VÕ TIẾN
Thảo luận kiến thức CNTT trường BK về KHMT(CScience), KTMT(CEngineering)
https://www.facebook.com/groups/khmt.ktmt.cse.bku

**Design Framework CS K23**

**CNPM & HTTT CS23**

# GIT trong thực tế đi làm

Thảo luận kiến thức CNTT trường BK
về KHMT(CScience), KTMT(CEngineering)
https://www.facebook.com/groups/khmt.ktmt.cse.bku

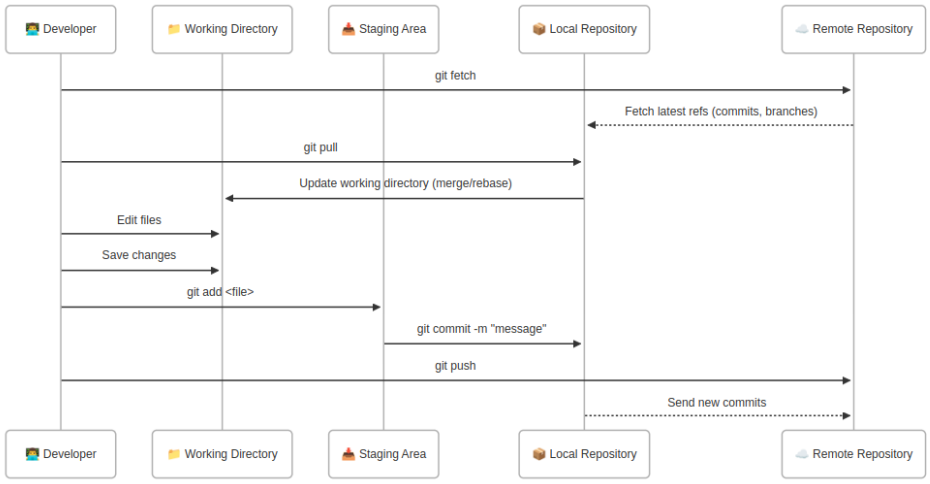
TP. HỒ CHÍ MINH, THÁNG 5/2025

# 1 Lý Thuyết

Git là một hệ thống quản lý phiên bản phân tán (Distributed Version Control System - DVCS)

**Vòng đời của một thay đổi trong Git**

1. Làm thay đổi file trong thư mục làm việc (Working Directory)
2. Thêm vào Staging Area bằng `git add`
3. Tạo Commit bằng `git commit`
4. Gửi lên Remote Repo bằng `git push`

