Họ và tên: Nguyễn Đức Toàn
MSV: B23DCCN831

Week 1: Git

Giới thiệu tổng quan về Git

1. Lịch sử về Git

Git là phần mềm quản lí mã nguồn phân tán. Ban đầu git được Linus Torvalds tạo ra vào năm 2005 để quản lí mã nguồn cho việc phát triển nhân Linux. Nhưng hiện nay git đã trở thành một trong những phần mềm quản lí mã nguồn phổ biến nhất.

Git là một hệ thống quản lý phiên bản phân tán (Distributed Version Control System - DVCS) được sử dụng phổ biến nhất hiện nay nhờ khả năng làm việc linh hoạt, tốc độ cao và hỗ trợ mạnh mẽ cho việc phát triển phần mềm theo nhóm. So với Git, SVN (Subversion) là hệ thống quản lý phiên bản tập trung (Centralized VCS), trong đó toàn bộ mã nguồn và lịch sử thay đổi được lưu trên một máy chủ trung tâm. Điều này khiến SVN dễ quản lý nhưng phụ thuộc hoàn toàn vào máy chủ, đồng thời không thể thực hiện commit hay theo dõi thay đổi khi làm việc offline. Trong khi đó, Git cho phép mỗi lập trình viên có một bản sao đầy đủ của toàn bộ repository, có thể thực hiện commit, tạo nhánh hoặc hợp nhất (merge) ngay cả khi không có kết nối mạng, và chỉ cần đồng bộ lại khi có Internet. So với Mercurial (Hg) – một hệ thống phân tán tương tự Git – cả hai đều có tốc độ nhanh, khả năng làm việc offline và cơ chế branch hiệu quả. Tuy nhiên, Git nổi bật hơn nhờ cộng đồng lớn, tài liệu phong phú và được hỗ trợ mạnh mẽ bởi các nền tảng như GitHub, GitLab hay Bitbucket. Mercurial có giao diện thân thiện và dễ học hơn, nhưng ít được sử dụng rộng rãi trong các dự án mã nguồn mở cũng như môi trường doanh nghiệp.

Github/GitLab/BitBucket là các nền tảng dịch vụ giúp lưu trữ, chia sẻ repository Git trên máy chủ, chúng trên trên server

2. Cài dặt và cấu hình git

a. Kiểm tra cấu hình

git config --list : Liệt kê ra tất cả các config đang áp dụng

git config --list --show-scope : Liệt kê ra tất cả các config đang áp dụng và kèm thêm phạm vi giá trị được lấy từ đâu

git config --global user.name “Your name” : Đặt tên người dùng sẽ được hiển thị trong các commit

git config --global user.email “Your email” : Đặt email lên kết với commit(là email để login github)

- Kiểm tra version của git

git --version

b. Khái niệm về **SSH key** :

Là một cặp khóa bảo mật dùng để xác thực danh tính người dùng khi kết nối giữa máy tính cá nhân và máy chủ Git qua giao thức SSH

SHH key xác minh danh tính người dùng mà không cần nhập mật khẩu Github mỗi lần

Thiết lập SSH key:

ssh-keygen -t ed25519 -C "Your email" : tạo ra cặp khóa SSH sử dụng thuật toán ED25519

Nếu lỗi thì sử dụng : **ssh-keygen -t ecdsa-sk -C "your_email@example.com"**

cat ~/.ssh/id_ed25519.pub : để mở ra nội dung của file id_ed25519.pub
Sau đó copy toàn bộ nội dung của file dán và SSH key trên github
ssh -T git@github.com : Kiểm tra kết nối

Làm việc với Repository

a. Khởi tạo và clone repository

git init :Dùng để khởi tạo một repository Git mới trong thư mục hiện tại. Git sẽ tạo **.git/** trong thư mục được khởi tạo nên thư mục đó đã trở thành một local repository

git clone: Dùng để sao chép một repository có sẵn trên github về máy tính cá nhân. Git sẽ tạo ra một thư mục và có sẵn **.git/**, toàn bộ mã nguồn, lịch sử commit

b. Cấu trúc thư mục .git gồm: HEAD, index, objects, config, hooks, refs

HEAD: tệp này là con trỏ tới commit hiện tại trên nhánh đang làm việc

object: thư mục này chứa các đối tượng dữ liệu của Git bao gồm các commit, cây blob(nội dung tệp) và các tag

config: tệp này chứa thông tin cấu hình cho kho lưu trữ cục bộ, như tên người dùng và email

hooks: một thư mục chứa các lệnh được kích hoạt tự dồng bởi các sự kiện khác nhau của Git

refs: Thư mục này chứa các tham chiếu đến các commit, chẳng hạn như tag, branch

c. Trạng thái file trong git

Untracked: File mới được thêm vào, chưa có trong hệ thống theo dõi của Git
Unmodified: File đã được git theo dõi nhưng chưa có bất kì thay đổi nào được thực hiện
Modified: File đã được git theo dõi và có những thay đổi nhưng chưa được chuẩn bị để commit
Staged: File đã được chuẩn bị sẵn sàng để đưa vào commit tiếp theo

d. Thêm và commit thay đổi

git add tên_file : thêm tên file vào vùng staging, lưu vào .git/index
git add . : thêm toàn bộ file thay đổi, lưu vào .git/index
git commit -m"message" : ghi nội dung trong staging và commit mới .git/objects

- Cấu trúc chuẩn của một message commit chuẩn

<type>: <short summary>
<body>
<footer>

- phần đầu

Ngắn gọn, không viết hoa tùy tiện, không chấm cuối dòng
Một số type phổ biến:
feat: thêm một feature
fix: fix bug cho hệ thống
refactor: Sửa code nhưng không fix bug cũng không thêm feature
docs: thêm/ thay đổi document
chore: những thay đổi nhỏ nhặt không liên quan đến code
style: những thay đổi không thay đổi ý nghĩa của code như thay đổi UI, css
perf: code cải tiến về mặt hiệu năng xử lý

vendor: cập nhập version cho các dependencies, packages
build : thay đổi liên quan đến build hoặc CI/CD
- phần body: giải thích lý do thay đổi
- phần footer: gắn issue ID, ghi chú về breaking changes

