

Quản lý mã nguồn với GIT

Nội dung video: nêu lên vấn đề thực tế về việc làm việc với các tập tin thư mục

1. Vấn đề cá nhân:

- ❖ Bạn đang làm việc với **một tập tin** nào đó. Khi muốn quay về trạng thái trước khi chỉnh sửa của tập tin thì bạn sẽ làm gì? → backup file trước khi chỉnh sửa.
 - ❖ Readme.txt 01.07.2017 02.07.2017 03.07.2017
 - ❖ Backup với cách Đặt tên File theo version và ghi chú cho version
 - 01.07.2017 Readme-version1.txt : giới thiệu về tác giả
 - 02.07.2017 Readme-version2.txt : giới thiệu về tác giả, website
 - 03.07.2017 Readme-version3.txt : giới thiệu về tác giả, website, thông tin liên lạc của tác giả
 - ❖ Backup với cách Đặt tên theo thời gian
 - Readme-10am-010717.txt : giới thiệu về tác giả
 - Readme-9pm-020717.txt : giới thiệu về tác giả, website
 - Readme-11am-03072017.txt : giới thiệu về tác giả, website, thông tin liên lạc của tác giả
 - Banner-01.psd banner-02.psd banner-finish.psd banner-finish-02.psd :
- ➔ Yêu cầu quay về file Readme ở thời điểm file chỉ gồm phần tác giả, website → dựa vào ghi chú ta quay về file Readme-version2.txt
- ➔ Kết quả phát sinh: mất thời gian tạo file, dễ nhầm lẫn, số lượng File càng nhiều càng mất thời gian và không đảm bảo sự chính xác

2. Vấn đề làm việc nhóm: tập hợp nhiều cá nhân

- ❖ Bạn và **Gả bàn đối diện** cùng làm việc với file Readme.txt để báo cáo với Sếp (**2 người**)
 - ❖ Ngày 1:
 - Bạn soạn File Readme.txt với 20 dòng văn bản.
 - Cùng ngày hôm đó: Gả bàn đối diện soạn File Readme.txt 30 dòng văn bản
 - Cùng ngày hôm đó: Bạn và hấn nhập File lại và gửi cho Sếp → **File Readme-Finish.txt**
 - ❖ Ngày 2:
 - Bạn soạn tiếp File **Readme-Finish.txt** 30 dòng văn bản
 - Cùng ngày hôm đó: Gả bàn đối diện soạn File **Readme.txt** 30 dòng văn bản
 - Cùng ngày hôm đó: Bạn và hấn nhập File lại và gửi cho Sếp → 2 File không đồng nhất → tìm vấn đề
- ➔ Kết quả phát sinh: 2 bạn mất thời gian tìm và đồng nhất File Readme.txt để gửi cho sếp, 10 bạn cùng làm chung 1 File thì sau, 10 bạn làm cùng 10 File thì sao ??

Nội dung video: giới thiệu về khóa học, cách học và tài liệu học

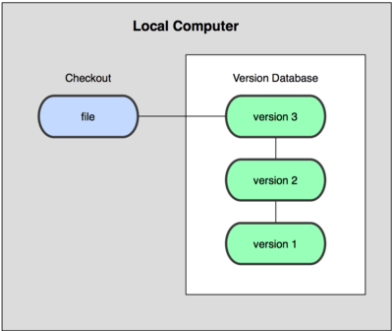
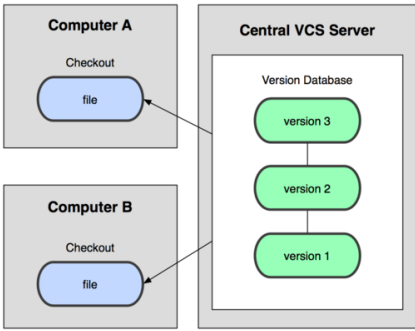
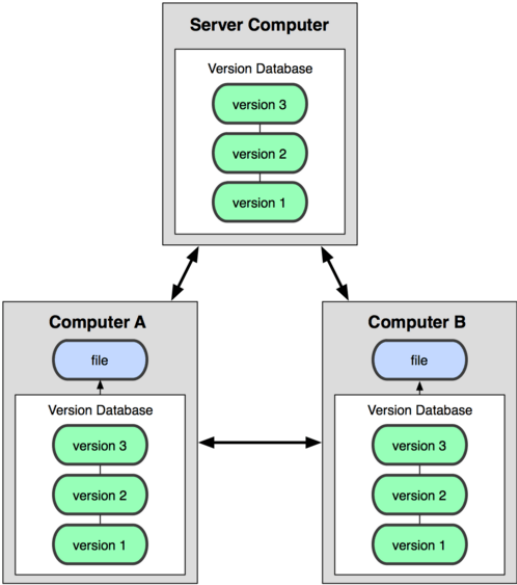
1. Khóa học
2. Đối tượng học & Cách học
3. Tài liệu học tập tham khảo

Nội dung video: Tìm hiểu các hệ thống quản lý phiên bản

1. Hệ thống quản lý phiên bản là gì

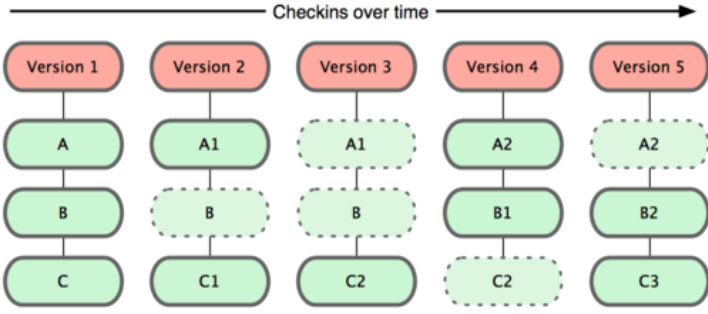
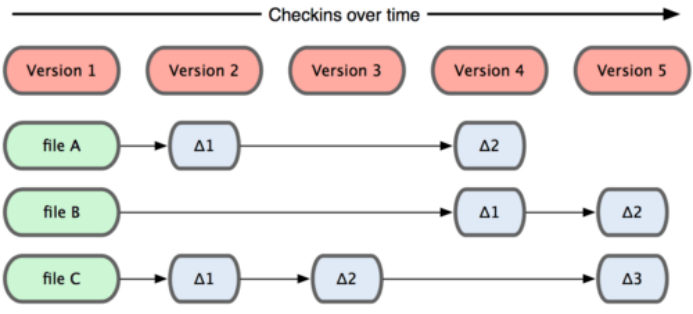
- ❖ Version Control System – VCS: lưu trữ các thay đổi của một hoặc nhiều tập tin theo các mốc thời gian
- ❖ Đối tượng sử dụng: Bất kỳ ai có thao tác trên tập tin (demo dành mã nguồn)
- ❖ Chức năng nổi bật
 - Xem lại các thay đổi đã được thực hiện theo thời gian
 - Xem ai là người thực hiện thay đổi
 - Khôi phục phiên bản cũ của các tập tin

2. Phân loại các hệ thống quản lý phiên bản:

Hệ Thống Quản Lý Phiên Bản Cục Bộ	Hệ Thống Quản Lý Phiên Bản Tập Trung	Hệ Thống Quản Lý Phiên Bản Phân Tán
		
- Version Control System - VCS - Ví dụ: RCS - Máy tính cá nhân Lưu trữ tất cả các sự thay đổi của các tập tin - Mỗi người chỉ biết phiên bản làm việc của mình	- Centralized Version Control System – CVCS - Ví dụ: CVS, Subversion, Perforce - Máy chủ Lưu trữ tất cả sự thay đổi của các tập tin, máy khách có quyền thay đổi các tập tin trên máy chủ - Tất cả thành viên đều biết phiên bản làm việc của nhau	- Distributed Version Control System (DVCS) - Ví dụ: Git , Mercurial, Bazaar hay Darcs - Máy chủ Lưu trữ tất cả sự thay đổi của các tập tin, máy khách có quyền thay đổi các tập tin trên máy chủ - Tất cả thành viên đều biết phiên bản làm việc của nhau - Máy khách sao chép phiên bản mới nhất của các tập tin và sao chép toàn bộ kho chứa

Nội dung video: các thông tin tổng quan về GIT

1. Hệ thống quản lý phiên bản phân tán
2. So sánh GIT và các VCS khác

GIT	VCS
	
<p>- Tại mỗi phiên bản lưu trữ tất cả các files (thay đổi hoặc không thay đổi) (snapshot)</p> <p>- Phần lớn các thao tác trong Git chỉ cần yêu cầu các tập tin hay tài nguyên cục bộ (ngay tại máy tính cá nhân)</p>	<p>- Tại mỗi phiên bản lưu trữ files thay đổi</p> <p>- Các thao tác đòi hỏi phải kết nối đến máy chủ chứa Repository</p>

3. Một số chức năng của GIT

- ❖ Theo dõi sự thay đổi của các tập tin
- ❖ Theo dõi sự thay đổi của các người dùng trên các tập tin
- ❖ Phục hồi trạng thái của các tập tin vào một thời điểm nào đó
- ❖ Hỗ trợ làm việc offline
- ❖ Hỗ trợ rất tốt trong môi trường làm việc nhóm
- ❖ Yêu cầu của các doanh nghiệp hiện nay
- ❖ Cộng đồng hỗ trợ và sử dụng rộng lớn

Nội dung video: cài đặt các phần mềm cần thiết để tìm hiểu GIT

1. Cài đặt GIT

- ❖ Download & Cài đặt
 - Link: <https://git-scm.com/>
- ❖ Kiểm tra cài đặt thành công
 - Git Bash `git --version`
 - CMD `git --version`

2. Cài đặt các phần mềm khác: Editor, IDE

- ❖ Git
- ❖ TortoiseGit
- ❖ SourceTree
- ❖ Git Bash
- ❖ Git Gui
- ❖ Editor: SublimeText, Notepad, Notepad++

Nội dung video: cấu hình GIT cho lần sử dụng đầu tiên

1. Các cấp độ cấu hình

- ❖ Cấu hình chung cho toàn bộ người dùng C:\ProgramData\Git\config
- ❖ Cấu hình cho một người dùng nào đó C:\Users\HaiLan\.gitconfig
- ❖ Cấu hình cho một repository D:\Course\Git\demo\git-one

2. Cấu hình GIT cho lần đầu tiên sử dụng

- ❖ Cấu hình
 - `git config --global user.name hailan`
 - `git config --global user.email hailan@gmail.com`
- ❖ Xem thông tin cấu hình
 - `git config --list`
 - `git config X` (X là tên thuộc tính cần xem giá trị cấu hình)
- ❖ Xem các thông số của câu lệnh X nào đó
 - `git help X`
 - `git X --help`
- ❖ Tùy chỉnh giao diện GishBash cho việc soạn thảo
 - Background
 - Font: size, color

Nội dung video: hệ thống các kiến thức đã được học trong chương đầu tiên “GIT những vấn đề cần biết”

- ❖ Hiểu được lý do sử dụng các VCS và vai trò của VCS
- ❖ Cài đặt thành công môi trường cho việc học tập sau này
- ❖ Tiếp xúc đầu tiên với GIT thông qua thực hiện thành công câu lệnh cấu hình

