

Git Basic

Huan Phan

2024

Nội dung chính

1. Mở đầu
2. Quick start
3. Làm việc với local repository
4. So sánh phiên bản, branch, merge
5. Làm việc với remote repository
6. Nâng cao
7. Git workflow

1. Mở đầu

Về khoá học

Version Control System

Về khoá học

- Dành cho những bạn mới làm quen với git để quản lý mã nguồn
- Các thao tác quan trọng nhất khi làm việc với git
- Học qua thực hành
 - Thực hành trên máy tính cá nhân các câu lệnh ví dụ
 - Quiz
 - Q&A

Version Control System

Quiz

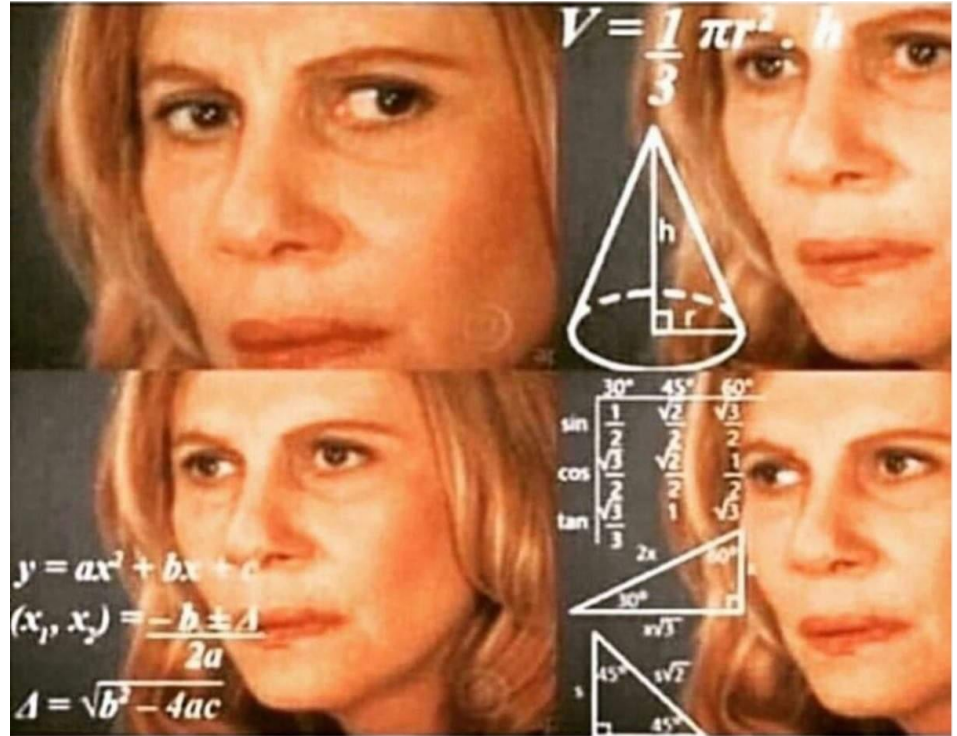


Các project ở trường thường yêu cầu làm việc nhóm, vậy các bạn làm cách nào để quản lý mã nguồn của những project đó?

Tại sao cần Version Control System?

Khi không có VCS để quản lý version

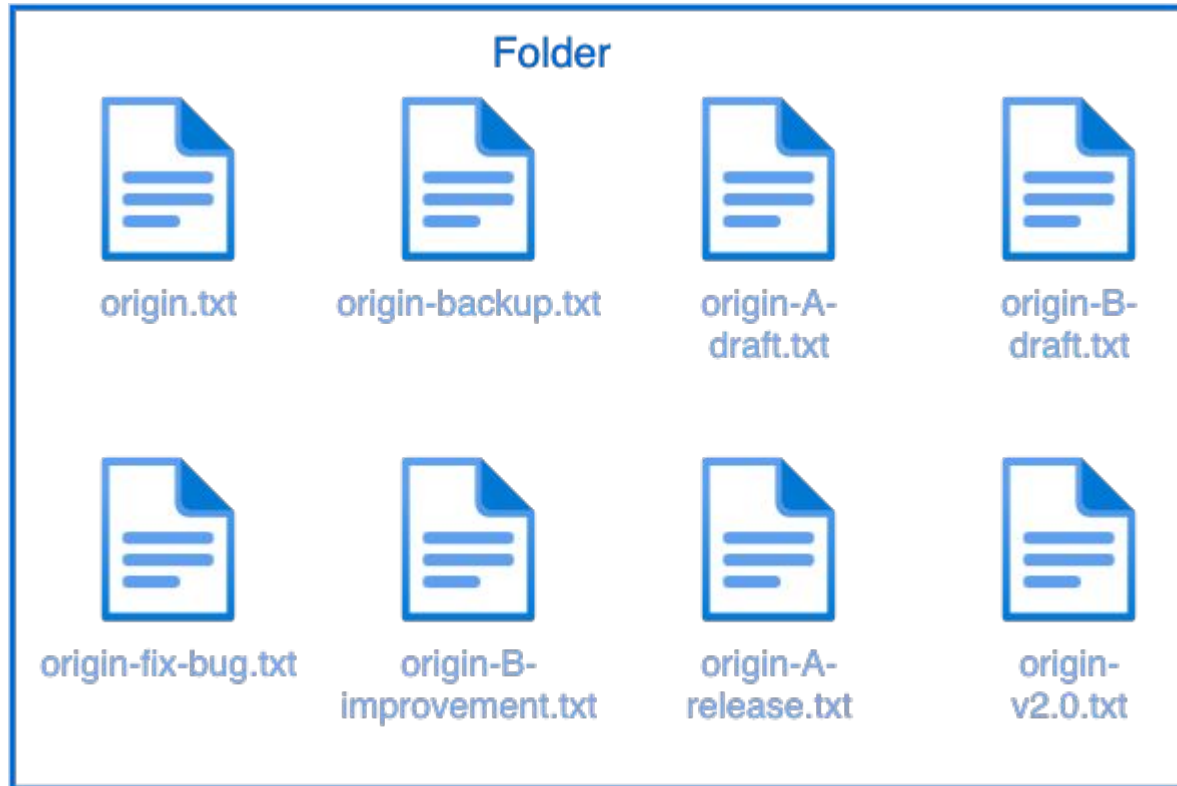
- Với mỗi thay đổi đến file, tạo 1 bản sao lưu tương ứng
- Khó khăn trong việc chia sẻ file/code cho các thành viên trong nhóm
- Khó quản lý phiên bản, xem lịch sử thay đổi, phục hồi phiên bản



Tại sao cần Version Control System?

Khi không có VCS

- Quản lý phiên bản cho các thay đổi một cách thủ công
- Tạo các bản copy để lưu giữ/sao lưu các thay đổi khác nhau cho cùng 1 file
- Không dễ dàng để tìm ra sự khác nhau giữa các phiên bản
- Khó theo dõi lịch sử các thay đổi, người tạo thay đổi, v.v.
- Khó gộp hay chọn lọc các thay đổi khác nhau



Quiz



A thêm 1 vài đoạn code và đặt tên cho file mới là *A-feature.py*. Không may đoạn code được thêm làm chương trình bị crash. Làm thế nào để B có thể tìm và sửa/xoá đoạn code đã được thêm?

Version Control System là gì?

- Hệ thống lưu trữ tất cả phiên bản của file cùng với lịch sử thay đổi theo thời gian, giúp dễ dàng khôi phục lại phiên bản mong muốn khi cần
- Dễ dàng chia sẻ các phiên bản cho các thành viên khác
- Cùng lúc làm việc với nhiều phiên bản của file
- Hỗ trợ làm việc nhóm, tăng hiệu suất làm việc

Quiz



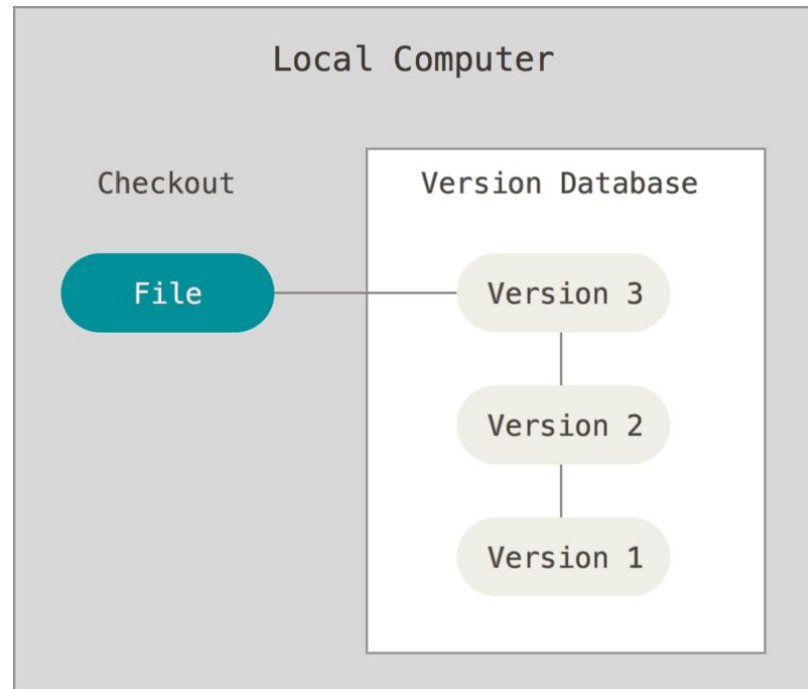
Những ai cần dùng VCS?

Các dạng VCS

- Tập trung
 - Các phiên bản của file được quản lý tập trung ở server, mọi thao tác kể cả thay đổi file cần kết nối đến server
 - Ví dụ: Subversion
- Phi tập trung
 - Các máy local lưu trữ toàn bộ các version của file và lịch sử
 - Các thao tác đến file có thể thực hiện độc lập ở local, khi cần có thể đồng bộ lên server
 - Ví dụ: Git
- Tham khảo thêm:
 - [Distributed version control](#)
 - [Comparison of version control system](#)

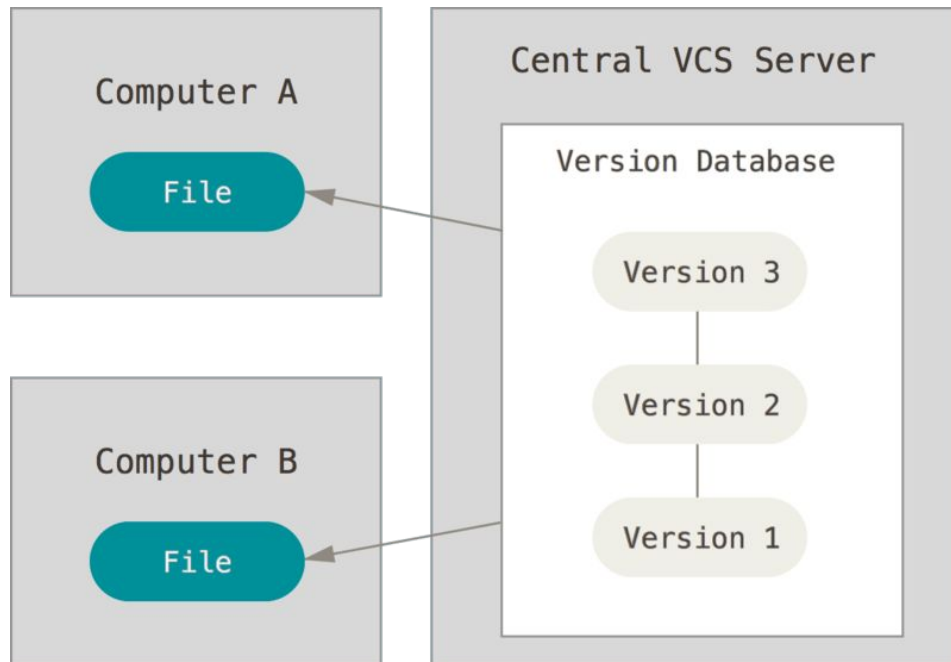
Các dạng VCS - Local

- Lưu trữ cục bộ trên máy cá nhân, khó chia sẻ và hợp tác với thành viên khác
- Dễ mất mát dữ liệu nếu xảy ra hỏng hóc vật lý



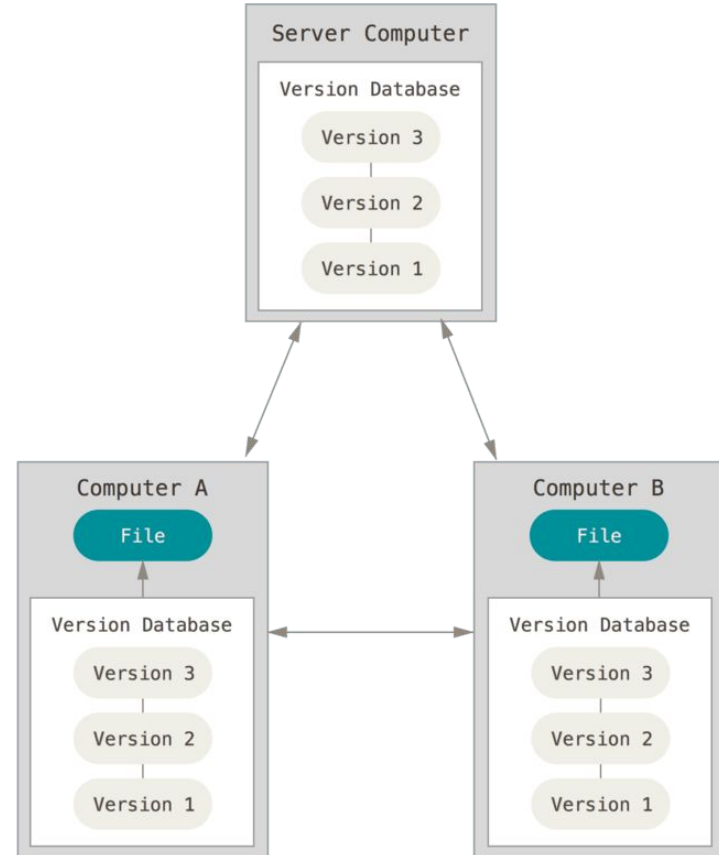
Các dạng VCS - Tập trung

- Client kết nối đến server tập trung để thao tác với các phiên bản của file
- Cho phép chia sẻ file
- Yêu cầu kết nối internet ổn định
- Rủi ro mất mát dữ liệu nếu xảy ra hỏng hóc vật lý
- Example:
 - SVN



Các dạng VCS - Phân tán

- Dữ liệu được lưu trữ phân tán ở server và client
- Các thao tác với phiên bản có thể thực hiện offline, chỉ cần kết nối internet để đồng bộ các thay đổi
- Làm việc nhóm dễ dàng
- Example:
 - Git



Các thuật ngữ

- **Repository (repo):** nơi chứa tất cả dữ liệu cần để quản lý các phiên bản và lịch sử các sửa đổi. Repo có thể được chia sẻ giữa các thành viên.
- **Commit:** tập các sửa đổi đã được thêm vào repo, đi kèm mô tả, tác giả, thời gian v.v.
- **Version (phiên bản):** một file có nhiều phiên bản tương ứng với mỗi sửa đổi được commit vào repo. Các phiên bản thường được đánh số tăng dần (revision) như 1, 2, 3, etc.
- **Branch (nhánh):** khi phân nhánh từ nhánh gốc, mỗi nhánh lưu giữ file/code độc lập, các thành viên có thể cùng lúc làm việc trên nhiều nhánh khác nhau.
- **Merge:** ghép nhánh
- **Pull:** Sao chép các thay đổi từ remote repo về local
- **Push:** Đồng bộ các thay đổi từ local lên remote repo

Git

Lịch sử ra đời

- Phát triển bởi Linus Torvalds (quen không?) vào năm 2005 nhằm phục vụ cho việc phát triển Linux Kernel thay thế cho BitKeeper
- Mong muốn một VCS phân tán, đơn giản, các thao tác phải nhanh và hiệu quả



Git là gì?

- Version Control System
- Phân tán, phi tập trung
- Đa số các operation có thể thực hiện offline, cục bộ
- Open source



Tại sao dùng Git?

- VCS phổ biến nhất thế giới ([93.9%](#))
- Phân tán, phi tập trung
- Đa số các operation có thể thực hiện offline, local, fast
- Free, open source
- Cộng đồng người sử dụng lớn
- Quản lý project lớn hiệu quả

Git Server

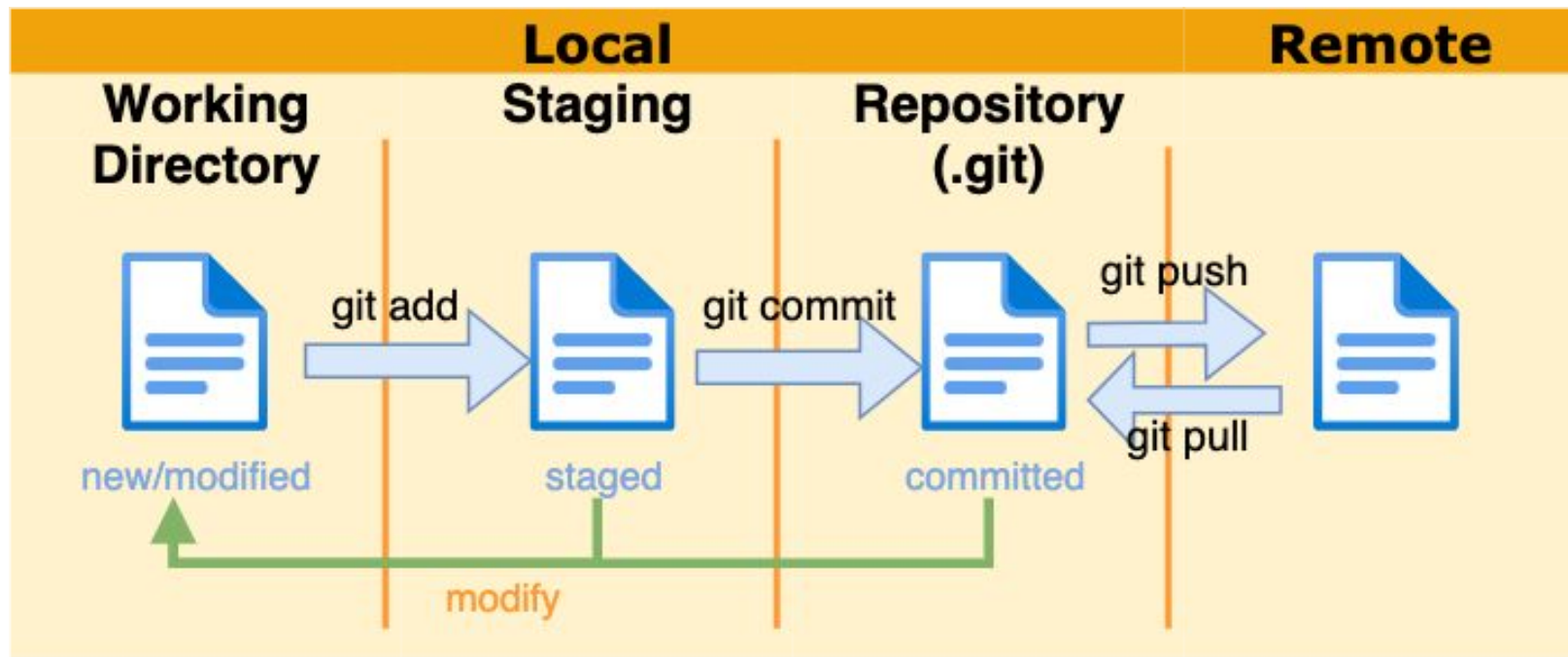
- Repository Hosting
- Web interface
- Continuous Delivery/ Continuous Integration
- Access control
- etc.



