Table of Contents

Intro to Git

Git sử dụng name và email address:

Để xác định tác giả của commit

git config

git init

tạo 1 git repo trên máy local

Những việc có thể làm trên máy local, ko cần network connection

+making committ
+viewing history
+requesting different between commits

Có hai cách để tạo 1 repo

+Clone từ repo có sẵn
+git init từ máy local
git init GitInPracticeRedux
--> tạo 1 folder có tên là tên repo từ câu lệnh trên, trong đó chưa thư mục .git

Full syntax of any command

git init –help

.git sub-directory

+Invisible: hidden, ko hiển thị trên file browser như Windows, nhưng có thể truy xuất
+Không nên thay đổi nội dung trong thư mục này

Typical workflow

add/commit/checkout

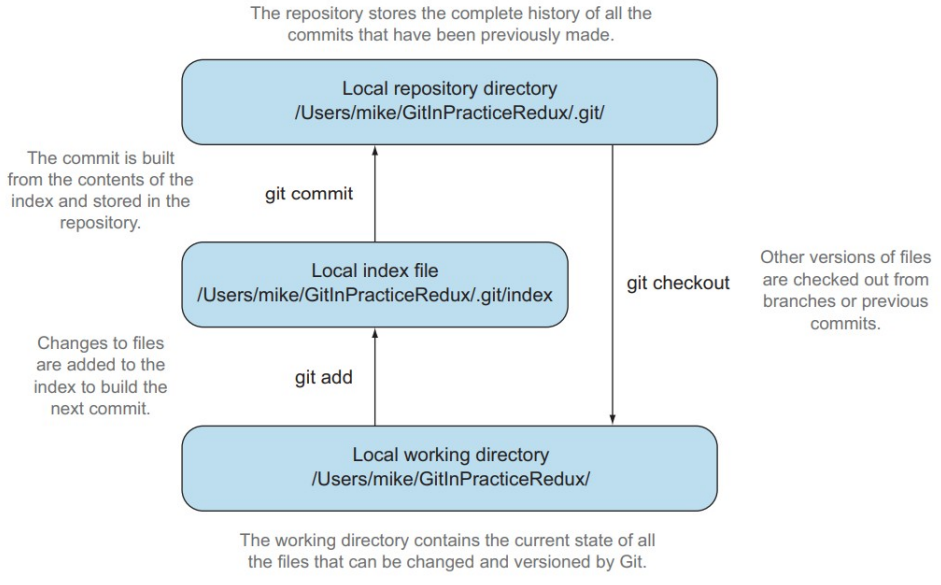


Figure 1.1 Git add/commit/checkout workflow

git add

+git ko add bất cứ thứ gì vào index stagging area mà ko có sự ra lệnh của bạn, vì vậy, việc đầu tiên bạn cần làm để git theo dõi file của bạn chính là git add.
+Có thể dùng để add cả directory
+Nếu sau khi add và commit file, ta có thay đổi thì cần git add lại file đó để chuẩn bị cho lần commit tiếp theo.

git status

+Nếu working tree ko clean thì trả về nội dung:
    --Dòng đầu chứa tên current branch
    --Initial commit (nếu file đó chưa được commit)
    --Changes to be commit

git commit

+Mỗi commit chứa nhiều thông tin như tác giả, point đến parent commit, commit content, commit id dưới dạng hash, ngày giờ commit.
+Sử dụng lệnh diff để so sánh
+Với những file commit lần đầu tiên thì sẽ xuất hiện tên branch, “root commit” và SHA-1, còn lại, thì là tên branch và shorted SHA-1.
+git --all: commit all, nhưng ko nên dùng vì khó theo dõi sau này

SHA-1 hash

+Là hash digest function được dùng nhiều trong git. Nó trả về 160-bit (20 bytes) hash value, thường được hiển thị dưới dạng 40-character hexadecimal string. Giá trị này thường dùng như ID để phân biệt các commit
+Giá trị ID này ko tăng dần như Subversion control do bản chất distributed của Git. Vì khi commit locally, ta ko biết được commit này nằm trước hay sau một commit khác. 40 character là quá nhiều nên Git cũng có thể dùng short version gồm 6 giá trị miễn là nó unique trong repo. Nếu Git accept long version thì sẽ accept short version, bất cứ nới nào (local hay web) miễn là short version vẫn còn unique trong repo đó.