Nội dung video: tìm hiểu về repository

1) Repository là gì

- ❖ Lưu trữ tất cả các trạng thái của các tập tin, các hành vi của người dùng trên các tập tin này
- ❖ Thường bố trí 1 repository tương đương 1 project mà ta thực hiện
- ❖ Để ghi lại việc thay đổi trạng thái của tập tin vào repository chúng ta thực hiện thao tác gọi là Commit
- ❖ Khi thực hiện commit, trong repository sẽ tạo ra commit (hoặc revision) đã ghi lại sự khác biệt từ trạng thái đã commit lần trước đến trạng thái hiện tại. Mỗi commit phân biệt nhau = 1 id phân biệt (chuỗi ký tự được tự động sinh ra có chiều dài 40 ký tự)

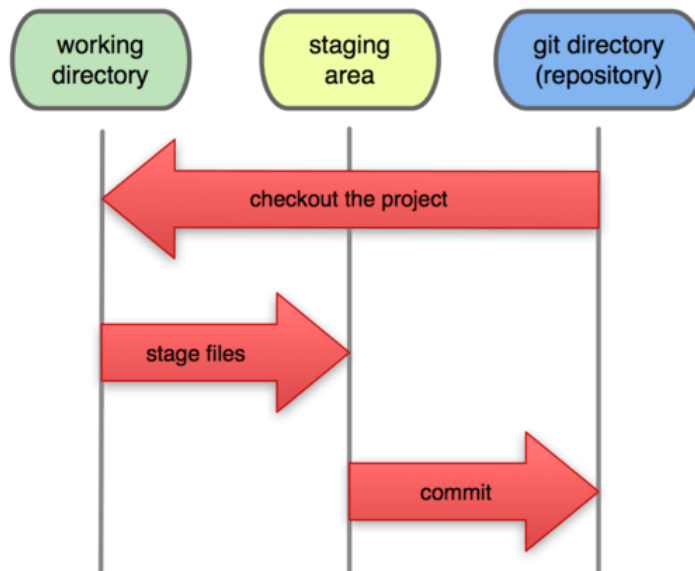
2) Phân biệt Repository

- ❖ Local Repository: repository lưu trữ trên máy tính cá nhân, một người sử dụng
 - ❖ Remote Repository: repository lưu trữ trên hệ thống máy chủ chuyên dụng, chia sẻ project cho nhiều người cùng thực hiện
- ➔ công việc bình thường thì có thể sử dụng local repository và thực hiện trên toàn bộ máy sẵn có
- ➔ công khai nội dung công việc mà bản thân đã làm trên local repository, thì sẽ upload lên remote repository rồi công khai. Thêm nữa, thông qua remote repository cũng có thể lấy về nội dung công việc của người khác.

3) Trạng thái của các tập tin

- ❖ Committed: dữ liệu đã được lưu trữ an toàn trong CSDL
 - ❖ Modified: đã thay đổi tập tin nhưng chưa commit vào cơ sở dữ liệu
 - ❖ Staged: đánh dấu sẽ commit phiên bản hiện tại của một tập tin đã chỉnh sửa trong lần commit sắp tới
- ➔ Sinh ra 3 vùng làm việc đối với một dự án sử dụng Git:
- ❖ Git directory: nơi Git lưu trữ các "siêu dữ kiện" (metadata) và cơ sở dữ liệu cho dự án của bạn
 - ❖ Working directory: các tập tin nằm trong dự án của bạn
 - ❖ Staging area (index): các tập tin sẵn sàng được đưa lên Git directory

Local Operations



4) Quy trình để commit một tập tin

- ❖ Working directory được add vào vùng Staged (index), vùng staged được đẩy vào repository

Nội dung video: Khởi tạo local repository và đưa file readme.txt vào repository

1. Tạo folder git-one
2. Git bash here ngay tại folder này
3. git init
4. git status kiểm tra trạng thái tập tin (ở working tree hay index)
5. Add readme.txt
6. git status
7. git add readme.txt thêm tập tin vào vùng index
8. git status
9. git commit -m "Init project"
10. git status

Nội dung video: Cập nhật file readme.txt vào gửi vào repository

Case 1: Edit readme.txt

1. `git status`
2. `git add readme.txt`
3. `git status`
4. `git commit -m "Commit 2"`
5. `git status`

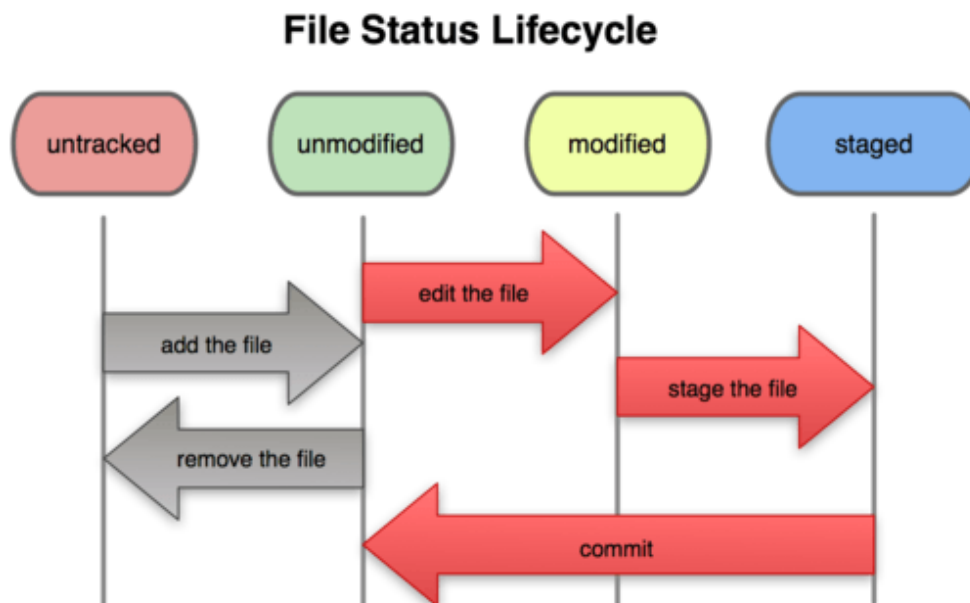
Case 2: Edit file readme.txt

1. `git status`
2. `git commit -m "C2 – Case2 Edit readme"`
3. `git commit -a -m "C2 – Case2 Edit readme"`
4. `git status`
5. `git log`

Nội dung video: Add nhiều file vào vùng Staged cùng một lúc

1. `Add author.txt`
2. `add folder documents`
3. `add file requirement.txt`
4. `git status`
5. `git add *`
6. `git status`
7. `git commit -m "C3 – Multi File"`
8. `git status`
9. `git log`

Nội dung video: Tìm hiểu kỹ hơn về vòng đời của các tập tin



1. Mỗi tập tin trong thư mục làm việc có thể ở một trong hai trạng thái : tracked hoặc untracked
2. Tập tin tracked là các tập tin đã có mặt trong ảnh (snapshot) trước;
 - a) Unmodified
 - b) Modified
 - c) Staged
3. Tập tin untracked là các tập tin còn lại - bất kỳ tập tin nào trong thư mục làm việc của bạn mà không có ở ảnh (lần commit) trước hoặc không ở trong khu vực tổ chức (staging area)
4. Ví dụ với project hiện tại
 - a) Tất cả các tập tin đang ở trạng thái unmodified và đang được theo dõi tracked
 - b) Tạo mới một tập tin lifecycle.txt có nội dung lifecycle
 - c) Tập tin này ở trạng thái untracked
 - d) `git add *`
 - e) Tập tin này đã được tracked
 - f) `git commit -m "Life cycle"`
 - g) Tập tin này ở trạng thái unmodified
 - h) Edit tập tin này nó ở trạng thái modified
 - i) Commit nó sang trạng thái staged

Nội dung video:

1. Giá sử project hiện tại đã clean
2. Edit author.txt
3. Git status
4. author.txt nằm trong trạng thái modified và chưa sẵn sàng commit
5. git add *
6. git status
7. author.txt nằm trong trạng thái modified và sẵn sàng commit
8. Edit author.txt
9. Git status
10. author.txt vừa nằm trong trạng thái modified chưa sẵn sàng commit và trạng thái modified đã sẵn sàng
11. git commit điều gì xảy ra

Nội dung video: : Tạo file report.txt, add file vào vùng staged, bây giờ làm sao xóa file này ra khỏi vùng stage

1. Tạo file report.txt
2. git status
3. git add *
4. git status
5. git reset HEAD report.txt Xóa file ra khỏi vùng staged
6. Hoặc git rm --cached <file> Xóa file ra khỏi vùng staged
7. git status

Nội dung video: Sử dụng .gitignore để không theo dõi các tập tin được chỉ định (Từ chối một số file được thêm vào vùng Staged)

1. Add me.txt, libs/string.txt, libs/image.txt nhưng không muốn commit
2. git status
3. Add file .gitignore bằng câu lệnh touch .gitignore
4. Cập nhật nội dung cho file .gitignore

```
1 # Folder libs
2 /libs
3
4 // File me.txt
5 me.txt
6
7 // File *.xml
8 *.xml
9
```

5. Không commit các file *.xml
6. Công cụ online <https://www.gitignore.io/>
7. Quy tắc cho các mẫu có thể sử dụng trong .gitignore như sau:
 - # ghi chú được bỏ qua
 - Không có nội dung được bỏ qua
 - abc/ thư mục abc
 - !abc.txt không phải tập tin abc.txt
 - Và áp dụng quy tắc biểu thức chính quy (regular expression)

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

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

Nội dung video: các trường hợp xóa tập tin

1. Tạo file abc.txt và xóa file này → đang ở vùng working tree
 - a) Cách 1: Xóa như tập tin bình thường ở hệ điều hành
 - b) Cách 2: Xóa câu lệnh `rm abc.txt`
2. Xóa tập tin trong GIT
 - a) Xóa khỏi danh sách tracked (xóa khỏi vùng staged) `git rm <filename>`
 - b) Sau đó `commit git commit`
3. Tập tin readme.txt đã được staged, edit nội dung của nó cho nó về trạng thái modified, chưa thêm vào staged, xóa thế nào
 - a) `git rm -f readme.txt`
4. Tập tin account.txt đã được staged, ko edit nội dung gì cả, xóa thế nào
 - a) `git rm account.txt`
 - b) `git commit`
5. Tập tin readme.txt đã được staged, edit nội dung của nó cho nó về trạng thái modified, thêm vào staged, xóa thế nào

Nội dung video: đổi tên tập tin

1. Tạo file task-01.txt
2. Đưa file này vào vùng staged → git status → git add * → git status
3. Đổi tên task-01.txt thành task-02.txt
4. git mv task-01.txt task-02.txt

Nội dung video: quay về trạng thái trước của một tập tin

1. Vấn đề: bạn chỉnh sửa file readme.txt, sau đó phát hiện rằng mình không muốn commit file này và muốn quay về trạng thái trước khi chỉnh sửa của file
2. `git checkout -- readme.txt`

Nội dung video: gộp commit đã thực hiện

1. Vấn đề: Bạn nhận task A yêu cầu phải tạo 2 file task-01.txt và task-02.txt, sau đó commit 2 file này lên. Bạn chỉ tạo 1 file task-01.txt và commit nó, bạn chợt nhớ ra là chưa tạo task-02.txt. Bây giờ nếu tạo task-02.txt và commit sẽ có đến 2 commit, trong khi yêu cầu chỉ có 1 commit
2. Add file task-01.txt → git status → git add * → git commit -m "task A"
3. Add file task-02.txt → git status → git add * → git commit --amend

