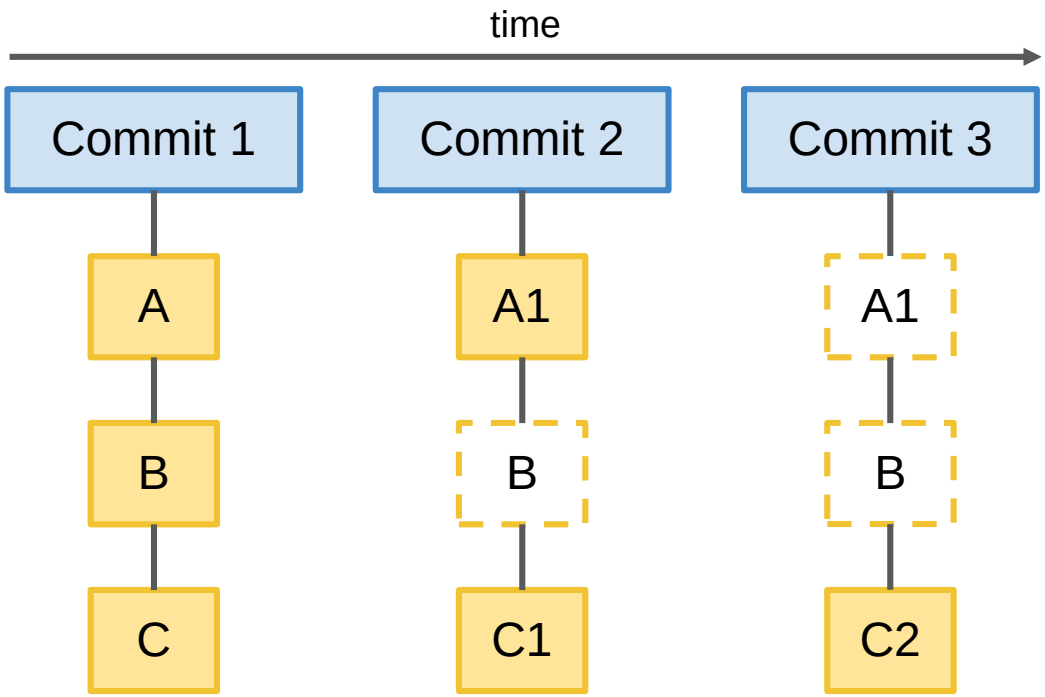
- Giả sử project của bạn có 3 files: ***A***, ***B*** và ***C***

Commits



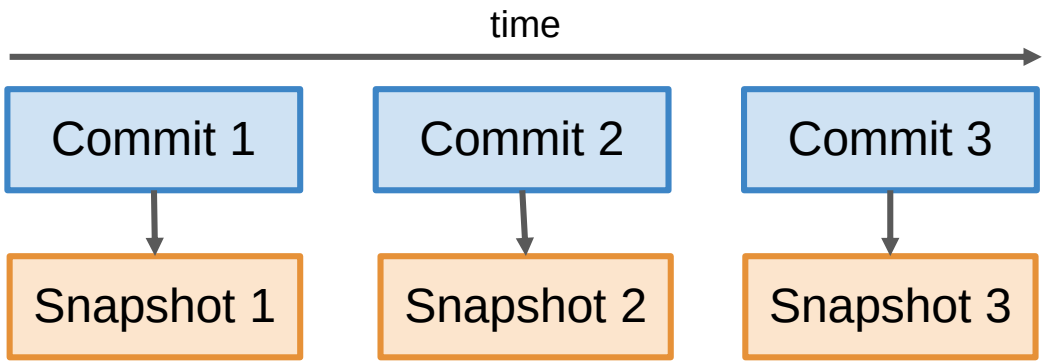
- Mỗi lần thay đổi nội dung của 3 file đó, bạn đều *commit* lên với 3 *commit* được tạo ra là ***Commit 1***, ***Commit 2***, ***Commit 3***
- Mỗi commit là một phiên bản đầy đủ project của bạn.

Commits



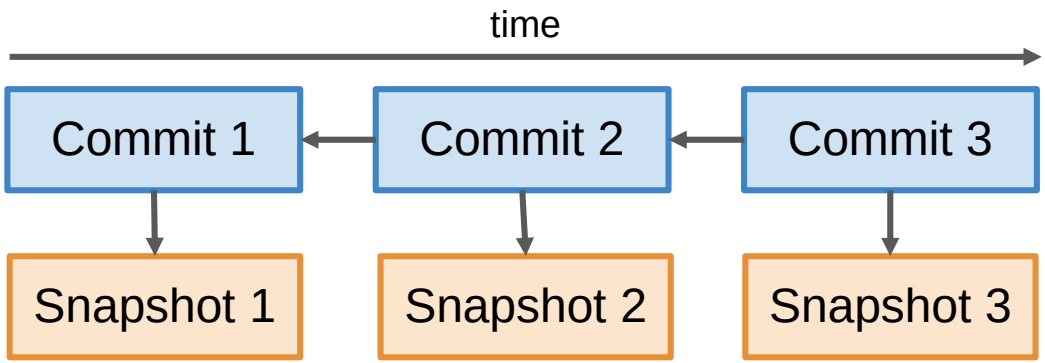
- Git tối ưu hóa việc lưu trữ và không tạo ra các bản copy của các file không thay đổi nội dung.

Commits



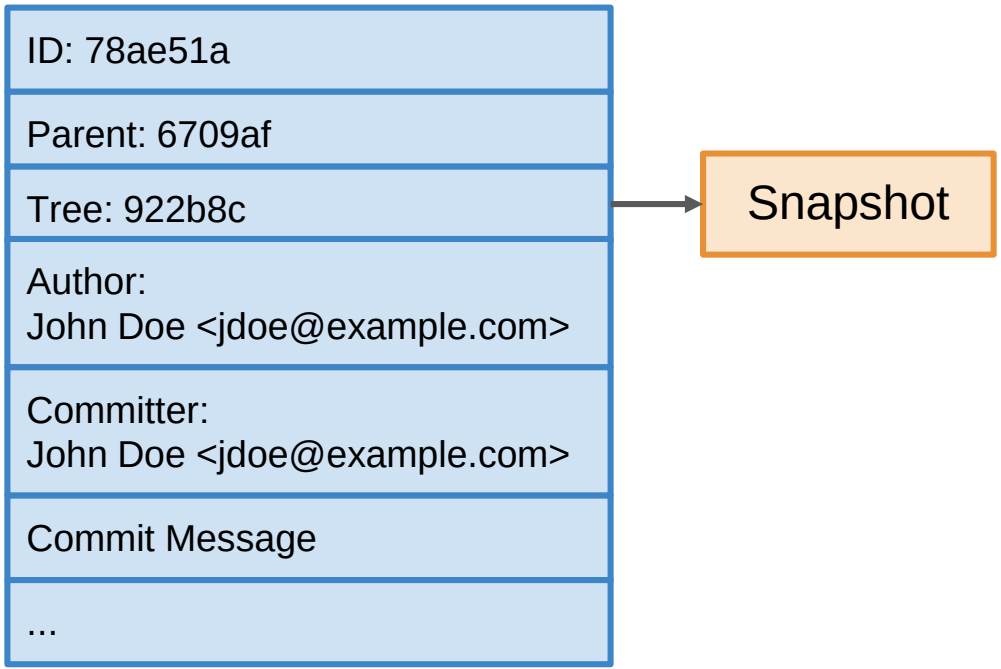
- Chỉ các file có sự thay đổi nội dung mới được lưu trữ.

