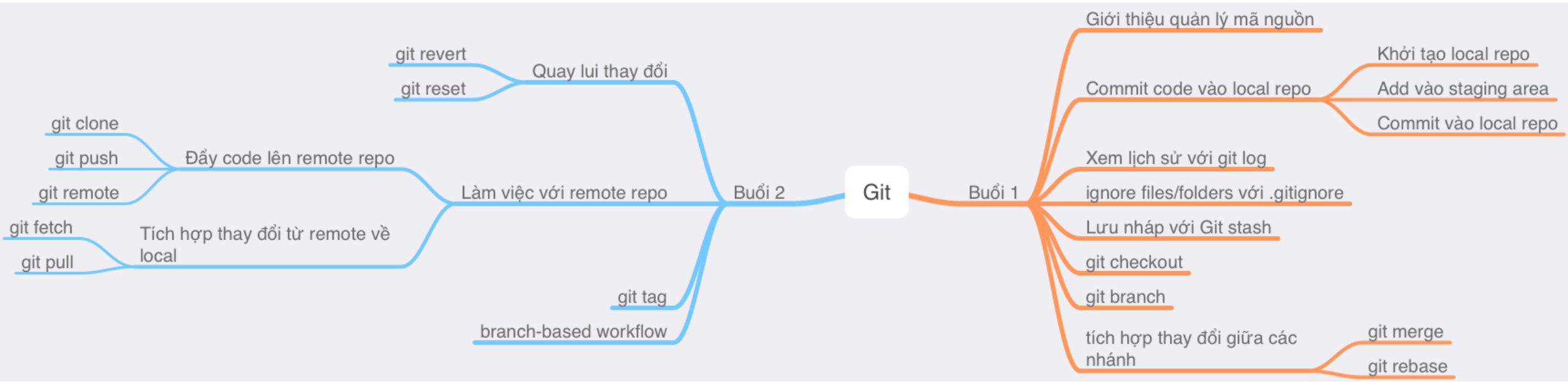
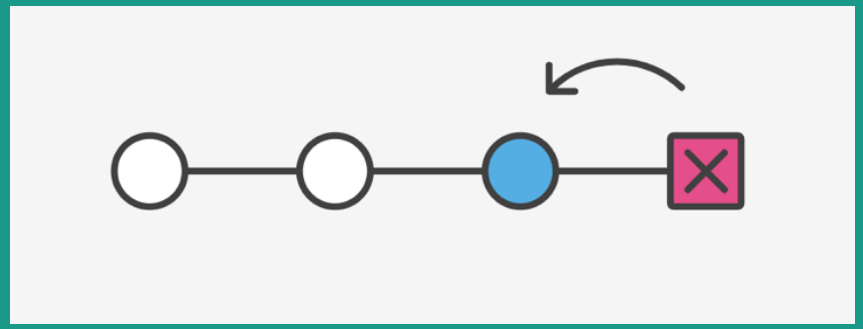
TechMaster
Học là có việc

# Git căn bản

cuong@techmaster.vn


Git
Buổi 1
Giới thiệu quản lý mã nguồn
Commit code vào local repo
Khởi tạo local repo
Add vào staging area
Commit vào local repo
Xem lịch sử với git log
ignore files/folders với .gitignore
Lưu nháp với Git stash
git checkout
git branch
tích hợp thay đổi giữa các nhánh
git merge
git rebase
Buổi 2
Quay lui thay đổi
git revert
git reset
Làm việc với remote repo
Đẩy code lên remote repo
git clone
git push
git remote
Tích hợp thay đổi từ remote về local
git fetch
git pull
git tag
branch-based workflow

# Git reset



# Tình huống

Sau khi commit vài lần những thay đổi hết sức ngớ ngẩn, tôi muốn viết lại lịch sử, loại bỏ hẳn các commit ngớ ngẩn vĩnh viễn

Nhận xét:

- Nếu chưa pushed lên remote repo, đây là cách gọn gang sạch sẽ
- Nếu đã pushed lên remote repo, cách này gây xung đột

```
$ git log --oneline
c8a4b05 (HEAD -> b4) Fix demo.py
00750f8 Shit demo.py
6e1c7fe Change demo.py
08a3022 2nd commit
5acc633 (branch2) Add ReadMe.md

$ git reset --hard 08a3022
$ git log --oneline
08a3022 (HEAD -> b4) 2nd commit
5acc633 (branch2) Add ReadMe.md
```

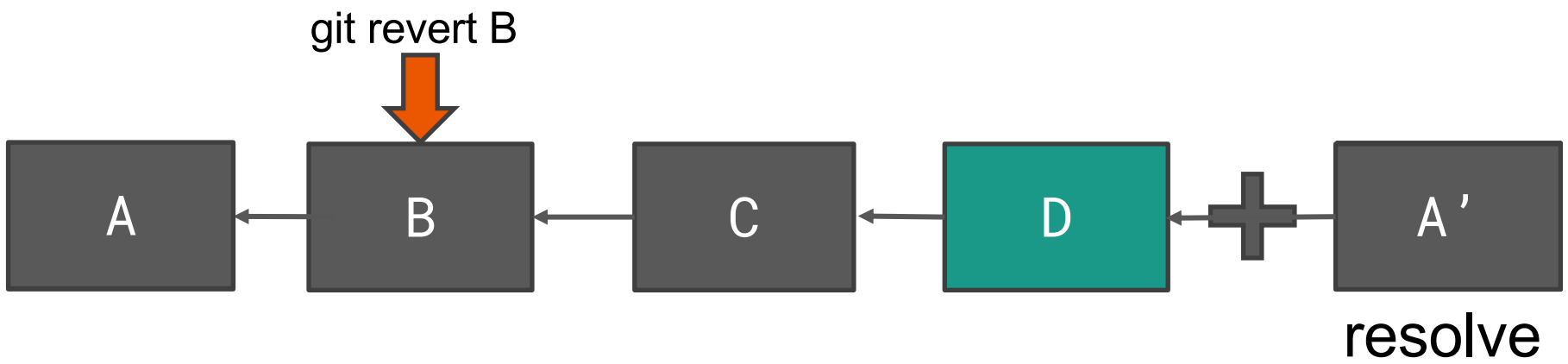

```
git reset --hard 08a3022
```

# Reset vs revert

# Tình huống

Tôi tạo ra commit A ← B ← C ← D. Tôi muốn lấy lại code của A nhưng không muốn xóa các commit B, C, D khỏi lịch sử.
Nếu dung git reset thì phũ quá.