Làm việc với lịch sử và phiên bản

a. Xem lịch sử

git log : hiển thị lịch sử commit trong repository gồm: mã commit, tên tác giả , ngày commit, nội dung message commit

Một số tùy chọn:

git log --oneline: hiển thị mỗi commit trên 1 dòng
git log --graph: hiển thị danh sách dạng cây
git log --stat: hiển thị file nào đc sửa và số dòng thêm/bớt
git log -p: Hiển thị chi tiết nội dung thay đổi trong từng commit
git log --author="ten" : lọc commit theo tên
git log --since="2 week ago" : hiện thị commit 2 tuần trước
git log --grep="keyword": tìm commit chứa từ khóa
git log branch1..branch2 : so sánh commit khác nhau giữa hai nhánh
git log -n 5: hiển thị 5 commit phần nhất

git show <commit_id>: Hiển thị chi tiết commit

Một số cú pháp :

git show HEAD: hiển thị commit mới nhất
git show HEAD~1: hiển thị commit trước đó
git show --name-only <commit_id> : chỉ hiển thị danh sách file bị thay đổi

git blame <file> : Dùng để xác định ai chỉnh sửa file, dòng đó ai commit, thời gian chỉnh sửa

git blame -L 5, 6 <file> : chỉ xe dòng từ 5 đến 6

- Commit Id, hay còn gọi là SHA-1 hash, là một mã định danh duy nhất gồm 40 kí tự thập lục phân, được tạo ra cho mỗi commit trong kho lưu trữ Git
Mỗi commit sẽ có một commit id riêng biệt. Nếu hai commit có cùng nội dung chúng sẽ có cùng một commit id. Git sử dụng commit Id để kiểm tra xem dữ liệu có bị thay đổi hay giả mạo không. Chỉ cần một bit thay đổi trong commit, commit Id cũng sẽ hoàn toàn khác. Commit Id dùng để tham chiếu đến một commit cụ thể khi thực hiện các lệnh git, như quay lại một phiên bản cũ hoặc kiểm tra lịch sử thay đổi

- Git checkout, git restore, git reset, git revert đều được dùng để hoàn tác, nhưng giữa chúng các những điểm khác nhau như: git checkout để chuyển đổi giữa các nhánh và commit, git restore dùng để hoàn tác thay đổi trong cây làm việc, git reset di chuyển con trỏ HEAD đến một commit khác và có thể xóa hoặc đánh dấu commit, git revert tạo một commit mới để đảo ngược một commit trước đó
- **git reset <commit_id>** : git reset có hai tùy chọn là git reset --hard và git reset --soft
+/ Với **--hard** : sẽ di chuyển con trỏ HEAD đến commit đã chỉ định và đặt lại trạng thái làm việc(working directory) để trạng thái quay về trạng thái đã được chỉ định. Được dùng khi muốn xóa các thay đổi chưa được commit trong trạng thái làm việc

+/ Với **--soft**: sẽ di chuyển con trỏ HEAD đến commit đã chỉ định, nhưng không thay đổi trạng thái làm việc. Được dùng để duy trì các thay đổi chưa được commit trong trạng thái làm việc và chỉ đặt lại chỉ số
- **git revert <commit_id>**: Được sử dụng đẻ tạo một commit mới, loại bỏ các thay đổi đã được thực hiện trong một hoặc nhiều commit trước đó , và áp dụng các thay đổi và nhánh hiện tại
- Khi nào nên dùng **git revert**, **git reset** ?

Sử dụng git reset khi muốn xóa các commit và thay đổi liên quan khỏi lịch sử commit và không quan tâm đến lịch sử commit ban đầu

Sử dụng git revert khi muốn hoàn tác các thay đổi của một hoặc nhiều commit trong lịch sử commit và duy trì lịch sử commit chính xác của dự án để mọi người tiện theo dõi
- **.gitignore**: là một file đơn giản nằm trong thư mục gốc của dự án Git, dùng để chỉ các file hoặc các folder mà muốn git không theo dõi. Lợi ích của file .gitignore là giảm dung lượng file lưu trữ trên git, tránh lưu trữ những thông tin nhạy cảm, quá trình làm việc nhóm dễ dàng hơn

Nhánh và hợp nhất

- **git branch**: là công cụ cho phép tách rẽ quá trình phát triển một dự án thành nhiều dòng thời gian khác nhau. Mỗi nhánh là một con đường phát triển riêng biệt
Branch có thể phân loại theo 2 tiêu chí : Theo chức năng và theo vòng đời

- **git checkout**: là một lệnh đa năng trong Git dùng để chuyển đổi giữa các nhánh, tạo nhánh mới, hoặc khôi phục các tệp trong thư mục về trạng thái commit trước đó
Một số cú pháp hay dùng:

git checkout <tên-nhánh>: Chuyển đến một nhánh chỉ định, cập nhập thư mục làm việc và HEAD để phản ánh trạng thái của nhánh đó

git checkout -b <tên-nhánh-mới>: Tạo 1 nhánh mới và nhảy sang nhánh đó làm việc

git checkout -- <tên-tep>: Hủy bỏ thay đổi trong một tệp cụ thể từ vùng làm việc, quay lại phiên bản mới nhất của tệp đó trong nhánh hiện tại

git checkout <commit-hash> <ten-tẹp> : Khôi phục một tệp về trạng thái của nó tại một commit cụ thể, được xác định qua commit-hash

git checkout <commit-hash>: Di chuyển HEAD đến một commit cụ thể trong lịch sử, cho phép xem lại dự án tại thời điểm đó

Chú ý: Thường sử dụng git switch để chuyển nhánh để tránh gây nhầm lẫn khi git restore được dùng cho việc khôi phục tệp
- **git switch**: là một cú pháp được thiết kế để chuyển đổi hoặc tạo các nhánh một cách an toàn hơn git checkout

