Table of Contents

[Setting up a repository 2](#__RefHeading___Toc28963_421221053)

[Useful link to learn 2](#__RefHeading___Toc68826_3670047285)

[Git trên local 2](#__RefHeading___Toc68818_3670047285)

[Git version 2](#__RefHeading___Toc68828_3670047285)

[git init 2](#__RefHeading___Toc28937_421221053)

[git config 2](#__RefHeading___Toc69981_3670047285)

[git clone 3](#__RefHeading___Toc69983_3670047285)

[Bare vs cloned repositories 3](#__RefHeading___Toc69989_3670047285)

[Full syntax of any command 4](#__RefHeading___Toc69923_3670047285)

[Saving changes 4](#__RefHeading___Toc69921_3670047285)

[Git ignore với .gitignore 4](#__RefHeading___Toc68907_3670047285)

[Things that can be done without network 6](#__RefHeading___Toc28939_421221053)

[Two ways to create a repo 6](#__RefHeading___Toc28941_421221053)

[Working directory vs repo 6](#__RefHeading___Toc69925_3670047285)

[Typical workflow 6](#__RefHeading___Toc28947_421221053)

[index stagging area 6](#__RefHeading___Toc68832_3670047285)

[git add 7](#__RefHeading___Toc28949_421221053)

[git checkout -- <filename> 7](#__RefHeading___Toc68834_3670047285)

[clear 7](#__RefHeading___Toc68836_3670047285)

[git status 7](#__RefHeading___Toc28951_421221053)

[git commit 7](#__RefHeading___Toc28953_421221053)

[Better commit rules 8](#__RefHeading___Toc70049_3670047285)

[Parent commits 9](#__RefHeading___Toc70051_3670047285)

[git log 9](#__RefHeading___Toc68838_3670047285)

[git log flags for searching through the repo history 9](#__RefHeading___Toc68903_3670047285)

[git branch 10](#__RefHeading___Toc68905_3670047285)

[How to name a branch 10](#__RefHeading___Toc37681_4212210531)

[Git diff 10](#__RefHeading___Toc70053_3670047285)

[tag 10](#__RefHeading___Toc37683_4212210531)

[HEAD pointer 11](#__RefHeading___Toc68820_36700472851)

[Before checkout 11](#__RefHeading___Toc68822_36700472851)

[Pushing a local branch remotely 11](#__RefHeading___Toc69985_3670047285)

[SHA-1 hash 11](#__RefHeading___Toc28955_421221053)

[Object store 11](#__RefHeading___Toc28957_421221053)

[Viewing history: git log, gitk, gitx 12](#__RefHeading___Toc28961_421221053)

[Rewriting history 12](#__RefHeading___Toc29482_421221053)

[Viewing the differences between two commits 12](#__RefHeading___Toc29484_421221053)

[word diff 13](#__RefHeading___Toc29486_421221053)

[git diff -stat 13](#__RefHeading___Toc29488_421221053)

[refs 13](#__RefHeading___Toc29490_421221053)

[Remote Git 13](#__RefHeading___Toc29492_421221053)

[Adding a remote repo 13](#__RefHeading___Toc29494_421221053)

[Kiểm tra lại git remote push url và fetch url 13](#__RefHeading___Toc29496_421221053)

[Default name for a remote 13](#__RefHeading___Toc29498_421221053)

[Authorative version storage 14](#__RefHeading___Toc29500_421221053)

[Pushing changes to remote repo 14](#__RefHeading___Toc29502_421221053)

[Cloning a remote repo 14](#__RefHeading___Toc29504_421221053)

[Remove a local repo and replace it from a cloned one 14](#__RefHeading___Toc29506_421221053)

[Pulling changes from another repo 14](#__RefHeading___Toc29508_421221053)

[Why did a merge happen in git pull 14](#__RefHeading___Toc29510_421221053)

[git fetch va git pull 14](#__RefHeading___Toc29512_421221053)

[Token mới nhất 15](#__RefHeading___Toc37453_421221053)

[Lỗi conflict dẫn đến ko git push được 15](#__RefHeading___Toc37455_421221053)

[git branch 15](#__RefHeading___Toc37679_421221053)

[Cách đặt tên branch 15](#__RefHeading___Toc37681_421221053)

[tag 15](#__RefHeading___Toc37683_421221053)

[HEAD pointer 16](#__RefHeading___Toc68820_3670047285)

[Điều cần lưu ý trước khi checkout 16](#__RefHeading___Toc68822_3670047285)

# Setting up a repository

## Useful link to learn

+Visual git cheatsheet: [https://ndpsoftware.com/git-cheatsheet.html#loc=index](https://ndpsoftware.com/git-cheatsheet.html" \l "loc=index);

+Visualizing git: <http://git-school.github.io/visualizing-git>

+Link1: <https://www.atlassian.com/git/tutorials>

## Git trên local

-Windows: Git Bash or Git GUI, download from https://gitforwindows.org

-Linux/Mac: terminal

## Git version

-Command: git --version

2.30.2

## git init

Tạo 1 git repo trên máy local

Một cách hiểu khác là track folder mình đang đứng --> lưu lại những thay đổi trên folder đó.