ATLASSIAN



Các trạng thái của file



Các trạng thái của file

- Đối với git, một file sẽ có thể có các trạng thái:
 - New/untracked
 - Modified
 - Staged
 - Committed

Các phân vùng

- Local: trên máy cá nhân
 - Working Directory: thư mục dự án hiện tại
 - Staging: chứa các file đã được staged để chuẩn bị cho commit
 - Repository: các file đã được commit
- Remote Repository
 - Lưu giữ các thay đổi và các phiên bản đã được đồng bộ từ 1 hoặc nhiều Local Repository
 - Repository tập trung

2. Quick start

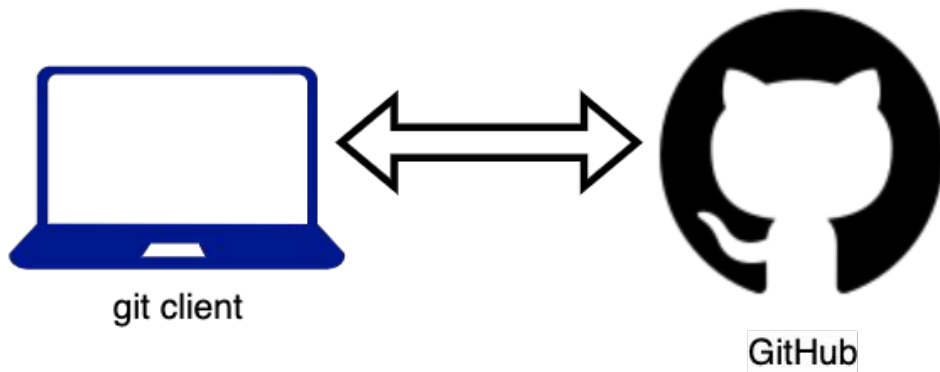
Cài đặt git và các công cụ

Cài đặt git client

- Windows
 - <https://git-for-windows.github.io/>
- MacOS
 - Simplest: git version
 - <https://git-scm.com/download/mac>
- Linux
 - `sudo apt-get install git`
 - <https://git-scm.com/download/linux>

Git Server

- Ta sẽ dùng GitHub làm git server
- <https://github.com/>
- Cài đặt:
 - Đăng ký account (free):
<https://docs.github.com/en/get-started/signing-up-for-github/signing-up-for-a-new-github-account>



Cấu hình cho git client

- Sau khi đã đăng ký thành công account trên GitHub, ta có thể thực hiện cấu hình user mặc định cho git client

- Username

```
> git config --global user.name "Van A"
```

```
> git config --global user.email "vana@gmail.com"
```

- Xem cấu hình hiện tại

```
> git config --global --list
```

Công cụ

- Diff & merge tool: P4Merge
- Cài đặt: <https://www.perforce.com/downloads/visual-merge-tool>
 - Mac
 - Windows

3. Làm việc với local repository

Các câu lệnh cơ bản

Thao tác cơ bản

Khởi tạo 1 project

- Mới hoàn toàn
- Từ source code có sẵn trên máy
- Từ GitHub

Khởi tạo project mới hoàn toàn

- Tạo empty folder và khởi tạo Git repository

```
> cd ~  
> mkdir first-repo  
> cd first-repo  
> git init  
> git status
```

```
→ first-repo git:(main) git status  
On branch main  
  
No commits yet  
  
nothing to commit (create/copy files and use "git add" to track)
```

- `.git` folder sẽ được khởi tạo, chứa các thông tin cần thiết giúp git hoạt động

```
→ first-repo git:(main) ls -la .git  
.  
..
```

	HEAD	description	info	refs
	config	hooks	objects	

Khởi tạo project từ code đã có

- Tạo empty folder và khởi tạo Git repository

```
> cd ~
> mkdir existing-project
> cd existing-project
> echo '# Hello World' > >
README.md
> echo '# VDT 2023' > intro.md
> git init
> git status
```

```
→ existing-project git:(main) ✖ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    README.md
    intro.md
```

- Mặc định các file sẽ ở trạng thái `untracked` (không được quản lý bởi git)
- `Untracked files`: danh sách các file `untracked`
- Thêm tất cả file vào `Staging (tracked)`:
`> git add -a`
- Thêm 1 file:

```
> git add README.md
```

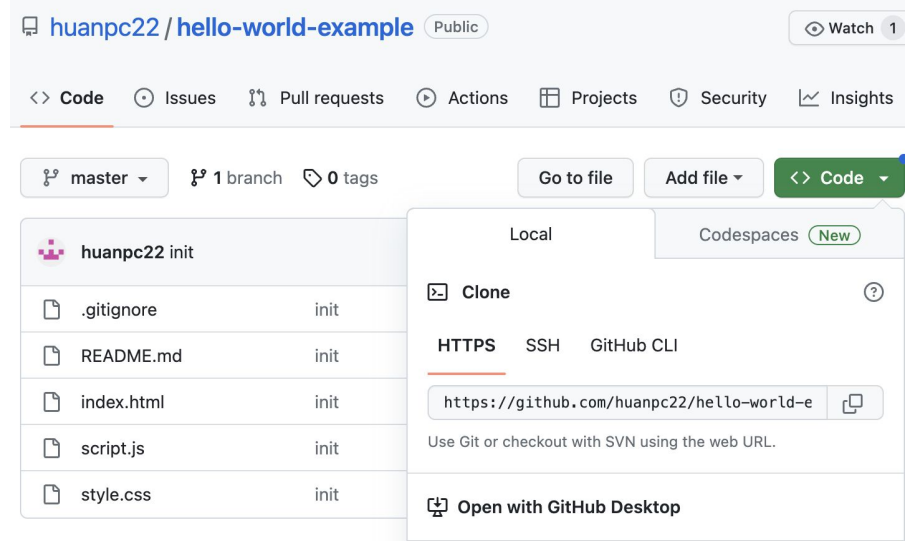
```
→ existing-project git:(main) ✖ git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   README.md
```

Khởi tạo project từ GitHub - git clone

- Project example:
<https://github.com/huanpc22/hello-world-example>
- Log in GitHub account
- Fork project
- Clone remote repository to local machine
 - use https (login by username/access token):
`> git clone https://github.com/huanpc/hello-world-example.git`
 - or use SSH (cần tạo SSH key):
`> git clone git@github.com:huanpc/hello-world-example.git`

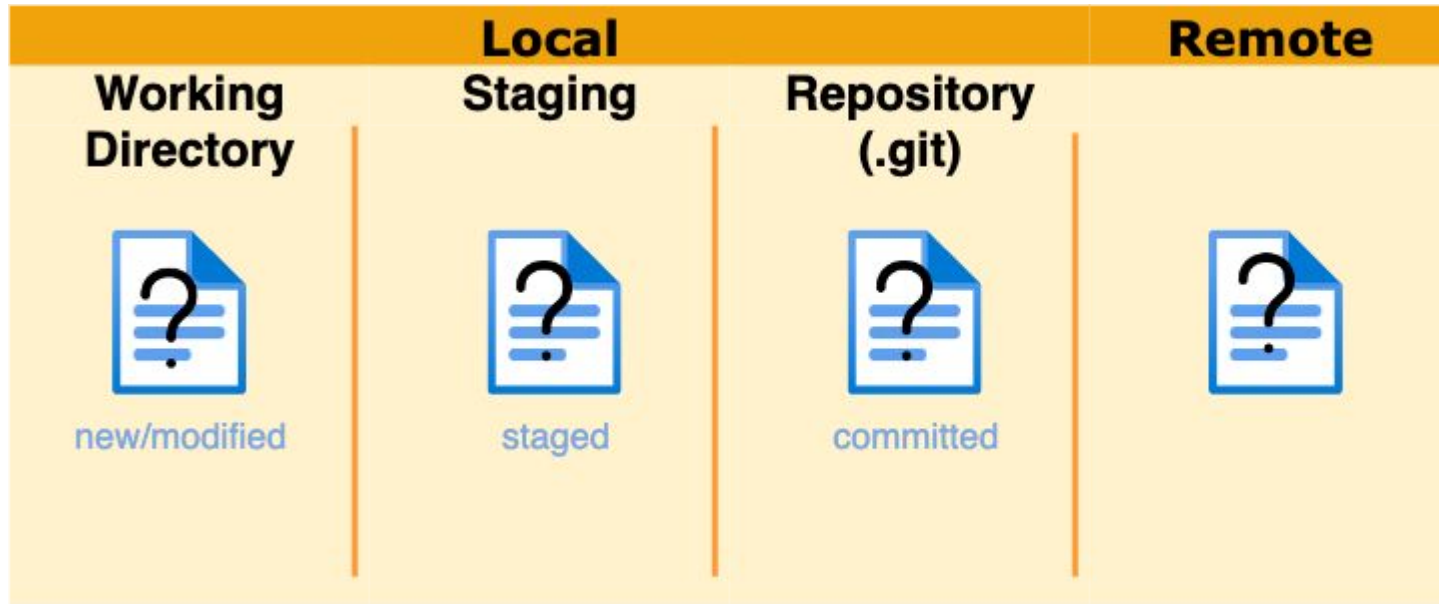


Khởi tạo project

- Nhánh (branch) mặc định: master
- Remote repository mặc định: origin

Câu lệnh git status

- Xem trạng thái của repository, trạng thái của các file



Git tracked file

- Untracked: file không được theo dõi và quản lý phiên bản bởi git
- File mới tạo sẽ ở trạng thái untracked
- Để track file

```
> git add <file_path>
```



Thêm thay đổi mới vào Staging

- File đã được track

```
cd hello-world-example
```

```
echo 'Hello World!' >> README.md
```

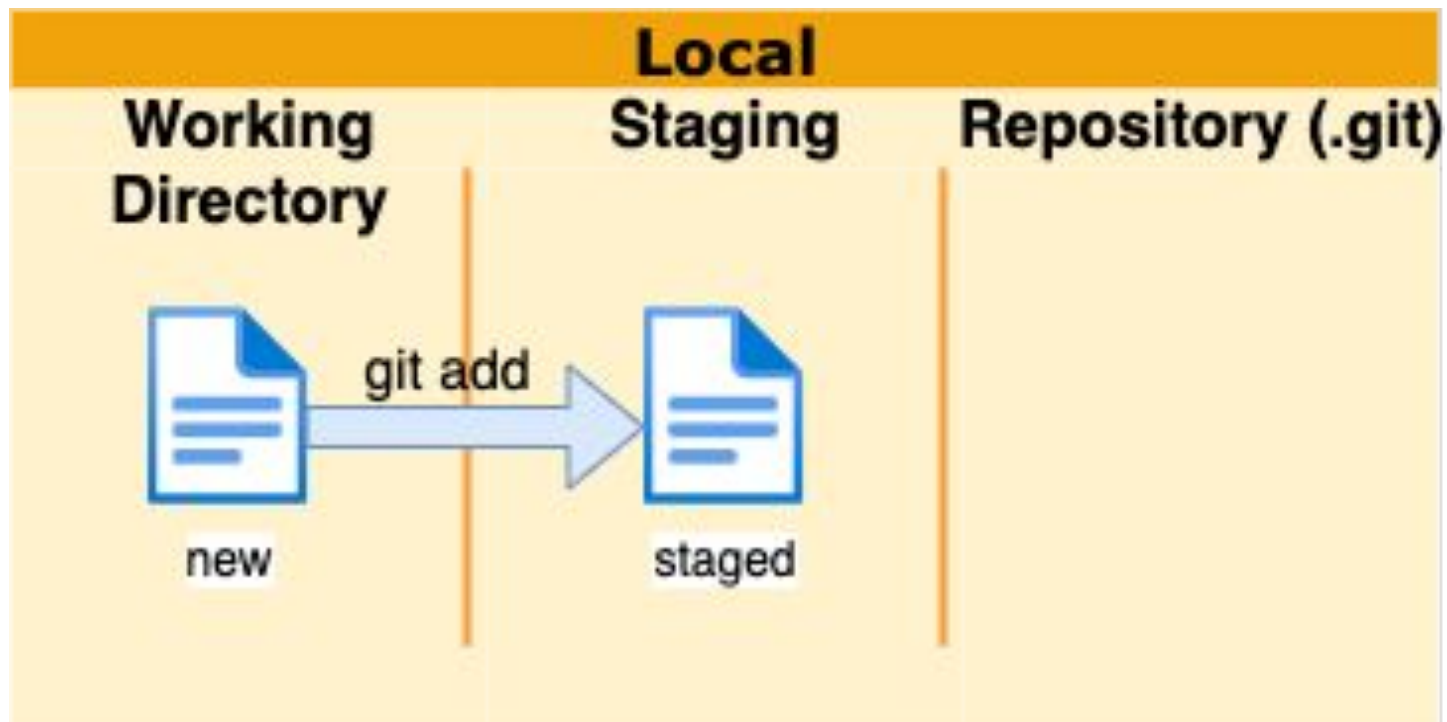
```
→ hello-world-example git:(master) ✕ git status
On branch master
Your branch is up to date with 'origin/master'.

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:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
```

```
git add README.md
```

Thêm thay đổi



Cập nhật thay đổi vào Local Repository

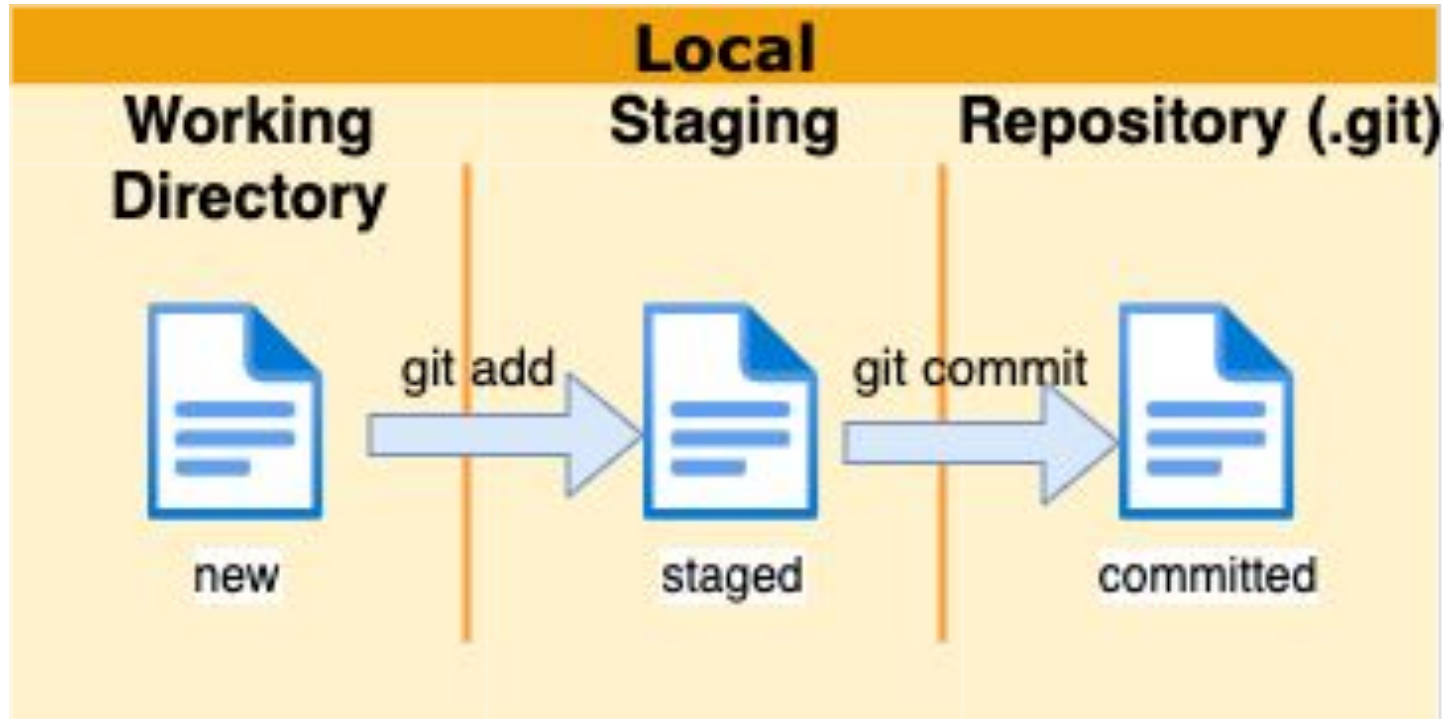
- Cập nhật kèm message:

```
> git commit -m "This is a message for new changes"
```

- Cùng lúc cập nhật thay đổi vào Staging và Repository (git add & git commit):

```
> git commit -am "This is a message for new changes"
```

Cập nhật thay đổi vào Local Repository



Hủy commit

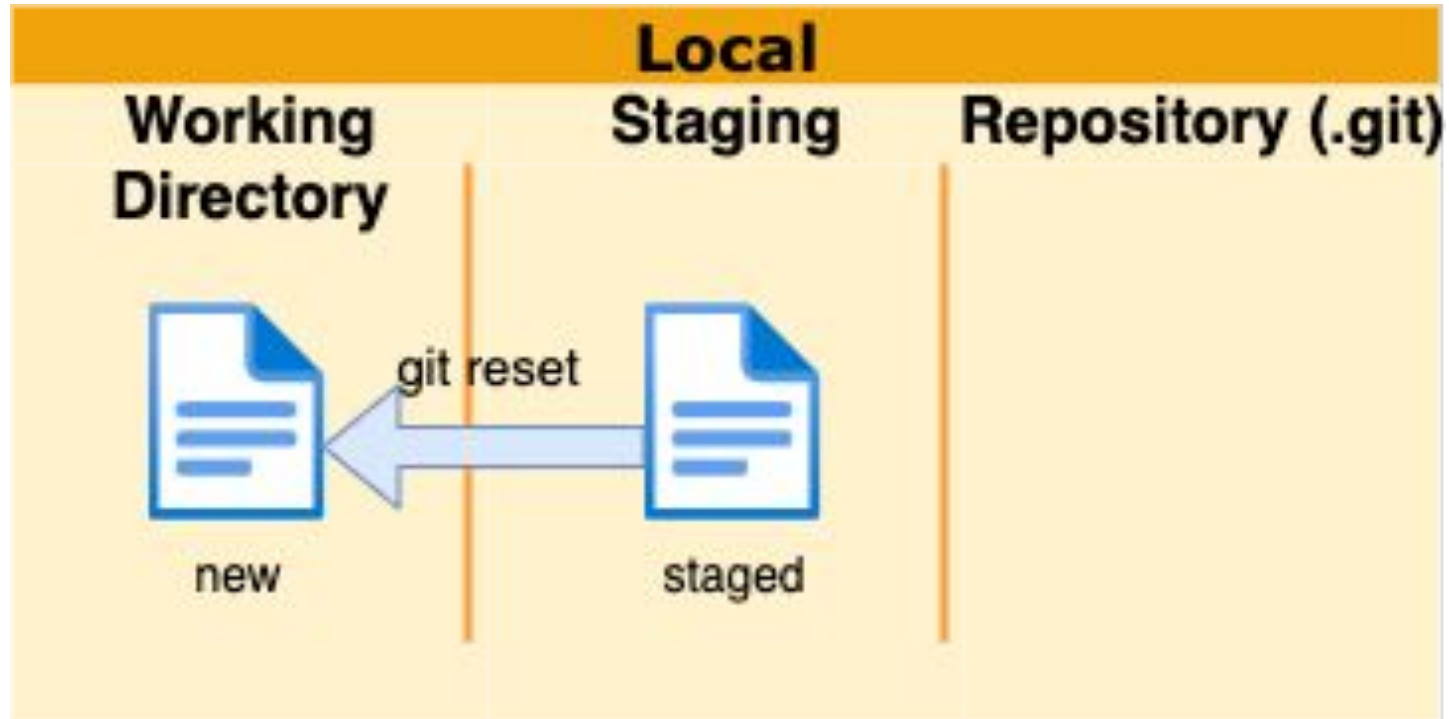
- Hủy commit, đưa thay đổi từ Staging về lại Working Directory (unstage):

```
> git reset HEAD README.txt
```

```
> cat README.txt
```

```
→ hello-world-example git:(master) x cat README.md  
Hello World!
```

Hủy commit



Huỷ thay đổi

- Huỷ bỏ thay đổi đã tạo ở Working Directory và đưa về trạng thái commit gần nhất:

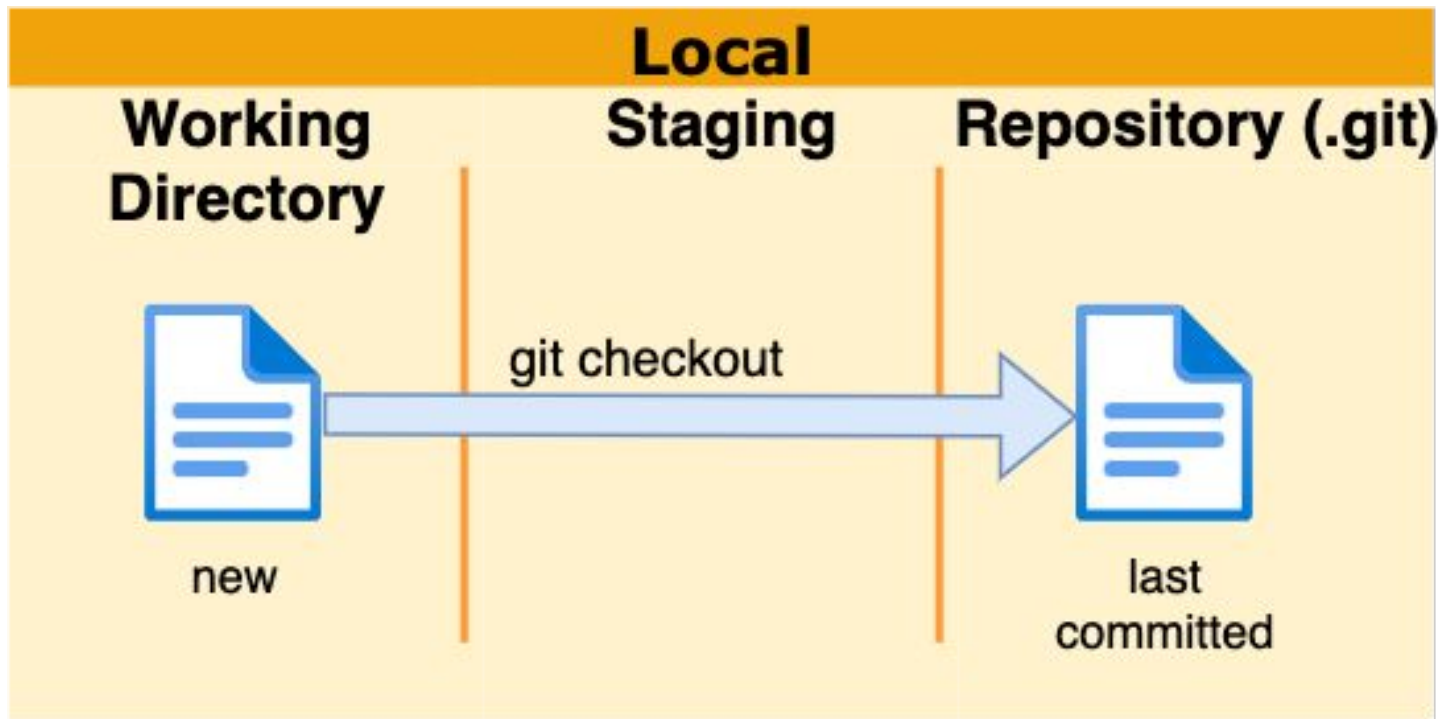
```
> git checkout -- README.md
```

```
> cat README.txt
```

```
# thay đổi mới tạo đã bị huỷ bỏ
```

```
→ hello-world-example git:(master) cat README.md  
→ hello-world-example git:(master) █
```

