**DÙNG GITHUB:**

Phần 1: Cài đặt và tạo liên kết chứng thực GITHUB local và GITHUB online

* Địa chỉ tải: <https://git-scm.com/downloads>
* Graphical user interface, application, website

  Description automatically generated
* Cài đặt:
  + Graphical user interface, text, application, email

    Description automatically generated
  + Chọn NEXT
  + Graphical user interface, text, application, email

    Description automatically generated
  + Chọn NEXT tiếp theo
  + Graphical user interface, text, application, email

    Description automatically generated
  + Chọn NEXT
  + Graphical user interface, text, application, email

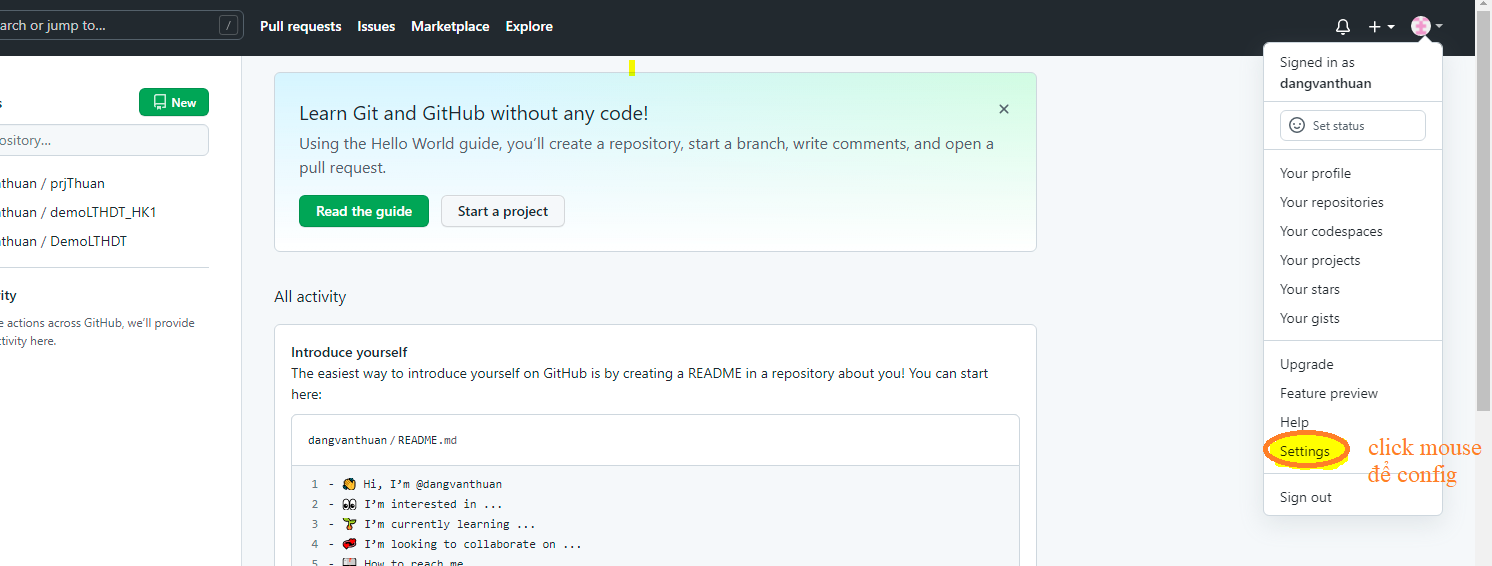
    Description automatically generated
  + OK thành công
* Tạo tài khoản GITHUB online:
  + Graphical user interface, website