Sau khi thực hiện lệnh git init, folder ẩn .git sẽ được tạo ra trên thư mục này. Thư mục .git chứa meta data của git local repo. Không nên thay đổi nội dung trong thư mục này.

Note: chú ý local repo là thư mục có track bằng git trên máy local, để phân biệt với web repo trên github.

## git config

Git sử dụng name và email address: để xác định tác giả của commit

Lệnh git config để thiết lập username và email, ở các level là system, global (cho tất cả các commit) và local (cho repo hiện tại - đây là default)

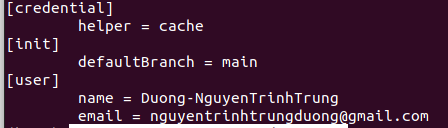
git config --global user.name <name>

git config --local user.email <email>

Setting trong global scope:

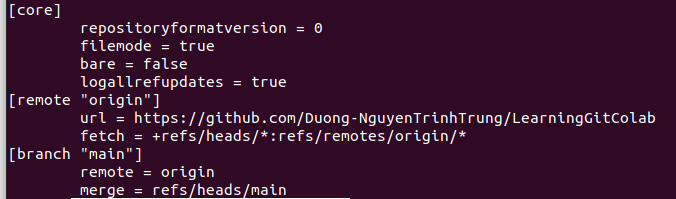
cat /home/trung/.gitconfig

(Trên Windows: C:\Users\<username>\.gitconfig)



Setting trong local scope:

cat .git/config

****

Xem tất cả các setting và scope

git config --list --show-origin –show-scope

Chỗ này ko hiểu sao thay đổi user.name trong local scope lại ko ảnh hưởng đến global scope theo quy tắc Local > Global > System. (Nhờ Khánh tìm hiểu thêm: https://www.atlassian.com/git/tutorials/setting-up-a-repository)

## **git clone**

Copy 1 repository về máy của mình

git clone <repo url>

## Bare vs cloned repositories

Cloned: git clone

+Dùng khi cần copy từ 1 repo

+Khi dùng git push, thì sẽ được hiểu là những thay đổi này sẽ được đưa lên repo đó

Bare: git init

+Được tạo từ máy local

+No có remote repo nào để push lên

+Thường người ta dùng lệnh này khi muốn tạo repo rỗng trên 1 hosted Git service như Bitbucket. Github thường dùng cho public repo trong khi Bitbucket thường dùng cho private repo, phù hợp với nhu cầu của doanh nghiệp

## Full syntax of any command

git init --help

# Saving changes

## Git ignore với .gitignore

+Git sẽ phân tất cả các file trong working directory thành 3 loại: tracked file, untracked file, và ignored file

+Ignored file thường là build artifact hoặc machine generated files và thường ko cần commit.

Ví dụ những file thông thường:

--dependency cache: nhưng nội dung của /node\_modules hoặc /packages

--compiled code: .o, .pyc, .class

--build output directories: /bin, /out, …

--files generated at runtime: .log, .lock, .tmp

--hidden system files: .DS\_Store, Thumb.db

--personal IDE config files: workspace.xml

+Những file này được track trong 1 file đặc biệt: /.gitignore trong thư mục root. Ko có lệnh git ignore trong Git mà ta cần phải tự edit và commit file này. Ngoài ra, khi tạo mới 1 repo trên github, ta có thể chọn option để tạo kèm file .gitignore, cùng với các ignored patterns phù hợp với ngôn ngữ lập trình mà mình làm.

+Git ignore patterns: <https://www.atlassian.com/git/tutorials/saving-changes/gitignore>

|  |  |
| --- | --- |
| Pattern | Example matches |
| \*\*/logs | logs/debug.log  logs/monday/foo.bar  build/logs/debug.log |
| \*\*/logs/debug.log | logs/debug.log  build/logs/debug.log  but not  logs/build/debug.log |
| \*.log | debug.log  foo.log  .log  logs/debug.log |
| \*.log  !important.log | debug.log  trace.log  but not  important.log  logs/important.log |
| /debug.log | debug.log but not logs/debug.log |

+Shared .gitignore file

.gitignore file tốt nhất nên thường được đặt trong top level directory của repo. Lúc này, nó được versioned và share với các collaborator.