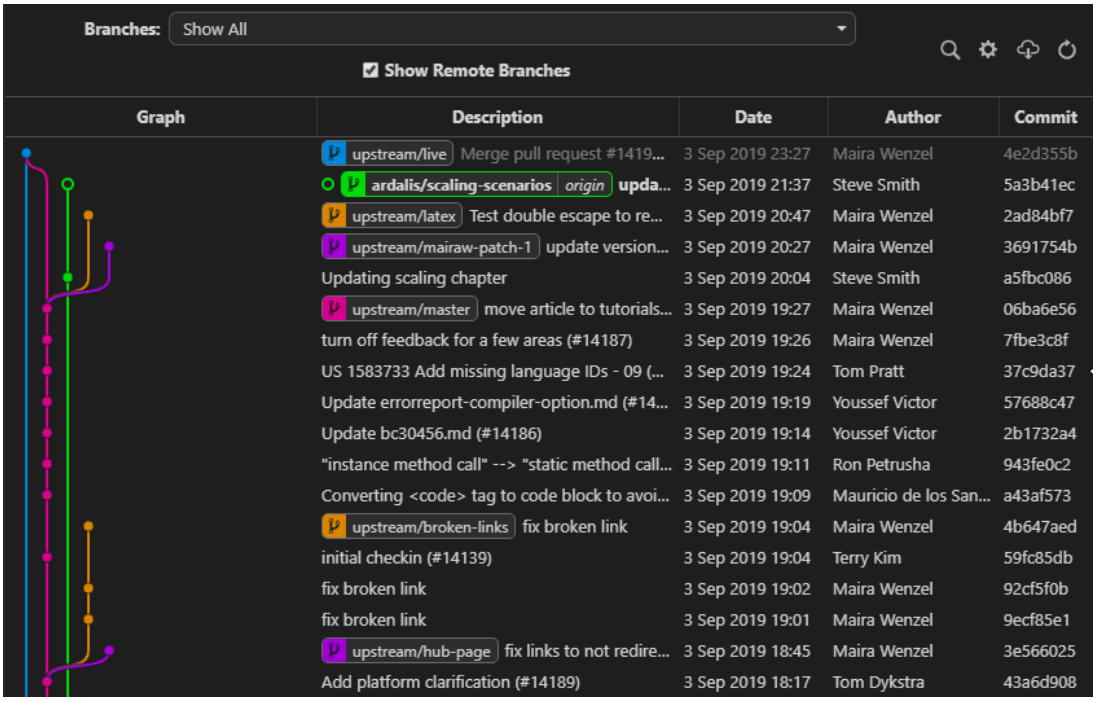



**Các Lệnh hay dùng**

| Nhóm lệnh | Lệnh | Mô tả |
|---|---|---|
| Khởi tạo / bắt đầu | `git init` | Tạo một repository Git mới trong thư mục hiện tại |
| | `git clone <url>` | Tải toàn bộ repository từ máy chủ từ xa về máy cục bộ |
| Làm việc với thay đổi | `git add <file>` | Thêm nội dung file vào vùng staging (chuẩn bị commit) |
| | `git rm <file>` | Xóa file khỏi thư mục làm việc và staging |
| | `git restore <file>` | Khôi phục nội dung file về phiên bản trước |
| Kiểm tra lịch sử | `git status` | Hiển thị trạng thái hiện tại của thư mục làm việc |
| | `git log` | Hiển thị lịch sử các lần commit |
| | `git diff` | So sánh sự khác biệt giữa các phiên bản |
| | `git show` | Hiển thị thông tin về một đối tượng Git (commit, file, v.v.) |
| Quản lý nhánh | `git branch` | Liệt kê, tạo hoặc xóa các nhánh |
| | `git switch <branch>` | Chuyển sang nhánh khác |
| | `git merge <branch>` | Gộp nhánh vào nhánh hiện tại |
| | `git rebase <branch>` | Áp dụng lại commit trên nền tảng mới |
| | `git reset` | Đặt lại HEAD về một trạng thái cụ thể |
| | `git commit -m "msg"` | Tạo một commit từ các thay đổi đã staged |
| Làm việc nhóm | `git fetch` | Tải dữ liệu mới từ máy chủ mà không hợp nhất |
| | `git pull` | Tải và tự động hợp nhất thay đổi từ máy chủ |
| | `git push` | Gửi commit từ local repo lên remote repo |

## 2 Tool anh hay dùng

- GitLens — Git supercharged này chủ yếu coi lịch sử thằng nào mới code đoạn này chứ sài mấy tính năng khác hơi phức tạp
- Git Graph thằng này hay dùng hơn nhưng phần show lịch sử nó ko hiện lên code nên phải tích hợp luôn trên
- Bản chất tool tích hợp các lệnh git ở trên ta chỉ cần nhấn vào thôi ko cần gõ lệnh trên terminal



# 2 Các vấn đề gặp phải (nên thực hành thử)

1. Để làm việc với 2 tài khoản GitHub khác nhau trên cùng 1 máy, bạn cần cấu hình Git sao cho có thể sử dụng được từng tài khoản tương ứng với từng repository ?

   - Tạo SSH key cho từng tài khoản GitHub

     ```
     # Tạo SSH key cho tài khoản 1
     ssh-keygen -t ed25519 -C "email_account_1@example.com" -f ~/.ssh/id_ed25519_account1

     # Tạo SSH key cho tài khoản 2
     ssh-keygen -t ed25519 -C "email_account_2@example.com" -f ~/.ssh/id_ed25519_account2
     ```

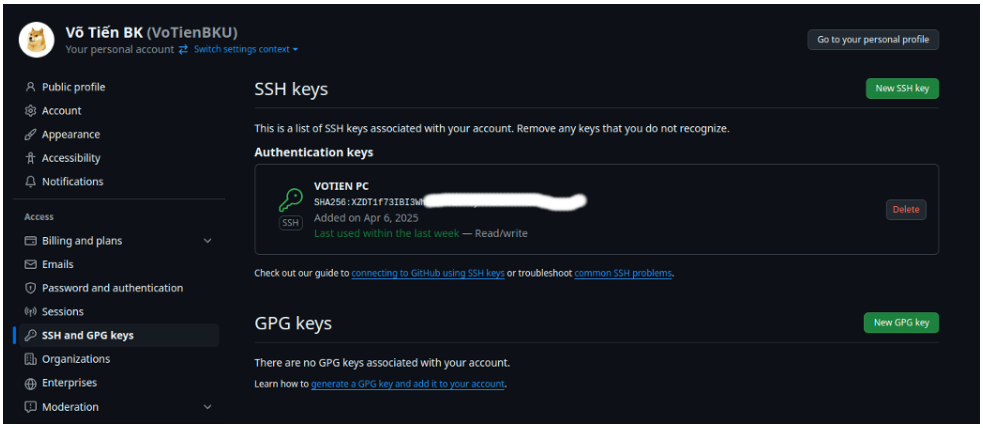
   - Cấu hình SSH config để phân biệt tài khoản