Một số cú pháp chuyển đổi:

git switch <ten-nhanh> : chuyển sang một nhánh đã tồn tại

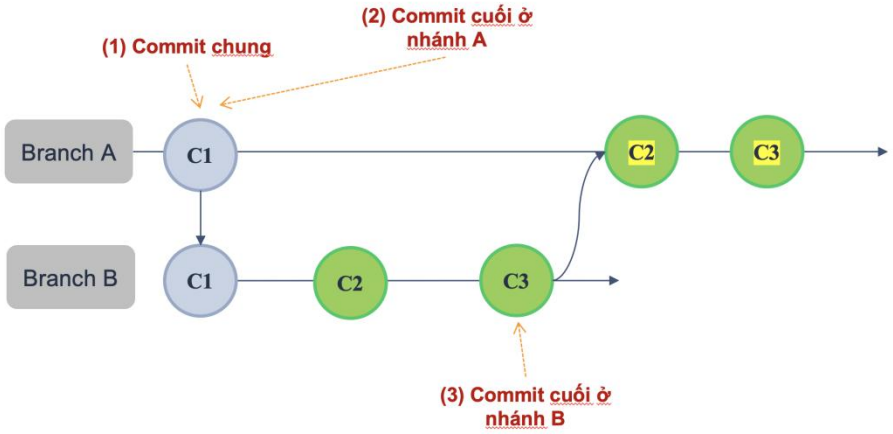
git switch -c <tên-nhanh-moi>: tạo 1 nhánh mới và chuyển sang nhánh mới luôn

git switch - : chuyển sang nhánh trước đó vừa làm việc

- Merger branch

Git merge là lệnh hợp nhất các thay đổi từ một nhánh nào đó vào nhánh hiện tại
Phân loại git merge

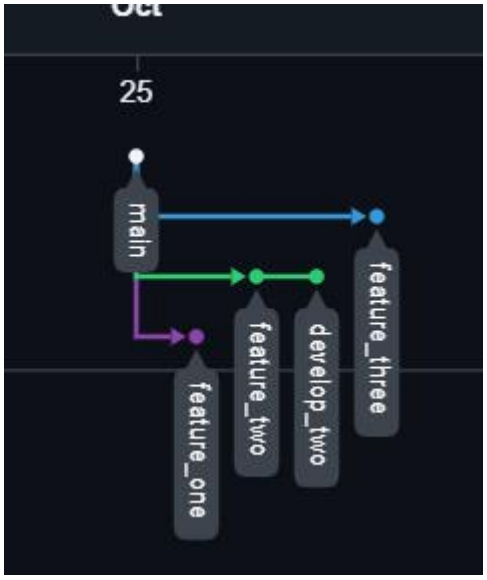
- Fast-forward merge



Trong trường hợp khi thực hiện lệnh git merge từ nhánh B vào nhánh A, nhưng trước đó nhánh A không có thay đổi gì kể từ khi nhánh B được tách ra (commit cuối cùng của nhánh A chính là commit chung với nhánh B), thì lúc này Git sẽ thực hiện Fast-forward merge. Git không tạo merge commit, mà chỉ đơn giản di chuyển con trỏ HEAD của nhánh A đến vị trí của commit cuối cùng của nhánh B

Ví dụ:

Trạng thái hiện tại của các nhánh:



Bước 1 : Thực hiện tạo 1 nhánh mới từ main:

Git checkout -b”feature/login”

Lúc này con trỏ HEAD được nhảy đến nhánh feature/login

Bước 2: Thực hiện add và commit những thay đổi trong nhánh feature/login và quay về nhánh main

Git add .

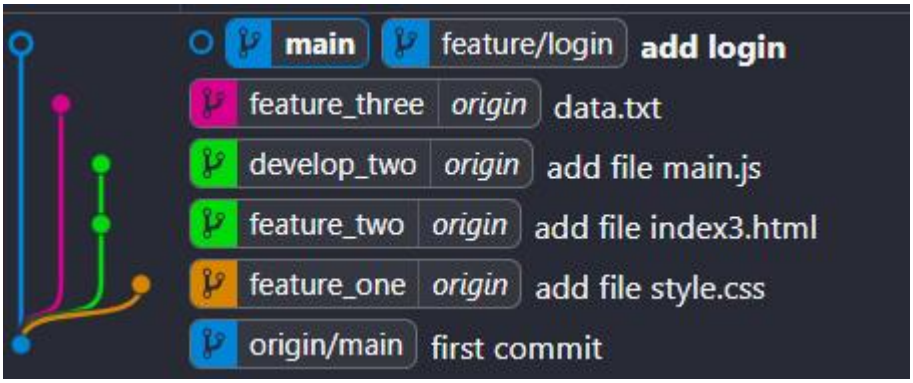
Git commit -m”add login.html”

Git switch main

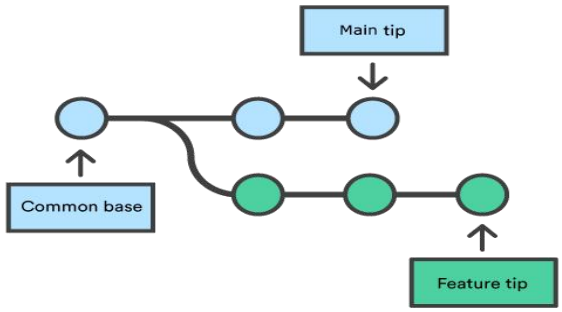
Bước 3: Thược hiện merge nhánh feature/login với nhánh main

git merge feature/login

Kết quả:



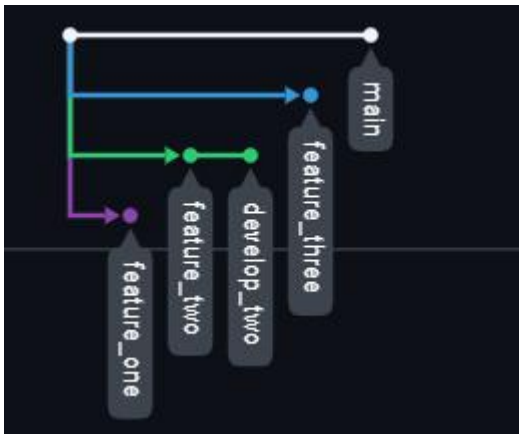
- **Three-way merge**



Là cách git gộp hai nhánh lại khi cả hai đều có sự thay đổi kể ừ lúc tách ra. Git xem nhánh main, nhánh feature có thay đổi gì so với commit chung. Sau đó git sẽ merge cả hai sự thay đổi và tạo ra merge commit mới

