Bài 1

VCS - Git

Agenda

- 1 Getting Started
- 1.1 About Version Control System
  - VCS in a nutshell
  - Local version control systems
  - Centralized version control systems
  - Distributed version control systems
- 1.2 Short history of Git
- 1.3 What is Git
- 1.4 Installing Git
- 1.5 First-time Git Setup
- 1.6 Getting Help

Agenda

- 2 Git Basics
- 2.1 Getting a Git Repository
- 2.2 Recording Changes to the Repository
- 2.3 Viewing th Commit History
- 2.4 Undoing Things
- 2.5 Working with Remotes

Agenda

- 3. Git branching
- 3.1 Branches in a Nutshell
- 3.2 Basic branching and merging
- 3.3 Branching management
- 3.4 Branching workflows
- 3.5 Remote branches
- 3.6 Rebaseing
- 3.7 Bonus

1.1 About Version Control

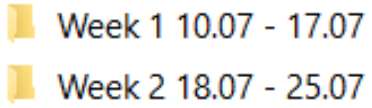
- Version control (Quản lý phiên bản) là gì
- Là công cụ cho phép lưu trữ sự thay đổi của tập tin (file) theo thời gian, phiên bản.
- Do đó bạn có thể quay lại một phiên bản – version tại bất kì thời điểm nào
- Hỗ trợ làm việc dễ dàng với nhóm nhiều người

- VCS – Version control System cho phép
  - Làm chung nhóm nhiều thành viên
  - Khôi phục lại phiên bản cũ của các file
  - Khôi phục lại phiên bản cũ của toàn bộ dự án
  - Xem lại các thay đổi đã được thực hiện theo thời gian
  - Xem lại ai là người thay đổi tập tin gây ra sự cố
  - Khôi phục các tập tin đã thay đổi, bị xóa

- Sử dụng dễ dàng và nhanh chóng thông qua cú pháp của VCS

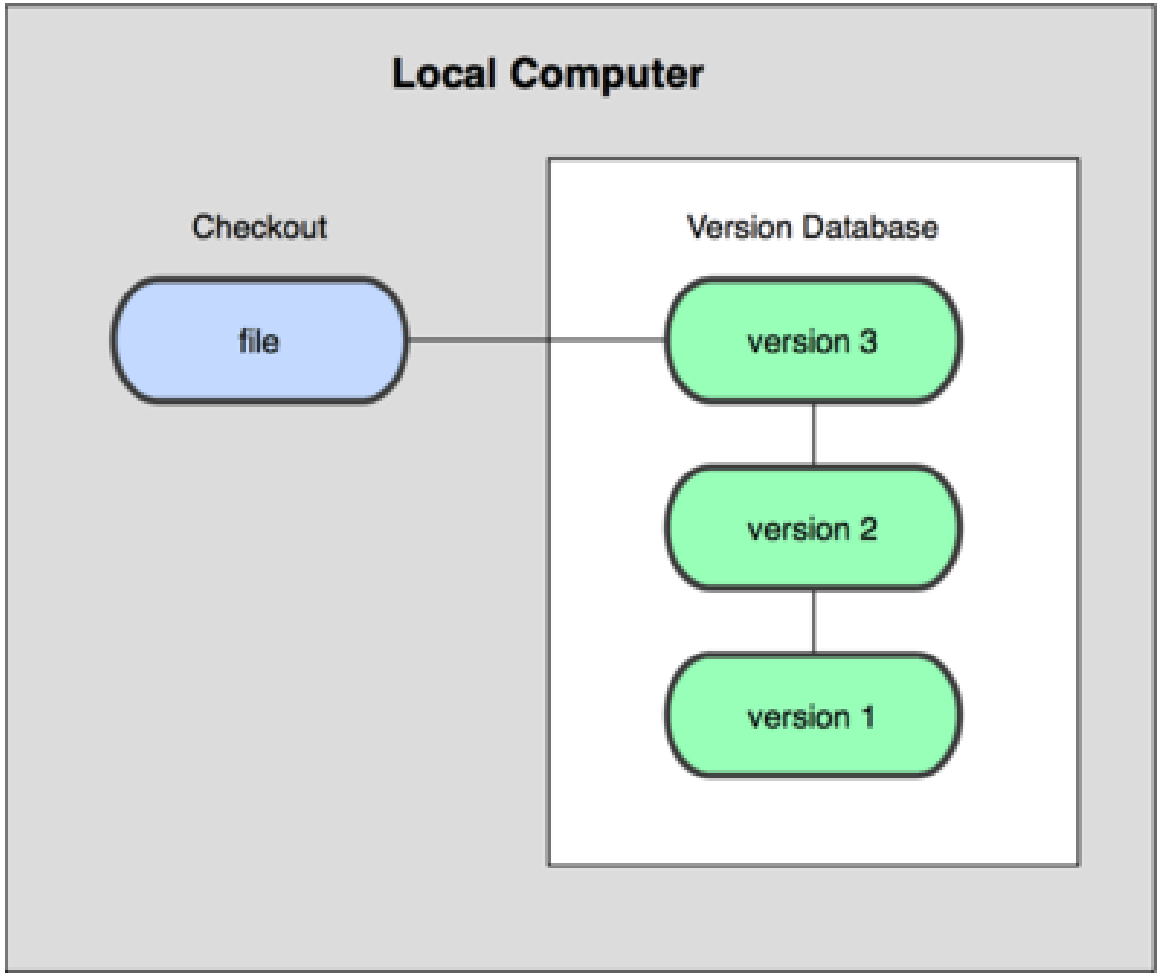
1.1 About Version Control

- Local version control systems – Hệ thống quản lý phiên bản cục bộ
- Vấn đề
- Nhiều người chọn phương pháp quản lý phiên bản bằng cách
  - Copy file tập tin đang làm sang một thư mục khác
  - Đặt tên các tập tin, thư mục theo thời gian
- Đây là phương pháp phổ biến cho người không biết kĩ thuật
  - Dễ phát sinh lỗi
  - Copy tập tin qua lại quá nhiều lần, tốn bộ nhớ
  - Chép, xóa nhầm file không mong muốn



1.1 About Version Control

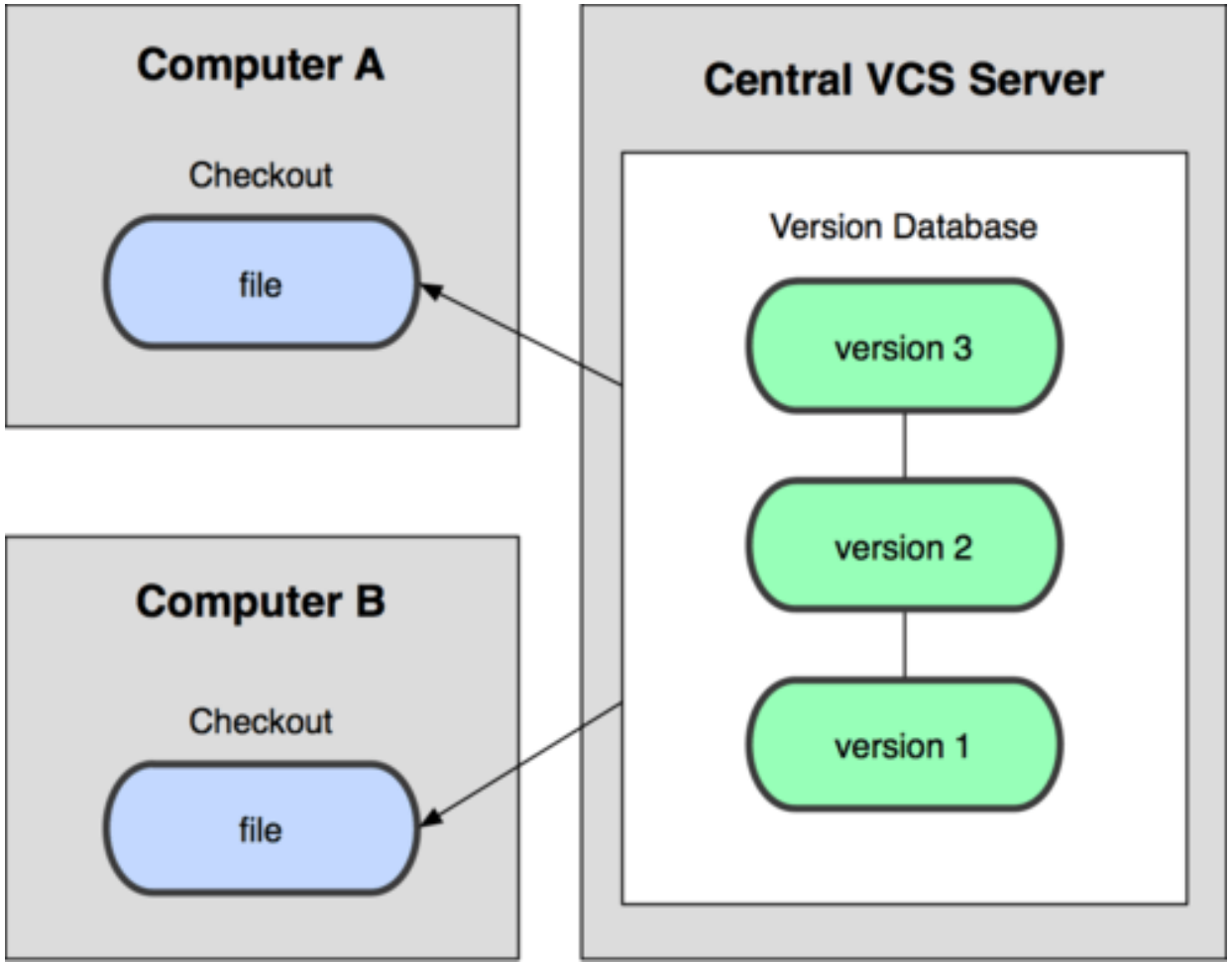
- Để giải quyết vấn đề này. VCS ra đời và có chức năng như một database đơn giản cho phép lưu trữ tất cả các sự thay đổi của files và kiểm soát theo theo phiên bản – version



Hình 1.1 Mô hình quản lý phiên bản cục bộ (Local version control systems)

1.1 About Version Control

- Vấn đề: Không thể tương tác với các thành viên khác.
- Phương pháp: Hệ thống quản lý phiên bản tập trung – Centralized vesion control system – CVCSs ra đời như một máy chủ, bao gồm các tập tin đã được phiên bản hóa (versioned) và các máy khác có quyền thay đổi các tập tin đó

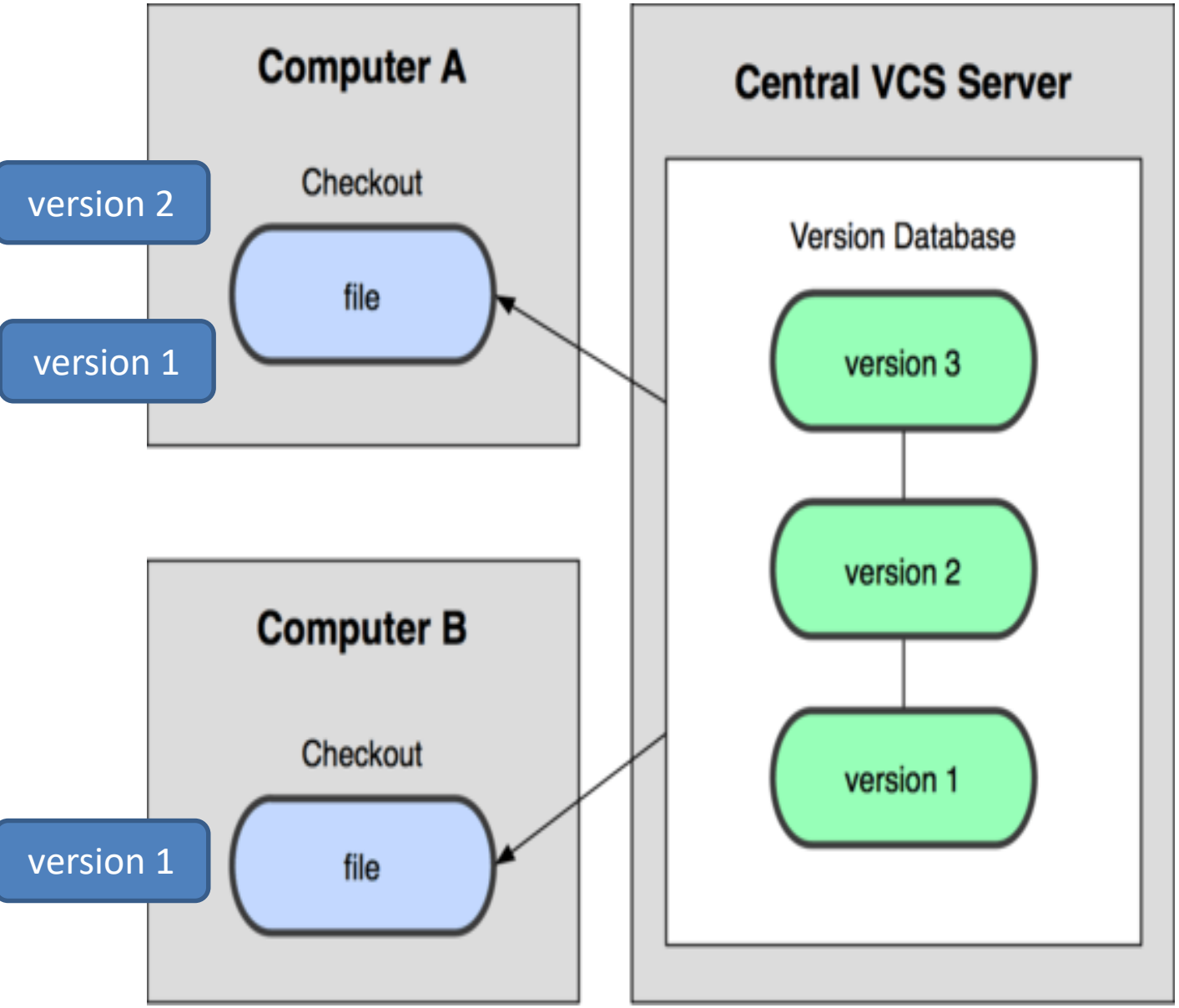


Hình 1.2 Mô hình quản lý phiên bản tập trung (Centralized version control systems)

Lợi ích của mô hình:

- Người dùng biết được một phần (version) nào đó những việc mà người khác đang làm trong dự án
- Người quản lý có quyền quản lý ai có thể làm gì theo ý muốn

1.1 About Version Control



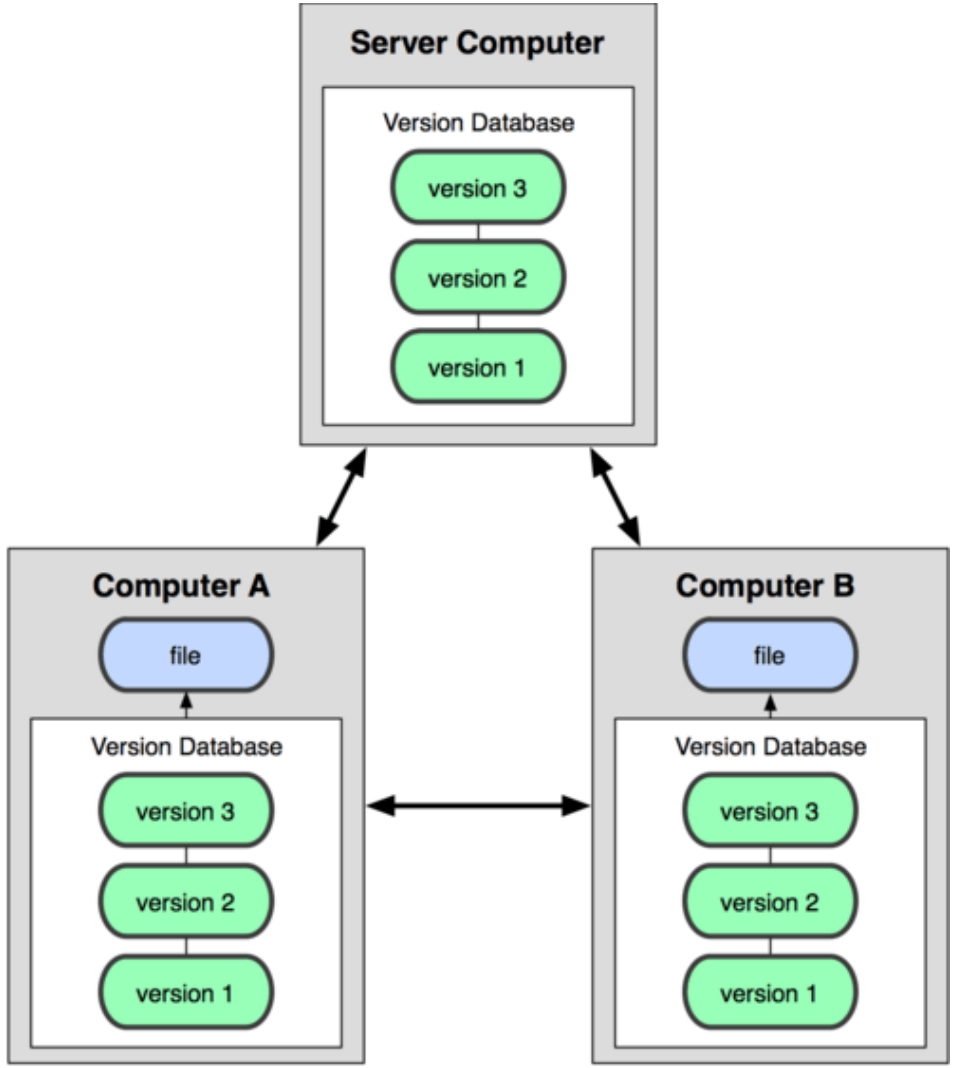
Hình 1.2 Mô hình quản lý phiên bản tập trung (Centralized version control systems)

Hạn chế:

- Nếu máy chủ không hoạt động trong một thời gian. Dẫn đến cộng tác viên không thể tương tác với nhau
- Nếu ổ cứng lưu trữ dữ liệu trung tâm bị hỏng, bạn sẽ mất toàn bộ lịch sử của dự án ngoại trừ những phiên bản cục bộ đã có trên máy cục bộ

1.1 About Version Control

- Hệ thống quản lý phiên bản phân tán – Distributed vesion control systems
- Trong các DVCS (Git, Mercirial, Darcs), các máy khác không chỉ "checkout" (sao chép versions về máy cục bộ) phiên bản mới nhất của các tập tin mà còn sao chép toàn bộ (clone) kho chứa



hể dụng máy khách sao chép ngược trở lại máy chủ

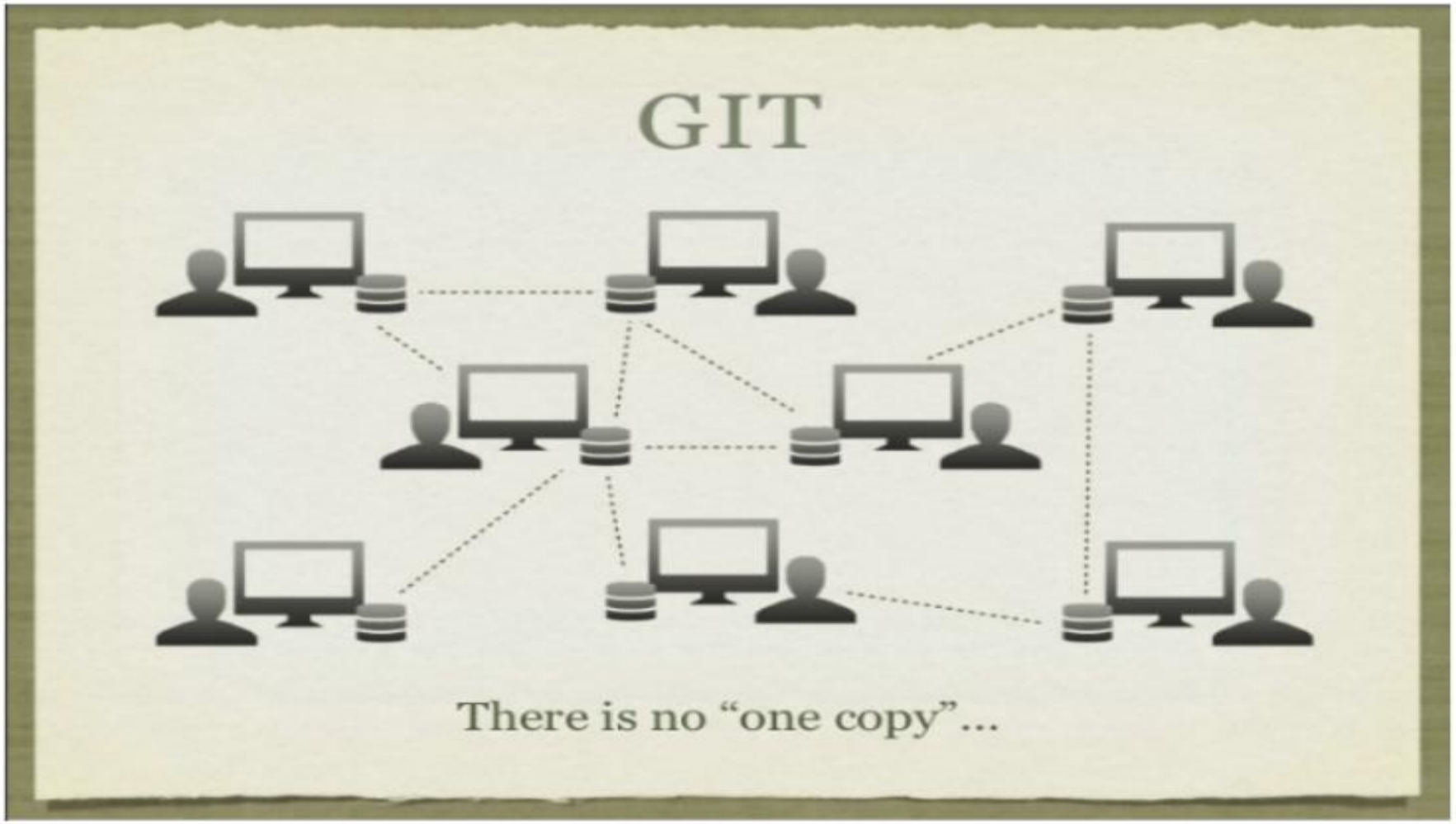
Hình 1.3 Mô hình quản lý phiên bản phân tán (Distributed version control systems)

Lợi ích của mô hình:

- Mỗi checkout thực sự là một bản sao đầy đủ của tất cả dữ liệu
- Dễ dàng khôi phục nếu máy chủ trung tâm có vấn đề
- Mỗi version, file đơn giản chỉ là một mã hash được băm từ nội dung các tập tin

1.1 About Version Control

all repository are created equal !!!