Hủy bỏ thay đổi



Xem lịch sử commit

- Xem lịch sử tất cả file

```
> git log
```

```
> git log --oneline --graph --decorate
```

- Xem lịch sử 1 file

```
> git log -- file_name
```

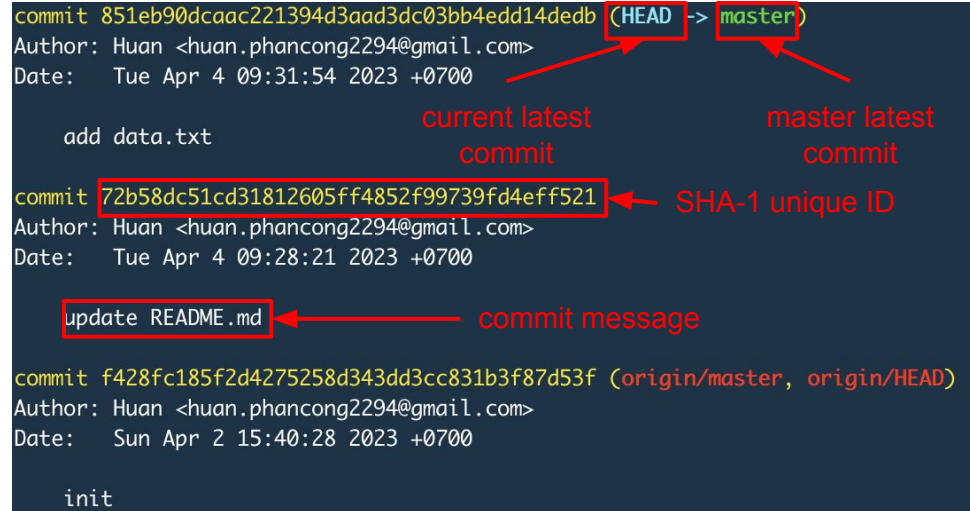
- Xem chi tiết các thay đổi của file

```
> git show
```

Xem lịch sử commit

- Commit all changes

```
> echo 'Hello World!' >> README.md
> git commit -am "update README.md"
> echo 'This is mock data' >
data.txt
> git add data.txt
> git commit -m "add data.txt"
```



The screenshot shows a terminal window with Git commit history. Red boxes and arrows highlight key elements: the HEAD to master transition, the current latest commit, the master latest commit, the SHA-1 unique ID, and the commit message.

```
commit 851eb90dcaac221394d3aad3dc03bb4edd14dedb (HEAD -> master)
Author: Huan <huan.phancong2294@gmail.com>
Date: Tue Apr 4 09:31:54 2023 +0700
    add data.txt

commit 72b58dc51cd31812605ff4852f99739fd4eff521
Author: Huan <huan.phancong2294@gmail.com>
Date: Tue Apr 4 09:28:21 2023 +0700
    update README.md

commit f428fc185f2d4275258d343dd3cc831b3f87d53f (origin/master, origin/HEAD)
Author: Huan <huan.phancong2294@gmail.com>
Date: Sun Apr 2 15:40:28 2023 +0700
    init
```


Xem lịch sử commit

- Rút gọn commit ID

```
git log --abbrev-commit
```

```
commit 851eb90 (HEAD -> master)  
Author: Huan <huan.phancong2294@gmail.com>  
Date: Tue Apr 4 09:31:54 2023 +0700  
  
    add data.txt
```

- Rút gọn thông tin và hiển thị dạng graph

```
git log --oneline --graph --decorate
```

```
* 851eb90 (HEAD -> master) add data.txt  
* 72b58dc update README.md  
* f428fc1 (origin/master, origin/HEAD) init
```

-> hữu dụng khi repo có nhiều branch!

Loại trừ file từ git repository

- Một số file có thể không cần thêm vào git:
 - log file
 - tmp: .DS_Store, <filename>~, .swp, .tmp
 - compiled file:
 - Java: *.class, *.jar
 - Python: *.pyc
- Dùng .gitignore file







```
> touch .gitignore
```

```
> echo "*.pyc" >> .gitignore
```

```
> echo "*.log" >> .gitignore
```

Loại trừ file từ git repository

- Template ví dụ cho từng loại project:
 - <https://github.com/github/gitignore>

	JBoss.gitignore	correct capitalization problems and add Oc...
	JENKINS_HOME.git...	Improved JENKINS_HOME example (#3332)
	Java.gitignore	Add replay_pid* to Java.gitignore
	Jekyll.gitignore	Ignore .bundle and vendor per official Jekyll...
	Joomla.gitignore	Updating Joomla.gitignore to Joomla! 3.9.8...
	Julia.gitignore	Add more standard ignored files for Julia (...)

- Java gitignore

```
1 # Compiled class file
2 *.class
3
4 # Log file
5 *.log
6
7 # BlueJ files
8 *.ctxt
9
10 # Mobile Tools for Java (J2ME)
11 *.mtj.tmp/
12
13 # Package Files #
14 *.jar
15 *.war
16 *.nar
17 *.ear
18 *.zip
19 *.tar.gz
20 *.rar
21
```

Loại trừ file từ git repository

- `.gitignore` rule:
 - glob pattern: [https://en.wikipedia.org/wiki/Glob_\(programming\)](https://en.wikipedia.org/wiki/Glob_(programming))
 - more detail: https://git-scm.com/docs/gitignore#_pattern_format

git references (refs)

- Thay vì dùng commit ID ta có thể dùng refs để tham chiếu đến 1 commit
- 4 kiểu refs
 - head: tham chiếu đến commit mới nhất của nhánh
 - HEAD: tham chiếu đến commit gần nhất, là head của nhánh nếu đang ở nhánh hoặc commit nếu ở trạng thái detached HEAD
 - tag: tham chiếu đến 1 commit bất kỳ, ví dụ: v1.0 -> <commit_id>
 - remote: tham chiếu đến commit gần nhất được đồng bộ từ remote repository
- Ví dụ

```
> git checkout v1.0
```

Recap

- Khởi tạo git repo

```
git init
```

- Track/Stage file

```
git add <file_path>
```

- Staged file (commit)

```
git commit -am "message"
```

- Xem trạng thái repo

```
git status
```

- Huỷ bỏ thay đổi đã staged

```
git reset HEAD <file_path>
```

- Huỷ bỏ thay đổi

```
git checkout -- <file_path>
```

- Xem lịch sử commit

```
git log
```

- Ignore file

```
"echo *.log" >> .gitignore
```

Q&A

Quiz



Câu lệnh nào được dùng để xem lịch sử các commit?

- A. git status
- B. git config
- C. git commit
- D. git log

Quiz



Câu lệnh nào được dùng để liệt kê các file có thay đổi ở Working Directory?

- A. git clone
- B. git status
- C. git config
- D. git log

Quiz



Kiểu file nào sẽ không được thêm thay đổi vào commit từ câu lệnh `git commit -a` ?

- A. File mới
- B. File đã được track
- C. File đã được stage

Quiz



.gitignore là file gì?

- A. File được bỏ qua bởi người dùng
- B. File bao gồm danh sách các thay đổi chưa được commit
- C. File bao gồm danh sách các file hoặc pattern cần git bỏ qua

Quiz



HEAD tham chiếu đến?

- A. 1 thay đổi trong commit
- B. Commit hiện tại sau khi dùng git checkout
- C. Commit đầu tiên của repo

4. So sánh, nhánh, merge

Hiểu về vai trò của nhánh,
merge nhánh và xử lý conflict

So sánh

Đối chiếu sự thay đổi giữa các phiên bản của file

So sánh

- So sánh Working Directory vs Staging

```
git diff --
```

- So sánh Working Directory vs Local Repo

```
git diff -- HEAD
```

- So sánh Staging vs Local Repo

```
git diff --staged HEAD
```

- So sánh 2 commit

```
git diff <ref_id_1> <ref_id_2>
```

- So sánh Local Repo vs Remote Repo

```
git diff <local_branch>  
<remote>/<remote_branch>
```

So sánh

- So sánh Working Directory vs Staging

```
> echo '## Introduction' >>
```

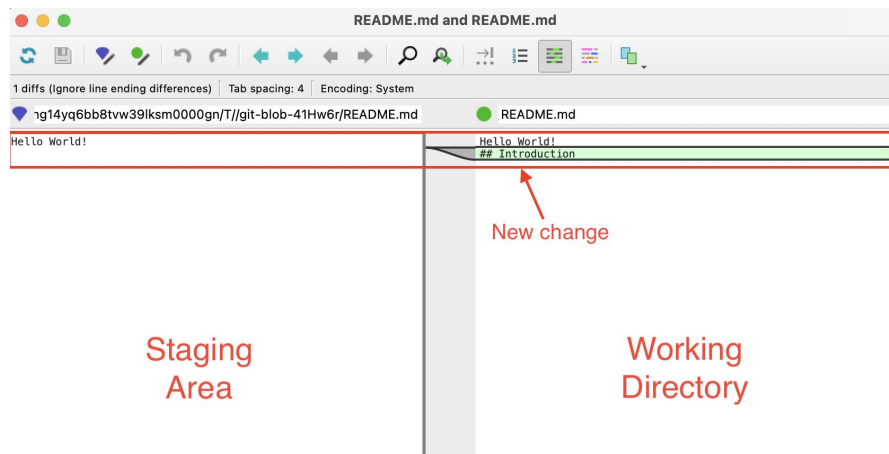
```
README.md
```

```
> git diff --
```

```
diff --git a/README.md b/README.md
index 980a0d5..05fd41d 100644
--- a/README.md
+++ b/README.md
@@ -1,2 @@
 Hello World!
+## Introduction
```

New change

- P4Merge tool



So sánh 2 commit

```
git commit -am "add Introduction"
```

```
git log --oneline --graph --decorate
```

```
* 68741b0 (HEAD -> master) add Introduction
* 851eb90 add data.txt
* 72b58dc update README.md
* f428fc1 (origin/master, origin/HEAD) init
```

```
git diff 851eb90 68741b0
```

```
diff --git a/README.md b/README.md
index 980a0d5..05fd41d 100644
--- a/README.md
+++ b/README.md
@@ -1,2 @@
  Hello World!
+## Introduction
```

Nhánh (branch)

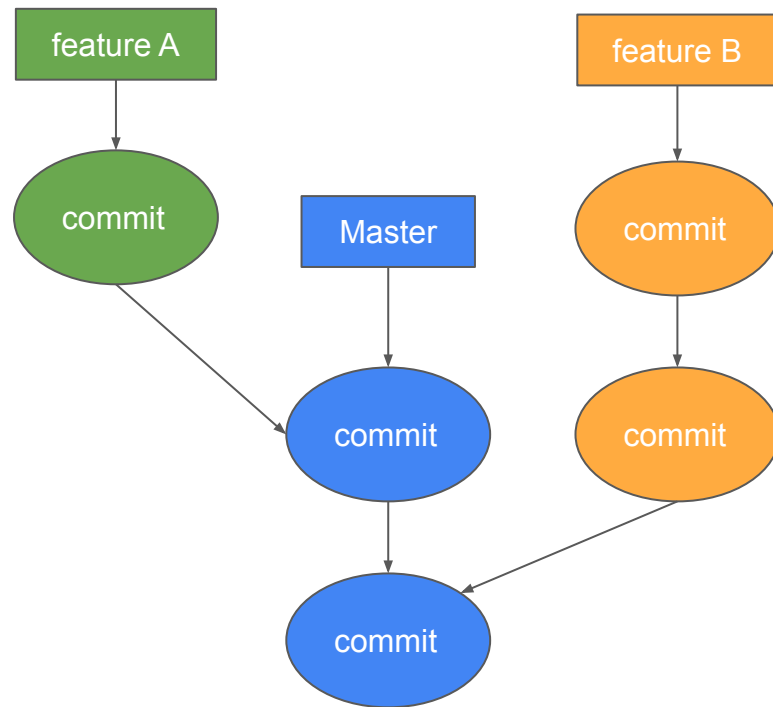
Làm việc với nhánh

Use case

- Nhánh cho phép phát triển các tính năng độc lập cùng lúc mà không làm ảnh hưởng đến nhánh khác
- Các mô hình nhánh (branching model) phổ biến:
 - Master: production code, stable
 - Test/UAT: kiểm thử, testing
 - Feature: rẽ nhánh từ Master, phát triển độc lập rồi merge lại vào Master khi release production

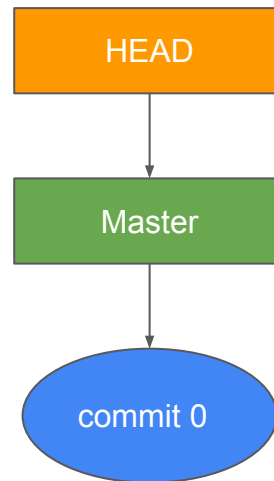
Use case

- Trong thực tế, branching model có thể rất phức tạp với hàng chục nhánh độc lập
- Cấu trúc dạng cây (tree)
 - Các nhánh có thể có cùng nút tổ tiên (common ancestor)
 - head của nhánh trở đến commit mới nhất



Ví dụ về nhánh

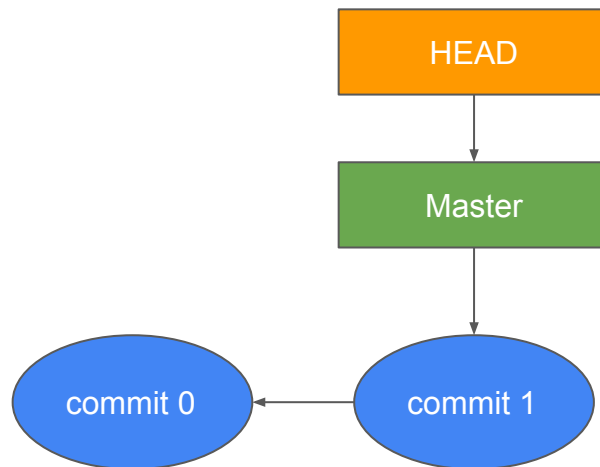
- Khởi đầu, nhánh mặc định là Master
- HEAD/Master cùng trỏ đến commit mới nhất
- Chúng ta sẽ thực hiện thao tác cơ bản nhất về nhánh là rẽ nhánh



Ví dụ về nhánh

- Thêm commit với file mới

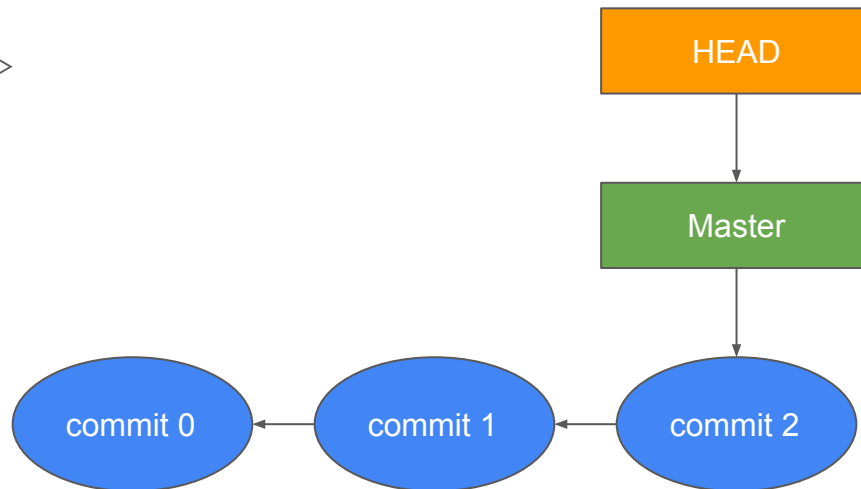
```
> echo 'example 1' >  
example.txt  
> git add .  
> git commit -m "commit 1"
```



Ví dụ về nhánh

- Thêm commit với file mới

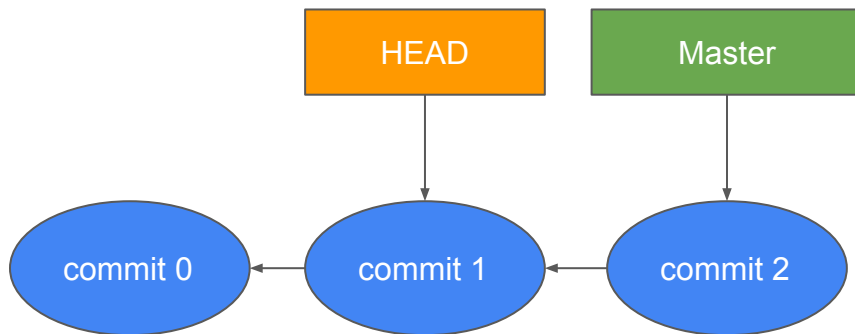
```
> echo 'example 2' >  
example_2.txt  
> echo 'adding new line 3' >>  
example.txt  
> git add .  
> git commit -m "commit 2"
```



Ví dụ về nhánh

- Trỏ HEAD vào commit bất kỳ
 - > `git checkout commit_1_id`
- Detached HEAD: HEAD không trỏ đến nhánh

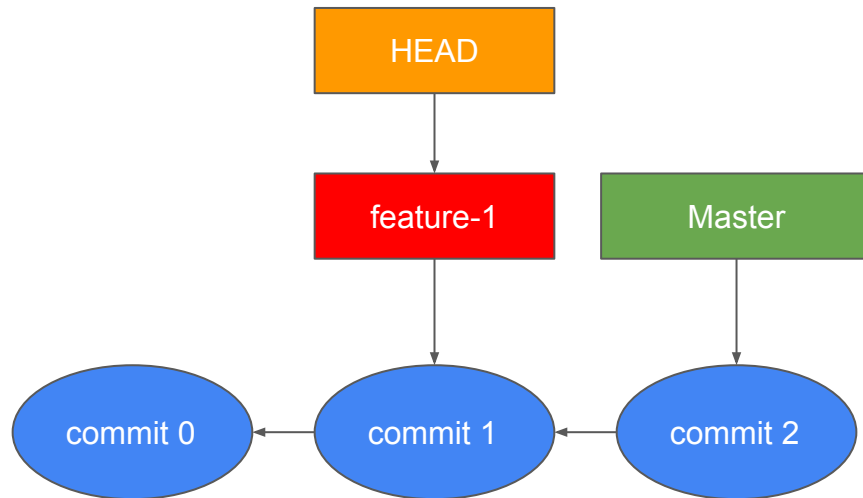
➤ HEAD & Master trỏ đến commit khác nhau!



