

# Git căn bản

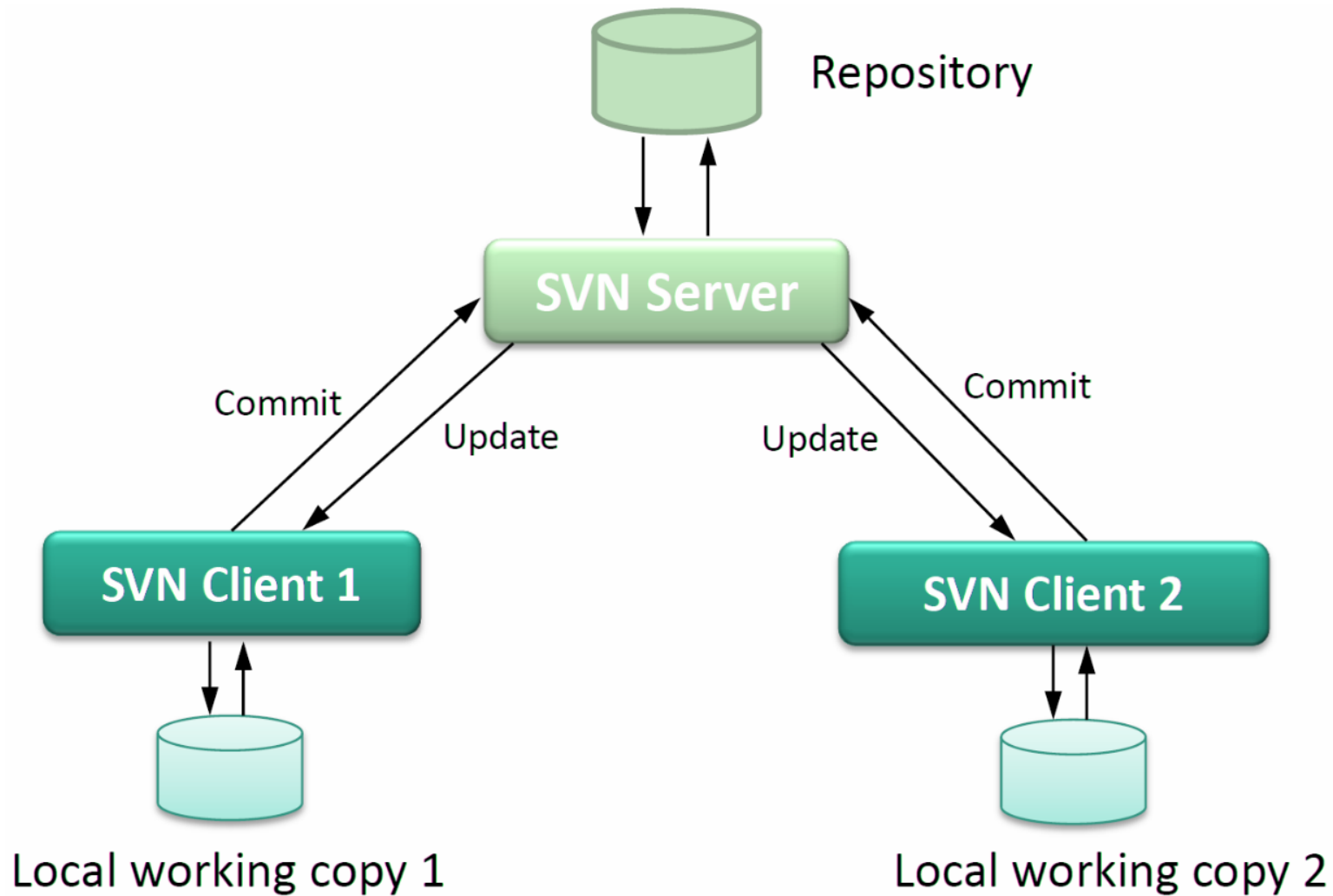
cuong@techmaster.vn

---

# Giới thiệu quản lý mã nguồn

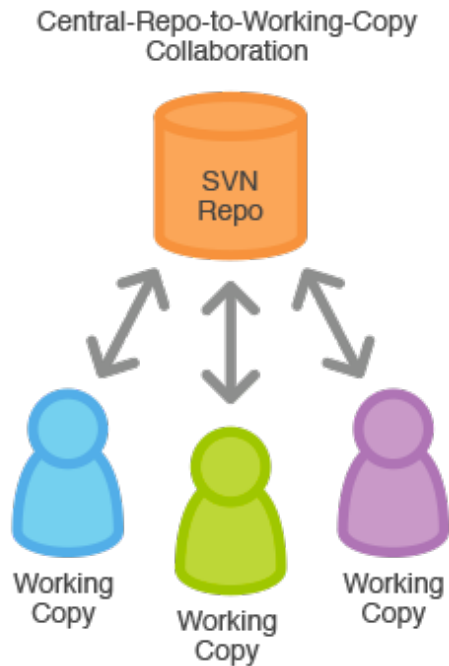
# Các cách quản lý mã nguồn

- **Sao lưu thư mục**, zip lại, đề tên theo ngày.
  - Ưu: ai cũng làm được, không cần tool, không cần học
  - Nhược: số lượng file tăng nhanh chóng, không cùng hợp tác code
- **Subversion**: quản lý mã nguồn mô hình client-server
  - Ưu: dễ hiểu, dễ dùng
  - Nhược: server chết, khởi dùng!
- **Git**: phương pháp quản lý mã nguồn phổ biến nhất
  - Ưu: phổ biến nhất, nhiều dịch vụ free, nhiều lựa chọn client-server. Server chết, chậm chờn, vẫn làm việc tốt
  - Nhược: tập lệnh phức tạp