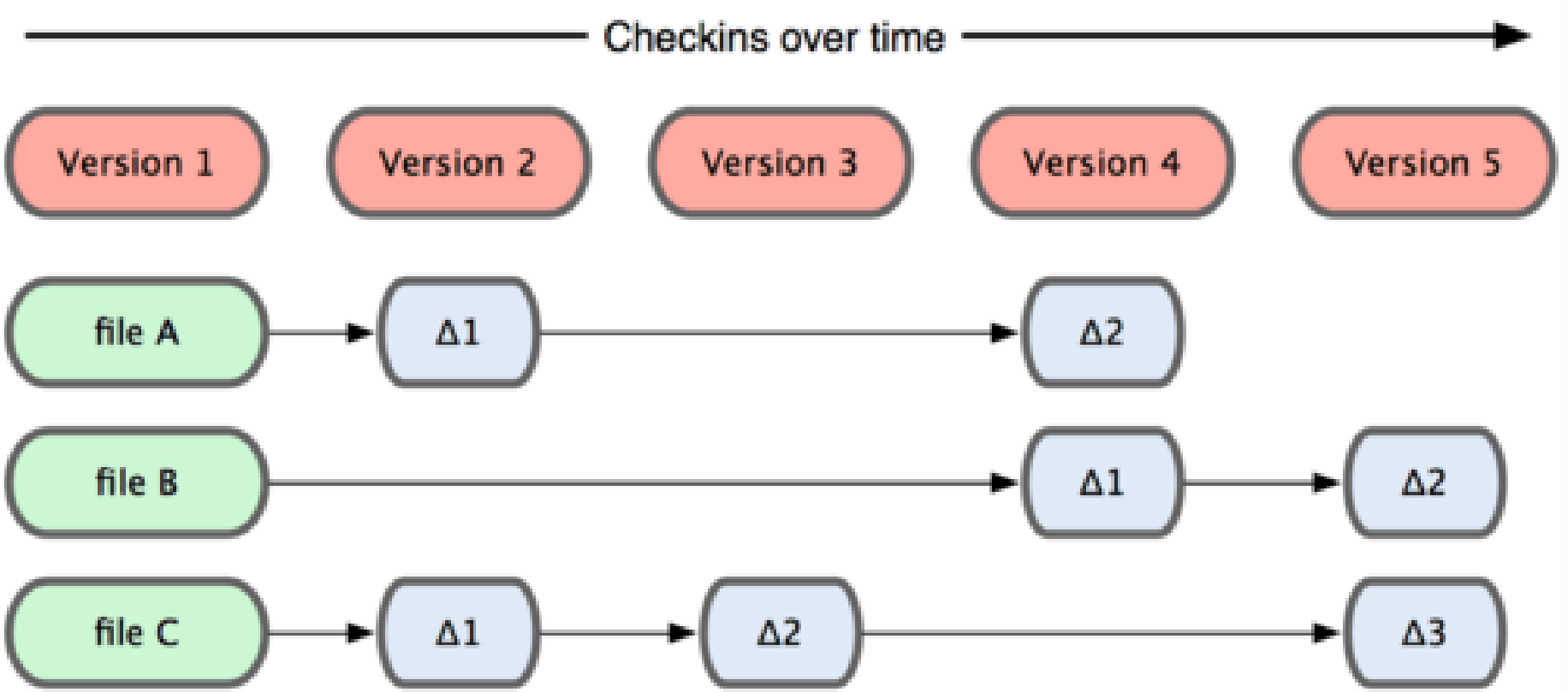


1.2 Short history of Git

- Trở về lịch sử phát triển Linux kernel
- Nhân Linux là dự án phần mềm mã nguồn mở trong phạm vi khá lớn
- Trong phần lớn thời gian bảo trì của nhân Linux (1991 - 2002), các thay đổi của phần mềm được truyền đi dưới dạng các bản vá và các tập tin lưu trữ. Vào năm 2002, dự án nhân Linux bắt đầu sử dụng một DVCS độc quyền có tên là BitKeeper
- Năm 2005, sự **hợp tác** giữa cộng đồng phát triển nhân **Linux** và công ty thương mại phát triển **BitKeeper** bị **phá vỡ**, và công cụ đó không còn được cung cấp miễn phí nữa
- Chính điều này đã thúc đẩy cộng đồng phát triển Linux (chính xác là Linus Torvalds, người sáng lập ra Linux) **phát triển công cụ của riêng** họ dựa trên kinh nghiệm sử dụng BitKeeper
- Mục tiêu
  - Nhanh
  - Thiết kế đơn giản
  - Hỗ trợ phân tán và xử lý song song
  - Có khả năng xử lý các dự án lớn giống như Linux về mặt hiệu quả

1.3 What is Git

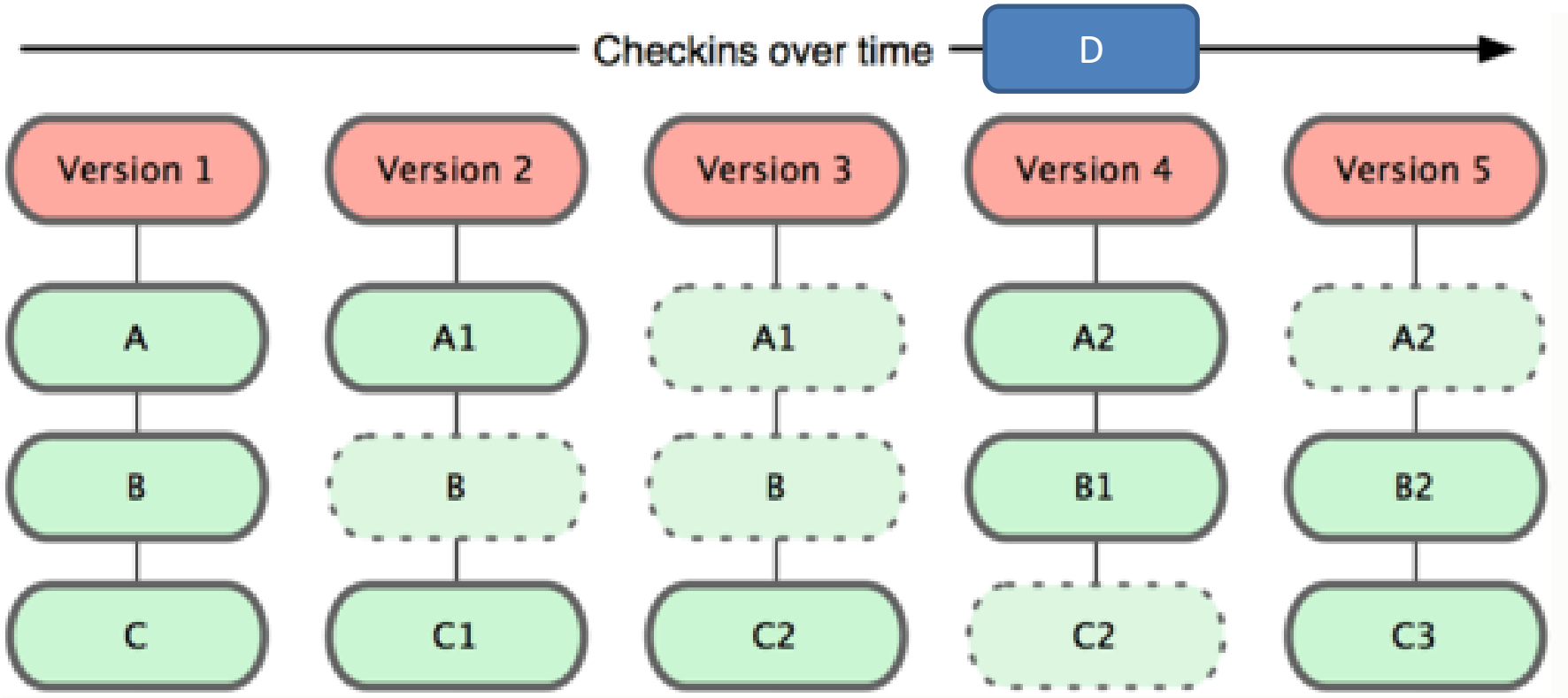
- Snapshots, No differences - Ảnh chụp, không phải sự khác biệt
- Điểm khác biệt chính giữa Git và các VCS khác là cách Git quản lý - nghĩ về dữ liệu
- Về mặt lý thuyết, phần lớn **các hệ thống khác lưu trữ thông tin dưới dạng danh sách các tập tin được thay đổi.**
- Các hệ thống (CVS, Subversion, Perforce) xem thông tin được lưu trữ là một tập hợp các tập tin và các thay đổi được thực hiện trên mỗi tập tin theo thời gian



Hình 3.1 Other VCS(s)

1.3 What is Git

- Git xem dữ liệu của nó như một tập hợp các ảnh (snapshot) của một hệ thống tập tin
- **Mỗi lần commit (git tạo version mới) để lưu lại trạng thái hiện tại của dự án trong Git.**
- **Git chụp một bức ảnh (tạo bản sao) ghi lại nội dung tất cả các tập tin tại thời điểm đó và tạo một tham chiếu (hashing) đến ảnh đó**
- Để hiệu quả hơn, nếu như tập tin không có sự thay đổi nào, Git không lưu trữ lại tập tin đó mà chỉ tạo liên kết tới tập tin gốc tồn tại trước đó



Commit = Version (GIT)

Hình 3.2 Git lưu trữ dữ liệu dưới dạng ảnh chụp của dự án theo thời gian

```
commit 668fb23d5bb03c4b47138730972f866f88204bcf
Author: Quyen Phan <Quyen.Phan@mgm-tp.com>
Date:   Wed Jul 21 23:02:20 2021 +0700

    VERSION 3

commit c161609a172620bb9c179de3cc622e342bc17ed7
Author: Quyen Phan <Quyen.Phan@mgm-tp.com>
Date:   Wed Jul 21 22:44:25 2021 +0700

    VERSION 2

commit 3d0a6ed98404906926e996c9fa2a169f494a5f90
Author: Quyen Phan <Quyen.Phan@mgm-tp.com>
Date:   Wed Jul 21 22:24:30 2021 +0700

    VERSION 1
```

1.3 What is Git

- **Phần lớn các thao tác diễn ra cục bộ**
- Toàn bộ dự án đều nằm trên ổ cứng của bạn, các thao tác được thực hiện gần như lập tức
- Giả sử bạn muốn xem lịch sử dự án, Git không cần phải lấy thông tin đó từ máy chủ khác mà đơn giản nó được đọc trực tiếp từ chính cơ sở dữ liệu cục bộ (máy của bạn)
- Nếu muốn so sánh sự thay đổi phiên bản hiện tại với phiên bản trước đó. Git chỉ truy xuất file ở phiên bản trước và hiện tại rồi so sánh. Không cần phải pull từ máy chủ trung tâm về.
- Ngay cả khi bạn không có internet hoặc VPN bị ngắt. Bạn vẫn có thể thực hiện mọi thứ ở local. Sau khi hoàn thành mọi thứ chỉ cần connect đến internet rồi đẩy code lên máy chủ

1.3 What is Git

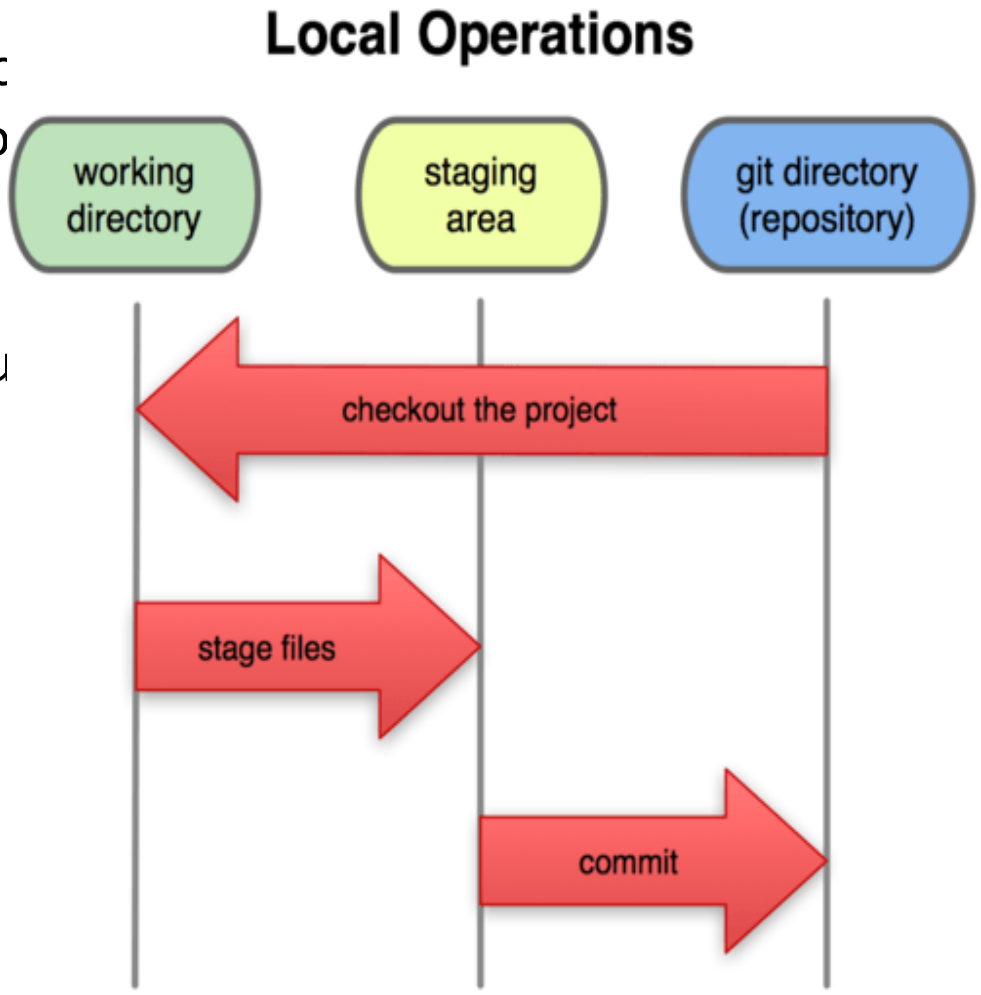
- **Git mang tính toàn vẹn**
- Mọi thứ trong Git được băm (checksum, hash) trước khi lưu trữ và được tham chiếu tới bằng mã băm đó.
- Có nghĩa là việc thay đổi nội dung của một tập tin hay một thư mục mà Git không biết là điều không thể
- Cơ chế Git sử dụng cho việc băm này là mã băm SHA-1 là một chuỗi gồm 40 ký tự của hệ cơ số 16 (0-9 và a-f) được tính toán dựa trên nôi dung của tập tin hoặc cấu trúc thư mục trong Git
- Định dạng: 24b9da6552252987aa493b52f8696cd6d3b00373
- Bạn sẽ thấy mã băm mọi nơi khi sử dụng Git.
- Trên thực tế Git không sử dụng tên của các tập tin để lưu trữ mà bằng các mã băm từ nội dung của tập tin và một cơ sở dữ liệu có thể truy vấn được

1.3 What is Git

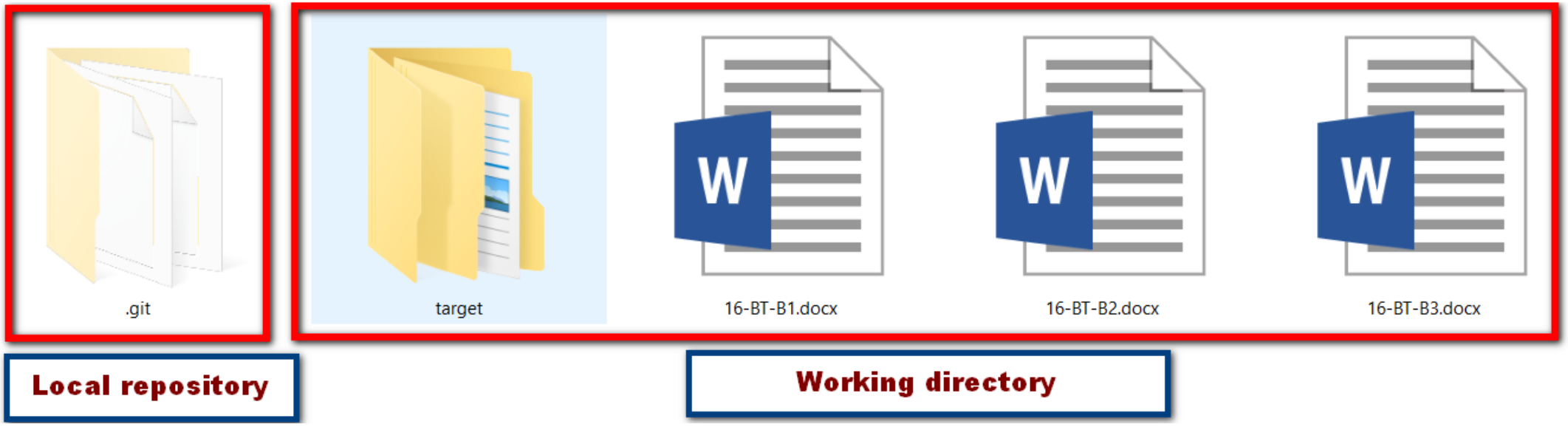
- **Git chỉ thêm mới dữ liệu**
- Khi bạn thực hiện các hành động trong Git, hầu hết các hành đó đều được thêm vào cơ sở dữ liệu của Git.
- Nếu dữ liệu đã được đưa vào cơ sở dữ liệu của Git (commit – version) thì dữ liệu sẽ rất khó bị mất, đặc biệt là bạn thuyên xuyên đẩy (push) cơ sở dữ liệu sang một kho chứa khác

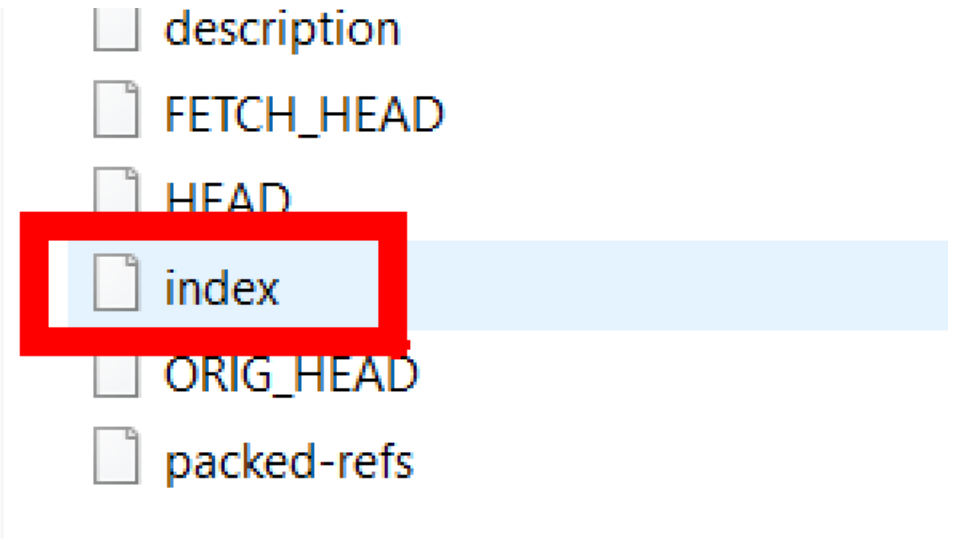
1.3 What is Git

- **Thành phần chính của thư mục được quản lý bởi GIT**
- **Git repository:** Là cơ sở dữ liệu của GIT. Chứa tất cả các phiên bản đã được tạo ra trong dự án. Có 2 cách để tạo ra git repo(.git)
  1. GIT INIT
  2. GIT CLONE từ remote repository (github, gitlab, bitbu
- **Working directory:** Chứa các tập tin, thư mục của một phiên bản, version bất kỳ trong dự án - HEAD
  Chỉ cần bạn chuyển (checkout) sang version, branch khác thì (working directory) bản sao hiện tại lập tức bị thay đổi
- **Staging area:** Là một tập tin đơn giản chứa trong thư mục Git, nó chứa thông tin những gì sẽ được commit trong lần commit sắp tới. Nó còn được biết đến với tên **index** (chỉ mục)



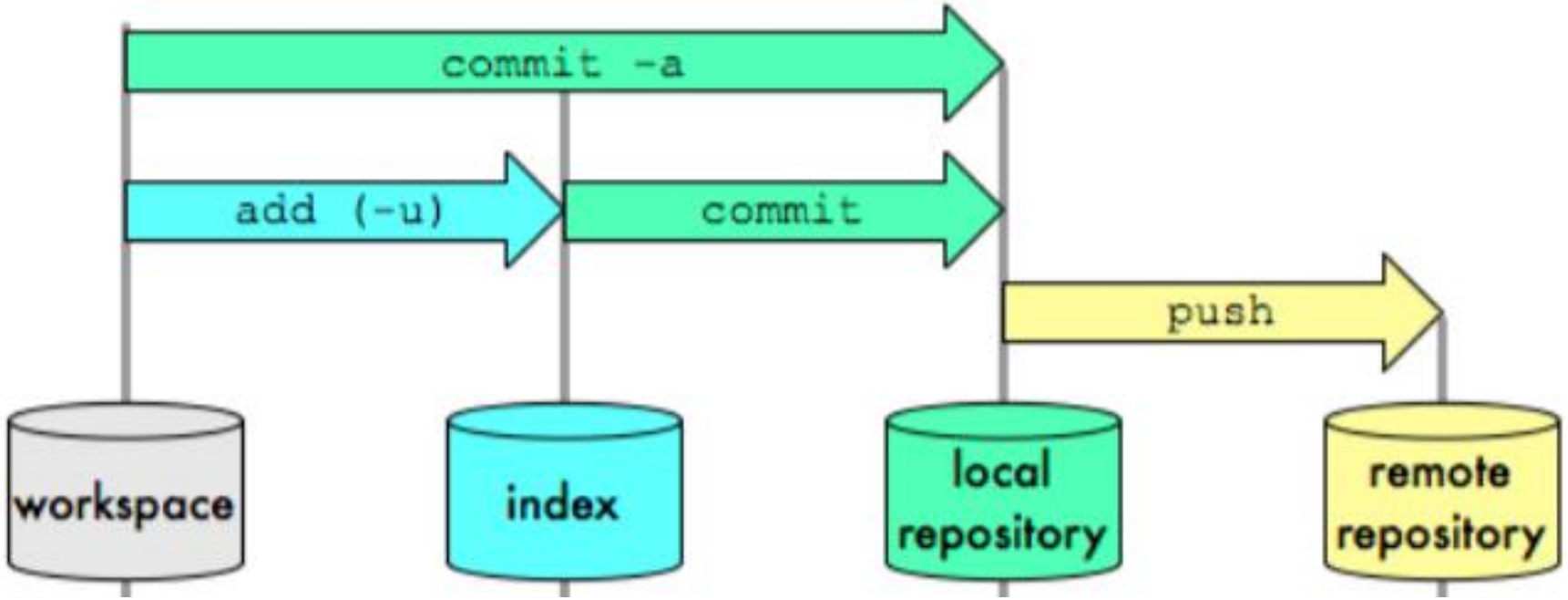
1.3 What is Git





1.3 What is Git

- **Tiến trình công việc (workflow) cơ bản của Git**
- Thay đổi các tập tin trong thư mục làm việc
- Chọn những tập tin bạn muốn commit trong lần commit tiếp theo. Đưa những tập tin này vào staging area – $ git add
- Thực hiện commit để đưa những tập tin đang ở staging area vào cơ sở dữ liệu của Git. Những tập tin đã được lưu trữ vào thư mục Git – git commit



1.4 Installing Git

- https://git-scm.com/download

1.5 First-Time Git Setup

- Sau khi đã hoàn thành cài đặt Git. Chúng ta có thể tùy biến một số lựa chọn cho môi trường Git trên máy cá nhân
- Các biến cấu hình được lưu trữ ở ba vị trí sau:
  - /etc/gitconfig: Chứa giá trị cho tất cả người dùng và kho chứa – repository. Nếu bạn sử dụng –system khi chạy git config. Thao tác đọc và ghi sẽ được thực hiện trên tệp tin này
  - ~/.gitconfig: Riêng biệt cho tài khoản của bạn. Bạn có thể chỉ định Git đọc và ghi trên tệp tin này bằng –global
  - .git/config: Chỉ áp dụng riêng cho mỗi kho chứa – repository. Mỗi cấp sẽ ghi đè các cấp trước đó với /etc/gitconfig và .gitconfig
- Thông tin trên windows
  - Thư mục /etc/gitconfig: C:\Program Files\Git\mingw64\etc\.gitconfig
  - Thư mục .gitconfig: C:\Users\@User\.gitconfig
  - Thư mục .git/config: $ git init - $ ls –a
- Tập lệnh
  - $ git config --list
  - **$ git config --global –edit**
  - $ git config --system -edit
  - $ git config --local -edit
- Thay đổi cấu hình: git config --[option] user.name "your_name"