Working Tree	commit	Local Repo
--------------	--------	------------

A	gửi thư	B
---	---------	---

A viết thư X,Y gửi B,

A viết thư (X, Y) gửi B

Nội dung video: xem các commit đã thực hiện

1. Sử dụng cơ bản

- a) `git log` hiển thị lịch sử commit
- b) `gitk` hiển thị lịch sử commit bằng GUI

2. Tham số

- a) `git log -p -3` hiển thị 3 commit gần đây nhất
 - b) `git log --stat` hiển thị lịch sử commit với các thông tin tóm tắt
 - c) `git log --pretty=oneline` hiển thị lịch sử commit với mỗi commit là 1 dòng
 - d) `git log --pretty=format:"%h - %an, %ar : %s"`
- ➔ Xem tài liệu để hiểu hơn về các param

Nội dung video: hệ thống các kiến thức đã học

#	Câu lệnh	Ý nghĩa
1	git init	Khởi tạo một repository (local)
2	git status	Kiểm tra trạng thái các files của repository
3	git add <fileName>	Thêm fileName vào trạng thái Staged
4	git add *	Thêm tất cả tập tin ở trạng thái Working vào trạng thái Staged
5	touch .gitignore	Tạo file .gitignore
6	git commit --m "Message"	Commit files nằm trong trạng thái Staged kèm theo thông điệp
7	git commit -a --m "Message"	Commit files đã hoặc chưa nằm trong trạng thái Staged kèm theo thông điệp
8	git log	Xem log các commit trong Repository (tham khảo thêm các param)
9	git reset HEAD <filename>	Xóa fileName ra khỏi trạng thái Staged (file vẫn còn tồn tại)
10	git rm --f <filename>	Xóa fileName ra khỏi trạng thái Staged và xóa khỏi ổ cứng (file không còn tồn tại)
11	git rm --cached <filename>	Xóa fileName ra khỏi trạng thái Staged (file vẫn còn tồn tại)
12	git checkout	Quay về trạng thái trước của một tập tin
13	git mv fileName1 fileName2	Đổi tên fileName1 thành fileName2 (chuyển nội dung fileName1 sang fileName2, sau đó xóa fileName1)
14	git commit --amend	Nhập 1 commit về commit trước đó

Nội dung video: yêu cầu về tình huống thực hành

1. Khởi tạo một Local repo có tên mweb
2. Tích hợp bootstrap và hiển thị nội dung “Xin chào, sản phẩm đầu tiên của tôi đây” (edit)
3. Commit lần đầu tiên “Init Project”
4. Tạo folder chứa ghi chú notes và không muốn git quản lý folder này
5. Task “Edit theme 123”: Commit 3
 - “Xin chào, sản phẩm đầu tiên của tôi đây” → Hello, First Project
 - **Style.css → change background**

Nội dung video: tìm hiểu các kiến thức xoay quanh Remote Repository

1. Phân biệt Repository

- Local Repository: repository lưu trữ trên máy tính cá nhân, một người sử dụng
- Remote Repository:** repository lưu trữ trên hệ thống máy chủ chuyên dụng, chia sẻ project cho nhiều người cùng thực hiện

2. Thao tác thường gặp

- Clone: sao chép remote repo. Tải về toàn bộ nội dung của remote repo, và tạo thành local repository
- Push: đưa dữ liệu từ Local repo lên remote repo.
- Pull cập nhật local repository từ remote repository

➔ Hình dung đơn giản

Remote Repository → clone → Local Repository → push (upload) Remote Repository
← pull (download) Remote Repository

Nội dung video: đăng ký một remote repository

1. Create Account
2. Create Repository

Nội dung video: cách clone (sử dụng, sao chép, tiếp tục phát triển, ...) một Remote repository

1. Sử dụng: git clone <repository>

- Clone <repository> từ Remote repo và đặt vào folder trùng tên với <repository>
- git clone <https://github.com/luutruonghailan/course-java.git>

2. Sử dụng: git clone <repository> [<directory>]

- Clone <repository> từ Remote repo và đặt vào folder directory
- git clone <https://github.com/luutruonghailan/course-java.git> abc

3. Xem thông tin Remote repository hiện tại

- git remote -v Xem thông tin Remote repository hiện tại
- git remote show <name> Xem thông tin <name> hiện tại

Nội dung video: Add một Local Repository lên Remote Repository tại <url> và đặt tên nó là <shortname>

1. Tại thư mục chứa Local Repository thực hiện
 - Cú pháp git remote add <name> <repository>
 - git remote add hailanjava https://github.com/luutruonghailan/test01.git
2. Đổi tên <name1> thành <name2>
 - git remote rename <name1> <name2>
3. Xóa <name>
 - a) git remote remove <name>

Nội dung video: các thao tác cơ bản trên Remote Repository

1. Kiểm tra sự thay đổi trên Remote Repository: git fetch
2. Kiểm tra sự thay đổi trên Remote Repository và merge vào phiên làm việc hiện tại: git pull
 - git pull <name> <branch> có thể được rút gọn chỉ còn git pull
3. Đẩy dữ liệu lên remote repository
 - git push <name> <branch>

Nội dung video: sử dụng alias để đơn giản hóa câu lệnh

1. Cú pháp

- `git config --global.config alias.st status`

Nội dung video: hệ thống các kiến thức đã học

#	Câu lệnh	Ý nghĩa
1	git clone <repository> [<directory>]	Clone Remote repository <repository> và đặt vào folder [<directory>]
2	git remote -v	Xem thông tin Remote repository hiện tại
3	git remote show <name>	Xem thông tin <name> hiện tại
4	git remote add <name> <repository>	Add một Local Repository lên Remote Repository tại <repository> và đặt tên nó là <name>
5	git remote rename <name1> <name2>	Đổi tên <name1> thành <name2>
6	git remote remove <name>	Xóa remote <name>
7	git pull <name> <branch>	Kiểm tra sự thay đổi trên Remote Repository và merge vào phiên làm việc hiện tại
8	git push <name> <branch>	Đẩy dữ liệu lên remote repository
9	git config --global.config alias.st status	Tạo alias

Nội dung video: các thao tác cơ bản trên Remote Repository

1. Chuẩn bị một project có tên *mweb* và hiển thị ra màn hình giá trị *Hello mweb*

2. Khởi tạo Local repo cho *mweb*

- Tạo local repository cho project này: `git init`
- Add các file vào vùng staged: `git add *`
- Commit các file này vào local repository: `git commit -m "init project"`

3. Đưa Local repo *mweb* lên Github (**remote-mweb**)

- Tạo repob *remote-mweb* trên Github `Thủ công`
- Add Local Repository lên Remote Repository `git remote add origin https://github.com/luutruonghailan/remote-mweb.git`
- Kiểm tra thông tin của remote `git remote -v`
- Đưa các data từ local lên remote `git push origin master`

4. Clone Github về với folder *mabc*

- Clone remote `git clone https://github.com/luutruonghailan/remote-mweb.git mabc`
- Di chuyển vào project `cd mabc`

5. Cập nhật nội dung cho **local repo mabc**

- Edit file *index.html*, thay **Hello, mweb!** Thành **Hello, mweb!**
- Add các file vào vùng staged: `git add *`
- Commit các file này vào local repository: `git commit -m "Fix index.html"`
- Kiểm tra trong **local repo mweb** thì tập tin *index.html* đã được chỉnh sửa chưa → chưa → update lại từ **remote-mweb** → vẫn chưa → lý do → do chưa đưa **local repo mabc** lên trên **remote-mweb**
 - `git pull origin master`

6. Push **local repo mabc** lên **remote-mweb**

- Kiểm tra có sự thay đổi gì từ remote- hay không `git pull origin master`
- Công bố sự thay đổi `git push origin master`
- Kiểm tra trong **local repo mweb** thì tập tin *index.html* đã được chỉnh sửa chưa → chưa → update lại từ **remote-mweb** → đã có
 - `git pull origin master`

Nội dung video: tìm hiểu các kiến thức về branch

1. Tại sao lại sử dụng Branch

- Website abc.com là một website tin tức đang vận hành. Khách hàng yêu cầu bổ sung:
 - Chức năng tin tức dạng video cho abc.com
 - Chức năng subscribe cho abc.com
- Lúc này chúng ta thấy abc.com vừa thực hiện phát triển các chức năng cũ (fix, update) và thực hiện phát triển thêm 2 chức năng mới → 1 nhóm task update, 1 nhóm task video, 1 nhóm task subscribe → nhiều phiên bản → mà cùng làm trên cùng 1 source code → rất dễ phát sinh vấn đề → branch ra đời
- Mỗi nhóm làm trên một branch khác nhau, các branch hoàn toàn độc lập nhau, sau khi đã hoàn thiện task của mình rồi thì sẽ nhập branch vào branch chính (branch master)

2. Phân loại Branch

- Long-Running Branches (Integration branch) branch tích hợp (**chức năng**): chức năng mới
- Topic Branches: bug, issues Branch chủ đề sẽ tạo ra bằng cách phân branch từ branch tích hợp đã ổn định, khi đã hoàn thành xong công việc sẽ sử dụng đưa vào branch tích hợp.

Nội dung video: tìm hiểu thao tác tạo và chuyển nhánh làm việc

1. Xem danh sách nhánh

- git branch
- nhánh nào có dấu * là nhánh chúng ta đang làm việc

2. Tạo nhánh task-login

- git branch **task-login**
- git branch

3. Chuyển nhánh làm việc

- git checkout **task-login**
- *Trước khi chuyển nhánh phải đảm bảo nhánh làm việc phải “nothing to commit, working tree clean”*

4. Demo làm việc trên nhánh

- git branch
- git checkout **task-login**
- tạo file login.html và commit trên nhánh **task-login**
- Chuyển nhánh master và xem file login.html có tồn tại hay không?
- git branch -v xem commit mới nhất ở mỗi nhánh

5. Mẹo

- git checkout -b <branchname> tạo branchname và chuyển đến branchname

```
Commit sdksldhls      ----- Master ---- task-logout
                        ----- task-login ---- login.html ---- bug-01
```

Nội dung video: tìm hiểu cách merge các nhánh làm việc lại với nhau

1. Di chuyển đến nhánh master

- `git merge task-login`

2. Xem nhánh đã merge hoặc chưa merge vào nhánh hiện tại

- `git checkout -b fixbug` tạo nhánh fixbug và di chuyển đến nhánh này
- `git branch --merged` xem các nhánh đã merge vào nhánh hiện tại
- `git branch --no-merged` xem các nhánh chưa merge vào nhánh hiện tại
- Edit file `login.html` → `commit`
- `git checkout master` di chuyển về nhánh master
- `git branch --merged` ra kết quả fixbug
- `git branch --no-merged` ra kết quả task-login

Nội dung video: tìm hiểu cách merge các nhánh làm việc lại với nhau

1. Đổi tên nhánh

- `git branch -m <oldname> <newname>` đổi tên nhánh

2. Xóa nhánh

- `git branch -d <branchname>` xóa nhánh <branchname> và nhánh này đã được merge
- `git branch -D <branchname>` xóa nhánh <branchname> và nhánh này chưa được merge