Commits



- Mỗi commit cần lưu trữ commit cha của nó.

Cấu trúc của commit object



SHA1:

- ID của commit
- Được hiển thị bằng lệnh

```
git log
```

Quan sát một commit:

- `git show <SHA1>`
- `git show --format=fuller <SHA1>`

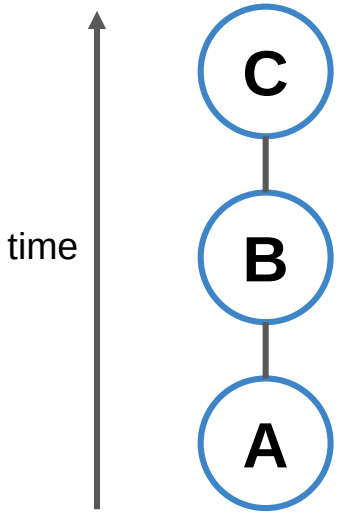
Khi đã được tạo ra, tên commit là không thể thay đổi

Commit Message

```
First line is the subject, should be shorter than 70 chars

Separate the body from the subject by an empty line. The
commit message should describe why you are doing the
change. That's what typically helps best to understand
what the change is about. The details of what you changed
are visible from the file diffs.

The body can have as many paragraphs as you want. Lines
shouldn't exceed 80 chars. This helps command line tools
to render it nicely. Paragraphs are separated by empty
lines.

Bug: Issue 123
```

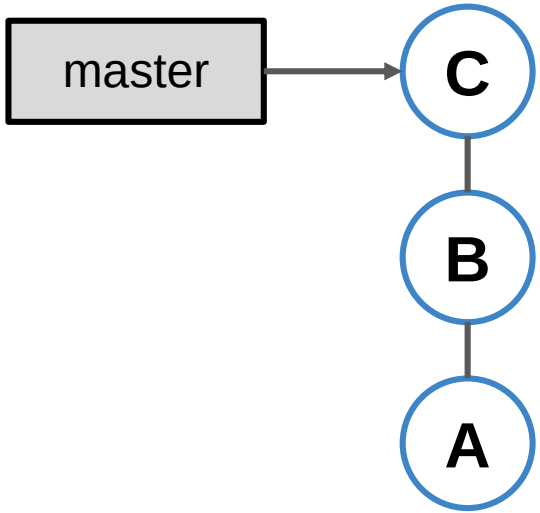
- Thường thì:
  - Dòng đầu tiên là tiêu đề mô tả ngắn gọn commit
  - Phần thân cách phần tiêu đề 1 dòng trống
  - Đoạn cuối cùng là metadata.

Lịch sử các commit



- ***C*** là commit con của ***B***
- ***B*** là commit con của ***A***
- Dòng commit như hình bên thể hiện mối quan hệ giữa các commit, tức là lịch sử của một project.
- Để xem lịch sử, có thể dùng:
  - `git log`
  - `git log --oneline`
  - `git log -graph`
  - `gitk`

Branches/Nhánh

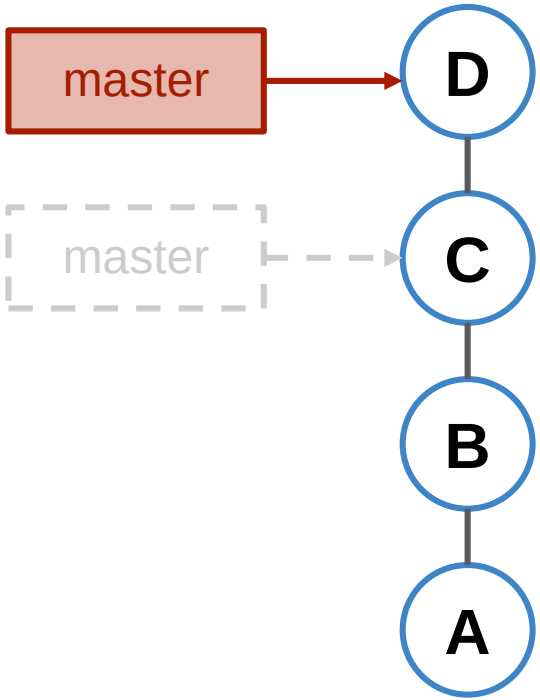


Một nhánh (***branch***) là một con trỏ có tên, trỏ vào một commit:

- Ví dụ: `master`
- Tên đầy đủ của con trỏ: `refs/heads/master`
- Tất cả cách commit có thể đến được từ một nhánh gọi là lịch sử nhánh (***branch history***)
- Commit được trỏ bởi branch gọi là ***branch tip***.

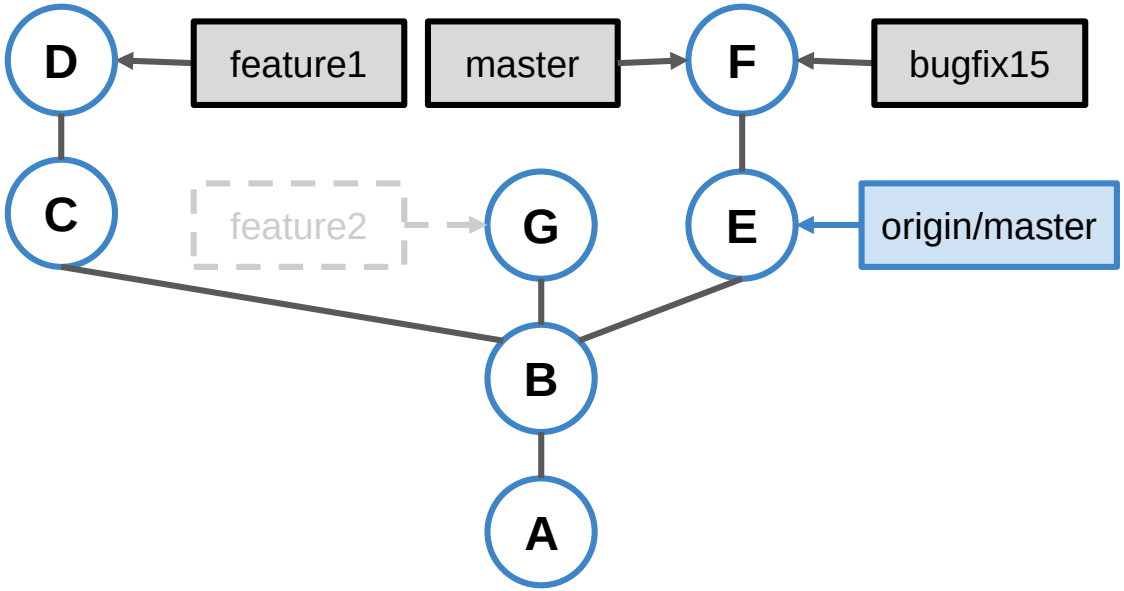
Q: Nếu commit một commit mới, điều gì sẽ xảy ra?