Ví dụ về nhánh

- Rẽ nhánh và chuyển HEAD qua nhánh mới

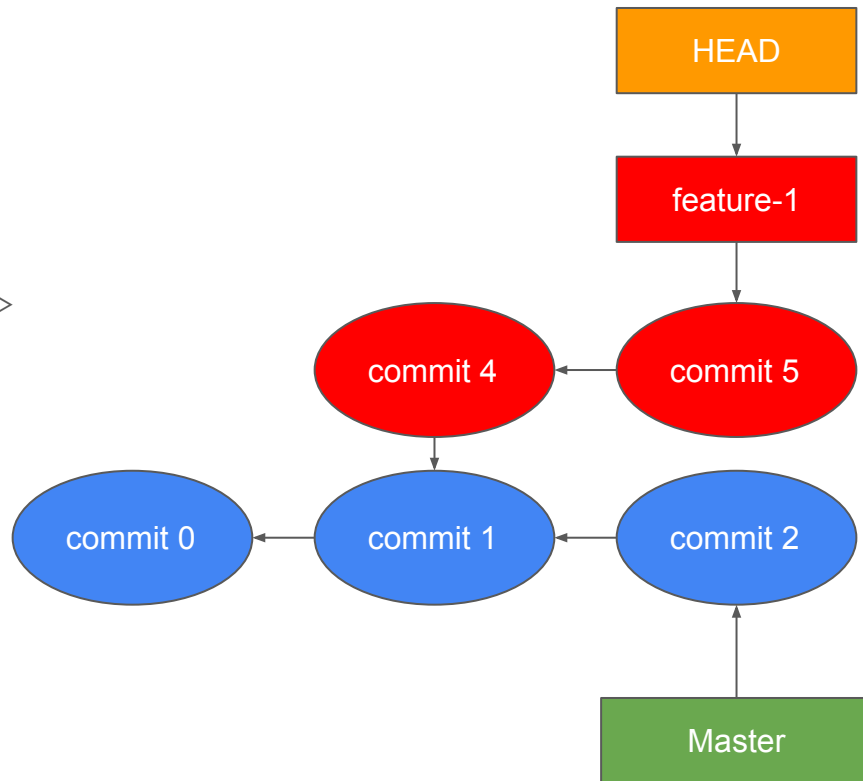
```
> git branch feature-1  
> git checkout feature-1
```



Ví dụ về nhánh

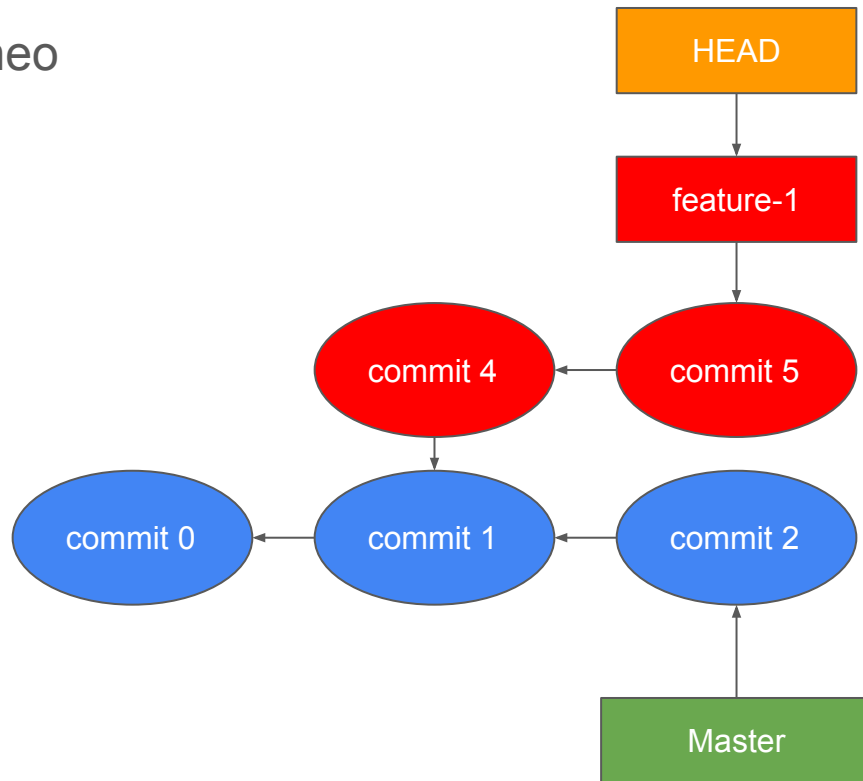
- Thêm commit cho feature branch

```
> echo 'adding new line' >  
example.txt  
> git commit -am "commit 4"  
> echo 'adding new line 2' >>  
example.txt  
> git commit -am "commit 5"
```



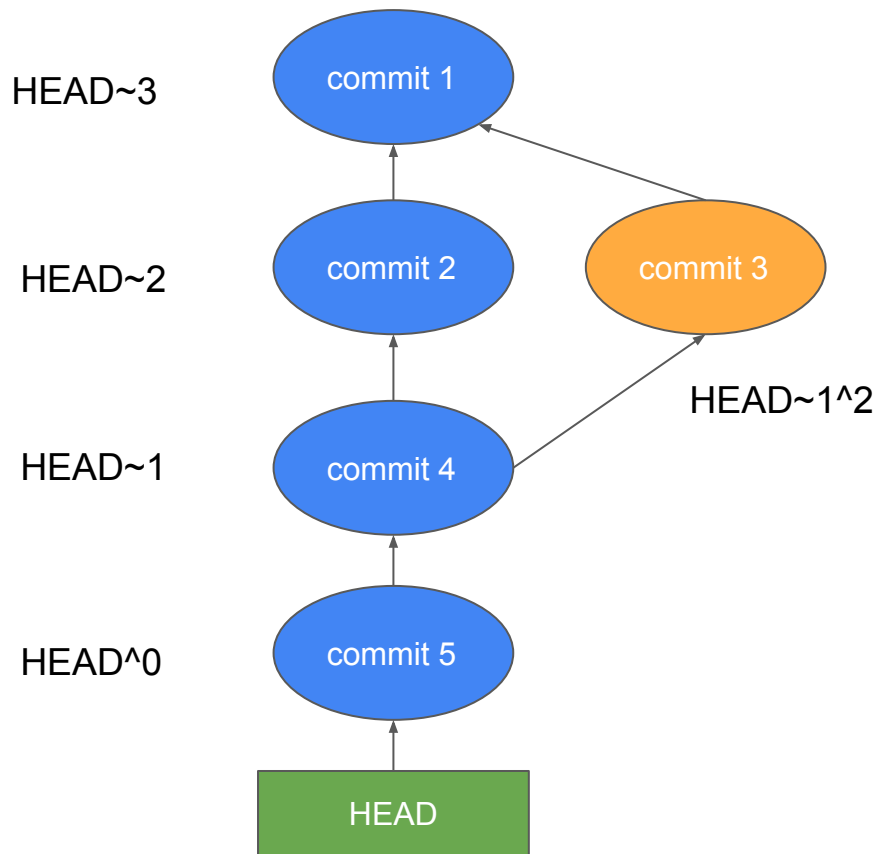
Ví dụ về nhánh

- Master & feature-1 branch đã rẽ theo 2 nhánh khác nhau!



Relative commit

- Một cách khác để trở đến commit mà không cần nhớ chính xác ID
- $\langle \text{commit} \rangle \sim [\langle n \rangle]$: tổ tiên thứ n của commit tính theo cha đầu tiên
 - HEAD~1: cha đầu tiên của commit tại HEAD
 - HEAD~2: cha đầu tiên của (HEAD~1)
 - HEAD~3: cha đầu tiên của (HEAD~2)
- $\langle \text{revision} \rangle ^{[\langle n \rangle]}$: cha thứ n của commit
 - HEAD^0: commit đó
 - HEAD^1: cha thứ nhất
 - HEAD^2: cha thứ 2
 - HEAD^3: cha thứ 3



Recap

- Xem danh sách local branch

```
git branch
```

-a: xem tất cả branch local + remote

- Tạo branch mới từ HEAD

```
git branch <branch_name>
```

- Chuyển branch và trở HEAD tới branch mới

```
git checkout <branch_name>
```

- Trở HEAD tới 1 commit bất kỳ

```
git checkout <commit_ref_id>
```

- Tạo branch mới và checkout

```
git checkout -b<new_branch>
```

- Xoá local branch

```
git branch -d <branch_name>
```

Merge nhánh

Thao tác merge nhánh

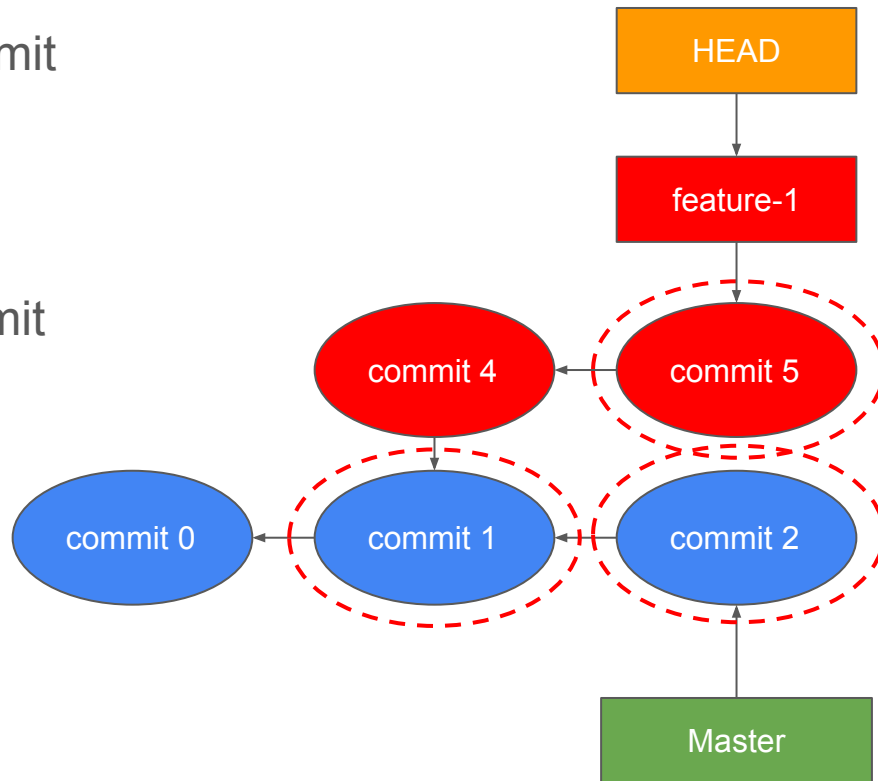
Xử lý merge conflict

Merge nhánh

- Merge: tích hợp các thay đổi từ 1 nhánh vào 1 nhánh khác
- How to merge?
 - Three way merge
 - Fast forward merge

Three way merge

- Three way merge: sử dụng 3 commit cho quá trình merge
 - latest commit of feature-1: commit 5
 - latest commit of Master: commit 2
 - common ancestor: commit 1



Why three way merge?

- Giả sử so sánh file a.txt tại 2 commit mới nhất ở branch Master và feature-1

Line	Master commit	feature-1 commit
1	data A	data 1
2	data B	data B
3	data 3	data C
4	data D	data D
5	data 5	data E

- Line 1, 3, 5 có sự khác nhau

⇒ *Nên chọn thay đổi nào từ Master hay feature ?*

Why three way merge?

- So sánh với common ancestor



- Line 1: feature-1 có thay đổi (**auto-merge**)
- Line 3: Master có thay đổi (**auto-merge**)
- Line 5: cả 2 cùng thay đổi -> **conflict!**

Line	Common ancestor	Master	feature-1
1	data A	data A	data 1
2	data B	data B	data B
3	data C	data 3	data C
4	data D	data D	data D
5	data E	data 5	data G

Why three way merge?

- Quay trở lại với ví dụ
- Thử merge feature-1 vào master

```
> git checkout master
```

```
> git merge --no-ff feature-1
```

```
Auto-merging example.txt
CONFLICT (content): Merge conflict in example.txt
Automatic merge failed; fix conflicts and then commit the result.
```

⇒ Conflict!

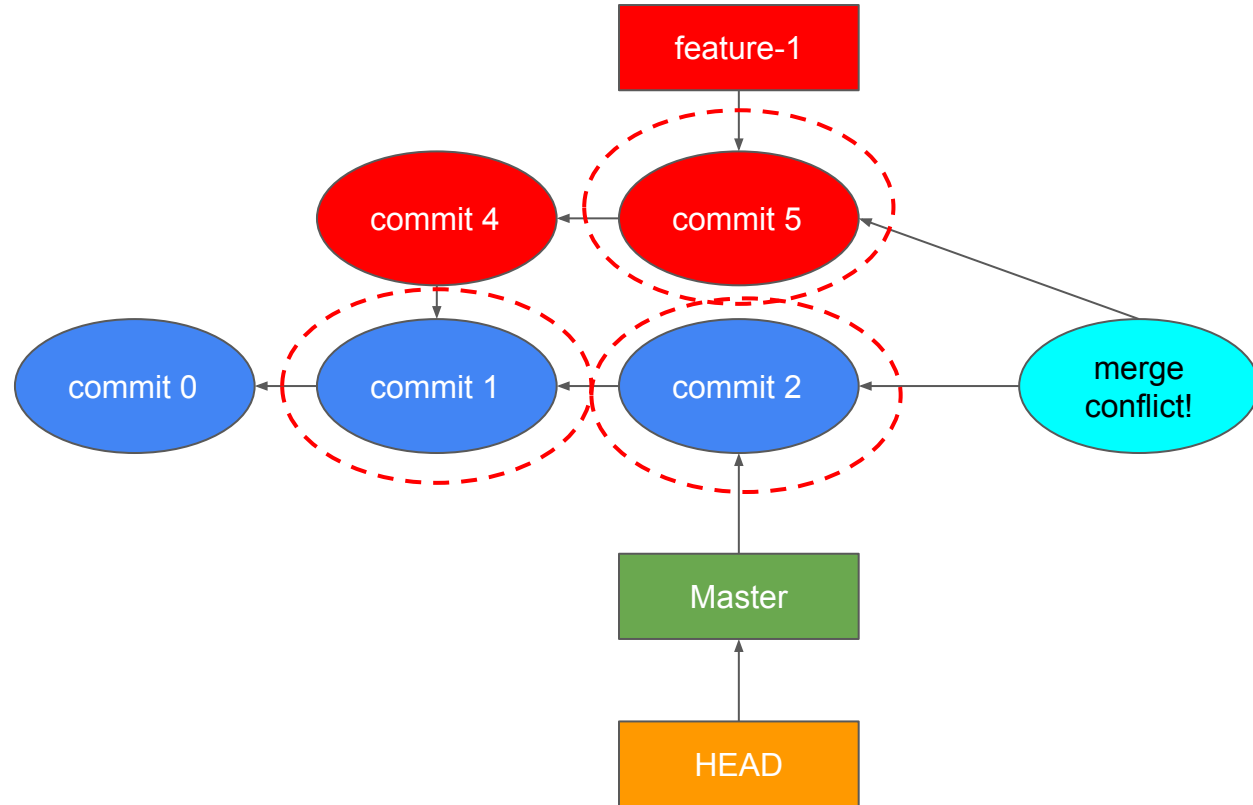
```
> cat example.txt
```

```
<<<<<<< HEAD
example 1
adding new line 3
=====
adding new line
adding new line 2
>>>>>>> feature-1
```

Conflict resolution

Cách giải quyết conflict

Merge conflict



Conflict resolution

- Conflict example.txt sẽ có dạng như bên với các ký tự đặc biệt (bôi đậm)
- Để resolve merge conflict:
 - Manual: dùng editor xoá các ký tự đặc biệt và chọn thay đổi cần giữ lại
 - Graphic Tool: P4Merge

```
> cat example.txt
```

```
<<<<<<< HEAD
```

```
example 1
```

master
branch

```
adding new line 3
```

```
=====
```

```
adding new line
```

```
adding new line 2
```

feature
branch

```
>>>>>>> feature-1
```

Manual resolution

- Chọn thay đổi cần giữ lại
- Xoá ký tự đặc biệt
- Stage & commit

```
> git commit -am "commit 6"
```

- Cách đơn giản nếu ít merge conflict

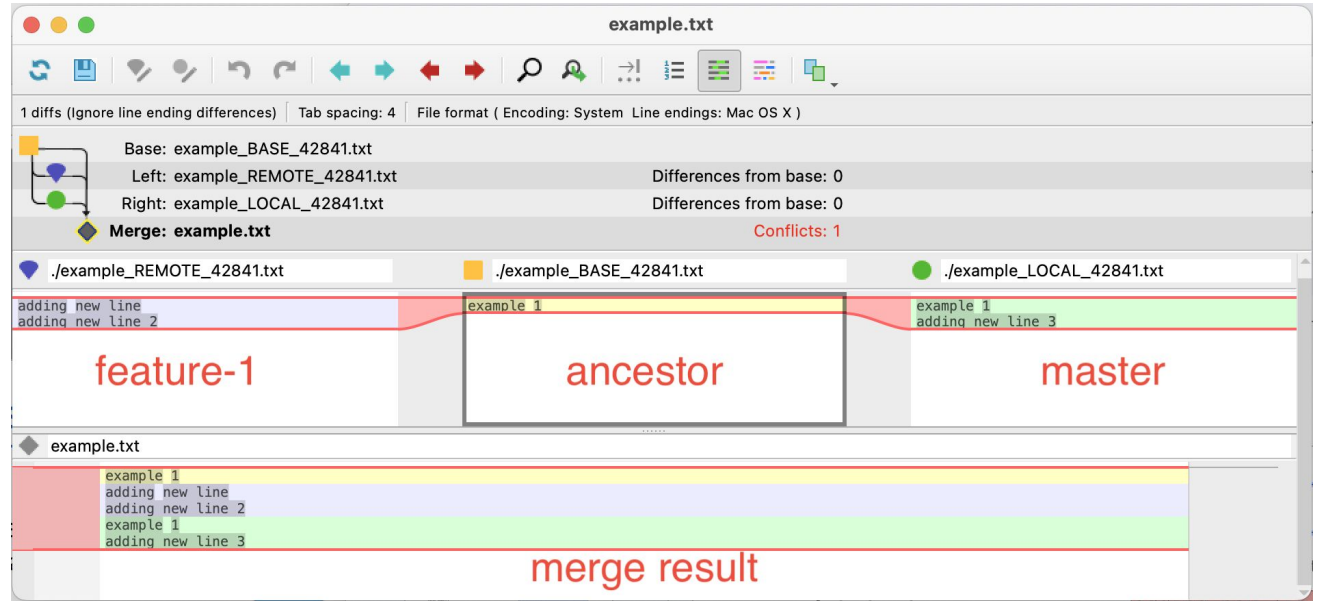
```
> cat example.txt
```

```
adding new line  
adding new line 2
```

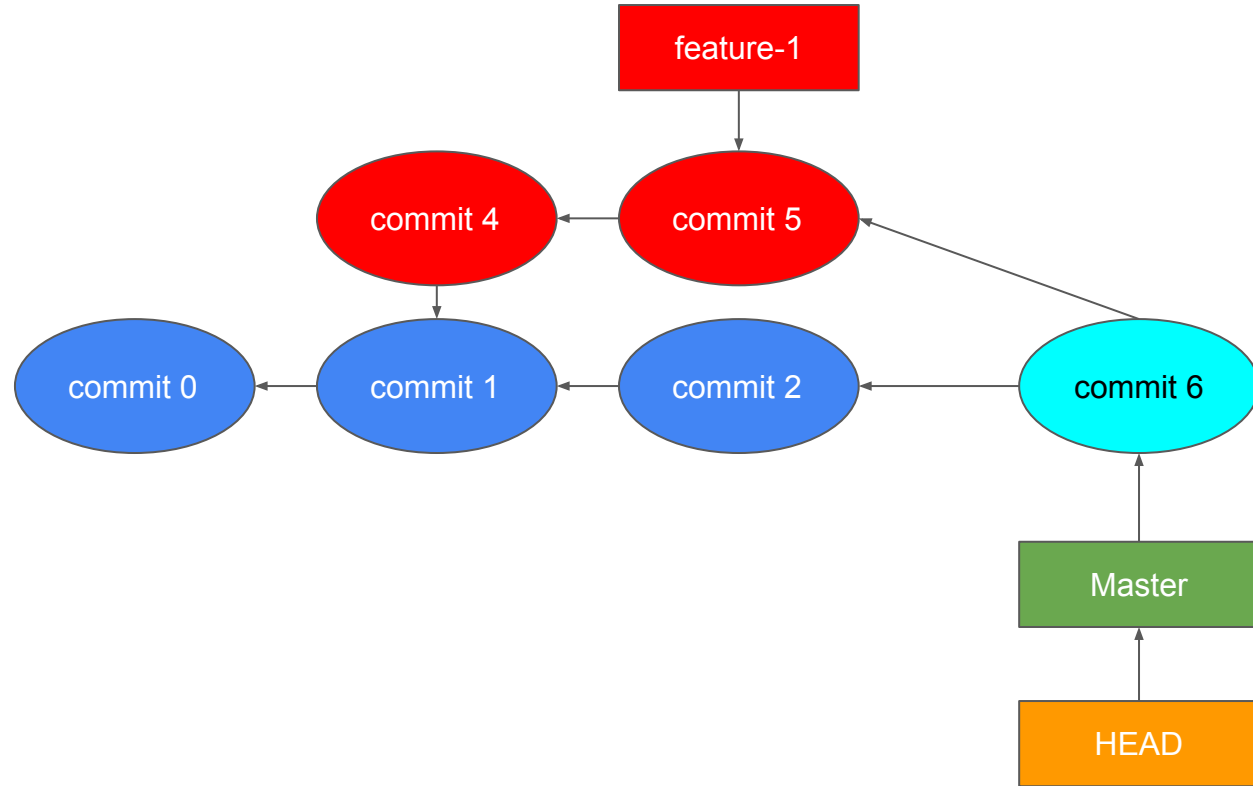
```
adding new line 3
```

Merge tool (P4Merge)