1.5 First-Time Git Setup

ID	Subject	Owner	Project	Branch	Updated	CR	V
I7f76dfc1	The zoneinfo stuff is no longer legacy.	Elliott Hughes	platform/build	master	9:36 PM		✓
Iec5ce7f1	[MIPS] Append private_to AES_set_encrypt_key and AES_set_decrypt_key for ...	Raghu Gandham	platform/external/openssl	master	9:04 PM		✓
Ia343c263	ADT: Detect when SDK platforms/addons might have changed.	Raphaël Moll	platform/sdk	master	9:01 PM	✓	✓
I5f35b32e	linker: Fix relocation of R_ARM_COPY type relocations.	Robin Burchell	platform/bionic	master	8:47 PM		✘
I8e2cb3fd	getopt(): fix missing carriage return on bad parameters	Tanguy Pruvot	platform/bionic	master	8:46 PM	-1	✓
Icf5eff3b	libicuuc: Fix memory corruption in libicuuc	Shuo Gao	platform/external/icu4c	master	8:15 PM	✘	✓
Iea1f1912	Do better comparisons for ENGINE-based keys	Kenny Root	platform/libcore	master	7:46 PM	✓	✓
I3f29b54f	RIL: Add message type to RIL_UNSOL_RESPONSE_NEW_BROADCAST_SMS	Rika Brooks	platform/hardware/ril	master	7:24 PM	-1	✓
I87884148	Added a persistent feature in WiFi Direct.	Yoshihiko Ikenaga	platform/frameworks/base	jb-dev	7:13 PM		✘
I90bc5e1b	Export flags needed for ifc_reset_connections API in netutils	Alex Yakavenka	platform/system/core	master	6:52 PM	+1	✓
I0b3a2b4a	Fix live wallpaper "Grass" display issue	CATHERINE LIU	platform/packages/wallpapers/Basic	master	6:39 PM		
Ie7fc097b	Fix usage DeflaterOutputStream constructor on android	Narayan Kamath	platform/external/okhttp	master	6:32 PM	✓	
I82cead37	Fix issue of capslock+shift'a-z' keys	CATHERINE LIU	platform/frameworks/base	master	6:17 PM		✓
I67fd10ce	ContentProviderRecord: Remove external process reference when finalized	Vairavan Srinivasan	platform/frameworks/base	master	6:17 PM		✓
I412c7479	decrease the net log frequency by cache the read length	Shuo Gao	platform/external/chromium	master	5:55 PM	-1	
I49e91460	Set net_log to NULL unless debugging.	Kristian Monsen	platform/frameworks/av	master	5:49 PM		✓
Ifd96cb4c	Returning early if net_log is NULL	Kristian Monsen	platform/external/chromium	master	5:49 PM		✓

1.6 Getting Help

- $ git help [-a|-g]
- $ git --version
- $ git help config
- $ git help add

- Thư mục
- C:\Program Files\Git\mingw64\share\doc\git-doc

2.1 Getting a Git Repository

- **Getting a Git Repository – Tạo một kho chứa Git**
- Tạo một dự án sử dụng Git dựa theo 3 phương pháp chính
  - Tạo mới một dự án từ đầu và sử dụng Git
  - Sử dụng một dự án, thư mục đã có sẵn rồi import vào Git để quản lý
  - Tạo bản sao một kho chứa Git đang hoạt động trên một máy chủ khác

2.1 Getting a Git Repository

- **Cách 1: Khởi tạo một kho chứa – repository từ thư mục có sẵn**
- $ git init
- .git: Chứa tất cả các tập tin cần thiết cho một kho chứa – repository:
- Đến thời điểm hiện tại, vẫn chưa có gì trong dự án hiện tại được theo dõi bởi Git
- Nếu muốn kiểm soát phiên bản cho các tập tin có sẵn của dự án. Bạn phải
  - Di chuyển các tập tin vào staging area - $ git add .
  - Di chuyển các tập tin vào git repository - $ git comit –m “Khởi tạo dự án-Phiên bản đầu tiên”

2.1 Getting a Git Repository

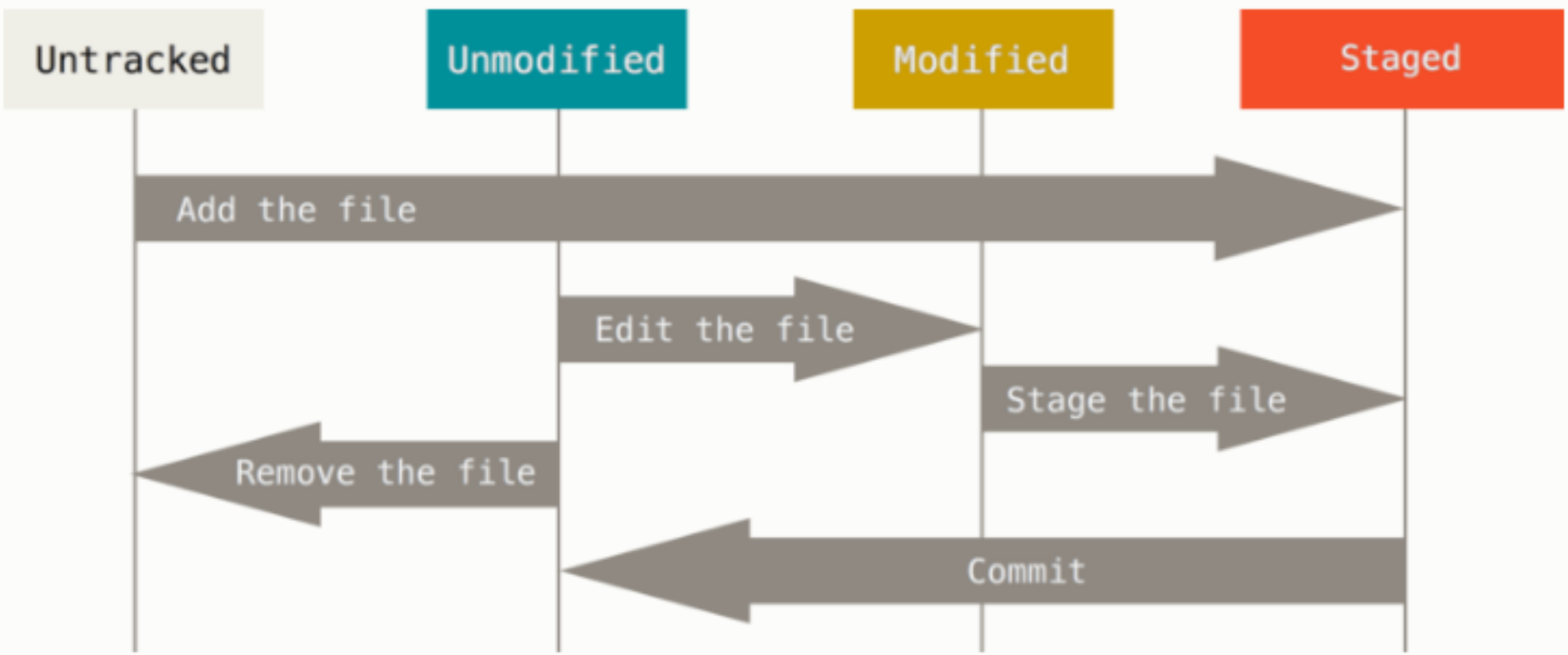
- **Cách 2: Sao chép (clone-tạo bản sao) từ một kho chứa đã tồn tại**
- Nếu bạn muốn có một bản sao của một kho chứa Git có sẵn: $ git clone [url]
- Nếu như bạn đã quen thuộc với các hệ thống VCS khác như là Subversion, bạn sẽ nhận ra câu lệnh tạo bản sao là $ git clone chứ không phải $ git checkout
- $ git clone: Tạo một bản sao gần như tất cả các dữ liệu mà máy chủ đang có và lịch sử nội dung của mỗi tập tin
- Tập lệnh
- $ git clone https://github.com/vinabkteam/resolution_2019
- Một thư mục mới có tên resolution_2019 sẽ được tạo kèm theo thư mục .git và bản sao mới nhất của tất cả dữ liệu trong kho chứa, và đã sẵn sàng cho bạn làm việc và sử dụng
- Nếu bạn muốn tạo một bản sao của resolution_2019 nhưng muốn chỉ định tên thư mục khác: $ git clone https://github.com/vinabkteam/resolution_2019 reso

2.2 Recording changes to the repository

- **Recording changes to the repo – Ghi lại những thay đổi vào kho chứa**
- Bây giờ bạn đã có một kho chứa Git thật sự và một bản sao dữ liệu của dự án làm việc
- Bạn cần thực hiện một số thay đổi và commit snapshots của chúng vào kho chứa mỗi lần dự án đạt đến một trạng thái nào đó mà bạn muốn ghi lại
- Mỗi tập tin trong thư mục làm việc có thể ở 1 trong 2 trạng thái: Tracked & Untracked
- Tracked: Là các tập tin đã từng được commit, đã có mặt trong ảnh (snapshot) trước chúng có thể là unmodified, modified, staged
- Untracked: Là các tập tin chưa từng được commit hoặc trong index (staging area)

2.2 Recording changes to the repository

- Khi bạn $ git init – tất cả các file đang ở trạng thái untracked
- Khi bạn $ git clone – tất cả các phải ở trạng thái tracked và unmodified – chưa thay đổi
- Khi bạn tạo mới hoặc thay đổi nội dung tập tin, Git xem chúng đã được tạo mới hoặc được thay đổi. Bạn cần stage các tập tin và commit tất cả các thay đổi đã được staged (tổ chức) đó.



2.2 Recording changes to the repository

- **Kiểm tra trạng thái của tập tin:** $ git status
- Nếu như bạn chạy lệnh này sau khi tạo bản sao xong - $ git clone
- Thông tin hiển thị
  - # On branch master
  - Nothing to commit, working directory clean
- Thư mục làm việc đang sạch, không có tập tin đang theo dõi nào bị thay đổi. Và cũng không có tập tin nào chưa theo dõi

2.2 Recording changes to the repository

- **Theo dõi – Hủy theo dõi các tập tin mới – Tracking – Move to staging area (tracked)**
- $ git add filename
- Bỏ theo dõi tập tin mới – Untracking – Move to working directory (Unttracked)
- $ git reset HEAD <file> - README.md
- $ git restore –staged <file> - README.md

2.2 Recording changes to the repository

- **Quản lý các tập tin đã thay đổi**
- Thay đổi nội dung tập tin đã từng được commit – Git_Note.txt
- $ git status
- Lưu – Ghi lại nội dung đã thay đổi của Git_Note.txt
- $ git commit –m “message”

- Bỏ qua những sự thay đổi trong Git_Note.txt, trở về trạng thái ban đầu
- $ git restore Git_Note.txt

2.2 Recording changes to the repository

- **Quản lý các tập tin đã thay đổi**
- **Vấn đề**
- Tạo mới và staging README.md $ git add README.md
- Thay đổi nội dung tập tin đã từng được commit – Git_Note.txt
- $ git add .
- Trạng thái: Cả 2 files này sẽ có mặt trong lần commit tiếp theo
- Tiếp tục thay đổi Git_Note.txt
- $ git status

```
Administrator@QPhan MINGW64 ~/Desktop/git (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   Git_Note.txt
        new file:   README.md

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   Git_Note.txt
```

2.2 Recording changes to the repository

```
Administrator@QPhan MINGW64 ~/Desktop/git (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   Git_Note.txt
        new file:   README.md

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   Git_Note.txt
```

- Lúc này Git_Note.txt vừa là staged và là unstaged
- Nếu thực hiện commit thì sẽ commit nội dung gì trong Git_Note.txt

- Git tổ chức các tập tin lúc người dùng chạy $ git add
- Nếu bây giờ bạn commit. Phiên bản lúc bạn thực hiện $ git add Git_Note.txt sẽ được commit chứ không phải phiên bản hiện tại của Git_Note.txt
- Lưu ý: Nếu như bạn chỉnh sửa tập tin sau khi $ git add, bạn phải chạy $ git add lại một lần nữa để đưa nó vào staging mới nhất

2.2 Recording changes to the repository

- **Ignoring files - Bỏ qua các tập tin**
- Thường sẽ có một số loại tập tin mà bạn không mong muốn Git tự động thêm nó vào hoặc thậm chỉ là hiển thị không được theo dõi
- Những tập tin này thường được ra ra tự động như các tập tin (log files) hay các tập tin sinh ra khi biên dịch chương trình (.class). Trong trường hợp này bạn có thể tạo ra một tập tin liệt kê các pattern để bỏ qua những tập tin này (ko theo dõi bởi Git) (.gitignore)
- Tạo tập tin: $ touch .gitignore và một số quy tắc
- # comment
- / chỉ định thư mục
- ! phủ định

2.2 Recording changes to the repository

- **Ignoring files - Bỏ qua các tập tin**
- Một số ví dụ về .gitignore
- # a comment - dòng này được bỏ qua
- # không theo dõi tập tin có đuôi .a

  ***.a**
- # nhưng theo dõi tập lib.a, mặc dù bạn đang bỏ qua tất cả tập tin .a ở trên !lib.a
- # chỉ bỏ qua tập TODO ở thư mục gốc, chứ không phải ở các thư mục con subdir/TODO

  /TODO
- # bỏ qua tất cả tập tin trong thư mục build/

  build/
- # bỏ qua doc/notes.txt, không phải doc/server/arch.txt

  doc/*.txt
- # bỏ qua tất cả tập .txt trong thư mục doc/

  doc//*.txt**

2.2 Recording changes to the repository

- **Xem các thay đổi Staged và Unstaged**
- $ git diff

2.2 Recording changes to the repository

- **Commit thay đổi**
- Bây giờ, sau khi đã tổ chức – staging các tập tin theo ý muốn. Bạn muốn commit các staged files, thực hiện $ git commit –m “message”
- Trước khi commit nên thực hiện $ git status để kiểm tra staged và unstaged files

- **Commit và Staging cùng lúc**
- $ git commit –a –m “message”

2.2 Recording changes to the repository

- **Xóa và khôi phục tập tin**
- $ rm hello.java
- $ git rm hello.java

- rm: Xóa tập tin khỏi working directory, “Changes not staged for commit” (RED)
- Xóa nhưng chưa stage
- Khôi phục: $ git restore hello.java to discard changes in working directory

- git rm: Xóa tập tin khỏi working directory, “Changes to be commited”
- Xóa nhưng đã stage, đã $ git add tự động
- Khôi phục: $ git restore –staged hello.java to unstage staged file

2.3 Viewing the Commit History

- **View the commit history – Xem lịch sử tạo version - commit**
- Sau khi bạn đã thực hiện rất nhiều commit, hoặc bạn đã sao chép một kho chứa với các commit có sẵn, bạn muốn xem lại những gì đã xảy ra trong dự án. Thực hiện
- $ git log
- $ git log --oneline

2.4 Undoing Things

- **Undoing Things – Phục hồi**
- Tại thời điểm nào đó bạn muốn phục hồi các thay đổi đã được thực hiện trong dự án
- **Thay đổi commit cuối cùng**
- Một trong những cách phục hồi phổ biến thường dùng khi bạn commit quá sớm/vội và có thể quên thêm một số tập tin hoặc message không như ý muốn
- Nếu bạn muốn thực hiện lại commit đó, chạy commit với tham số --amend [--no-edit]
- $ git commit –amend [--no-edit]

- Xóa commit hiện tại đồng thời giữ nguyên nội những tập tin đã thay đổi
- Đưa các tập tin vào trạng thái staged, tiếp tục thực hiện và commit
- $ git reset –soft HEAD~i

- Xóa commit hiện tại, đồng thời loại bỏ những thứ đã thay đổi
- **$ git reset –hard HEAD~I**

2.4 Undoing Things

- **Undoing Things – Phục hồi**
- Tại thời điểm nào đó bạn muốn phục hồi các thay đổi đã được thực hiện trong dự án
- **Revert changes**
- $ git revert badcommit

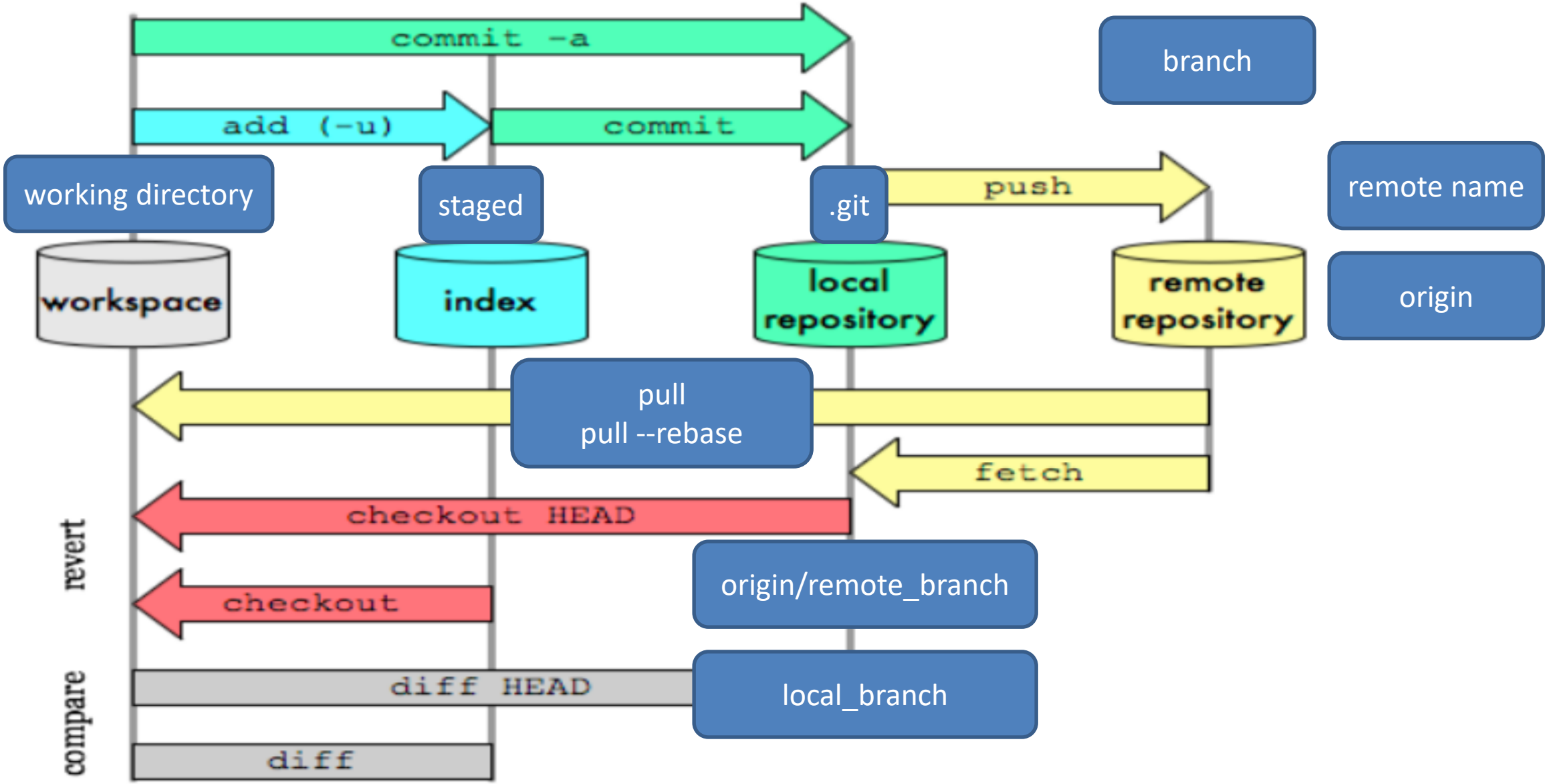
2.5 Working with Remotes

- Tạo tài khoản GitHub hoặc Bitbucket
- Cài đặt extension: https://chrome.google.com/webstore/search/octotree

- **Working with Remotes – Làm việc từ xa**
- Để có thể cùng cộng tác với các thành viên khác trên bất kỳ dự án nào sử dụng Git, bạn cần phải biết quản lý các kho chứa của bạn
- Các kho chứa từ xa – remote repository là các phiên bản của dự án của bạn được lưu trữ trên internet hoặc một mạng lưới nào đó
- Bạn có thể có nhiều kho chứa khác nhau, thường chỉ để đọc và ghi
- Công tác với nhiều thành viên khác trong cùng một dự án

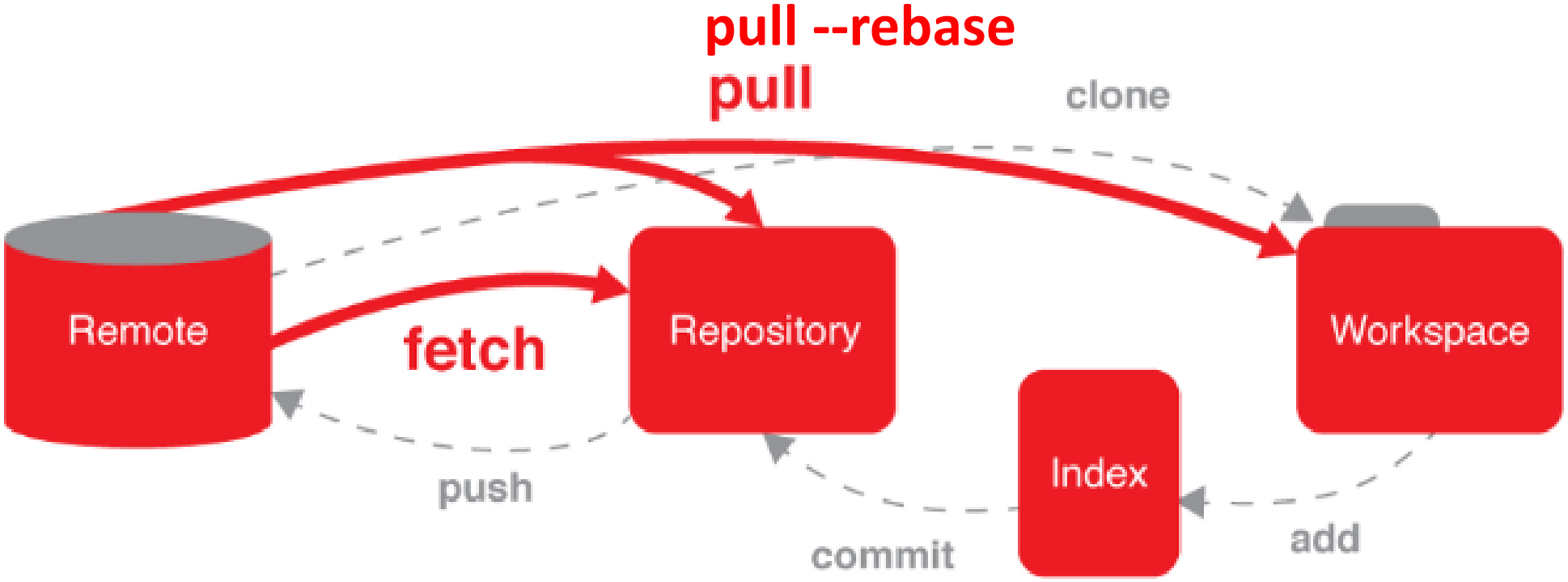
2.5 Working with Remotes

- Git process



2.5 Working with Remotes

- Git process



2.5 Working with Remotes

- Hiển thị máy chủ - remote repository của kho chứa cục bộ - local repository
- $ git init
- $ git remote –v

- $ git clone https://github.com/vinabkteam/resolution_2019/
- $ git remote –v

- Thêm các kho chứa từ xa – remote repository
- $ git remote add origin <URL> [<remote_name>]
- $ git remote -v

2.5 Working with Remotes

- **Thêm, Xóa các kho chứa từ xa – remote repository**
- Trường hợp A
  - Remote: Đang là một empty repository – chưa có bất kì commit nào
  - Local: Đang là một empty repository – chưa có bất kì commit nào
- Thực hiện
  - Local repository
  - Remote repository

- <remote_name> is a shorthand name for the remote repository that a project was originally cloned from.
- More precisely, it is used instead of that original repository's URL - and thereby makes referencing much easier.

2.5 Working with Remotes

- **Thêm, Xóa các kho chứa từ xa – remote repository**
- Trường hợp B
  - Remote: Đang là một empty repository – chưa có bất kì commit nào
  - Local: Đã có một số commits – branches
- Thực hiện
  - Local repository
  - Remote repository

- <remote_name> is a shorthand name for the remote repository that a project was originally cloned from.
- More precisely, it is used instead of that original repository's URL - and thereby makes referencing much easier.