Object store

+Git creates and stores a collection of object when you commit

+3 loại Git objects ta thường quan tâm là commits, blobs, trees
+Khi sử dụng Git, ta thường ko tương tác với objects và object files và cũng ko cần quan tâm quá nhiều đến tree và blobs với beginer users.

Parent commits

+Mỗi commit trừ cái đầu tiên sẽ point đến parent commit theo linear, branch history

Viewing history: git log, gitk, gitx

+history trong Git bao gồm tất cả các commit được tạo ra từ lúc tạo repo
+history chứa thông tin gồm: branches, merges, tags được tạo ra trong repo
+git log: liệt kê commit history trong current branch, theo thứ tự sớm nhất trước. Để thoát khỏi lệnh git log: q
+Ta sử dụng history để: nhắc cho nhớ lại công việc, xem lại vì sao các thay đổi được tạo ra, xem các thay đổi mới được tạo ra bởi người khác
+history sẽ trở nên hữu ích hơn nếu chất lượng data trong nó cao
+Nếu các commit được mô tả kỹ càng thì xem lại history sẽ rất dễ hiểu
+Gitk (linux/windows) hoặc gitx (mac) là 1 tool để visualize git history

Better commit rules

+Small commits are better --> chỉ chọn commit một số nhỏ file, chừa các file còn lại cho lần commit tiếp theo. Với cách làm này, ta dễ mô tả commit hơn --> sau này ta dễ học history hơn
+Avoid unrelated changes clumped together
+Commit messages: nên format như 1 email gồm subject and body. Subject ko nên dài quá 50 character, body ko nên quá 72 character, cách biệt với subject bằng 1 dòng trống.

Rewriting history

+Cho phép thiết lập để gộp nhiều commit thành 1 commit trong history (squashing work).
+Chỉ được thực hiện khi bạn đang làm trên 1 separated branch, ko có ai khác nữa

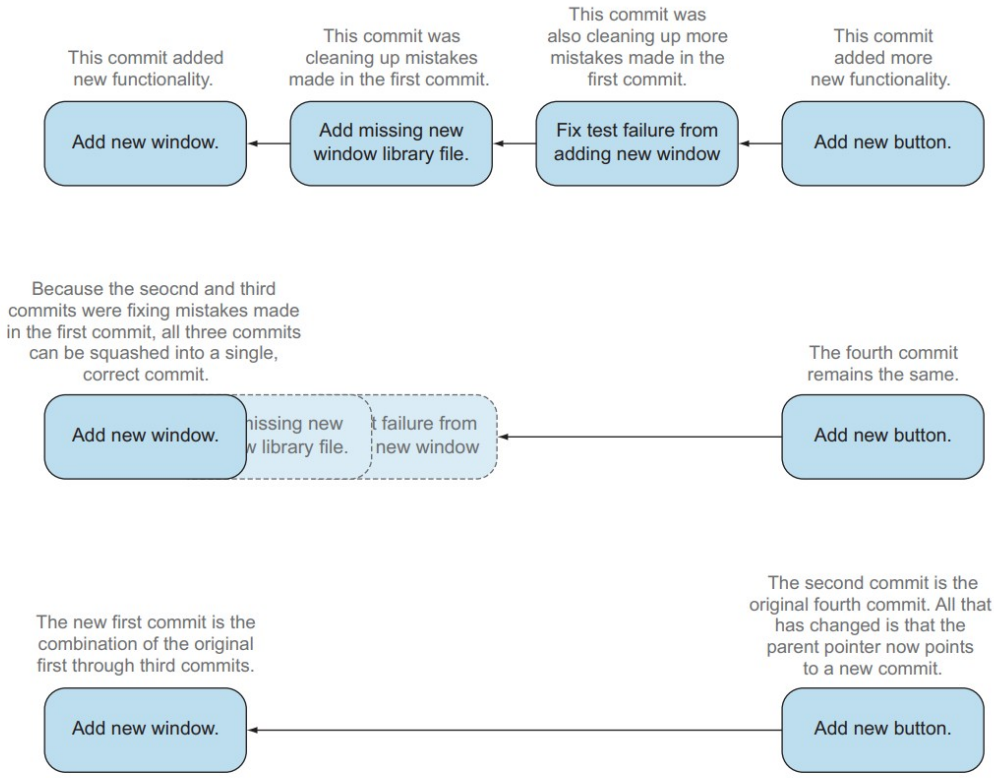


Figure 1.9 **Squashing multiple commits into a single commit**

Viewing the differences between two commits

+git diff commit1 commit2
+hoặc git diff commit1 commit2 interested.file

word diff

git diff -stat

refs

+Được dùng để xác định các commit, và làm tham số trong lệnh git diff
+Top commit id = master
+master^ = master~1: one commit before master commit
+master^^ = master~2
+HEAD: point to the top of the current branch. Nếu current branch là master, thì HEAD = master