Nội dung video: tìm hiểu các kiến thức về việc sử dụng tag

1. Tại sao lại sử dụng tag

- Đặt tên cho 1 commit đặc biệt nào đó (thời điểm phát hành) để dễ dàng truy xuất đến nó

2. Phân loại Tag

- Annotated Tag: đặt tag cho commit, thông tin: tên, đính kèm tên và email của người tag, thời gian đặt tag, miêu tả của tag, ... (sử dụng nhiều)
- Lightweight Tag: đặt tag cho commit, thông tin: tên (Local repo)

3. Tạo tag cho commit cuối cùng

- `git tag <tagname>` Tạo mới một Lightweight Tag có tên là tagname
- `git tag -a <tagname> -m "Message"` Tạo mới một Annotated Tag có tên là tagname cùng với thông điệp

4. Xem danh sách các tag & Thông tin tag & Xóa tag

- `git tag` Xem tất cả các tag hiện có
- `git tag -n` Xem tất cả các tag hiện có và phần miêu tả của tag
- `git show tagname` Xem thông tin tagname (demo với 2 loại loại)
- `git tag -d tagname` Xóa tag có tên tagname

5. Tạo Tag cho một commit bất kỳ

- `git log --pretty=oneline` Xem ID của các commit trước đó, chọn ID của commit cần gắn tag, giả sử ta có X
- `git tag -a <tagname> <X> -m <message>` Thêm annotated tag vào commit có ID là X
- **TIP: Google git log pretty colors (<https://coderwall.com/p/euwpig/a-better-git-log>)**

6. Nhập tag vào branch

- `git checkout -b <branchname> <tagname>` Tạo một nhánh mới branchname từ tag đã tồn tại <tagname>
 - `git checkout -b finish v03`

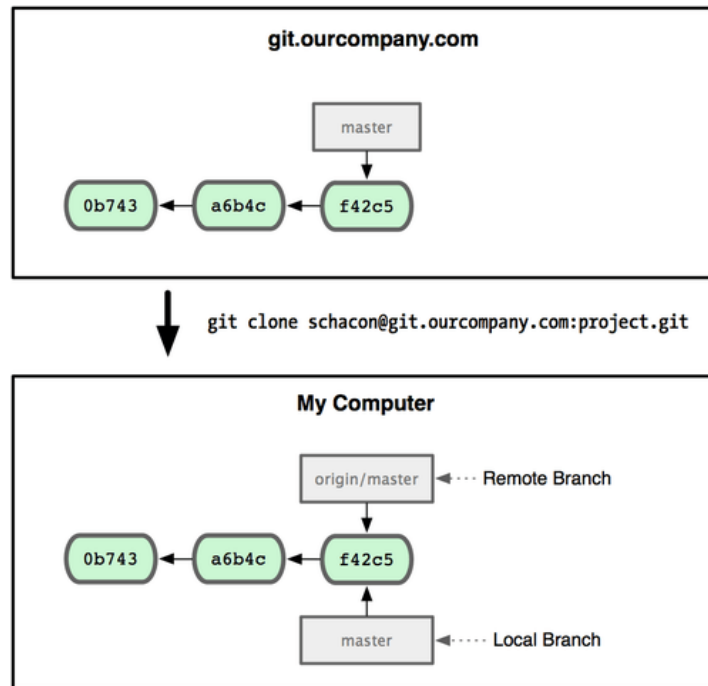
7. Đưa tag lên Remote Repo

- `git push <name> <tagname>` Thêm <tagname> vào remote <name>
- `git push <name> --tags` Thêm tất cả tag vào remote <name>
- `git push <name> -d <tagname>` Xóa tagname tại Remote repository có tên name

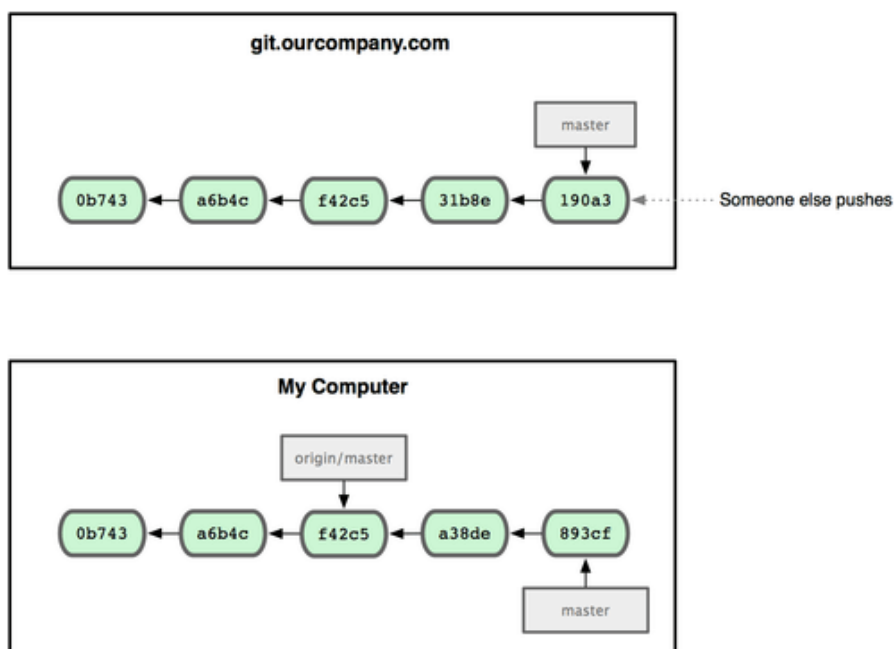
Nội dung video: tìm hiểu các thao tác trên Remote branch

1. Tìm hiểu Remote Branch

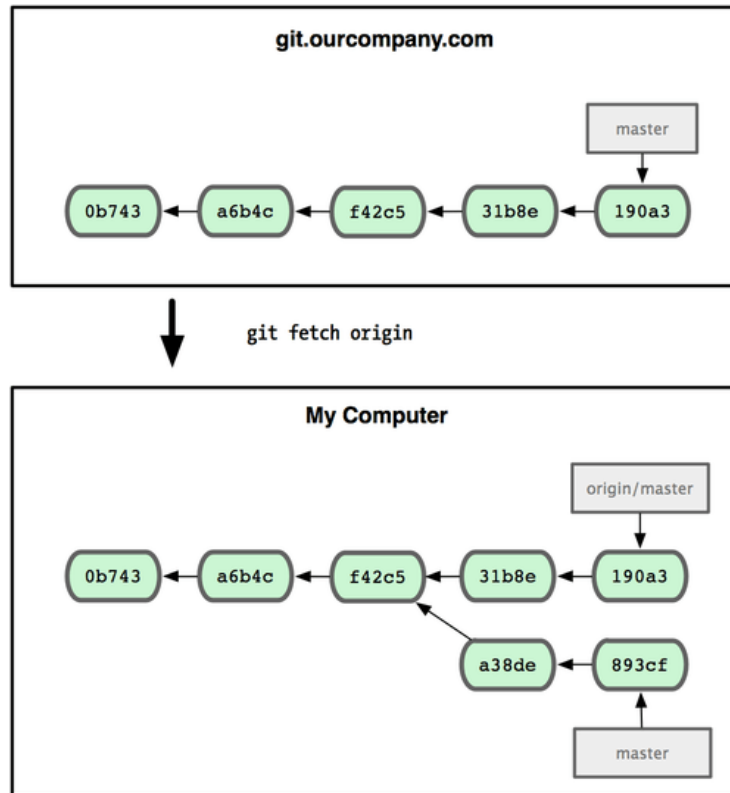
- Hình 1: Một bản sao Git cung cấp cho bạn nhánh master riêng và nhánh origin/master trở tới nhánh master của origin.



- Hình 2: Làm việc nội bộ và ai đó đẩy lên máy chủ khiến cho lịch sử thay đổi khác biệt nhau.



- Hình 3: Lệnh **git fetch** cập nhật các tham chiếu từ xa.



2. Xem danh sách các nhánh ở Local Repo và Remote Repo

- `git branch -a`

3. Đẩy nhánh từ Local Repo lên Remote Repo

- `git remote push <remote_name> <branch>`

4. Xem danh sách các nhánh trên Remote Repo

- `git remote show <remote_name>`
- `git ls-remote <remote_name>`

5. Chuyển đổi làm việc sang nhánh remote

- `git checkout <remote_name>/<branch_name>`

6. Clone một Branch chỉ định từ Remote repo

- `git clone -b <branch> <remote_url>`

7. Xóa một Branch từ Remote repo

- `git push origin :<branch_name>`

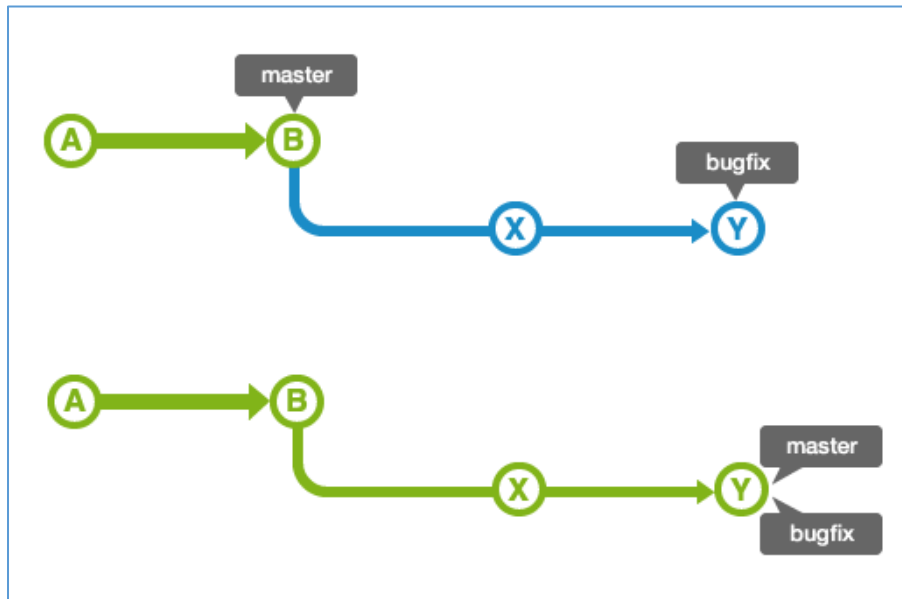
Nội dung video: tìm hiểu cách sử dụng Git cheat sheet