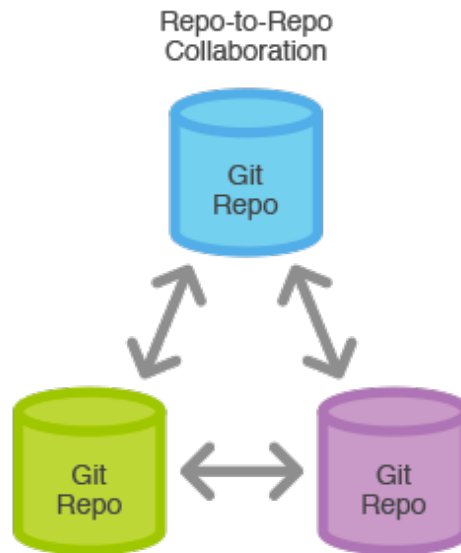
# Subversion

- Tập trung hóa lên SVN server
- Ở client chỉ là bản sao

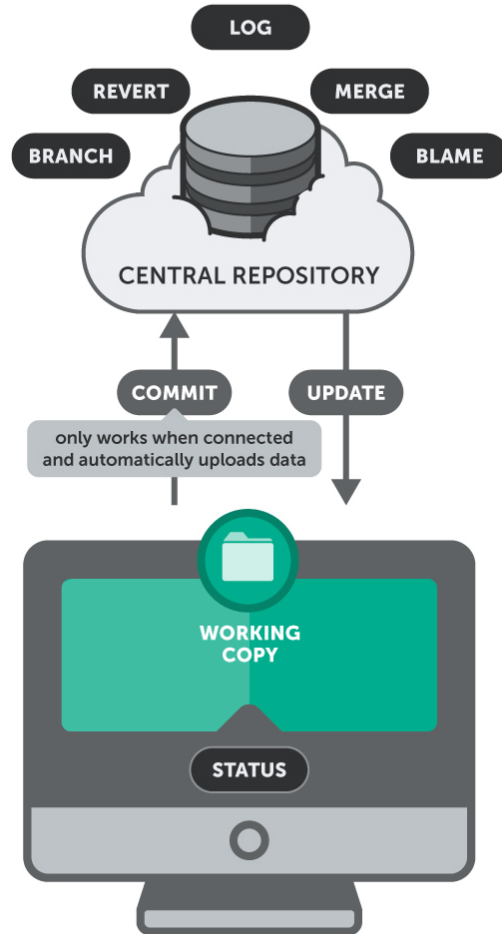


# Git

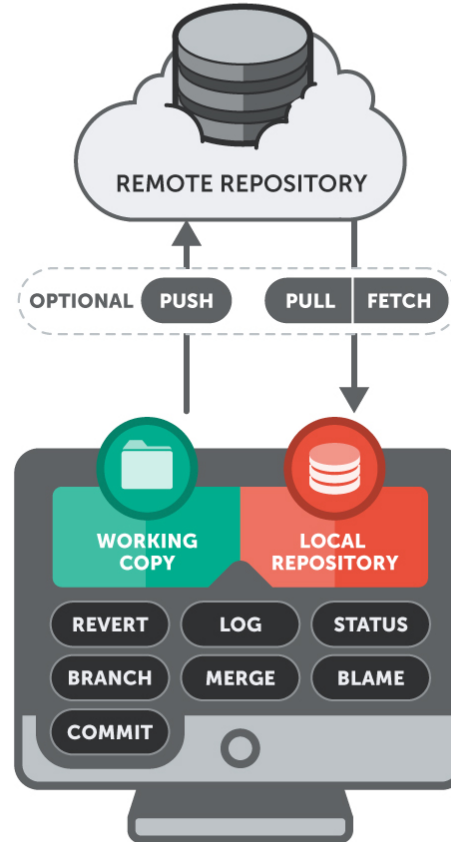
- Phân tán thành Git Repo:
  - Local Repository
  - Remote Repository



# SUBVERSION

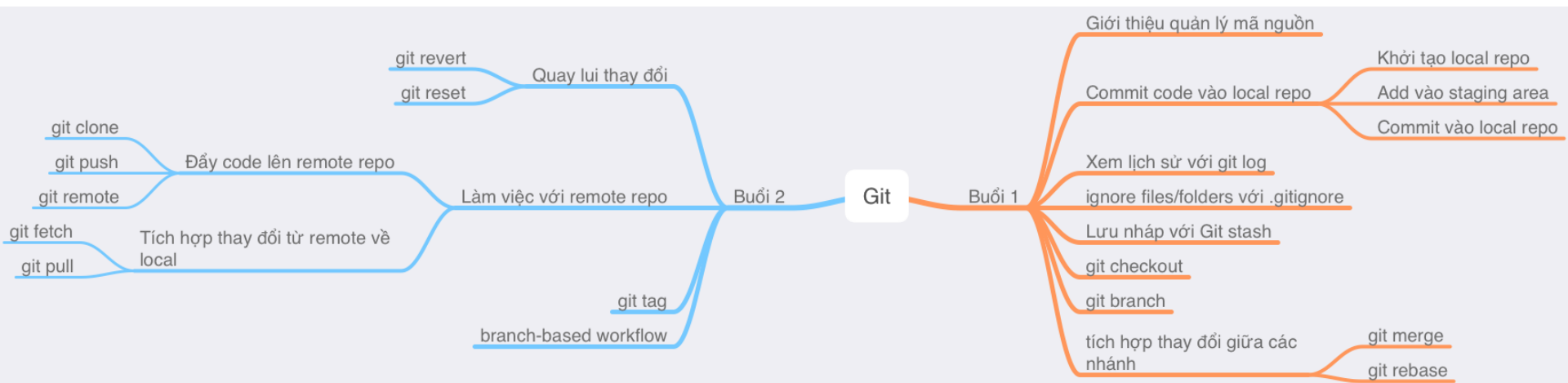


# GIT



# Tóm lại git là gì?

- Lưu vết lịch sử thay đổi mã nguồn ~ phiên bản
- Kết hợp quản lý mã nguồn tập trung sử dụng remote repo và phân tán ra các máy trạm local repo
- Cấu hình remote repo mềm dẻo, cho phép có nhiều remote repo





---

# Tạo local git repo

# Tình huống

- Bạn bắt đầu một dự án mới tình cần quản lý mã nguồn dự án này.
- Mã nguồn ban đầu chỉ có file ReadMe.md (định dạng Markdown)
- Sau đó thêm file demo.py (Python script)

```
## Tính diện tích tam giác
```

ReadMe.md

```
# Python Program to find the area of triangle
a = float(input('Enter first side: '))
b = float(input('Enter second side: '))
c = float(input('Enter third side: '))
# calculate the semi-perimeter
s = (a + b + c) / 2
# calculate the area
area = (s*(s-a)*(s-b)*(s-c)) ** 0.5
print('The area of the triangle is %0.2f' %area)
```

demo.py

# Tạo một local git repo

```
$ mkdir myweb
```

Tạo một thư mục

```
$ cd myweb
```

Chuyển vào thư mục đó

```
$ git init
```

Khởi tạo git repo

```
$ ls
```

Liệt kê nội dung thư mục kể cả file, folder ẩn

```
.git
```

Thấy ngay một thư mục ẩn .git

# Git repository là gì?

- Kho chứa các commit
- Mỗi commit lưu lại trạng thái của project tại 1 thời điểm nhất định
- Có 2 loại: local repo và remote repo
  - Local repo: repo lưu trên máy cá nhân của mỗi dev
  - Remote repo: repo lưu trên 1 server bên ngoài

# Thêm một file

\$ **nano** ReadMe.md

Tạo một file ReadMe.md bằng trình editor nano

\$ **git status**

Xem trạng thái của git repo hiện tại

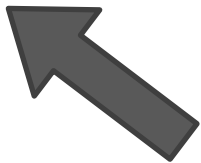
On branch master

No commits yet

Untracked files:

(use "git add <file>..." to include in what will be committed)

ReadMe.md



File mới tạo chưa được giám sát (untracked)

# Staging và commit file vào repo

```
$ git add .
```

Staging để giám sát thay đổi file này.  
Dấu . lấy tất nội dung trong thư mục hiện tại

```
$ git status
```

Kiểm tra trạng thái git repo

On branch master

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

new file: ReadMe.md

```
$ git commit -m "Tạo ReadMe.md"
```

Chốt thay đổi vào git repo

```
[master (root-commit) 31f9ba8] Tạo file ReadMe.md
```

```
1 file changed, 1 insertion(+)
```

```
create mode 100644 ReadMe.md
```

---

# Xem lịch sử thay đổi

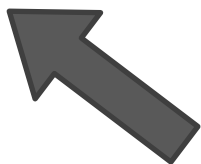
# \$ git log

commit 31f9ba8ca7572347121dacbad102ad9392edda45 (HEAD -> master)

Author: techmaster <cuong@techmaster.vn>

Date: Mon Oct 14 11:56:11 2019 +0700

Tạo file ReadMe.md  
(END)



commit message

*Chú ý: gõ vào terminal **:q** để thoát*



---

**Sửa đổi → commit**  
**Sửa đổi → hủy sửa đổi**

# Tình huống

User sửa file ReadMe.md, thêm vào vài dòng text đồng thời tạo mới file demo.py

# Thêm file mới và sửa file đã commit

```
$ git status
```

On branch master

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

new file: demo.py

File thêm mới

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: ReadMe.md

File sửa đổi

# Có 4 khả năng user muốn

1. Sửa nhầm file ReadMe.md, nên phải hủy thay đổi nhầm
2. Muốn commit thay đổi file ReadMe.md và thêm cả file mới tạo demo.py
3. Chỉ commit thay đổi file ReadMe.md, không commit file demo.py
4. Hủy tất cả thay đổi với ReadMe.md và demo.py, đưa thư mục làm việc về trạng thái lần commit trước đó

# TH1: Hủy thay đổi lên file ReadMe.md

```
$ cat ReadMe.md
```

```
Hello World
```

```
Thêm thay đổi
```

Xem nội dung file ReadMe.md

```
$ git restore ReadMe.md
```

Hủy thay đổi, đưa nội dung về lần commit gần nhất

```
$ cat ReadMe.md
```

```
Hello World
```

Xem nội dung file ReadMe.md để thấy dòng chữ sửa đổi đã mất

## TH2: staging cả ReadMe.md và demo.py

```
$ git add demo.py ReadMe.md
```

Cách này ghi rõ từng file

```
$ git status
```

On branch master

Changes to be committed:

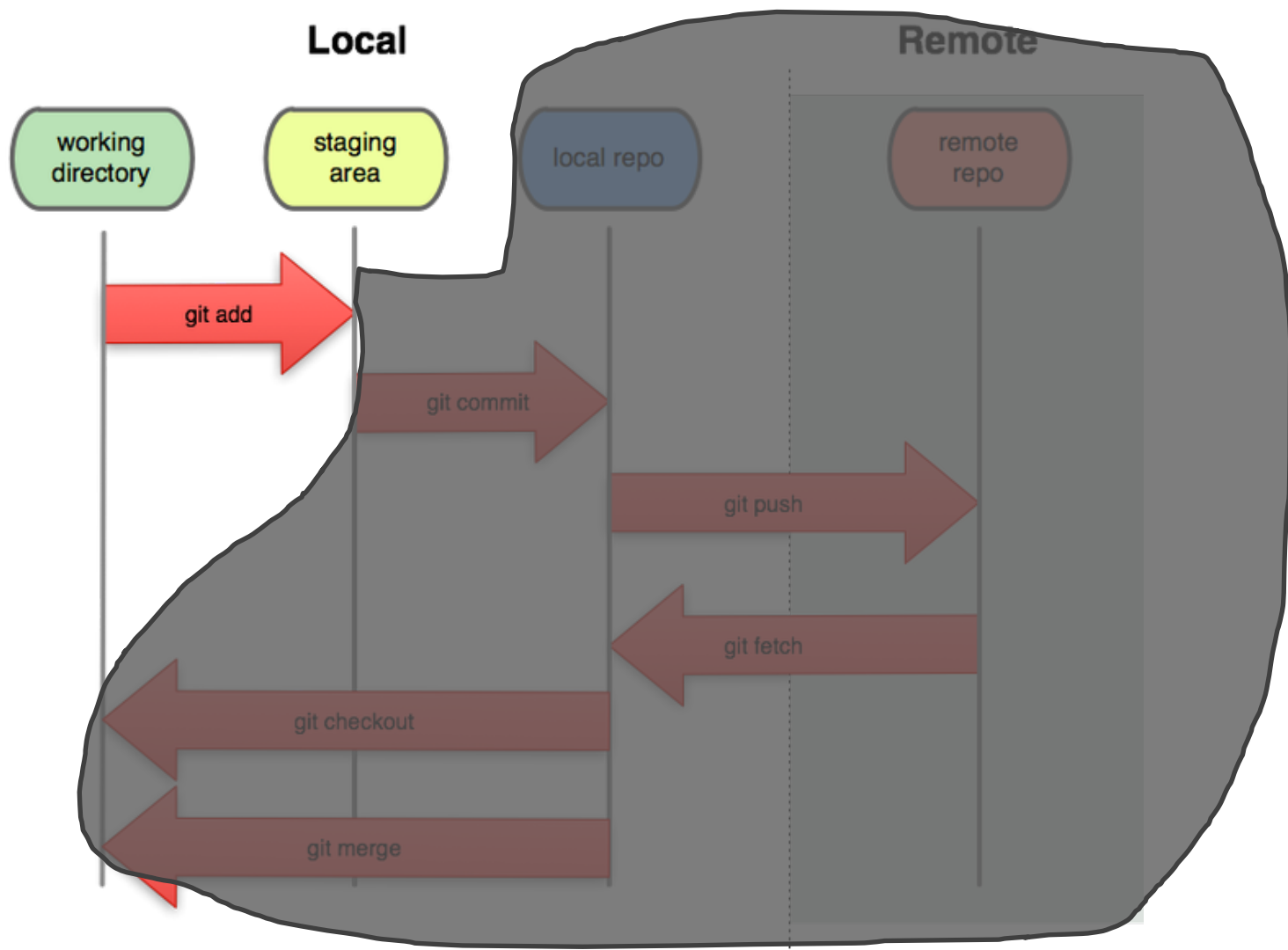
(use "git restore --staged <file>..." to unstage)

modified: ReadMe.md

new file: demo.py

```
$ git add .
```

Cách này stage tất cả thêm, sửa, xóa

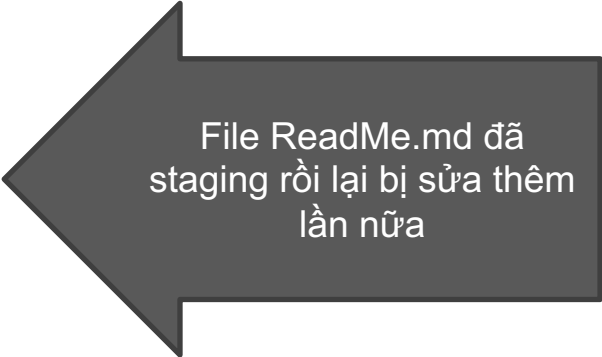


# Giữa working directory và local repo là staging area

- Working directory là thư mục mã nguồn dev đã sửa đổi
- Staging area là vùng đệm. Git chỉ commit những file/folder ở staging area

```
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   ReadMe.md
        new file:   demo.py

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   ReadMe.md
```



File ReadMe.md đã  
staging rồi lại bị sửa thêm  
lần nữa



## Hỏi: muốn hủy nội dung đã lên staging thì làm thế nào?

```
$ git restore --staged ReadMe.md
```

```
$ git status
```

On branch master

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

new file: demo.py

Staging  
area

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working  
directory)

modified: ReadMe.md

Working  
directory

## TH3: Chỉ commit thay đổi ReadMe.md bỏ qua demo.py

```
$ git status
```

```
On branch master
```

```
Changes to be committed:
```

```
(use "git restore --staged <file>..." to unstage)
```

```
    modified:   ReadMe.md
```

```
    new file:   demo.py
```

```
$ git restore --staged demo.py
```

Loại demo.py ra khỏi staging

```
$ git status
```

```
On branch master
```

```
Changes to be committed:
```

```
(use "git restore --staged <file>..." to unstage)
```

```
    modified:   ReadMe.md
```

```
Untracked files:
```

```
(use "git add <file>..." to include in what will be committed)
```

```
demo.py
```

## TH4: Hủy tất cả thay đổi, kể cả staging area đưa working directory về nội dung commit mới nhất

```
$ git status
```

```
On branch master
```

```
Changes to be committed:
```

```
(use "git restore --staged <file>..." to unstage)
```

```
    modified:   ReadMe.md
```

```
    new file:   demo.py
```

```
$ git reset --hard
```

```
HEAD is now at 31f9ba8 Tạo file ReadMe.md
```

```
$ git status
```

```
On branch master
```

```
nothing to commit, working tree clean
```

---

**.gitignore loại file, thư mục  
không cần quản lý phiên bản**

# Tình huống

Hãy tạo thêm một thư mục temp cùng một số file bên trong và một file junk.csv ở ngoài

```
$ mkdir temp  
$ touch temp/1.csv  
$ touch temp/2.csv  
$ touch junk.csv  
$ tree .
```

```
.  
├── ReadMe.md  
├── junk.csv  
└── temp  
    ├── 1.csv  
    └── 2.csv
```

1 directory, 4 files

# Không muốn quản lý thư mục temp và file tạm junk.csv thì làm sao?

Hãy tạo một file .gitignore có nội dung như sau

```
temp/  
junk.csv
```

```
$ git status
```

```
On branch master
```

```
Untracked files:
```

```
  (use "git add <file>..." to include in what will be  
  committed)
```

```
    .gitignore
```

Giờ thì thư mục temp và file junk.csv đã bỏ qua trong các lệnh git add

# .gitignore có đặc điểm gì?

- Hỗ trợ wild card \*.file\_extension
- Có thể tạo .gitignore trong thư mục con để loại bỏ chi tiết hơn: Rule của file .gitignore trong thư mục con sẽ có mức độ ưu tiên cao hơn so với .gitignore ở thư mục cha
- Dùng để bỏ qua những file quá lớn, những file tạm không có ý nghĩa trong việc quản lý phiên bản

