GitHub

Nền tảng đám mây cho phép bạn lưu trữ, chia sẻ và cùng làm việc với người khác để viết mã nguồn. Lưu trữ mã nguồn của bạn trong một "repository" (kho lưu trữ) trên GitHub giúp bạn:

- Trưng bày, chia sẻ công việc của mình.
- Theo dõi, quản lý các thay đổi trong mã nguồn theo thời gian.
- Cho phép người khác xem xét mã và đưa ra đề xuất cải thiện.
- Cộng tác trong một dự án chung mà không lo những thay đổi của bạn ảnh hưởng đến công việc của người khác trước khi bạn sẵn sàng tích hợp chúng.

GitHub cung cấp tính năng **làm việc cùng nhau**, được thực hiện nhờ vào **Git**, một phần mềm mã nguồn mở giúp quản lý và theo dõi các thay đổi trong mã nguồn. GitHub là nền tảng dựa trên công nghệ Git, tức là GitHub sử dụng Git để hỗ trợ việc làm việc nhóm, theo dõi sự thay đổi và hợp tác trên các dự án phần mềm.

Git

Hệ thống kiểm soát phiên bản, giúp theo dõi sự thay đổi trong các tệp tin. Git đặc biệt hữu ích khi bạn và nhóm của bạn cùng làm việc và thay đổi các tệp tin giống nhau cùng một lúc.

- Tạo một nhánh (branch) từ bản chính của các tệp tin mà bạn (và cộng tác viên của bạn) đang làm việc.
- Thực hiện chỉnh sửa các tệp tin một cách độc lập và an toàn trên nhánh cá nhân của bạn.
- Git sẽ tự động xử lý việc **gộp các thay đổi** mà bạn đã thực hiện vào **bản chính** của các tệp tin mà nhóm bạn đang làm việc, sao cho các thay đổi của bạn không làm xung đột hoặc ảnh hưởng đến các thay đổi mà những người khác đã thực hiện trên tệp tin đó.

  + Khi bạn và đồng nghiệp gộp thay đổi vào nhánh chính, Git sẽ nhận thấy rằng hai bạn đã thay đổi các phần khác (bạn chỉnh sửa phần tiêu đề và đồng nghiệp chỉnh sửa phần nội dung), nên Git sẽ tự động **gộp các thay đổi** lại mà không gặp phải xung đột.
  + Nếu bạn và đồng nghiệp thay đổi cùng một phần của tệp (ví dụ, cả hai đều thay đổi dòng tiêu đề), Git sẽ không thể tự động gộp các thay đổi và sẽ báo lỗi xung đột. Khi đó, bạn sẽ cần phải **giải quyết xung đột** bằng cách chỉnh sửa tệp tin thủ công và chọn giữ thay đổi nào, rồi sau đó tiếp tục gộp lại.

- Để Git theo dõi các thay đổi của bạn và những người khác, giúp tất cả mọi người làm việc trên phiên bản cập nhật nhất của dự án.

Git và GitHub hoạt động cùng nhau như thế nào?

Khi bạn tải tệp lên **GitHub**, những tệp đó sẽ được lưu trữ trong một **Git repository** (kho lưu trữ Git). Có nghĩa là khi bạn thực hiện các thay đổi (hay còn gọi là "commits") đối với các tệp của mình trên GitHub, Git sẽ tự động bắt đầu theo dõi và quản lý các thay đổi đó.

Làm việc trực tiếp trên GitHub

Bạn có thể thực hiện nhiều thao tác liên quan đến Git trực tiếp trên GitHub thông qua trình duyệt của mình. Ví dụ:

- Tạo một **Git repository**.
- Tạo các **nhánh (branches)**.
- Tải lên và chỉnh sửa các tệp trực tiếp trên GitHub.

Làm việc cục bộ (trên máy tính của bạn)

Phần lớn các lập trình viên làm việc trên các tệp tin của họ **cục bộ** (trên máy tính cá nhân) và sau đó **đồng bộ hóa** những thay đổi này cùng tất cả các dữ liệu Git liên quan với **repository** từ xa trên GitHub.

Để thực hiện điều này, bạn có thể sử dụng các công cụ như **GitHub Desktop**, hoặc sử dụng dòng lệnh với Git.

Quy trình làm việc khi cộng tác

Khi bạn bắt đầu cộng tác với người khác và tất cả cùng làm việc trên một repository chung, quy trình làm việc sẽ bao gồm các bước:

- **Pull** (kéo) tất cả các thay đổi mới nhất từ các cộng tác viên của bạn từ **repository remote** trên GitHub về máy của bạn.
- **Push** (đẩy) các thay đổi của bạn lên cùng **repository remote** trên GitHub.

Tạo tài khoản GiTHub

Truy cập vào https://github.com/.

Cài đặt Git trên máy cá nhân

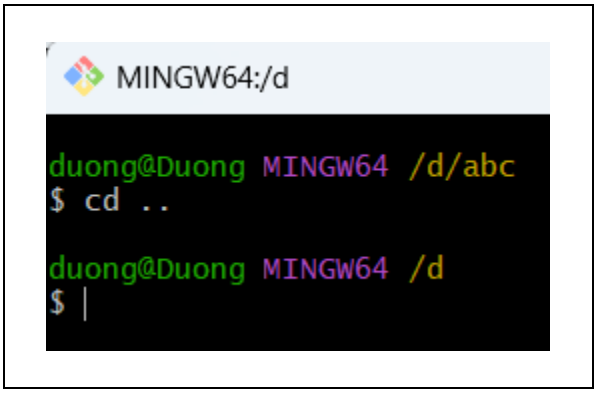
Truy cập: https://git-scm.com/downloads

CÁC CÂU LỆNH CƠ BẢN ĐỂ QUẢN LÝ FILE VÀ THƯ MỤC TRONG CMD

Chọn một thư mục bất kỳ → Nhấn phải chuột → Show more options → Open Git Bash here
Sau đây sẽ ví dụ với một thư mục **abc** chứa ba thư mục con **abc1**, **abc2** và **abc3**

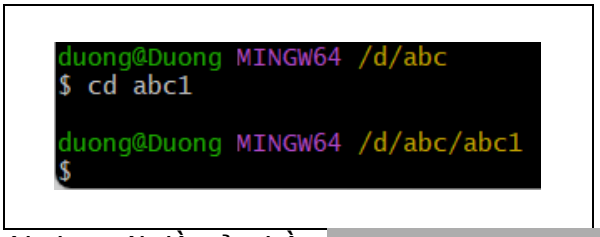
Quay về thư mục cha

Cú pháp: `cd ..`
Kết quả



Truy cập thư mục con

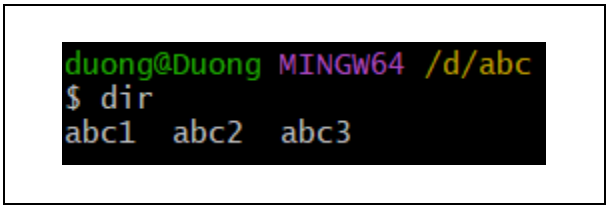
Cú pháp: `cd <ten_thu_muc_con>`
Kết quả



Lưu ý: có thể chỉ cần gõ một vài chữ cái đầu ở phần `<tên thư mục con>`

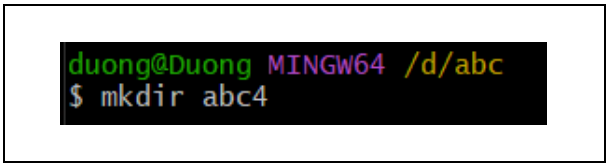
Hiển thị các thư mục/tệp bên trong thư mục đang làm việc