- Link: <https://github.com/luutruonghailan/cheat-git.git>

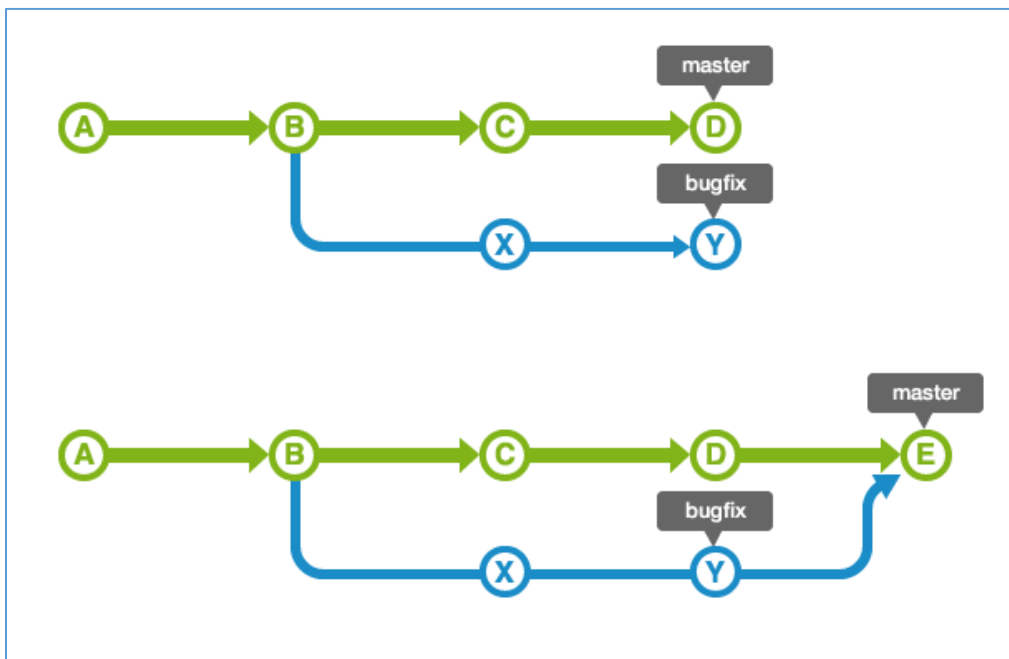
Nội dung video: tìm hiểu các thao tác trên Remote branch

1. Phân nhánh & Tích hợp nhánh

- Xem lại mục “3.2 Phân Nhánh Trong Git - Cơ Bản Về Phân Nhánh và Tích Hợp” và hiểu rõ các khái niệm **nhánh, phân nhánh, fast forward**,
- Fast-forward (chuyển tiếp nhanh): lịch sử của branch bugfix sẽ bao gồm tất cả lịch sử của branch master

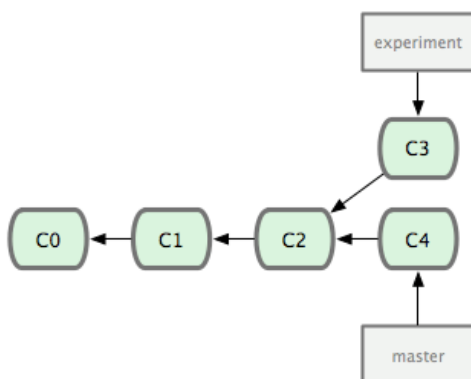


- Non fast-forward:

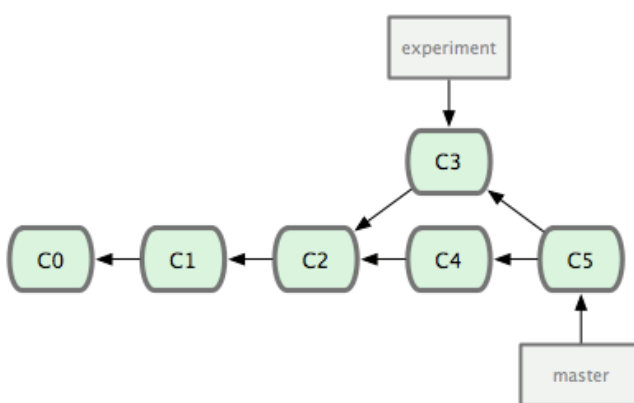


2. Rebase

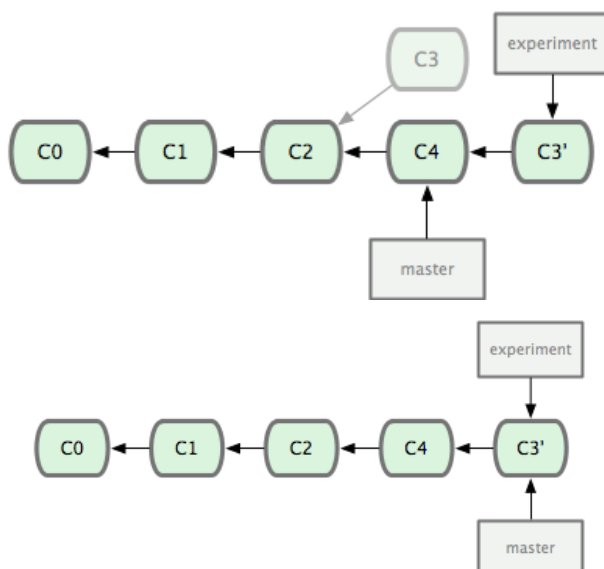
- Trong Git có 2 cách chính để tích hợp nhánh: merge và rebase
- Phân nhánh lần đầu tiên

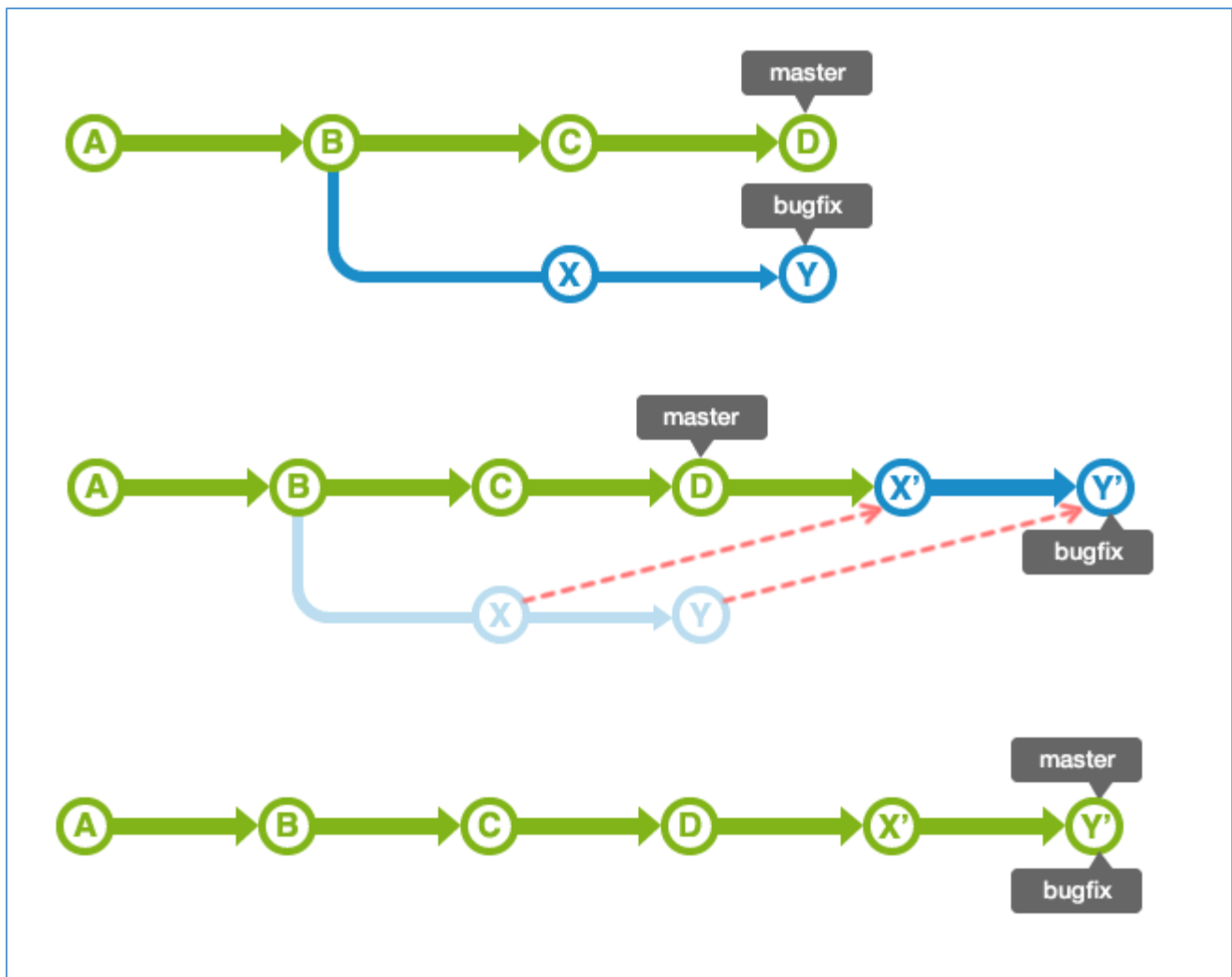


- Merge



- Rebase





3. Tích hợp nhánh sử dụng Rebase

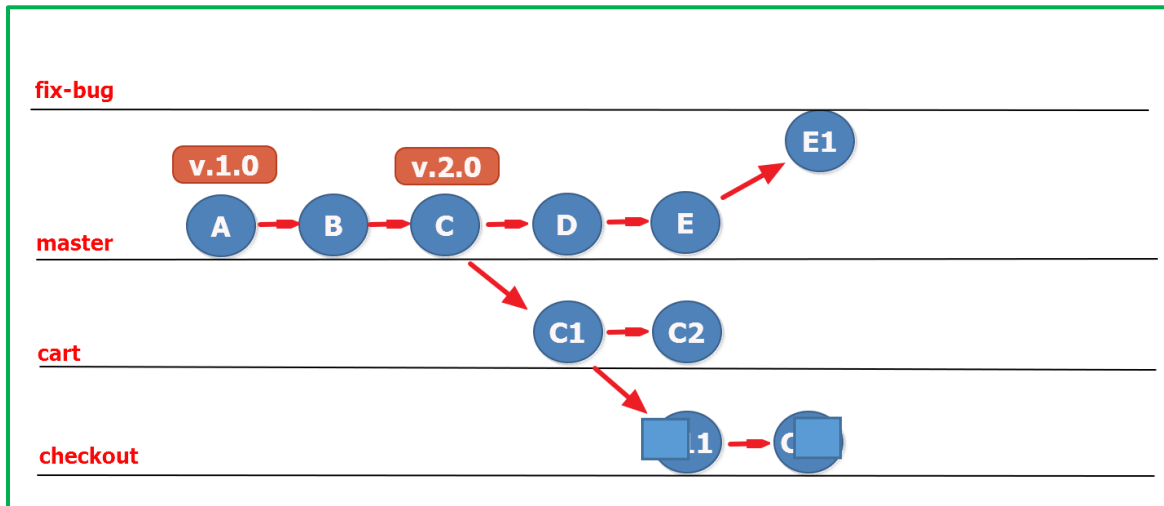
- Chuẩn bị một Loca repo mô phỏng hình vẽ trên
- Sử dụng merge
- Sử dụng rebase