2.5 Working with Remotes

- **Thêm, Xóa các kho chứa từ xa – remote repository**
- Trường hợp C
  - Remote: Đã có một số commits - branches
  - Local: : Đang là một empty repository – chưa có bất kì commit nào
- Thực hiện
  - Local repository
  - Remote repository

- <remote_name> is a shorthand name for the remote repository that a project was originally cloned from.
- More precisely, it is used instead of that original repository's URL - and thereby makes referencing much easier.

2.5 Working with Remotes

- **Thêm, Xóa các kho chứa từ xa – remote repository**
- Trường hợp D
  - Remote: Đã có một số commits - branches
  - Local: : Đã có một số commits - branches
- Thực hiện
  - Local repository
  - Remote repository

- <remote_name> is a shorthand name for the remote repository that a project was originally cloned from.
- More precisely, it is used instead of that original repository's URL - and thereby makes referencing much easier.

2.5 Working with Remotes

- **Xóa các kho chứa từ xa – remote repository**
- $ git remote rm <remote_name>
- $ git remote rm just remove the reference from Local Repository with Remote Repository. It does not remove the remote repository from server

2.5 Working with Remotes

- **Phân biệt Rebase, Merge khi sử dụng $ git pull [--rebase]**
- Trường hợp A
  - R: 1
  - L: 2 3
  - Pull Merge: Error (due to have no same parent/root)
  - Pull Rebase: Rewinding head to replay your work on top of it
    - Move HEAD of R to L, find the same root node then replay L on top of it
    - Local: 1 2 3
- Trường hợp B
  - R: 1 2
  - L: 3
  - Pull Merge: Error (due to have no same parent/root)
  - Pull Rebase: Rewinding head to replay your work on top of it
    - Local: 1 2 3

2.5 Working with Remotes

- **Phân biệt Rebase, Merge khi sử dụng $ git pull [--rebase]**
- Trường hợp C
  - R: 1 2
  - L: 1
  - Pull Merge: Fast-forward
    - Local 1 2
  - Pull Rebase: Fast-forward
    - Local: 1 2
- Trường hợp D
  - R: 1 2
  - L: 1 3
  - Pull Merge: Correct but incorrect. Version 3 (newest code) should on the top of Local after PULL
    - Local: 1 3 2 CM_MERGED(3 & 2)
  - Pull Rebase: Rewinding head to replay your work on top of it
    - Local: 1 2 3

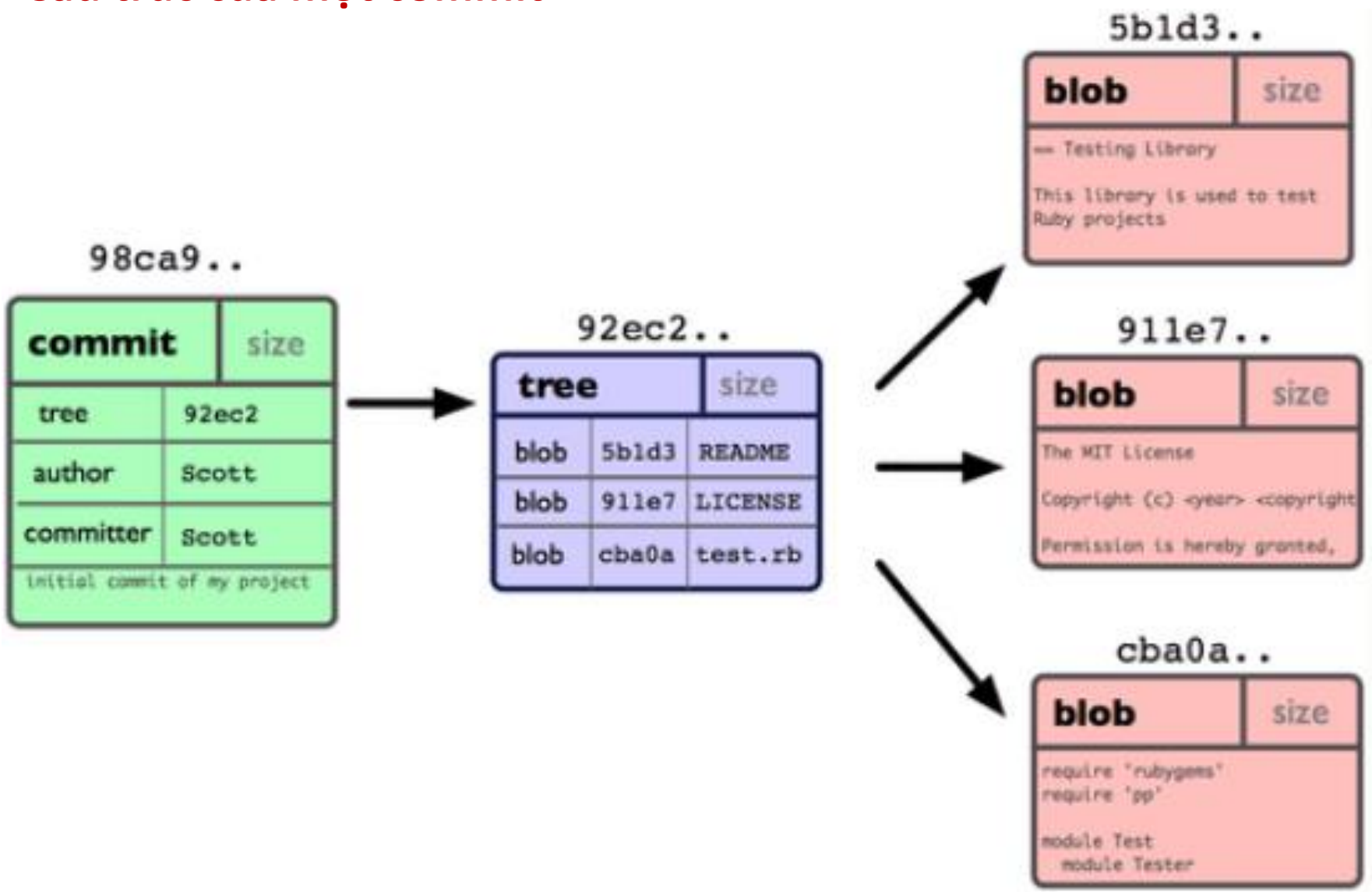
3.1 Branches in a Nutshell

- **Cách thức git lưu trữ dữ liệu**
- Khi bạn thực hiện commit, Git tạo đối tượng commit có chứa con trỏ, trỏ tớp snapshot của nội dung bạn đã tổ chức (staged)
- Commit đầu tiên không có cha, commit bình thường có một cha, và nhiều cha cho cùng một commit là kết quả tích hợp từ hai hoặc nhiều nhánh

- Giả sử:
  - B1: Tạo mới 3 files (README.md, test.rb, LICENSE)
  - B2. $ git add . $ git commit
- Lệnh $ git commit sẽ băm tất cả các tập tin trong dự án và lưu lại chúng dưới đối tượng tree.
- Sau đó git tạo một đối tượng commit có chứa thông tin mô tả (metadata) và một con trỏ trỏ đến đối tượng tree vì thế nó có thể tạo, trở về các ảnh trước khi cần thiết

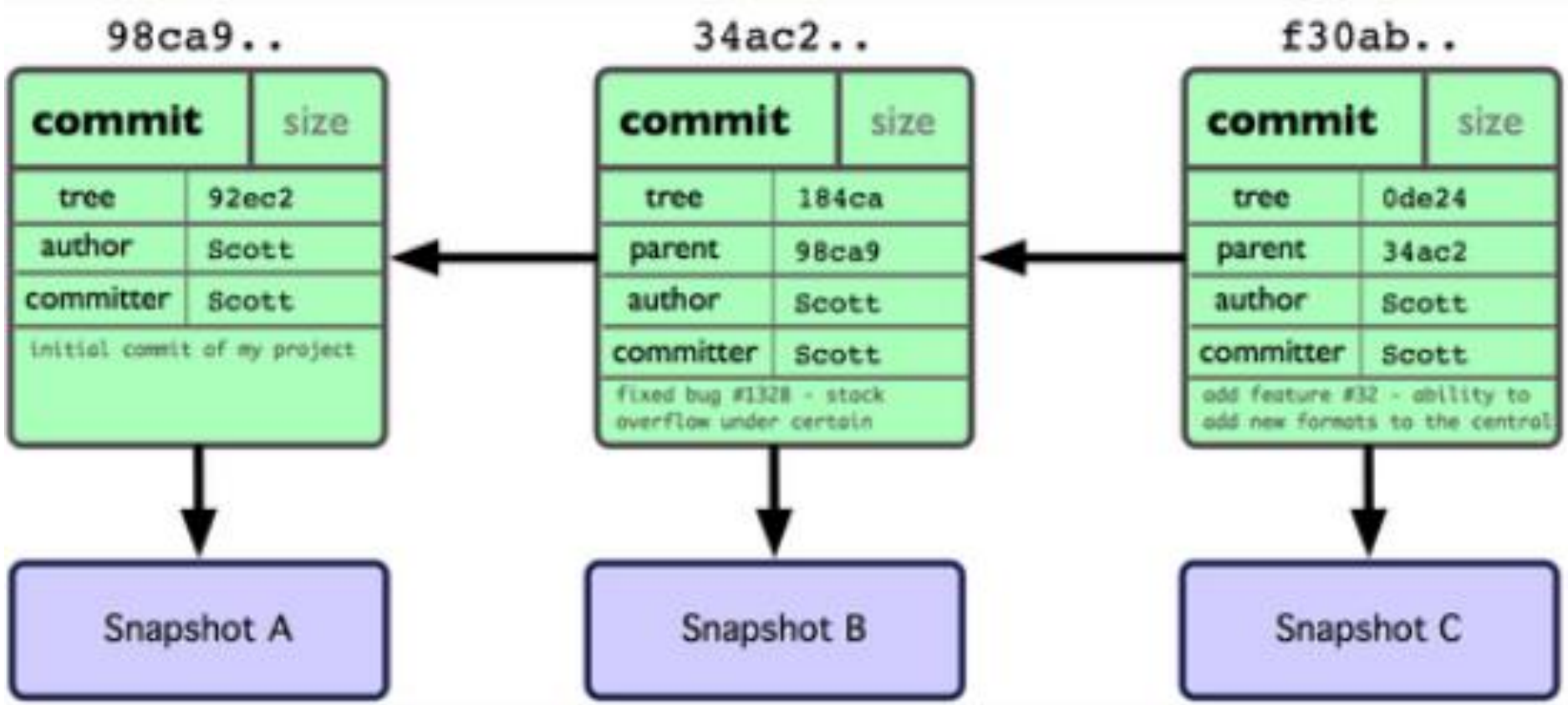
3.1 Branches in a Nutshell

- **Cấu trúc của một commit**



3.1 Branches in a Nutshell

- **Khi bạn thực hiện commit, git sẽ tạo ra một snapshot như thế này**



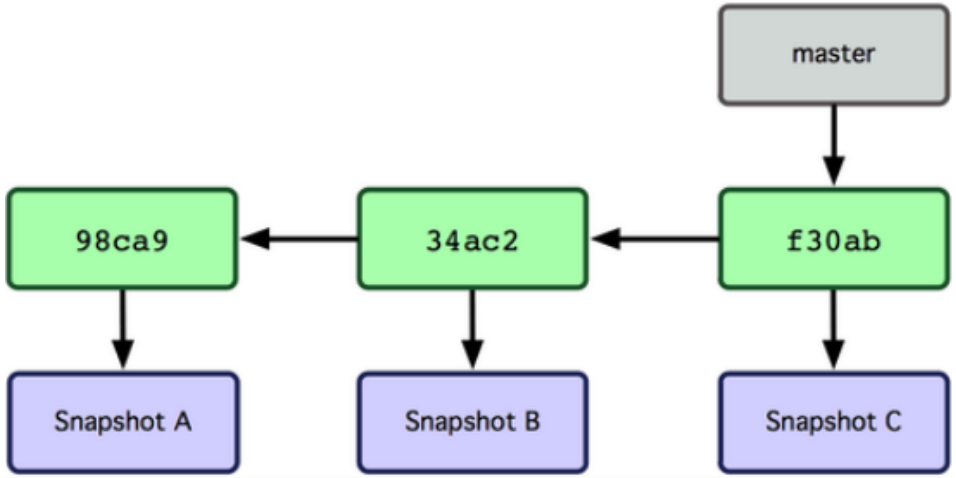
3.1 Branches in a Nutshell

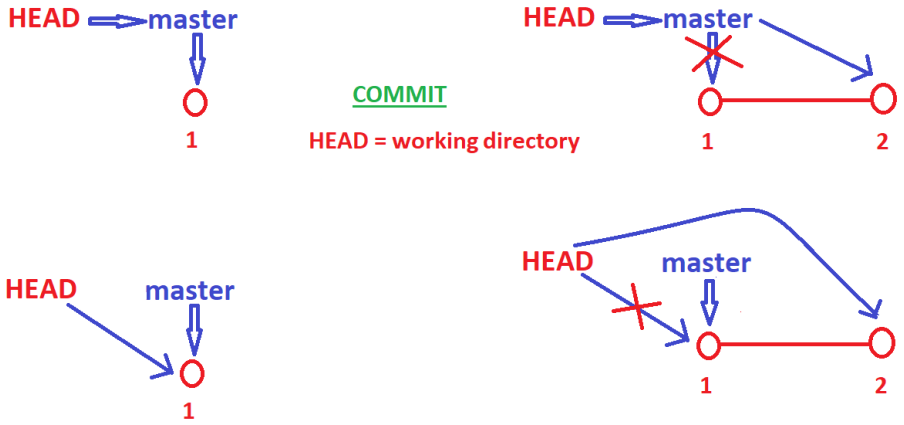
- **Tập lệnh để kiểm tra mã băm của các tập tin, commit**
- $ git hash-object <filename>
- $ git ls-files –s <filename>
- $ git ls-tree <branch-name>
- $ git ls-tree hash-commit

```
$ git ls-tree master
100644 blob 30a7e176c4de2ebc0d368ef75847314416e7f606    Ex01.txt
100644 blob e69de29bb2d1d6434b8b29ae775ad8c2e48c5391    Ex02.txt
```

3.1 Branches in a Nutshell

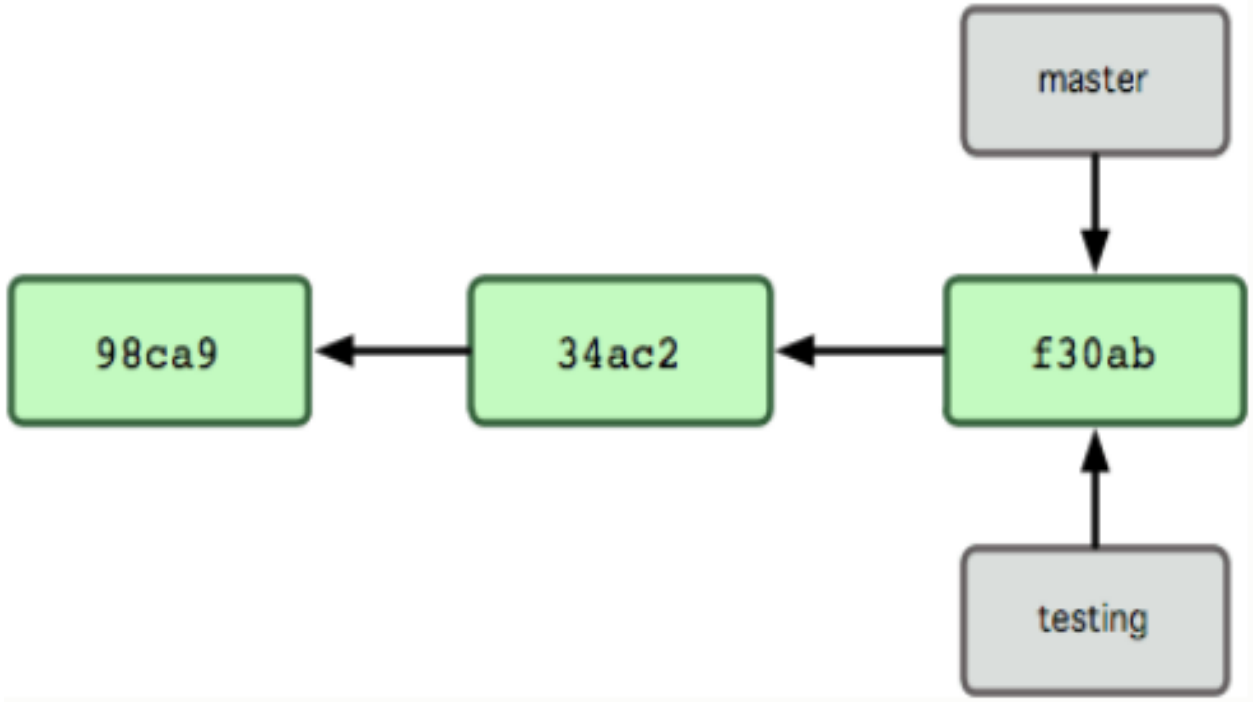
- **Nhánh là gì**
- Nhánh đơn thuần là một con trỏ, trỏ đến các commit và có khả năng di chuyển được
- GIT có nhánh mặc định là master | main, HEAD -> master | main
- HEAD cũng là con trỏ đặc biệt, trỏ đến các commit(detach-head) hoặc trỏ đến nhánh
- Khi HEAD trỏ đến version nào, working directory sẽ show thông tin của version đó
- Khi khởi tạo repository tên nhánh mặc định là master. Trong những lần commit đầu tiên, các commit sẽ được trỏ đến master. Mỗi lần bạn thực hiện commit, con trỏ nhánh master (hiện tại) sẽ tự động tiến lên và trỏ đến commit đó (move forward)





3.1 Branches in a Nutshell

- **Tạo nhánh mới – Chuyện gì sẽ xảy ra khi tạo nhánh mới**
- Khi tạo nhánh mới, git sẽ tạo ra một con trỏ mới cho phép bạn di chuyển vòng quanh
- Giả sử bạn muốn tạo nhánh mới có tên testing
  - Cách 1: git branch testing
  - Cách 2: git branch testing <from_branch>
- Git sẽ tạo ra con trỏ, trỏ đến commit mới nhất của nhánh hiện tại hoặc from_branch



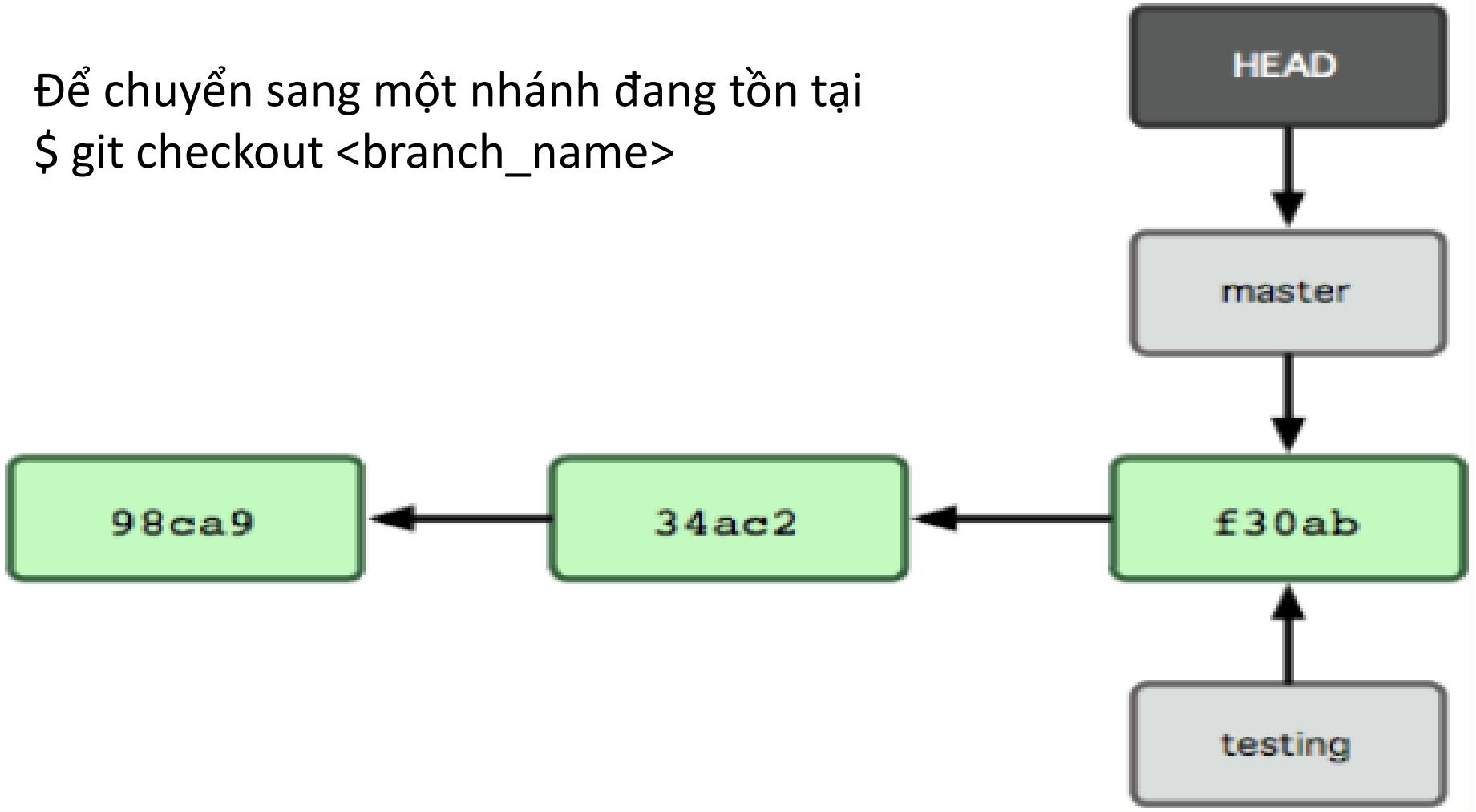
Giả sử bạn tạo ra commit mới. Bây giờ branch nào sẽ tiến lên và trỏ đến commit đó ?

Làm sao để Git biết bạn đang làm việc trên nhánh nào ?