Cú pháp: `dir` hoặc `ls`
Kết quả:



Tạo thư mục mới

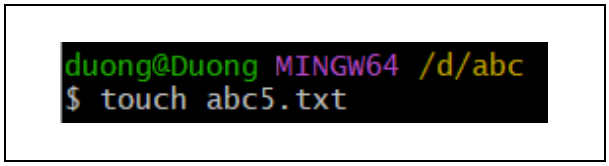
Cú pháp: `mkdir <folder_name>` hoặc `mkdir "<folder_name>"` (nháy kép nếu tên chứa dấu cách)
Kết quả:



Tạo tệp mới

Cú pháp: `touch <file_name>`

Kết quả:



In chuỗi ra màn hình

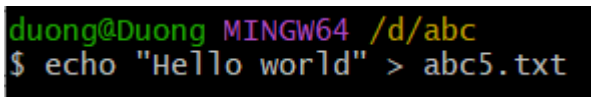
Cú pháp: `echo “<thông_điệp>”`

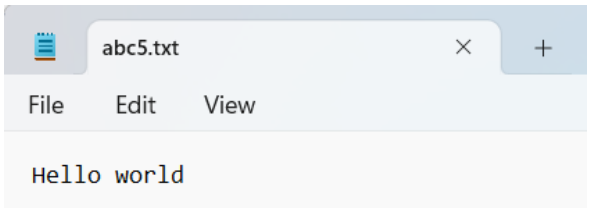
Ví dụ:

Ghi đè nội dung lên một file

Cú pháp: `echo “<thông_điệp>” > <tên_file>`

Ví dụ

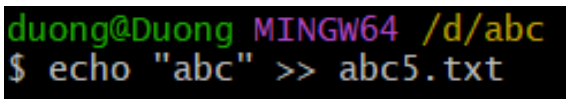


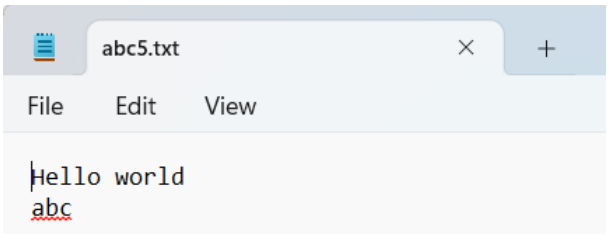


Thêm dòng mới lên một file mà không ghi đè

Cú pháp: `echo “<thông_điệp>” >> <tên_file>`

Ví dụ





Tạo file mới bằng lệnh echo

Cú pháp: `echo > <tên_file>`

Hiển thị nội dung của file

Cú pháp: `cat <tên_file>`

Tìm các nội dung khác biệt giữa hai file

Cú pháp: `diff <tên_file_1> <tên_file_2>`

Xóa

Cú pháp xóa file: `rm <tên_file>`

Cú pháp xóa folder: `rm -d <tên_folder>` (thư mục phải rỗng mới có thể xóa)

Cú pháp xóa folder: `rm -r <tên_folder>` (thư mục bất kỳ)

TẠO REPOSITORY TRÊN MÁY TÍNH

Thuật ngữ cơ bản

repository: kho lưu trữ tệp tin, hình ảnh, video hoặc thư mục con.
commit: lưu những thay đổi vào repository
branch: những bản sao của dự án, nơi các thay đổi không làm ảnh hưởng đến sự ổn định của công việc hiện tại.
main/master: tên của nhánh chính, phiên bản chính thức và ổn định của dự án.
merge/rebase: kết hợp các nhánh lại với nhau
Tải xuống một bản sao các tệp trong repository dưới dạng tệp zip về máy tính (local).

Tải một **snapshot** (bản sao) của các tệp trong repository dưới dạng tệp zip về máy tính cá nhân.
Phù hợp khi chỉ muốn tải về các tệp mà không cần sử dụng Git.

Clone một repository về máy tính của bạn sử dụng Git.

Tạo ra một bản sao của repository **trên máy tính của mình** và có thể sử dụng Git để quản lý các thay đổi sau này.
Phù hợp khi muốn theo dõi các thay đổi và hợp nhất chúng vào dự án của mình.

Fork một repository để tạo ra một repository mới trên GitHub.

Tạo ra một bản sao của repository **trên GitHub**, nơi bạn có thể tự do thay đổi mà không ảnh hưởng đến repository gốc. Sau đó, bạn có thể mở pull request để đề xuất thay đổi của mình với repository gốc.
Phù hợp khi muốn sử dụng dữ liệu từ repository gốc làm cơ sở cho dự án riêng trên GitHub, hoặc muốn đề xuất thay đổi cho repository gốc. Sau khi fork repository, bạn có thể clone repository đó về máy tính để làm việc trên các thay đổi của mình.

Câu lệnh cơ bản

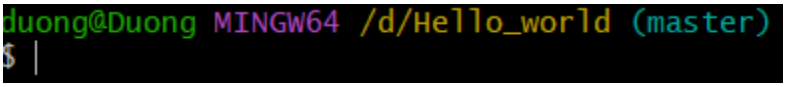
Trợ giúp/hướng dẫn: `git --help`
Hiển thị trạng thái kho lưu trữ: `git status`
Hiển thị lịch sử các commit: `git log`

Tạo một repository trống

Tạo repository trống: `git init <repo_name>`
Tạo một bản sao được liên kết với repository: `git clone <repo_name> <clone_name>`
Xem cấu hình hiện tại: `git config -l`

Thực hành

1. Chọn một thư mục bất kỳ
2. Mở Git Bash
3. Gõ lệnh `git init Hello_world` (tạo repository mới)
4. Gõ lệnh `cd Hello_world`
5. Kiểm tra thông tin bên trong



(master) cho thấy bạn đang ở nhánh master trong Git.

CẤU HÌNH THÔNG TIN CHO REPOSITORY

Cấu hình: `git config -l [--scope] [option_name] [value]`
Mục đích:

- Xác định phạm vi mà Git sẽ áp dụng, giúp Git biết các value bạn cung cấp bên trên sẽ được dùng ở đâu.
- Giúp người khác biết được các chỉnh sửa thuộc về ai

Có ba phạm vi chính trong Git:

1. --global
   Cấu hình này áp dụng cho tất cả các kho lưu trữ Git của bạn trên máy tính.
   Ví dụ: Nếu bạn cấu hình tên người dùng (user.name) hoặc email (user.email) với --global, Git sẽ sử dụng giá trị đó cho tất cả các dự án của bạn.
   ```
   git config --global user.name "Nguyễn Văn A"
   ```
2. --local
   Cấu hình này chỉ áp dụng cho kho lưu trữ Git hiện tại mà bạn đang làm việc.
   Nếu bạn không chỉ định phạm vi khi sử dụng Git, mặc định Git sẽ áp dụng cấu hình này (tức là --local).
   ```
   git config user.email "example@example.com"
   ```
3. --system
   Cấu hình này áp dụng cho tất cả người dùng trên máy tính của bạn, tức là nếu bạn làm việc trên máy tính mà nhiều người cùng sử dụng Git, cấu hình này sẽ ảnh hưởng đến tất cả mọi người.