4. Câu hỏi: So sánh 2 phương pháp gộp nhánh: merge vs rebase

Nội dung video: hệ thống các kiến thức đã học, xem git cheat sheet

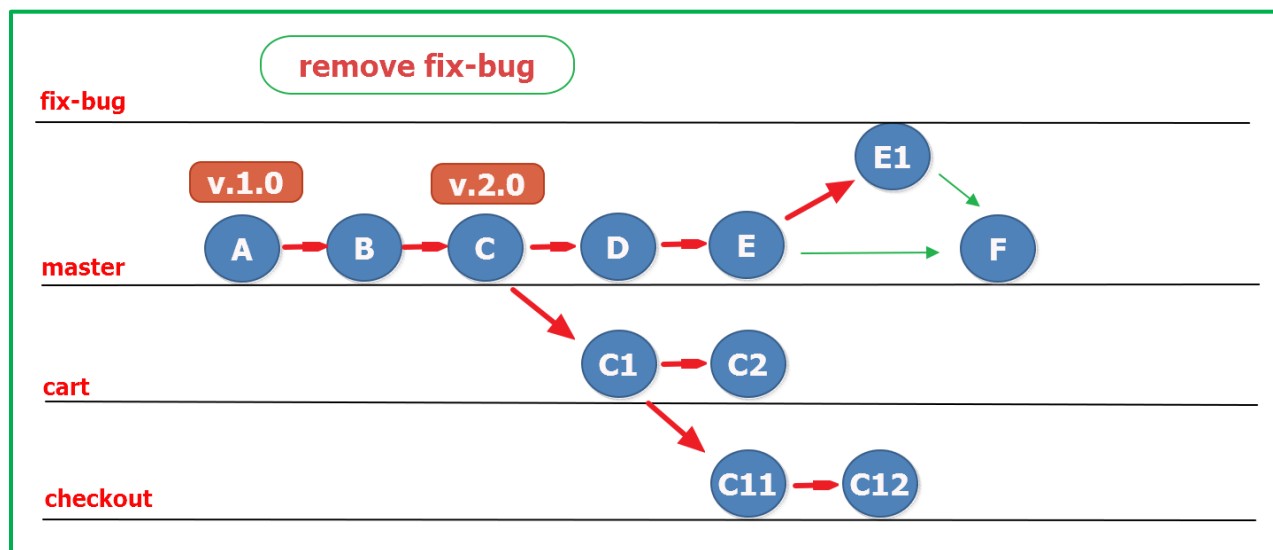
Nội dung video: thực hành ôn tập lại các kiến thức đã học

1. Xây dựng project được GIT quản lý với lịch sử commit và phân nhánh như hình vẽ (Local Repo)

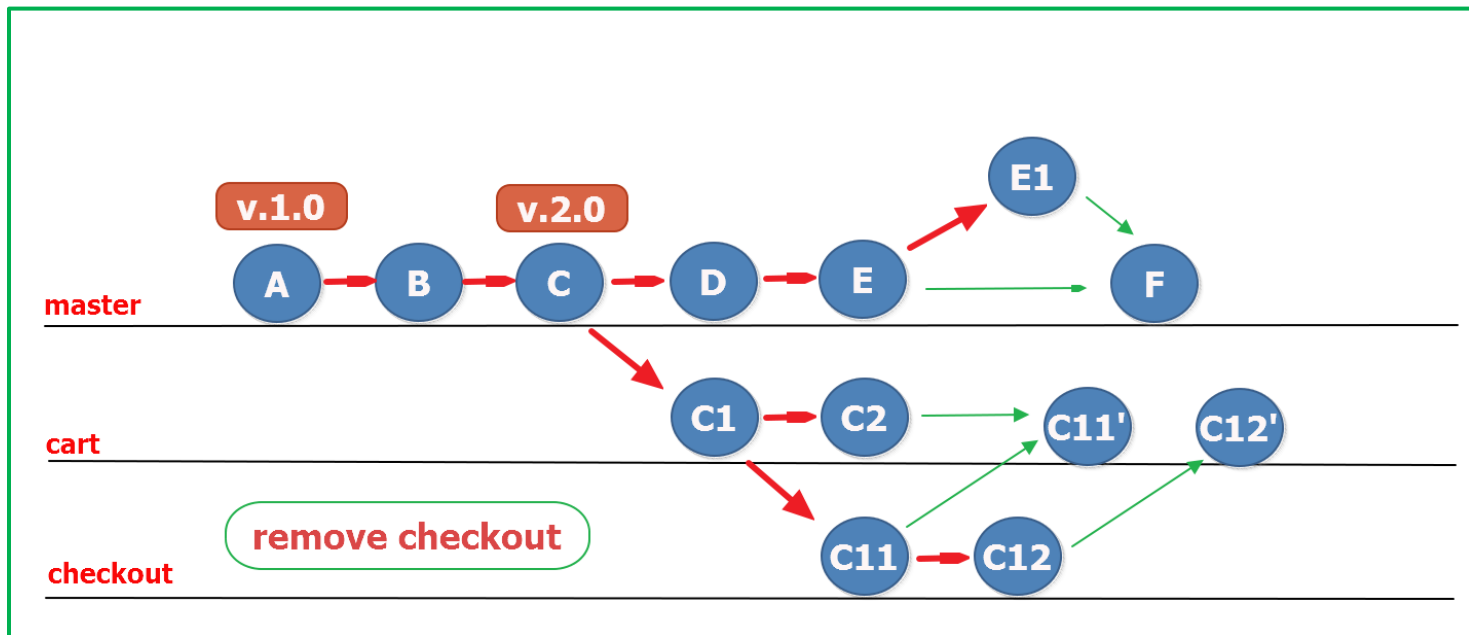


- Tạo tag v.1.0 sau khi commit A
- Hoàn thiện yêu cầu phát hiện quên đặt tag cho commit C
- Đặt tên nhánh sai **checout** và tìm cách sửa lại cho đúng

2. Tạo một Remote Repo trên Github có tên là exe-p4
3. Push nhánh cart và master lên Remote Repo
4. Push tất cả các tag lên Remote Repo
5. Nhập nhánh master và fix-bug và tạo ra một commit mới

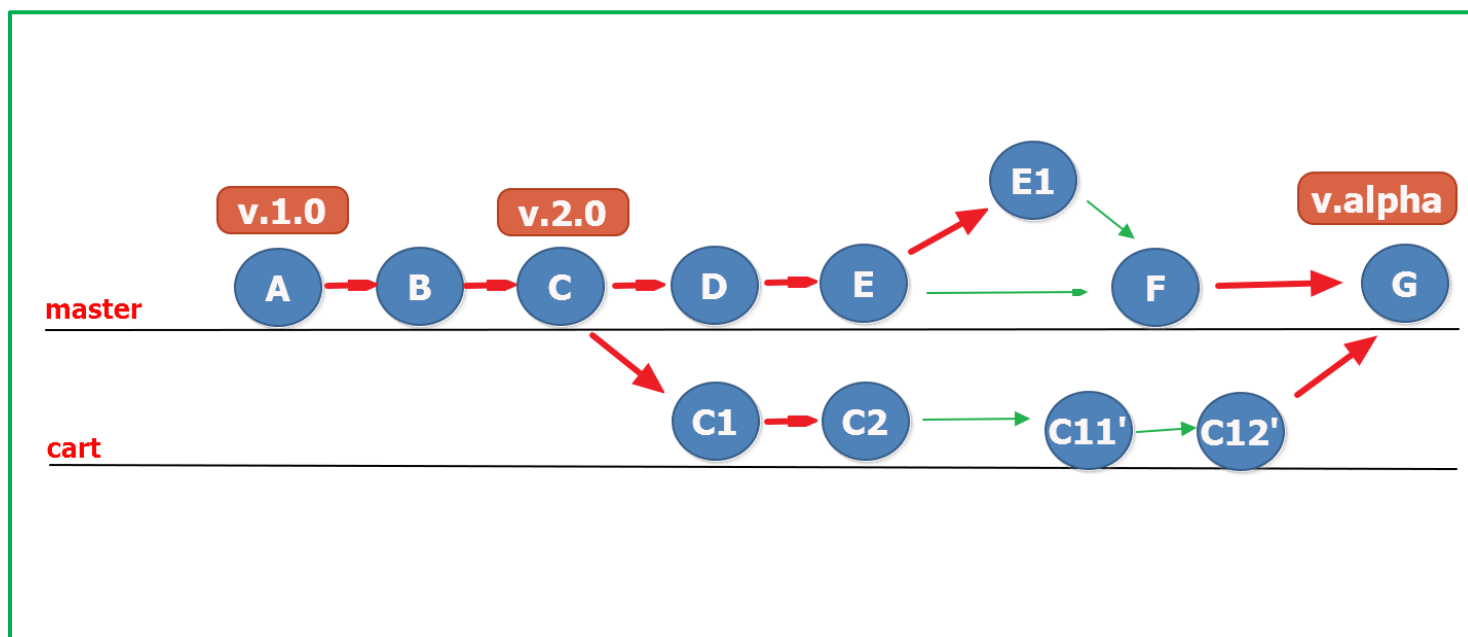


6. Rebase nhánh



7. Push nhánh cart lên remote

8. Nhập nhánh cart và master sau đó đánh dấu tag fisinh và push lên Remote



Nội dung video: các trường hợp so sánh tập tin với nhau

1. Cài đặt và cấu hình phần mềm hỗ trợ:

- DiffMerge https://sourcegear.com/diffmerge/webhelp/sec_git_windows_github.html
- p4merge <https://gist.github.com/dgoguerra/8258007>

2. Các trường hợp so sánh

- 1 tập tin có thể tồn tại ở 4 vùng sau: Working Directory, Staging Area, Local Repo(.git folder) và Remote Repo
- Staging vs Working (L-R) git diff git difftool
- Local Repo vs Working (L-R) git diff HEAD git difftool HEAD
- Local Repo vs Staging Area (L-R) git diff --staged HEAD git difftool --staged HEAD
git diff --cached
- Local Repo vs Remote Repo (L-R) git diff master origin/master ...

	Working Directory	Staging Area	Local Repo (.git)	Remote Repo
File Readme.txt	init readme local repo working tree	init readme local repo staging area	init readme local repo	init readme

- Comit vs Commit git diff comitID commitID git difftool
- Tag vs Tag git diff tagname vs tagname

	Commit 1	Commit 2	Tag 1	Tag 2
File Readme.txt	Commit 1	Commit 2	Tag 1	Tag 2

Nội dung video: các trường hợp so sánh tập tin với nhau

1. Tình huống:

- Bạn đang làm việc trên nhánh phụ, có 1 số tính năng nhưng chưa hoàn thiện để commit, đi ra ngoài → đặt code ở đâu để an toàn → sử dụng không gian tạm stash (phạm vi nội bộ - Local Repo)
- Stash: lưu lại các thay đổi chưa commit (khi bạn muốn đổi sang 1 branch khác mà lại đang làm dở ở branch hiện tại)
- Tài liệu <https://git-scm.com/book/en/v2/Git-Tools-Stashing-and-Cleaning>

2. Thực hành

- git stash lưu toàn bộ nội dung công việc đang làm dở
 - Sử dụng git stash bao nhiêu lần tùy thích, mỗi lần Git sẽ lưu toàn bộ lần thay đổi đó như 1 phần tử trong Stack.
- Xem lại danh sách các lần lưu thay đổi
 - git stash list
 - git stash list -p
 - git stash show stash@{1}
- Áp dụng thay đổi từ stash:
 - git stash apply stash@{1}
 - git stash pop [stash@{1}]
- Xóa các thay đổi không cần thiết
 - git stash drop stash@{1}
 - git stash clear

Nội dung video: tìm hiểu Conflict và Xử lý conflict

1. Conflict:

- Xung đột, đụng độ, mâu thuẫn, đụng code ... có 2 sự thay đổi trong một dòng trên cùng một tập tin và GIT không biết lựa chọn sử dụng dòng lệnh nào
- Khi làm việc nhóm: 2 người cùng chỉnh sửa một file, một dòng code, khi đồng bộ sẽ xảy ra conflict
- Chuyển branch, thao tác trên cùng một file sẽ xảy ra conflict.