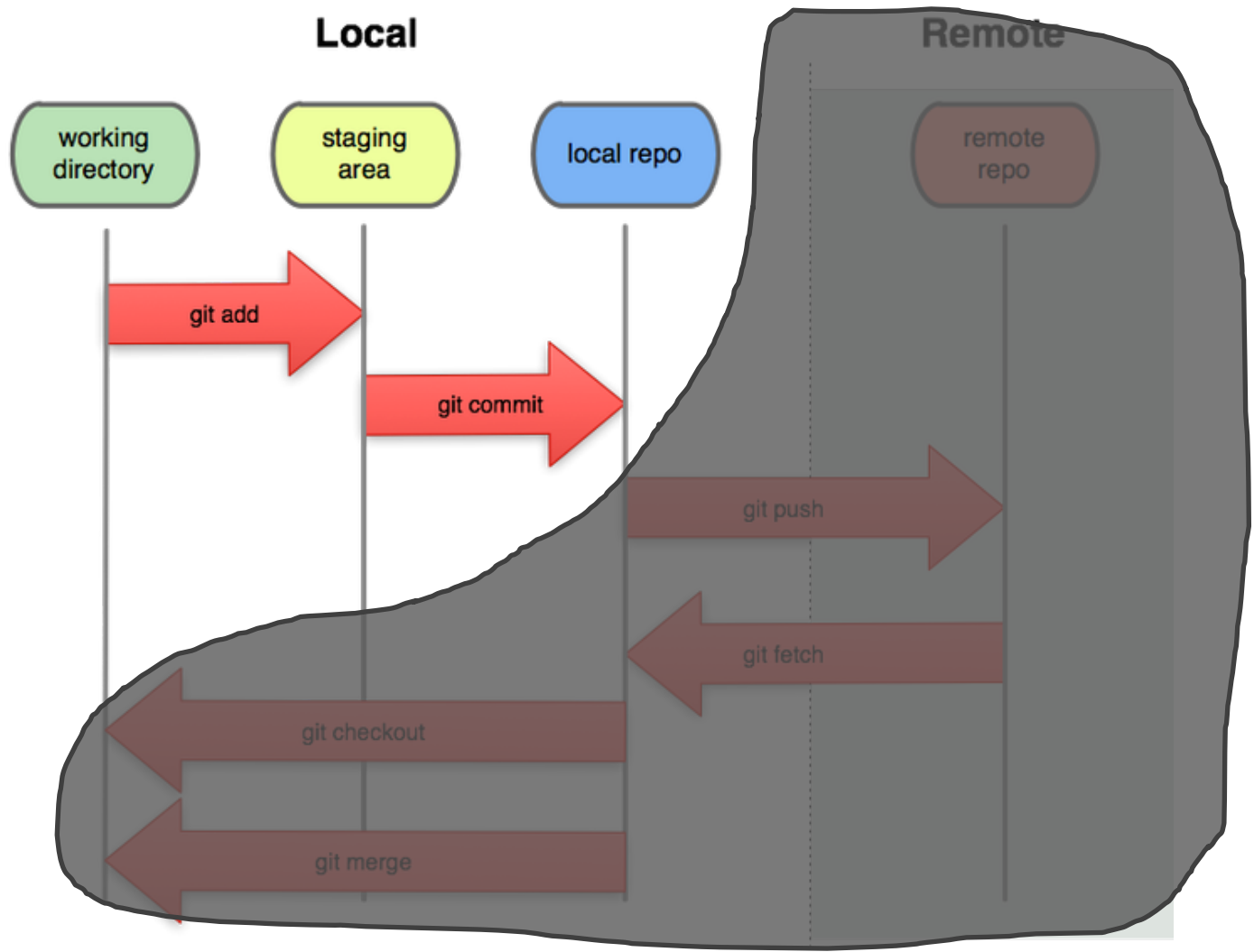
---

# git commit



# Tình huống

- Ở phần thực hành trước chúng ta mới làm việc ở working directory và staging area, giờ hãy tìm hiểu tiếp commit vào local repo
- Commit là một tập các thay đổi được người chốt hạ thành một phiên bản
- Staging có thể quay lui, thì commit cũng có thể quay lui



# Commit lần 2 vào local repo

```
$ git add .
```

```
$ git commit -m "Fix issue #JIRA-333"
```

```
[master be16b94] 2nd commit  
2 files changed, 3 insertions(+)  
create mode 100644 .gitignore  
create mode 100644 demo.py
```

# Commit best practice

- Mỗi lần commit tập trung vào một nhiệm vụ: fix 1 bug, refactor một số đoạn code, thêm một chức năng gọi tên cụ thể
- Không nên tạo big commit gồm quá nhiều thay đổi
- Không đặt tên commit message vô nghĩa
  - "fix something" → "fix issue #2301"
  - "Add some file" → add ReadMe.md, toc.md
  - "Make big change" → "refactor Person.go"

---

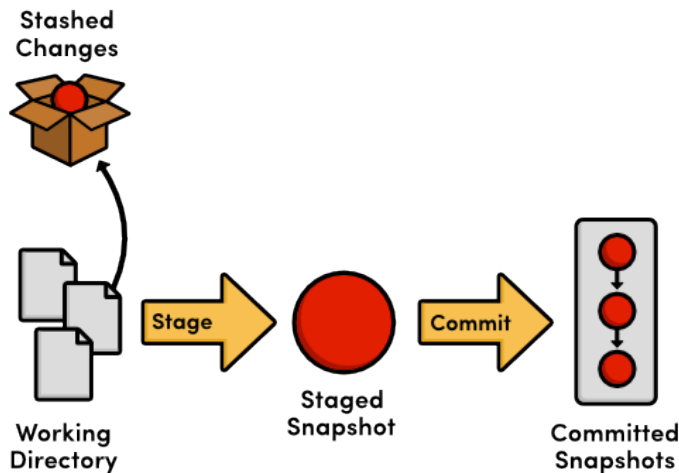
# Git stash

# Khi nào dùng git stash

Cần lưu công việc đang làm dở vào Git trước khi:

- Tắt máy đi về 😅 😅 😅
- Chuyển sang nhánh khác (có bug cần fix gấp trên nhánh hot-fix)

*Không lưu công việc đang làm dở vào commit*



# Sử dụng git stash

Lưu công việc hiện tại đang làm dở, đưa working directory về commit gần nhất:

```
$ git stash save "Đang code dở trang chủ"
```

Lưu cả các untracked files:

```
$ git stash save --include-untracked "code thêm trang nữa"
```

Xem danh sách các stash (Last-In-First-Out, stash mới nhất trên đầu)

```
$ git stash list
```

Lấy nội dung 1 stash (ví dụ stash gần nhất) đổ vào working directory để code tiếp:

```
$ git stash apply stash@{0}
```

Lấy stash gần nhất đổ vào working directory đồng thời xóa luôn stash đó đi:

```
$ git stash pop
```

---

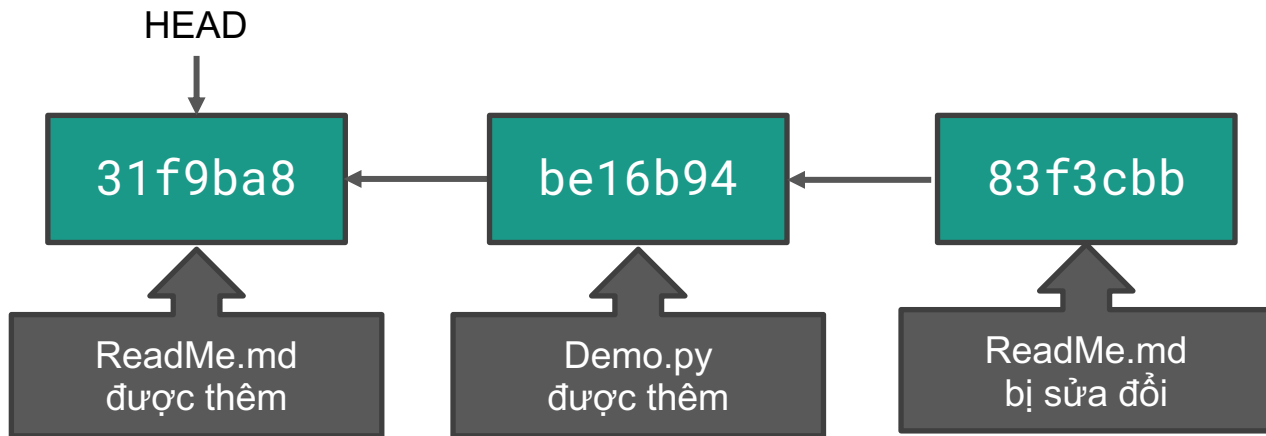
# git checkout



# Dùng git checkout để làm gì?

- Từ “check out” có thể hiểu là lấy ra để kiểm tra
- Lật lại mã nguồn trong quá khứ, tìm nguyên nhân gây lỗi
- Chuyển nhánh