Thực hành

1. Chọn thư mục chứa repository **Hello_world**
2. Mở Git Bash
3. Gõ lệnh `cd Hello_world`
4. Gõ lệnh `git config -l`
   Kết quả:
5. Cấu hình (global) ‘user.name’: `git config --global user.name “Nguyen Tuan Duong”`
6. Cấu hình (global) ‘user.email’: `git config --global user.email “duonggk05@gmail.com”`
7. Làm tương tự như bước 5 và 6 nhưng với local thay vì global

THỰC HÀNH GIT ADD | GIT COMMIT | GIT STATUS | GIT DIFF | GIT LOG

Các khu vực chính trong quy trình làm việc của Git

Working Directory (Thư mục làm việc)

- Nơi làm việc trực tiếp, tức là thư mục trên máy tính chứa các file code (hoặc bất kỳ file đang quản lý bằng Git).
- Nghĩ đơn giản: Nó giống như cái bàn làm việc bừa bộn. Bạn viết code, sửa file, thêm bớt lung tung ở đây. Nhưng Git chưa quan tâm mấy thay đổi này cho đến khi bạn bảo nó.
- **Ví dụ**: Bạn có file game.py trong thư mục dự án. Bạn mở file đó, thêm vài dòng code, lưu lại. Lúc này, thay đổi chỉ nằm trong "working directory", Git chưa biết gì.

Staging Area (Khu vực chuẩn bị)

- Đây là "giỏ đựng đồ" tạm thời trước khi chính thức lưu thay đổi vào kho. Bạn chọn những thay đổi nào muốn giữ, bỏ vào đây để sẵn sàng "commit".
- Nghĩ giống như: Bạn dọn bàn làm việc, nhặt mấy thứ cần giữ (file bạn sửa) bỏ vào giỏ, còn mấy thứ vớ vẩn thì để lại.
- **Ví dụ**: Bạn sửa 3 file: game.py, readme.txt, bug.txt. Nhưng bạn chỉ muốn lưu thay đổi của game.py thôi. Bạn dùng lệnh **git add game.py** để đưa nó vào staging area, 2 file kia vẫn nằm ngoài.
- **Lệnh cơ bản**:
  - `git add <tên_file>:` Đưa file vào staging.
  - `git add .:` Đưa hết mọi thay đổi vào staging.

Repository (Kho lưu trữ)

- Đây là "két sắt" chính thức của Git, nơi mọi thay đổi được lưu vĩnh viễn dưới dạng các phiên bản (commit). Có 2 loại:
  - **Local Repository**: Kho trên máy.
  - **Remote Repository**: Kho trên mạng (như GitHub).
- Nghĩ giống như: Sau khi nhặt đồ vào giỏ (staging), bạn khóa nó vào két sắt (commit) để giữ an toàn, sau này muốn xem lại hay lấy ra xài vẫn được.
- **Ví dụ**: Bạn commit thay đổi của game.py bằng lệnh git commit -m "thêm tính năng bắn súng". Lúc này, nó được lưu vào repository, kèm ghi chú để bạn nhớ.
- **Lệnh cơ bản**:
  - `git commit -m "mô tả"`: Lưu từ staging vào repository.

Working Directory: Bạn sửa file thoải mái.
Staging Area: Bạn chọn thay đổi nào cần giữ (git add).
Repository: Bạn lưu chính thức (git commit).

HEAD, Index và Working Directory

Đây là cách Git gọi các khái niệm trên nhưng với góc nhìn kỹ thuật hơn

Working Directory:

- Giống y chang cái Working Directory ở trên: thư mục thực tế trên máy, nơi bạn sửa file.
- Không có gì khác biệt cả, chỉ là tên gọi lại để so sánh với HEAD và Index.

Index

- Chính là **Staging Area**! Tên "Index" là cách gọi kỹ thuật của Git để chỉ khu vực tạm chứa thay đổi trước khi commit.
- Ví dụ: Bạn chạy **git add abc3**, thì abc3 nằm trong Index (hay Staging Area).

HEAD:

- Là "con trỏ" chỉ đến phiên bản hiện tại bạn đang làm việc trong repository.
- Nghĩ đơn giản: HEAD giống như cái "điểm đánh dấu" trên dòng thời gian của repository. Nó thường trỏ đến commit mới nhất trên nhánh bạn đang đứng (thường là main hoặc master).
- Ví dụ: Repository có 3 commit, HEAD đang trỏ vào commit cuối cùng. Nếu bạn checkout nhánh khác hoặc commit mới, HEAD sẽ di chuyển.

Các câu lệnh

Thêm các file vào index: `git add <file_name(s)>`

Thêm tất cả các file vào index: `git add .`

Tạo một commit mới (thay đổi đã thêm vào index sẽ vào repository): `git commit -m “message”`

Hiển thị các thông tin được liệt kê bên dưới: `git status`

Thay đổi trong Working Directory:

- File nào đã sửa nhưng chưa được thêm vào staging area (chưa git add).
- File mới tạo (untracked) mà Git chưa quản lý.
- File bị xóa nhưng chưa được ghi nhận.

Thay đổi trong Staging Area (Index):

- File nào đã được git add và sẵn sàng để commit.

Trạng thái so với Repository:

- Nhánh đang đứng (ví dụ: main).
- Bạn có commit gì mới hơn remote repository (nếu liên kết với GitHub chẳng hạn) hay không.

Tìm sự khác biệt: `git diff`

`git diff` (không có tham số)

- So sánh **Working Directory** với **Staging Area**.
- Hiển thị những thay đổi đã làm trong file nhưng chưa git add.

`git diff --staged` (hoặc `--cached`)

- So sánh **Staging Area** với **Repository** (HEAD).
- Hiển thị những thay đổi đã git add nhưng chưa commit.

`git diff HEAD`

- So sánh **Working Directory** với **Repository** (HEAD).
- Hiển thị tất cả thay đổi trong working directory (bao gồm cả chưa add và đã add) so với commit mới nhất.

Ý nghĩa các ký hiệu:

- **-**: Dòng bị xóa (trong phiên bản cũ).

- **+**: Dòng được thêm (trong phiên bản mới).
- **@@ -x,y +a,b @@**: Vị trí thay đổi (x,y là dòng cũ, a,b là dòng mới).

Khi nào dùng?

- Chưa git add: Dùng git diff để kiểm tra thay đổi trong working directory.
- Đã git add: Dùng git diff --staged để xem thay đổi chuẩn bị commit.
- So sánh tổng thể: Dùng git diff HEAD hoặc giữa các commit để xem lịch sử.

Xem lịch sử commit trong repository: `git log`

Liệt kê tất cả các commit trong repository, từ commit mới nhất (nơi HEAD đang trỏ) ngược về commit đầu tiên. Mỗi commit đi kèm thông tin:

Commit hash: Mã định danh duy nhất (dạng chuỗi dài như a1b2c3d4...).

Tác giả: Tên và email của người commit.

Ngày giờ: Khi nào commit được tạo.

Thông điệp: Nội dung bạn ghi khi commit (ví dụ: git commit -m "thêm tính năng mới").