2. Xử lý conflict (merge)

- Branch
 - Master Commit 1: init project
 Commit 2: master - fix readme.txt
 Commit 3: video - fix readme.txt
 Commit 4: Finish merge
 - Video Commit 1: init project
 Commit 2: video - fix readme.txt
- Remote

3. Xử lý conflict (rebase)

- `git rebase --continue`
- `git rebase --abort`
- `git rebase --skip`

4. Xử lý conflict – Sử dụng GUI

- Cấu hình mergetool

```
git config --global merge.tool p4merge
git config --global mergetool.p4merge.path "C:/Program Files/Perforce/p4merge.exe"

git config --global mergetool.p4merge.cmd "'C:/Program Files/Perforce/p4merge.exe' \
$PWD/$LOCAL $PWD/$BASE $PWD/$REMOTE $PWD/$MERGED"
git config --global mergetool.p4merge.cmd "p4merge $BASE $LOCAL $REMOTE $MERGED"
```

- Thực hiện mergetool

5. Xử lý conflict – Sử dụng GUI

- Cấu hình mergetool

```
git config --global merge.tool p4merge
git config --global mergetool.p4merge.path "C:/Program Files/Perforce/p4merge.exe"

git config --global mergetool.p4merge.cmd "'C:/Program Files/Perforce/p4merge.exe' $PWD/$BASE
$PWD/$LOCAL $PWD/$REMOTE $PWD/$MERGED"
```

- Thực hiện mergetool

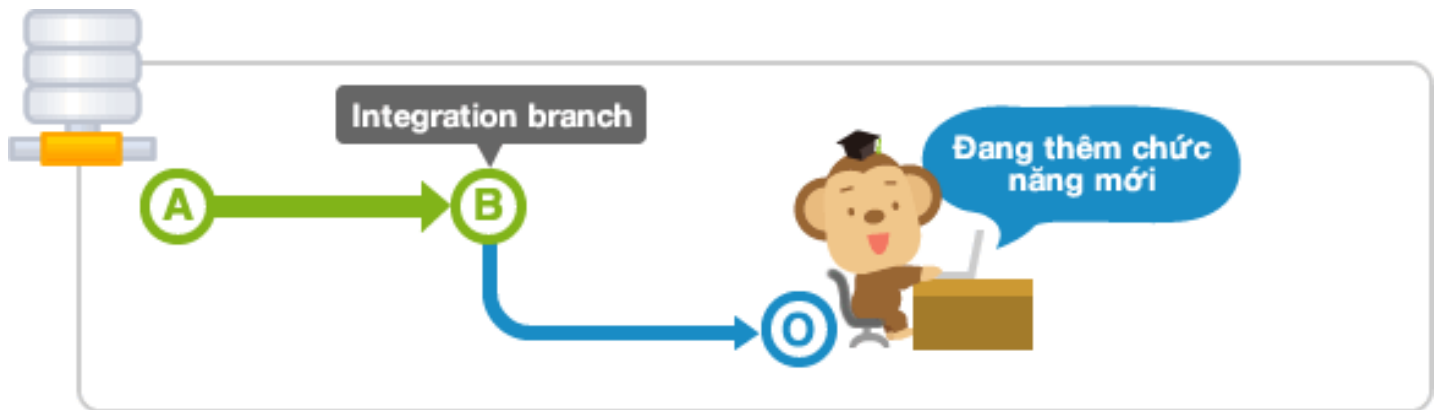
Nội dung video: vận dụng Long-Running Branches và Topic Branches

1. Tham khảo:

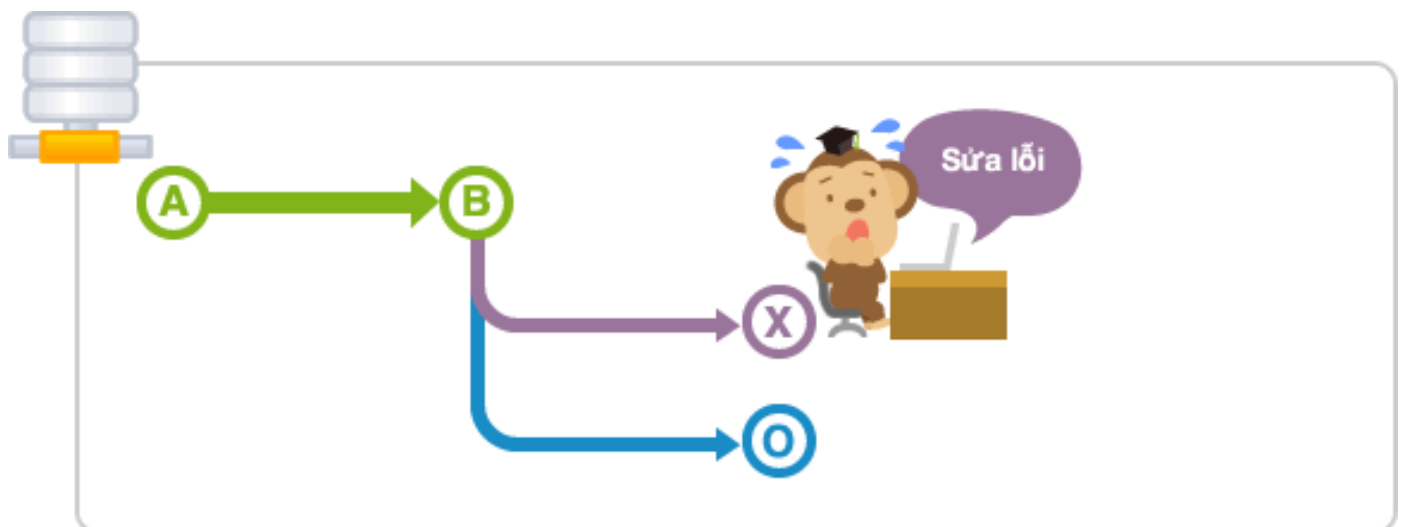
- https://backlogtool.com/git-guide/vn/stepup/stepup1_5.html
- <http://nvie.com/posts/a-successful-git-branching-model/>

2. Nội dung:

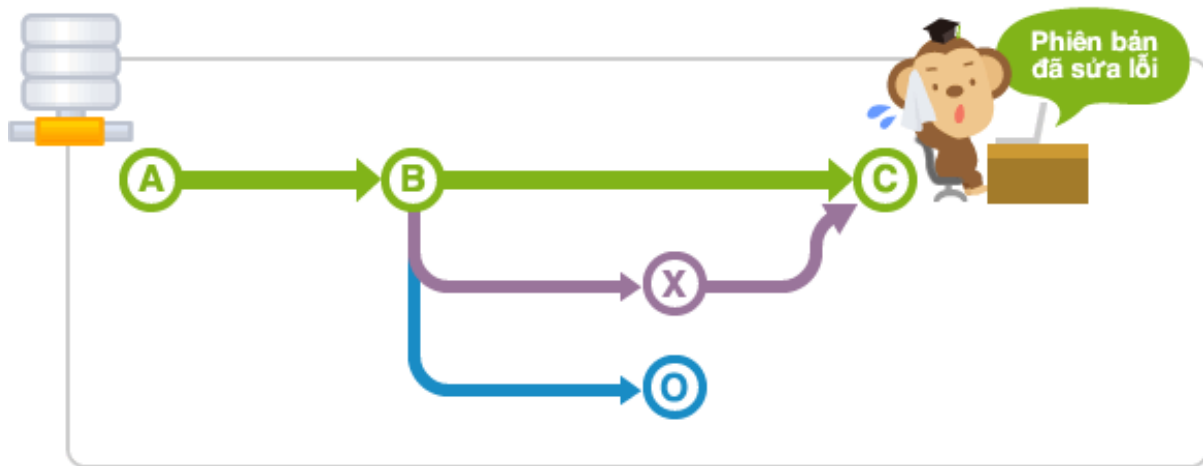
- Hình 1: Làm việc với một chức năng mới “Chức năng Video”



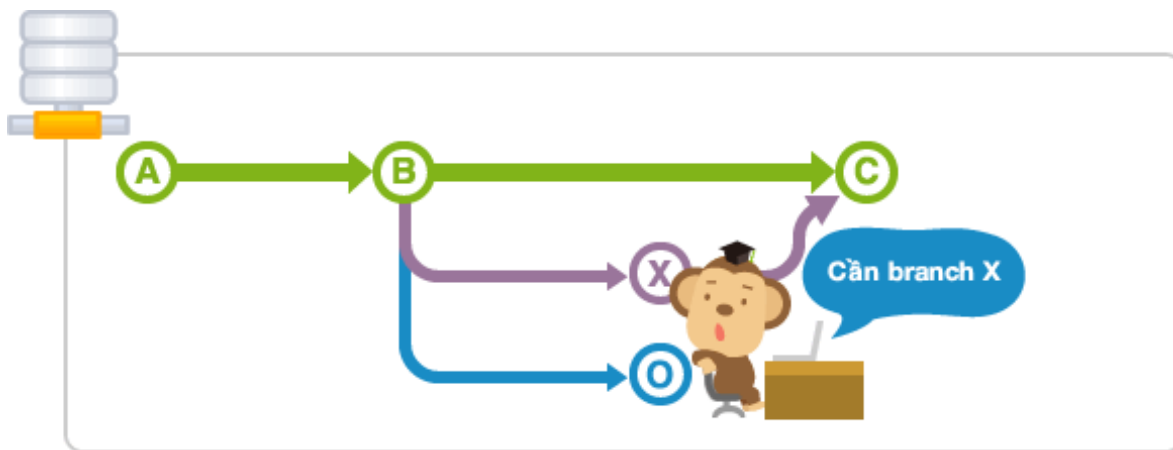
- Hình 2: Tạo branch Hotfix để tiến hành sửa lỗi



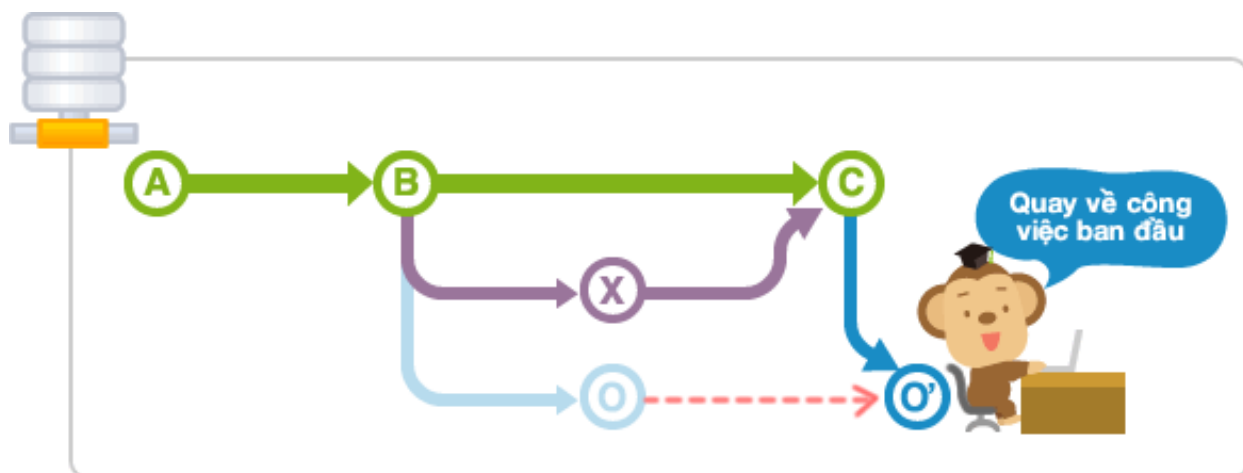
- Hình 3: Tích hợp nhánh Hotfix vào nhánh Master



