Cẩm Nang Git & GitHub

1. Git Cơ Bản & Quy Trình Hoàn Chỉnh

Q1. Git là gì?

Git là một Distributed Version Control System giúp lập trình viên theo dõi thay đổi trong code, cộng tác với người khác và quản lý các phiên bản khác nhau một cách hiệu quả.

Q2. Git vs GitHub

GIT	GITHUB
• Là một VCS	• Là một nền tảng
• Chạy trên máy local	• Chạy trên cloud
• Công cụ dòng lệnh	• Dịch vụ nền web
• Theo dõi thay đổi file	• Hỗ trợ cộng tác nhóm
• Lưu dữ liệu dạng snapshot	• Lưu trữ Git repository

Q3. Vì sao nên dùng Git?

- Theo dõi mọi thay đổi trong dự án
- Làm việc trên nhiều phiên bản
- Cộng tác theo nhóm
- Branching và Merging
- Sao lưu & khôi phục code
- Mã nguồn mở & miễn phí

Q4. Quy trình Git hoàn chỉnh

Working Directory
– File dự án của bạn
– Thay đổi chưa được theo dõi

git add ↓

Staging Area
– Thay đổi được stage tại đây
– Sẵn sàng để commit

git commit ↓

Local Repository
– Thay đổi đã commit vào .git repo ở local

git push ↓

Remote Repository (GitHub)
– Thay đổi được tải lên remote repository

Q5. Cấu hình Git (lần đầu sử dụng)

```
git config --global user.name "Your Name"
git config --global user.email "you@example.com"
```

Q6. Các lệnh Git dùng nhiều nhất

Lệnh	Mô tả	Ví dụ
git init	Khởi tạo một Git repository mới	git init
git status	Kiểm tra trạng thái working directory	git status
git add <file>	Thêm file vào staging area	git add app.py
git add .	Thêm tất cả file vào staging area	git add .
git commit -m "msg"	Commit các thay đổi đã stage	git commit -m "Initial commit"
git log	Xem lịch sử commit	git log
git diff	Xem thay đổi (chưa stage)	git diff
git diff --staged	Xem thay đổi (đã stage)	git diff --staged
git branch	Liệt kê tất cả branch	git branch
git checkout <branch>	Chuyển sang một branch	git checkout main
git checkout -b <branch>	Tạo và chuyển sang branch mới	git checkout -b feature/login
git merge <branch>	Gộp branch vào branch hiện tại	git merge feature/login
git remote -v	Liệt kê tất cả remote repository	git remote -v
git push origin <branch>	Đẩy branch lên remote repository	git push origin main
git pull origin <branch>	Kéo thay đổi mới nhất từ remote	git pull origin main
git clone <url>	Clone một remote repository	git clone https://github.com/user/repo.git

Q7. Trạng thái file trong Git

- Untracked – File mới, Git chưa theo dõi.
- Unmodified – File đã theo dõi, không đổi.
- Modified – File đã đổi, chưa stage.
- Staged – File đã stage, sẵn sàng commit.

Q8. .gitignore

Dùng để bỏ qua các file/thư mục không nên được Git theo dõi.