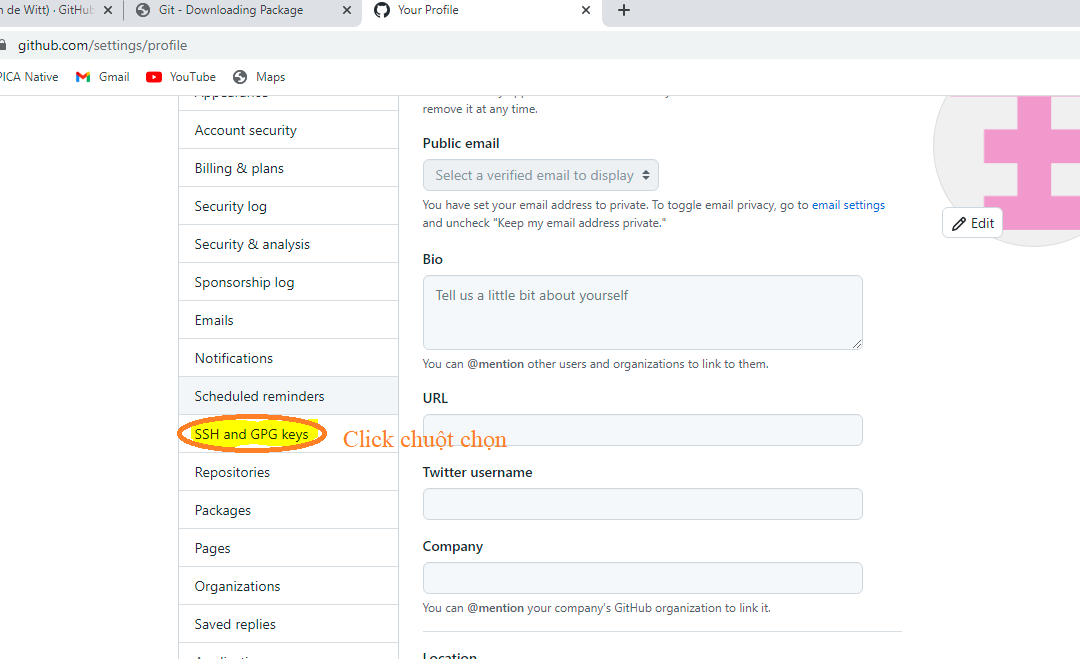
    Description automatically generated
  + Click sign up for GitHub
  + Graphical user interface, application

    Description automatically generated
  + Click chọn continue

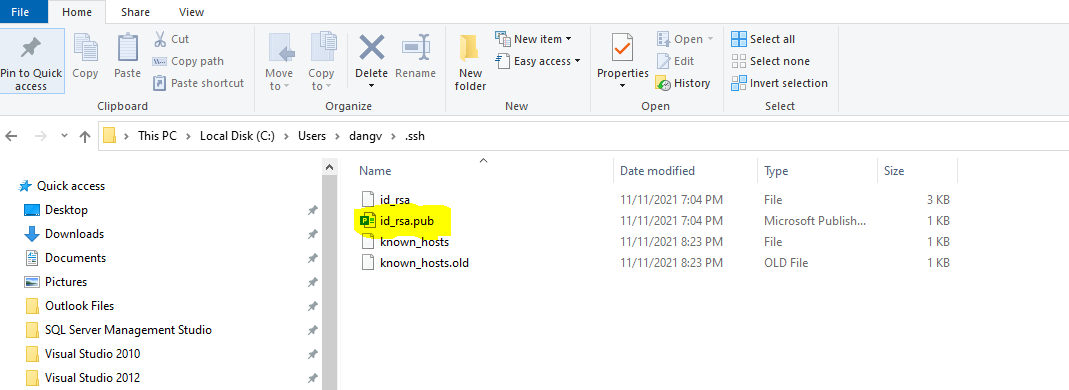
Graphical user interface, text, application

Description automatically generated



* Chọn theo mục:
* 
  + Graphical user interface, text, application

    Description automatically generated
  + HÌNH ####
  + Setting: copy SSH từ máy tính của bạn vào:
    - Mở my computer:



* **Nếu máy tính đang ẩn file phần mở rộng ta cấu hình view option folder:**
  + Graphical user interface, text, application

    Description automatically generated
* **sao cho thấy được đuôi flie:**
  + **Id\_rsa.pub**
  + **Mở file này với wordpad lên và chép phần nộp dung dáng vào** HÌNH ####

**Thực hiện liên kết xong giữa máy tính và tài khoản online của github**

* Phần 2: Tạo thư mục chứa project file code
  + Kích chuột phải chuột vào thư mục vừa tạo chọn:
* Text