`git log` (không tham số)

- Hiển thị danh sách đầy đủ các commit theo thứ tự thời gian (mới nhất ở trên).
- **Ví dụ**: Giả sử bạn có 3 commit trong repository:

```
commit a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6 (HEAD -> main)
Author: Tao <tao@example.com>
Date:   Wed Apr 9 12:00:00 2025 +0700
    Sửa abc1

commit q1w2e3r4t5y6u7i8o9p0a1s2d3f4g5h
Author: Tao <tao@example.com>
Date:   Tue Apr 8 10:00:00 2025 +0700
    Thêm abc3

commit z1x2c3v4b5n6m7a8s9d0f1g2h3j4k5l
Author: Tao <tao@example.com>
Date:   Mon Apr 7 09:00:00 2025 +0700
    Khởi tạo dự án
```

`git log --oneline`

- Hiển thị lịch sử commit ngắn gọn, mỗi commit chỉ 1 dòng.

```
a1b2c3d Sửa abc1
q1w2e3r Thêm abc3
z1x2c3v Khởi tạo dự án
```

`git log -n <số>`

- Chỉ hiển thị <số> commit gần nhất.
- **Ví dụ**: `git log -n 2`:

```
commit a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6 (HEAD -> main)
Author: Tao <tao@example.com>
Date:   Wed Apr 9 12:00:00 2025 +0700
    Sửa abc1

commit q1w2e3r4t5y6u7i8o9p0a1s2d3f4g5h
Author: Tao <tao@example.com>
Date:   Tue Apr 8 10:00:00 2025 +0700
    Thêm abc3
```

`git log –stat`

- Hiển thị thêm thông tin về file nào thay đổi trong mỗi commit.

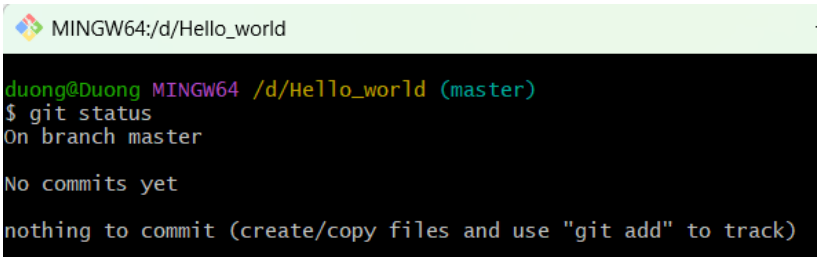
`git log -p (hoặc --patch)`

- Hiển thị cả chi tiết thay đổi (diff) trong từng commit, giống như git diff.

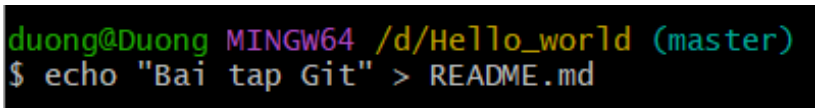
Thực hành

1. Mở repository **Hello_world** đã tạo

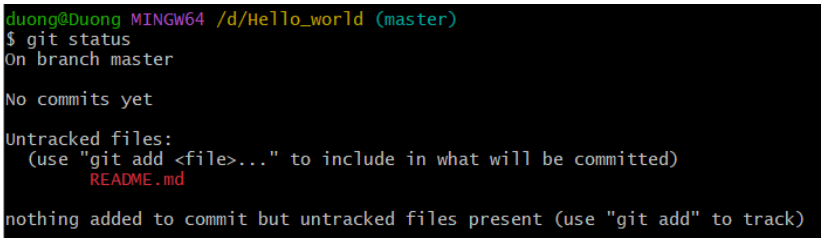
2. Kiểm tra trạng thái: `git status`



3. Tạo file readme: echo “Bai tap Git” > README.md

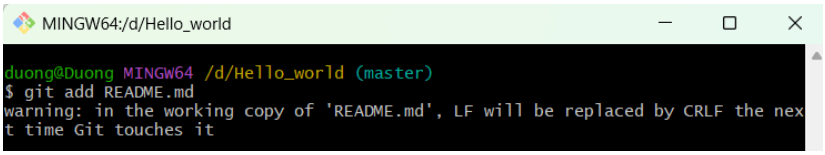


4. Kiểm tra trạng thái: `git status`



Màu đỏ → Chưa được đưa vào Index, cần lệnh **git add** để đưa vào

5. Thêm vào Index: `git add README.md`



6. Kiểm tra trạng thái: `git status`

```
duong@Duong MINGW64 /d/Hello_world (master)
$ git status
On branch master

No commits yet

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

Màu xanh lá cây → Thông báo file mới/file bị xóa đi/các dòng được thay đổi

7. Thêm các thay đổi trong Index vào Repository: `git commit -m "Them tap tin README"`

```
duong@Duong MINGW64 /d/Hello_world (master)
$ git commit -m "Them tap tin README"
[master (root-commit) a9c94c0] Them tap tin README
 1 file changed, 1 insertion(+)
 create mode 100644 README.md
```

8. Kiểm tra trạng thái: git status

```
duong@Duong MINGW64 /d/Hello_world (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Tất cả các thay đổi đã được lưu vào Repo

9. Xem lịch sử: `git log`

```
duong@Duong MINGW64 /d/Hello_world (master)
$ git log
commit a9c94c044d693365d97154313d25fc5778460aa4 (HEAD -> master)
Author: Nguyen Tuan Duong <duonggk05@gmail.com>
Date:   Thu Apr 10 14:57:32 2025 +0700

    Them tap tin README
```

Có thể xác định được ai đã chỉnh sửa, lúc nào, chỉnh sửa cái gì

BỎ QUA CÁC FILE KHÔNG CẦN GIÁM SÁT

Để bỏ qua các file hoặc thư mục không cần Git giám sát (tracked), bạn cần dùng file .gitignore. Đây là một file đặc biệt trong Git, cho bạn mày liệt kê các file, thư mục, hoặc mẫu (pattern) mà Git sẽ không quan tâm, dù chúng nằm trong working directory.

.gitignore là gì?

- Là file text nằm trong thư mục dự án (thường ở thư mục gốc).
- Bạn ghi vào đó các quy tắc để Git bỏ qua file/thư mục khi kiểm tra trạng thái (git status) hoặc khi thêm file (git add).

Thực hành

1. Mở repository **Hello_world** đã tạo

2. Tạo file ignore: `touch .gitignore`

```
duong@Duong MINGW64 /d/Hello_world (master)
$ touch .gitignore
```

3. Tạo một file txt với nội dung "Hello abc": `echo "Hello abc" > hello.txt`

```
duong@Duong MINGW64 /d/Hello_world (master)
$ echo "Hello abc" > hello.txt
```

4. Tạo một file "abc.log": touch abc.log

```
duong@Duong MINGW64 /d/Hello_world (master)
$ touch abc.log
```

5. Tạo một thư mục: `mkdir "Thu Muc 1"`

```
duong@Duong MINGW64 /d/Hello_world (master)
$ mkdir "Thu Muc 1"
```

6. Bỏ qua thư mục "Thu Muc 1": `echo "Thu Muc 1/" > .gitignore`

```
duong@Duong MINGW64 /d/Hello_world (master)
$ echo "Thu Muc 1/" > .gitignore
```

   Git sẽ **bỏ qua thư mục có tên chính xác là "Thu Muc 1"** (bao gồm tất cả file và thư mục con bên trong).

7. Bỏ qua tất cả các file có đuôi .log: `echo "*.log" > .gitignore`

```
duong@Duong MINGW64 /d/Hello_world (master)
$ echo "*.log" > .gitignore
```

   Git sẽ **bỏ qua tất cả các file có đuôi .log** trong toàn bộ dự án, bất kể chúng nằm ở thư mục nào.