Ví dụ (.gitignore)

```
node_modules/
.env
*.log
dist/
__pycache__/
```

Cẩm Nang Git & GitHub

2. Branch & Quản Lý Branch

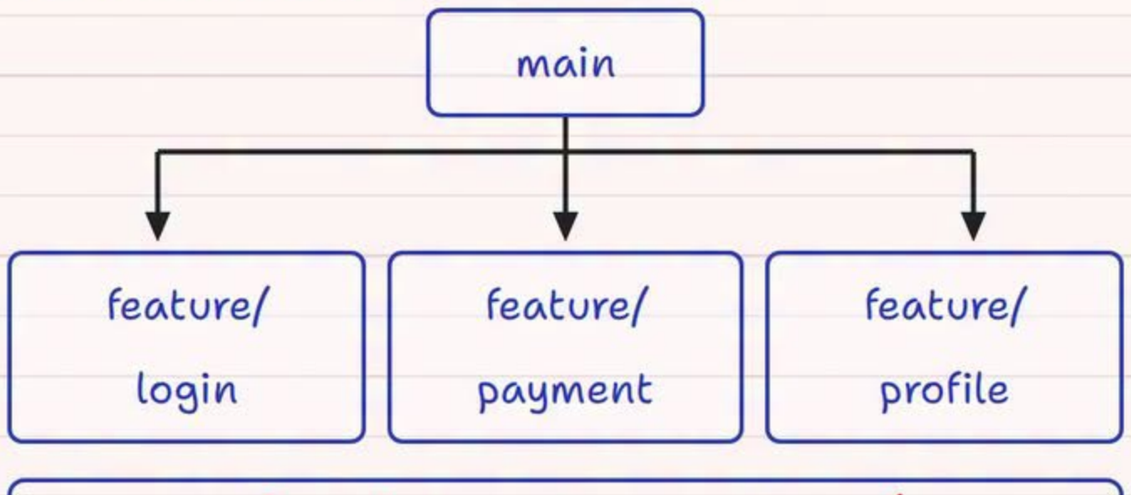
Q1. Branch là gì?

Branch là một nhánh phát triển riêng biệt. Nó cho phép bạn làm việc trên các tính năng mới mà không ảnh hưởng đến code chính.

Q2. Các lệnh về Branch

- git branch → Liệt kê branch ở local
  git branch / git branch -v
- git branch -a → Liệt kê tất cả branch
  git branch -a
- git checkout <branch> → Chuyển sang branch
  git checkout feature/login
- git switch <branch> → Chuyển branch (cách mới)
  git switch feature/login
- git checkout -b <branch> → Tạo & chuyển
  git checkout -b feature/payment
- git switch -c <branch> → Tạo & chuyển (mới)
  git switch -c feature/profile
- git branch -m <new-name> → Đổi tên branch
  git branch -m new-name
- git branch -d <branch> → Xóa branch (an toàn)
  git branch -d feature/login
- git branch -D <branch> → Xóa branch (ép buộc)
  git branch -D old-feature
- git merge <branch> → Gộp branch
  git merge feature/login

Q3. Luồng làm việc với Branch



Cấu trúc branch trong thực tế

```
main ┬─ feature / login
     ├─ feature / payment
     └─ feature / profile
```

Q4. Ví dụ một quy trình

```
git checkout -b feature/login      # tạo branch
git add .                          # stage thay đổi
git commit -m "Login API"          # commit thay đổi
git checkout main                  # chuyển về main
git merge feature/login            # gộp branch
git branch -d feature/login        # xóa branch
```

Q5. Tóm tắt nhanh

Hành động	Lệnh
Liệt kê branch	git branch
Liệt kê tất cả branch	git branch -a
Chuyển branch	git switch <branch>
Tạo branch	git checkout -b <branch>
	git switch -c <branch>

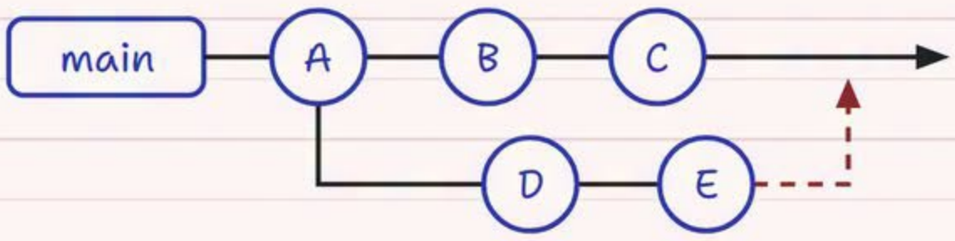
Hành động	Lệnh
Xóa branch (an toàn)	git branch -d <branch>
Xóa branch (ép buộc)	git branch -D <branch>
Gộp branch	git merge <branch>
Xem branch hiện tại	git branch --show-current
Xem sơ đồ branch	git log --oneline --graph

Cẩm Nang Git & GitHub

3. Merge, Rebase & Cherry-Pick

Q1. MERGE

Gộp các thay đổi từ một branch vào một branch khác.