Nếu có nhiều
merge conflict



Merge result



git log history

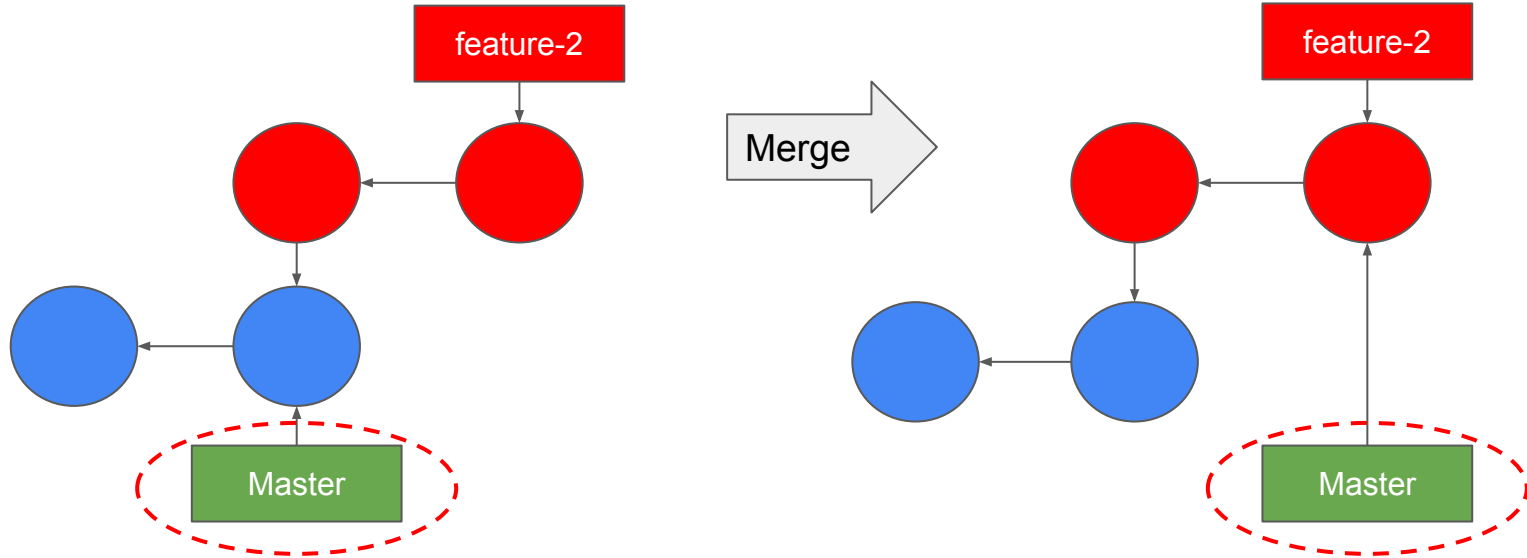
```
> git log --oneline  
--graph --decorate
```

```
*    07952ea (HEAD -> master) commit 6  
|\  
| * cfccbe9 (feature-1) commit 5  
| * d60589e commit 4  
* | 5eeae38 commit 2  
|/  
* ae4a0d9 commit 1  
* 68741b0 add Introduction  
* 851eb90 add data.txt  
* 72b58dc update README.md  
* f428fc1 (origin/master, origin/HEAD) init
```

Happy Path - Fast Forward Merge

- Xảy ra khi các nhánh cần merge (feature) có common ancestor là latest commit của nhánh được merge (master)
 - Không có merge conflict!
- Không tạo merge commit, chỉ trở branch tới merge commit mới

Happy Path - Fast Forward Merge



git log history

```
> git log --oneline  
--graph --decorate
```

- master branch trở vào
feature-2 latest commit

```
* 512e774 (HEAD -> master, feature-2) commit 8  
* b26506a commit 7  
* 07952ea (feature-b) commit 6  
|\  
| * cfccbe9 (feature-1) commit 5  
| * d60589e commit 4  
* | 5eeae38 commit 2  
|/  
* ae4a0d9 commit 1  
* 68741b0 add Introduction  
* 851eb90 add data.txt  
* 72b58dc update README.md  
* f428fc1 (origin/master, origin/HEAD) init
```

git log history preserve branch off

- Bỏ qua fast forward merge và giữ nguyên branch history

> git merge --no-ff feat-2

- Empty merge commit sẽ được tạo

```
*    a6a83c4 (HEAD -> master) Empty commit to retain branch off
| \
| * 512e774 (feature-2) commit 8
| * b26506a commit 7
| /
*    07952ea (feature-b) commit 6
| \
| * cfccbe9 (feature-1) commit 5
| * d60589e commit 4
* | 5eeae38 commit 2
| /
* ae4a0d9 commit 1
* 68741b0 add Introduction
* 851eb90 add data.txt
* 72b58dc update README.md
* f428fc1 (origin/master, origin/HEAD) init
```

Recap

- Danh sách local branch

```
git branch
```

- Tạo branch mới

```
git branch <branch_name>
```

- Trở HEAD tới branch

```
git checkout <branch_name>
```

- Trở HEAD tới commit

```
git checkout <commit_ref_id>
```

- Tạo branch mới và trở HEAD

```
git checkout -b<new_branch>
```

- Xoá local branch

```
git branch -d <branch_name>
```

- Merge branch

```
git merge
```

```
git mergetool
```

Quiz



Câu lệnh nào để so sánh thay đổi đã được commit với thay đổi sau khi dùng git add?

A. `git diff --staged HEAD`

B. `git log --graph --online`

C. `git status`

D. `git add`

Quiz



git ref nào tham chiếu commit mới nhất của nhánh master - nhánh hiện tại?

A. HEAD

B. tag

C. HEAD~3

Quiz



A cần merge feature branch vào master branch, tuy nhiên xảy ra merge conflict.
A có thể merge theo phương pháp?

A. three way merge

B. fast forward merge

Quiz



A đang xử lý merge conflict và muốn giữ lại thay đổi từ branch hiện tại. Đây là kết quả đúng mà A đã thực hiện?

```
<<<<<< HEAD
print("AB")
=====
print("BC")
>>>>>>
feature-a
```

A. `print("AB")`

B. `print("BC")`

C. `print("AB")`
`print("BC")`

Quiz



A có thể sử dụng câu lệnh nào để huỷ và bắt đầu lại quá trình merge?

- A. `git merge --abort`
- B. `git checkout -b <branch>`
- C. `git log --graph --online`

Quiz



Phương pháp merge nào sẽ tạo commit mới?

A. fast forward merge

B. three way merge

5. Làm việc với remote repository

Đồng bộ local và remote
repository

Remote Repository

- Trong chương này, chúng ta sẽ sử dụng GitHub để quản lý remote repository

Cấu hình Remote Repository

- Add remote repository

```
> git remote add <remote_name> <remote_url>
```

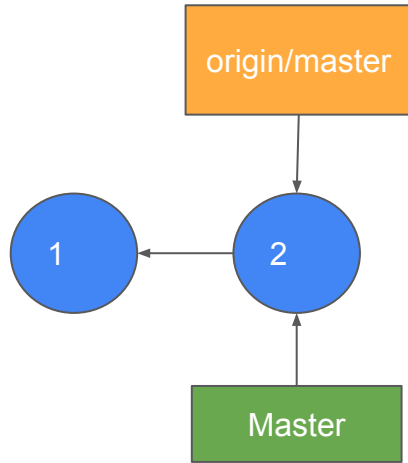
- Mặc định remote name là **origin**
- Xem danh sách remote repository

```
> git remote -v
```

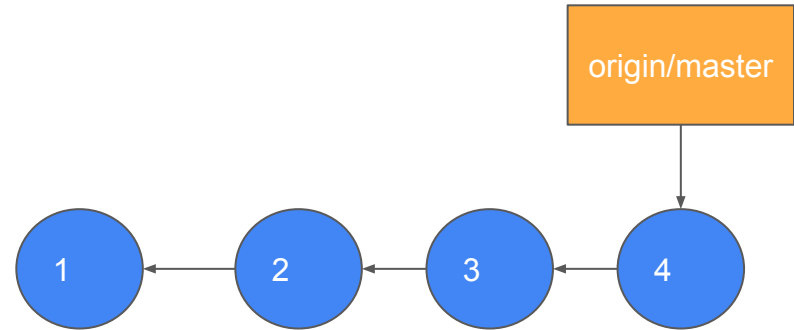
```
→ hello-world-example git:(master) git remote -v  
origin https://github.com/huanpc22/hello-world-example.git (fetch)  
origin https://github.com/huanpc22/hello-world-example.git (push)
```


Sự khác nhau giữa Local & Remote Repo

Local



Remote



git fetch

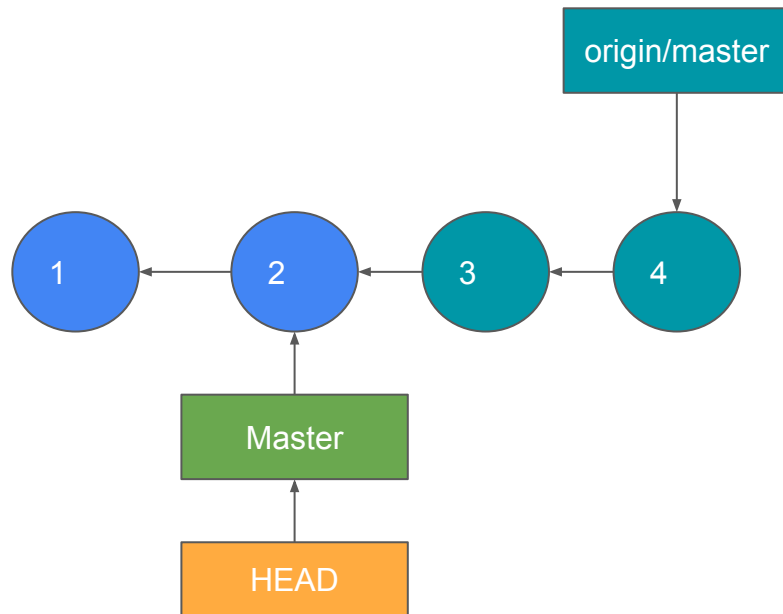
- Cập nhật local git repo với những thay đổi mới nhất từ remote repo
 - commit
 - branch
 - etc.

```
> git fetch origin master  
> git status
```

```
On branch master  
Your branch and 'origin/master' have diverged,  
and have 11 and 3 different commits each, respectively.  
  (use "git pull" to merge the remote branch into yours)  
  
nothing to commit, working tree clean
```

git fetch

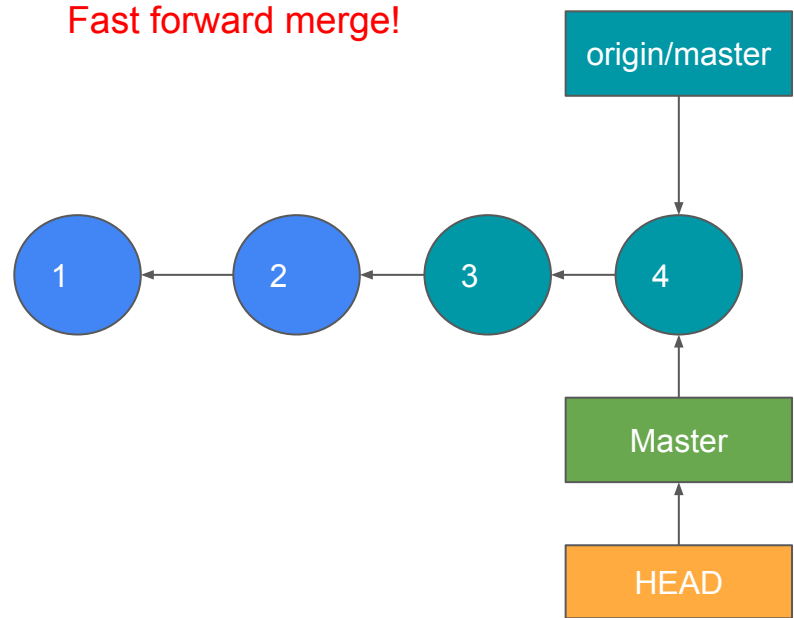
- origin/master branch được cập nhật
- Master branch không được cập nhật



Cập nhật master branch

- Để cập nhật Master về origin/master

```
> git merge origin/master
```



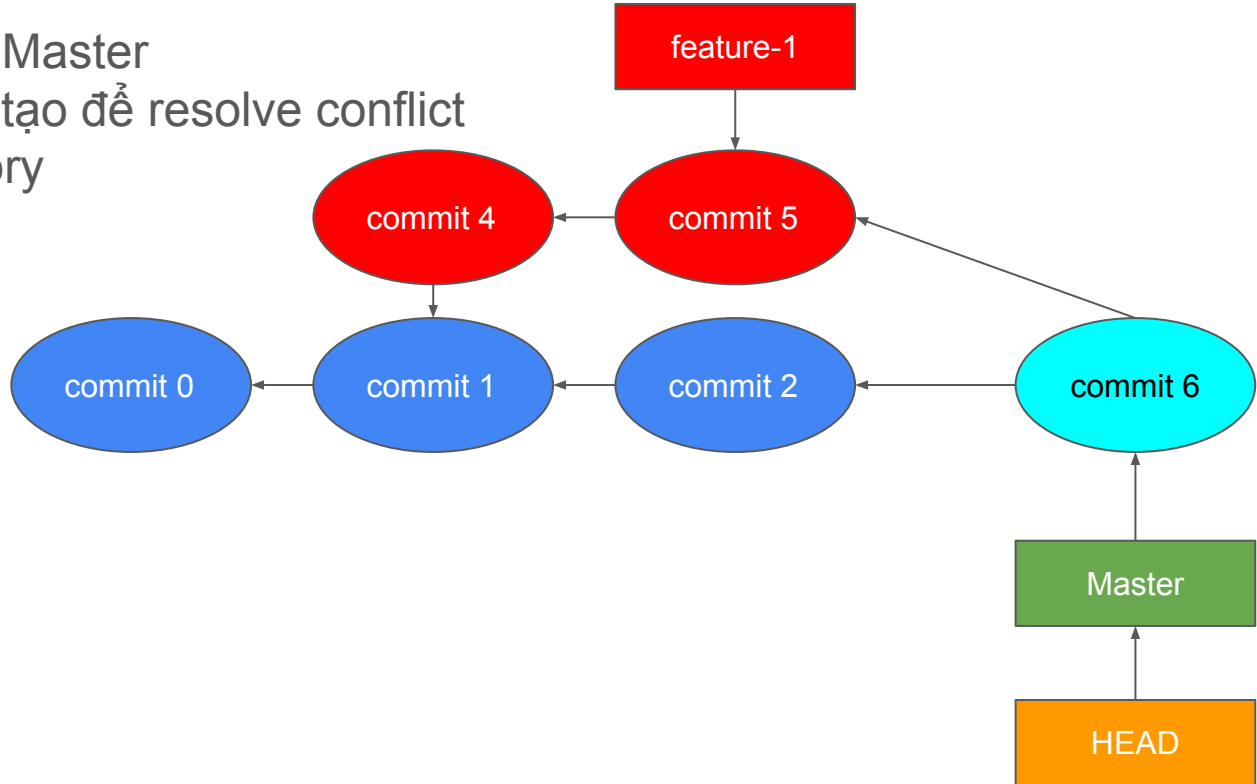
git pull

- git pull = git fetch + git merge

```
> git pull origin/master
```

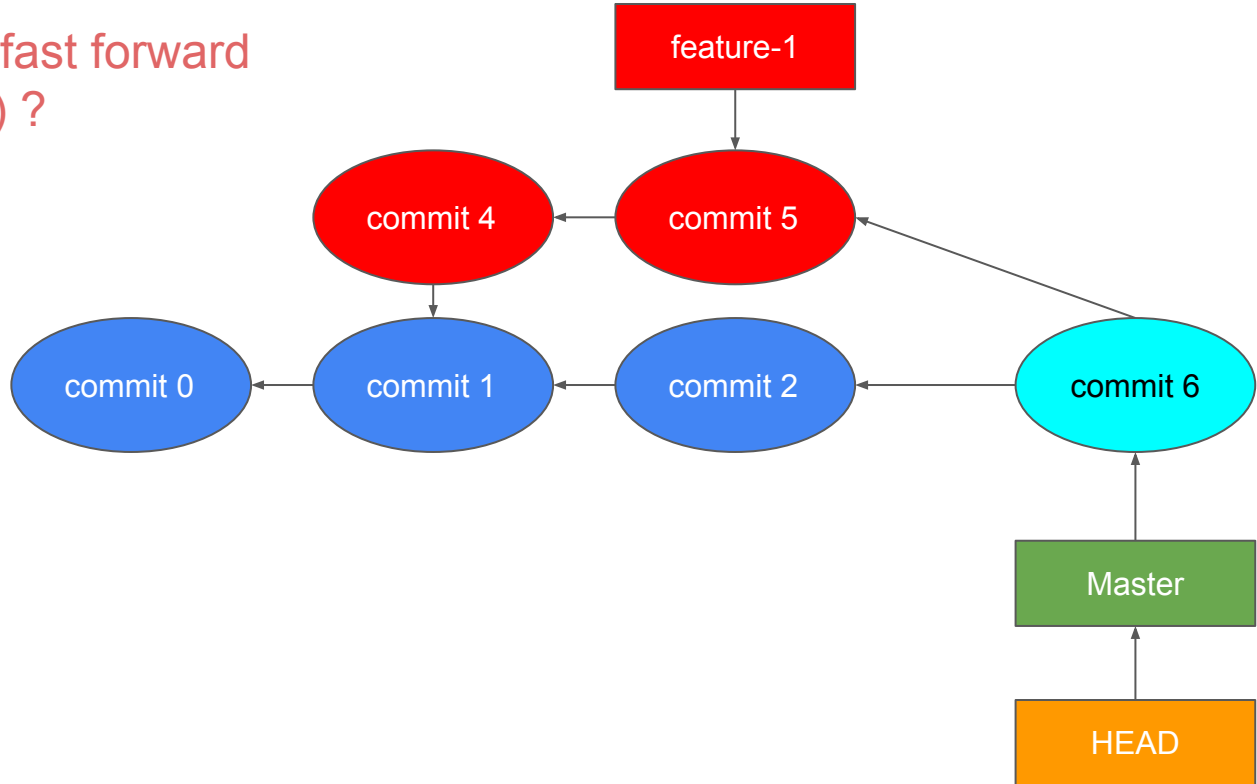
Recall git merge & conflict resolution

- Merge feature-1 vào Master
- Merge commit được tạo để resolve conflict
- Bảo toàn git log history



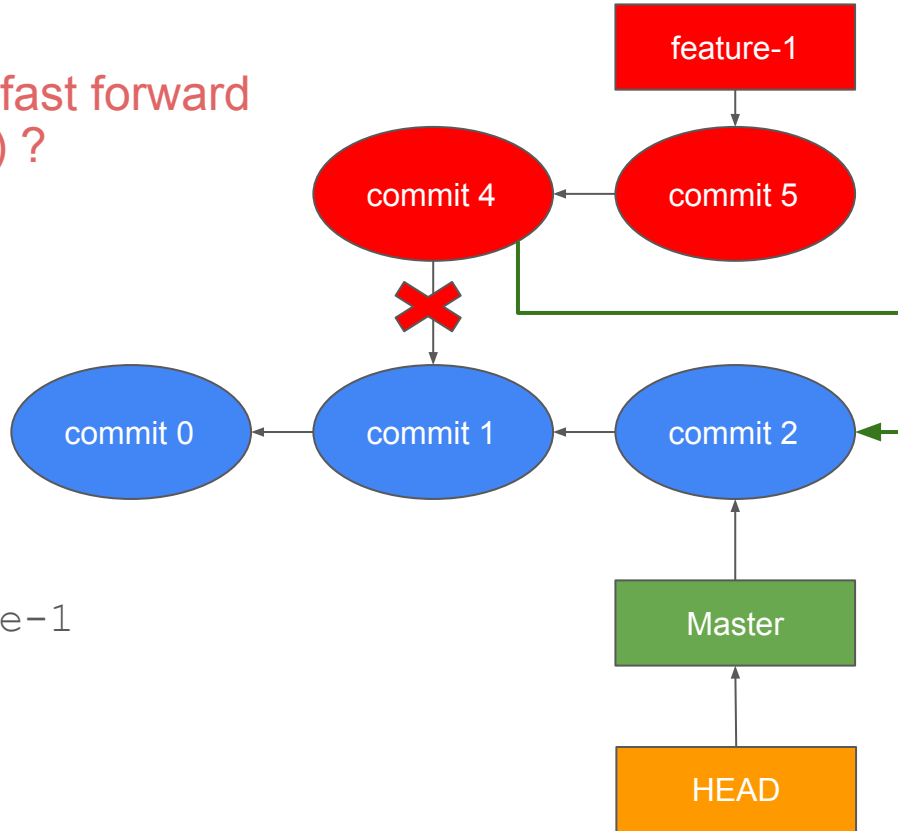
Recall git merge & conflict resolution

? Làm thế nào để dùng fast forward merge (no merge conflict) ?