Vậy hãy dùng **`$ git revert B`**



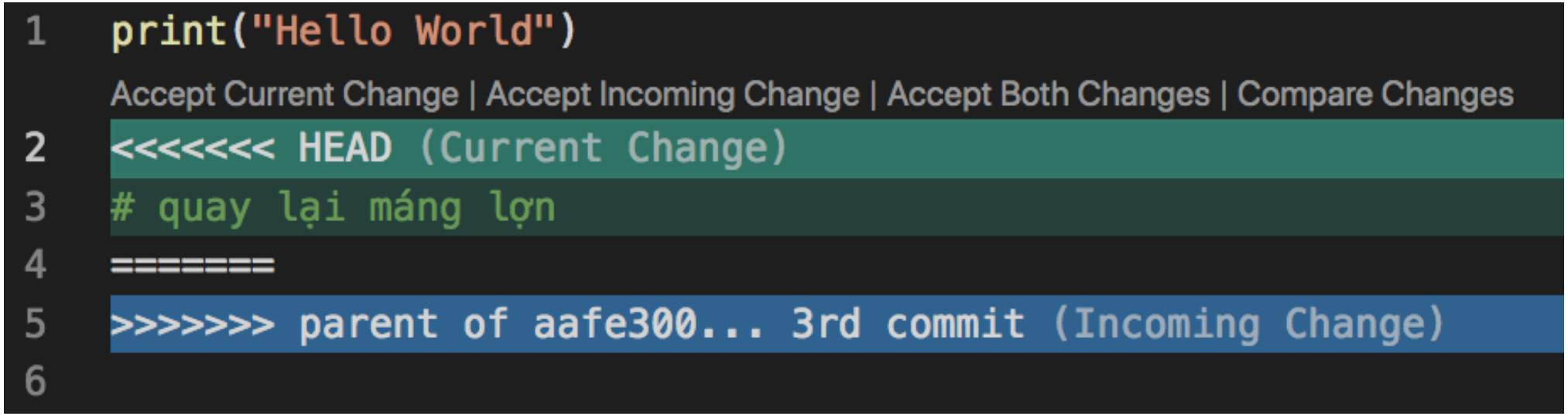
```
$ git revert commit_hash

$ git status

Resolve conflict

$ git add .

$ git revert --continue
```

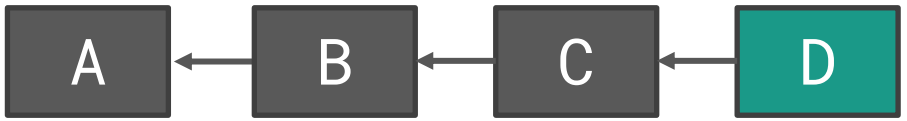



Ảnh chụp VSCode tại bước resolve conflict:

- Incoming change là nội dung của commit cũ A
- Current change là nội dung của commit gần nhất

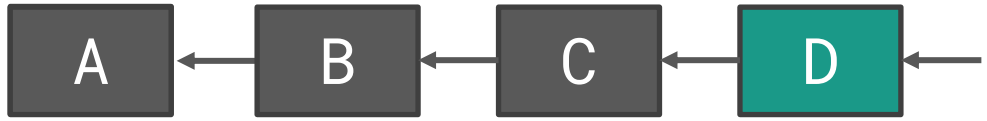
## reset

- Xóa các commit ra khỏi lịch sử của local repo
- Có thể gây lộn xộn, mất đồng bộ với remote repo

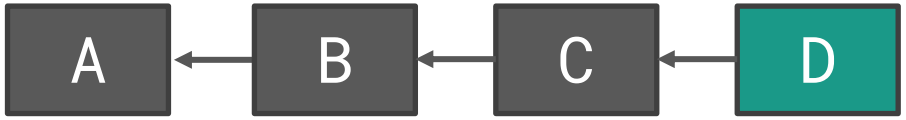



## revert

- Không xóa commit cũ, mà chỉ tạo commit mới có nội dung lấy từ commit cũ
- Cần phải resolve conflict
- Không gây xung đột với remote repo



# Xử lý trường hợp push nhầm




Nếu nhánh chỉ có 1 người làm
Giả sử commit D là commit lỗi

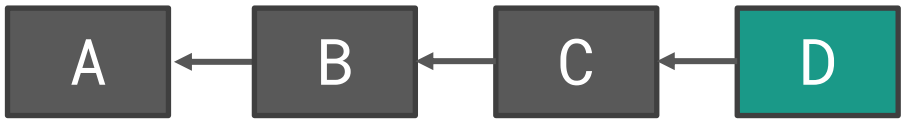
Cách 1: Xóa commit D và push lại:

```
$ git reset --hard commit_C
$ git push --force origin master
```

Cách 2: Giữ nguyên commit cũ, tạo commit mới sửa lại:

```
$ git revert commit_D
$ git push origin master
```