8. Kiểm tra trạng thái: git status

```
duong@Duong MINGW64 /d/Hello_world (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        hello.txt

nothing added to commit but untracked files present (use "git add" to track)
```

Chỉ báo file .gitignore và hello.txt chưa được thêm vào Index, vì các file còn lại đã bị bỏ qua

9. Đưa thay đổi vào Index: `git add.`

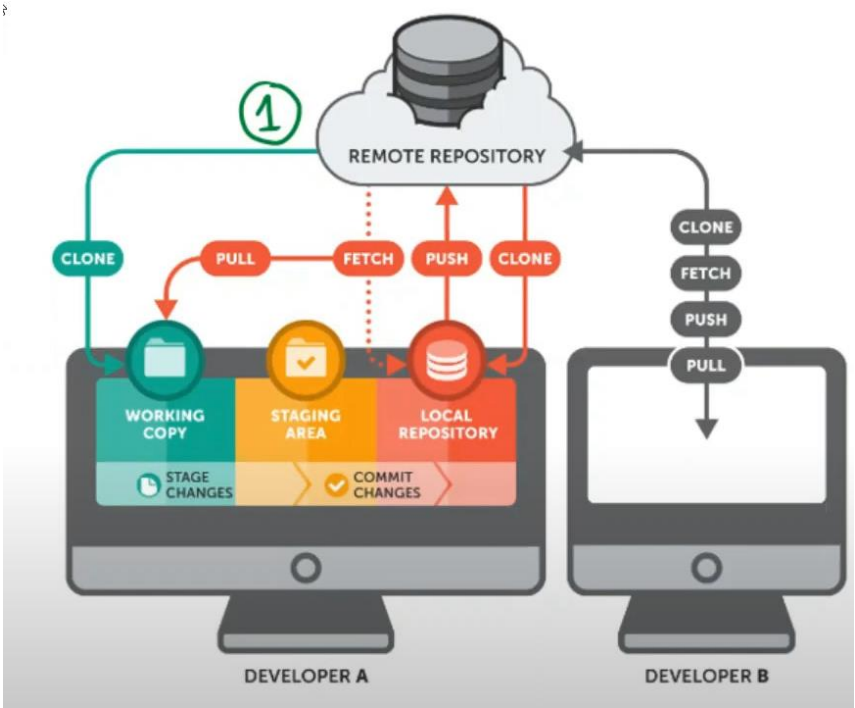
```
duong@Duong MINGW64 /d/Hello_world (master)
$ git add .
warning: in the working copy of '.gitignore', LF will be replaced by CRLF the ne
xt time Git touches it
warning: in the working copy of 'hello.txt', LF will be replaced by CRLF the nex
t time Git touches it
```

10. Đưa thay đổi vào Repository: `git commit -m “Text ignore”`

```
duong@Duong MINGW64 /d/Hello_world (master)
$ git commit -m "Test ignore"
[master 663be4a] Test ignore
 2 files changed, 2 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 hello.txt
```

REMOTE REPOSITORY/CENTRAL REPOSITORY

Remote repository: Một phiên bản của repository (kho lưu trữ) được lưu trữ **trên một máy chủ từ xa** (thường là online), không nằm trên máy tính của bạn. Nó giống như "kho chính" trên mạng mà bạn và người khác có thể đẩy code lên (push) hoặc kéo code về (pull). Ví dụ: GitHub



git clone

- Sao chép toàn bộ một remote repository về máy, tạo một **local repository** mới.
- Nó giống như "tải nguyên cái kho" từ mạng về, bao gồm cả lịch sử commit.

git pull

- Kéo cập nhật từ remote repository về local repository.
- Nó giống như "đồng bộ" những thay đổi mới nhất từ remote về máy.

git push

- Đẩy thay đổi từ local repository lên remote repository.
- Nó giống như "upload" những commit bạn đã làm lên kho từ xa.

Bare repository

Là một repository không có **working directory** hay **staging area**. Nó chỉ chứa dữ liệu Git thuần túy (lịch sử commit, nhánh, v.v.), không có file thực tế mà bạn có thể sửa trực tiếp.
Thư mục này được thiết kế để làm **remote repository** trên máy chủ (GitHub), nơi mọi người đẩy code lên (push) và kéo code về (pull/clone).
Không cần working directory: Vì không ai làm việc trực tiếp trên bare repo, nó phù hợp để lưu trữ và quản lý lịch sử commit.

Thực hành

1. Tạo một thư mục tên **Example**: `mkdir Example`
2. Vào trong thư mục trên, chọn **Git Bash here**
3. Tạo một central repository (minh họa cho một kho lưu trữ chung của nhóm):
   `git init --bare Central_Repo`

   ```
   duong@Duong MINGW64 /d/Example
   $ git init --bare Central_Repo
   Initialized empty Git repository in D:/Example/Central_Repo/
   ```

   Lúc này, trong **Example** xuất hiện thư mục **Central_Repo** chứa dữ liệu Git thuần túy (không có working directory).
   Central_Repo là một kho Git "trống", không có file để chỉnh sửa trực tiếp. Nó giống như một "remote repo" trên server, nơi bạn sẽ đẩy/kéo code.
4. Tạo hai local repository mới tên **Dev1 và Dev2** (minh họa cho hai lập trình viên làm việc tại hai máy tính cá nhân):
   `git clone Central_Repo Dev1`
   `git clone Central_Repo Dev2`

   ```
   duong@Duong MINGW64 /d/Example
   $ git clone Central_Repo Dev1
   Cloning into 'Dev1'...
   warning: You appear to have cloned an empty repository.
   done.
   ```

   Dev1 là local repo với working directory, liên kết với **Central_Repo** (remote mặc định là origin).
   Dev1 là nơi bạn có thể làm việc (thêm file, commit), còn Central_Repo là kho trung tâm để đồng bộ.
   *Trong trường hợp trên central repository **Central_Repo** nằm trong máy của bạn. Thực tế khi làm việc nhóm, central repository nằm trên GitHub, các thành viên clone chúng từ web về máy để chỉnh sửa. Lúc này lệnh sẽ là:* `git clone <link_to_central_repo> Dev1`
5. Chuyển vào trong local repo Dev1: `cd Dev1`
6. Cấu hình (để cho mọi người biết bạn là người chỉnh sửa,…):
   `git config --local user.name “Developer_1”`
7. Tạo một file và thêm vào Staging area, sau đó commit:
   `echo “Day la repo Dev1” > intro.txt”`
   `git add .`
   `git commit -m “Them file intro.txt”`
8. Đẩy các chỉnh sửa từ local repo của Dev1 lên central repo **Central_Repo**: `git push`
   commit: lưu thay đổi vào local repo
   push: đẩy commit từ local repo lên remote repo
   Ví dụ:
   ***git commit**: Bạn viết bài tập vào vở nháp. Bạn có thể viết 1 trang, 10 trang, hay cả cuốn vở mà không cần nộp.*
   ***git push**: Bạn nộp vở nháp lên Google Drive để cô giáo xem hoặc bạn bè lấy. Nhưng nếu bạn chưa muốn nộp, cứ để trong vở nháp cũng được.*

9. Chuyển sang Dev2:

   ```
   cd ..
   cd Dev 2
   git config --local user.name “Developer_2”
   ```

10. Cập nhật công việc Dev1 đã làm (Dev1 đã thêm file intro.txt lên remote repo)

    ```
    git pull
    ```

    Phân biệt **`git pull`** và **`git fetch`**

`git pull`	`git fetch`
Tải và áp dụng luôn thay đổi từ remote vào nhánh hiện tại Kết hợp của git fetch và git merge Code trong thư mục làm việc được cập nhật ngay lập tức.	Tải thông tin mới nhất từ remote về máy, nhưng không tự động cập nhật code đang làm Lưu vào một khu vực tạm thời trong nhánh Code hiện tại chưa bị thay đổi Dùng khi muốn xem có gì thay đổi mới không mà không muốn áp dụng ngay