Branches/Nhánh



- Một commit mới được tạo ra ***D***.
- Nhánh được dịch chuyển để trỏ vào commit mới.

Branches/Nhánh



Thông thường, có nhiều nhánh trong một kho chứ:

- Nhiều nhánh có thể trỏ vào cùng một commit.
- Nhánh có thể xóa bằng câu lệnh: `git branch -D <branchname>`
- Nhánh `master` chỉ là nhánh bình thường, không có gì đặc biệt ngoài tên của nó có vẻ thượng đẳng.
- `origin/master` là nhánh remote tracking (giải thich sau).

Tạo nhánh bằng cách:

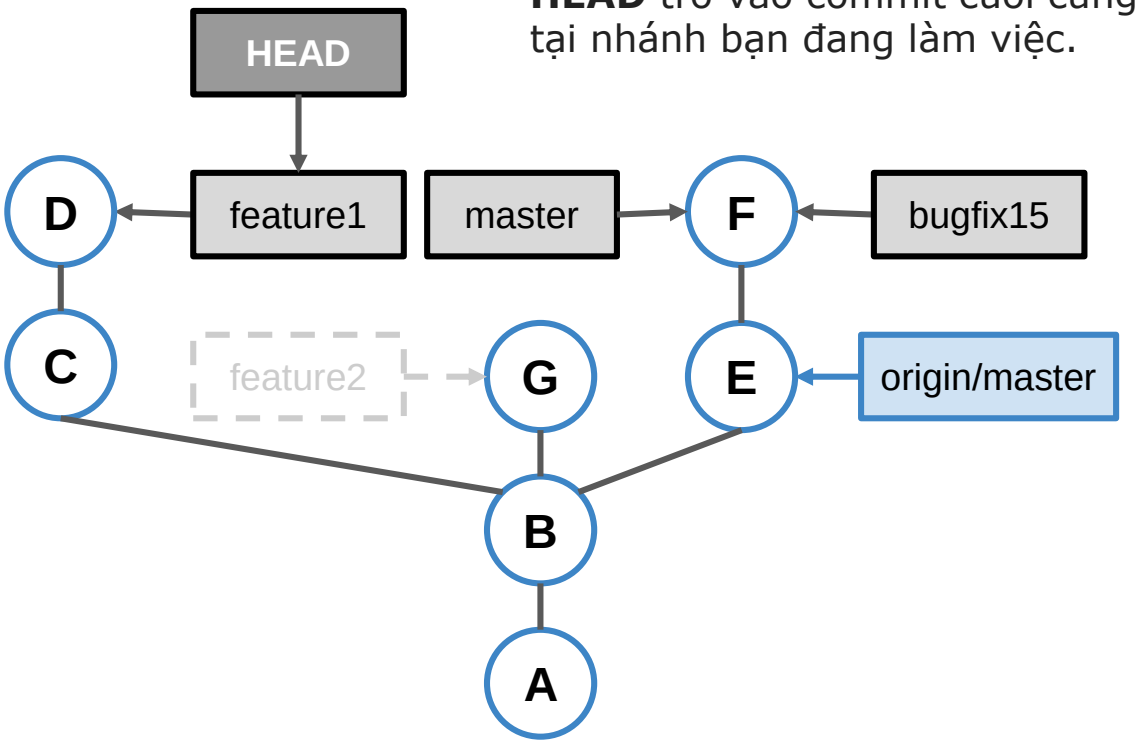
- `git branch <branchname>`
- `git checkout -b <branchname>`

List all branches:

- `git branch -a`

Q: Git làm cách nào để biết là commit mới sẽ được đặt lên trên branch nào?

HEAD



HEAD trỏ vào nhánh hiện tại:

- `git commit` cập nhật commit mới vào nhánh hiện tại.
- `git checkout` gán giá trị cho nhánh hiện tại. Tức là muốn thay đổi giá trị của branch hiện tại thì dùng lệnh này.

Q: Lệnh sau làm gì `git checkout bugfix15` *?*

HEAD

- Về cơ bản, HEAD luôn trỏ vào *commit* cuối cùng của nhánh hiện tại.
- Khi dùng lệnh checkout với nhánh, HEAD sẽ trỏ vào commit cuối cùng của nhánh bạn muốn chuyển tới.
- Khi dùng lệnh checkout với *commit*, con trỏ HEAD sẽ bị *detach* (không trỏ vào commit cuối của nhánh)

[Nhớ lại] `checkout` có 2 cách dùng:

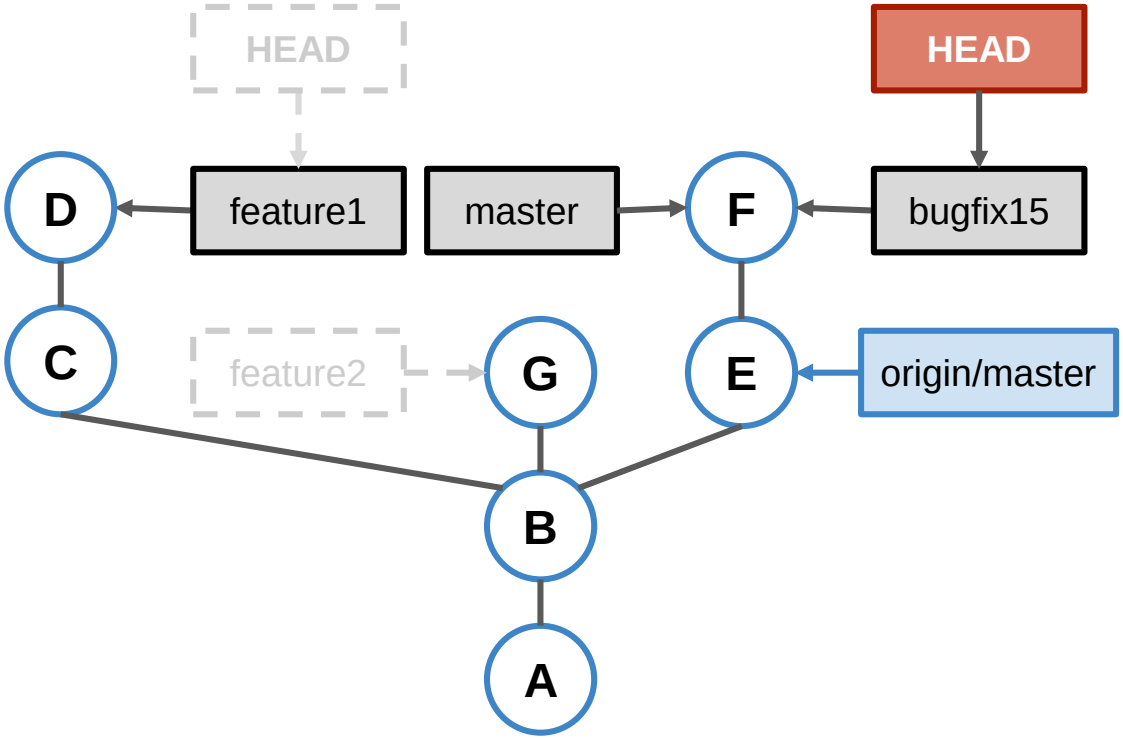
- `git checkout <tên nhánh>`

Đưa con trỏ HEAD về commit đầu tiên của nhánh.