  Description automatically generated

**Step 1: Tạo Repository để lưu code offline**

- CMD: git init

**step 2: xem trạng thái Repository**

- CMD: git status

**step 3: Tạo nhánh mới có tên develop**

- CMD: git checkout -b develop

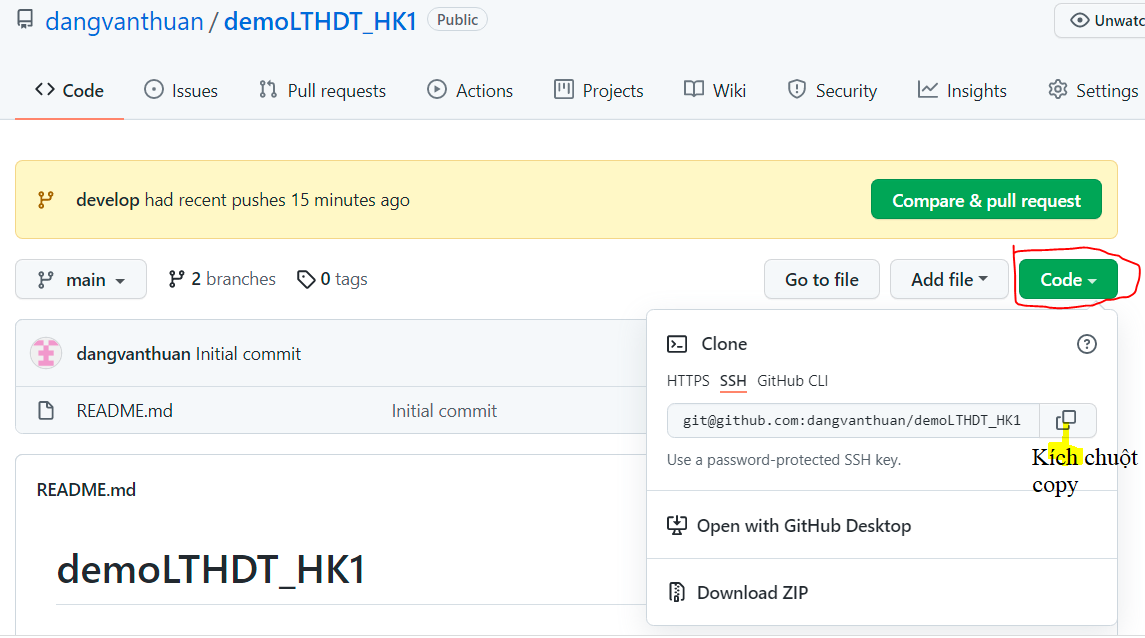
**step 4: Đưa file vào môi trường github:**

- CMD: git add <tên file> // Đưa một file vào môi trường github

- CMD: git add . // Đưa tất cả các file vào môi trường github

- CMD: git commit -m “lần xác nhận” // xác nhận file vào môi trường github nhớ close các file

**step 5: Ðưa lên online github:**



- CMD: git remote add dautien [git@github.com:dangvanthuan/demoLTHDT\_HK1.git//](mailto:git@github.com:dangvanthuan/demoLTHDT_HK1.git//) chép code liên kết:

**step 6: xem kết quả:**

- CMD: git remote -v

$ git remote -v

dautien git@github.com:dangvanthuan/demoLTHDT\_HK1.git (fetch)

dautien git@github.com:dangvanthuan/demoLTHDT\_HK1.git (push)

**kết thúc:**

$ git push dautien develop

The authenticity of host 'github.com (20.205.243.166)' can't be established.

RSA key fingerprint is SHA256:nThbg6kXUpJWGl7E1IGOCspRomTxdCARLviKw6E5SY8.

This key is not known by any other names

Are you sure you want to continue connecting (yes/no/[fingerprint])? y

Please type 'yes', 'no' or the fingerprint: yes

Warning: Permanently added 'github.com' (RSA) to the list of known hosts.

Enumerating objects: 4, done.

Counting objects: 100% (4/4), done.

Delta compression using up to 4 threads

Compressing objects: 100% (3/3), done.

Writing objects: 100% (4/4), 336 bytes | 112.00 KiB/s, done.

Total 4 (delta 0), reused 0 (delta 0), pack-reused 0

remote:

remote: Create a pull request for 'develop' on GitHub by visiting:

remote: https://github.com/dangvanthuan/demoLTHDT\_HK1/pull/new/develop

remote:

To github.com:dangvanthuan/demoLTHDT\_HK1.git

\* [new branch] develop -> develop

**step 7: Tạo nhánh cho repository:**

- CMD: git checkout -b <tên nhánh> hoặc nhánh con theo <nhanh1/nhanh2>

dangv@PAUL MINGW64 /d/GitProjectCode/code (develop)

$ git checkout -b develop2/develop3

Switched to a new branch 'develop2/develop3'

- CMD: git checkout <tên nhánh> // về nhánh mẹ

Step 8: xóa nhánh :

-CMD: git branch -d <tên nhánh>

Các cú pháp câu lệnh

dangv@PAUL MINGW64 /c/baitap (baitapv1)

$ git status -?

error: unknown switch `?'

usage: git status [<options>] [--] <pathspec>...

-v, --verbose be verbose

-s, --short show status concisely

-b, --branch show branch information

--show-stash show stash information

--ahead-behind compute full ahead/behind values

--porcelain[=<version>]

machine-readable output

--long show status in long format (default)

-z, --null terminate entries with NUL

-u, --untracked-files[=<mode>]

show untracked files, optional modes: all, normal, no. (Default: all)

--ignored[=<mode>] show ignored files, optional modes: traditional, matching, no. (Default: traditional)

--ignore-submodules[=<when>]

ignore changes to submodules, optional when: all, dirty, untracked. (Default: all)

--column[=<style>] list untracked files in columns

--no-renames do not detect renames

-M, --find-renames[=<n>]

detect renames, optionally set similarity index

--show-ignored-directory

(DEPRECATED: use --ignore=matching instead) Only show directories that match an ignore pattern name.