# Xử lý trường hợp push nhầm



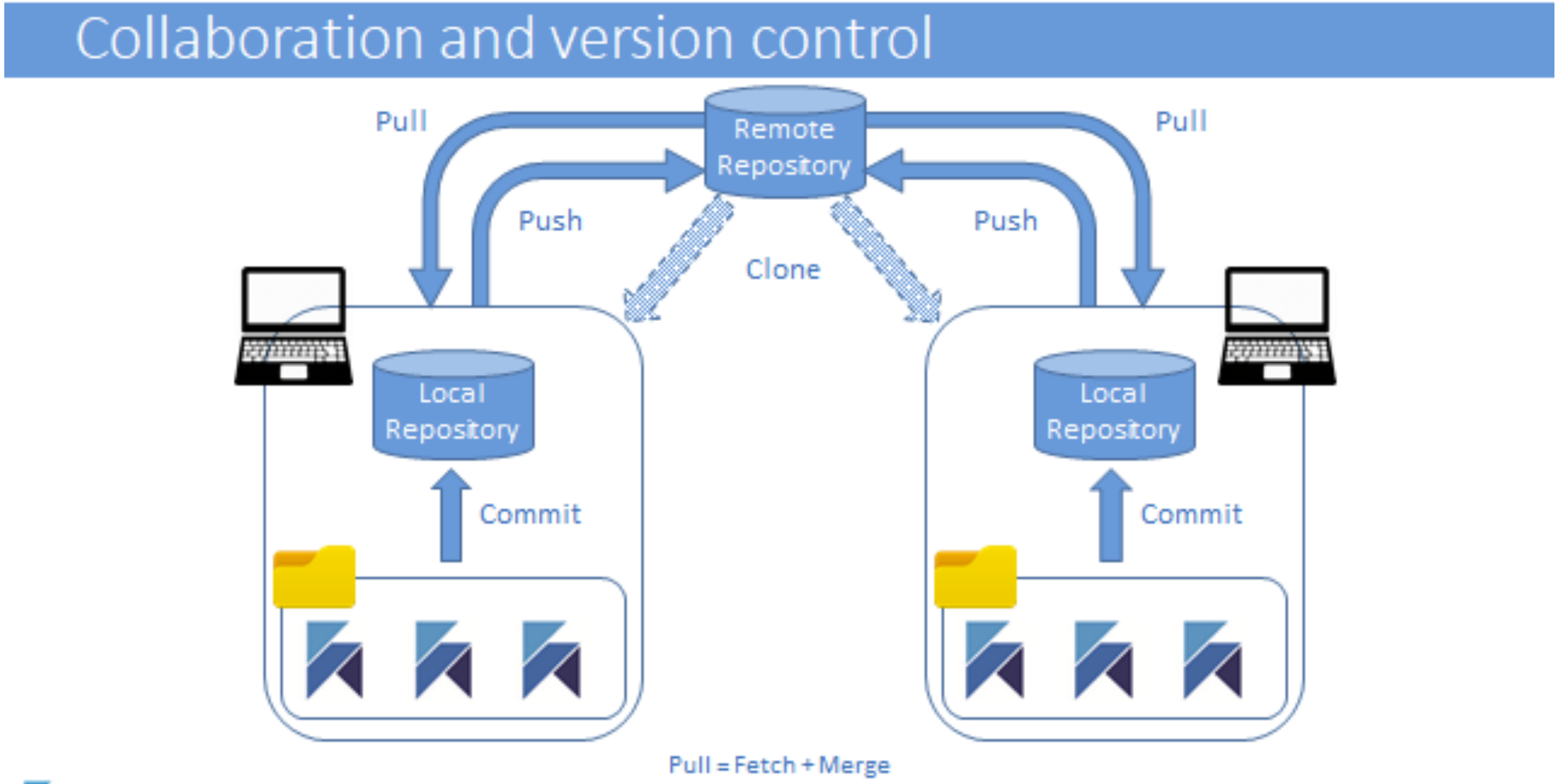

Nếu nhánh có nhiều hơn 1 người cùng làm
Giả sử commit D là commit lỗi

Không nên xóa commit D mà nên revert:

```
$ git revert commit_D
$ git push origin master
```

# Làm việc với Remote Repo

# Remote repo lưu giữ và đồng bộ hóa công việc giữa các dev



## Đẩy code từ local repo lên remote repo mới

1. Đặt tên thay thế (thường là ***origin***) cho địa chỉ của remote repo

```
$ git remote add origin remote_repo_URL
```

2. Đẩy code từ local repo lên remote repo

```
$ git push -u origin master
```

3. Liệt kê các remote repo

```
$ git remote -v
```

## Lấy code từ 1 remote repo có sẵn về máy

1. Download source code từ remote repo về máy

```
$ git clone remote_repo_URL
```

2. Repo lúc này đã có sẵn tên thay thế (thường là ***origin***) cho remote repo URL, do đó không cần git remote add nữa:

```
$ git push origin master
```

3. Liệt kê các remote repo

```
$ git remote -v
```

# Đồng bộ code từ remote repo về local repo

TH1: Nhiều dev code trên cùng 1 nhánh

- Ban đầu, 1 dev push nhánh của mình từ local repo lên remote repo
- Các dev còn lại cần lấy code từ remote repo về máy mình, sau đó mới push lên được

TH2: Mỗi dev code trên 1 nhánh riêng (feature-*), sau đó tích hợp code vào 1 nhánh chung (develop)

- Senior/Lead lần lượt kéo code của từng nhánh feature-* trên remote repo về nhánh develop trên máy mình
- Resolve conflict (nếu có) và đẩy nhánh develop từ local repo lên remote repo

Có 2 cách để lấy code từ remote về local repo: **git pull** và **git fetch**

## git pull

Lấy các commit của nhánh *origin/master* trên remote repo về local repo, sau đó merge nhánh *origin/master* vào nhánh *master*:

```
$ git pull origin master
```

## git fetch

Chỉ lấy các commit từ nhánh origin/master trên remote repo về local repo, nhánh master ở local repo không bị ảnh hưởng:

```
$ git fetch origin master
```

Dùng lệnh *git checkout* để kiểm tra nội dung commit *origin/master* trước khi merge:

```
$ git checkout origin/master
```