     ```
     ## file ~/.ssh/config
     # Account 1
     Host github-account1
       HostName github.com
       User git
       IdentityFile ~/.ssh/id_ed25519_account1

     # Account 2
     Host github-account2
       HostName github.com
       User git
       IdentityFile ~/.ssh/id_ed25519_account2
     ```

   - Lấy SSH public key

     ```
     cat ~/.ssh/id_ed25519_account1.pub
     cat ~/.ssh/id_ed25519_account2.pub
     ```

   - Đăng nhập tài khoản GitHub tương ứng, vào Settings > SSH and GPG keys, thêm public key mới.

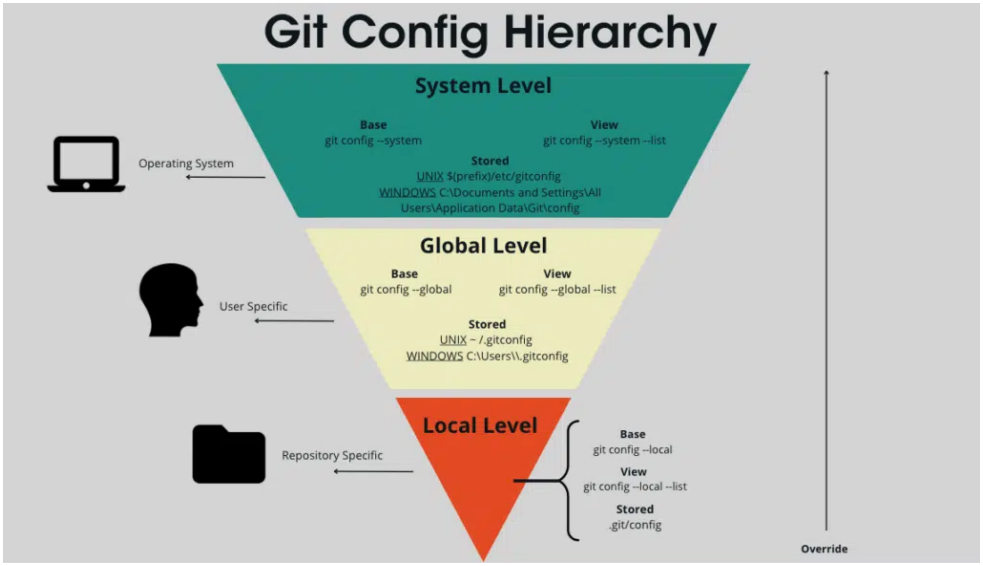
   - Cấu hình remote URL cho repo tương ứng

     ```
     # Repo cho tài khoản 1:
     git clone git@github-account1:username1/repo.git

     # Repo cho tài khoản 2:
     git clone git@github-account2:username2/repo.git
     ```

   - Cấu hình Git user/email cho từng repo

     ```
     # Repo cho tài khoản 1:
     git config user.name "User Name 1"
     ```

```
git config user.email "email_account_1@example.com"

# Repo cho tài khoản 2:
git config user.name "User Name 2"
git config user.email "email_account_2@example.com"
```




2. Bạn đã push lên GitHub một nhánh có 2 commit

```
* Commit B  (bạn muốn sửa lại)
* Commit A
```

   - Quay lại commit A (gỡ bỏ commit B tạm thời)

     ```
     git reset HEAD~1
     ```

   - Sửa đổi nội dung nếu cần
   - Force push để cập nhật lên GitHub

     ```
     git push --force
     ```

3. Có 3 commit nhưng bạn chỉ muốn lấy 2 commit đầu và cuối

```
* Commit C (cuối)
* Commit B (bạn muốn bỏ)
* Commit A (đầu)
```