Ví dụ:

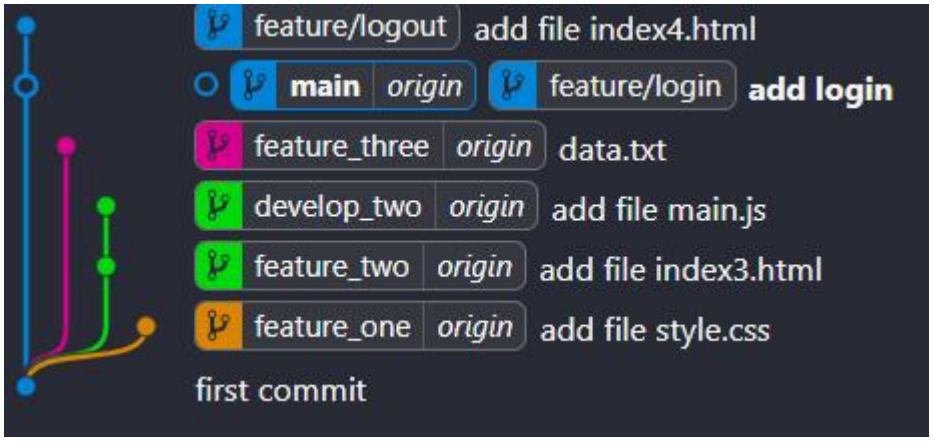
Trạng thái hiện tại của nhánh trong git



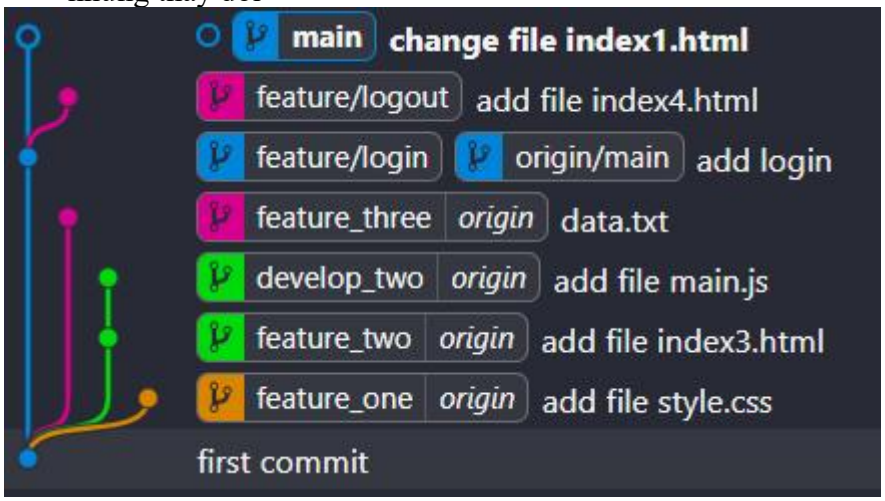
Bước 1: từ nhánh main tạo 1 nhánh mới feature/logout

Git checkout -b “feature/logout”

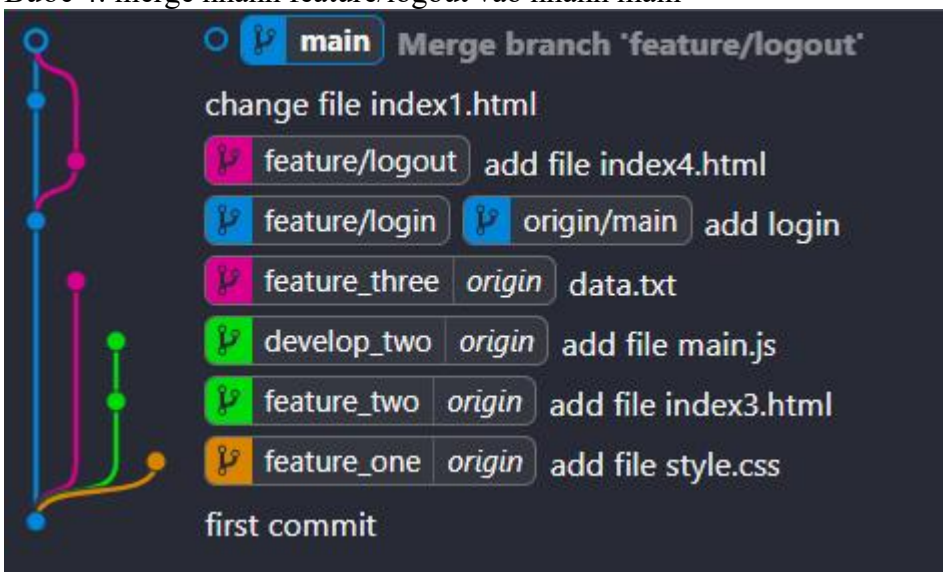
Bước 2: Thêm 1 file index4.html thêm nội dung vào file, add và commit lại những thay đổi



Bước 3: Quay trở lại nhánh main chỉnh sửa lại file index1.html, add và commit lại những thay đổi



Bước 4: merge nhánh feature/logout vào nhánh main



Merge conflict là khi merge code, git không biết merge code thế nào, do có nhiều sự thay đổi trên cùng một file

Bước 1: mở file chứa conflict

Bước 2: Sửa tên file trên, quyết định nội dung cần giữ/xóa

Bước 3: Xóa các conflict markers (HEAD, ===)

Bước 4: Commit lại sự thay đổi

- Chiến lược branching phổ biến

Gitflow là một chiến lược phổ biến được giới thiệu bởi Vincent Dríessen. Chiến lược này là cách tổ chức và quản lí các nhánh trong Git để nhiều người cùng làm việc trên một dự án mà không có xung đột, và dễ dành triển khai code