3.1 Branches in a Nutshell

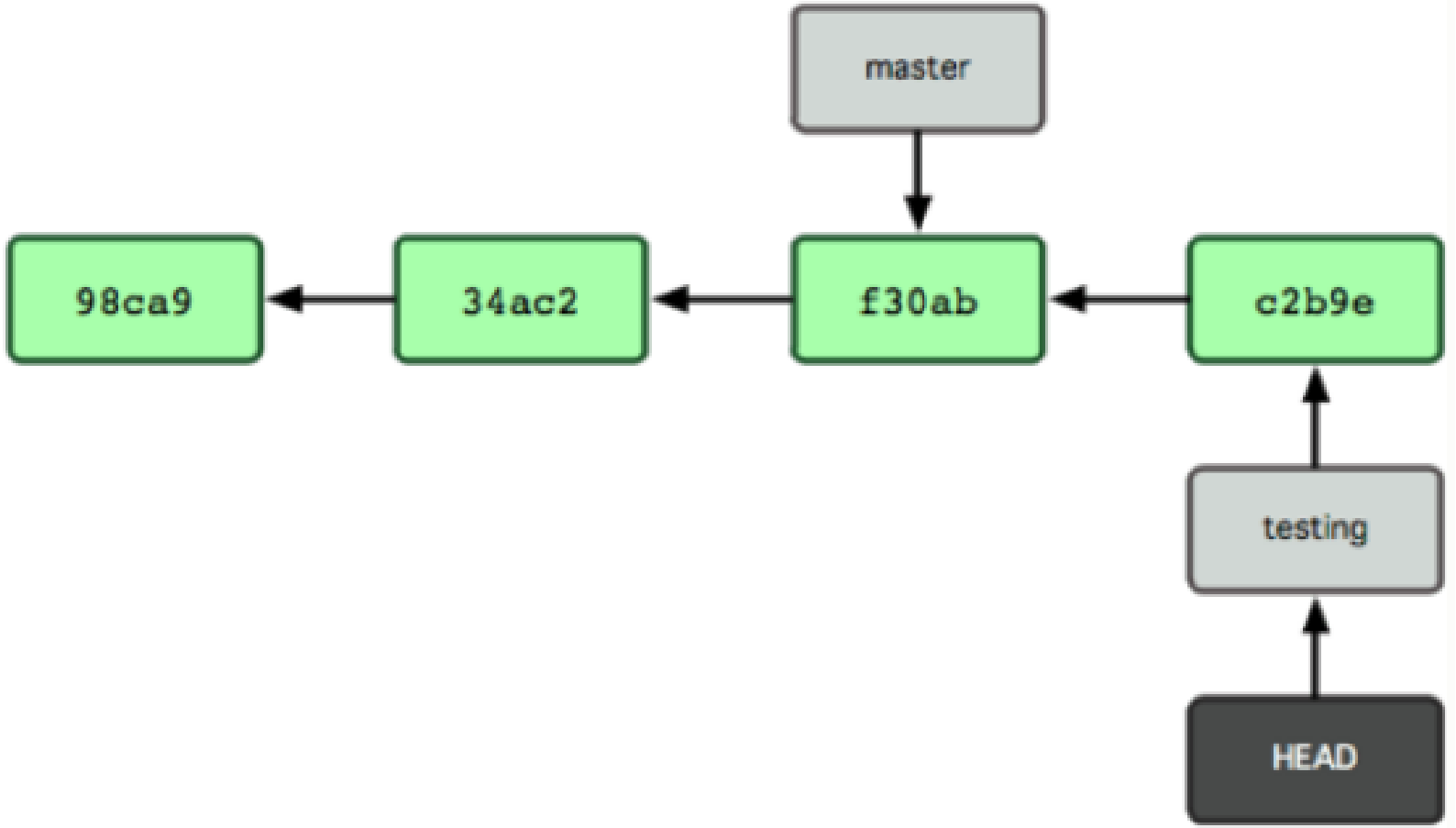
- Git giữ một con trỏ đặc biệt **HEAD** là một con trỏ trỏ đến nhánh nội bộ mà bạn đang làm việc. Trong trường hợp này bạn vẫn đang ở trên nhánh **master**. **$ git branch testing** chỉ tạo mới nhánh testing chứ không tự chuyển sang nhánh đó

- Để chuyển sang một nhánh đang tồn tại
  $ git checkout <branch_name>



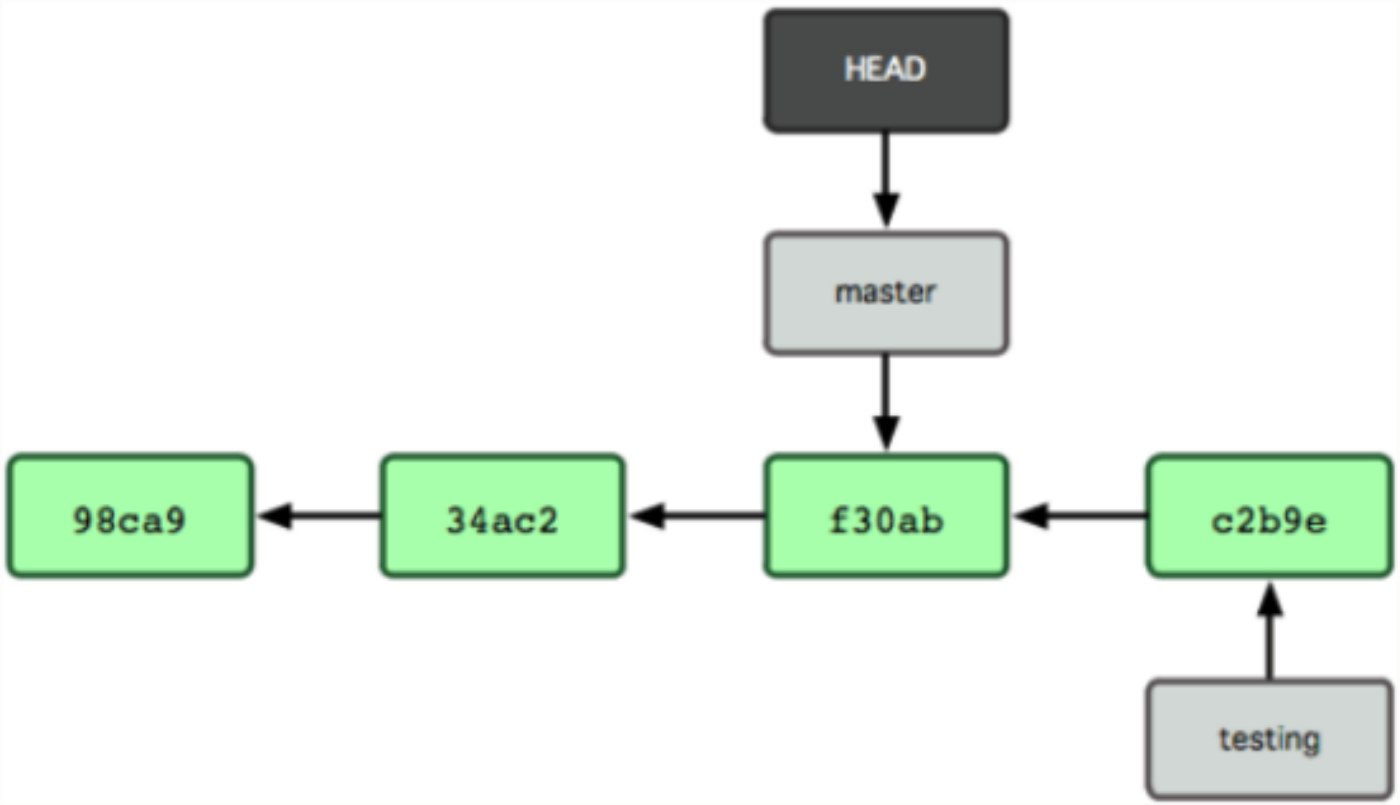
3.1 Branches in a Nutshell

- $ git checkout testing
- $ git add $ git commit (Update project)



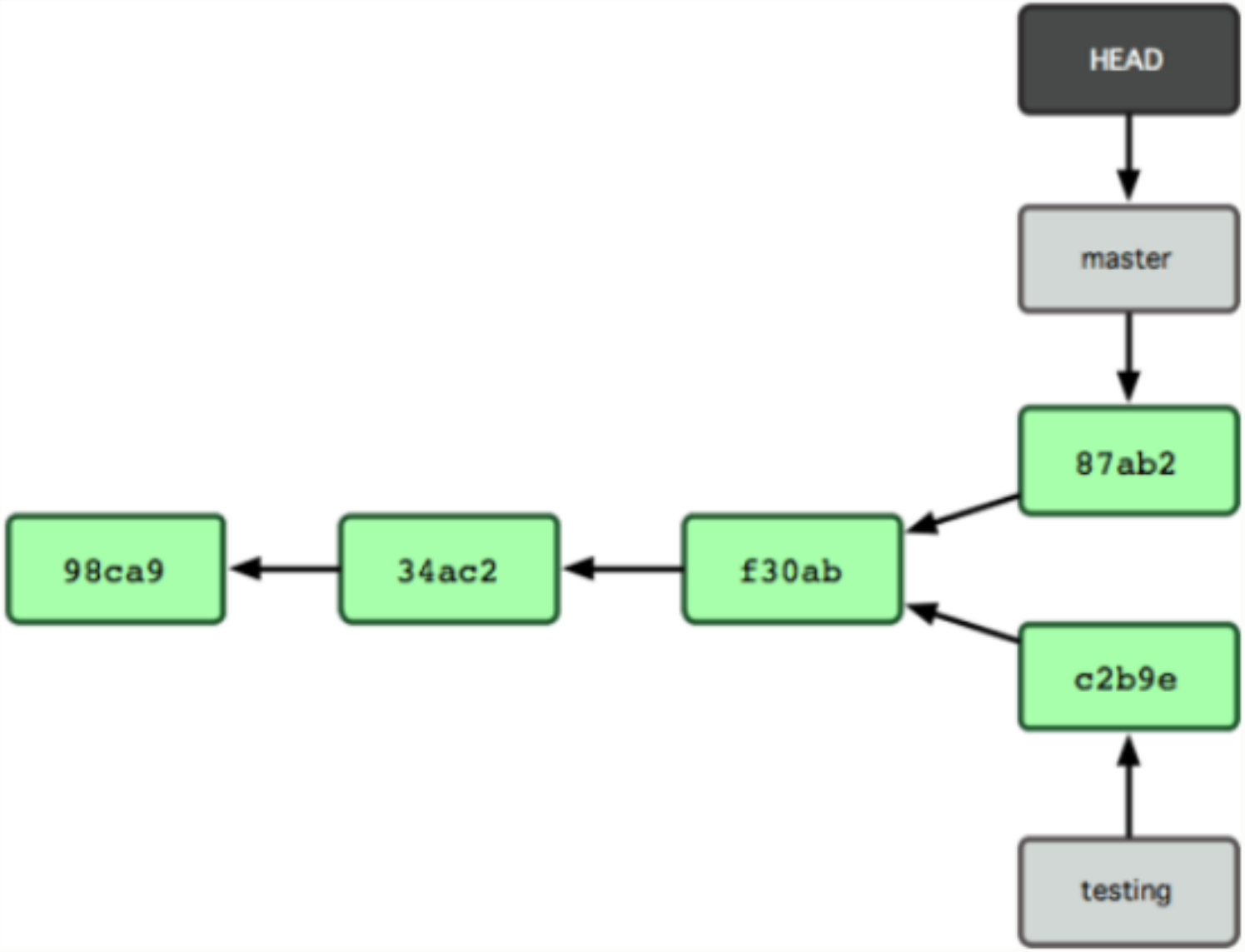
3.1 Branches in a Nutshell

- $ git checkout master



3.1 Branches in a Nutshell

- $ git add $ git commit (Update project)



3.1 Branches in a Nutshell

- Bây giờ lịch sử của dự án đã tách ra trên 2 nhánh master và testing có cùng chung một số commit
- Bởi vì một nhánh trong Git thực tế là một tập tin đơn giản chứa một mã băm SHA-1 có độ dài 40 ký tự của commit mà nó trỏ tới, chính vì thế tạo mới cũng như hủy các nhánh đi rất đơn giản. Tạo mới một nhánh nhanh tương đương với việc ghi 41 bytes vào một tập tin (40 ký tự cộng thêm một dòng mới).
- Điều này đối lập rất lớn với cách mà các **VCS khác** phân nhánh, chính là **copy toàn bộ các tập tin hiện có của dự án sang một thư mục thứ hai**. Việc này có thể mất khoảng vài giây, thậm chí vài phút, phụ thuộc vào dung lượng của dự án, trong khi đó trong Git thì quá trình này luôn xảy ra ngay lập tức.
- Thêm một lý do nữa là, **chúng ta đang lưu trữ cha của các commit, nên việc tìm kiếm gốc/cơ sở để tích hợp lại được thực hiện một cách tự động và rất dễ dàng**. Những tính năng này giúp khuyến khích các lập trình viên tạo và sử dụng nhánh thường xuyên hơn.

3.2 Basic branching and merging

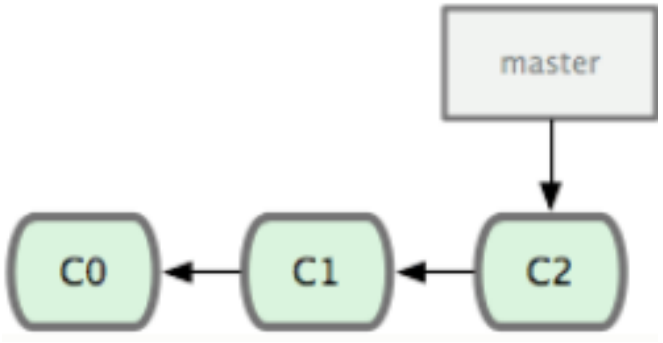
- Xem qua một ví dụ đơn giản về phân nhánh và tích hợp
- 1. Làm việc trên một website đã hoàn thành (todo-app-1.1) trên master
- 2. Tạo nhánh cho version mới todo-app-1.2
- 3. Làm việc trên nhánh đó

- Thông báo có vấn đề nghiêm trọng ở version todo-app-1.1 cần sửa ngay trong hôm nay

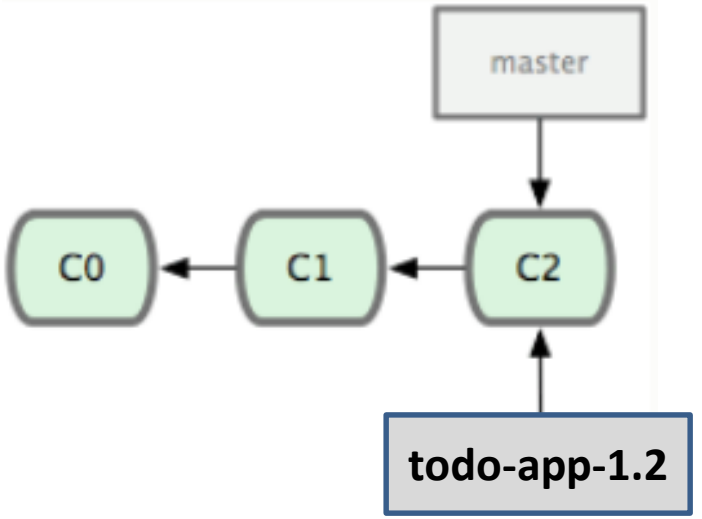
- 1. Chuyển lại về nhánh sản xuất (production - master)
- 2. Tạo mới nhánh khác để khác phục lỗi (todo-app-1.1-hotfix)
- 3. Sau khi đã kiểm tra ổn định, tích hợp nhánh hotfix vào lại master để giao cho khách
- 4. Chuyển về lại nhánh todo-app-1.2 để tiếp tục thực hiện

3.2 Basic branching and merging

- Đầu tiên, bạn đang làm trên nhánh master (todo-app-1.1)

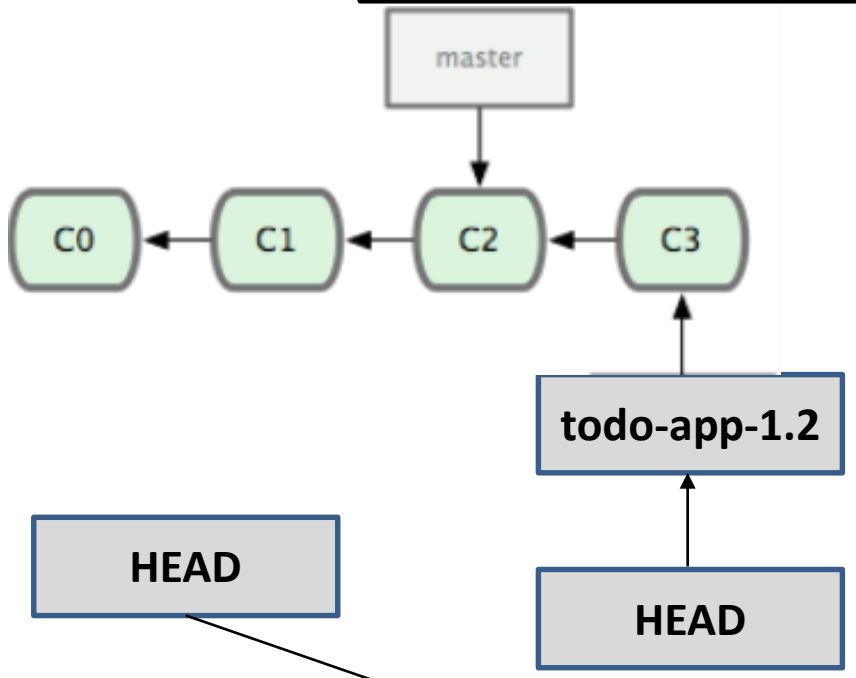


- B2. Tạo branch todo-app-1.2 để thực hiện
  - $ git branch todo-app-1.2
  - $ git checkout todo-app-1.2
  - Rút gọn: $ git checkout –b todo-app-1.2



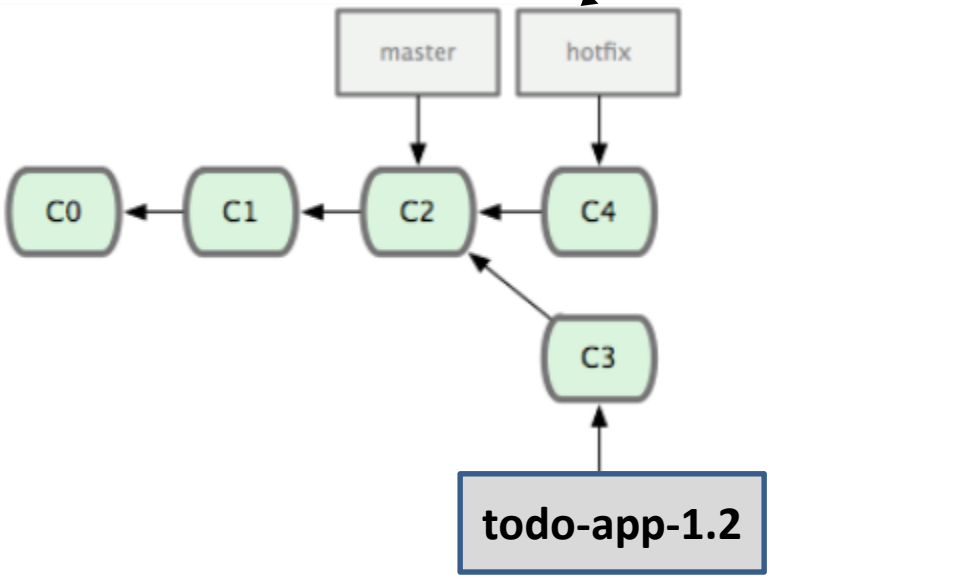
3.2 Basic branching and merging

- B3. Bạn đang làm việc trên nhánh todo-app-1.2
  Thay đổi và thực hiện commit
- HEAD đang trỏ đến todo-app-1.2 branch



- **Vấn đề xảy ra ở production todo-app-1.1 (master)**

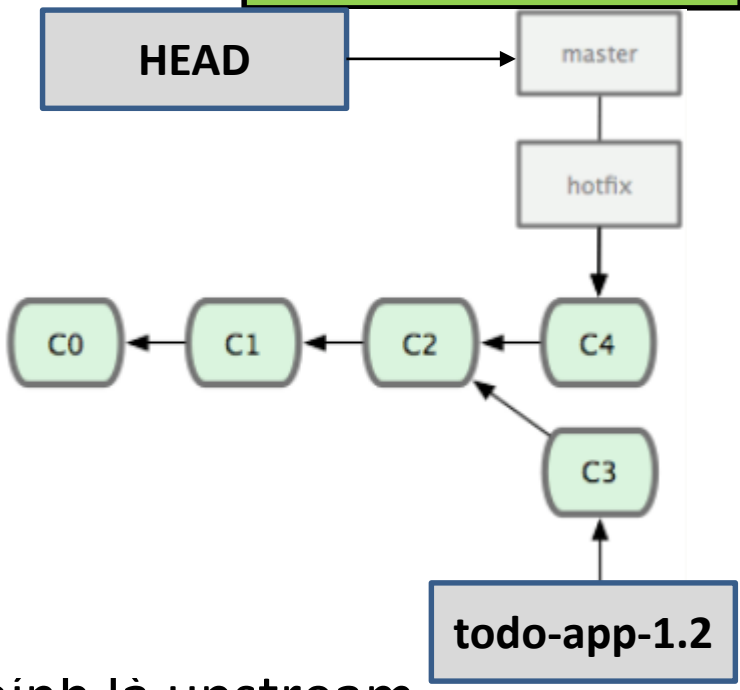
- B4. Trở về lại nhánh master để khác phục sự cố
- $ git checkout master (clean-tree before checkout)



- **Nhánh master là nhánh ổn định của dự án**

- Tạo nhánh hotfix để khắc phục sự cố
- $ gitcheckout –b todo-app-1.1-hotfix
- Thay đổi và thực hiện commit

3.2 Basic branching and merging



- B5. Sau khi đã kiểm tra tính ổn định của nhánh hotfix
- Tích hợp nhánh hotfix (todo-app-1.2-hotfix) vào lại nhánh chính (master)
- $ git checkout master
- $ git merge hotfix
- “Updating 177181 . 19191”
- “Fast forward”: Khi branch mà bạn muốn tích hợp vào (hotfix) đang trỏ đến một commit mà commit này lại chính là upstream của commit mà nhánh hiện tại (master) đang trỏ đến.

VD: Nhánh hotfix được phát triển tử nhánh master và không có sự rẻ nhánh nào từ master. Git sẽ đơn giản hóa bằng cách di chuyển con trỏ về phía trước, vì không có sự rẻ nhánh nào để tích hợp – fast forward

3.2 Basic branching and merging

- Bây giờ bạn có thể xóa nhánh hotfix đi và trở về todo-app-1.2 để tiếp tục
- [$ git branch –d hotfix]
- $ git checkout todo-app-1.2

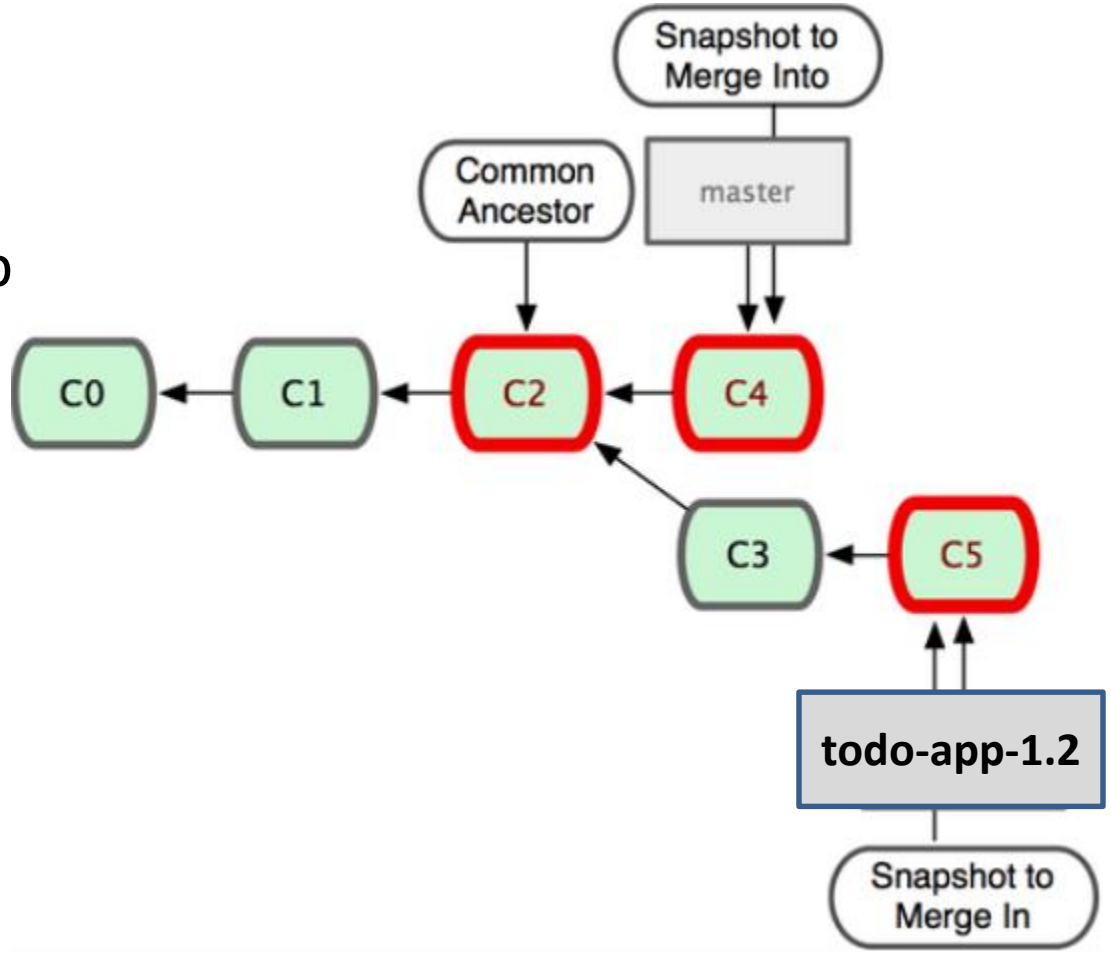
- Lưu ý những thứ bạn fix trong hotfix không chứa trong nhánh todo-app-1.2. Nếu bạn muốn đưa chúng vào todo-app-1.2 Thực hiện $ git merge master
- Hoặc đợi sau khi hoàn tất version 1.2, merge nhánh 1.2 vào lại master