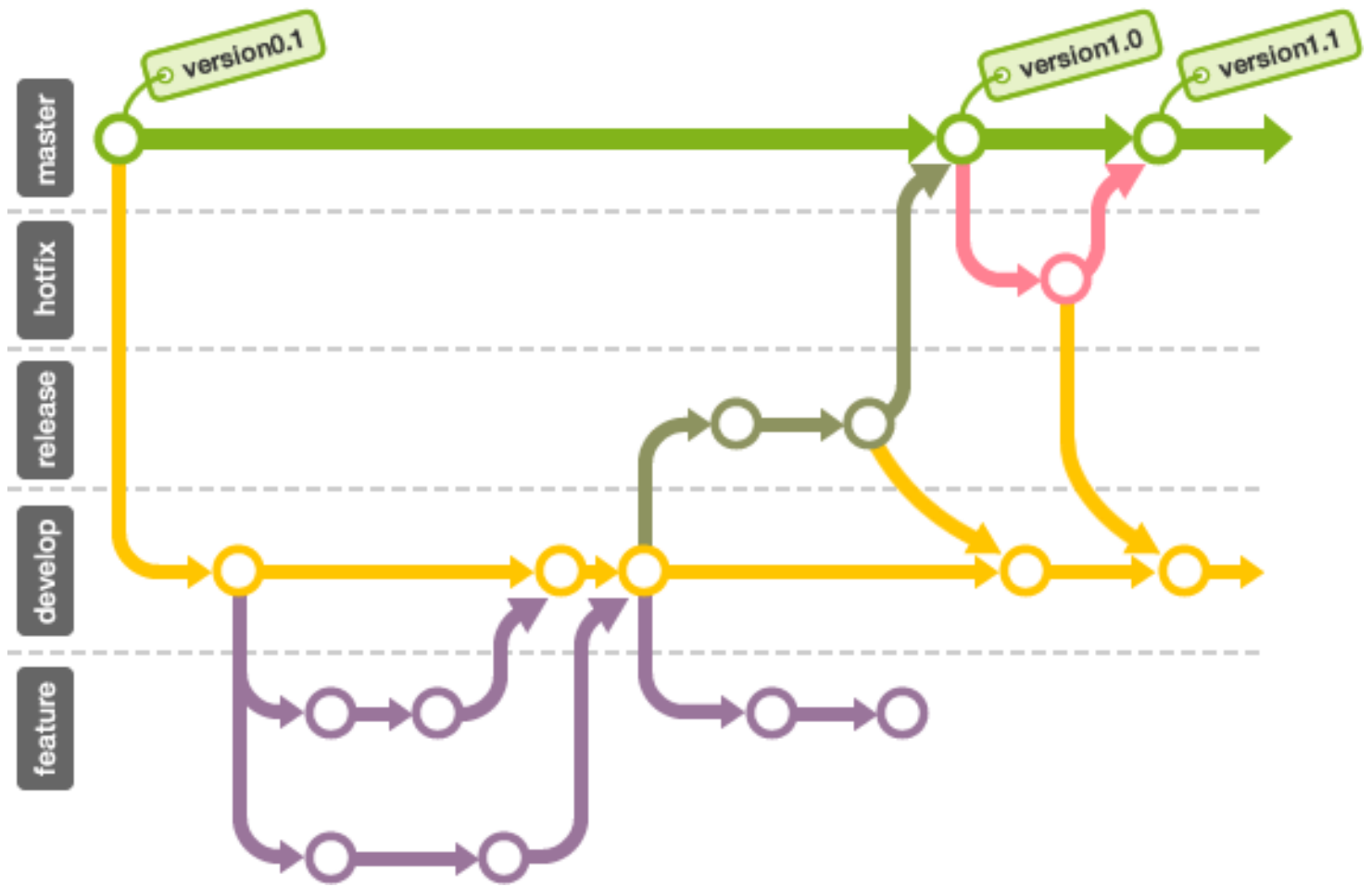
- Hình 4: Nhánh video cũng cần nhánh Hotfix để phát triển tiếp



- Hình 5: Rebase nhánh



3. Mô hình ứng dụng phân nhánh



- Master: quản lý trạng thái có thể release, đánh tag
- Develop: phát triển thông thường hướng đến trước khi release
- Feature: phát triển chức năng mới hay sửa lỗi
- Release: release-xxx
- Hotfix: hotfix-xxx chỉnh sửa gấp đối với sản phẩm đã release

Nội dung video: hiểu rõ hơn về CLONE vs PULL vs FETCH

1. CLONE:

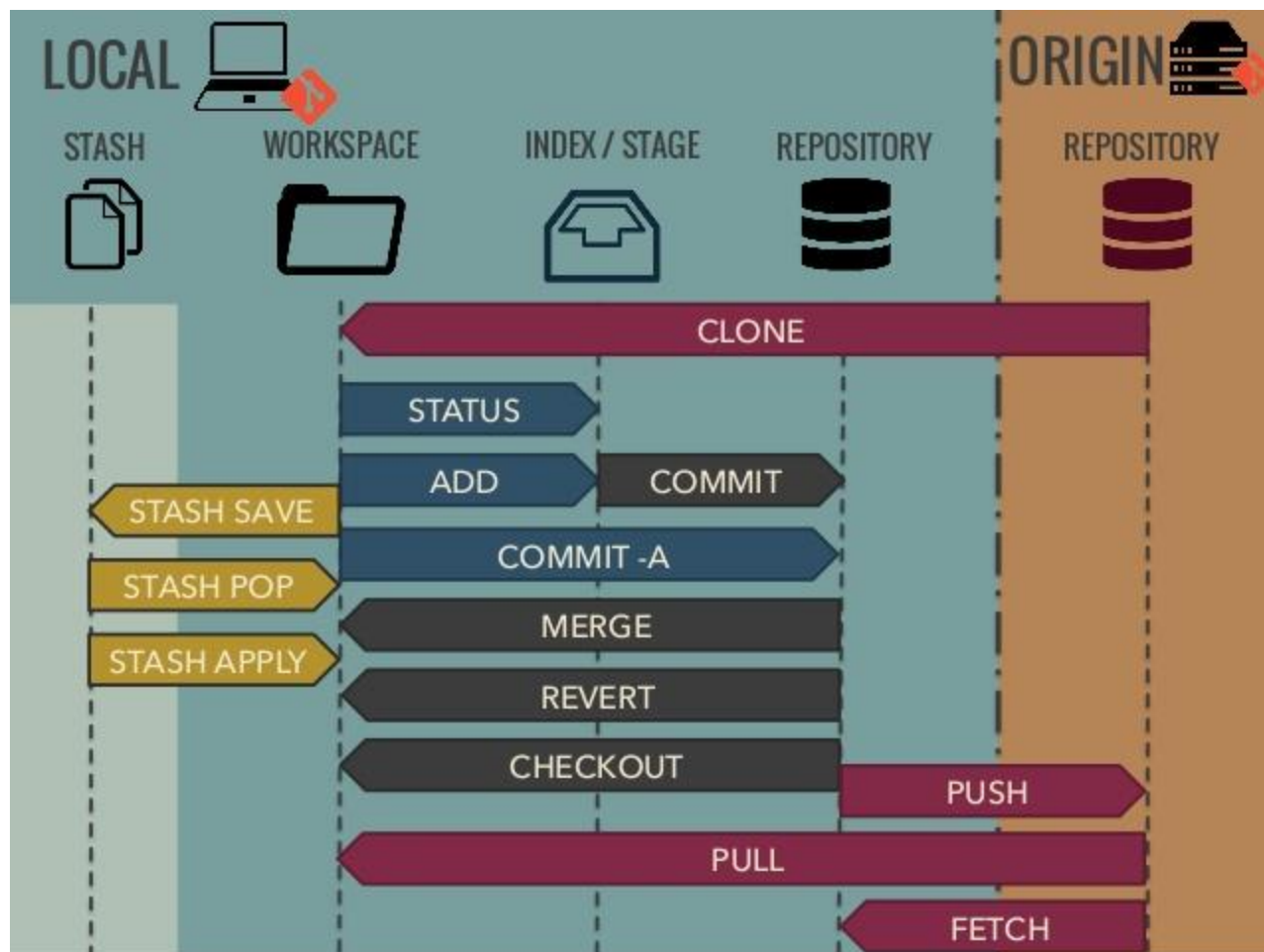
- Sao chép toàn bộ Remote Repo (Sử dụng khi bạn cần tạo mới một Git mới trên máy tính với toàn bộ dữ liệu và thiết lập của một remote repository)

2. PULL

- Lấy toàn bộ dữ liệu từ Remote Repo và gộp vào branch hiện tại.

3. FETCH

- Lấy toàn bộ dữ liệu từ Remote Repo nhưng sẽ cho phép bạn gộp thủ công vào một branch nào đó tại Local Repo



Nội dung video: hiểu rõ hơn REBASING vs MERGING

	<pre> graph LR A((A)) --> B((B)) B --> C((C)) C --> D((D)) D --- master[master] B --> X((X)) X --> Y((Y)) Y --- bugfix[bugfix] </pre>
MERGE	<pre> graph LR A((A)) --> B((B)) B --> C((C)) C --> D((D)) D --> E((E)) E --- master[master] B --> X((X)) X --> Y((Y)) Y --- bugfix[bugfix] </pre> <ul style="list-style-type: none"> • Lịch sử commit không bị thay đổi vẫn còn lại nhưng sẽ trở nên phức tạp hơn. • Không bảo lưu tổng số commit
REBASE	<pre> graph LR A((A)) --> B((B)) B --> C((C)) C --> D((D)) D --- master[master] B --> X((X)) X --> Y((Y)) Y -.-> Xp((X')) Xp --> Yp((Y')) Yp --- bugfix[bugfix] </pre> <ul style="list-style-type: none"> • Lịch sử sẽ trở nên đơn giản • Lịch sử commit sẽ bị thay đổi (X khác X', ...) • Lịch sử commit sẽ là lịch sử commit mới nhất • Bảo lưu tổng số commit
	<ul style="list-style-type: none"> • Trường hợp đưa code mới nhất của branch tích hợp vào branch chủ đề thì sử dụng rebase. • Trường hợp đưa branch chủ đề vào branch tích hợp, thì trước hết hãy rebase rồi merge. • Trường hợp không nắm rõ lịch sử commit trên một branch, nên sử dụng merge để không làm thay đổi lịch sử commit

Nội dung video: hệ thống lại các kiến thức đã học