- `git checkout <ID commit>`

Đưa HEAD về commit cụ thể. Nếu nó không phải là commit cuối của nhánh, HEAD bị detach.

HEAD



git checkout:

- Chuyển con trỏ ***HEAD***
- Cập nhật lại ***working tree***

Q: Điều gì xảy ra nếu thay đổi mã lệnh của branch sai? (quên checkout về branch cần thay đổi mã lệnh)

Checkout khi working tree có mã lệnh thay đổi

Nếu ta thay đổi mã lệnh trên nhầm branch, cần làm các thao tác sau:

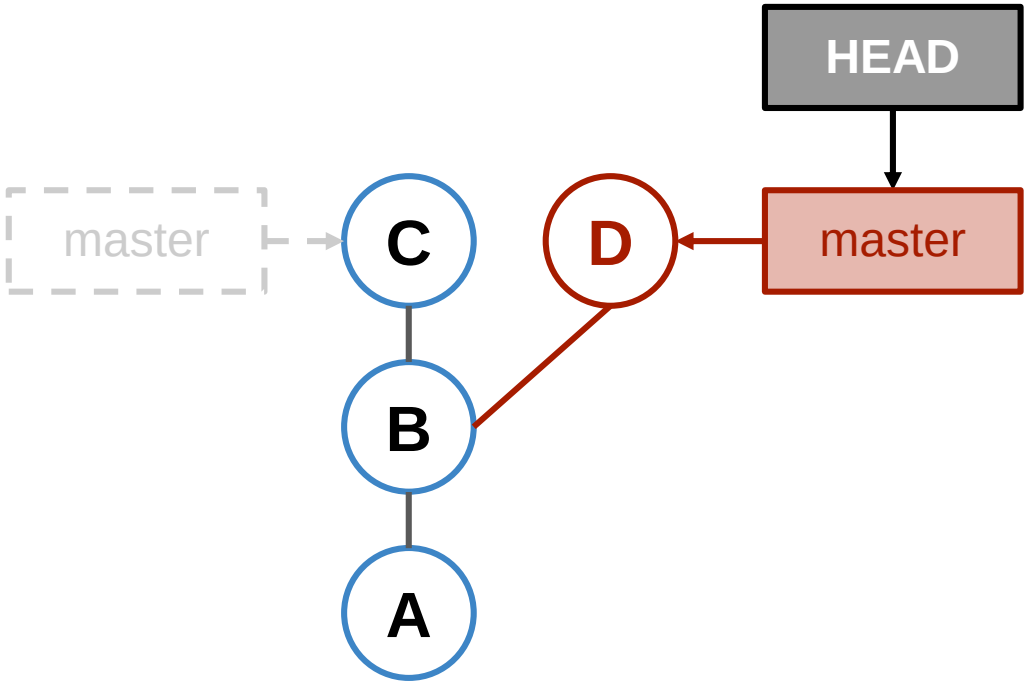
- Thử checkout về branch đúng, nếu không có *conflicts*, chỉ việc làm tiếp
- Nếu có conflicts vè checkout không thành công, cần làm:

```
$ git stash
$ git checkout <correct-branch>
$ git stash pop
$ <resolve conflicts>
$ git stash drop
```

- ***working tree*** có sự thay đổi nội dung so với commit gọi là bẩn (***dirty***)
- ***working tree*** không bị thay đổi nội dung gọi là sạch (***clean***)
- `git stash` đặt tất cả các thay đổi của một working tree bẩn sang một bên. Ngược lại, `git stash pop` đưa các sự thay đổi đó trở lại branch hiện tại.

Q: Đã commit một commit lên, nhưng muốn thêm một sự thay đổi vào commit đó thì làm thế nào?

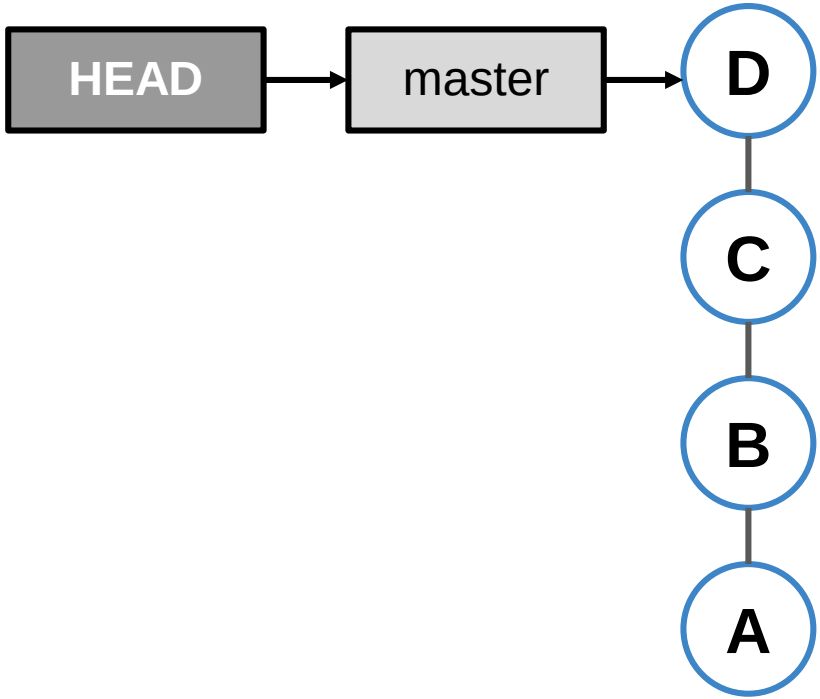
Viết lại lịch sử: Amend Commit



git commit --amend viết lại nội dung của commit cuối:

- Tạo ra một commit ***D*** là anh em của commit cuối ***C*** và ***trỏ lại*** nhánh hiện tại vào D.
- Commit message của C được copy sang D, và có thể thay đổi nó nếu muốn.
- commit ***C*** vẫn trong kho chứa.
- Lệnh này viết lại lịch sử của branch (khi bạn đã chia sẻ code cho người khác, không nên viết lại lịch sử)