--no-lock-index (DEPRECATED: use `git --no-optional-locks status` instead) Do not lock the index

dangv@PAUL MINGW64 /c/baitap (baitapv1)

$ git init -?

error: unknown switch `?'

usage: git init [-q | --quiet] [--bare] [--template=<template-directory>] [--shared[=<permissions>]] [<directory>]

--template <template-directory>

directory from which templates will be used

--bare create a bare repository

--shared[=<permissions>]

specify that the git repository is to be shared amongst several users

-q, --quiet be quiet

--separate-git-dir <gitdir>

separate git dir from working tree

-b, --initial-branch <name>

override the name of the initial branch

--object-format <hash>

specify the hash algorithm to use

dangv@PAUL MINGW64 /c/baitap (baitapv1)

$ git checkout -?

error: unknown switch `?'

usage: git checkout [<options>] <branch>

or: git checkout [<options>] [<branch>] -- <file>...

-b <branch> create and checkout a new branch

-B <branch> create/reset and checkout a branch

-l create reflog for new branch

--guess second guess 'git checkout <no-such-branch>' (default)

--overlay use overlay mode (default)

-q, --quiet suppress progress reporting

--recurse-submodules[=<checkout>]

control recursive updating of submodules

--progress force progress reporting

-m, --merge perform a 3-way merge with the new branch

--conflict <style> conflict style (merge or diff3)

-d, --detach detach HEAD at named commit

-t, --track set upstream info for new branch

-f, --force force checkout (throw away local modifications)

--orphan <new-branch>

new unparented branch

--overwrite-ignore update ignored files (default)

--ignore-other-worktrees

do not check if another worktree is holding the given ref

-2, --ours checkout our version for unmerged files

-3, --theirs checkout their version for unmerged files

-p, --patch select hunks interactively

--ignore-skip-worktree-bits

do not limit pathspecs to sparse entries only

--pathspec-from-file <file>

read pathspec from file

--pathspec-file-nul with --pathspec-from-file, pathspec elements are separated with NUL character

dangv@PAUL MINGW64 /c/baitap (baitapv1)

$ git branch -?

error: unknown switch `?'

usage: git branch [<options>] [-r | -a] [--merged] [--no-merged]

or: git branch [<options>] [-l] [-f] <branch-name> [<start-point>]

or: git branch [<options>] [-r] (-d | -D) <branch-name>...

or: git branch [<options>] (-m | -M) [<old-branch>] <new-branch>

or: git branch [<options>] (-c | -C) [<old-branch>] <new-branch>

or: git branch [<options>] [-r | -a] [--points-at]

or: git branch [<options>] [-r | -a] [--format]

Generic options

-v, --verbose show hash and subject, give twice for upstream branch

-q, --quiet suppress informational messages

-t, --track set up tracking mode (see git-pull(1))

-u, --set-upstream-to <upstream>

change the upstream info

--unset-upstream unset the upstream info

--color[=<when>] use colored output

-r, --remotes act on remote-tracking branches

--contains <commit> print only branches that contain the commit

--no-contains <commit>

print only branches that don't contain the commit

--abbrev[=<n>] use <n> digits to display object names

Specific git-branch actions:

-a, --all list both remote-tracking and local branches

-d, --delete delete fully merged branch

-D delete branch (even if not merged)

-m, --move move/rename a branch and its reflog

-M move/rename a branch, even if target exists

-c, --copy copy a branch and its reflog

-C copy a branch, even if target exists

-l, --list list branch names

--show-current show current branch name

--create-reflog create the branch's reflog

--edit-description edit the description for the branch

-f, --force force creation, move/rename, deletion

--merged <commit> print only branches that are merged

--no-merged <commit> print only branches that are not merged

--column[=<style>] list branches in columns

--sort <key> field name to sort on

--points-at <object> print only branches of the object

-i, --ignore-case sorting and filtering are case insensitive

--format <format> format to use for the output

dangv@PAUL MINGW64 /c/baitap (baitapv1)

$ git add -?

error: unknown switch `?'

usage: git add [<options>] [--] <pathspec>...