Conflict (xung đột) có thể xảy ra khi nhiều người làm việc với cùng một file
Ví dụ:

7h	7h15	7h30
Dev1 và Dev2 cùng pull một file về để làm việc	Dev1 sửa file đó và push lên. Dev2 vẫn đang làm việc với file cũ chưa được cập nhật	Dev2 push file lên → Conflict

Thực hành

1. Tạo một thư mục **Example**, bên trong thư mục đó tạo một remote repo **Central_Repo**

```
duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example
$ git init --bare Central_Repo
Initialized empty Git repository in D:/TaiLieuuuuuuuuuu/Git/Example/Central_Repo
/
```

2. Tạo hai local repository mới tên **Dev1 và Dev2** (minh họa cho hai lập trình viên làm việc tại hai máy tính cá nhân):

```
duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example
$ git clone Central_Repo Dev1
Cloning into 'Dev1'...
warning: You appear to have cloned an empty repository.
done.

duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example
$ git clone Central_Repo Dev2
Cloning into 'Dev2'...
warning: You appear to have cloned an empty repository.
done.
```

3. Lập trình viên **Dev1** tạo một file a.txt và ghi nội dung “Toi la Dev 1”, sau đó add, commit và push.

```
duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example
$ cd Dev1

duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Dev1 (master)
$ echo "Toi la Dev 1" > a.txt

duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Dev1 (master)
$ git add .
warning: in the working copy of 'a.txt', LF will be replaced by CRLF the next ti
me Git touches it

duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Dev1 (master)
$ git commit -m "Them file a.txt"
[master (root-commit) 36ee620] Them file a.txt
 1 file changed, 1 insertion(+)
 create mode 100644 a.txt

duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Dev1 (master)
$ git push
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 236 bytes | 236.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To D:/TaiLieuuuuuuuuuu/Git/Example/Central_Repo
 * [new branch]      master -> master
```

4. Lập trình viên **Dev2** pull file a.txt về máy mình

```
duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Dev1 (master)
$ cd ..

duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example
$ cd Dev2

duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Dev2 (master)
$ git pull
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (3/3), 216 bytes | 43.00 KiB/s, done.
From D:/TaiLieuuuuuuuuuu/Git/Example/Central_Repo
 * [new branch]      master     -> origin/master

duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Dev2 (master)
$
```

5. Lập trình viên **Dev1** thêm một dòng mới “Toi van la Dev 1” vào file a.txt, sau đó add, commit và push.

```
duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Dev2 (master)
$ cd ..

duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example
$ cd Dev1

duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Dev1 (master)
$ echo "Toi van la Dev 1" >> a.txt

duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Dev1 (master)
$ git add .
warning: in the working copy of 'a.txt', LF will be replaced by CRLF the next time Git touches it

duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Dev1 (master)
$ git commit -m "Them mot dong moi vao file a.txt"
[master 53caeb7] Them mot dong moi vao file a.txt
 1 file changed, 1 insertion(+)

duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Dev1 (master)
$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Writing objects: 100% (3/3), 283 bytes | 283.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To D:/TaiLieuuuuuuuuuu/Git/Example/Central_Repo
   36ee620..53caeb7  master -> master
```

Lúc này, Dev2 vẫn đang làm việc với file a.txt cũ, bởi vì anh ấy chưa pull file a.txt mới về.

6. Dev2 thêm một dòng vào file a.txt “Toi la Dev 2”, sau đó add, commit và push

```
duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Dev1 (master)
$ cd ..

duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example
$ cd Dev2

duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Dev2 (master)
$ echo "Toi la Dev2" >> a.txt

duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Dev2 (master)
$ git add .
warning: in the working copy of 'a.txt', LF will be replaced by CRLF the next time Git touches it

duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Dev2 (master)
$ git commit -m "Dev2 them mot dong vao file a.txt"
[master 5295040] Dev2 them mot dong vao file a.txt
 1 file changed, 1 insertion(+)
```

```
duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Dev2 (master)
$ git push
To D:/TaiLieuuuuuuuuuu/Git/Example/Central_Repo
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'D:/TaiLieuuuuuuuuuu/Git/Example/Central_Repo'
hint: Updates were rejected because the remote contains work that you do not
hint: have locally. This is usually caused by another repository pushing to
hint: the same ref. If you want to integrate the remote changes, use
hint: 'git pull' before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Thông báo này nghĩa là: bạn đang cố đẩy (push) lên Central_Repo, **nhưng repo từ xa đã có commit mới mà bạn chưa có trong máy** → Git không cho phép bạn đẩy thẳng vì sợ làm mất dữ liệu của người khác.

7. Dev2 phải pull bản mới nhất của file a.txt xuống

```
duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Dev2 (master)
$ git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (3/3), 250 bytes | 25.00 KiB/s, done.
From D:/TaiLieuuuuuuuuuu/Git/Example/Central_Repo
   a723a14..a0cb342  master     -> origin/master
Updating a723a14..a0cb342
error: Your local changes to the following files would be overwritten by merge:
        a.txt
Please commit your changes or stash them before you merge.
Aborting
```

Git đã thử tự động gộp (merge) file a.txt giữa bản bạn đang có và bản mới trên Central_Repo.
Nhưng có nội dung **xung đột**, không thể tự giải quyết.
Bạn cần **mở file a.txt và xử lý phần bị đụng độ** thủ công.

`Toi la Dev1`
`<<<<<<< HEAD`
`Toi la Dev2`
`=======`
`Toi van la Dev1`
`>>>>>>> 2c04d8ff68cf33943f54cb90cb057363628f8d19`
	Toi la Dev1
	Toi la Dev2
	Toi van la Dev1

Lưu file, add, commit, push

Cú pháp: `git checkout <commit_hash>`

Thực hành

1. Tạo một local repo **Example**, trong repo đó có một file **a.txt** và một thư mục **B,** bên trong thư mục B tạo một file **b.txt**

```
duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example
$ git init Example
Initialized empty Git repository in D:/TaiLieuuuuuuuuuu/Git/Example/Example/.git
/

duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example
$ cd Example

duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Example (master)
$ echo "A" > a.txt

duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Example (master)
$ mkdir B

duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Example (master)
$ cd B

duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Example/B (master)
$ echo "B" > b.txt
```

Bên trong repo Example, add và commit

```
duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Example/B (master)
$ cd ..

duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Example (master)
$ git add .
warning: in the working copy of 'B/b.txt', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'a.txt', LF will be replaced by CRLF the next time Git touches it

duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Example (master)
$ git commit -m "Them mot file va mot thu muc"
[master (root-commit) 9b07698] Them mot file va mot thu muc
 2 files changed, 2 insertions(+)
 create mode 100644 B/b.txt
 create mode 100644 a.txt
```

2. Thêm một dòng vào file a.txt, add và commit

```
duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Example (master)
$ echo "AA" >> a.txt
```

```
duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Example (master)
$ git add .
warning: in the working copy of 'a.txt', LF will be replaced by CRLF the next time Git touches it

duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Example (master)
$ git commit -m "Them mot dong vao file a.txt"
[master 8cdc280] Them mot dong vao file a.txt
 1 file changed, 1 insertion(+)
```

3. Xem các lần commit: `git log`