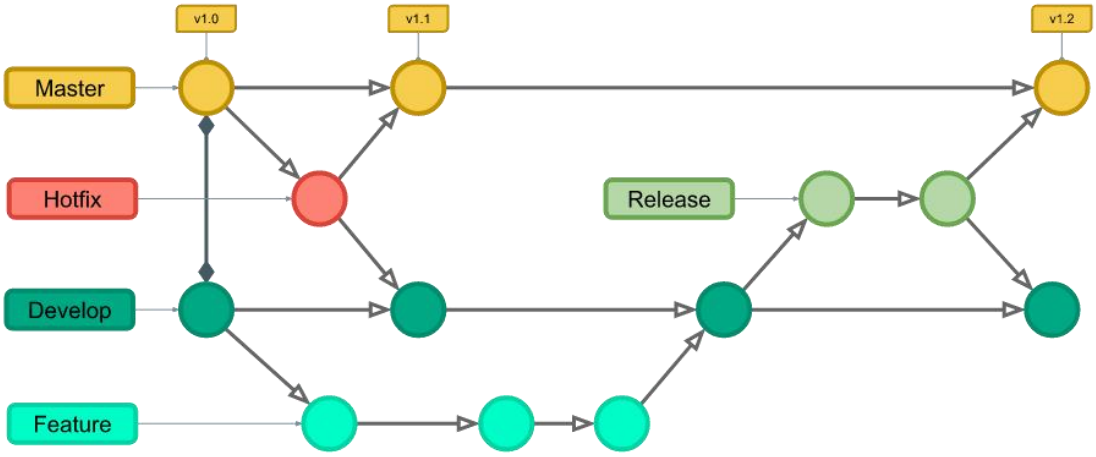
Nhánh main: là nhánh chứa code sẵn sàng phát hành(production-ready code). nhánh main được tạo ra ngay từ khi bắt đầu dự án và được duy trì xuyên suốt quá trình phát triển. Các commit trong nhánh này có thể được đánh tag để biểu thị các phiên bản hoặc các bản phát hành khác . các nhánh khác như release hoặc hostfix sẽ chỉ được merge và main khi đã kiểm thử và xác nhận ổn định

Nhánh develop: là nhánh được tạo ngay từ đầu dự án và được duy trì liên tục trong quá trình phát triển. Nó chứa các tính năng mới đang được phát triển và kiểm thử. Khi tạo một nhánh mới, sẽ tạo các nhánh feature/* dựa trên develop, sau khi hoàn thành và kiểm tra, các nhánh feature sẽ được merge trở lại develop để chuẩn bị cho quá trình test hoặc release

Nhánh feature: là loại nhánh dùng để thêm một tính năng mới vào dự án

Nhánh release: là nhánh được dùng để chuẩn bị bản phát hành mới. Thông thường các công việc trên nhánh này là chỉnh sửa nhỏ, fix bug nhẹ, hoặc hành thiện chi tiết cuối cùng để sẵn sàng phát hành. Nhánh này giúp tách biệt phần code chuẩn bị phát hành khỏi nhánh develop

Nhánh Hotfix: là nhánh được dùng để xử lí nhanh các lỗi khẩn cấp phát sinh trên nhánh main, base của nhánh hotfix là nhánh main, sau khi fix bug xong nhánh này sẽ được merge vào nhánh main và develop



Remote repository

a. Thêm remote và đẩy code

git remote add <tên-remote> <url> : thêm một kho lưu trữ từ xa mới

git push : đẩy các commit đã thực hiện trên nhánh cục bộ lên kho lưu trữ từ xa

Lưu ý:

1. Cần kiểm tra trạng thái trước khi push git status đảm bảo các commit tất cả thay đổi
2. Nếu người khác đã push lên cùng nhánh thì phải pull về trước khi push
3. Kiểm tra repository đang liên kết với cái gì

Một số cú pháp liên quan:

git push <remote-name> <branch-name>
git push -u <remote-name> <branch-name> : Lần đầu tiên thiết lập tracking giữa local và remote, các lần push sau chỉ cần git push

git pull : kéo các thay đổi mới nhất từ kho lưu trữ từ xa và hợp nhất chúng vào nhánh cục bộ hiện tại(git pull = git fetch + git merge)
Một số cú pháp liên quan:
git pull <remote-name> <branch-name>

git fetch : tải về các thay đổi mới nhất từ kho lưu trữ từ xa nhưng chưa hợp nhất chúng với nhánh cục bộ. Có thể xem các thay đổi trước khi hợp nhất sử dụng git log, git diff

b. Folk, clone, pull request trong làm việc nhóm

Folk được dùng để sao chép toàn bộ repo của người khác về github của mình. Không có quyền push trực tiếp lên repo gốc chỉ được chỉnh sửa mà không ảnh hưởng đến repo gốc

Clone được dùng để tải toàn bộ repo từ github về máy tính local

Cú pháp git clone <url>

Pull request: Yêu cầu chủ repo gốc xem xét và merge code của của bạn vào máy tính

Các quy trình làm việc nhóm:

Bước 1 : mỗi thành viên cần folk về tài khoản github của mình

Bước 2: clone về máy tính local

Bước 3: tạo nhánh mới tách khỏi nhánh main

Bước 4: push lên github cá nhân

Bước 5: thực hiện pull request

- Giải quyết conflict khi làm việc nhóm

- Giả sử tình huống thực thế có nhánh main
- Tại nhánh main có 1 file index.html
- Thực hiện commit lại và tạo 1 nhánh mới từ nhánh main có tên là feature
- Tại nhánh này cần tại một file mới (ví dụ main.js) và thực hiện chỉnh sửa trong file index.html. Commit lại và switch về nhánh main
- Tại nhánh main chỉnh sửa file index.html sau đó tại thực hiện merge nhánh feature với nhánh main. Lúc này xuất hiện conflict
- Thực hiện xóa và giữ lại những code cần thiết sau đó commit lại

Công cụ và kĩ năng nâng cao

a. Tag và versioning

Tag dùng để tham chiếu tới một commit thường dùng để đánh dấu phiên bản phát hành. Tag giống như một nhánh nhưng không thay đổi. Khác với nhánh, tag không có lịch sử khi được tạo. Khi tag được tạo không thể thêm commit mới vào nó, nhưng có thể dễ dàng tham chiếu bằng tag