-n, --dry-run dry run

-v, --verbose be verbose

-i, --interactive interactive picking

-p, --patch select hunks interactively

-e, --edit edit current diff and apply

-f, --force allow adding otherwise ignored files

-u, --update update tracked files

--renormalize renormalize EOL of tracked files (implies -u)

-N, --intent-to-add record only the fact that the path will be added later

-A, --all add changes from all tracked and untracked files

--ignore-removal ignore paths removed in the working tree (same as --no-all)

--refresh don't add, only refresh the index

--ignore-errors just skip files which cannot be added because of errors

--ignore-missing check if - even missing - files are ignored in dry run

--chmod (+|-)x override the executable bit of the listed files

--pathspec-from-file <file>

read pathspec from file

--pathspec-file-nul with --pathspec-from-file, pathspec elements are separated with NUL character

dangv@PAUL MINGW64 /c/baitap (baitapv1)

$ git commit -?

error: unknown switch `?'

usage: git commit [<options>] [--] <pathspec>...

-q, --quiet suppress summary after successful commit

-v, --verbose show diff in commit message template

Commit message options

-F, --file <file> read message from file

--author <author> override author for commit

--date <date> override date for commit

-m, --message <message>

commit message

-c, --reedit-message <commit>

reuse and edit message from specified commit

-C, --reuse-message <commit>

reuse message from specified commit

--fixup [(amend|reword):]commit

use autosquash formatted message to fixup or amend/reword specified commit

--squash <commit> use autosquash formatted message to squash specified commit

--reset-author the commit is authored by me now (used with -C/-c/--amend)

--trailer <trailer> add custom trailer(s)

-s, --signoff add a Signed-off-by trailer

-t, --template <file>

use specified template file

-e, --edit force edit of commit

--cleanup <mode> how to strip spaces and #comments from message

--status include status in commit message template

-S, --gpg-sign[=<key-id>]

GPG sign commit

Commit contents options

-a, --all commit all changed files

-i, --include add specified files to index for commit

--interactive interactively add files

-p, --patch interactively add changes

-o, --only commit only specified files

-n, --no-verify bypass pre-commit and commit-msg hooks

--dry-run show what would be committed

--short show status concisely

--branch show branch information

--ahead-behind compute full ahead/behind values

--porcelain machine-readable output

--long show status in long format (default)

-z, --null terminate entries with NUL

--amend amend previous commit

--no-post-rewrite bypass post-rewrite hook

-u, --untracked-files[=<mode>]

show untracked files, optional modes: all, normal, no. (Default: all)

--pathspec-from-file <file>

read pathspec from file

--pathspec-file-nul with --pathspec-from-file, pathspec elements are separated with NUL character

### 1. Branch

Các **Branch**(nhánh) đại diện cho các **phiên bản cụ thể** của một kho lưu trữ tách ra từ project chính của bạn.

Branch cho phép bạn theo dõi các thay đổi thử nghiệm bạn thực hiện đối với kho lưu trữ và có thể hoàn nguyên về các phiên bản cũ hơn.

### 2. Commit

Một commit đại diện cho một thời điểm cụ thể trong lịch sử dự án của bạn. Sử dụng lệnh commit kết hợp với lệnh **git add** để cho git biết những thay đổi bạn muốn lưu vào local repository.

### 3. Checkout

Sử dụng lệnh git checkout để chuyển giữa các branch. Chỉ cần nhập git checkout theo sau là tên của branch bạn muốn chuyển đến hoặc nhập git checkout master để trở về branch chính (master branch).

### 4. Fetch

Lệnh git fetch tìm nạp các bản sao và tải xuống tất cả các tệp branch vào máy tính của bạn. Sử dụng nó để lưu các thay đổi mới nhất vào kho lưu trữ của bạn. Nó có thể tìm nạp nhiều branch cùng một lúc.

### 5. Fork

Một fork là một bản sao của một kho lưu trữ (repository). Các lập trình viên thường tận dụng lợi ích của fork để thử nghiệm các thay đổi mà không ảnh hưởng đến dự án chính.

### 6. Head

Các commit ở đầu của một branch được gọi là head. Nó đại diện cho commit mới nhất của repository mà bạn hiện đang làm việc.

### 7. Index

Bất cứ khi nào bạn thêm, xóa hoặc thay đổi một file, nó vẫn nằm trong chỉ mục cho đến khi bạn sẵn sàng commit các thay đổi. Nó như là khu vực tổ chức (stagging area) cho Git. Sử dụng lệnh git status để xem nội dung của index của bạn.