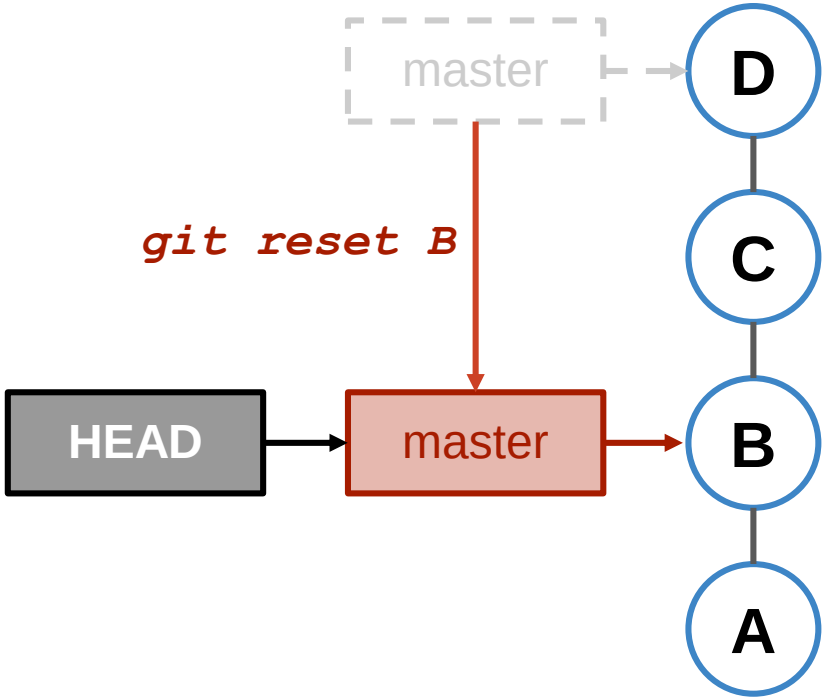
Resetting Branches



- Nhánh có thể dịch chuyển một cách thủ công bằng lệnh `git reset`.

Q: Làm thể nào để reset nhánh về commit B?

Resetting Branches

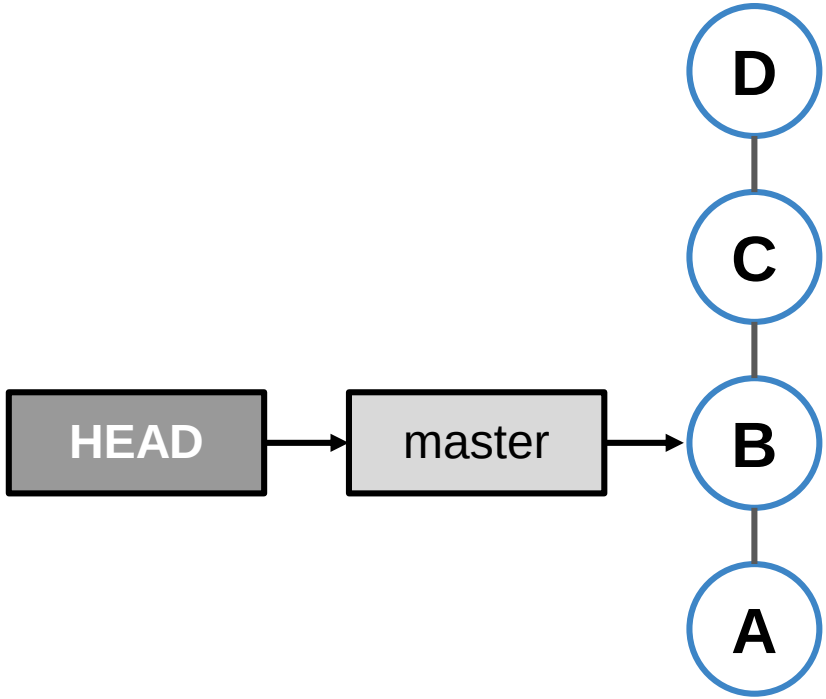


`git reset B`

- Cập nhật ***branch*** hiện tại để cho nó chỉ vào ***B***.
- Commit ***C*** và ***D*** không còn nằm trong lịch sử của branch nữa.

Q1: C và D sẽ ra sao?

Non-reachable Commits/Commits không truy cập được

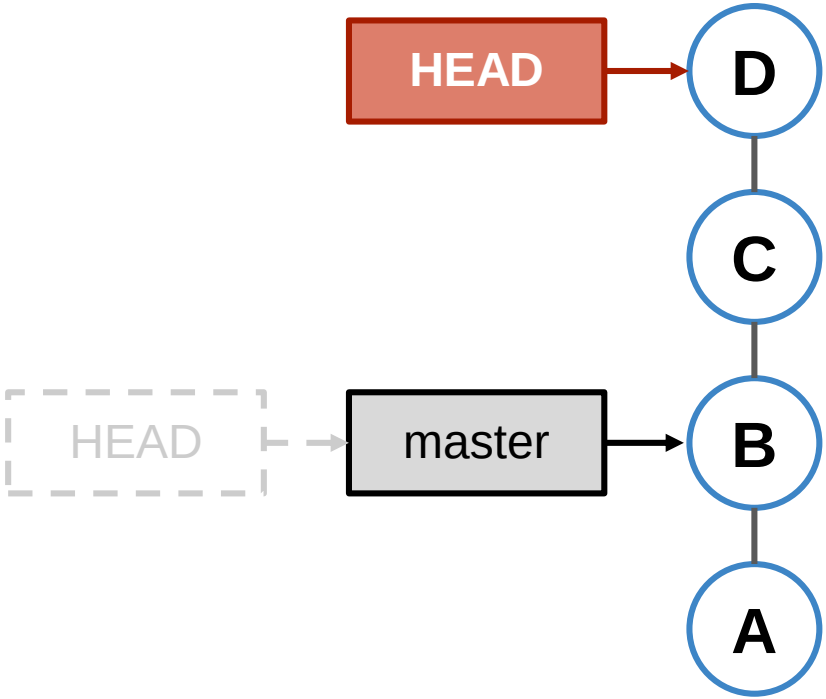


Các commit không truy cập được

- Được giữ trong 2 tuần
- Sau 2 tuần, chúng sẽ bị xóa bằng trình dọn rác của git
- Có thể truy cập thông của mã SHA1 của chúng.

Q: Điều gì sẽ xảy ra nếu check out vào commit không truy cập được?

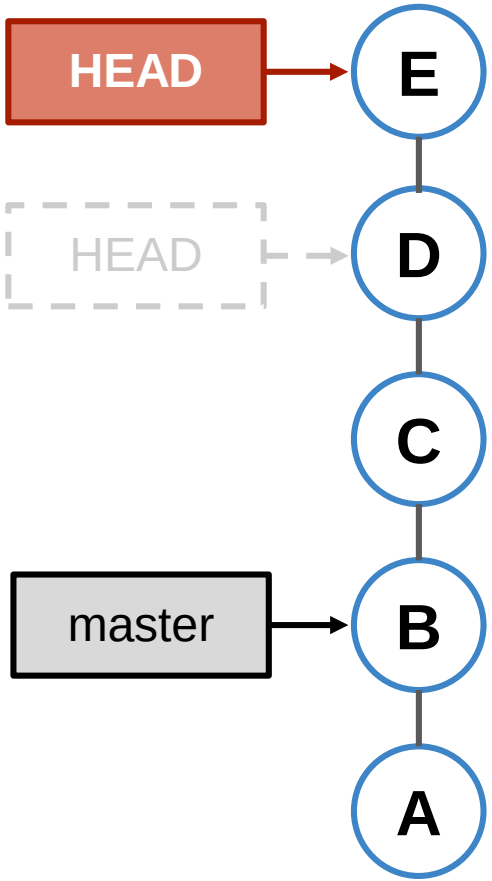
Detached HEAD



Nếu `HEAD` trỏ trực tiếp vào commit (thay vì trỏ vào 1 branch) nó gọi là ***detached HEAD***.