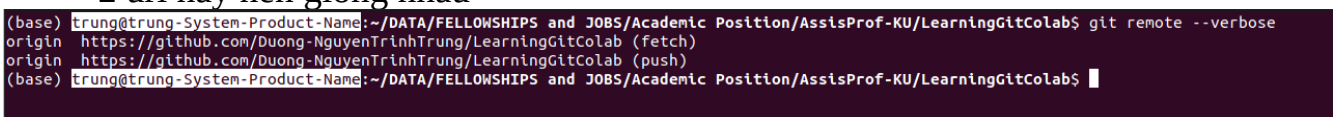
Remote Git

Adding a remote repo

+Git: distributed version control system
+Mỗi user có 1 complete repo trên máy local
+Mặc dù có centralized repo mà ở đó các member có thể commit, nhưng nó chỉ được access theo yêu cầu cụ thể
+git push: send commits to remote repo
+git fetch/pull: retrieves commits from remote repo
+Please sign up for a remote repo on GitHub
+git remote add origin fetch_url

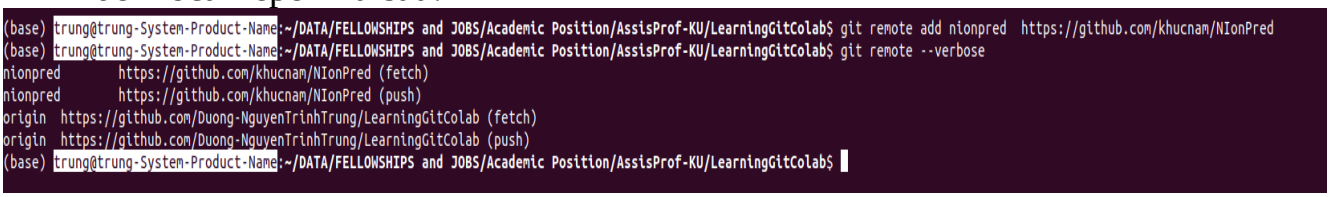
Kiểm tra lại git remote push url và fetch url

+git remote --verbose
2 url này nên giống nhau



Default name for a remote

+Là origin nếu local repo chỉ connect đến 1 remote repo
+Hiểu kỹ hơn thì origin point đến remote repo
+Cần chú ý là 1 local repo có thể connect đến nhiều remote repo. Giả sử ta connect thêm 1 remote repo nữa, ta có thể xem lại các remote repo được connect đến local repo như sau:



Authorative version storage

+git ko cho local repo này tốt hơn local repo khác
+By convention: centralized repo được xem là authorative version và người ta được khuyến khích là push commit và branch lên đó
+Điều này nghe có vẻ như ko đúng nhưng nó có sức mạnh. Linux là 1 ví dụ

Pushing changes to remote repo

+git push
+git đảm bảo rằng chỉ có những thay đổi giữa các repo được gửi lên remote repo
+Ví dụ: git push origin master
+--set-upstream option: ban muốn local master branch mà bạn mới push lên remote repo sẽ track the origin remote's branch master (remote origin master branch = tracking branch = upstream của local master branch).
+Flag của --set-upstream là -u
+Flag -u này chỉ cần cho lần đầu tiên bạn push để tạo remote branch. Nếu lệnh git push mà ko có tham số, thì git sẽ mặc định đó là git push origin master
(Chú ý: origin là tên của remote repo đầu tiên hoặc duy nhất mà local repo connect đến).
+Chú ý: với lệnh git clone thì mặc định là git clone origin master
+Chú ý: tracking branch = remote origin master là default push hoặc fetch location for a branch

Cloning a remote repo

+git clone (Subversion gọi đó là check out)
+Cloning = making a complete copy of a remote repo (all commits, branches, tags, complete history) onto your local repo