+Personal Git ignore rules

--Với local repo, ta có thể định nghĩa các rule trong file .git/info/exclude và nó chỉ áp dụng với local repo đó thôi.

--Nếu manually edit những file này rồi gọi lệnh git status thì thấy thay đổi mới này ko cần được add/commit.

--Tuy nhiên, vẫn có thể force commit .gitignore file

$git add -f debug.log

$git commit -m “Force adding debug.log”

--Một tình huống khác, giả sử mình muốn bỏ qua tất cả các log file (\*.log) trừ file debug.log. Lúc đó, cần thực hiện những việc sau:

$echo \*.log > .gitignore

$echo !debug.log >> .gitignore

$git add debug.log

$git commit -m “Adding debug.log”

+Debugging .gitignore file

--Nếu có nhiều ignored patterns nằm rải rác ở nhiều .gitignore file --> ta ko biết được vì sao một hay nhiều file nào đó bị bỏ qua, ko track --> ta có thể check với lệnh:

git check-ignore -v debug.log init.log

Output, ví dụ, là:

.gitignore:3:\*.log debug.log init.log

Trong output: .gitignore là file có chứa pattern làm cho các file bị untrack, tiếp theo là line number, tiếp theo là pattern (một phần nội dung trong .gitignore) và cuối cùng là những file ta muốn biết bị sao ko được track.

## Things that can be done without network

+making committ

+viewing history

+requesting different between commits

## Two ways to create a 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

Note: cần git init trên 1 clean folder

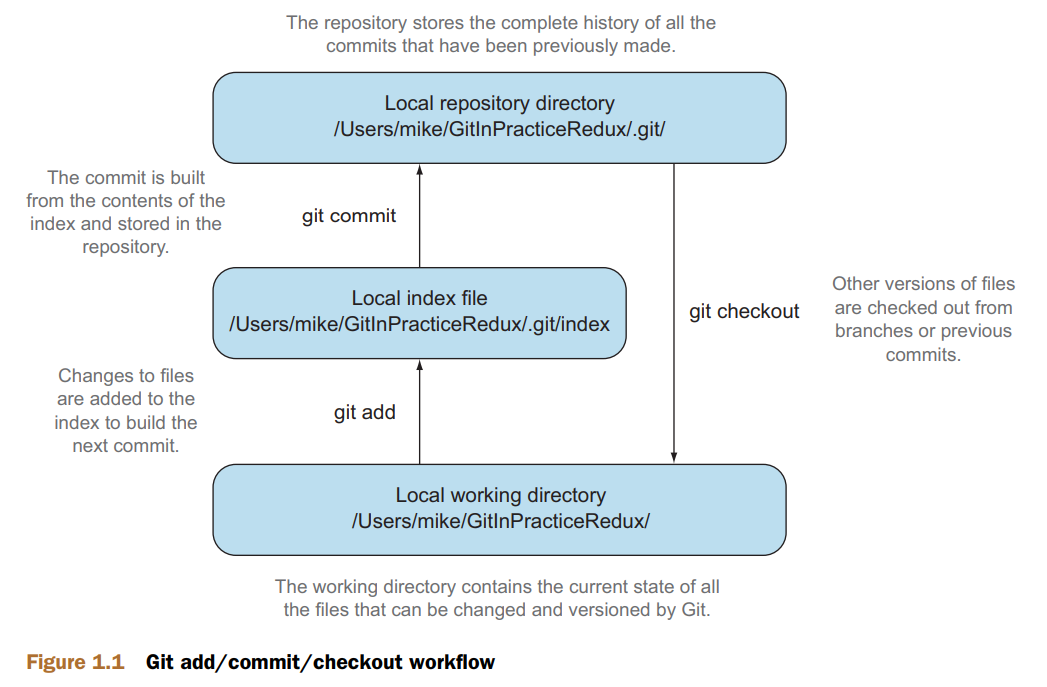
## Working directory vs repo

+Working directory: là thư mục tại đó mình đang tạo ra các thay đổi, nên hiểu là local repo

+Repo: nên hiểu là web repo

## Typical workflow

add/commit/checkout



## index stagging area

+Trong git, có 3 nơi có thể chứa những thay đổi là working directory, staging area, và repo.