*STAGGING LÀ MỘT BƯỚC TRƯỚC KHI COMMIT TRONG GIT.*

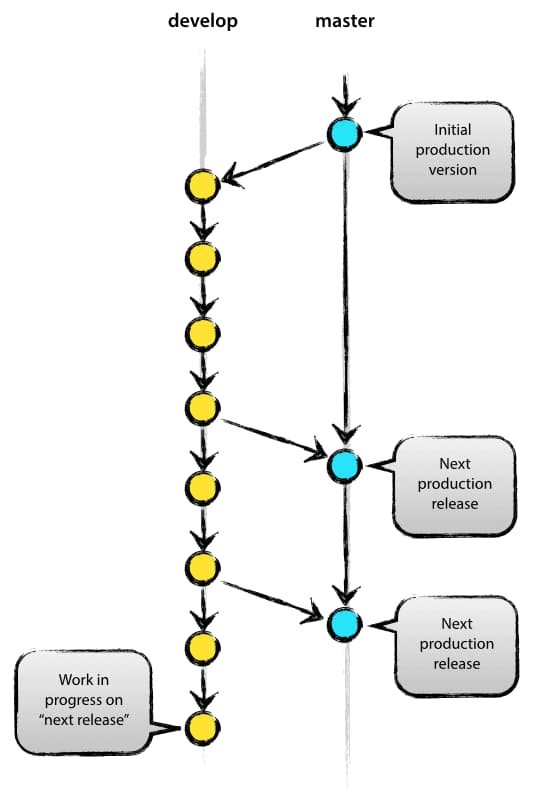
*MỘT COMMIT TRONG GIT ĐƯỢC THỰC HIỆN THEO HAI BƯỚC: STAGGING VÀ COMMIT THỰC TẾ. MIỄN LÀ NHỮNG THAY ĐỔI NẰM TRONG KHU VỰC TỔ CHỨC (STAGGING AREA), GIT CHO PHÉP BẠN CHỈNH SỬA NÓ THEO Ý MUỐN (THAY THẾ CÁC TỆP ĐƯỢC PHÂN ĐOẠN BẰNG CÁC PHIÊN BẢN KHÁC CỦA CÁC TỆP ĐƯỢC PHÂN LOẠI, LOẠI BỎ CÁC THAY ĐỔI KHỎI PHÂN ĐOẠN, V.V.)*

Xem tiếp...

Những thay đổi được tô sáng bằng màu xanh lá cây đã sẵn sàng để được commit trong khi những thay đổi màu đỏ thì chưa.

### 8. Master

Master là nhánh chính của tất cả các repository của bạn. Nó nên bao gồm những thay đổi và commit gần đây nhất.

Nhánh Master

### 9. Merge

Lệnh git merge kết hợp với các yêu cầu kéo (pull requests) để thêm các thay đổi từ nhánh này sang nhánh khác.

### 10. Origin

Ảnh có chứa văn bản, biểu đồ, bản đồ, hàng

Mô tả được tạo tự động

Origin là phiên bản mặc định của repository. Origin cũng đóng vai trò là bí danh hệ thống để liên lạc với nhánh chính.

Lệnh git push origin master để đẩy các thay đổi cục bộ đến nhánh chính.

### 11. Pull

Pull requests thể hiện các đề xuất thay đổi cho nhánh chính. Nếu bạn làm việc với một nhóm, bạn có thể tạo các pull request để yêu cầu người bảo trì kho lưu trữ xem xét các thay đổi và hợp nhất chúng.

Lệnh git pull được sử dụng để thêm các thay đổi vào nhánh chính.

### 12. Push

Lệnh git push được sử dụng để cập nhật các nhánh từ xa với những thay đổi mới nhất mà bạn đã commit.

### 13. Rebase

Lệnh git rebase cho phép bạn phân tách, di chuyển hoặc thoát khỏi các commit. Nó cũng có thể được sử dụng để kết hợp hai nhánh khác nhau.

### 14. Remote

Một Remote (kho lưu trữ từ xa) là một bản sao của một chi nhánh. Remote giao tiếp ngược dòng với nhánh gốc (origin branch) của chúng và các Remote khác trong kho lưu trữ.

### 15. Repository

Kho lưu trữ Git chứa tất cả các tệp dự án của bạn bao gồm các branch, tags và commit.

### 16. Stash

Lệnh git stash sẽ loại bỏ các thay đổi khỏi chỉ mục của bạn và xóa stashes chúng đi sau.

Nó có ích nếu bạn muốn tạm dừng những gì bạn đang làm và làm việc khác trong một khoảng thời gian. Bạn không thể đặt stash nhiều hơn một bộ thay đổi ở cùng một thời điểm.

### 17. Tags

Tags cung cấp cho bạn một cách để theo dõi các commit quan trọng. Các tags nhẹ chỉ đơn giản đóng vai trò là con trỏ trong khi các tags chú thích được lưu trữ dưới dạng các đối tượng đầy đủ.

### 19. Upstream

Trong ngữ cảnh của Git, upstream đề cập đến nơi bạn push các thay đổi của mình, thường là nhánh chính (master branch).

Xem [Git docs reference](https://git-scm.com/docs) để biết thêm chi tiết về thuật ngữ liên quan đến Git.