Có hai loại tag chính trong Git là annotated tag và lightweight tag

Annotated tag được lưu trữ dưới dạng các đối tượng đầy đủ trong db của git. Nó bao gồm các metadata bổ sung như người tạo, ngày tạo, mô tả chi tiết. Annotated được khuyến nghị sử dụng cho việc đánh dấu các điểm quan trọng như phiên bản release

Các cú pháp thường dùng với tag
git tag -a <tên-tag> : tạo một tag mới
git tag -a <tên-tag> -m “message”: tạo một tag mới kèm thêm nội dung mô tả
git show <ten_tag> : xem thông tin của một tag
Git tag -d <tên_tag> : xóa tag
Ví dụ:

```
PS D:\git test> git tag -a v1.1.0 -m "đây là phiên bản mới phát hành"
PS D:\git test> git show v1.1.0
tag v1.1.0
Tagger: ToanND <toannguyencnptit2k5@gmail.com>
Date:   Sat Oct 25 22:09:51 2025 +0700

đây là phiên bản mới phát hành
```

Lightweight tag: là một tham chiếu trực tiếp đến commit mà không cần bất kì metadata nào. Phù hợp cho các commit mang tính tạm thời, đơn giản . Lightweight tag không sử dụng kèm với các tùy chọn -a, -m
Cú pháp:
git tag <tên-tag> : tạo một tag mới

b. Git describe
Dùng để xác định phiên bản hiện tại của dự án dựa trên các tag và commit, giúp tạo ra chuỗi phiên bản, lịch sử phát hành và hiểu rõ bối cảnh của một commit cụ thể
Cú pháp:
git describe : mô tả commit hiện tại
git describe HEAD~3 : mô tả commit cánh HEAD 3 bước
git describe 1a2b3c : mô tả commit có hash cụ thể
Ví dụ

```
PS D:\git test> git describe
v1.1.0
```

c. Stash

Git stash là một cơ chế tạm thời lưu lại các thay đổi chưa commit trong workspace directory và staged changes. Dễ hiểu hơn git stash là “nút tạm dừng” cho công việc đang dang dở, giúp Dev linh hoạt giữa các task không lo bị mất commit
Lí do để dùng git stash: để có một workspace sạch sẽ, dễ apply, pop. Commit tạm thời lịch sử commit bị “bẩn”. Git reset --hard sẽ bị mất code commit
Git stash được dùng khi nào?
Khi đang code dở, nhưng cần quay lại code product hoặc switch branch để fix bug hotfix
Không muốn mất công việc viện tại
Muốn workspace sạch sẽ mà không commit những thay đổi chưa hoàn chỉnh
Một số cú pháp :
Git stash hoặc git stash save: Lưu tạm thay đổi của bạn và stash (giống như git reset --hard nhưng lại thay đổi lịch sử)
git stash save “message”: Lưu stash kèm message

git stash list: Liệt kê tất cả các stash đã lưu
git stash show: xem nhanh các thay đổi gần đây (show -p để xem chi tiết)
git stash apply: để áp dụng vào stash gần nhất vào workspace nhưng không xóa stash
git stash pop: áp dụng stash gần nhất và xóa nó khỏi stash list(apply + xóa)

```
PS D:\git test> git stash save "đang sửa dở file index4.html"
Saved working directory and index state On main: đang sửa dở file index4.html
PS D:\git test> git stash list
stash@{0}: On main: đang sửa dở file index4.html
PS D:\git test> git stash show
 index4.html | 1 +
 1 file changed, 1 insertion(+)
PS D:\git test>
```

Rebase và Squash

Git rebase là để di chuyển một nhánh lên commit mới nhất của một nhánh khác , tạo lịch sử thẳng, gọn hơn merge. Ưu điểm là commit sạch, dễ đọc. Nhược điểm là không nên rebase những commit đã push, dễ gây conflict với team

Cú pháp:
git rebase <tên_nhánh>

Git Squash là gom nhiều commit nhỏ thành 1 commit duy nhất. Ưu điểm là làm sạch lịch sử commit
Cú pháp:
git rebase -I HEAD~3

Git alias & log formatting
Git alias là tạo tên viết tắt cho Git dài hoặc thường dùng để không phải gõ nhiều lần
Cú pháp:
Git config --global alias.<ten-alias> “<lenh git goc”
Ví dụ như st -> status, co -> checkout, br -> branch, cm -> commit, lg -> log
Git log formatting là sử dụng tùy chọn --pretty=format: để hiển thị commit theo định dạng mà bạn mong muốn. Mục đích để xem các thông tin commit quan trọng, hiển thì lịch sử commit ngắn gọn và đẹp, kết hợp với alias để tại log tiện hơn, nhanh cho workflow hằng ngày
Cú pháp:
git log --pretty=format:”<format>”
Một số placeholder hay dùng:
%h : hash rút gọn commit
%H : full has commit
%an: tên tác giả
%ae: tên email
%s : commit message