Recall git merge & conflict resolution

? Làm thế nào để dùng fast forward merge (no merge conflict) ?



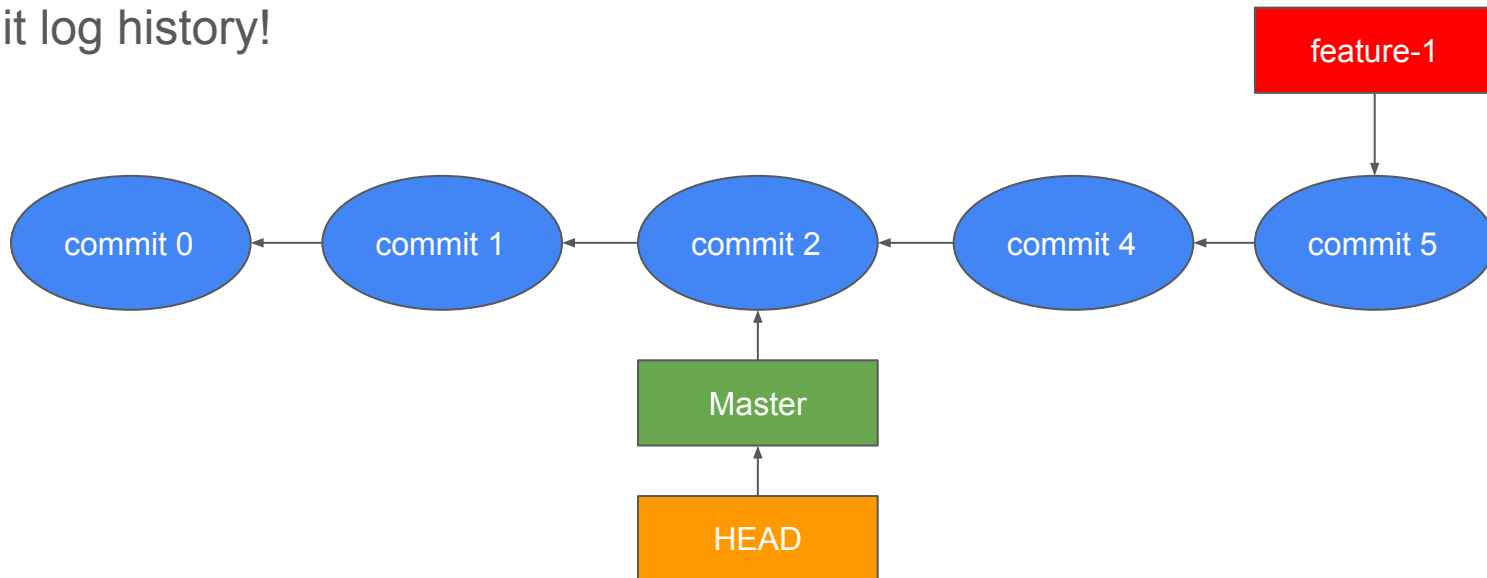
```
> git checkout feature-1  
> git rebase master
```


git rebase

```
> git checkout feature-1
```

```
> git rebase master
```

- Đưa commit tách nhánh nối vào sau commit mới nhất của master
- Viết lại git log history!



Resolve conflict: rebase vs merge

	Rebase	Merge
Dùng khi	Cần gộp và viết lại log history từ base branch, loại bỏ phân nhánh	Bảo toàn log history và nhánh

git rebase flow

```
> git rebase  
# fix conflict (manual or mergetool)  
> git add  
> git rebase --continue
```

git pull with rebase

- Trường hợp khi remote branch có conflict với local branch
- `git pull`
 - `--rebase`: dùng rebase để resolve conflict
 - `--no-rebase`: dùng merge để resolve conflict

git pull with rebase

- Before

```
* 9cd9490 (origin/master, origin/HEAD) Create data.txt
* 44a6c6c Update README.md
* 2652981 Update README.md
| * a6a83c4 (HEAD -> master) Empty commit to retain branch off
| | \
| | * 512e774 (feature-2) commit 8
| | * b26506a commit 7
| | /
| * 07952ea (feature-b) commit 6
| | \
| | * cfccbe9 (feature-1) commit 5
| | * d60589e commit 4
| | * 5eeae38 commit 2
| | /
| * ae4a0d9 commit 1
| * 68741b0 add Introduction
| * 851eb90 add data.txt
| * 72b58dc update README.md
| /
* f428fc1 init
```

- After

```
* a6c612a (HEAD -> master) commit 8
* 9e33455 commit 7
* 440eba0 commit 5
* ed578db commit 4
* c189d2c commit 2
* 28d8b57 commit 1
* a232c46 add Introduction
* f1e8412 add data.txt
* 65f8651 Rebase from remote master
* 9cd9490 (origin/master, origin/HEAD) Create data.txt
* 44a6c6c Update README.md
* 2652981 Update README.md

| * 512e774 (feature-2) commit 8
| * b26506a commit 7
| * 07952ea (feature-b) commit 6
| | \
| | * cfccbe9 (feature-1) commit 5
| | * d60589e commit 4
| | * 5eeae38 commit 2
| | /
| * ae4a0d9 commit 1
| * 68741b0 add Introduction
| * 851eb90 add data.txt
| * 72b58dc update README.md
| /
* f428fc1 init
```

rewrite history

duplicate commit

git pull with merge

- Before

```
* 9cd9490 (origin/master, origin/HEAD) Create data.txt
* 44a6c6c Update README.md
* 2652981 Update README.md
| * a6a83c4 (HEAD -> master) Empty commit to retain branch off
| | \
| | * 512e774 (feature-2) commit 8
| | * b26506a commit 7
| | /
| * 07952ea (feature-b) commit 6
| | \
| | * cfccbe9 (feature-1) commit 5
| | * d60589e commit 4
| | * 5eeae38 commit 2
| | /
| * ae4a0d9 commit 1
| * 68741b0 add Introduction
| * 851eb90 add data.txt
| * 72b58dc update README.md
| /
* f428fc1 init
```

- After

```
* b55f930 (HEAD -> master) resolve conflict with remote
| \ merge commit
| * 9cd9490 (origin/master, origin/HEAD) Create data.txt
| * 44a6c6c Update README.md
| * 2652981 Update README.md
| * a6a83c4 Empty commit to retain branch off
| | \
| | * 512e774 (feature-2) commit 8
| | * b26506a commit 7
| | /
| * 07952ea (feature-b) commit 6
| | \
| | * cfccbe9 (feature-1) commit 5
| | * d60589e commit 4
| | * 5eeae38 commit 2
| | /
| * ae4a0d9 commit 1
| * 68741b0 add Introduction
| * 851eb90 add data.txt
| * 72b58dc update README.md
| /
* f428fc1 init
```

Preserve
history

git push

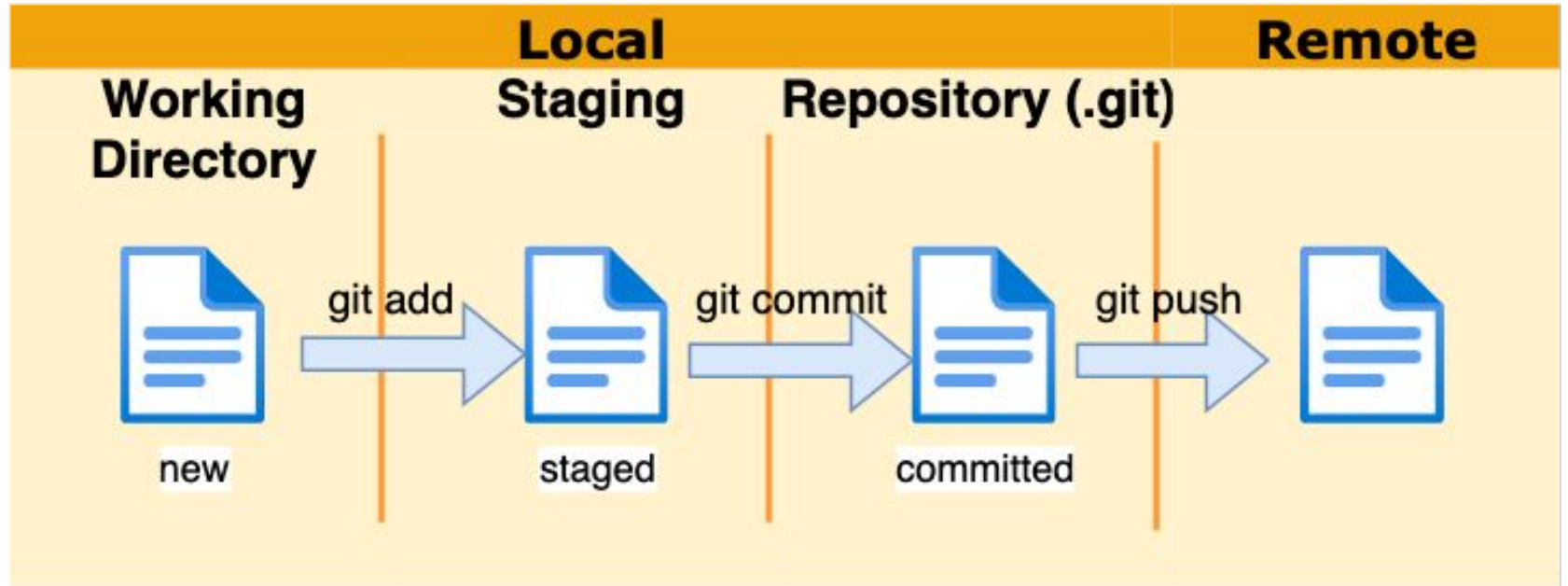
- Cập nhật remote branch với những thay đổi từ local branch
- Best practice
 - Luôn luôn cập nhật các nhánh ở local trước khi thêm các thay đổi
 - Flow:

```
> git pull origin master  
# resolve merge conflict  
> git push origin master
```

- Why?
 - Reuse code, commit đã có ở remote
 - Giảm thiểu khả năng xảy ra code conflict
 - Nếu có conflict, git sẽ không cho push trước khi pull

```
+ hello-world-example git:(master) git push  
To https://github.com/huanpc22/hello-world-example.git  
! [rejected]        master -> master (non-fast-forward)  
error: failed to push some refs to 'https://github.com/huanpc22/hello-world-example.git'  
hint: Updates were rejected because the tip of your current branch is behind  
hint: its remote counterpart. Integrate the remote changes (e.g.  
hint: 'git pull ...') before pushing again.  
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

git push



git push

- Push local branch to different remote branch

```
git push -u origin <local_branch>:<remote_branch>
```

- Overwrite remote branch

```
git push --force origin master
```

- Đổi remote branch

```
git branch --set-upstream-to=<remote_branch>
```

Q&A

Quiz



Câu lệnh để cập nhật tất cả thay đổi từ remote repo và merge vào local repo?

- A. git pull
- B. git merge
- C. git clone
- D. git fetch

Quiz



Câu lệnh để cập nhật tất cả thay đổi từ local repo lên remote?

- A. git pull
- B. git commit -am
- C. git clone
- D. git push

Quiz



Câu lệnh để cập nhật các nhánh mới từ remote repo mà không merge vào local repo?

- A. git pull
- B. git checkout
- C. git fetch

Quiz



Đây là lý do để dùng rebase thay cho merge

- A. Duy trì 1 nhánh trong commit history
- B. Bảo toàn phân nhánh trong commit history
- C. Khi đang làm việc trên nhánh chung

Quiz



Câu lệnh nào đồng bộ và ghi đè commit history từ local lên remote repo, có khả năng làm mất commit đã có?

A. `git push -f`

B. `git rebase`

C. `git reset -- hard`

6. Nâng cao

stash
cherry pick
undo change
squashing
reflog

git stash

git stash

- Use case 1
 - Bạn đang code hăng say trên feature branch
 - Production code có bug, cần hot fix gấp
 - Bạn muốn tạo branch mới để hot fix nhưng vẫn lưu giữ những đoạn sửa đổi dang dở từ feature branch để quay lại hoàn thiện sau đó

? Làm cách nào để quay trở lại hoàn thành feature branch?

git stash

- Use case 1
 - Bạn đang code hăng say trên feature branch
 - Production code có bug, cần hot fix gấp
 - Bạn muốn tạo branch mới để hot fix nhưng vẫn lưu giữ những đoạn sửa đổi dang dở từ feature branch để hoàn thiện sau đó

? Làm cách nào để quay trở lại hoàn thành feature branch?

`git commit ?`

git stash

- Use case 2

- Bạn đang làm việc trên feature branch A để giải quyết vấn đề nào đó
- Feature B có giải pháp tốt hơn nên bạn muốn merge nó vào branch hiện tại
- Tuy nhiên để merge thành công thì bạn cần xoá bỏ những thay đổi gần đây chưa được commit
- Bạn vẫn muốn giữ lại những sửa đổi đó sau khi merge xong để tiếp tục công việc

? Làm cách nào để merge ?

git stash

- Use case 2

- Bạn đang làm việc trên feature branch A để giải quyết vấn đề nào đó
- Feature B có giải pháp tốt hơn nên bạn muốn merge nó vào branch hiện tại
- Tuy nhiên không thể merge vì có conflict với sửa đổi gần đây chưa được commit của branch A
- Bạn vẫn muốn giữ lại những sửa đổi đó sau khi merge xong

? Làm cách nào để merge ?

`git reset --hard` có thể giúp xóa bỏ những sửa đổi để merge thành công, nhưng không thể giữ lại chúng

git stash

- git stash giúp cất giữ các sửa đổi chưa được commit và khôi phục lại version đã được commit trước đó

```
> echo "new line for stashing" >> example.txt  
> git stash -m "backup my changes"
```

- git stash cất giữ các thay đổi trong một cấu trúc dữ liệu giống như stack (last in first out)

```
> git stash list  
stash@{0}: On test-stash: my third backup  
stash@{1}: On test-stash: my second backup  
stash@{2}: On test-stash: my first backup
```

git stash

- Lấy lại sửa đổi gần nhất (top of stack)

```
> git stash apply
```

- Lấy lại sửa đổi và xoá khỏi stack

```
> git stash pop
```

- Xem chi tiết sửa đổi gần nhất

```
> git stash show
```

- Đưa thay đổi gần nhất vào branch và xoá khỏi stack

```
> git stash branch <branch_name>
```

- Xem chi tiết sửa đổi bất kỳ

```
> git stash show stash@{1}
```

- Lấy lại sửa đổi bất kỳ

```
> git stash pop stash@{2}
```

- Xoá toàn bộ stash stack

```
> git stash clear
```

- Stash untracked file

```
> git stash -u
```

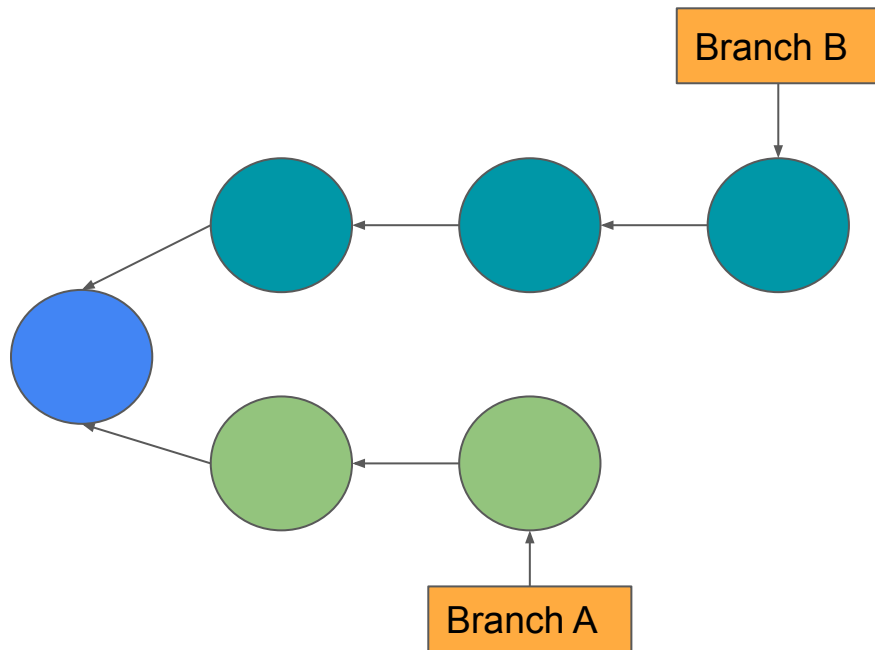
Cherry pick

Cherry pick

- **Use case:** bạn đang làm việc trên branch nhưng muốn apply một vài commit từ branch khác. Lý do có thể là tái sử dụng code từ branch khác.
- Mang một vài commit từ branch B vào branch A

> `git cherry-pick <commit_hash>`

- Cherry pick chỉ lấy những thay đổi trong commit áp dụng vào branch hiện tại



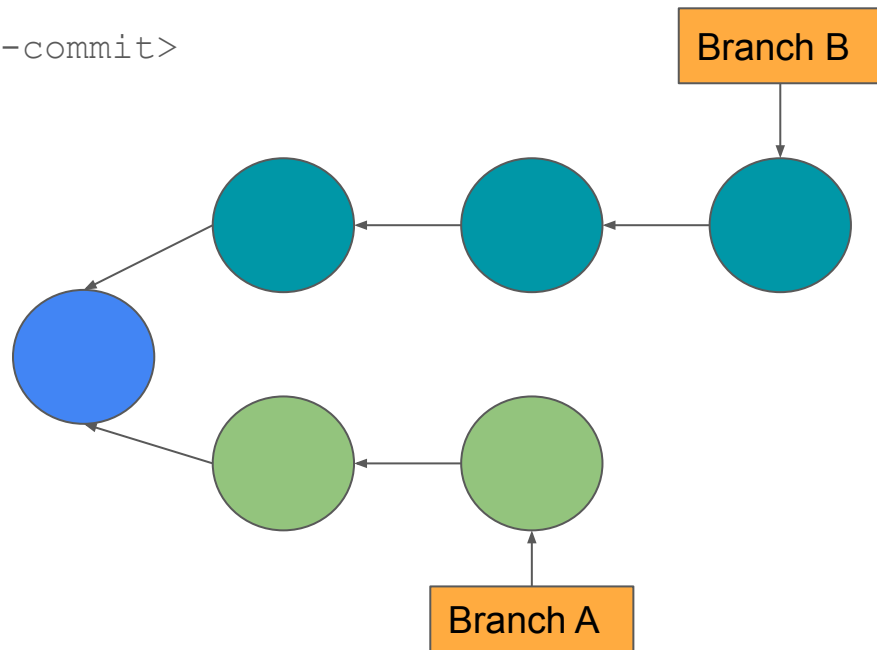
Cherry pick

- Cherry pick range commit

```
> git cherry-pick <older-commit>..<newer-commit>
```

- Cherry pick last 2 commit

```
> git cherry-pick branch-B~2..branch-B
```



Undo change

Undo change

- Undo change (Huỷ bỏ thay đổi) là một tính năng quan trọng của VCS, giúp khôi phục file trở về version trước đó.
- Tình huống:
 - Bạn đang làm việc trên feature branch A, file B
 - Bạn nhận ra rằng những sửa đổi gần đây có sai sót, nên muốn “reset” branch A về lại các version trước đó của file B
 - Undo changes!

Undo change options

- **git checkout**
 - Huỷ bỏ sửa đổi và đưa file trở về phiên bản trước đó
- **git reset**
 - Huỷ bỏ sửa đổi ở local repo, staging, hoặc working directory
 - Có thể thay đổi commit history
- **git revert**
 - Huỷ bỏ sửa đổi bằng cách tạo Reverted (đảo ngược) commit
 - Thường áp dụng khi cần huỷ bỏ commit đã được **publish lên remote repo**
 - Không làm thay đổi commit history

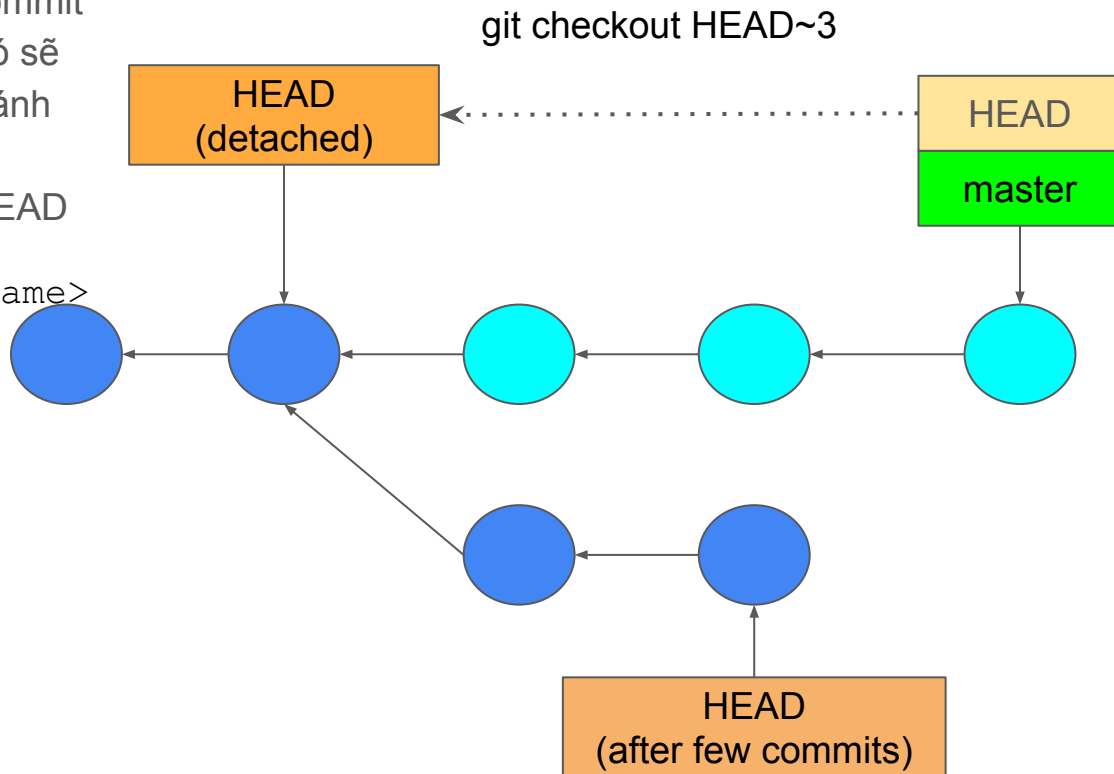
git checkout

- Trỏ HEAD về commit bất kỳ
- Phục hồi file về version tương ứng với commit
- Không làm thay đổi lịch sử commit

git checkout

- Detached HEAD nếu checkout commit
- Các commit được khởi tạo sau đó sẽ được rẽ sang nhánh mới khỏi nhánh ban đầu
- Để tạo nhánh mới từ detached HEAD