Remove a local repo and replace it from a cloned one

+Moving to desired folder
+rm -rf repo_name
+git clone url

Pulling changes from another repo

+git pull: thực hiện các việc sau
--download commit từ các repo khác đã được push lên centralized remote repo
--merge remote branch vào current branch

Why did a merge happen in git pull

+Để thống nhất những thay đổi giữa local và remote branch --> cần thêm ví dụ, chưa hiểu lắm

git fetch va git pull

+Cần xem lại, tìm ví dụ cho đến khi hiểu

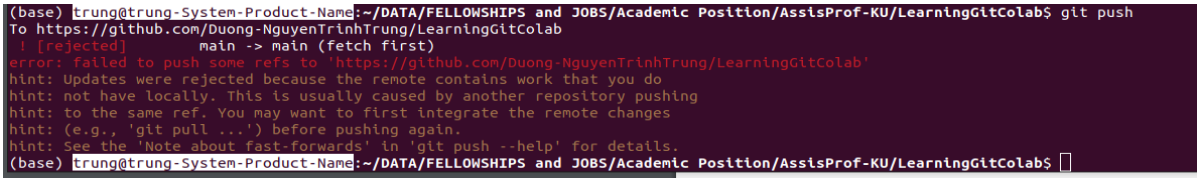
+Đã thử thực hành, git fetch cập nhật commit mà ko cập nhật file mới --> ko hiểu

Token mới nhất

(07-Mar-2022)
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

Lỗi conflict dẫn đến ko git push được

-Giả sử bạn đã clone từ repoX và chỉnh sửa file abc.txt, add, commit và push lên repo --> thành công
-Đồng thời, 1 người khác cũng clone từ repoX, và chỉnh sửa file abc.txt đó, add, commit và push --> không thành công



-Lúc này nếu git log sẽ thấy commit cuối cùng là (HEAD, origin/main)
-Một cách giải quyết (theo đề nghị từ git, thường là ko đúng ý mình) là dùng lệnh
    git rebase –skip --> dẫn đến commit cuối là (HEAD -> main, origin/main)
    git pull --> commit cuối là (HEAD -> main, origin/main)
--> dẫn đến là người thứ 2 chấp nhận chỉnh sửa mới nhất người thứ nhất đã update trên remote, lấy file bị conflict đó về local của mình rồi có thể chỉnh sửa tiếp.
-Cuối cùng, cần thực hiện lệnh git rebase --continue

git branch

-Một branch có thể hiểu là pointer trỏ đến 1 new commit khi commit đó được tạo ra trên branch đó
-Nói một cách khác là nó luôn trỏ đến commit cuối cùng được tạo ra trên branch của nó
-Khi nào nên dùng branch: ngay cả khi ko làm việc trong 1 team vẫn nên dùng branch, ví dụ khi bắt đầu fix bug. Khi bắt đầu tạo 1 feature mới, ta cũng có thể dùng branch
-Tạo 1 branch mới: git branch tên_branch
-Xem mình đang ở branch nào: git branch
--> sẽ hiện ra danh sách các branch, và branch có dấu * phía trước là branch hiện hành

-Chuyển sang 1 branch khác: git checkout tên_branch
-Chú ý: git branch branch_name = git branch branch_name current_branch

Cách đặt tên branch

-Tên branch ko được chứa space, ko được chứa ..
-Tên branch nên gợi nhớ ý nghĩa của branch

tag

-Tag tương tự như branch nhưng nó chỉ trỏ đến 1 single commit và chỉ trỏ đến duy nhất 1 commit mà thôi.
-Tag thường được dùng cho annotating commit, ví dụ tag trỏ đến commit được dùng để build release version 1.0 và tag có tên 1.0
-Sau này ta có thể quay lại, rebuild release đó mà ko sợ nó sẽ bị thay đổi tự động
-Cần thêm ví dụ --> tag này khá lợi hại

HEAD pointer

-Được cập nhật để trỏ đến current branch pointer và current branch pointer thì trỏ đến top commit của branch đó

Điều cần lưu ý trước khi checkout

-Cần phải commit tất cả thay đổi trong branch hiện tại trước khi check out một branch mới, nếu ko git sẽ từ chối
-Có thể sử dụng git checkout --force để overwrite uncommitted changeschanges

Pushing a local branch remotely