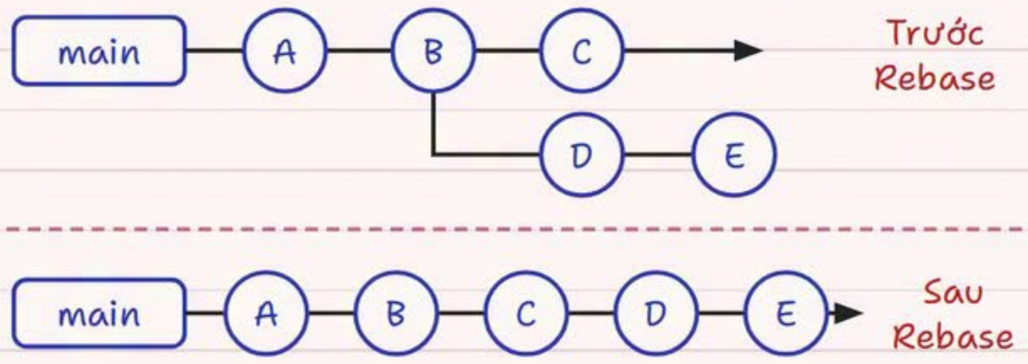


Lệnh

```
git checkout main
git merge feature/login
```

Q2. REBASE

Di chuyển hoặc gộp một chuỗi commit sang một base mới.

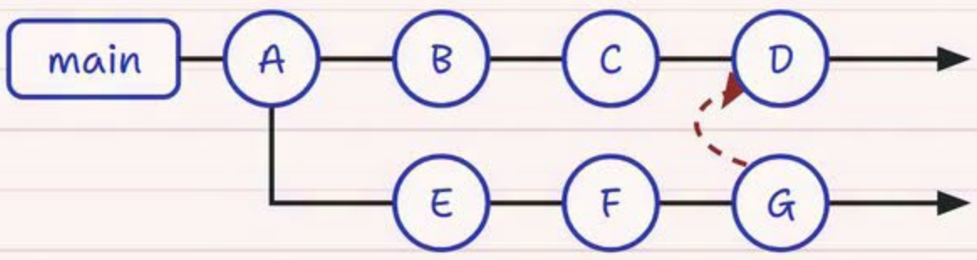


Lệnh

```
git checkout feature/login
git rebase main
```

Q3. CHERRY-PICK

Áp dụng một (hoặc vài) commit cụ thể từ branch này sang branch khác.



Lệnh

```
git checkout main
git cherry-pick <commit-id>
```

Q4. MERGE vs REBASE

Tiêu chí	MERGE	REBASE
Lịch sử	Giữ nguyên toàn bộ lịch sử	Tạo lịch sử tuyến tính
Commit ID	Sinh ra một merge commit mới	Các commit cũ bị ghi lại
Phù hợp nhất với	Cộng tác nhóm (branch dùng chung)	Branch cá nhân, lịch sử gọn gàng
Rủi ro	An toàn	Ghi lại lịch sử (cần dùng cẩn thận)

Q5. KHI NÀO DÙNG CÁI GÌ?

- Dùng MERGE khi làm việc nhóm và muốn giữ lại toàn bộ lịch sử.
- Dùng REBASE khi làm trên branch riêng và muốn lịch sử gọn, tuyến tính.
- Dùng CHERRY-PICK khi chỉ cần một commit cụ thể, không cần cả branch.

Q6. VÍ DỤ THỰC TẾ

· VÍ DỤ MERGE

```
git checkout main
git merge feature/login
# đưa toàn bộ thay đổi từ
  feature/login vào main
```

· VÍ DỤ REBASE

```
git checkout feature/login
git rebase main
git checkout main
git merge feature/login
# lịch sử tuyến tính, gọn gàng
```

· VÍ DỤ CHERRY-PICK

```
git checkout main
git cherry-pick f1a2b3c
# chỉ áp dụng đúng một
  commit vào main
```

Cẩm Nang Git & GitHub

4. Sửa Sai (Reset, Revert, Restore, Rm, Mv)

Q1. Hoàn tác thay đổi (chưa commit)

- Bỏ các thay đổi trong working directory

```
git restore <file>          # hoàn tác thay đổi trong một file
git restore .               # hoàn tác thay đổi trong tất cả file
```

- Bỏ stage các thay đổi

```
git restore --staged <file>     # bỏ stage file
git restore --staged .          # bỏ stage tất cả
```