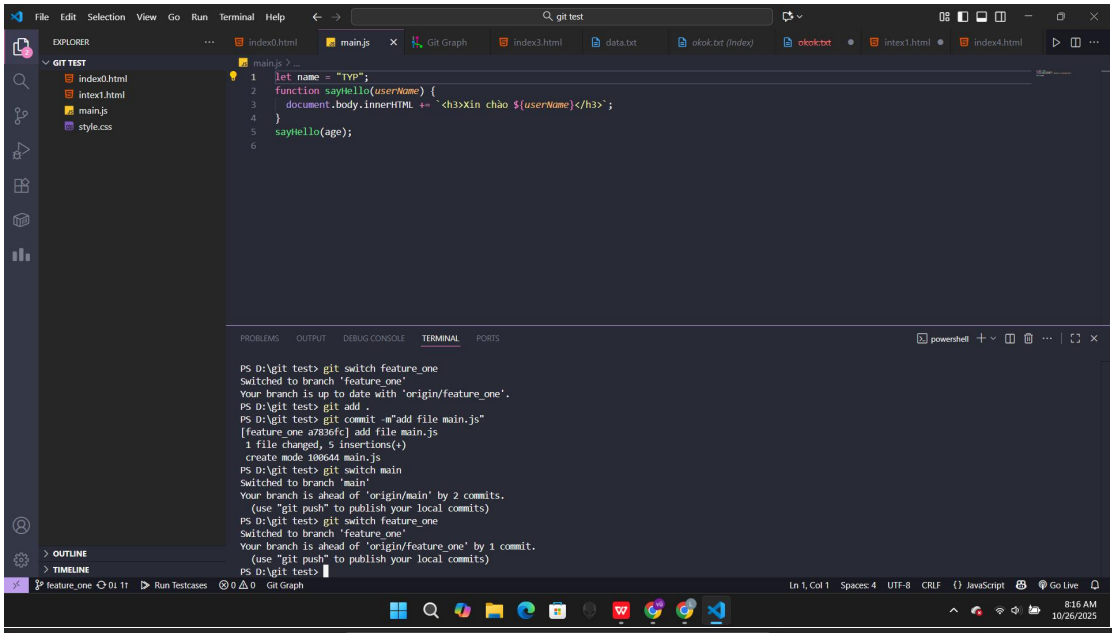
Reflog
Git reflog là để ghi lại mọi thay đổi của HEAD trong repo kể cả commit, checkout, reset, rebase….Mục đích là theo dõi và khôi phục các commit”mất tích” mà git log bình thường không hiển thị. Coi reflog giống như lịch sử di chuyển HEAD

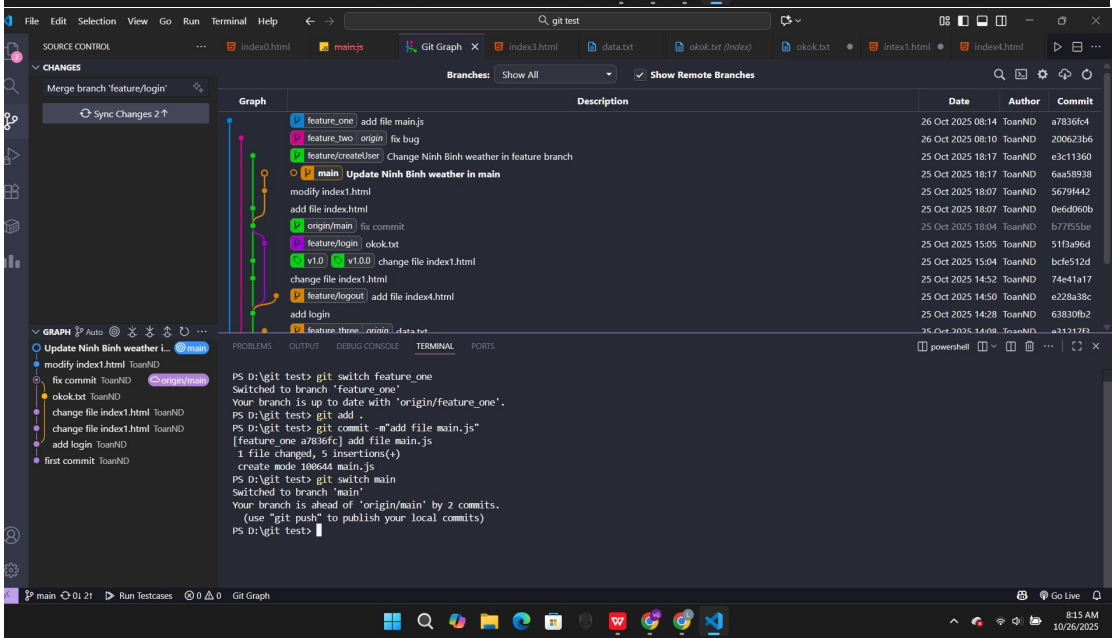
Cú pháp:
Git reflog

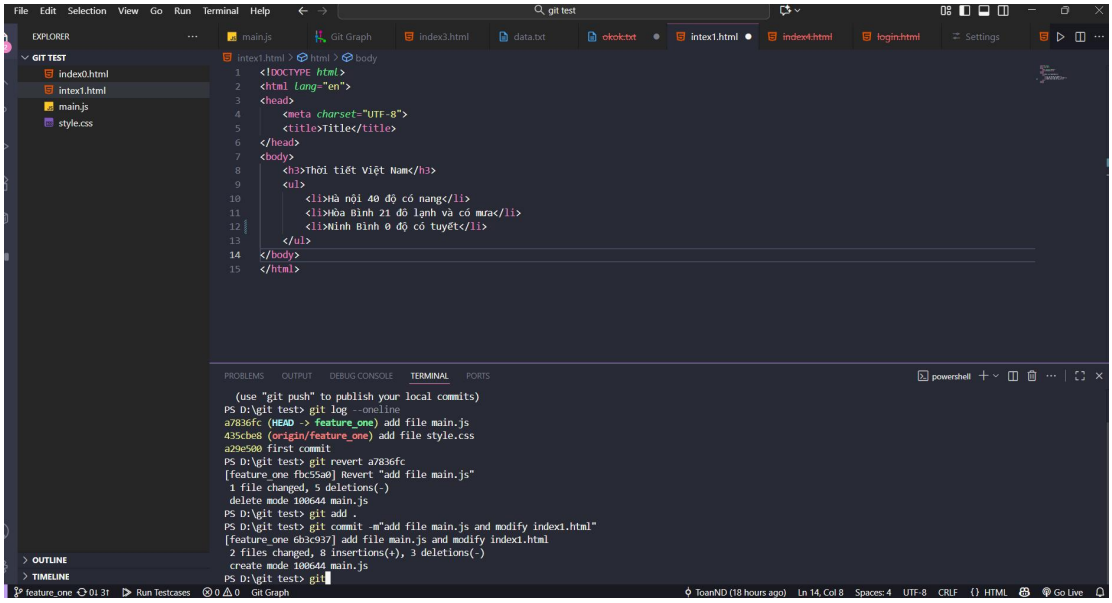
Thực hành

A. Merge conflict

Chuyển sang nhánh feature_one, tạo 1 file main.js và chỉnh sửa file index1.html, sau đó add và commit lại những thay đổi và trở về nhánh main