```
> git switch -c <new-branch-name>
```



git reset

- Huỷ bỏ commit, có thể làm thay đổi commit history
- Đưa HEAD và branch trở về commit trước đó

```
> git reset <commit>
```

- 3 kiểu reset
 - --soft
 - --mixed
 - --hard

Reset types

Type	Uncommit	Unstage thay đổi ở staging	Xoá thay đổi ở working directory
--soft	v		
--mixed	v	v	
--hard	v	v	v

? git checkout và git reset --soft giống hay khác nhau?

Reset types

Type	Uncommit	Unstage thay đổi ở staging	Xoá thay đổi ở working directory
--soft	v		
--mixed	v	v	
--hard	v	v	v

? git checkout và git reset --soft giống hay khác nhau?

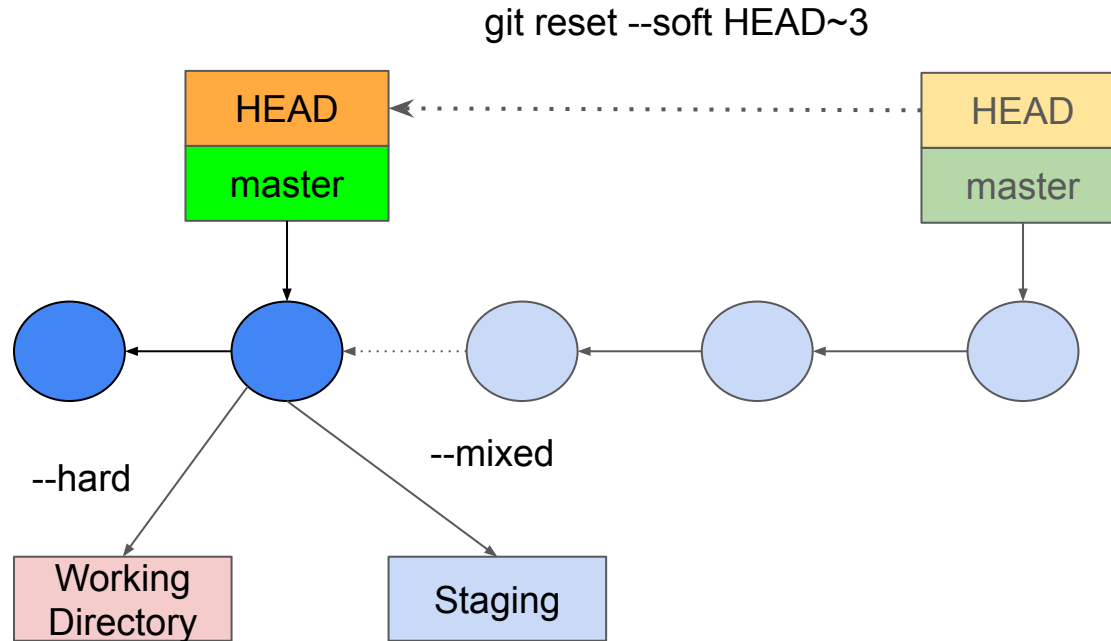
Đều dịch HEAD về commit bất kỳ

```
> git checkout <commit>
```

```
> git reset --soft <commit>
```

Khác?

Reset types



Reset types

```
> git checkout -b reset-example  
  
> echo "line 1" >> "reset.txt"  
  
> git add .  
  
> git commit -m "add example for reset"  
  
# 1 change committed  
  
> echo "line 2" >> "reset.txt"  
  
# 1 change in working directory
```

```
On branch reset-example  
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:   reset.txt  
  
no changes added to commit (use "git add" and/or "git commit -a")
```

```
> git reset --soft HEAD~1
```

```
+ hello-world-example git:(reset-example) ✖ git status  
On branch reset-example  
Changes to be committed:  
  (use "git restore --staged <file>..." to unstage)  
        modified:   .gitignore  
        new file:   reset.txt      revert from commit  
  
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:   reset.txt      remain change
```

```
> git reset --mixed HEAD~1
```

```
+ hello-world-example git:(reset-example) ✖ git status  
On branch reset-example  
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:   .gitignore  
  
Untracked files:  
  (use "git add <file>..." to include in what will be committed)  
        reset.txt ← unstage change  
  
no changes added to commit (use "git add" and/or "git commit -a")
```

Reset types

```
> git reset --hard HEAD~1
```

reset.txt trở về trạng thái trước đó (chưa tồn tại)

- --hard có thể huỷ commit ở các branch chưa được publish hoặc các sửa đổi chưa được commit/stage, vì vậy cần cẩn thận khi dùng

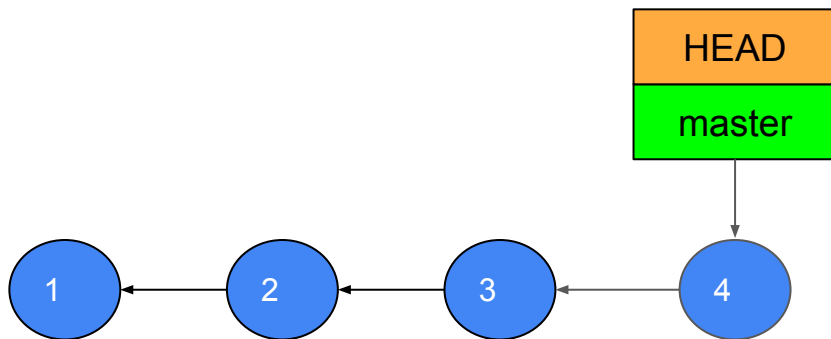
```
→ hello-world-example git:(reset-example) x git ls-files
.gitignore
README.md
data.txt
example.txt
example_2.txt
example_3.txt
index.html
script.js
style.css
```

git revert

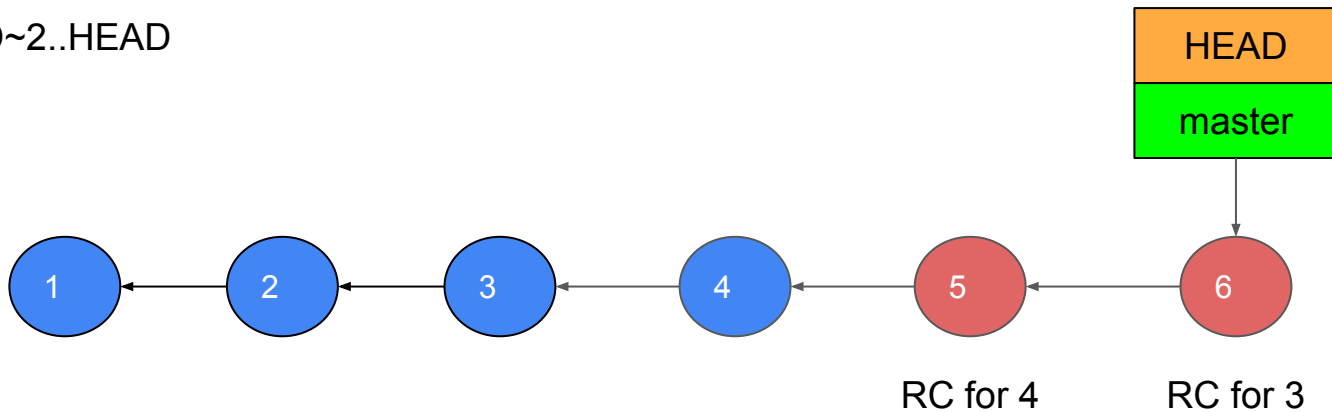
- Huỷ bỏ sửa đổi bằng cách tạo Reverted commit để đảo ngược các sửa đổi
- Thường áp dụng khi cần huỷ bỏ commit đã được publish lên remote repo
- Không thay đổi commit history

git revert

Before



git revert HEAD~2..HEAD



Quiz



A lỡ push 1 commit chưa hoàn thiện lên remote repo, A cần làm gì để huỷ bỏ commit đó?

- A. `git revert`
- B. `git commit --amend`
- C. `git reset`
- D. `git checkout -- <file_name>`

Quiz



A đang làm việc trên feature branch. A nhận ra những thay đổi gần đây khá lộn xộn, muốn xóa bỏ chúng và phục hồi về 2 commit trước đó. A có thể dùng?

A. `git reset --hard HEAD~2`

B. `git reset --soft HEAD~2`

C. `git reset HEAD~2`

D. `git reset --hard HEAD~1`

Squashing

Squashing

- Tình huống: Bạn nhận ra rằng 10 commit đã tạo rải rác những sửa đổi nhỏ và sẽ là tốt hơn nếu gộp chung vào 1 commit
- Squash: Gộp tất cả thay đổi từ các commit vào 1 commit duy nhất
- Lưu ý: squash sẽ thay đổi commit history, vì vậy không nên squash các sửa đổi đã được push hoặc merge

Squashing

- Cách 1: dùng interactive rebase

```
> git checkout -b squash-example
> echo "line 1" >>
squash_data.txt
> git add .
> git commit -m "first commit"
> echo "line 2" >>
squash_data.txt
> git commit -m "second commit"
echo "line 3" >> squash_data.txt
> git commit -m "third commit"
```

Squashing

- Cách 1: dùng interactive rebase (continue)

```
> git rebase -i HEAD~3  
# text editor sẽ mở ra, thay pick  
bằng squash
```

```
pick 0802cc5 first commit  
squash f2f6a07 second commit  
squash 8619716 third commit  
#  
# Rebase 8d3af6e..8619716 onto 8d3af6e (3 commands)  
#  
# Commands:  
# p, pick <commit> = use commit  
# r, reword <commit> = use commit, but edit the commit message  
# e, edit <commit> = use commit, but stop for amending  
# s, squash <commit> = use commit, but meld into previous commit
```

```
> git log --oneline --graph  
--decorate --all
```

Squashing

- Cách 2: dùng reset

```
> git reset --soft HEAD~3  
> git commit -m "first commit"  
> git log --oneline --graph  
--decorate --all
```

- Cách này sẽ dời HEAD trong khi vẫn giữ nguyên các sửa đổi đã được stage

Reflog

Reflog

- Tình huống
 - Reset về nhầm commit, các sửa đổi gần đây không còn
 - Rebase nhầm commit chứa version không mong muốn
 - etc.

⇒ Cần phục hồi trạng thái về trước khi thực hiện undo change!

Reflog

- Tình huống
 - Reset về nhầm commit, các sửa đổi gần đây không còn
 - Rebase nhầm commit chứa version không mong muốn
 - etc.

⇒ Cần phục hồi trạng thái về trước khi thực hiện undo change!

git reflog có thể giúp phục hồi trạng thái!

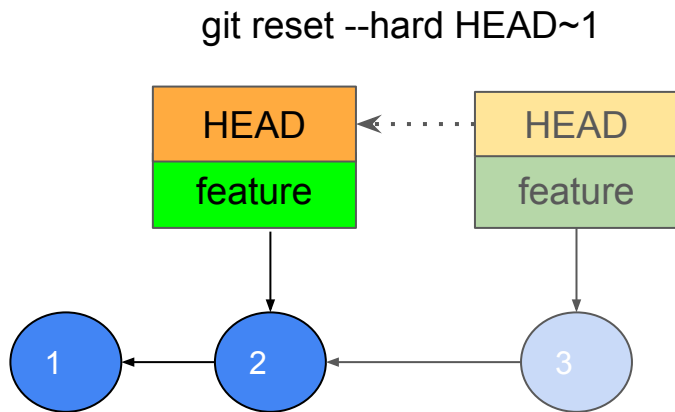
Reflog

- git reset không xoá commit
- git rebase tạo commit mới
- commit sẽ chỉ bị xoá đi (bởi garbage collector) khi không còn được tham chiếu đến
- git reflog cho ta xem lịch sử các thay đổi tác động đến commit history và cho phép phục hồi về 1 thời điểm xác định

Reflog - Ví dụ

- Ban đầu, chúng ta có branch feature với 3 commit

```
> git checkout -b "reflog-example"
> echo "line 1" >> relog_data.txt
git add .
> git commit -m "first commit"
> echo "line 2" >> relog_data.txt
> git commit -am "second commit"
> echo "line 3" >> relog_data.txt
> git commit -am "third commit"
```



Reflog - Ví dụ

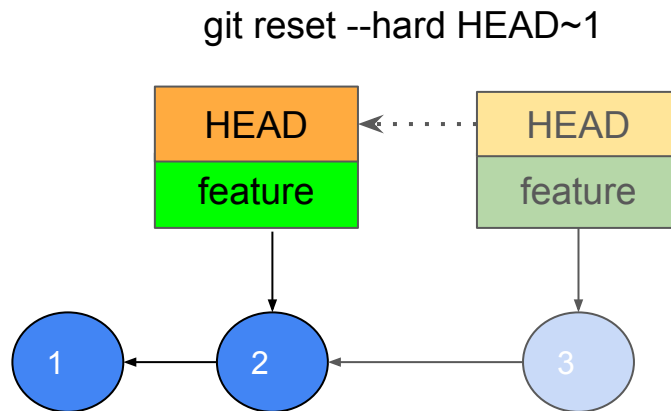
- Reset trạng thái về commit 2 và loại bỏ các thay đổi sau đó

```
> git reset --hard HEAD~1
```

- Xem lịch sử các thay đổi đến HEAD

```
> git reflog
```

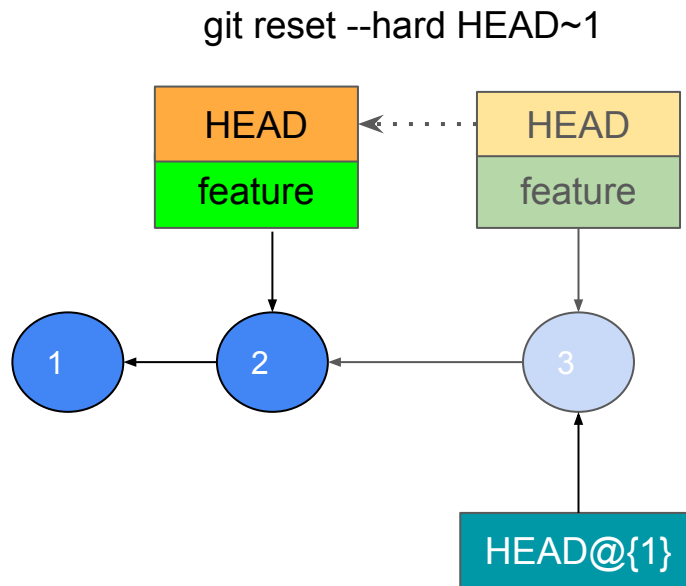
```
fa8507a (HEAD -> reflog-example) HEAD@{0}: reset: moving to HEAD~1
a063453 HEAD@{1}: commit: third commit
fa8507a (HEAD -> reflog-example) HEAD@{2}: commit: second commit
d1e8ac7 HEAD@{3}: commit: first commit
2499d00 HEAD@{4}: checkout: moving from 2499d00467dca1ed90a544455ef6
```



Reflog - Ví dụ

```
fa8507a (HEAD -> reflog-example) HEAD@{0}: reset: moving to HEAD~1
a063453 HEAD@{1}: commit: third commit
fa8507a (HEAD -> reflog-example) HEAD@{2}: commit: second commit
d1e8ac7 HEAD@{3}: commit: first commit
2499d00 HEAD@{4}: checkout: moving from 2499d00467dca1ed90a544455ef6
```

- HEAD@{0} trạng thái hiện tại của HEAD
- HEAD@{1} trạng thái trước đó 1 thao tác
- HEAD@{2} trạng thái trước đó 2 thao tác
- etc.

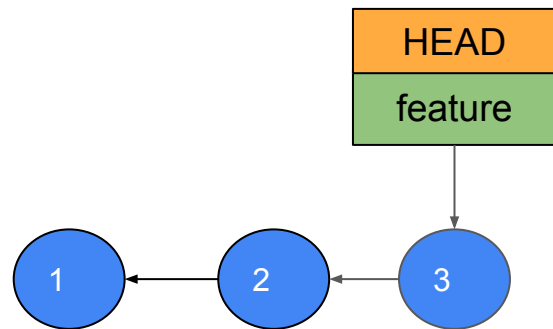


Reflog - Ví dụ

```
fa8507a (HEAD -> reflog-example) HEAD@{0}: reset: moving to HEAD~1
a063453 HEAD@{1}: commit: third commit
fa8507a (HEAD -> reflog-example) HEAD@{2}: commit: second commit
d1e8ac7 HEAD@{3}: commit: first commit
2499d00 HEAD@{4}: checkout: moving from 2499d00467dca1ed90a544455ef6
```

- HEAD@{0} trạng thái hiện tại của HEAD
- HEAD@{1} trạng thái trước đó 1 thao tác
- HEAD@{2} trạng thái trước đó 2 thao tác
- etc.
- Đưa HEAD về lại commit 3