Q: Với trạng thái hiện tại, Nếu thay đổi nội dung working tree và commit lên thì điều gì sẽ xảy ra?

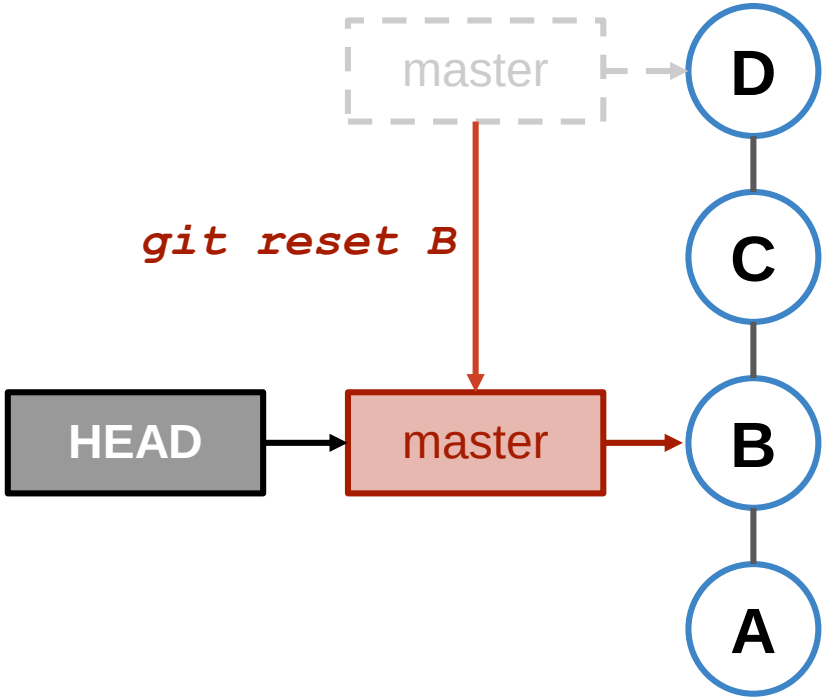
Detached HEAD



Có thể tạo ra commit mới, kể cả trong trường hợp con trỏ HEAD bị *detached*:

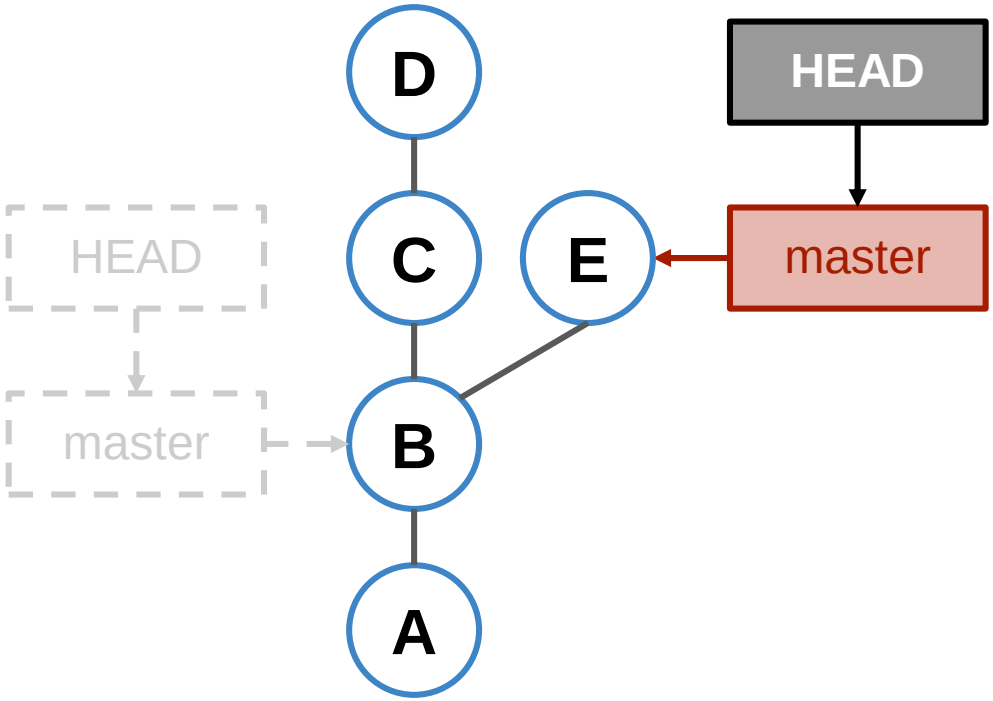
- Nếu bạn checkout một commit khác trong lịch sử nhánh, commit hiện tại sẽ không thể dò lại được (trừ khi bạn có mã SHA1 của nó).

Resetting Branches



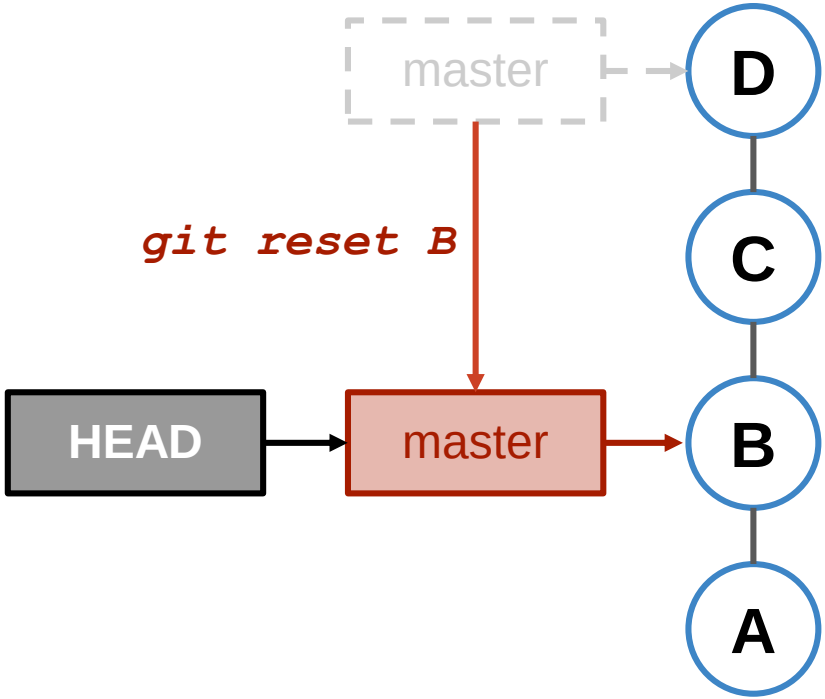
Q2: Sau khi reset, commit mới sẽ được đặt vào đâu?