```
duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Example (master)
$ git log
commit 8cdc2807d691149aa761b7c5a2f2482e89881ad6 (HEAD -> master)
Author: Nguyen Tuan Duong <duonggk05@gmail.com>
Date:   Sat Apr 12 16:50:34 2025 +0700

    Them mot dong vao file a.txt

commit 9b07698f7d49b4ff20c0b79ca89c8f44b4cd86ba
Author: Nguyen Tuan Duong <duonggk05@gmail.com>
Date:   Sat Apr 12 16:45:58 2025 +0700

    Them mot file va mot thu muc
```

4. Quay lại trạng thái trước đó, trong trường hợp này là commit có mã hash là 9b076…(chỉ cần gõ một vài kí tự đầu):

`git checkout 9b076`

```
duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Example (master)
$ git checkout 9b076
Note: switching to '9b076'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 9b07698 Them mot file va mot thu muc
```

main/master: là tên nhánh mặc định trong repository Git. Nó giống như "phiên bản chính" của code, nơi chứa các thay đổi ổn định hoặc hoàn chỉnh nhất. Khi tạo một repo mới bằng git init hoặc clone từ đâu đó, Git thường tự tạo nhánh main.
Trong Git, bạn có thể **rẽ nhánh** (branch) để làm việc song song, ví dụ như tạo nhánh fix-bug để sửa lỗi, trong khi vẫn giữ nhánh main nguyên vẹn và tiếp tục làm việc trên đó nếu cần.
Nhánh trong Git giống như các "phiên bản song song" của code.

Cú pháp lệnh

Tạo nhánh mới: `git branch <branch_name>`
Chuyển sang nhánh khác: `git checkout <branch_name>`

Thực hành

1. Tạo một thư mục **Example**, bên trong thư mục đó tạo một remote repo **Central_Repo.** Tạo hai local repository mới tên **Dev1 và Dev2** clone từ **Central_Repo.**

```
duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example
$ git init --bare Central_Repo
Initialized empty Git repository in D:/TaiLieuuuuuuuuuu/Git/Example/Central_Repo
/

duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example
$ git clone Central_Repo Dev1
Cloning into 'Dev1'...
warning: You appear to have cloned an empty repository.
done.

duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example
$ git clone Central_Repo Dev2
Cloning into 'Dev2'...
warning: You appear to have cloned an empty repository.
done.
```

2. Dev1 tạo một file **f1.txt,** commit

```
duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example
$ cd Dev1

duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Dev1 (master)
$ echo "Day la file 1" > f1.txt

duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Dev1 (master)
$ git add .
warning: in the working copy of 'f1.txt', LF will be replaced by CRLF the next time Git touches it

duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Dev1 (master)
$ git commit -m "Them file f1.txt"
[master (root-commit) 9edc7a1] Them file f1.txt
 1 file changed, 1 insertion(+)
 create mode 100644 f1.txt
```

2. Dev1 tạo một file **f2.txt,** commit

```
duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Dev1 (master)
$ echo "Day la file 2" > f2.txt

duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Dev1 (master)
$ git add .
warning: in the working copy of 'f2.txt', LF will be replaced by CRLF the next time Git touches it

duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Dev1 (master)
$ git commit -m "Them file f2.txt"
[master 68429e9] Them file f2.txt
 1 file changed, 1 insertion(+)
 create mode 100644 f2.txt
```

3. Dev1 push lên Central_Repo
4. Dev2 pull hai file từ Central_Repo về
5. Xem nhánh **cục bộ** hiện tại của Dev2: `git brach -l`

```
duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Dev2 (master)
$ git branch -l
* master
```

6. Dev1 tạo một nhánh mới tên **nhanh_1**: `git branch nhanh_1`

```
duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Dev1 (master)
$ git branch nhanh_1

duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Dev1 (master)
$ git branch -l
* master
  nhanh_1
```

7. Dev1 chuyển sang nhánh **nhanh_1** làm việc: `git checkout nhanh_1`

```
duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Dev1 (master)
$ git checkout nhanh_1
Switched to branch 'nhanh_1'
```

8. Bên trong **nhanh_1**, Dev1 tạo một file f_trong_nhanh_1.txt, add, commit và push

```
[nhanh_1 54f234f] Tao file dau tien trong nhanh 1
 1 file changed, 1 insertion(+)
 create mode 100644 f_trong_nhanh_1.txt

duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Dev1 (nhanh_1)
$ git push
fatal: The current branch nhanh_1 has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin nhanh_1

To have this happen automatically for branches without a tracking
upstream, see 'push.autoSetupRemote' in 'git help config'.
```

Lỗi: **nhanh_1** chưa được kết nối với bất kỳ nhánh nào trên Central_Repo. Khi chạy git push, Git không biết đẩy nhánh này đi đâu, vì đây là lần đầu tiên đẩy nhanh_1 và Git cần được chỉ rõ "đẩy lên nhánh nào trên remote".

Fix: `git push --set-upstream origin nhanh_1`

`git push`: Đẩy các commit từ nhánh hiện tại (nhanh_1) lên remote.

`--set-upstream`: Liên kết nhánh nhanh_1 cục bộ với một nhánh cùng tên (nhanh_1) trên remote origin (tức Central_Repo).

`origin`: Tên remote repo (ở đây là Central_Repo).

`nhanh_1`: Tên nhánh muốn tạo và đẩy lên origin.

```
duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Dev1 (nhanh_1)
$ git push --set-upstream origin nhanh_1
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 343 bytes | 343.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To D:/TaiLieuuuuuuuuuu/Git/Example/Central_Repo
 * [new branch]      nhanh_1 -> nhanh_1
branch 'nhanh_1' set up to track 'origin/nhanh_1'.
```

9. Dev2 pull Central_Repo về

```
duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example
$ cd Dev2

duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Dev2 (master)
$ git pull
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (3/3), 323 bytes | 46.00 KiB/s, done.
From D:/TaiLieuuuuuuuuuu/Git/Example/Central_Repo
 * [new branch]      nhanh_1    -> origin/nhanh_1
Already up to date.

duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Dev2 (master)
$ git branch -l
* master