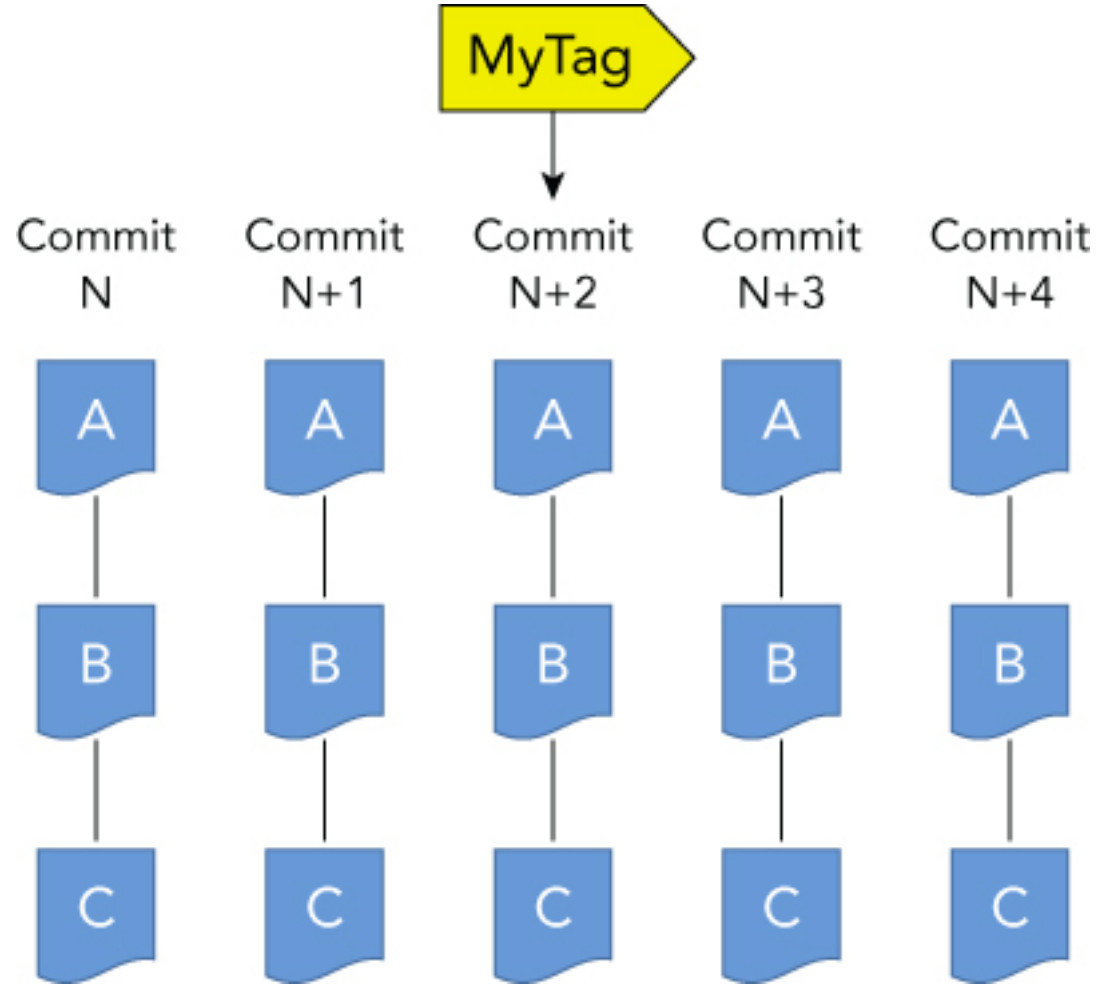
Kiểm tra xong thấy ok thì quay về nhánh *master* rồi merge nhánh *origin/master* vào *master*:

```
$ git checkout master
$ git merge origin/master
```

# Git tag

# Git tag

- Dán nhãn cho 1 commit object
- Thường dùng cho commit đánh dấu 1 cột mốc cụ thể trong dự án: SIT_v1, release_v1.0.1, production_v2.0.1
- Hỗ trợ truy vết lịch sử commit



# Cú pháp git tag

Đánh tag cho commit HEAD đang trỏ đến:

```
$ git tag –a tag_name -m ”message”
```

Đánh tag cho 1 commit bất kì:

```
$ git tag –a tag_name commit_sha -m ”message”
```

Liệt kê tất cả các tag

```
$ git tag
```

Xem thông tin 1 tag cụ thể

```
$ git show tag_name
```

# Branch-based workflow

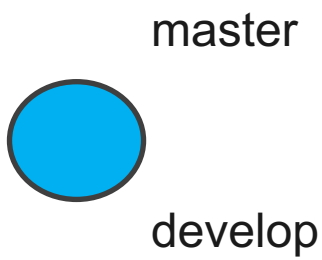
# Branch-based workflow

Tận dụng khả năng tạo nhánh dễ dàng của Git tạo các nhánh phục vụ cho những mục đích khác nhau:

- Các nhánh **feature-***: Tạo tính năng mới
- **develop**: tích hợp code từ các nhánh feature-*
- **release**: kiểm tra trước khi triển khai production
- **master**: triển khai production
- **hotfix**: fix gấp những lỗi trên production

# Tạo nhánh develop từ nhánh master

**$ git branch develop**



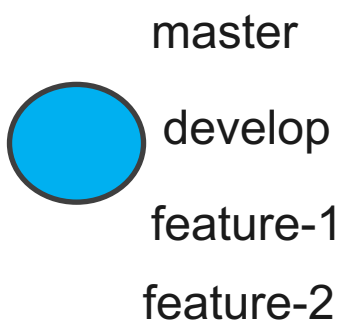

- Nhánh **develop** là nơi merge code từ các nhánh feature
- QA/QC sẽ test trên nhánh này