- Giả sử bạn đã hoàn thành version 1.2 todo-app-1.2 và muốn tích hợp vào master
- Thực hiện

3.2 Basic branching and merging

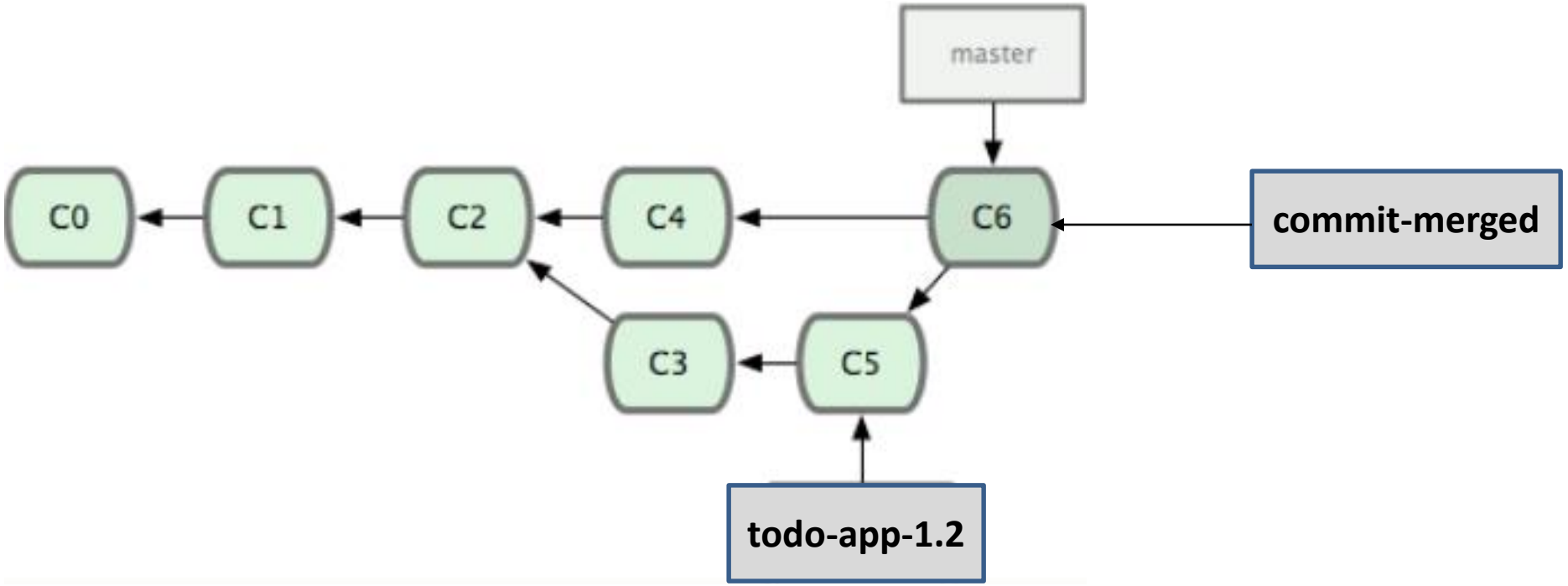
- $ git chechout master
- $ git merge todo-app-1.2

- Trong trường hợp này Git thực hiên tích hợp 3 chiều vì lịch sử nhánh đã tách biệt
- Git sử dụng 2 commit (snapshot) đầu nhánh mà nhánh đang trỏ đến (C4 (master), C5(todo-app-1.2)) và commit cha chung của cả 2 (C2 (common ancestor))



3.2 Basic branching and merging

- Thay vì di chuyển về phía trước (fast-forward)
  Git tạo mới một snapshot – được tạo thành từ lần tích hợp 3 chiều và cũng tự tạo một commit trỏ đến nó. Nó được biết như là commit-merge (có nhiều hơn một cha)
- Lưu ý: Git tự chọn cha chung phù hợp nhất để thực hiện việc tích hợp này. Điều này khí cho Git dễ dàng hơn rất nhiều so với các VCS khác



3.2 Basic branching and merging

- Đôi khi quá trình này không thực hiện một cách suôn sẻ. Nếu bạn thay cùng một nội dung của cùng một tập tin ở 2 nhánh khác nhau, 2 nhánh mà bạn muốn tích hợp vào
- Git không thể thực hiện chúng một cách gọn gàng, bạn sẽ thấy conflict như sau
- [$git checkout master]
- $ git merge todo-app-1.2
- “Auto-merging index.html
- “CONFLICT (content): Merge conflict in Ex01.txt”
- “Automatic merge failed; fix conflicts and then commit the result.”

- Nếu bạn muốn xem tập tin nào đang bị conflict
- $ git status

- Thực hiện fix conflict sau đó tiếp tục commit như thường lệ

3.3 Branch management

- **Xem thông tin nhánh**
- $ git branch
- $ git branch –a
- $ git branch –v
- $ git branch - - merged
- $ git branch - - no-merge

- **Xóa nhánh local**
- $ git branch –d | -D branch

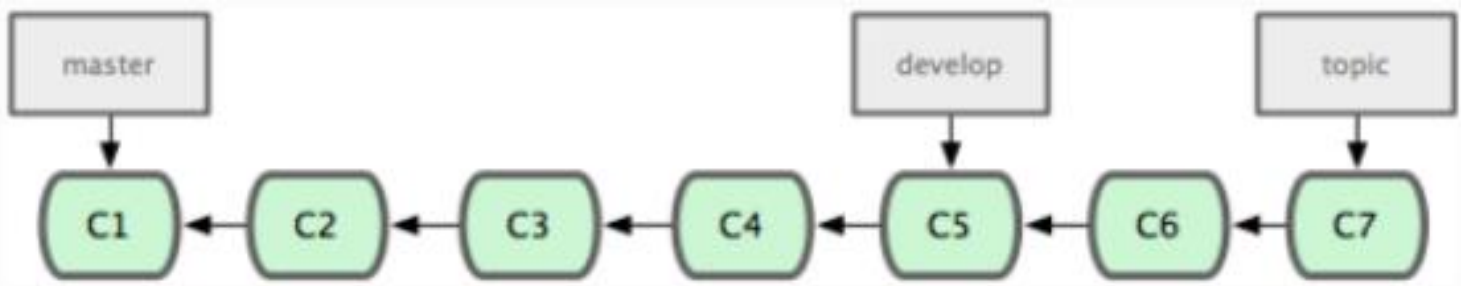
3.3 Branch management

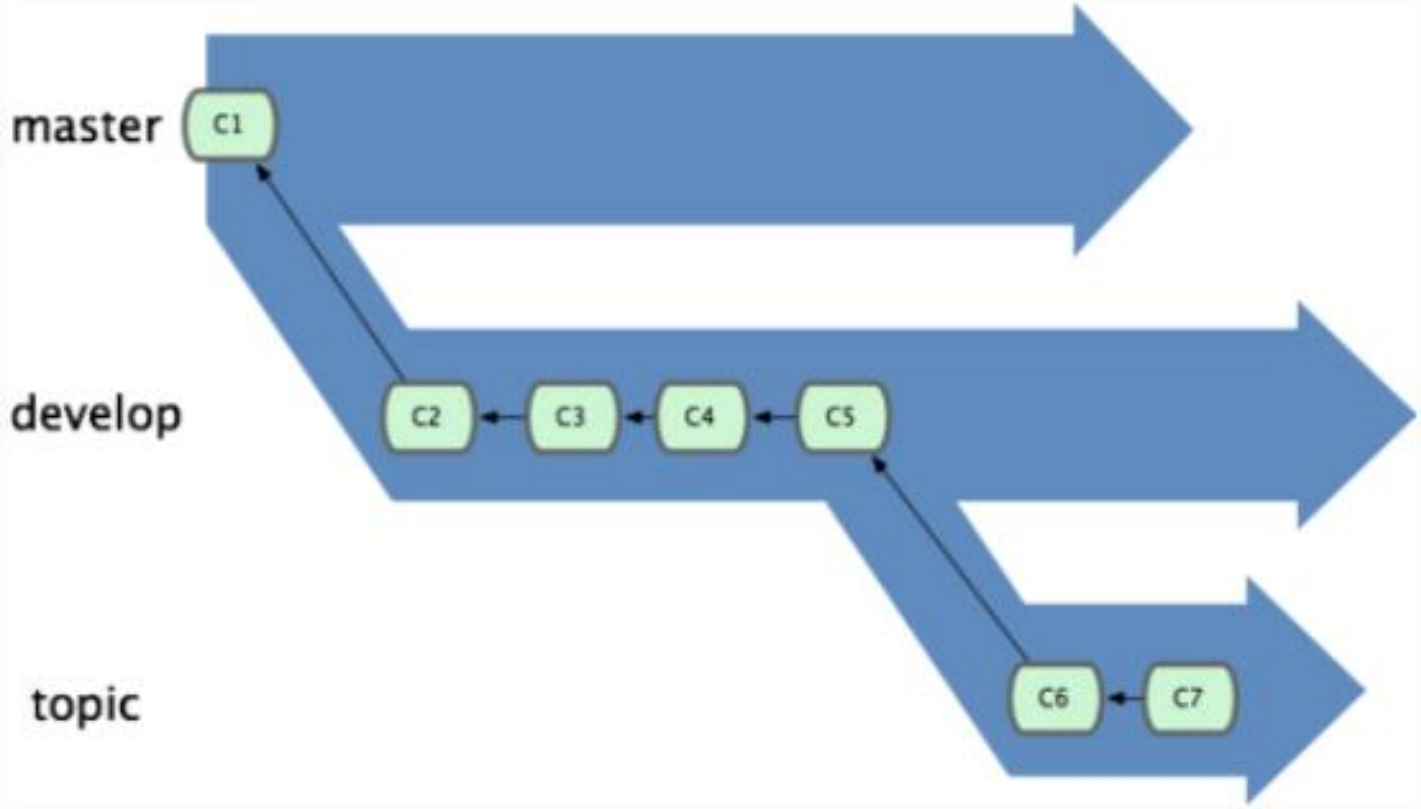
- **Tạo nhánh remote**
- $ git push <remote_name> <localbranch>:<remote_branch>

- **Xóa nhánh remote**
- $ git push –d <remote_name> <branch_name>

- **Tạo nhánh và dịch chuyển**
- $ git branch <branch_name> [<from_branch>] | [<commit-id>]
- $ git checkout <branch_name>
- $ git checkout –b <branch_name> [<from_branch>] | [<commit-id>]
- $ git checkout commit-id (detach HEAD)

3.4 Branching Workflows





3.5 Remote branches

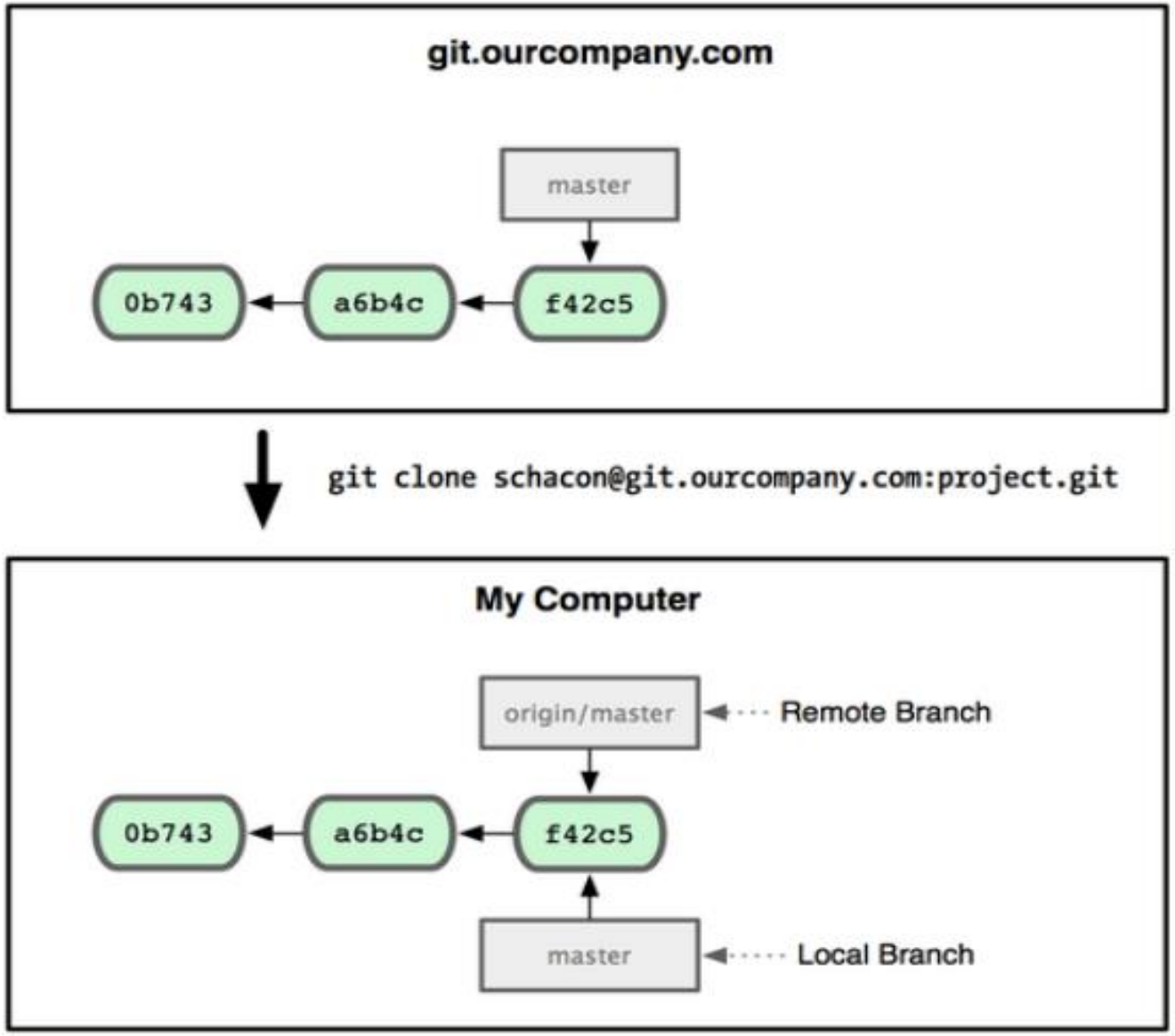
- Remote branches: Là các tham chiếu tới trạng thái của các nhánh trên local-repository
- Chúng là các nhánh nội bộ mà bạn không thể di chuyển được, chúng chỉ di chuyển một cách tự động khi bạn thực hiện giao tiếp qua mạng lưới
- Hoạt động như là các bookmark để bạn biết được các nhánh trên local-repo đang ở đâu vào lần cuối cùng bạn commit tới
- Ví dụ muốn xem nhánh **master** trên remote origin của bạn như thể nào kể từ lần giao tiếp cuối cùng bạn sẽ dùng **orgin/master**

- Nếu bạn và người bạn khác đang làm việc trên version 1.3, người kia đẩy dữ liệu lệnh nhánh **app-1.3**. Bạn có thể có riêng nhánh app-1.3 trên local-repo nhưng nhánh trên máy chủ sẽ là **orgin/app-1.3**

- Để xem cách nhánh ở remote
- $ git branch -a

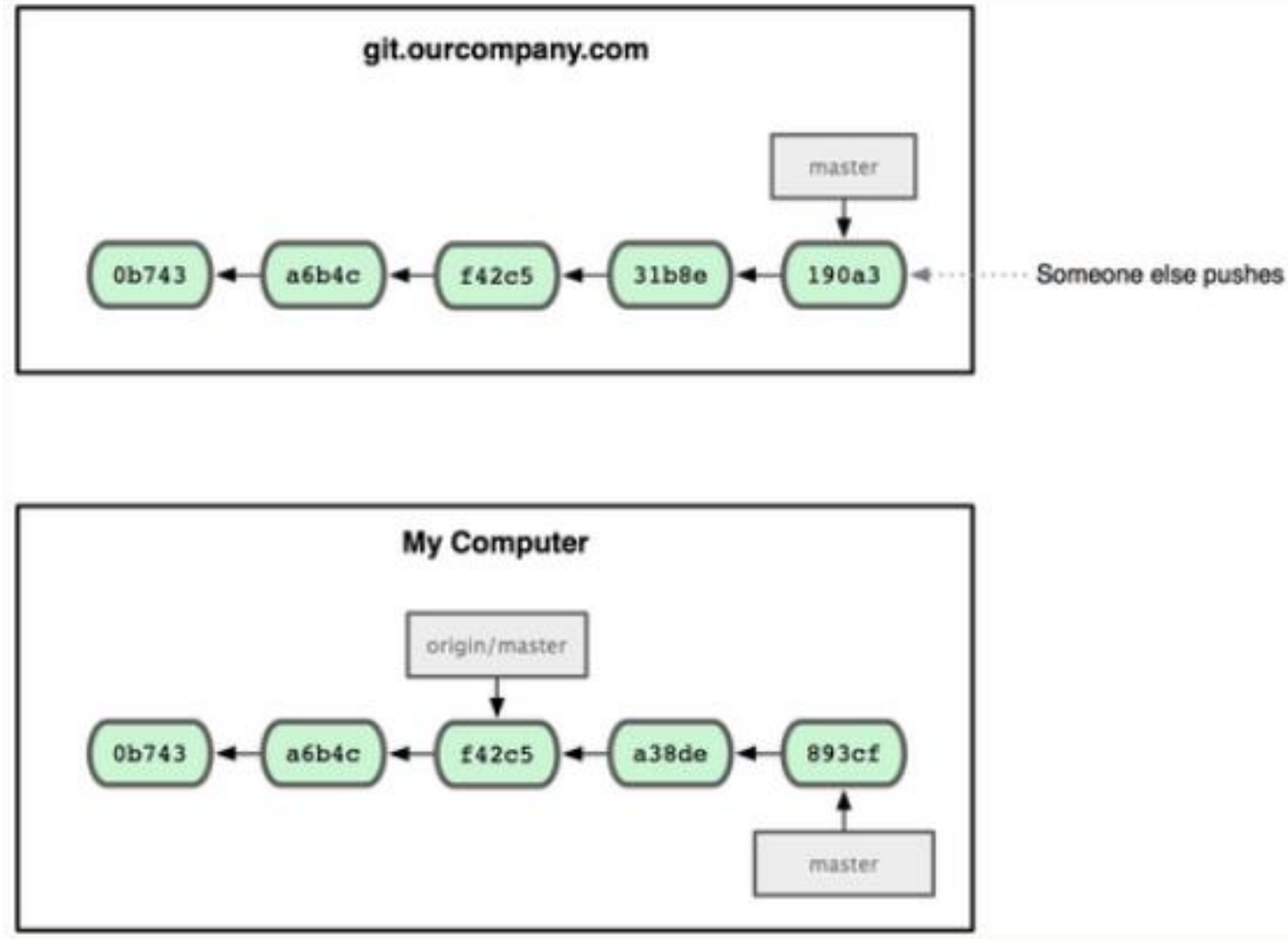
3.5 Remote branches

- orgin remote: git.ourcompany.com



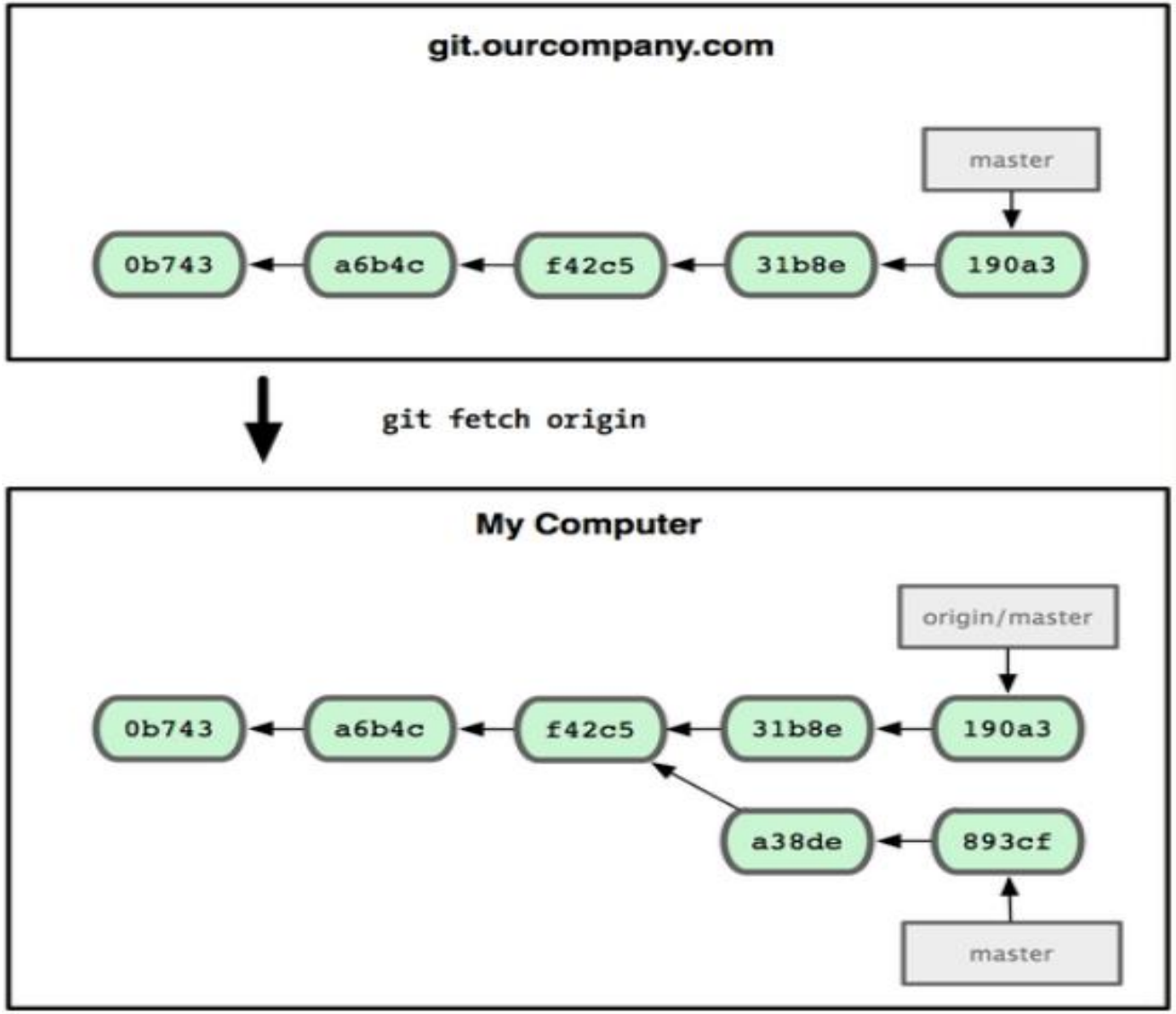
3.5 Remote branches

- Bạn thực hiện một số thay đổi ở master trên local - repo. Trong lúc đó cũng có người khác đẩy code lên master trên remote – repo