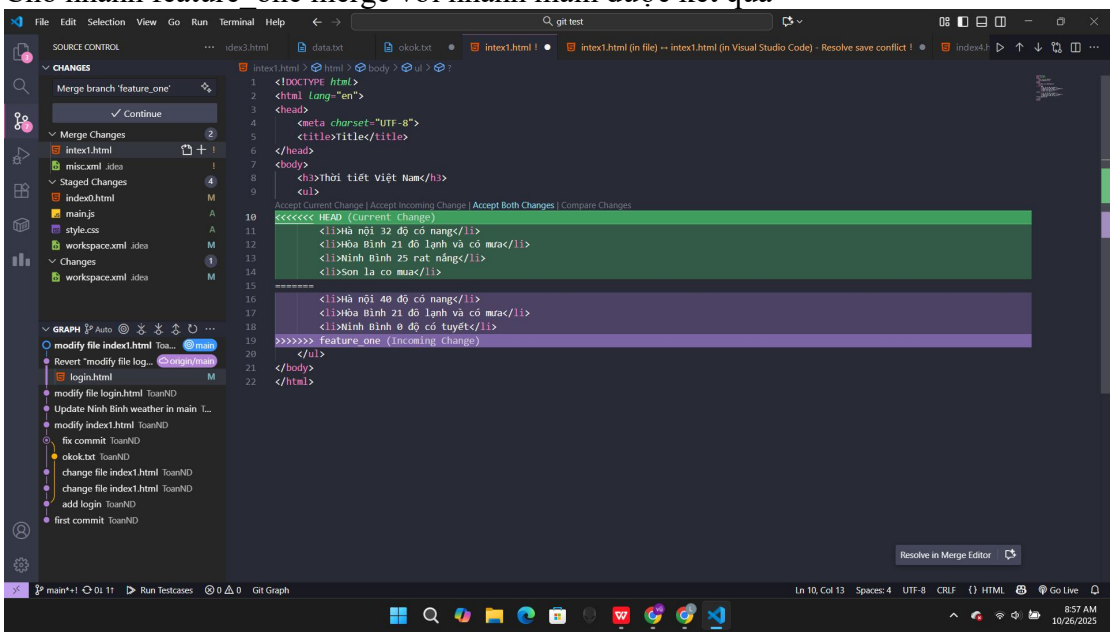




Tại nhánh main chỉnh sửa lại file index1.html

Thực hiện add và commit lại những thay đổi tại nhánh main

Cho nhánh feature_one merge với nhánh main được kết quả



Sau đó xóa hoặc giữa code cần thiết va thực hiện commit lại

Revert code

Kiểm tra commit gần đây: **git log --oneline --graph**

```
PS D:\git test> git log --oneline --graph
*   3e34aae (HEAD -> main) Merge branch 'feature_one'
|\
| * e23abfa (feature_one) okok
| * 6b3c937 (origin/feature_one) add file main.js and modify index1.html
| * fbc55a0 Revert "add file main.js"
| * a7836fc add file main.js
| * 435cbe8 add file style.css
* | e868f4c (origin/main) modify file index1.html
* | dbd6119 Revert "modify file login.html"
* | ec62cd2 modify file login.html
* | 6aa5893 Update Ninh Binh weather in main
* | 5679f44 modify index1.html
```

Git Graph

Sau đó revert lại commit gần nhất: **git revert 3e34aae**

Rollback

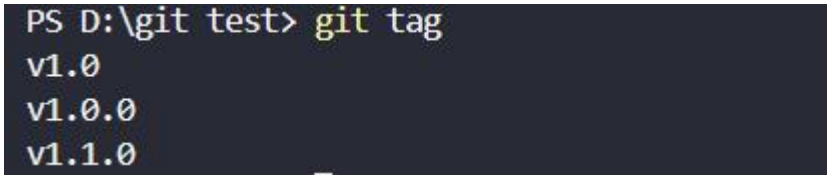
Các cách rollback phiên bản như:
Git checkout <commit_id> : nhược điểm là sẽ mất lịch sử commit
git revert <commit_id> : không mất lịch sử commit , được dùng để xem lại nội dung đã commit khi đã push
git reset --hard <commit_id> : xóa lịch sử của commit sau, chỉ xem khi chưa push

Tạo release

Kiểm tra commit hiện tại
git log --oneline

```
PS D:\git test> git log --oneline
1f6b123 (HEAD -> main, origin/main) Revert "okok"
3e34aae Merge branch 'feature_one'
e23abfa (feature_one) okok
e868f4c modify file index1.html
6b3c937 (origin/feature_one) add file main.js and modify index1.html
fbc55a0 Revert "add file main.js"
dbd6119 Revert "modify file login.html"
ec62cd2 modify file login.html
a7836fc add file main.js
6aa5893 Update Ninh Binh weather in main
5679f44 modify index1.html
b77f55b fix commit
51f3a96 (feature/login) okok.txt
```

Tạo tag cho release
git tag -a v1.0 -m"release version 1.0"
Kiểm tra tag
Git tag



Git show v.1.0

```
PS D:\git test> git show v1.0
tag v1.0
Tagger: ToanND <toannguyencnptit2k5@gmail.com>
Date:   Sun Oct 26 09:23:10 2025 +0700

release version v1.0

commit 1f6b123a5baa633d2628c1000e3afb6aa2cea260 (HEAD -> main, tag: v1.0, origin/main)
Author: ToanND <toannguyencnptit2k5@gmail.com>
Date:   Sun Oct 26 09:20:23 2025 +0700

    Revert "okok"
```

Đẩy tag lên remote
git push origin v1.0

Git Pull/Fetch

git fetch: Tải về các thay đổi mới từ remote nhưng không ảnh hưởng đến code hiện tại, dùng để kiểm tra update trước khi merge.

git pull: Tải về và merge/rebase các thay đổi từ remote vào branch hiện tại, dùng khi muốn cập nhật branch local trước khi tiếp tục làm việc hoặc push code.

Khi nào dùng: fetch để xem update, pull để đồng bộ branch. Trước khi push: luôn pull để tránh xung đột. Chỉ kiểm tra thay đổi: dùng fetch kết hợp xem log commit.