## Các lệnh git cơ bản

#### ****1) git config****

Tác dụng : Để set user name và email của bạn trong main configuration file.  
Cách xài : Để kiểm tra tên và kiểu email trong cấu hình dùng ***git config -- global user.name*** và ***git config -- global user.email***. Để set email hoặc tên mới ***git config -- global user.name = “Hải Nguyễn”*** và ***git config -- global user.email = “hainguyen@gmail.com”***

#### ****2) git init****

Tác dụng : Khởi tạo 1 git repository 1 project mới hoặc đã có.

Cách xài: ***git init*** trong thư mục gốc của dự án.

#### ****3) git clone****

Tác dụng: Copy 1 git repository từ remote source.

Cách xài: ***git clone <:clone git url:>***

#### ****4) git status****

Tác dụng: Để check trạng thái của những file bạn đã thay đổi trong thư mục làm việc. VD: Tất cả các thay đổi cuối cùng từ lần commit cuối cùng.

Cách xài: ***git status*** trong thư mục làm việc.

#### ****5) git add****

Tác dụng: Thêm thay đổi đến stage/index trong thư mục làm việc.

Cách xài: ***git add***

#### ****6) git commit****

Tác dụng: commit nghĩa là một action để Git lưu lại một snapshot của các sự thay đổi trong thư mục làm việc. Và các tập tin, thư mục được thay đổi đã phải nằm trong Staging Area. Mỗi lần commit nó sẽ được lưu lại lịch sử chỉnh sửa của code kèm theo tên và địa chỉ email của người commit. Ngoài ra trong Git bạn cũng có thể khôi phục lại tập tin trong lịch sử commit của nó để chia cho một branch khác, vì vậy bạn sẽ dễ dàng khôi phục lại các thay đổi trước đó.

Cách dùng: ***git commit -m ”Đây là message, bạn dùng để note những thay đổi để sau này dễ dò lại”***

**[5 tip về GitHub cho lập trình viên](https://topdev.vn/blog/5-tip-ve-github-cho-lap-trinh-vien/" \t "_blank)**

#### ****7) git push/git pull****

Tác dụng: Push hoặc Pull các thay đổi đến remote. Nếu bạn đã added và committed các thay đổi và bạn muốn đẩy nó lên hoặc remote của bạn đã update và bạn apply tất cả thay đổi đó trên code của mình.

Cách dùng: ***git pull <:remote:> <:branch:>*** and ***git push <:remote:> <:branch:>***

#### ****8) git branch****

Tác dụng: liệt kê tất cả các branch (nhánh).

Cách dùng: ***git branch*** hoặc ***git branch -a***

#### ****9) git checkout****

Tác dụng: Chuyển sang branch khác

Cách dùng: ***git checkout <: branch:>*** hoặc ***\*\* \_ git checkout -b <: branch:>*** nếu bạn muốn tạo và chuyển sang một chi nhánh mới.

#### ****10) git stash****

Tác dụng: Lưu thay đổi mà bạn không muốn commit ngay lập tức.

Cách dùng: ***git stash*** trong thư mục làm việc của bạn.

#### ****11) git merge****

Tác dụng: Merge 2 branch lại với nahu.

Cách dùng: Chuyển tới branch bạn muốn merge rồi  dùng ***git merge <:branch\_ban\_muon\_merge:>***

#### ****12) git reset****

Tác dụng: Bạn đã đưa một tập tin nào đó vào Staging Area nhưng bây giờ bạn muốn loại bỏ nó ra khỏi đây để không phải bị commit theo.

Cách dùng: ***git reset HEAD tên\_file***

#### ****13) git remote****

Tác dụng: Để check remote/source bạn có hoặc add thêm remote

Cách dùng: ***git remote*** để kiểm tra và liệt kê. Và ***git remote add <: remote\_url:>*** để thêm.

#### ****14) git add****

Tác dụng: Để đưa một tập tin vào Staging Area

Cách dùng: ***git add tên\_file*** hoặc muốn thêm hết file của thư mục thì ***git add all***

### Git Cheet Sheets

# Những lệnh Git cơ bản cần nhớ

### 1. Cơ bản vê Git

Git là tên gọi của một Hệ thống quản lý phiên bản phân tán (Distributed Version Control System – DVCS) là một trong những hệ thống quản lý phiên bản phân tán phổ biến nhất hiện nay. DVCS nghĩa là hệ thống giúp mỗi máy tính có thể lưu trữ nhiều phiên bản khác nhau của một mã nguồn được nhân bản (clone) từ một kho chứa mã nguồn (repository), mỗi thay đổi vào mã nguồn trên máy tính sẽ có thể ủy thác (commit) rồi đưa lên máy chủ nơi đặt kho chứa chính. Và một máy tính khác (nếu họ có quyền truy cập) cũng có thể clone lại mã nguồn từ kho chứa hoặc clone lại một tập hợp các thay đổi mới nhất trên máy tính kia. Trong Git, thư mục làm việc trên máy tính gọi là Working Tree.