Q2. Hoàn tác Commit

- Hoàn tác commit cuối (giữ thay đổi)

```
git reset --soft HEAD~1
```

- Hoàn tác commit cuối (bỏ stage)

```
git reset --mixed HEAD~1    # mặc định
```

- Hoàn tác commit cuối (xóa thay đổi)

```
git reset --hard HEAD~1     # cẩn thận!
```

Q3. Reset vs Revert

Tiêu chí	RESET	REVERT
Tác dụng	Đưa HEAD về commit trước đó	Tạo commit mới để hoàn tác
Lịch sử	Ghi lại lịch sử	Không thay đổi lịch sử
An toàn trên branch chung	Không (nên tránh)	Có (an toàn)
Dùng khi	Làm việc ở local	Làm việc theo nhóm

Q4. Revert Commit

- Tạo một commit mới để hoàn tác thay đổi

```
git revert <commit-id>
```

- Revert commit cuối cùng

```
git revert HEAD
```

Q5. Xóa & Di chuyển file

- Xóa file

```
git rm <file>               # xóa khỏi working dir & staging area
git rm --cached <file>      # chỉ xóa khỏi staging area
```

- Di chuyển / Đổi tên file

```
git mv old-name new-name    # đổi tên
```

Q6. Xóa hẳn các Commit

- Xóa N commit gần nhất

```
git reset --hard HEAD~N
```

Ví dụ (N = 3)

```
$ git log --oneline
a1b2c3d (HEAD -> main)  Commit 3
b2c3d4e  Commit 2
c3d4e5f  Commit 1
d4e5f6g  Commit 0
$ git reset --hard HEAD~3 # xóa 3 commit cuối
$ git log --oneline
d4e5f6g (HEAD -> main)  Commit 0
```

Q7. Khôi phục commit / thay đổi đã mất

- Xem toàn bộ reflog

```
git reflog      # hiện mọi hành động, kể cả commit đã xóa
```

- Quay lại một commit

```
git reset --hard <commit-id>
```

- Khôi phục branch đã xóa

```
git checkout -b <branch-name> <commit-id>
```

Q8. Tóm tắt (Tra cứu nhanh)

Tác vụ	Lệnh
Hoàn tác thay đổi (chưa stage)	git restore <file> / git restore .
Bỏ stage thay đổi	git restore --staged <file>
Hoàn tác commit cuối (giữ thay đổi)	git reset --soft HEAD~1
Hoàn tác commit cuối (bỏ stage)	git reset --mixed HEAD~1
Hoàn tác commit cuối (xóa thay đổi)	git reset --hard HEAD~1
Revert một commit (an toàn)	git revert <commit-id>
Xóa file	git rm <file>
Đổi tên / Di chuyển file	git mv old-name new-name

Cẩm Nang Git & GitHub

5. Remote Repository & Quy Trình GitHub

Q1. Remote cơ bản

- Remote là phiên bản dự án của bạn được lưu trữ trên internet (GitHub).
- Nó cho phép cả nhóm cùng cộng tác.

Q2. Các lệnh Remote

```
git remote -v                          # liệt kê remote
git remote add origin <url>            # thêm remote
git remote remove origin               # xóa remote
git fetch origin                       # lấy cập nhật
git pull origin <branch>               # fetch + merge
git push origin <branch>               # đẩy thay đổi
git clone <url>                        # clone repo
```

Q3. Clone vs Fetch vs Pull

Lệnh	Tác dụng
git clone	Tải toàn bộ repo (lần đầu tiên)
git fetch	Chỉ tải thay đổi về (không merge)
git pull	Tải về + merge vào branch của bạn
git push	Đẩy commit ở local lên remote

Q4. Quy trình GitHub (hằng ngày)

1. Clone Repo — `git clone <url>`
   ↓
2. Tạo Branch — `git checkout -b feature/login`
   ↓
3. Code & Commit — `git add .` / `git commit -m "Login API"`
   ↓
4. Push Branch — `git push origin feature/login`
   ↓
5. Tạo Pull Request — Mở PR trên GitHub
   ↓
6. Code Review — Cả nhóm review code
   ↓
7. Merge PR — Gộp vào branch main
   ↓
8. Pull bản mới nhất — `git pull origin main`