# Ôn lại quá trình commit



Mỗi commit được gắn với một chuỗi ký tự unique – gọi là hash string

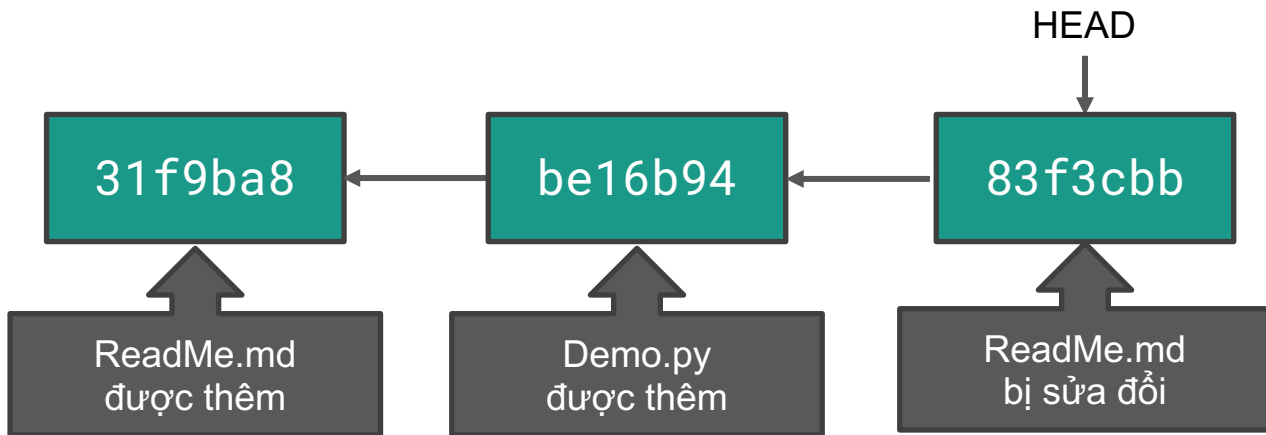
Thực ra chuỗi này dài nhưng khi gõ `git log --oneline` ta chỉ thấy chuỗi rút gọn

Để di chuyển, tìm lại nội dung mã nguồn tại một thời điểm trong quá khứ, chúng ta cần đến chuỗi rút gọn này.

# git checkout chuyển đến commit cụ thể

Nội dung working directory mới nhất sẽ bị thay thế bằng nội dung của working directory tại thời điểm commit git trước đây. Nhưng file/folder bị bỏ qua trong .gitignore không bị thay đổi

```
$ git checkout 31f9ba8
```



# Quay về commit mới nhất

Sau xem xét, tìm hiểu commit cũ có 2 khả năng:

1. User không thay đổi gì và muốn đưa working folder về lần commit gần đây nhất

**\$ git checkout master**

2. User thay đổi nội dung file → có 2 khả năng:

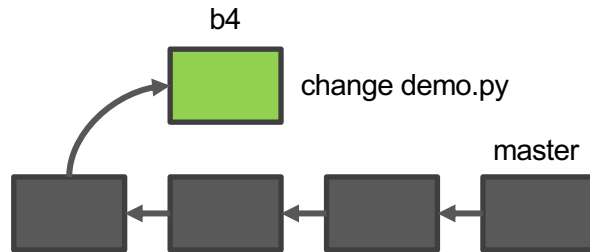
- a. Hủy thay đổi này: **\$ git restore .**

- b. Commit thay đổi này, rõ ràng không thể vào nhánh hiện tại mà phải rẽ nhánh

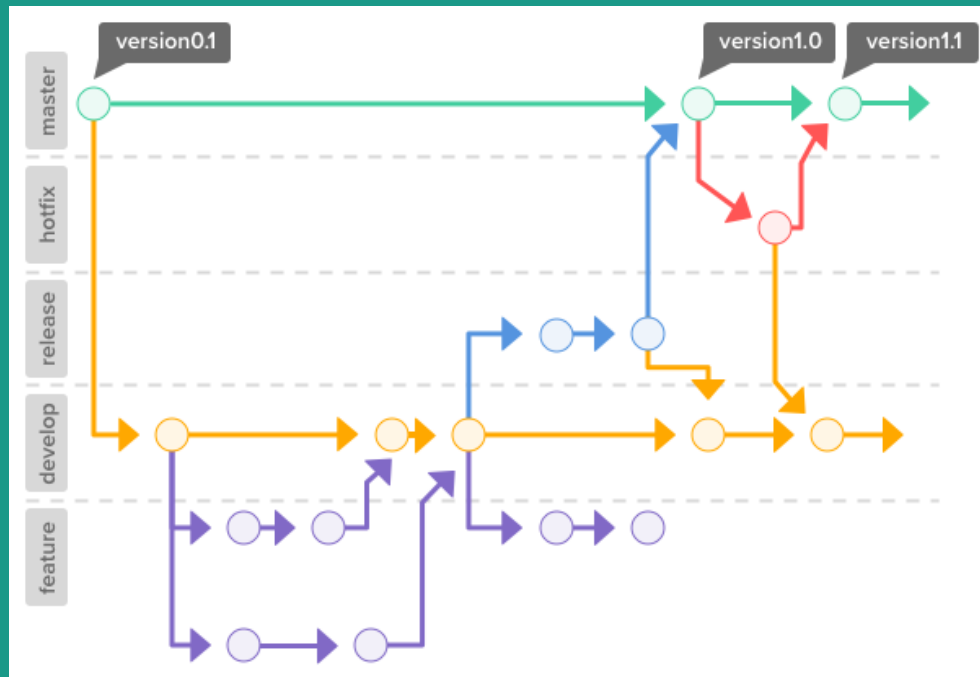
**\$ git branch -c b4**

**\$ git add .**

**\$ git commit -m "change demo.py"**

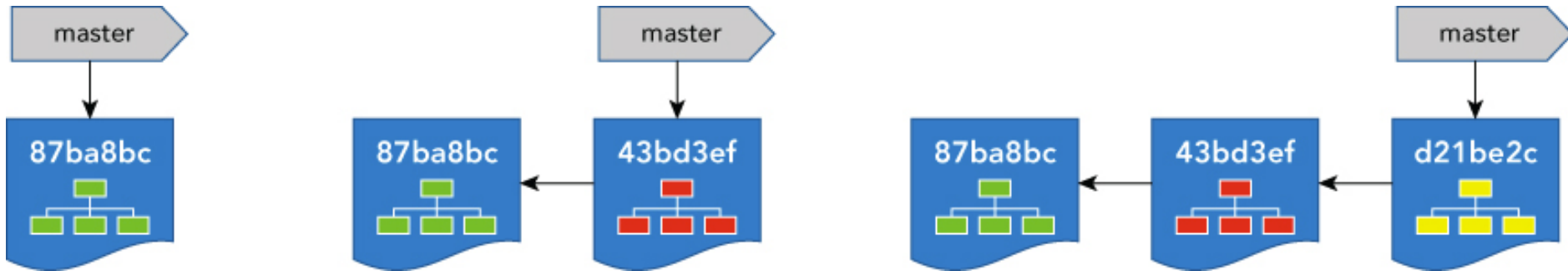


# Git branch



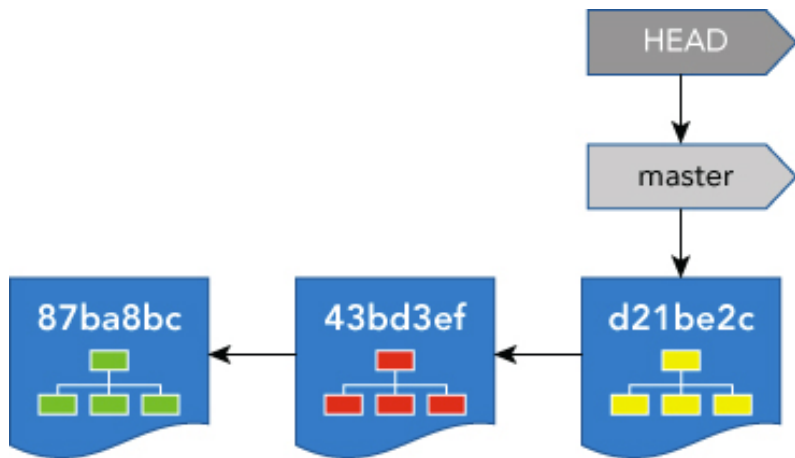
# Khái niệm

- Bản chất branch chỉ là 1 pointer trỏ đến 1 commit
- Mỗi lần thực hiện commit: git cập nhật vị trí pointer trỏ đến commit mới