### 2. Flow cơ bản khi sử dụng Git

Sau đây là flow cơ bản khi bạn sử dụng Git:

* Clone project từ server về Local Repository
* Check-out 1 nhánh từ Local Repository về Working Space
* Bạn sẽ làm việc (thêm, sửa, xoá tại Working Space)
* Add : xác nhận sự thay đổi của các files (đưa đến vùng Staging Area)
* Commit: cập nhật sự thay đổi lên Local Repository

Về cơ bản đến đây là bạn đã hoàn thành 1 chu trình sử dụng Git. Lúc này, nếu như bạn muốn cập nhật sự thay đổi này lên server thì bạn sẽ dùng lệnh push để đẩy chúng lên server.

### 3. Những câu lệnh cơ bản thường sử dụng

### Thiết lập chứng thực cá nhân

$ git config --global user.name "User Name"

$ git config --global user.email "username@gmail.com"

Lưu ý: --global được sử dụng để áp dụng cho tất cả các projects. Nếu bạn ko sử dụng --global thì settings sẽ chỉ dùng cho riêng project đó.

### Tạo một kho chứa Git

$ git init

Nếu như bạn muốn theo dõi một dự án cũ trong Git, bạn cần ở trong thư mục của dự án đó. Lệnh này sẽ tạo một thư mục mới có tên .git, thư mục này chứa tất cả các tập tin cần thiết cho kho chứa.

### Sao chép một kho chứa đã tồn tại

$ git clone https://github.com/user/repository.git

Câu lệnh trên sẽ tạo một thư mục mới có tên giống trên của repo.

### Nhánh trong git

Khi sử dụng Git, bạn có thể tạo ra nhiều nhánh (branch) khác nhau. Câu lệnh Git này dùng để kiểm tra branch hiện tại:

$ git branch

Để tạo mới một branch:

$ git branch <name\_branch>

Để chuyển và tạo mới:

$ git checkout -b <name\_branch>

### Chuyển nhánh

Trước khi muốn thay đổi source code, điều đầu tiên mà bạn cần phải làm là checkout một nhánh. Để checkout một nhánh, bạn dùng câu lệnh Git sau:

$ git checkout <name\_branch>

### Cập nhật thay đổi

Sau khi bạn thay đổi source code: thêm mới, sửa, xoá files,… Bạn cần phải cập nhật lên Staging Area. Để cập nhật hết các files:

$ git add .

Sau lệnh add, bạn cần sử dụng câu lệnh Commit để đây thông tin thay đổi lên Local Respository:

$ git commit -m "Message"

### Cập nhật lên server

Sau câu lệnh Commit, thông tin mới chỉ được cập nhật lên Local Repository. Nếu muốn cập nhật lên server thì bạn phải sử dụng câu lệnh push:

$ git push origin <name\_branch>

Ngoài ra, nếu chưa tồn tại remote trên server thì bạn cần phải add mới một remote trước rồi mới push:

$ git remote add origin <remote\_url>

$ git push origin <name\_branch>

### Gộp nhánh

Sau một thời gian cập nhật các file và push lên git trên branch mới, bây giờ mình cần ghép (merge) code lại vào nhánh gốc (master). Trước tiên, cần phải checkout ra khỏi branch hiện tại cần gộp để vào branch master, sau đó thì dùng lệnh merge để ghép branch mới vào master:

$ git checkout master

$ git merge <new\_branch>

### Xem lại lịch sử commit

$ git log

Lệnh git log sẽ cho bạn biết về người commit, ngày giờ, message của những lần commit đó.

### Xem thay đổi trước khi push

$ git diff

Lệnh này giúp bạn biết những gì đã được thay đổi giữa nhánh hiện tại và nhánh trước nó.

### Gộp commit

$ git rebase -i HEAD~

Sau dấu ~ là số commit bạn muốn gộp. Sau khi gõ lệnh này một cửa sổ trình soạn thảo hiện ra. Thay đổi ký tự pick của dòng các dòng sau dòng đầu thành s rồi lưu lại/kết thúc. Khi đó, trình soạn thảo để chỉnh sửa giải thích commit thiết lập cho commit sau khi đã tổng hợp sẽ được hiển thị, nên hãy chỉnh sửa lưu lại/kết thúc.

### Pull từ remote repository

$ git pull origin master

Lệnh trên sẽ gộp những thay đổi mới kéo về từ máy chủ từ xa với nhánh hiện tại trên máy local.