# Tạo nhánh feature từ nhánh develop

**$ git checkout develop**

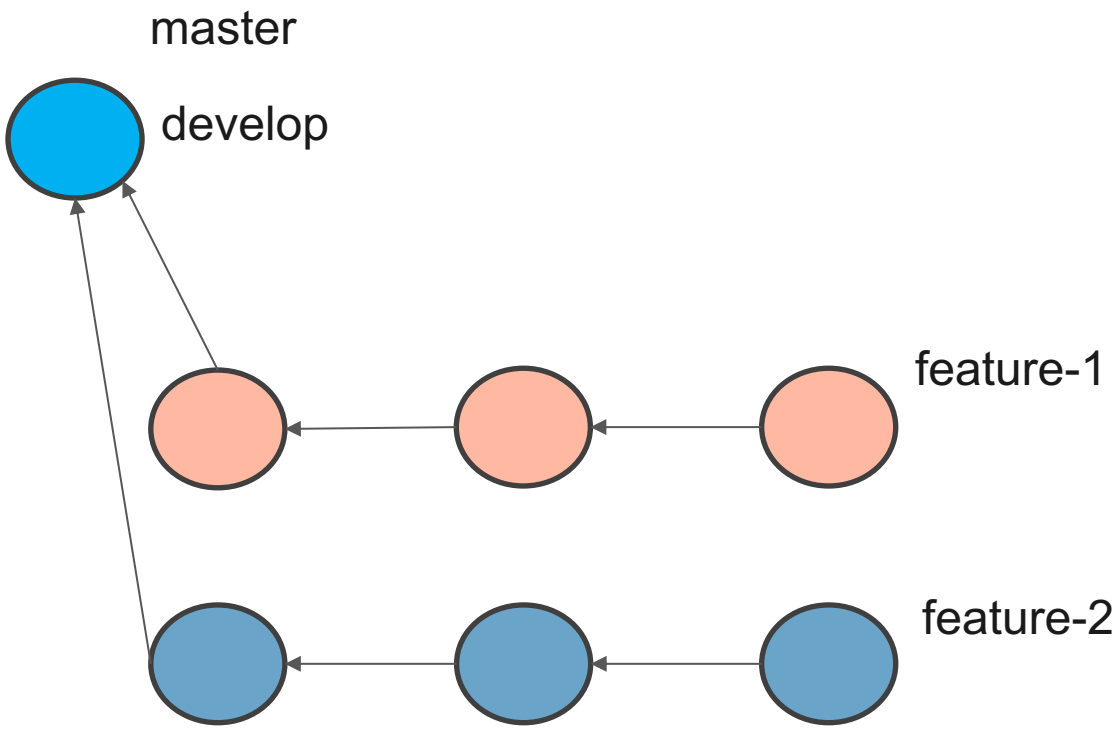
**$ git branch feature-1**

**$ git branch feature-2**



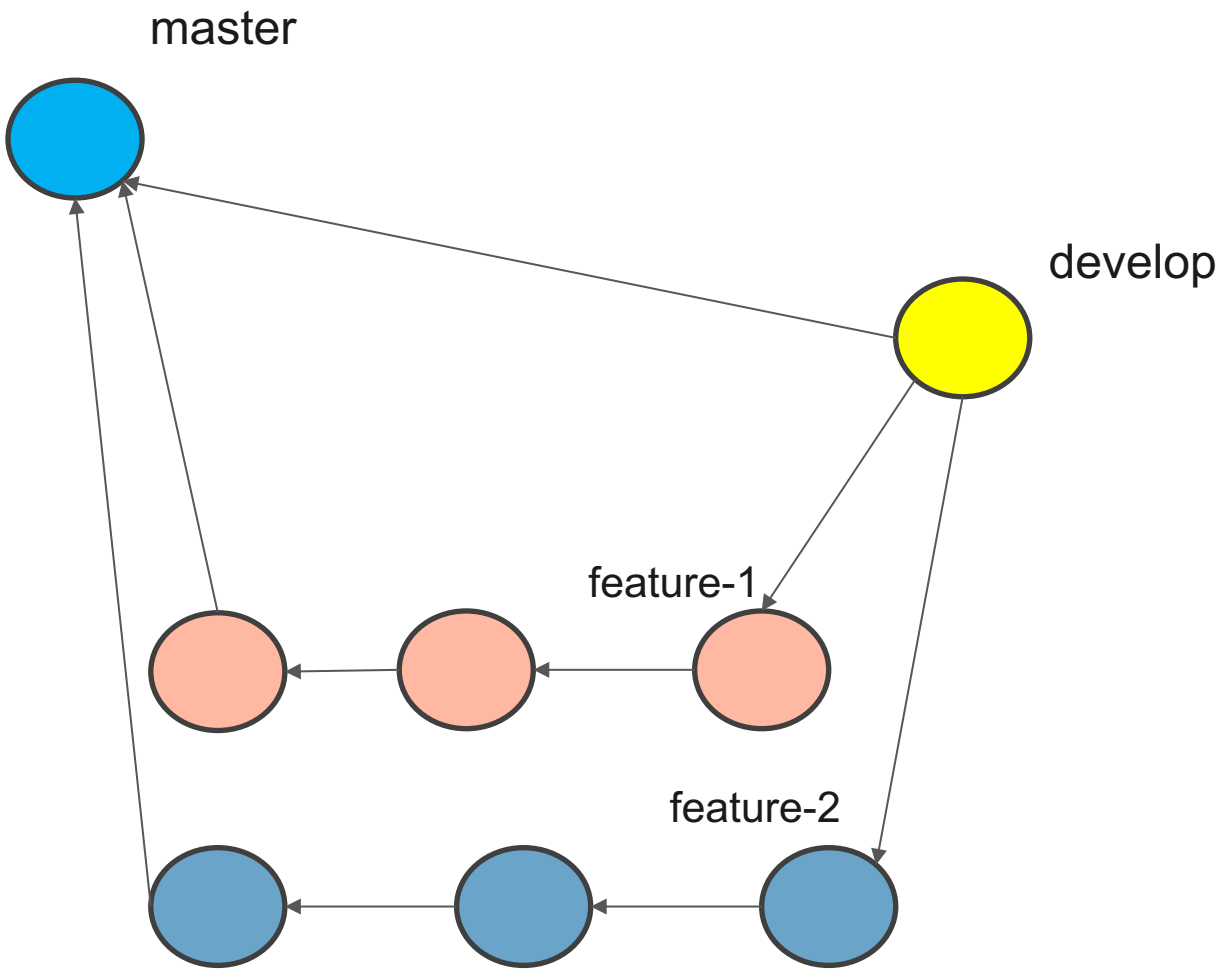

- Các nhánh **feature-*** là nơi dev tiến hành code các chức năng
- Phát sinh chức năng mới → tạo nhánh feature mới từ nhánh develop
- Sau khi code xong: merge nhánh feature vào nhánh develop