```
> git reset -hard HEAD@{1}
```



Q&A

Quiz



Khi sử dụng interactive rebase, lựa chọn nào dùng để gộp những thay đổi từ 1 commit vào commit trước đó?

A. squash

B. pick

C. reword

D. edit

Quiz



Cách nào sẽ tạo commit mới?

- A. fast forward merge
- B. squashing
- C. three way merge

Quiz



Bạn thêm 1 thay đổi mới vào file A mới và chưa git add thay đổi đó. Cách nào có thể lấy lại thay đổi sau khi dùng git reset --hard HEAD~?

A. git revert

B. git reflog

C. git reset

D. Không thể lấy lại

7. Git Workflow

Sơ qua về các git workflow phổ biến

Git Workflow là gì?

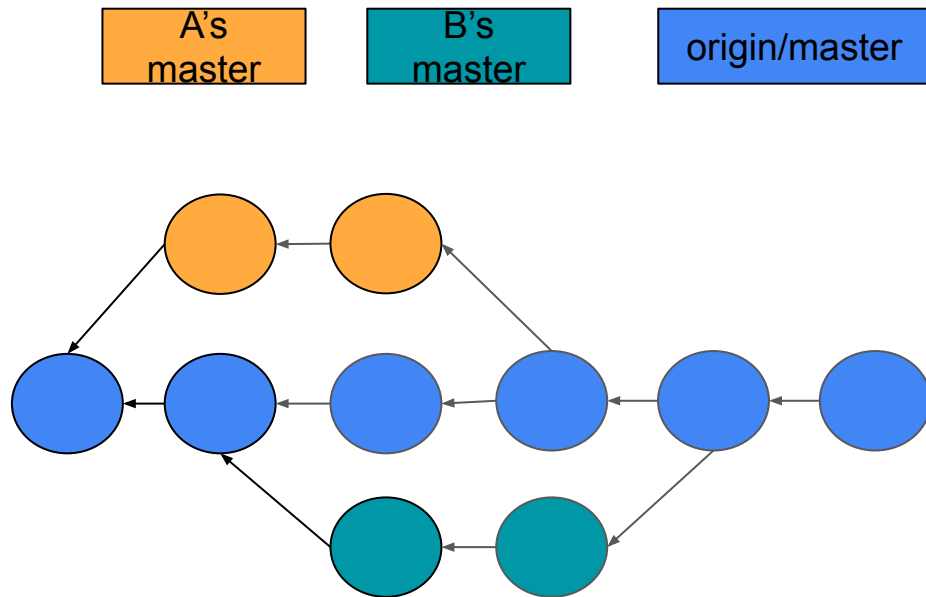
- Tập hợp các quy trình và quy tắc để quản lý git repository khi làm việc nhóm mà mọi thành viên trong dự án tuân theo
- Các khái niệm chính:
 - Môi trường: nơi ứng dụng được thực thi để phục vụ tập người dùng nào đó, ví dụ: môi trường production cho khách hàng, test, staging cho kiểm thử chất lượng nội bộ.
 - Nhánh deploy: được sử dụng để build và deploy code lên môi trường tương ứng, được giữ ổn định và được merge từ nhánh phát triển theo chu kỳ
 - Release: quá trình deploy ứng dụng lên môi trường production
 - Nhánh feature: thành viên sẽ hoạt động trên nhánh này để phát triển tính năng mới hoặc fix bug v.v.
 - Nhánh phát triển: nhánh feature sẽ được merge vào nhánh này, lưu giữ tất cả tính năng đã và đang được phát triển

Những Workflow phổ biến

- Centralized Flow
- Feature Flow
- Git Flow
- GitLab Flow

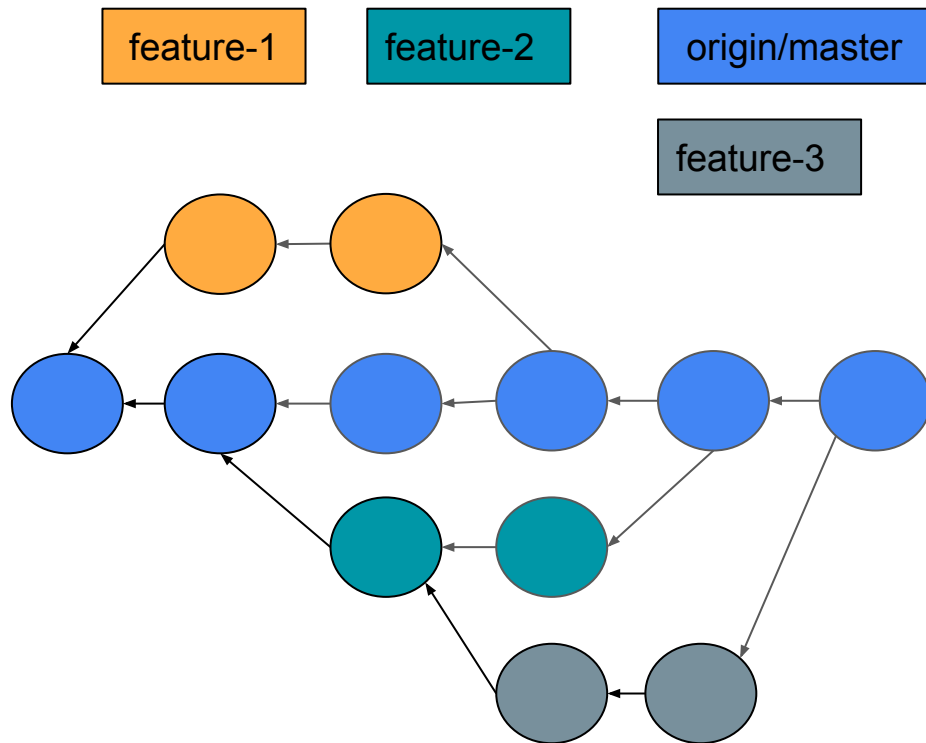
Centralized Flow

- Cả nhóm duy trì 1 remote repository với 1 nhánh mặc định (master)
- Các thành viên clone remote repo về máy cá nhân, thêm thay đổi, commit và push lên nhánh chung
- Vì cùng làm việc trên 1 nhánh, chỉ khi các thay đổi được hoàn tất thì mới được push lên nhánh chung và release
- Đơn giản, phù hợp với team nhỏ, mỗi thành viên duy trì 1 local master duy nhất



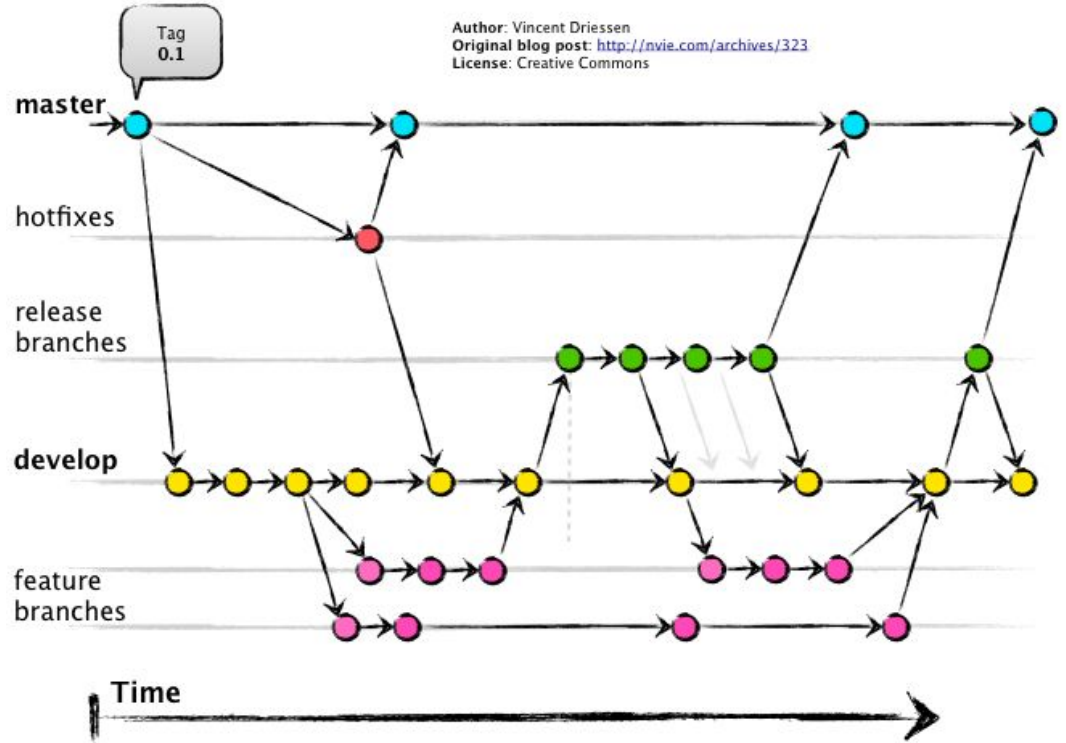
Feature Flow

- Mỗi khi làm việc với 1 feature mới, thành viên sẽ tạo và làm việc trên nhánh feature tương ứng
- Khi cần release, tạo merge request (MR) để merge nhánh feature vào master, leader sẽ review MR và cho phép merge
- Nhánh master sẽ được giữ ổn định và sử dụng cho quá trình release
- Dễ dàng mở rộng, cho phép các thành viên cùng lúc làm việc trên nhiều feature độc lập



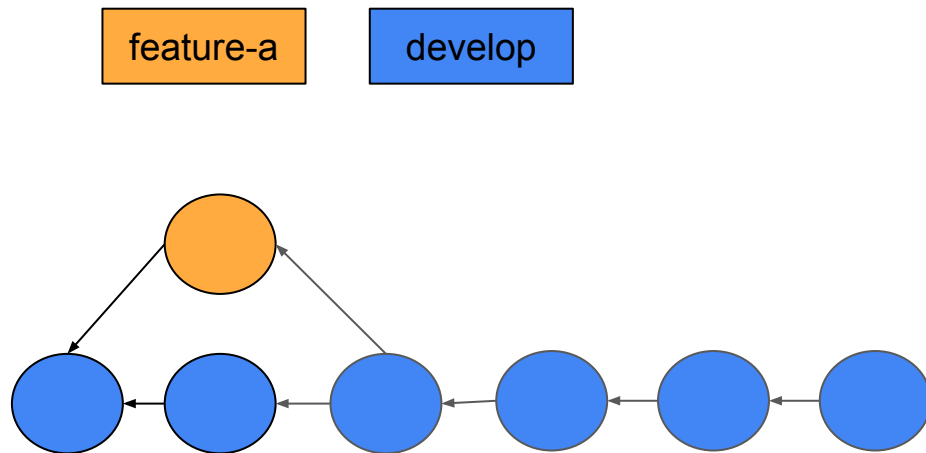
Git Flow

- Nhánh chính
 - Nhánh deploy: master
 - Nhánh phát triển: develop
- Nhánh phụ
 - feature
 - release
 - hotfix
- Tách biệt quá trình phát triển với release
- Nhánh release dùng để chuẩn bị cho quá trình release



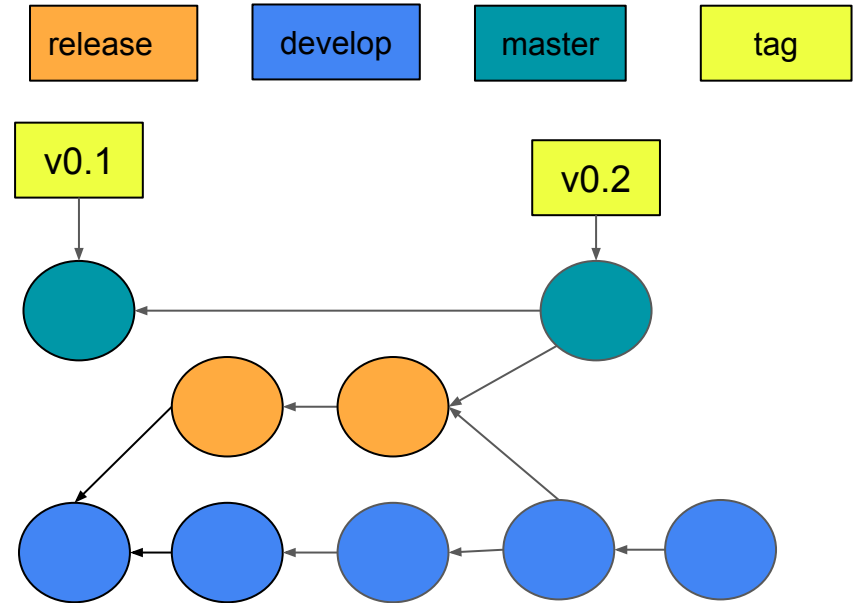
Git Flow

- Khi cần phát triển 1 feature mới, checkout feature branch từ **develop** branch với tên feature/<feature_name>
- Sau khi hoàn thiện feature branch, developer tạo merge request (MR) đến develop
- Sau khi MR được review, feature branch sẽ được merge vào develop



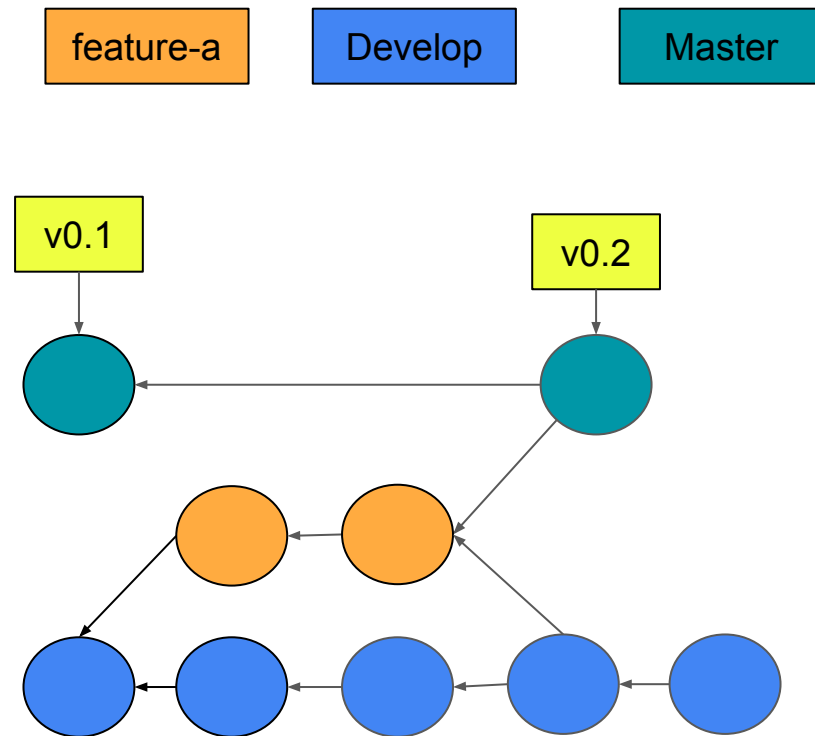
Git Flow - Release

- Để chuẩn bị cho release code, **release** branch sẽ được checkout từ **develop**, kiểm thử và bug fix có thể được thực hiện sau đó.
- Khi đã sẵn sàng để release, **release** branch sẽ được merge vào **master** branch và tạo tag cho quá trình release.
- Những thay đổi ở release branch cần được merge lại develop branch sau quá trình release.



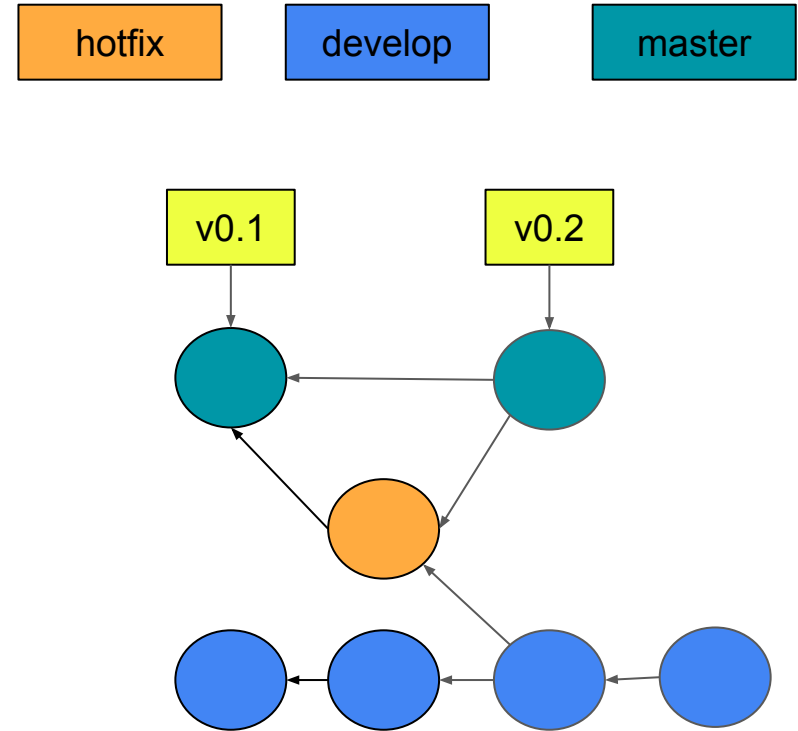
Quiz

- Tại sao cần merge lại vào develop branch



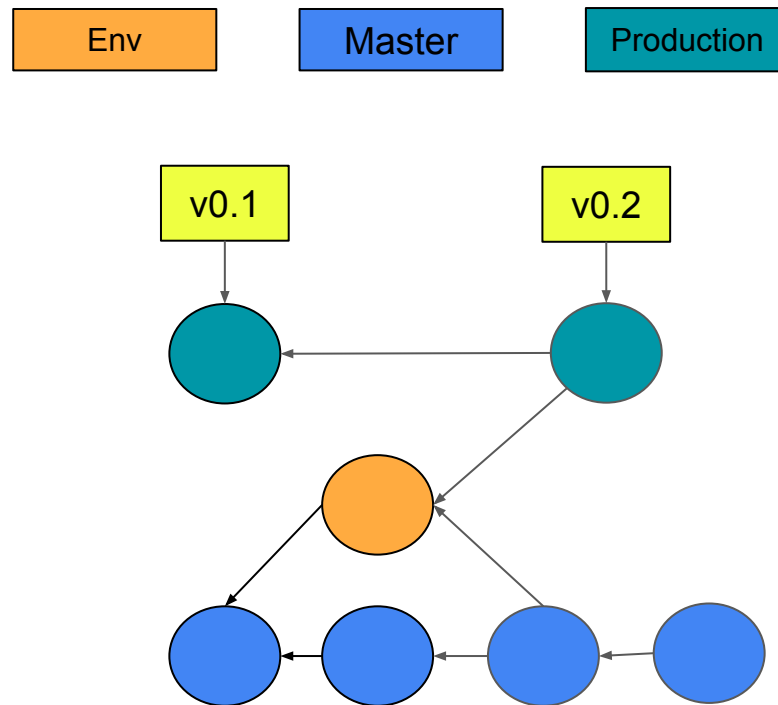
Git Flow - Hotfix

- Khi có bug hay issue trên môi trường production, cần phải sửa gấp, **hotfix** sẽ được tạo từ master branch để sửa issue.
- Sau khi đã được hoàn thiện, hotfix branch sẽ được merge lại master để chuẩn bị cho quá trình release.
- Sau quá trình release, hotfix cần được merge lại vào develop



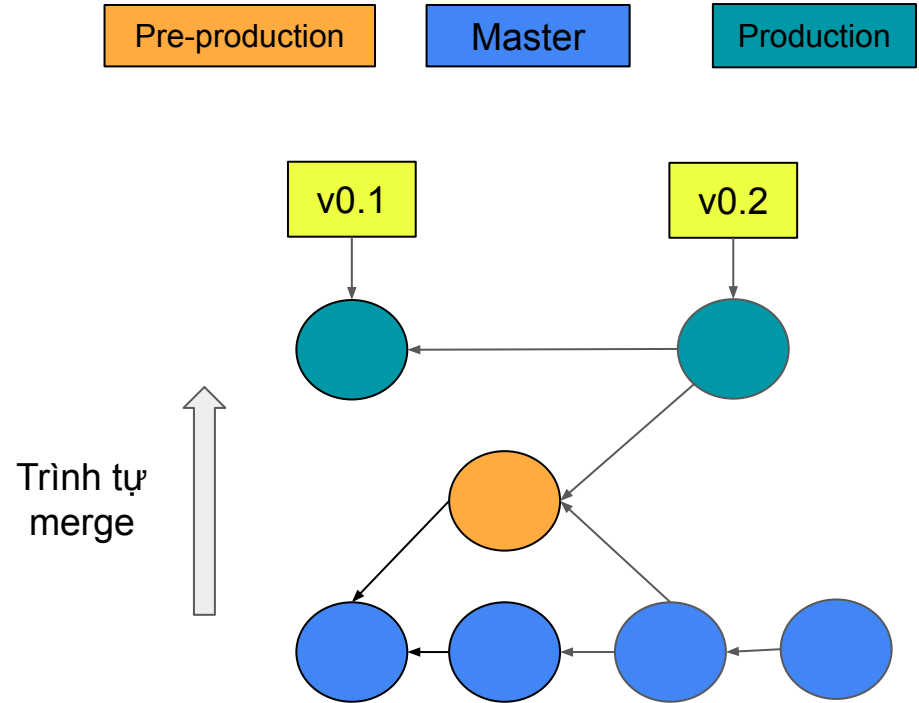
GitLab Flow

- Nhánh chính
 - Deploy: production, **environment**
 - Environment: test, uat, pre-production, etc.
 - Phát triển: master
- Nhánh phụ
 - Feature
 - Hotfix
 - ...
- Dùng Environment branch là bước đệm giữa Master & Production, quá trình phát triển sẽ linh hoạt hơn trong việc deploy và kiểm thử trên các môi trường đa dạng khác nhau



GitLab Flow

- Production branch sẽ được dùng cho quá trình release
- Master branch sẽ được dùng để deploy cho môi trường staging
- Feature branch sẽ được tạo từ master và merge lại vào master
- Master sẽ merge vào Env branch và được deploy lên môi trường tương ứng. Ví dụ Pre-production có thể được sử dụng cho kiểm thử toàn diện trước khi deploy Production



Git Workflow Recap

Flow	Đặc điểm	Ứng dụng
Centralized Flow	Nhánh duy nhất: master	Team nhỏ Ít feature
Feature Flow	Nhánh deploy: master Nhánh phụ: feature	Team vừa Phát triển nhiều feature Quy trình release đơn giản với 1 version trên production
Git Flow	Nhánh deploy: master Nhánh phát triển: develop Nhánh phụ: feature, release, hotfix	Team vừa và lớn Phát triển nhiều feature Cần quy trình release phức tạp với nhiều version
GitLab Flow	Nhánh deploy: production, environment Nhánh phát triển: master Nhánh phụ: feature, hotfix, v.v.	Team vừa và lớn Phát triển nhiều feature và version, cần nhiều bước kiểm thử chất lượng trên các môi trường khác nhau

Q&A

Tài liệu tham khảo

- git scm: <https://git-scm.com/doc>
- Pro Git: <https://git-scm.com/book/en/v2>
- Atlassian Git Tutorials: <https://www.atlassian.com/git/tutorials>
- Google

Thank you for listening!