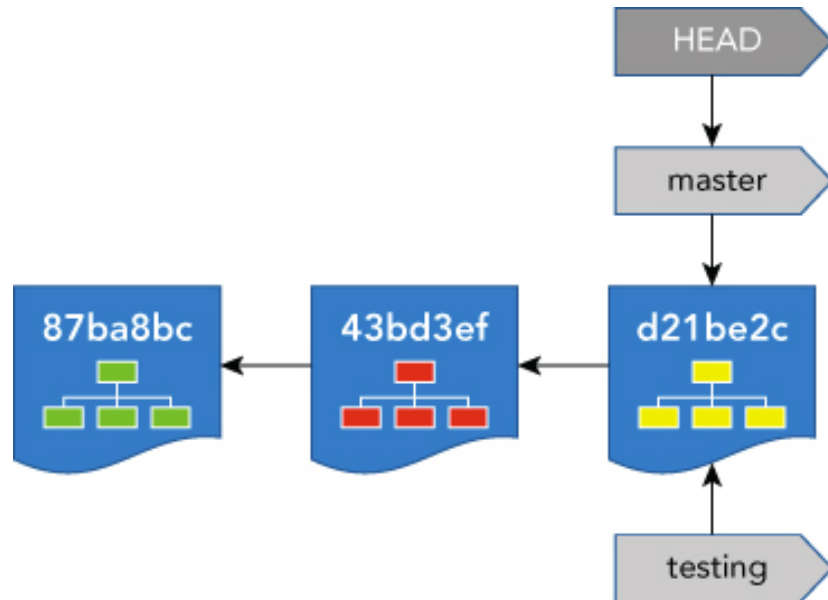


# Tạo nhánh

Trước khi tạo nhánh



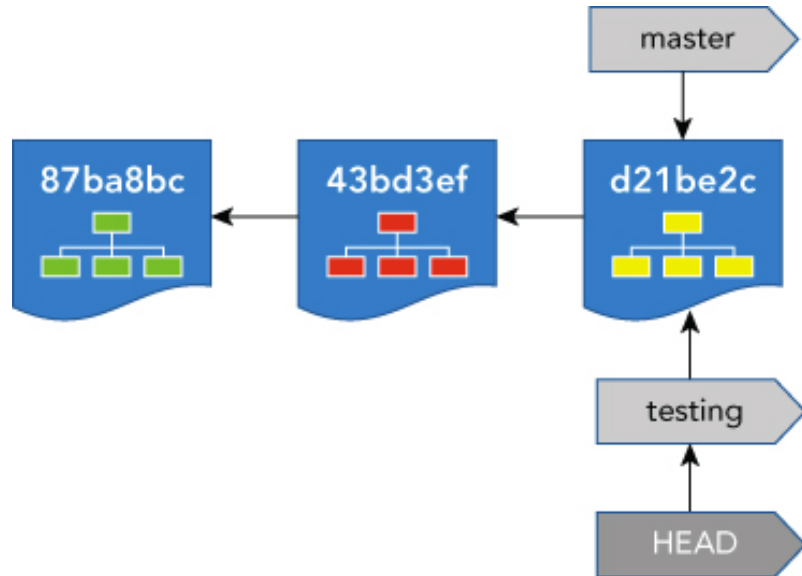
Tạo nhánh testing:  
\$ `git branch testing`



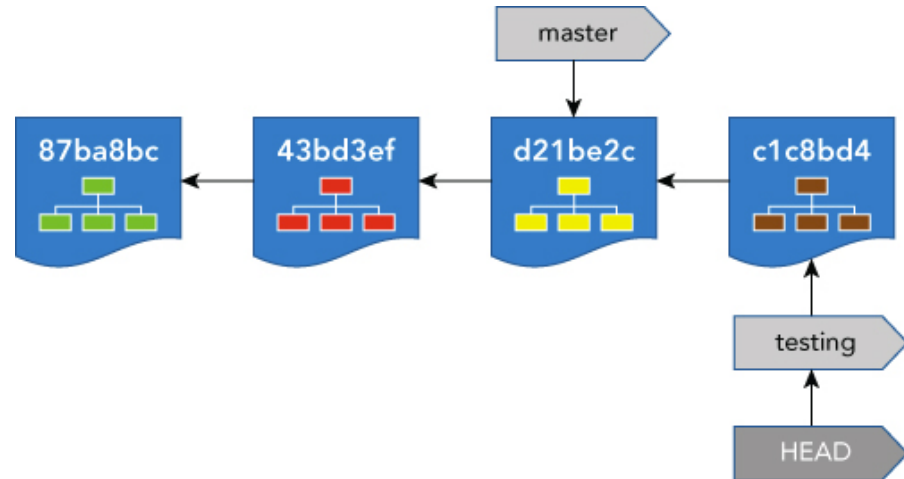
# Chuyển nhánh và commit vào nhánh mới

Chuyển nhánh:

`$ git checkout testing`



Commit vào nhánh testing

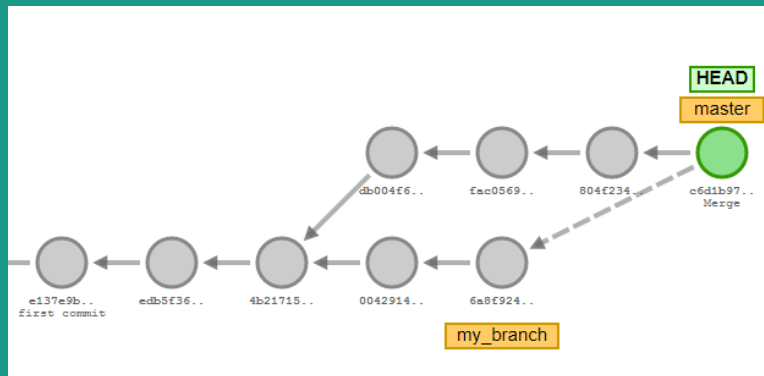




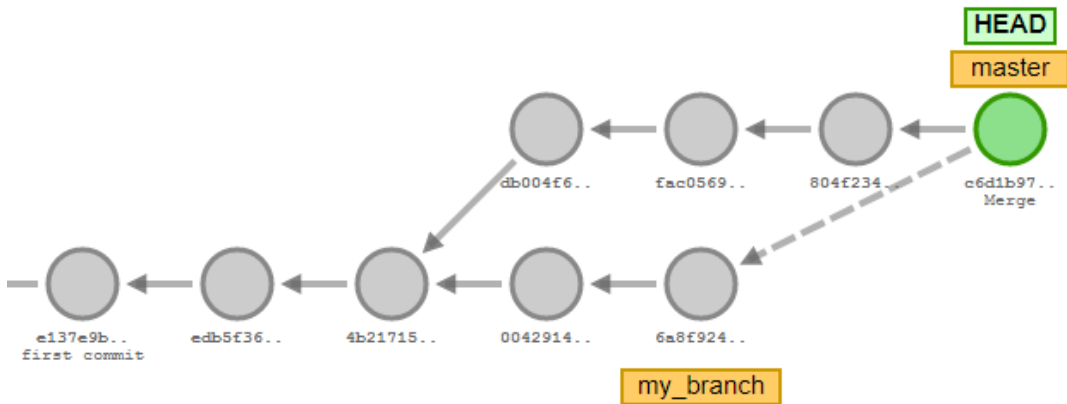
# Git branch cheatsheet

<https://devhints.io/git-branch>

# Git merge



# git merge tích hợp code từ nhánh A vào nhánh B



Tích hợp code nhánh  
my\_branch vào master:

```
$ git checkout master
```

```
$ git merge my_branch
```