Commit mới sau reset



- Commit mới sẽ là con của commit đang được trỏ bởi HEAD.
- Nhánh hiện tại được *updated*.

Resetting Branches



Q3: What happens to the working tree and the index on branch reset?

Resetting Branches

git reset	branch	index	working tree
--soft	**Yes**	**No**	**No**
--mixed (default)	**Yes**	**Yes**	**No**
--hard	**Yes**	**Yes**	**Yes**

Khi chạy câu lệnh reset, nhánh luôn được reset, ***index*** và ***working tree*** được reset phụ thuộc vào reset mode (*soft*, *mixed*, *hard*).

Dùng *git reset --hard*
Tất cả các thay đổi trên working tree sẽ bị mất.

Q: Các modes dùng khi nào?

Resetting Branches

`git reset`	branch	index	working tree
`--soft`	**Yes**	**No**	**No**
`--mixed` (default)	**Yes**	**Yes**	**No**
`--hard`	**Yes**	**Yes**	**Yes**

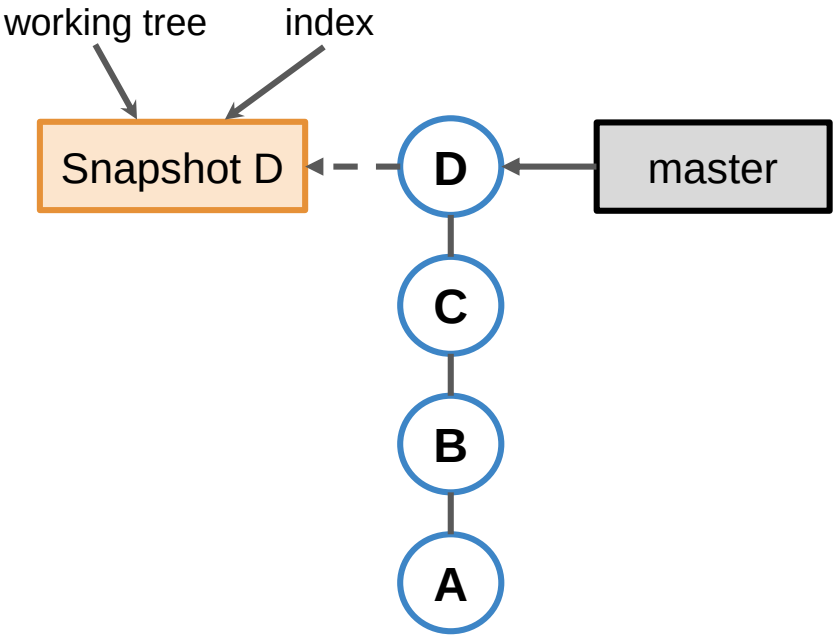
Sử dụng:

- `git reset --hard`: bỏ qua toàn bộ các thay đổi trên working tree.
- `git reset --soft`: Squash commits.
- `git reset --mixed`: phân tách commits.

Q: Làm thế nào để: (1) xóa commit trong lịch sử nhánh, hoặc (2) tách commit ra thành 2?

Xóa commit ở giữa bằng soft reset

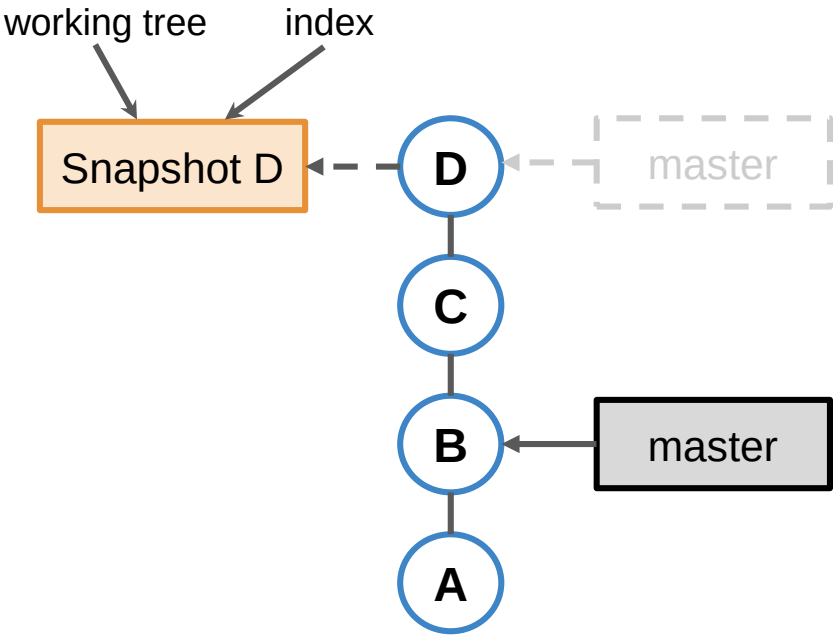
git reset	branch	index	working tree
--soft	**Yes**	**No**	**No**



1. *git checkout D*:
   Nhánh hiện tại trỏ vào D, index và working tree chứa nội dung của D.

Xóa commit ở giữa bằng soft reset

git reset	branch	index	working tree
--soft	**Yes**	**No**	**No**

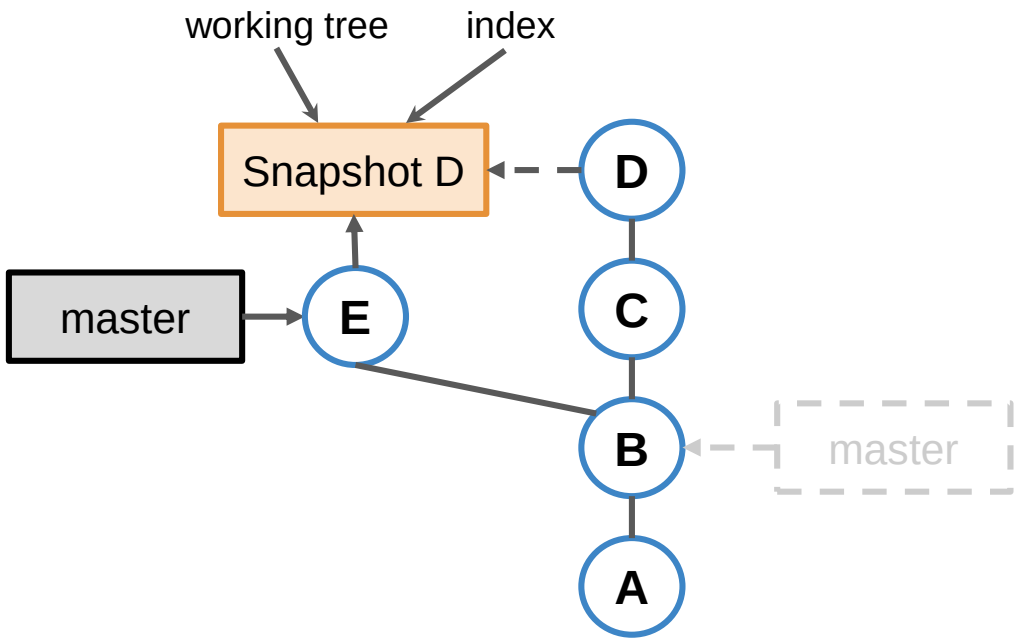


1. *git checkout D*:
   Nhánh hiện tại trỏ vào D, index và working tree chứa nội dung của D.