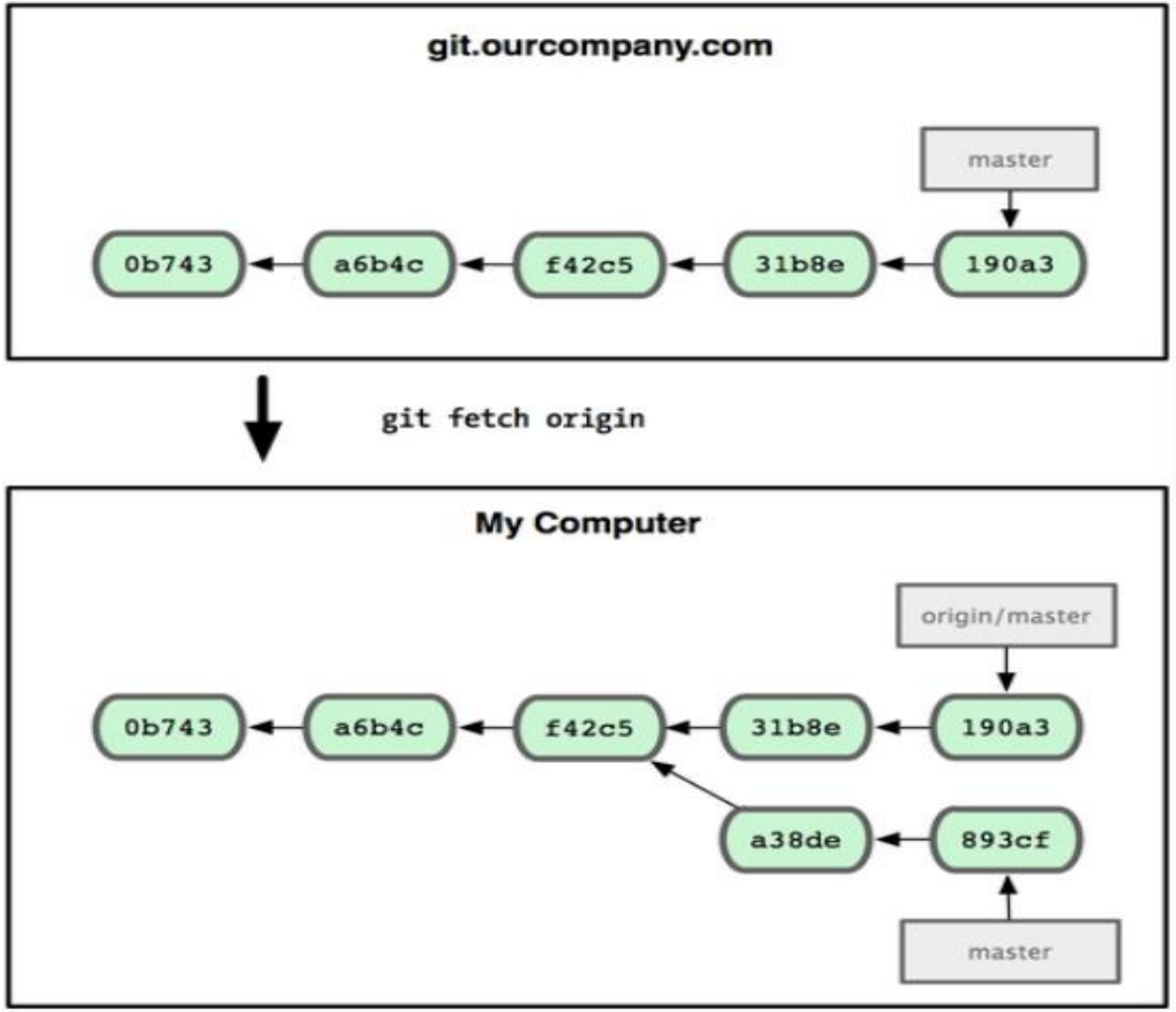
3.5 Remote branches

- Bạn thực hiện một số thay đổi ở master trên local - repo. Trong lúc đó cũng có người khác đẩy code lên master trên remote – repo
- Như vậy: Trên local – repo chưa có code mới nhất trên remote
- Để đồng bộ hóa thay đổi
- $ git fetch origin
- Lệnh $ git fetch giúp cập nhật các tham chiếu từ xa



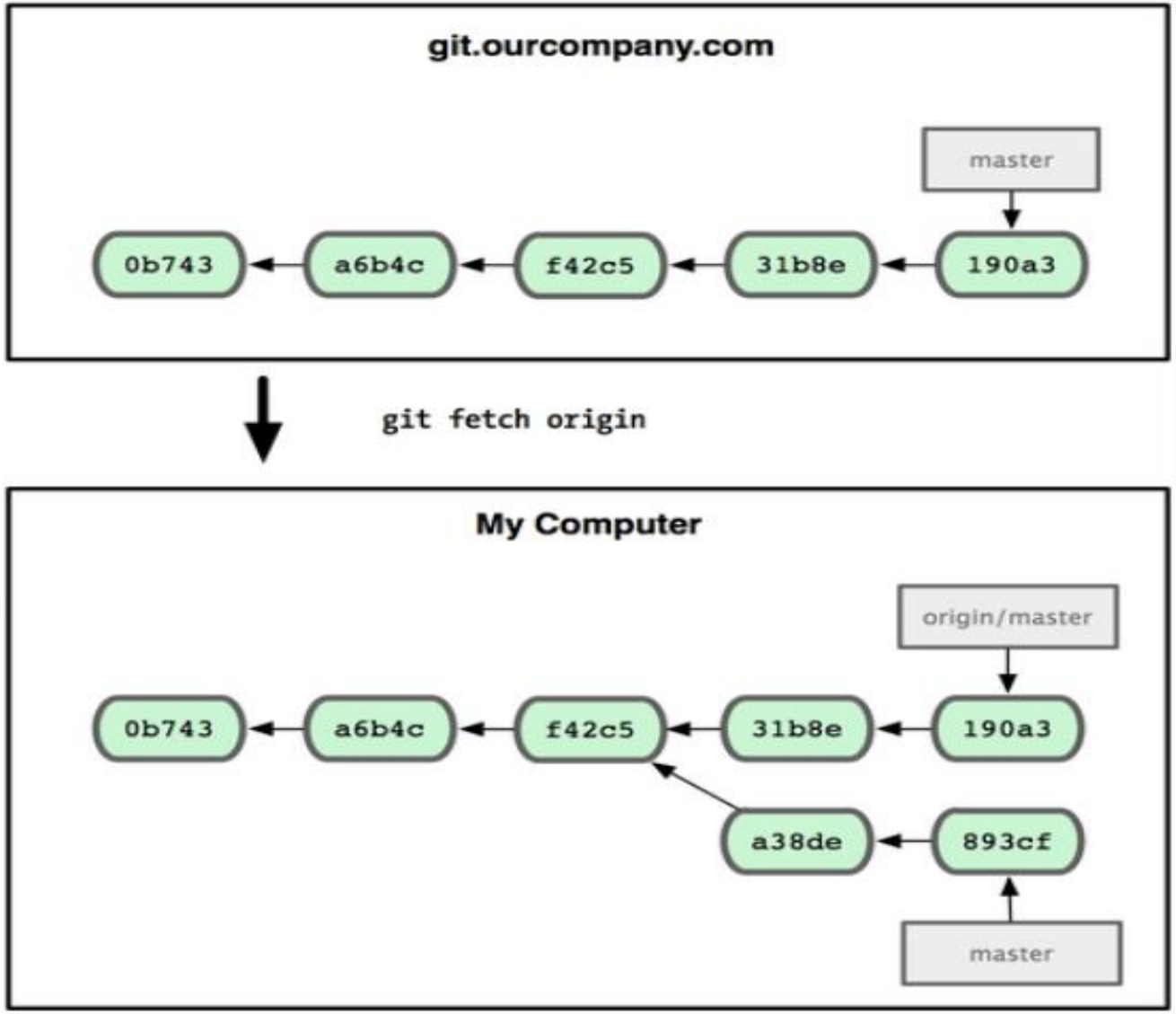
3.5 Remote branches

- Lệnh $ git fetch giúp cập nhật các tham chiếu từ xa
- Nếu bạn muốn tích hợp và có code của origin/master vào nhánh master ở local
- Thực hiện
- $ git pull [--rebase]



3.5 Remote branches

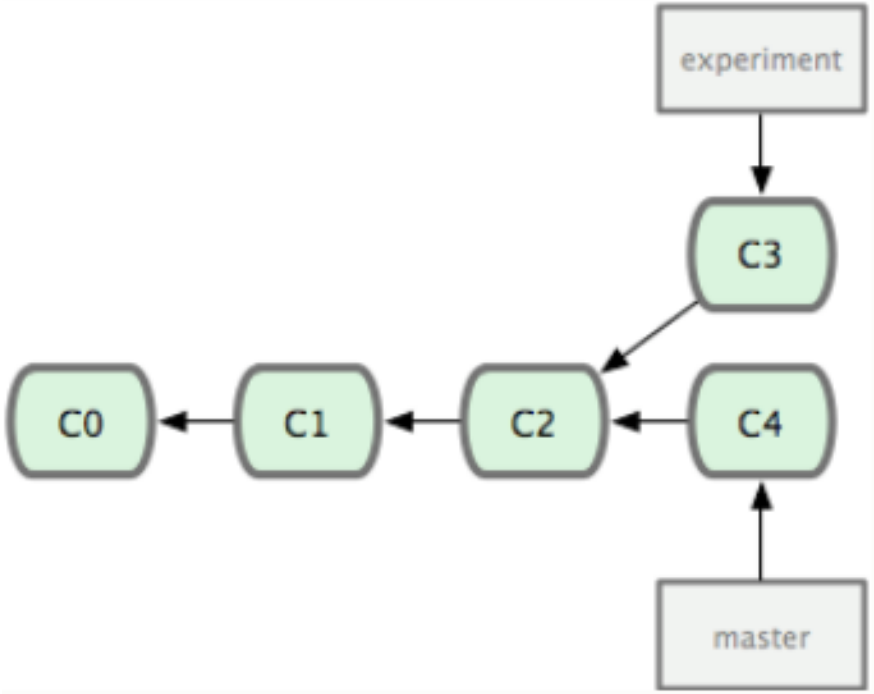
- Push (Local - Remote) – Đẩy lên
- Khi bạn muốn chia sẽ một nhánh ở local với mọi người bạn cần đẩy nó lên máy chủ mà bạn có quyền ghi trên đó
- $ git push <remote> <branch>



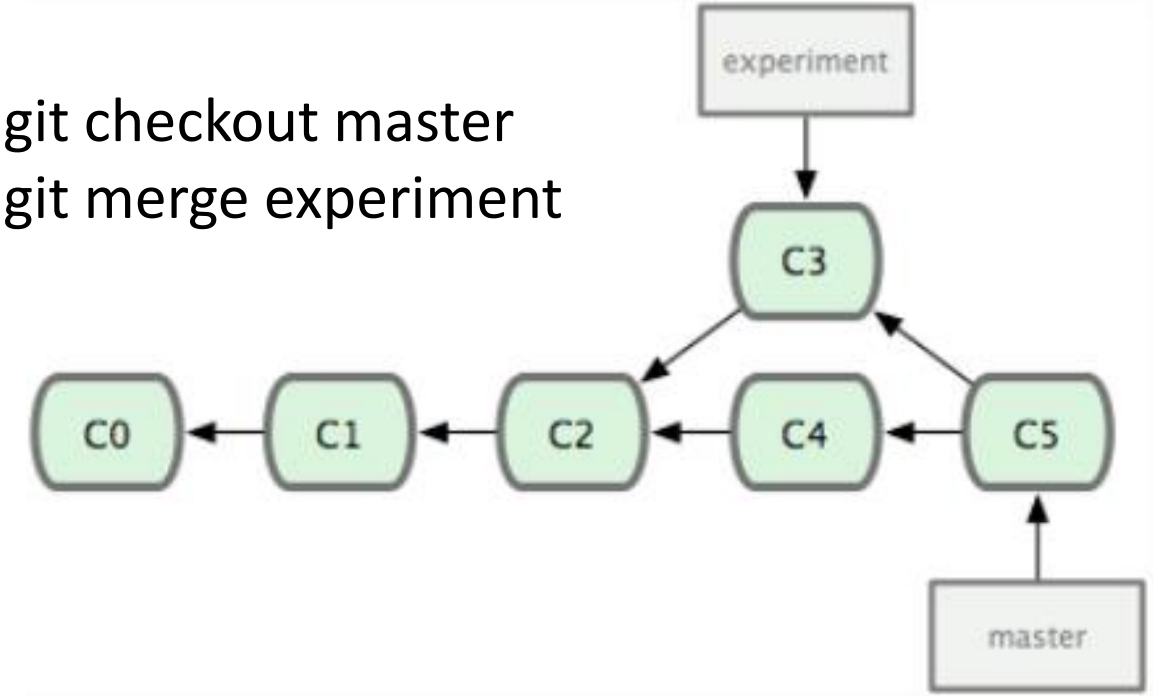
- Trước khi push
- B1. $ git pull [–rebase] để có code mới nhất từ remote
- B2. Fix conflict nếu có xảy ra
- B3. Kiểm tra commit history của loca và remote. Đảm bảo commit ở local phải upstream so với remote
- B4. Chọn remote branch để push

3.6 Git rebasing

- Trong Git có 2 cách để tích hợp các thay đổi từ nhánh này vào nhánh khác đó là merge và rebase. Chúng ta sẽ cùng nhau tìm hiểu rebase là gì và áp dụng merge và rebase trong các trường hợp cụ thể
- **Cơ bản về Rebase**
- Xem lại phần Tích hợp và giả sử chúng ta đang có 2 commit trên 2 nhánh khác nhau
- Trước và sau khi tích hợp sử dụng merge

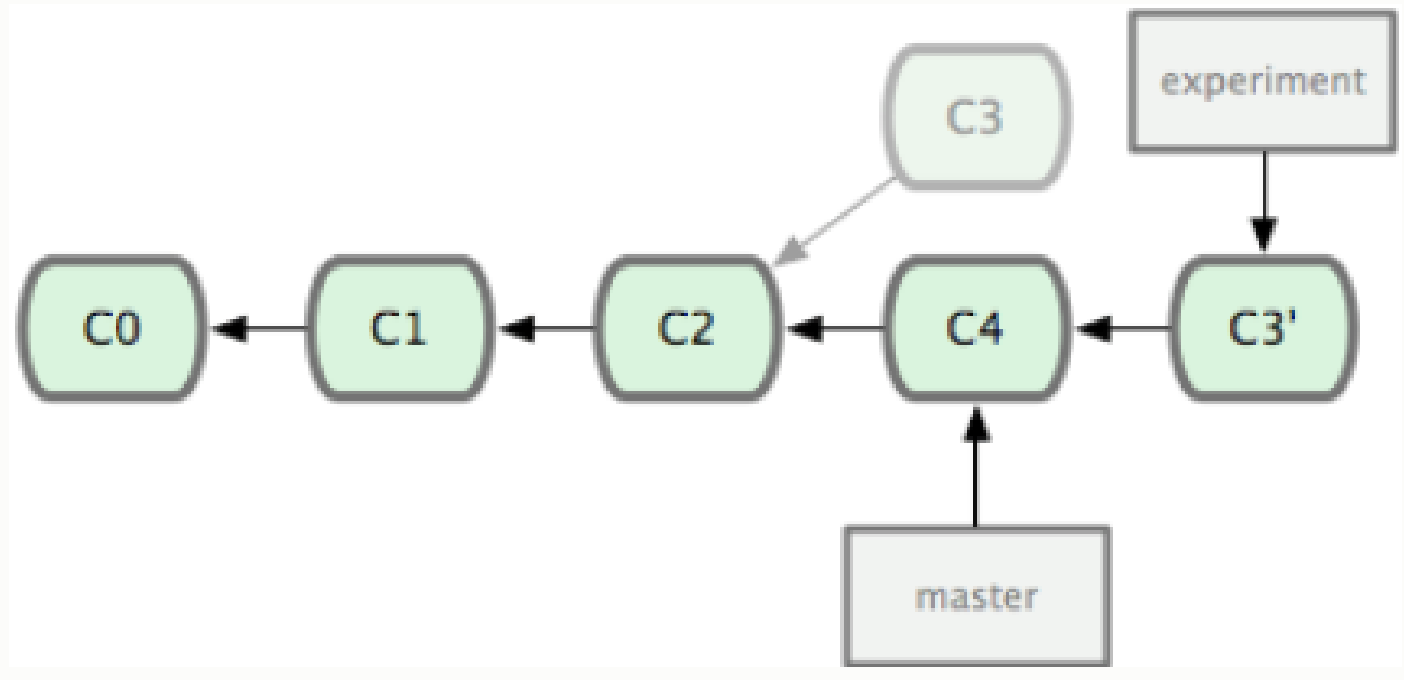


$ git checkout master
$ git merge experiment



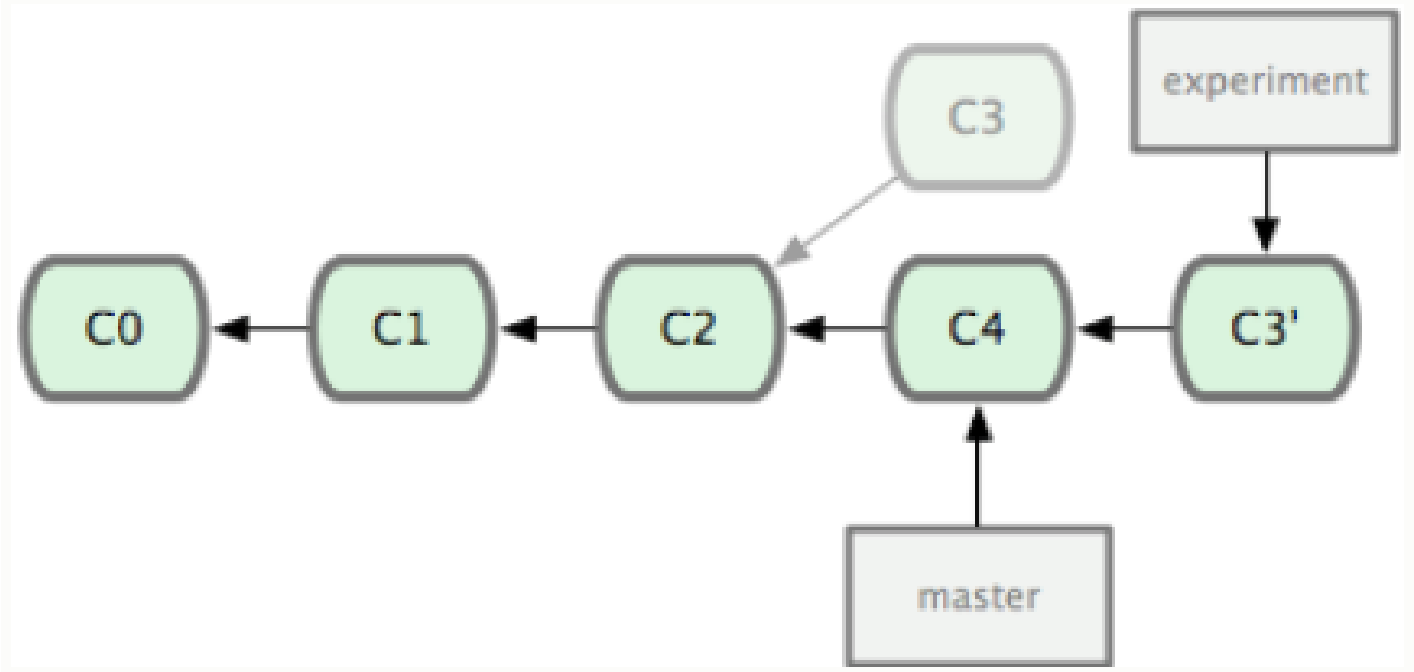
3.6 Git rebasing

- **Cơ bản về Rebase**
- Cách khác: Sử dụng bản vá được thay đổi ở C3(exp) và áp dụng nó **lên trên** C4(master)
- Rebase: Sử dụng tất cả các thay đổi được commit ở một nhánh (exp) và chạy lại chúng trên một nhánh khác (master)
- $ git checkout experiment
- $ git rebase master



3.6 Git rebasing

- **Cơ bản về Rebase**
- $ git checkout experiment
- $ git rebase master
- Giải thích: Đi đến commit cha chung của 2 nhánh (C2). Tìm sự khác biệt (changed) trong mỗi commit của nhánh đang làm việc (exp), lưu lại các tệp đó vào tập tin tạm thời.
- Khôi phục nhánh hiện tại về cùng commit (C4) với nhánh mà bạn đa rebase (master)
- Áp dụng lần lượt các thay đổi
- Quay lại nhánh master để fast-forward
- $ git merge experiment



3.6 Git rebasing

- **Cơ bản về Rebase**
- Bây giờ snapshot mà C3’ đang trỏ đến giống như snapshot của C5 khi sử dụng merge
- Không có sự khác biệt nào khi so sánh kết quả của hai phương pháp này, nhưng sử dụng rebase sẽ cho chúng ta lịch sử rõ ràng hơn.
- Nếu bạn xem xét lịch sử của nhánh mà chúng ta rebase vào, nó giống như một đường thẳng: mọi thứ dường như xảy ra theo trình tự, thậm chí ban đầu nó diễn ra song song.