Q5. Remote Tracking Branch

- Theo dõi branch trên remote để cộng tác dễ dàng hơn.

```
git branch -r                          # liệt kê remote branch
git branch -a                          # liệt kê tất cả branch
git checkout -b <branch> origin/<branch>   # tạo branch local từ remote
git push -u origin <branch>            # đặt upstream & theo dõi branch
```

Q6. Upstream vs Origin

Origin	Upstream
• Remote của bản fork của bạn • Trỏ tới GitHub repo của bạn • Bạn push thay đổi lên đây	• Repo gốc ban đầu • Trỏ tới dự án gốc • Bạn fetch thay đổi từ đây

Ví dụ

```
# Thêm upstream (chỉ dùng khi fork)
git remote add upstream <original_repo_url>
# Fetch từ upstream
git fetch upstream
# Merge upstream/main vào branch của bạn
git checkout main
git merge upstream/main
```

Q7. Fork & Pull Request

Q8. Các tính năng GitHub hữu ích

Tính năng	Mục đích
Pull Request (PR)	Đề xuất thay đổi & review
Issues	Theo dõi bug, task, tính năng
Projects	Bảng Kanban quản lý công việc
Actions (CI/CD)	Tự động build, test, deploy

Tính năng	Mục đích
Fork	Sao chép repo về tài khoản của bạn
Watch	Nhận thông báo cập nhật của repo
Star	Đánh dấu các repo quan trọng
GitHub Pages	Đăng website tĩnh miễn phí

Cẩm Nang Git & GitHub

6. Merge Conflict & Git Stash

Q1. Merge Conflict là gì?

Conflict xảy ra khi Git không thể tự động gộp các thay đổi.

Q2. Ví dụ về Conflict

File: app.txt	
HEAD (main)	**feature/login**
Line 1 Line 2 Old Code	Line 1 Line 2 New Code

Sau khi merge (bị conflict)

```
<<<<<<< HEAD (main)
Old Code
=======
New Code
>>>>>>> feature/login
```

Cách xử lý:

- Sửa file và giữ lại phần code đúng.
- Xóa các conflict marker.
- Lưu file và đánh dấu là đã xử lý.
- Commit kết quả merge.

```
git add app.txt
git commit
```

Q3. Conflict xảy ra khi nào?

- Merge hai branch cùng sửa một chỗ.
- Pull thay đổi mới khi đang có thay đổi ở local.
- Rebase khi các thay đổi chồng lấn nhau.
- Conflict khi cherry-pick.

Q4. Các bước xử lý Conflict

1. Git báo file đang bị conflict.
2. Mở file và tìm các conflict marker.
3. Giữ lại phần thay đổi đúng.
4. Xóa <<<<<<<, =======, >>>>>>>.
5. Lưu file.
6. git add <file>
7. git commit

Q5. Hủy Merge / Rebase

- Hủy Merge

```
git merge --abort        # quay về trạng thái trước
```

- Hủy Rebase

```
git rebase --abort       # hủy rebase
```

- Tiếp tục sau khi đã xử lý conflict

```
git rebase --continue    # tiếp tục rebase
git merge --continue     # tiếp tục merge
```

Q6. Git Stash (Cất tạm công việc)

Stash lưu lại các thay đổi hiện tại mà không cần commit.

Lệnh	Mục đích
git stash	Cất tạm thay đổi ở local
git stash list	Liệt kê tất cả stash
git stash pop	Áp dụng stash mới nhất & xóa nó
git stash apply	Áp dụng stash mới nhất (vẫn giữ lại)
git stash apply stash@{n}	Áp dụng một stash cụ thể
git stash drop	Xóa stash mới nhất
git stash clear	Xóa toàn bộ stash

Q7. Ví dụ – Quy trình Stash

- Bạn đang sửa một file nhưng cần chuyển branch.

```
$ git status                  # xem các thay đổi
$ git stash                   # cất tạm thay đổi
$ git switch main             # chuyển branch
$ git pull                    # lấy thay đổi mới nhất
$ git switch feature/login    # quay lại branch cũ
$ git stash pop               # lấy lại thay đổi
```