   - Xác định commit hash

     ```
     git log --oneline

     c3f9e5c Commit C
     a1b2c3d Commit B   ← bỏ qua
     e5f6g7h Commit A
     ```

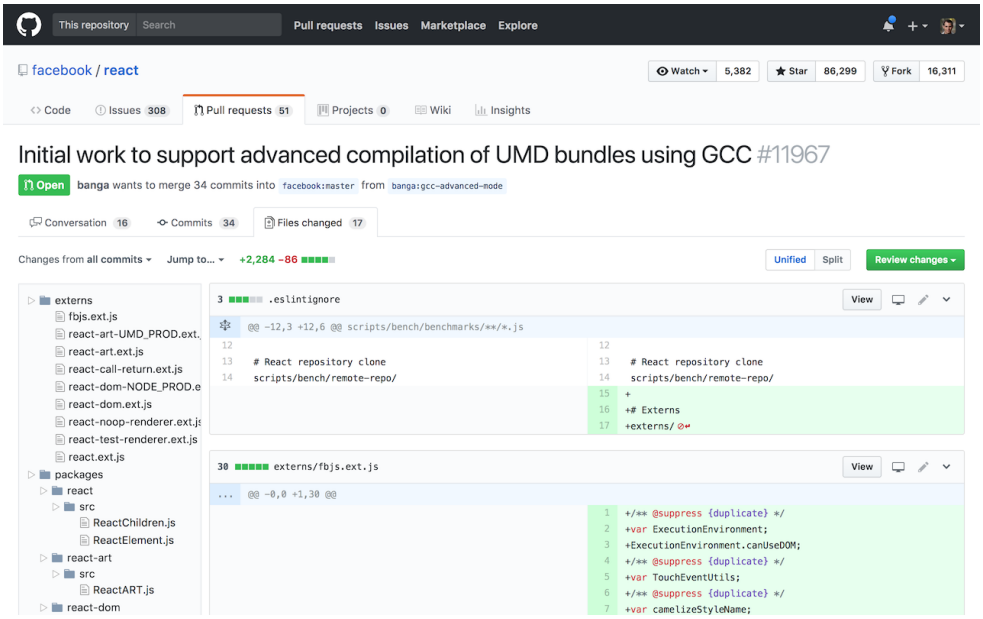
   - Tạo nhánh mới và Cherry-pick commit A và C

     ```
     git checkout -b new-branch
     git cherry-pick e5f6g7h  # commit A
     git cherry-pick c3f9e5c  # commit C
     ```

4. Tìm lỗi trong các commit gần đây, khôi phục các file ổn định, chỉnh sửa file bị lỗi, rồi tạo commit mới.

| Bước | Mục đích | Git lệnh tương ứng |
|---|---|---|
| 1. Kiểm tra | Tìm commit lỗi | `git log`, `git checkout <commit>` |
| 2. Quay về nhánh chính | Trở về HEAD mới nhất | `git checkout main` |
| 3. Revert commit lỗi | Đảo ngược commit mà vẫn giữ lịch sử | `git revert <hash>` |
| 4. Reset nếu lỗi là commit cuối | Loại commit cuối để sửa | `git reset -soft HEAD~1` |
| 5. Revert file không lỗi | Giữ nguyên file không cần sửa | `git checkout HEAD - <file>` |
| 6. Sửa file lỗi | Chỉnh sửa code | `vim, code, ...` |
| 7. Commit lại | Ghi nhận thay đổi đã sửa | `git commit -m "<message>"` |

5. Khi bạn code xong trên nhánh bạn và yêu cầu cho lead merge vào với nhanh main.
   - Tạo merge request `Repositories → Pull Request → New Pull Request`
   - Mô tả description nếu có
   - Gửi `pull request → review → merge.` chọn người Review & Approve
   - Lead sẽ Comment nếu cần chỉnh không thì Approve




6. Khi đang làm việc trên một nhánh feature/login, bạn phát hiện nhánh main đã được cập nhật (có thêm commit mới).
   - Fetch cập nhật mới nhất từ remote

     `git fetch origin`
   - Rebase nhánh main vào nhánh đang làm (feature/login)

     `git rebase origin/main`
   - Nếu có xung đột khi rebase, Git sẽ báo và yêu cầu bạn xử lý.
   - Nếu muốn hủy rebase:

     `git rebase --abort`
7. Có thể dùng git graph cho nhanh chứ ai rảnh đầu mà gõ từng lệnh (gõ từng lệnh sẽ hoạt động trên server vì lúc nào ko có git graph mà sài)

# 3 Trắc Nghiệm

1. Bạn muốn lấy cập nhật mới nhất từ nhánh main và giữ lịch sử commit sạch sẽ, bạn chọn cách nào?

   a) git merge main  b) git rebase main

2. Bạn vừa commit một đoạn code sai logic và chưa push lên remote. Bạn muốn quay lại trước commit đó để sửa lại, đồng thời không mất code đã viết, bạn nên dùng lệnh nào?

   a) git reset –soft HEAD 1  b) git revert HEAD

3. Bạn đã thực hiện 3 commit nhỏ liên tiếp (A, B, C) để hoàn thành một tính năng. Giờ bạn muốn gộp cả 3 commit này lại thành 1 commit duy nhất trước khi push, bạn nên dùng?

   a) git reset –soft HEAD 3 rồi git commit -m "Gộp lại"
   b) git revert HEAD 2 rồi git commit -m "Gộp lại"

4. Sau khi Leader merge code của bạn vào main và triển khai lên production thì phát hiện có lỗi. Leader nên làm gì tiếp theo?

   a) Thực hiện git revert commit gây lỗi để đảo ngược ngay trên main
   b) Reset lại toàn bộ nhánh main về trước đó bằng git reset –hard và force push lên remote

5. Bạn đã làm nhiều commit trên nhánh feature nhưng phát hiện một commit quan trọng trên nhánh main cần đưa vào ngay để chạy thử. Bạn sẽ làm gì để chỉ lấy riêng commit đó vào nhánh feature?

   a) Dùng git cherry-pick <commit_hash> để áp commit đó vào feature mà không cần merge toàn bộ main.
   b) Dùng git merge main để lấy toàn bộ thay đổi trên main vào feature.

6. Bạn muốn cập nhật thông tin mới nhất từ repository trên GitHub về máy local, nhưng chưa muốn thay đổi gì trên nhánh hiện tại. Bạn sẽ dùng lệnh nào?

   a) git fetch  b) git pull

7. Khi bạn muốn cập nhật nhánh hiện tại bằng cách tải và tự động kết hợp các thay đổi mới nhất từ remote repository, bạn sẽ dùng lệnh nào?

   a) git fetch  b) git pull

8. Bạn đã có một số commit đã đẩy lên remote. Giờ bạn muốn sửa lại commit cuối cùng trên local (dùng git commit –amend) và cập nhật lại remote. Lúc này bạn sẽ dùng lệnh nào?

   a) git push origin <branch> để đẩy bình thường.
   b) git push -f origin <branch> để ghi đè lịch sử trên remote.

9. Bạn đang làm việc trên nhánh feature và đã commit 3 lần (C1, C2, C3). Trong lúc đó, nhánh main trên remote có thêm 2 commit mới. Bạn muốn cập nhật nhánh feature của mình với những thay đổi mới nhất từ main mà giữ lịch sử commit sạch sẽ (linear history). Quy trình bạn thực hiện:

   - git fetch origin
   - git rebase origin/main trên nhánh feature
   - Trong quá trình rebase, xảy ra conflict ở commit C2. Bạn chỉnh sửa file, dùng git add <file>, rồi chạy git rebase –continue để tiếp tục.
   - Sau khi rebase thành công, bạn muốn đẩy feature lên remote.

   a) git push origin feature  b) git push -f origin feature

10. Bạn nghi ngờ một commit nào đó đã gây ra lỗi trong dự án. Quy trình nào dưới đây là đúng nhất để xác định và sửa lỗi?

    - Kiểm tra lịch sử commit và chọn commit nghi ngờ.
    - Dùng `git checkout <commit>` để xem mã nguồn tại commit đó.

- Nếu commit đó có lỗi, quay về nhánh chính `main`.
- Dùng `git revert <commit>` để đảo ngược commit lỗi mà không làm mất lịch sử.
- Nếu cần, chỉnh sửa code cho đúng và tạo commit mới.

a) Quy trình đúng và đầy đủ.

b) Thiếu bước kiểm tra lỗi bằng cách chạy code trên commit đó.

11. Luyện https://learngitbranching.js.org/

Chương 2 tiếp theo ta học CI/CD và deploy nên sẽ gặp lại phần github này nhiều nhưng mà ứng dụng trên gitlab https://about.gitlab.com/