+Những thay đổi mình tạo ra trong working directory vẫn chỉ nằm đó cho đến khi mình add nó lên staging area. Có thể hiểu staging area là nơi để preview những gì mà mình sẽ commit.

+Khi lệnh commit được thực hiện, Git sẽ lấy những thay đổi đang có trong staging are và làm thành những commit mới. Ở staging area, mình có thể add hoặc remove changes. Git ko sử dụng thư mục để làm staging area mà dùng index file chứa các blobs (file changes).

## 

## 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 checkout -- <filename>

+Dùng khi thay đổi trên file đã được đưa vào staging area mà ta unregister những thay đổi này

+Kiểm tra lại bằng git status

## clear

+Xóa màn hình cho dễ nhìn

+Vẫn có thể roll up để xem những lệnh trước đó

## 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 commit --all: commit all, nhưng ko nên dùng vì khó theo dõi sau này

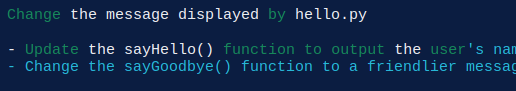
+git commit –amend: cho phép thay đổi commit gần nhất

## 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.



## Parent commits

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

## git log

+Liệt kê tất cả các commits mình đã thực hiện. Commit cuối cùng nằm đầu tiên. Nhấn q để thoát.

+Cấu trúc của 1 git log: bao gồm:

--commit SHA

--date

--author email

--username

--commit message

+10 helpful flags to use with git log

<https://dev.to/ansdb/10-helpful-flags-to-use-with-git-log-command-1l3k>

## git log flags for searching through the repo history

+Kỹ năng này giúp rất nhiều trong tracking your changes and your teammate’s changes

+Xem mỗi commit trên 1 dòng:

--git log –oneline

--Thông tin rút gọn lại còn: SHA chỉ có 7 kí tự, commit message

+Xem tất cả commit, kể cả ở các branch khác với current branch mà chưa được merge vào

--git log --all --oneline

+Chỉ xem những merge commit:

--git log --oneline --merge

+Chỉ xem những no-merge commit:

--git log --oneline --merge

+Liệt kê commit cùng với những thay đổi

--git log --patch

Tương đương với git log -p

Dấu +: thể hiện những dòng code được thêm vào, dấu -: ngược lại

+Liệt kê commit cùng với thống kê

--git log --stat

Thống kê có bao nhiêu file thay đổi, bao nhiêu insertion, deletion

+Liệt kê những commit có liên quan đến file có chứa đoạn text cho trước

--git log --oneline -S”your\_text\_here”

+Liệt kê những commit mà commit message có chứa đoạn text cho trước

--git log --grep=”your\_text\_here”

+Liệt kê những commit được thực hiện bởi 1 hay nhiều author nào đó

--git log --author=”username or email”

--git log --author=”username1\username2”

+Liệt kê n commit gần nhất

--git log -n

+Liệt kê những commit mà track những thay đổi trong một hay nhiều file cụ thể

--git log --oneline -- filename1 filename2 filename3

## 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

## How to name a branch

-Tên branch ko được chứa space, ko được chứa ..

-Tên branch nên gợi nhớ ý nghĩa của branch

## Git diff

-Git diff dùng để track sự khác biệt giữa 2 input (file, commit, branch, …)

-Git diff thường được dùn với git status và git log để kiểm tra tình hình của repo