Q8. Ví dụ xử lý Conflict

1. $ git merge feature/login # thử merge (conflict xảy ra)
2. Sửa file → xử lý conflict
3. $ git add app.txt # đánh dấu đã xử lý
4. $ git commit # hoàn tất merge

Q9. Tóm tắt nhanh

Tình huống	Lệnh
Cất tạm công việc	git stash
Xem danh sách stash	git stash list
Áp dụng & xóa stash	git stash pop
Áp dụng stash (vẫn giữ)	git stash apply
Hủy Merge	git merge --abort
Hủy Rebase	git rebase --abort
Tiếp tục Rebase	git rebase --continue
Tiếp tục Merge	git merge --continue

Cẩm Nang Git & GitHub

7. Git Cheat Sheet (Dùng Nhiều Nhất)

1. Setup & Config

```
git config --global user.name "Your Name"
git config --global user.email "you@email.com"
git config --list   # xem toàn bộ config
```

2. Khởi tạo dự án

```
git init                # tạo repo mới
git clone <url>         # clone repo có sẵn
```

3. Quy trình cơ bản


```
git status              # trạng thái
git add <file>          # thêm file
git add .               # thêm hết
git commit -m "msg"     # commit
git log --oneline       # log gọn
git log --graph         # log sơ đồ
  --oneline --all
```

4. Branching

```
git branch              # liệt kê branch
git branch -a           # liệt kê tất cả
git branch <name>       # tạo branch
git switch <name>       # chuyển branch
git switch -c <name>    # tạo & chuyển
git branch -m <new-name> # đổi tên
git branch -d <name>    # xóa (an toàn)
git branch -D <name>    # xóa (ép buộc)
```

5. Merge, Rebase & Cherry-Pick

```
git merge <branch>      # gộp branch
git rebase <branch>     # rebase branch
git rebase --abort      # hủy rebase
git cherry-pick <commit-id>  # lấy 1 commit
```

Quy trình hằng ngày

1. git pull origin main
2. git switch -c feature/xyz
3. Viết code, chỉnh sửa
4. git add .
5. git commit -m "message"
6. git push -u origin feature/xyz
7. Tạo PR trên GitHub
8. Sau khi review → git pull origin main

```
git switch main
git merge feature/xyz
```
} giữ branch main của bạn luôn mới nhất

6. Remote & Push / Pull

```
git remote -v                  # liệt kê remote
git remote add origin <url>    # thêm remote
git push -u origin <branch>    # push & đặt upstream
git push                       # đẩy thay đổi
git pull                       # fetch + merge
git fetch                      # chỉ fetch
git pull --rebase              # pull kèm rebase
```

7. Hoàn tác & Khôi phục

```
git restore <file>             # bỏ thay đổi trong file
git restore .                  # bỏ mọi thay đổi
git restore --staged <file>    # bỏ stage file
git reset --soft HEAD~1        # hoàn tác commit (giữ)
git reset --mixed HEAD~1       # hoàn tác (bỏ stage)
git reset --hard HEAD~1        # hoàn tác (xóa hẳn)
git revert <commit-id>         # tạo commit hoàn tác
git rm <file>                  # xóa file
git mv old new                 # đổi tên / di chuyển
```

8. Stash

```
git stash              # cất tạm thay đổi
git stash list         # liệt kê stash
git stash pop          # áp dụng & xóa stash
git stash apply        # áp dụng (vẫn giữ)
git stash drop         # xóa stash
git stash clear        # xóa toàn bộ stash
```

9. Các lệnh hay dùng (một dòng)

- git status – xem trạng thái hiện tại
- git log --oneline --graph – xem lịch sử
- git diff – xem các thay đổi
- git show <commit-id> – xem chi tiết commit
- git branch --show-current – tên branch hiện tại
- git pull origin <branch> – lấy bản mới nhất
- git push origin <branch> – đẩy thay đổi lên
- git remote show origin – thông tin remote
- git clean -fd – xóa file/thư mục chưa track

Nguyên tắc vàng khi dùng Git

1. Commit thường xuyên & đặt message có ý nghĩa.
2. Pull bản mới nhất trước khi bắt đầu việc mới.
3. Dùng feature branch, đừng làm thẳng trên main.
4. Không bao giờ push code trực tiếp lên main.
5. Xử lý conflict cẩn thận, test rồi mới commit.