# Dev commit vào các nhánh feature




Các chức năng được phát triển độc lập trên từng nhánh feature-*

# Senior/Lead merge các nhánh feature vào develop




**$ git checkout develop**

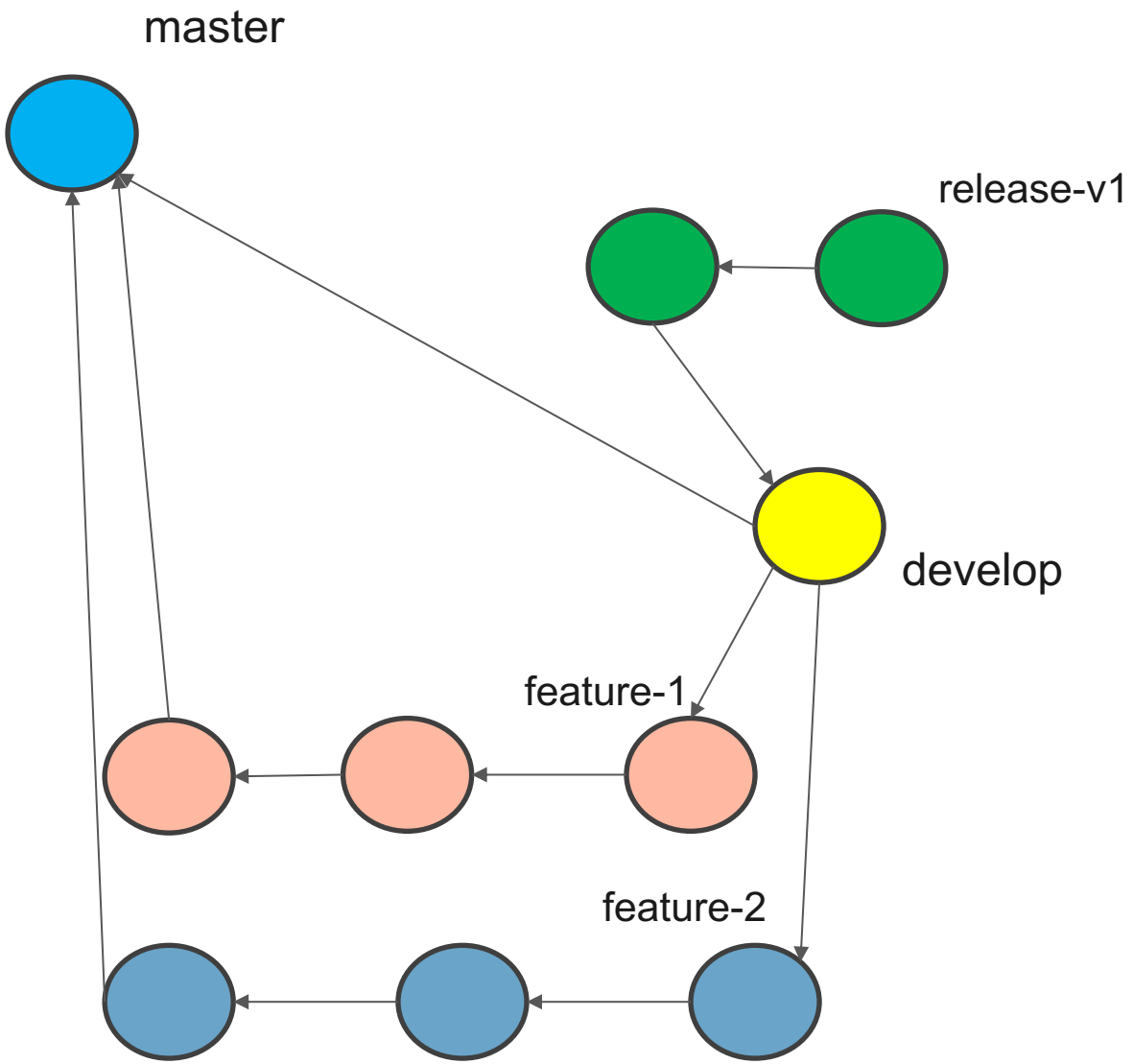
**$ git merge feature-1**

**$ git merge feature -2**

**$ resolve conflict (nếu có)**

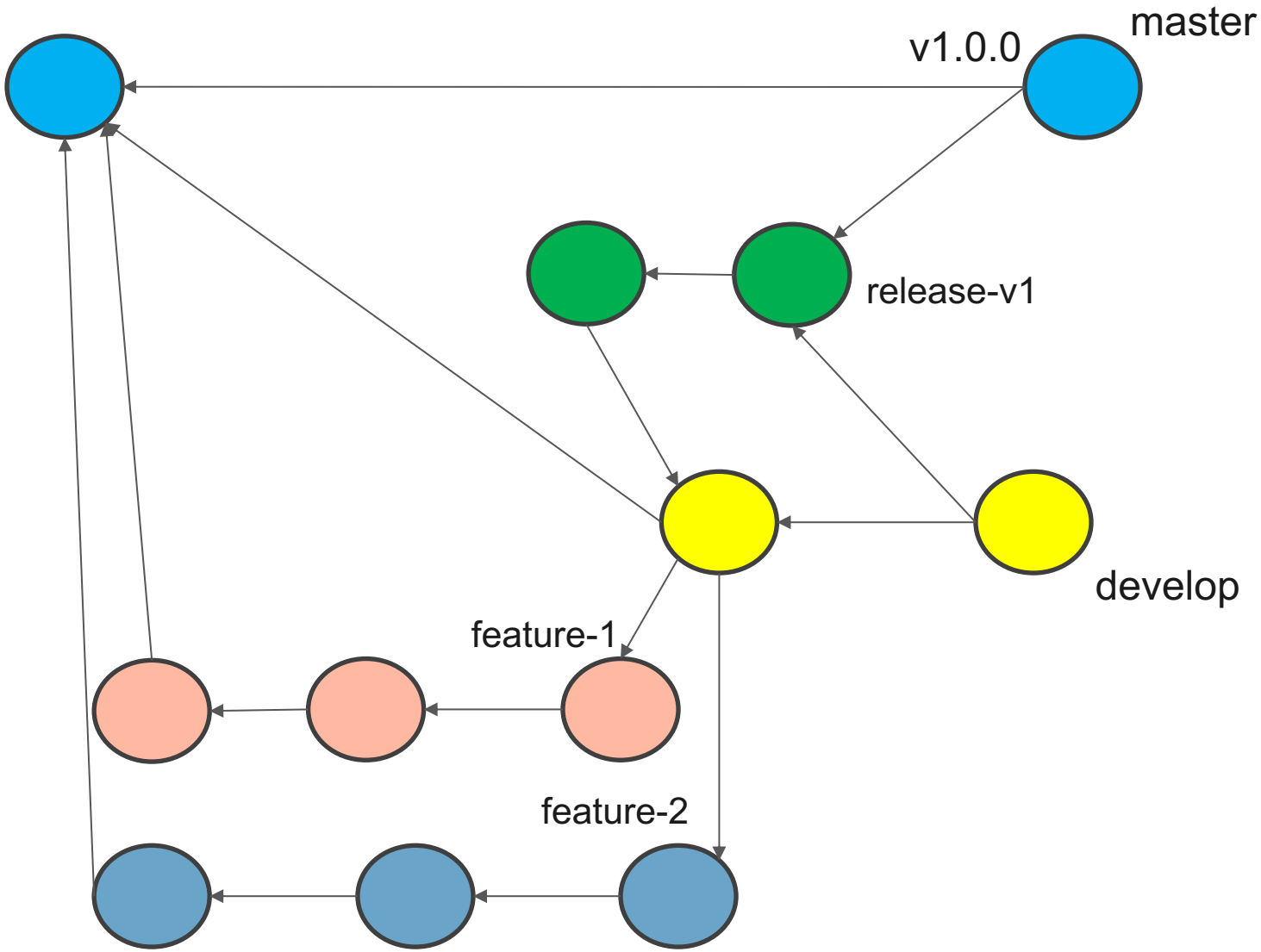
- QA/QC tiến hành test