2. *git reset --soft B*:
   Nhánh hiện tại trỏ vào B, index và working tree vẫn chứa nội dung của D.

Xóa commit ở giữa bằng soft reset

git reset	branch	index	working tree
--soft	**Yes**	**No**	**No**



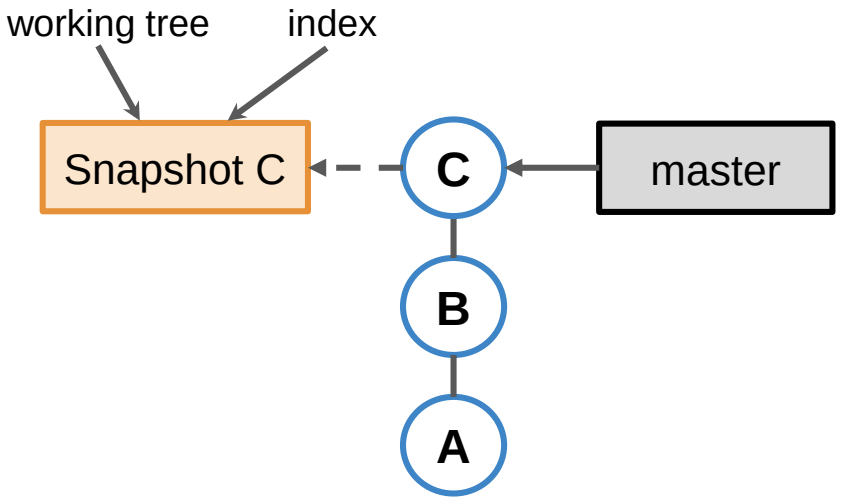
1. *git checkout D*:
   Nhánh hiện tại trỏ vào D, index và working tree chứa nội dung của D.

2. *git reset --soft B*:
   Nhánh hiện tại trỏ vào B, staging index và working tree vẫn chứa nội dung của D.

3. *git commit*:
   Commit mới E được tạo ra từ staging index. Mà hiện tại, staging index là nội dung của D, nên E và D có nội dung giống nhau

Tách commit bằng mixed reset

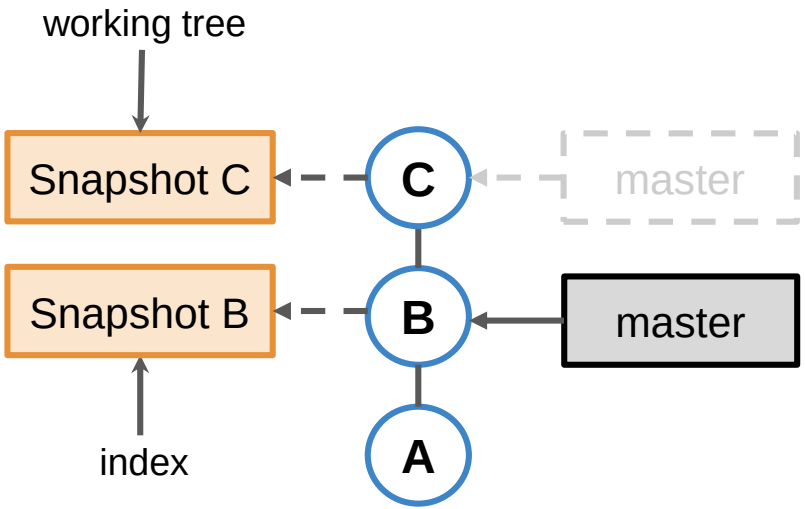
git reset	branch	index	working tree
--mixed (default)	**Yes**	**Yes**	**No**



1. *git checkout C*:
   Nhánh hiện tại là master. Nội dung của staging index và working tree là nội dung của C

Tách commit bằng mixed reset

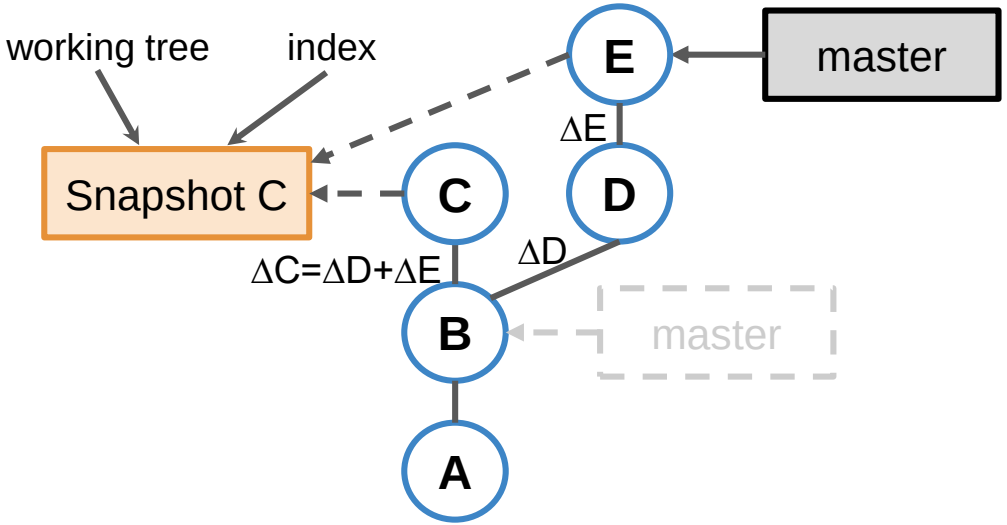
git reset	branch	index	working tree
--mixed (default)	**Yes**	**Yes**	**No**



1. *git checkout C*:
   Nhánh hiện tại là master. Nội dung của staging index và working tree là nội dung của C
2. *git reset --mixed B*:
   Nhánh hiện tại trỏ vào B, staging index chứa nội dung của B, working tree vẫn là nội dung của C

Tách commit bằng mixed reset

git reset	branch	index	working tree
--mixed (default)	**Yes**	**Yes**	**No**



1. *git checkout C*:
   Nhánh hiện tại là master. Nội dung của staging index và working tree là nội dung của C
2. *git reset --mixed B*:
   Nhánh hiện tại trỏ vào B, staging index chứa nội dung của B, working tree vẫn là nội dung của C
3. *git add <file> && git commit*:
   Đưa một vài thay đổi vào D.
4. *git add <file> && git commit*:
   Đưa phần thay đổi còn lại vào E.

BÀI TẬP