-Lấy ví dụ từ: <https://www.atlassian.com/git/tutorials/saving-changes/git-diff>

## 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

## Before 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 changes changes

## Pushing a local branch remotely

-

## 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.

## 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

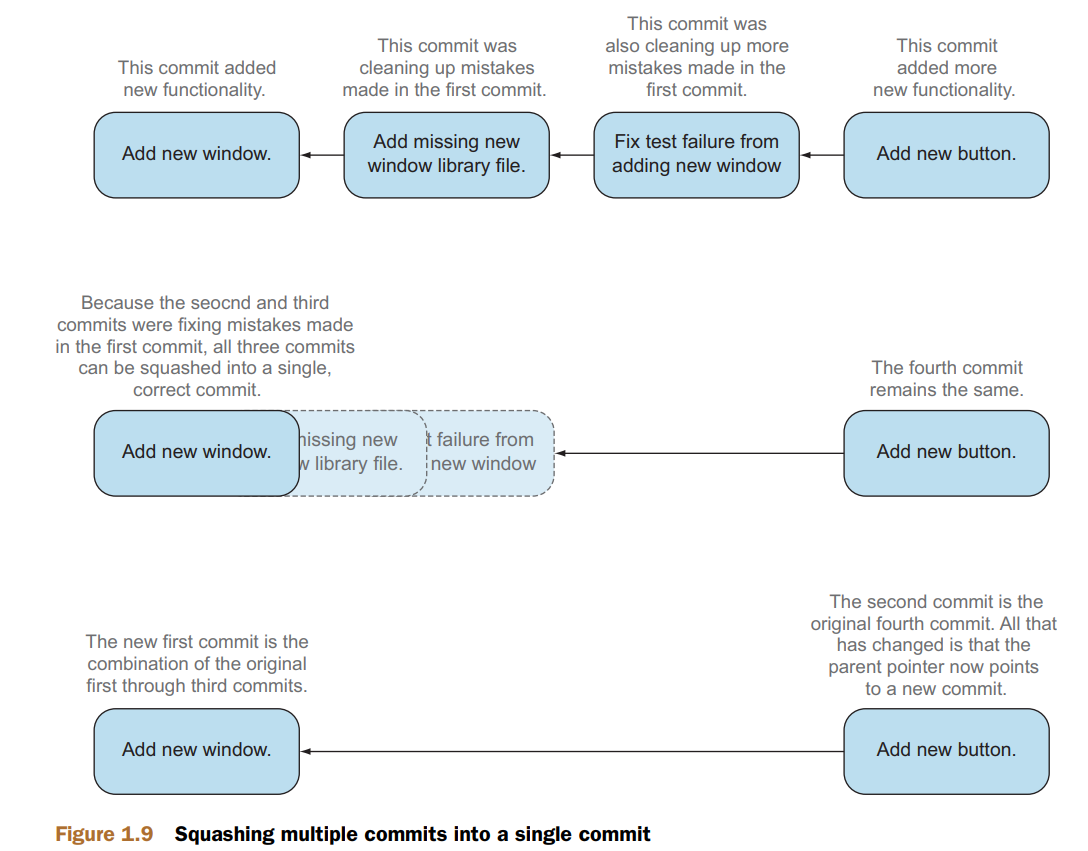
+Nếu các commit được mô tả kỹ càng thì xem lại history sẽ rất dễ hiểu

+Gitk (linux/windows) hoặc gitx (mac) là 1 tool để visualize git history

## 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



## 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

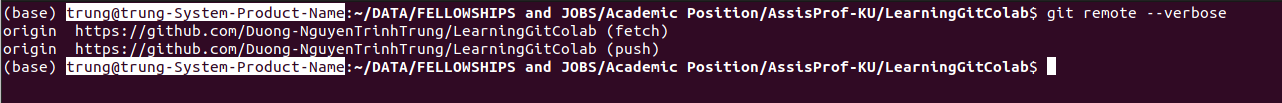
+Please sign up for a remote repo on GitHub

+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

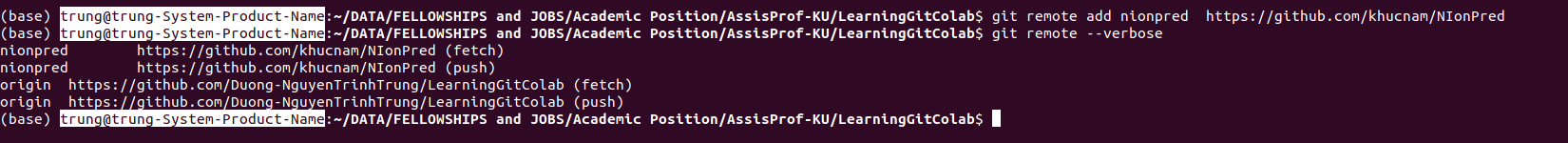


## Default name for a remote

+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

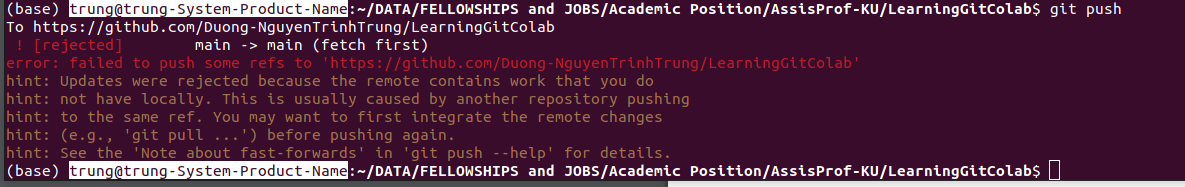
(07-Mar-2022)

ghp\_zgTZ9MUKPL9BQW8w5lPDX0uP3EbV2I37a9FZ

## 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