- Nếu phát hiện lỗi: dev fix trên nhánh feature tương ứng, sau đó merge lại vào nhánh develop để test lại

# Tạo nhánh release từ develop để chuẩn bị release




- QA/QC test một lần nữa: Nếu phát hiện lỗi thì dev fix và *commit trực tiếp vào nhánh release*
- *Trên nhánh release chỉ có fix bug, không thêm tính năng mới*

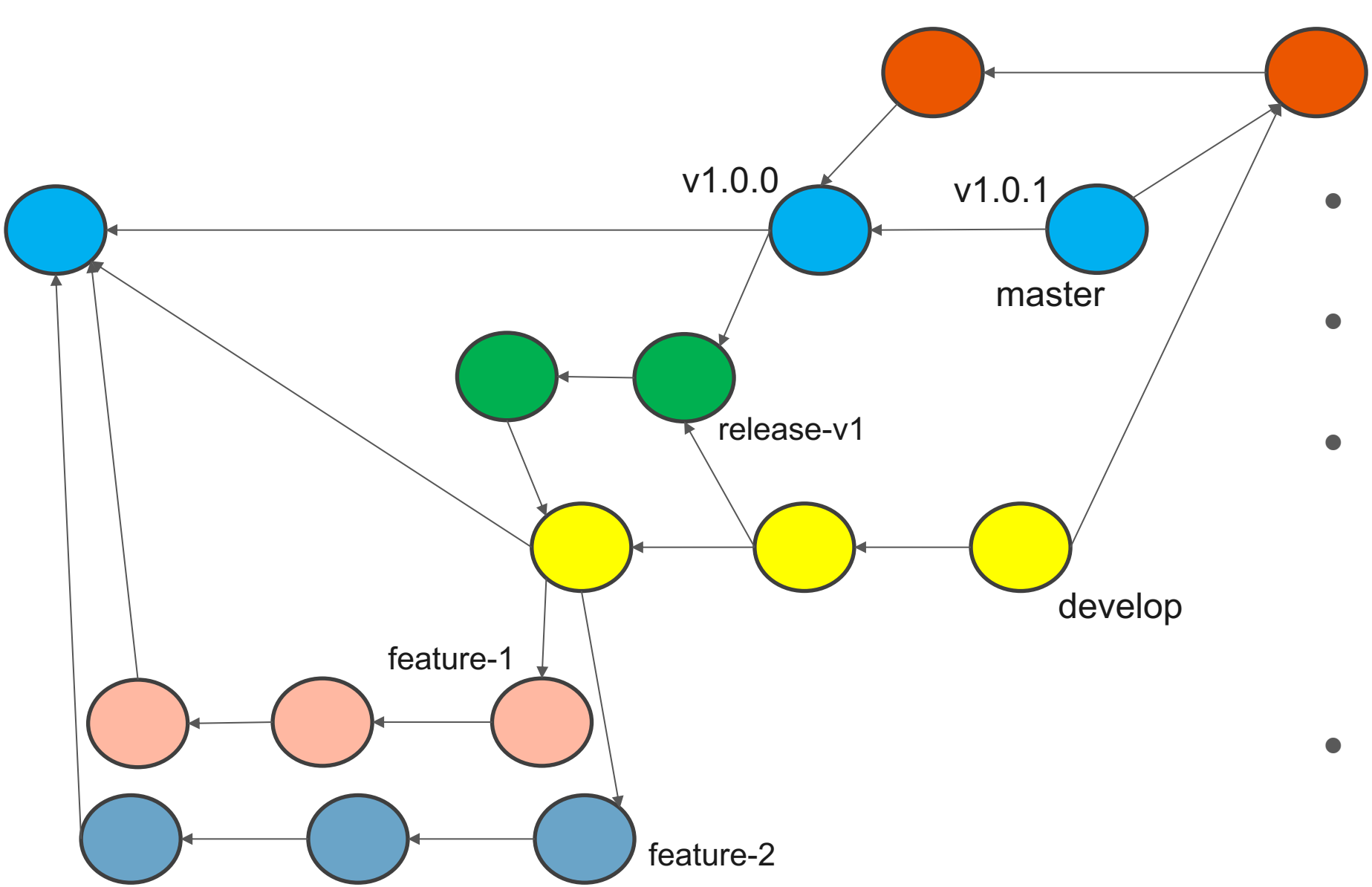
# Merge nhánh release vào master và develop