- Thông thường bạn áp dụng rebase trên remote để đảm bảo các commit được áp dụng một các rõ ràng theo thời gian

- Đúng trình tự bạn sẽ thực việc công việc trên một nhánh và sau đó rebase trở lại orgin/master khi đã sẵn sàng push. Theo cách này, commit của bạn sẽ fast-forward tiến lên phía trước và xem history để thấy commit của bạn là mới nhất so với remote/master

3.6 Git rebasing

- **Cơ bản về Rebase**

- Lưu ý: Snapshot được trỏ đến cuối cùng cho dù sử dụng rebase hay merge thì nó vẫn giống nhau – chỉ khác nhau các bước thực hiện. Rebase được thực hiện bằng cách thực hiện lại các thay đổi của nhánh này qua nhánh khác theo thứ tự chúng thực hiện. Trong khi đí merge lấy điểm kết thúc và tích hợp chúng lại với nhau

- **Tập lệnh xem commit – history**
- & git log --oneline
- $ git log --graph --oneline --all
- $ git log --oneline --decorate --all --graph
- $ git config --global alias.tree "log --oneline --decorate --all --graph"

3.7 Bonus

- **Một số thủ thuật khác và lưu ý kèm theo**
- Pull code
- Cách 1: (Tránh xuất hiện merged-commit)
  - Local: Không commit code
  - Remote: Upstream với local branch
  - $ git pull
  - [fix conflict]
  - $ git add $ git commit $ git push
- Cách 2: (Nên sử dụng)
  - Local: $ git commit code (Tránh mất code)
  - Remote:
  - $ git pull –rebase
  - [fix conflict]
  - $ git add $ git commit $ git push

3.7 Bonus

- **Một số thủ thuật khác và lưu ý kèm theo**
- **Merge**
- When: Merge nhánh đang phát triển (gần ổn định) vào nhánh ổn định (master)
- Cách thực hiện
  - $ git checkout master
  - $ git merge todo-app-1.2
- Giải thích: Đang ở nhánh master, tích hợp nhánh todo-app-1.2 vào nhánh master
- **Rebase**
- When: Rebase nhánh đang phát triển trên nhánh đã ổn định(master) để tích hợp những thay đổi của master vào nhánh đang phát triển
- Cách thực hiện
  - $ git checkout todo-app-1.2
  - $ git rebase master
- Giải thích: Đang ở nhánh todo-app-1.2, tích hợp nhánh master vào nhánh todo-app-1.2. Bây giờ 1.2 đang upstream so với master

3.7 Bonus

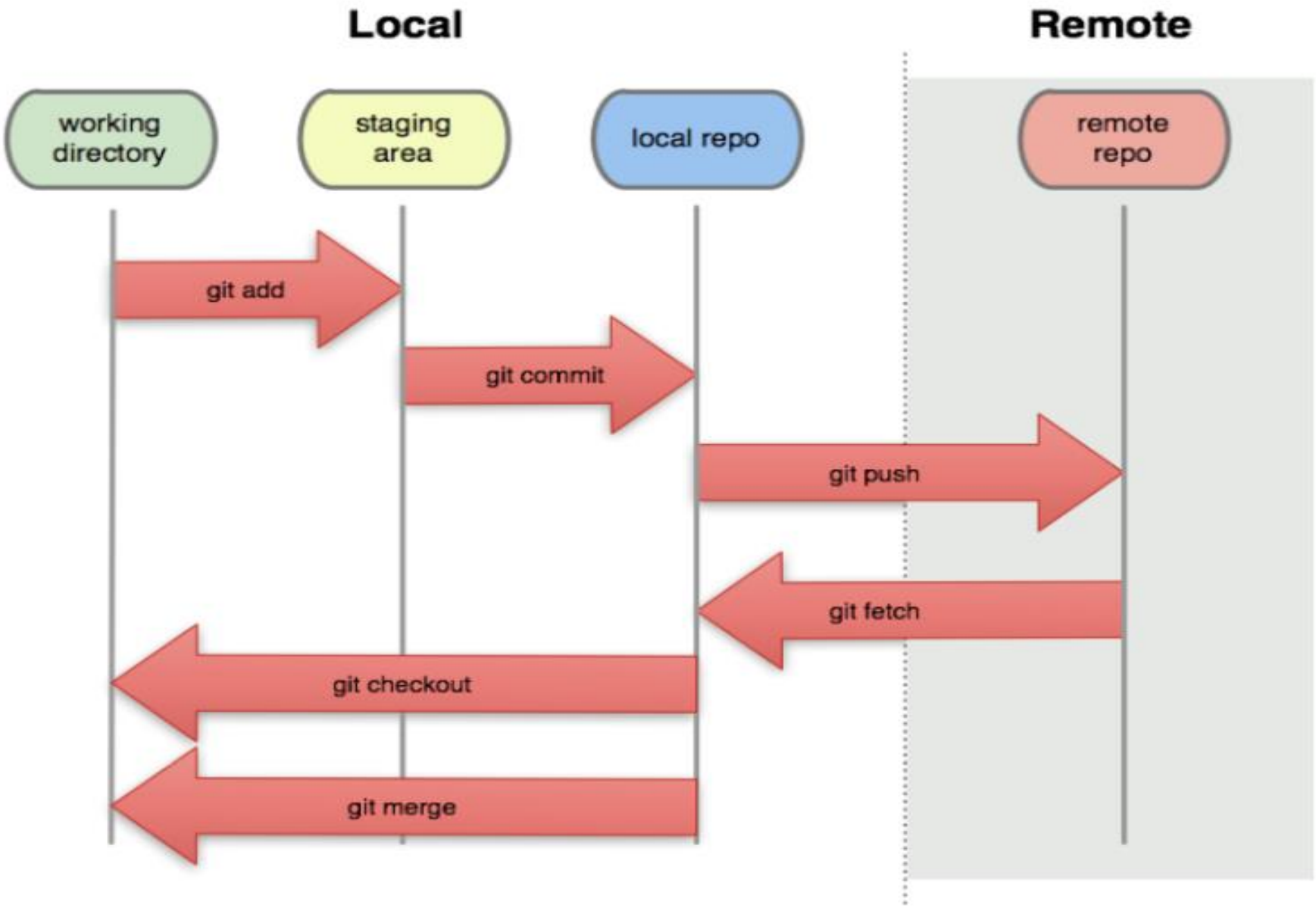
- **Một số thủ thuật khác và lưu ý kèm theo**
- **Squash**
- Áp dụng với feature branch
- Khi bạn có một chức năng cần thực hiện và chia sẽ bởi nhiều người. Nhưng nó chỉ là một chức năng và muốn khi tích hợp và development. Nó chỉ là một commit cho 1 chức năng này

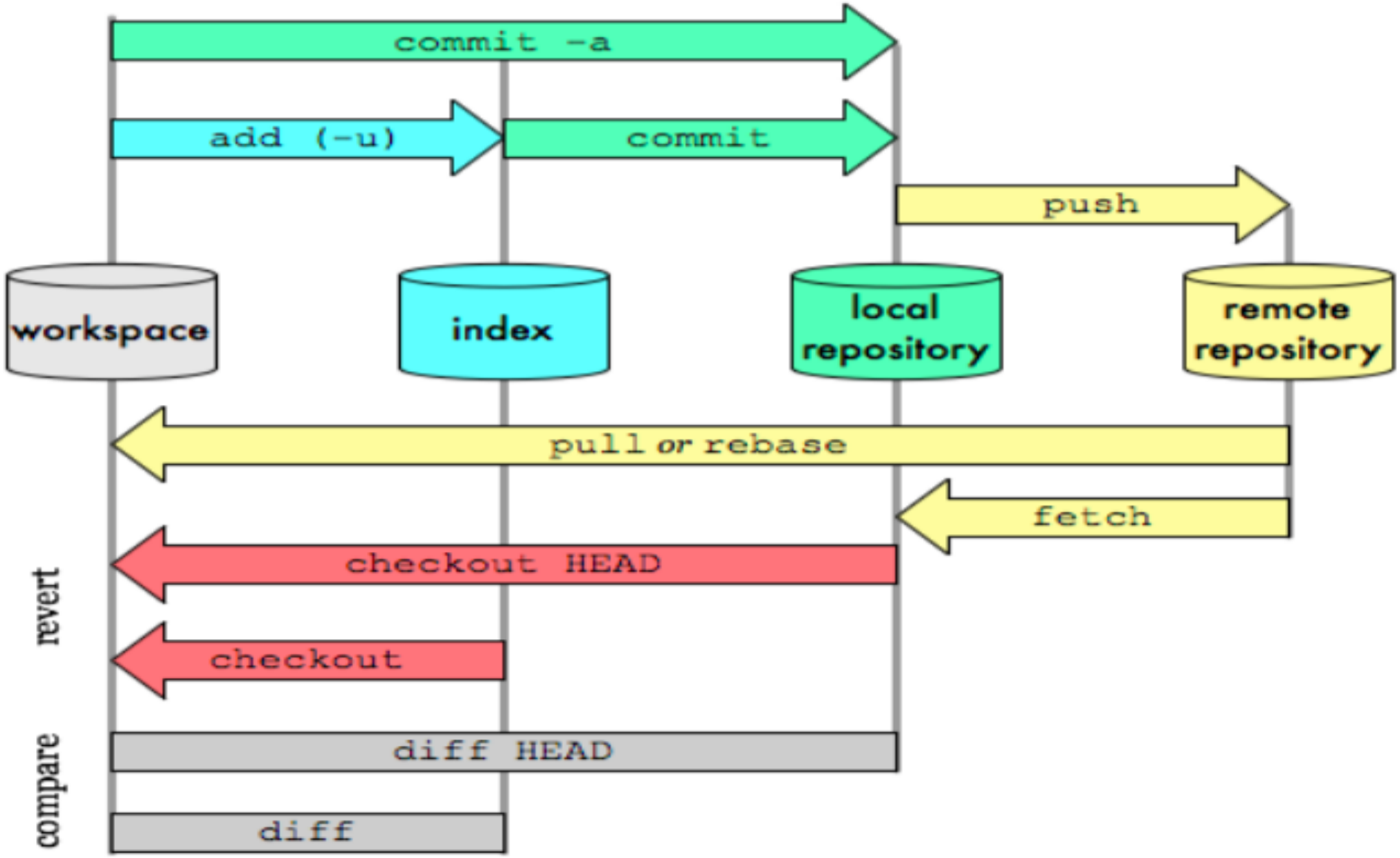
- B1. Tạo feature branch – feature/todo-app-draw-graph (ở local | remote)
- Nếu ở local thì 1 người push lên để sử dụng chung
- Nếu ở remote thì ở local chỉ cần checkout từ origin/feature/todo-app-draw-graph
- B2. Nhiều người cùng thay đổi và đẩy commit lên remote/feature/…
- B3. Tích hợp vào lại development với 1 commit
- $ git checkout development
- $ git merge –squash feature/….

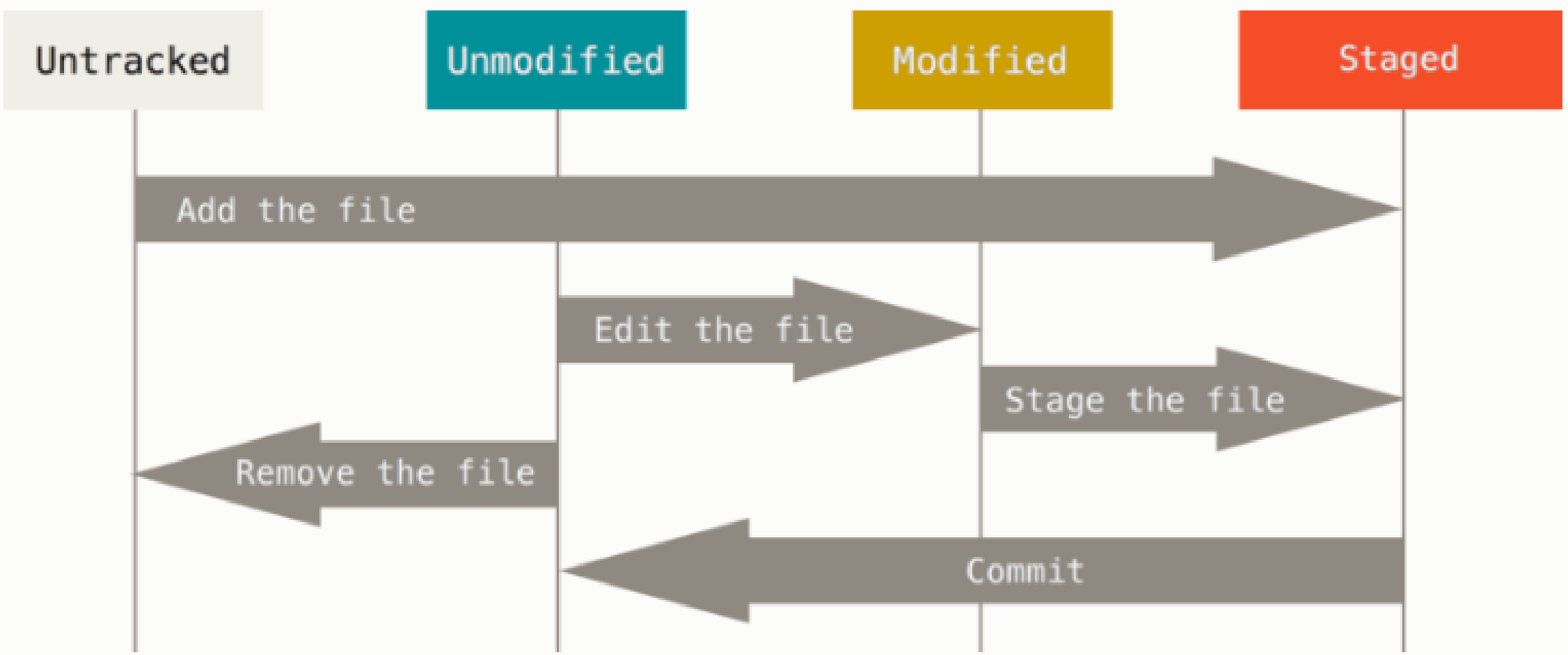
3.7 Bonus

- **Một số thủ thuật khác và lưu ý kèm theo**
- Clean tree with Stash
- $ git stash
- $ git stash save message
- $ git stash list
- $ git stash show
- $ git stash apply [stash@{n}]
- $ git stash drop stash@{n}

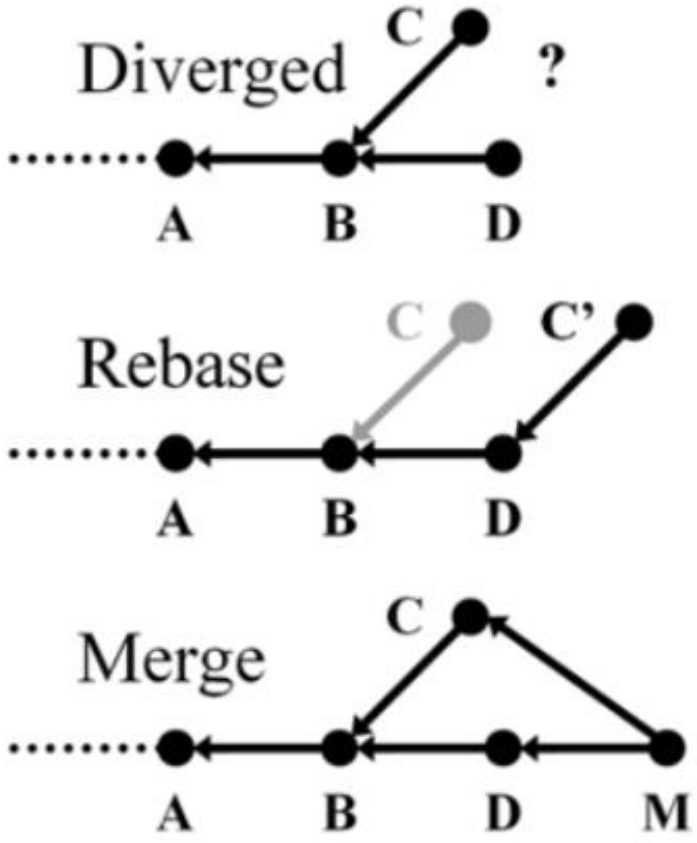
- Cherry-pick and Shelve changes


Local
Remote
working directory
staging area
local repo
remote repo
git add
git commit
git push
git fetch
git checkout
git merge

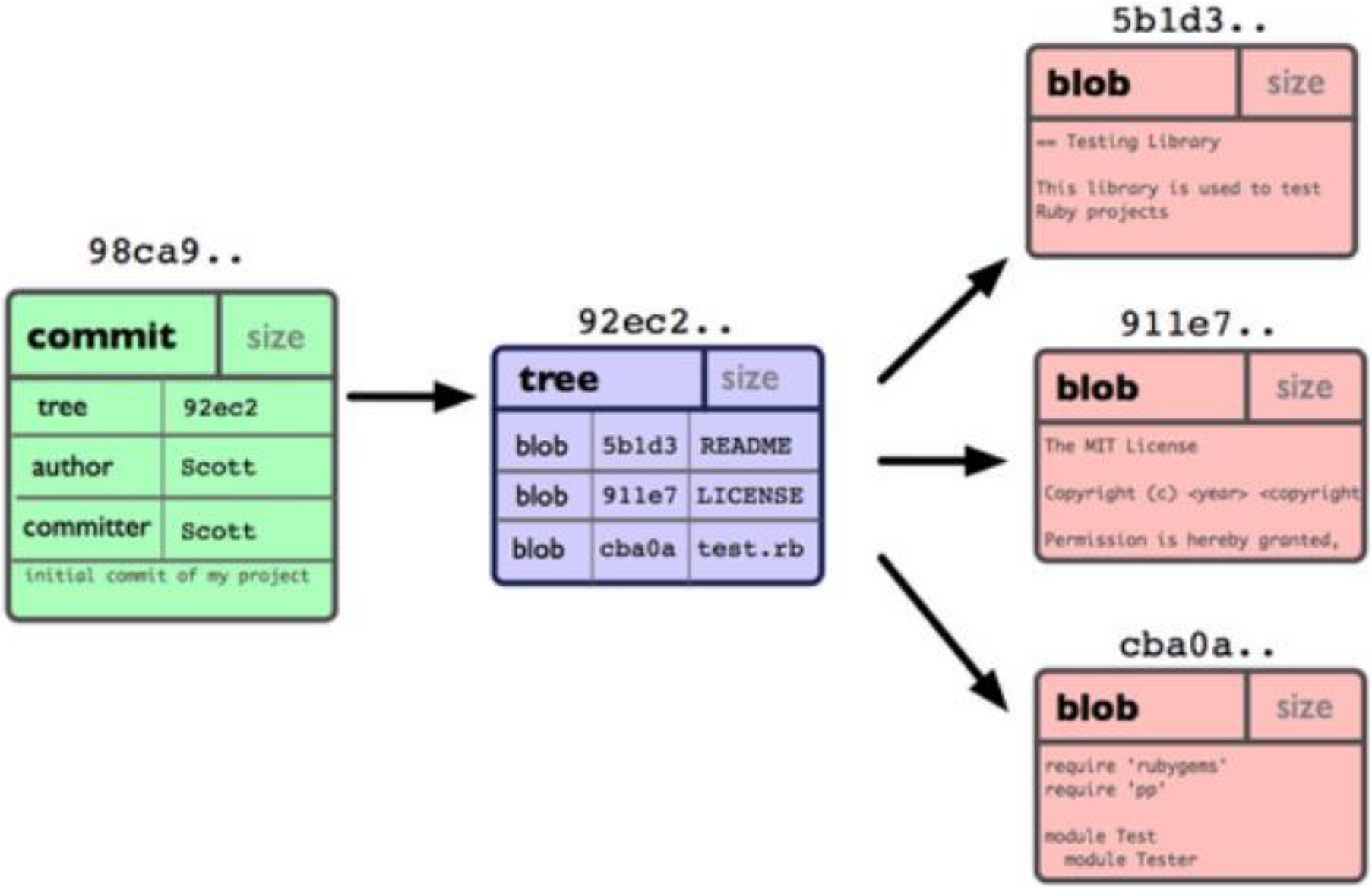

commit -a
add (-u)
commit
push
workspace
index
local repository
remote repository
pull or rebase
fetch
checkout HEAD
revert
checkout
compare
diff HEAD
diff

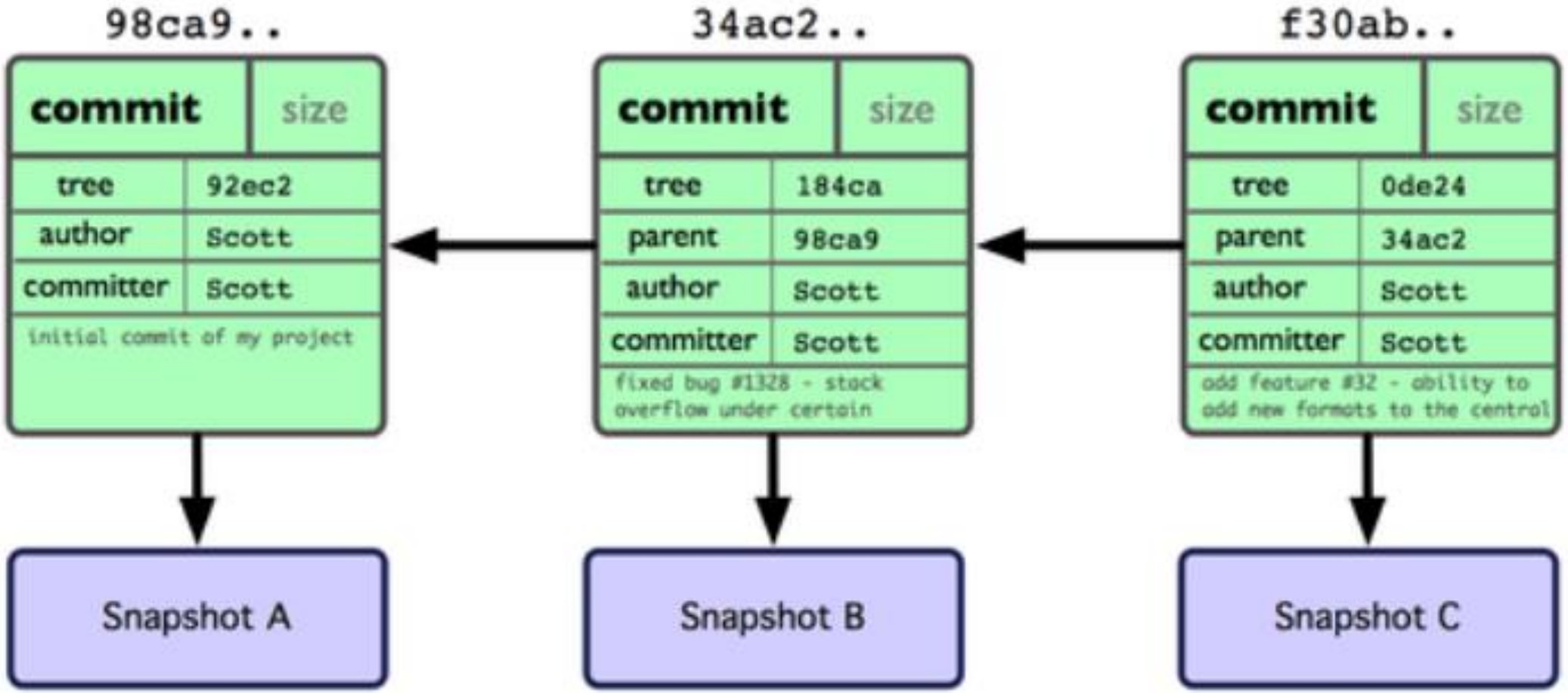

Untracked
Unmodified
Modified
Staged
Add the file
Edit the file
Stage the file
Remove the file
Commit

use rebase for a linear history



rebase rewrites history, use it for **local** changes only!

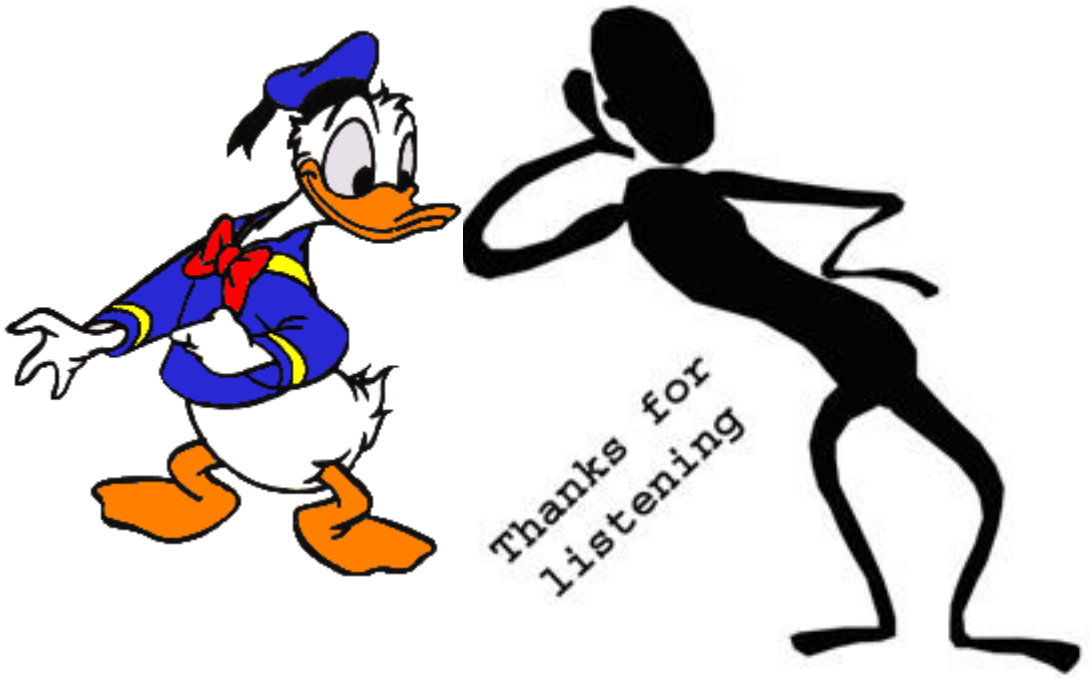

98ca9..
commit
size
tree
92ec2
author
Scott
committer
Scott
initial commit of my project
92ec2..
tree
size
blob
5b1d3
README
blob
911e7
LICENSE
blob
cba0a
test.rb
5b1d3..
blob
size
== Testing Library
This library is used to test
Ruby projects
911e7..
blob
size
The MIT License
Copyright (c) <year> <copyright
Permission is hereby granted,
cba0a..
blob
size
require 'rubygems'
require 'pp'
module Test
module Tester


98ca9..
commit
size
tree
92ec2
author
Scott
committer
Scott
initial commit of my project
34ac2..
commit
size
tree
184ca
parent
98ca9
author
Scott
committer
Scott
fixed bug #1328 - stack overflow under certain
f30ab..
commit
size
tree
0de24
parent
34ac2
author
Scott
committer
Scott
add feature #32 - ability to add new formats to the central
Snapshot A
Snapshot B
Snapshot C

Basic command for VIM

- Vim has 2 modes
  - Command
  - Insert
- Basic command line

1. Lưu file - "**:w**"
2. Thoát khỏi file mà không lưu - "**:q**"
3. Lưu và thoát khỏi file - "**:wq**"
4. Chuyển sang chế độ insert - "**i**"
5. Chuyển sang chế độ command - **ESC**
6. Xóa một từ - "**dw**"
7. Xóa một dòng - "**dd**"
8. Quay trở lại các bước trước - "**u**"



END