duong@Duong MINGW64 /d/TaiLieuuuuuuuuuu/Git/Example/Dev2 (master)
$ git branch -r
  origin/HEAD -> origin/master
  origin/master
  origin/nhanh_1
```

origin/HEAD → origin/master:

- origin/HEAD là con trỏ trên remote repo (Central_Repo) chỉ đến **nhánh mặc định** của repo.
- Ở đây, nó trỏ tới origin/master, nghĩa là master là nhánh mặc định của Central_Repo. Khi clone repo, Git tự động checkout nhánh này (vì thế Dev2 đang ở master).
- Không có gì bất ngờ, chỉ là cách Git đánh dấu nhánh chính.

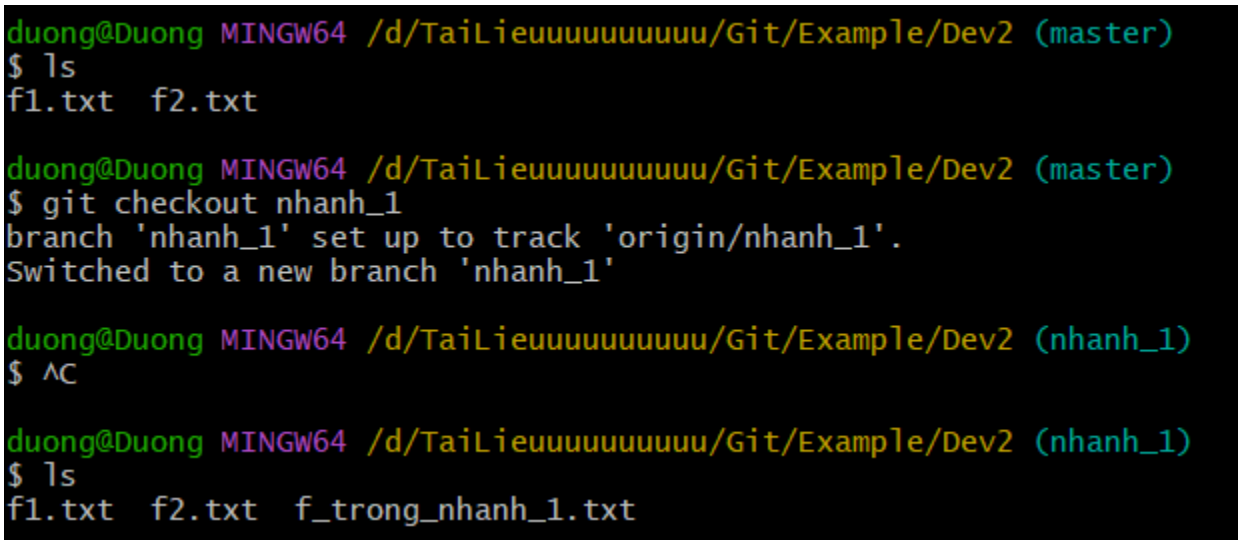
origin/master:

- Đây là nhánh master trên Central_Repo. Nó tồn tại từ khi repo được tạo và là nhánh mặc định mà Dev2 đang theo dõi (tracking) trong nhánh cục bộ master.

origin/nhanh_1:

- Đây là nhánh nhanh_1 trên Central_Repo, do bạn đẩy lên từ Dev1 trước đó (bằng git push --set-upstream origin nhanh_1).
- Nó tồn tại trên remote, nhưng **chưa có nhánh cục bộ tương ứng** trong Dev2. Vì thế, khi bạn chạy git branch -l trước đó, chỉ thấy master.

10. Dev2 kiểm tra các file có trong từng nhánh



KẾT HỢP NỘI DUNG TỪ CÁC NHÁNH

Lệnh **git merge** dùng để **hợp nhất** một nhánh khác vào nhánh hiện tại.
Cú pháp: `git merge <tên_nhánh>`

- `<tên_nhánh>` là nhánh bạn muốn merge vào nhánh hiện tại.
- Ví dụ: Nếu đang ở **nhanh_1** và chạy **git merge master**, Git sẽ lấy các thay đổi từ master và hợp nhất vào nhanh_1.

Với một nhánh B được rẽ ra từ nhánh A, thực hiện lệnh **git rebase** trong nhánh B sẽ nối nhánh B vào commit cuối cùng của nhánh A.
Cú pháp:
`git rebase <tên nhánh>`: Chuyển các commit của nhánh hiện tại (ví dụ, nhanh_1) lên trên nhánh được chỉ định (<tên_nhánh>, ví dụ master), làm lịch sử commit tuyến tính. Nếu có xung đột (conflict), Git tạm dừng và yêu cầu giải quyết (xem phần --continue và --skip).
Cách hoạt động:

- Git tìm điểm chung giữa nhánh hiện tại (nhanh_1) và nhánh cơ sở (master).
- Lưu các commit riêng của nhanh_1 (như commit tạo f_trong_nhanh_1.txt).
- Áp dụng các commit của master làm nền mới.
- Áp dụng lại các commit của nhanh_1 lên trên.

`git rebase --continue`: tiếp tục quá trình rebase sau khi giải quyết xung đột (conflict).
Trong lúc rebase (ví dụ, git rebase master), nếu nhanh_1 và master chỉnh sửa cùng file ở cùng chỗ (như f_trong_nhanh_1.txt), Git dừng lại và báo xung đột.
Bạn phải:

1. Sửa file thủ công để giải quyết xung đột.
2. Đánh dấu file đã sửa bằng:
   `git add <file>`
3. Chạy:
   `git rebase –ccontinue`
   để Git tiếp tục áp dụng các commit còn lại của nhanh_1.

`git rebase --skip`: Bỏ qua (skip) một commit đang gây xung đột trong quá trình rebase. Thay vì giải quyết, có thể chọn **bỏ qua commit** đang được áp dụng (thường là commit từ nhánh hiện tại, như nhanh_1).
***Lưu ý**: Cẩn thận với --skip, vì nó **xóa commit vĩnh viễn** trong lịch sử rebase. Nếu nhầm, bạn sẽ mất thay đổi (trừ khi dùng git reflog để khôi phục).*

XÓA NHÁNH - HỦY BỎ COMMIT – QUAY LẠI COMMIT MÀ KHÔNG HỦY COMMIT MỚI

Xóa một nhánh cục bộ trong repo: `git branch -d <tên nhánh>`
Điều kiện:

- Nhánh muốn xóa **phải đã được merge** vào nhánh khác (thường là nhánh chính như master) hoặc không có commit chưa merge. Nếu không, Git sẽ báo lỗi để bảo vệ dữ liệu.
- Bạn **không thể xóa nhánh đang đứng** (ví dụ, nếu bạn ở nhanh_1, không xóa được nhanh_1 trừ khi chuyển sang nhánh khác).

Reset
`git reset --soft <commit_id>`: Di chuyển HEAD về vị trí commit. Trạng thái của stage và tất cả sự thay đổi của file được giữ nguyên

- **Di chuyển HEAD về vị trí commit**:
  - Lệnh git reset --soft <commit_id> đưa **HEAD** của nhánh hiện tại (ví dụ, nhanh_1) về commit được chỉ định (<commit_id>).
  - Các commit sau <commit_id> bị xóa khỏi lịch sử nhánh, nhưng vẫn truy cập được qua git reflog.
- **Trạng thái của stage và tất cả sự thay đổi của file được giữ nguyên**:
  - **Working directory**: Tất cả thay đổi của file (như f_trong_nhanh_1.txt) vẫn y nguyên, không bị xóa.
  - **Staging area**: Các thay đổi từ các commit bị xóa được đưa vào trạng thái **staged** (như thể vừa chạy git add cho chúng).

`git reset (--mixed) <commit_id>:` Di chuyển HEAD về vị trí commit reset, giữ tất cả thay đổi của file nhưng loại bỏ các thay đổi stage

Di chuyển HEAD:

- HEAD của nhánh hiện tại (như nhanh_1) trỏ về <commit_id>.
- Các commit sau <commit_id> bị xóa khỏi lịch sử nhánh.

Giữ working directory:

- Tất cả thay đổi từ các commit bị xóa (như thêm/sửa file) vẫn tồn tại trong thư mục làm việc (f_trong_nhanh_1.txt không mất).

Xóa staging area:

- Staging area bị làm trống (khác với --soft, nơi thay đổi được staged).
- Các thay đổi từ commit bị xóa trở thành **unstaged** (hiển thị trong git status dưới "Changes not staged for commit").

`git reset –hard <commit_id>:` Di chuyển con trỏ HEAD về vị trí commit reset và loại bỏ tất cả sự thay đổi của file

Revert
Tạo một commit mới để **hoàn tác (undo)** các thay đổi của một commit cụ thể (<commit_id>), mà không xóa lịch sử commit cũ.
Cú pháp: git revert <commit_id>