# Lưu ý trước khi git merge

Đảm bảo working directory clean:

- Commit code vào local repo

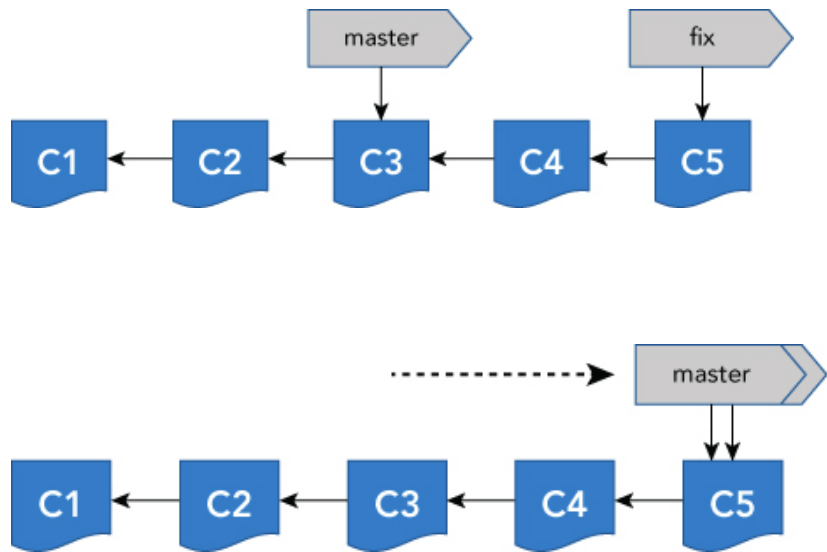
```
$ git commit -m "commit message"
```

- Hoặc lưu nháp với git stash

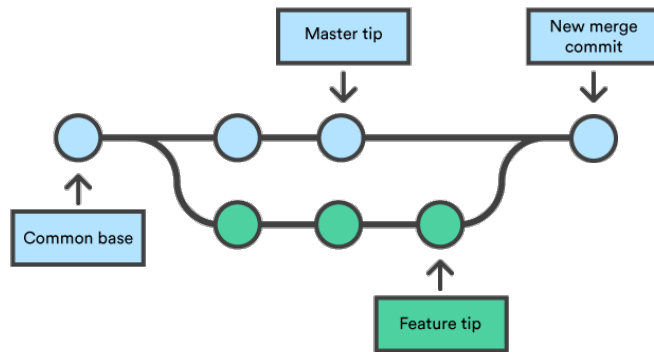
```
$ git stash save
```

# Có 2 kiểu git merge

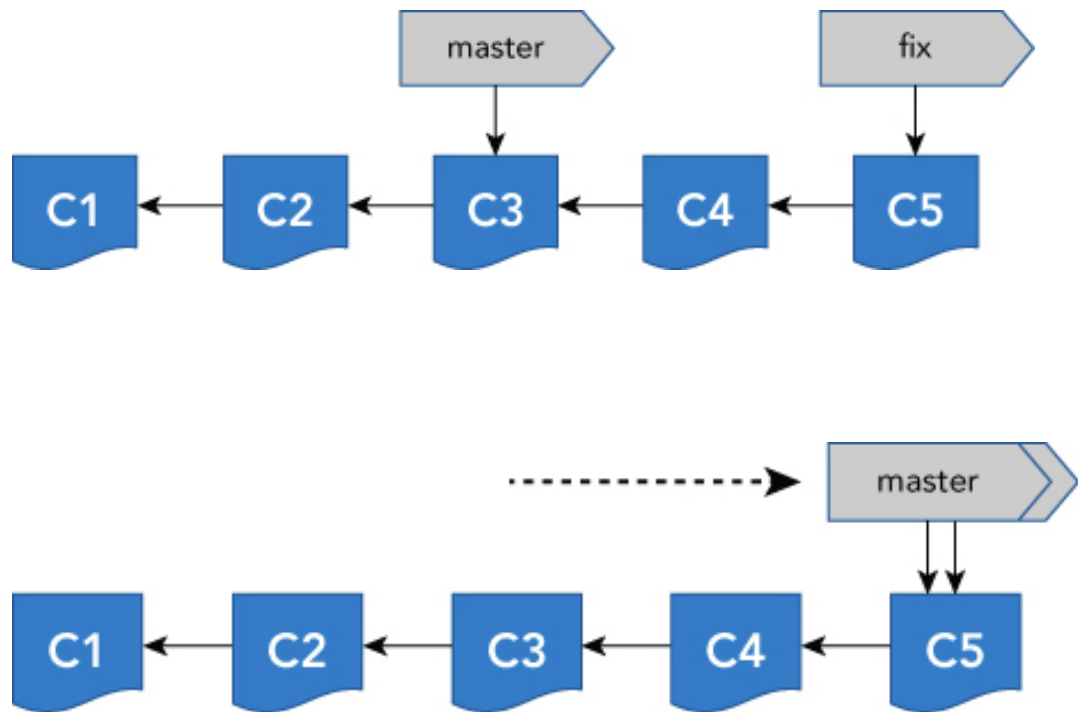
Fast-forward



3-way merge

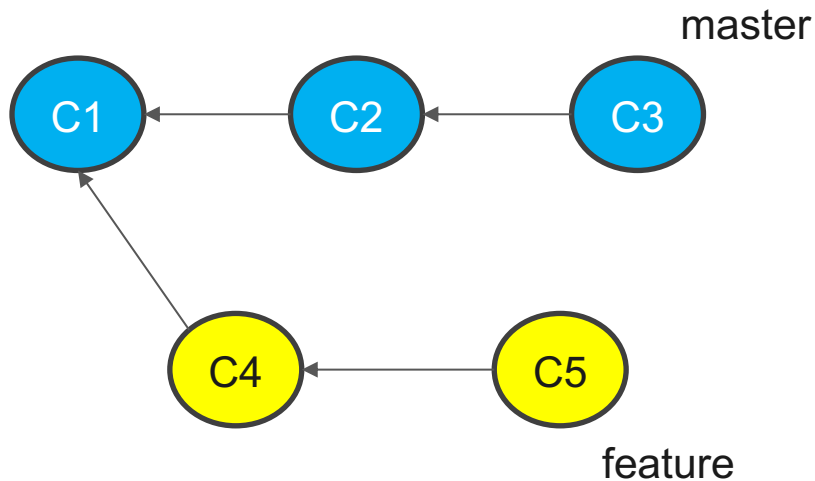


# Fast-forward



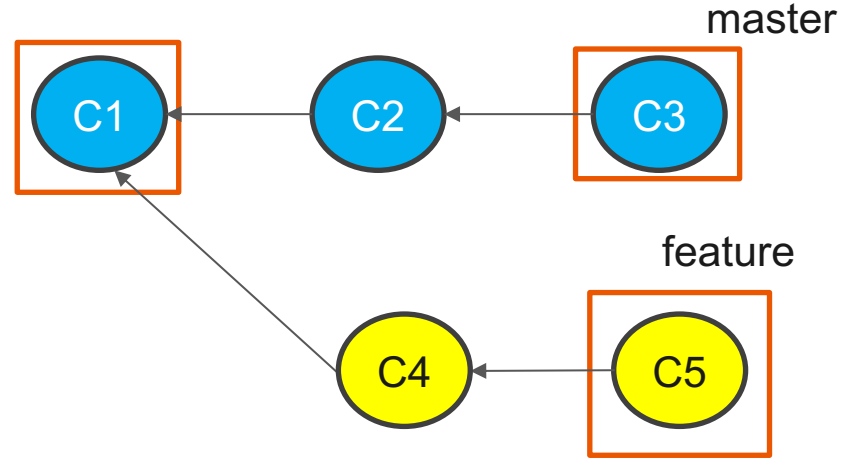
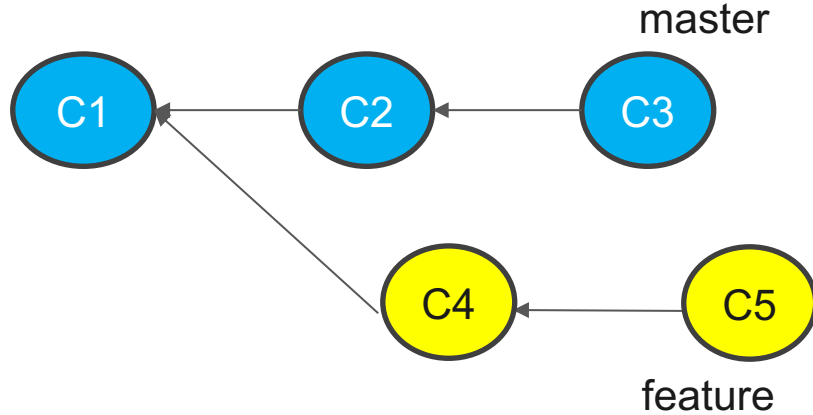
- Nhánh fix được tạo từ nhánh master (commit C3)
- Trong thời gian code nhánh fix, nhánh master không có commit mới
- Khi merge nhánh fix vào master: Git chỉ cần cập nhật vị trí con trỏ master, *đẩy nó từ commit C3 lên commit C5*