*Nếu code trên master chạy ổn định không lỗi thì merge nhánh release vào master và develop*

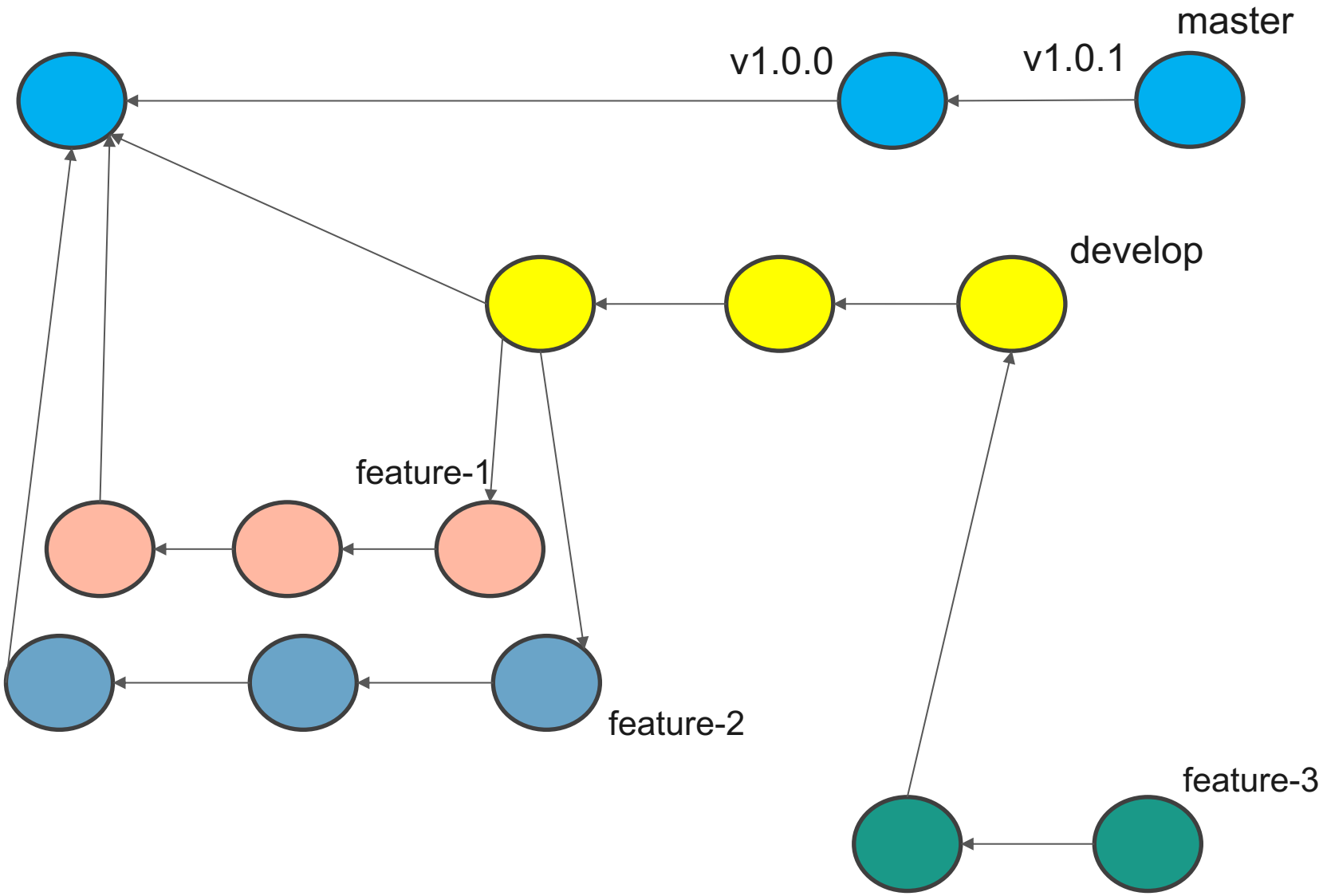
*Có thể đánh tag cho commit trên nhánh master, ví dụ: v1.0.0*

# Phát hiện lỗi cần fix gấp trên production




- *Tạo nhánh hotfix từ master*
- Dev fix và commit vào nhánh hotfix.
- *Fix xong thì merge lại vào nhánh master và develop, và xóa nhánh release-v1 và hotfix đi*
- *Cập nhật git tag trên nhánh master*

# Triển khai tính năng mới




- *Tạo nhánh feature-3 từ nhánh develop*
- *Dev commit vào nhánh feature-3 và lặp lại quy trình*