# 3-way merge



- Nhánh feature được tạo từ nhánh master (commit C1)
  - Sau đó, cả 2 nhánh feature và master đều có commit mới
- Không thể dùng fast-forward merge để merge nhánh feature vào master

# 3-way merge

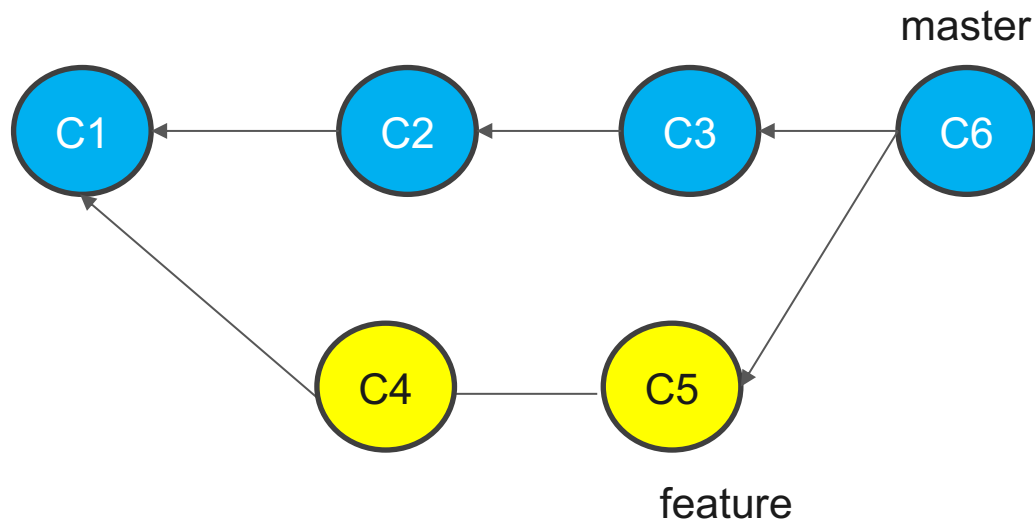


Git chọn 3 commit:

- 2 commit nhánh master và feature đang trở tới
- Commit chung giữa 2 nhánh master và feature để tiến hành merge



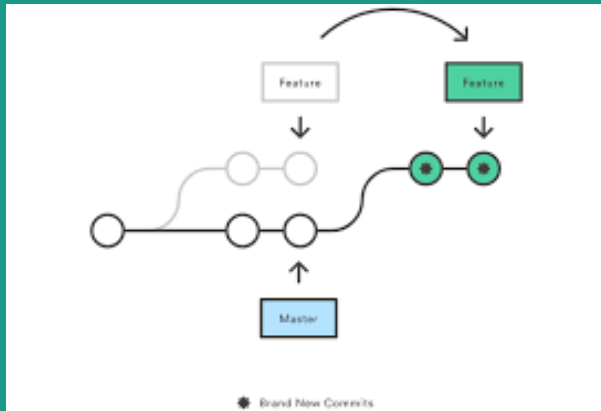
# Kết quả 3-way merge



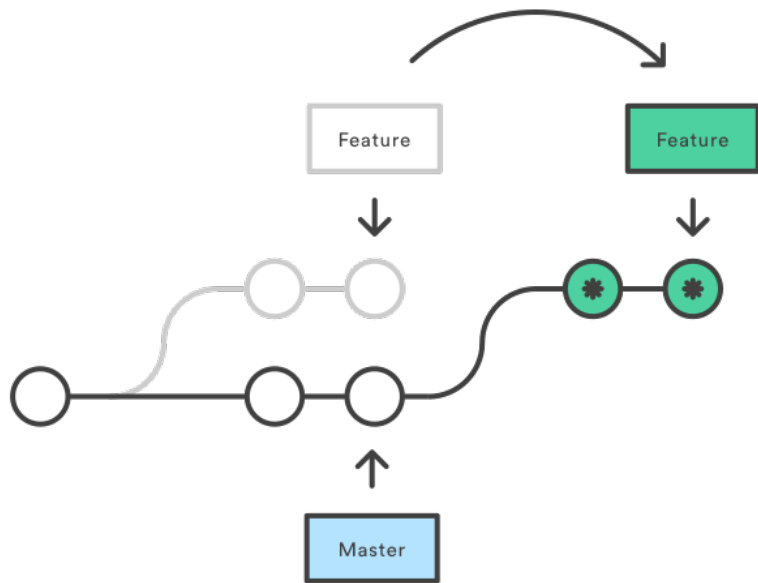
- 1 commit mới (C6) được tạo ở nhánh master
- Commit này có 2 parent commit là C3 và C5

---

# Git rebase



# git rebase cũng tích hợp code từ nhánh A vào nhánh B



\* Brand New Commits

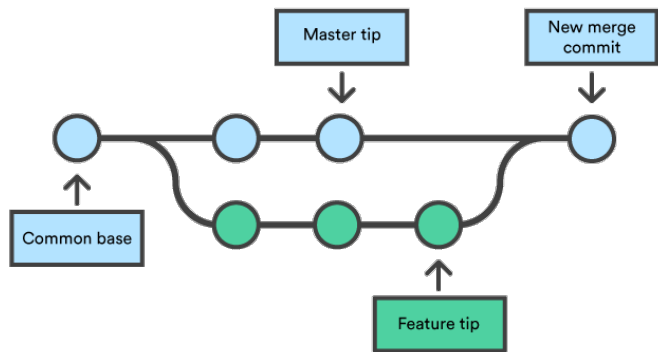
Tích hợp code nhánh feature vào master:

```
$ git checkout feature
```

```
$ git rebase master
```

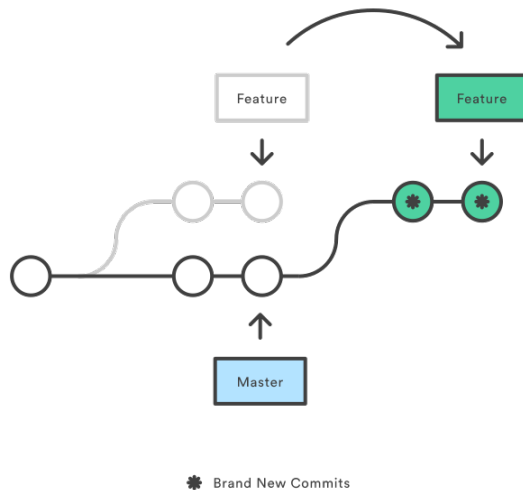
Git xóa 2 commit cũ và tạo 2 commit mới ở nhánh feature dựa trên commit mà nhánh master đang trỏ đến

# Merge



- Lịch sử commit được giữ nguyên
- Lịch sử các nhánh bị phân nhánh, khó nhìn

# rebase



- Lịch sử commit bị thay đổi
- Lịch sử các nhánh trên 1 đường thẳng (linear), dễ nhìn