

Git & Github For version control

Learning Objectives



Hiểu rõ về Git & GitHub

Nắm vững các khái niệm cơ bản về hệ thống quản lý phiên bản và cách Git hoạt động



Thành thạo quy trình làm việc

Làm chủ các kỹ thuật quản lý mã nguồn, nhánh và hợp tác trong dự án phần mềm



Cộng tác hiệu quả

Phát triển kỹ năng làm việc nhóm thông qua GitHub với Fork, Pull Request và CI/CD



Tùy chỉnh và mở rộng

Nâng cao kỹ năng với cấu hình nâng cao, Git Hooks và tự động hóa quy trình

Agenda



Phần 1: Giới thiệu hệ thống quản lý phiên bản và Git

Khái niệm VCS, lược sử Git, nguyên lý hoạt động, cài đặt và cấu hình Git



Phần 2: Quy trình làm việc Git cơ bản

Khởi tạo repository, theo dõi thay đổi, xem lịch sử, hoàn tác, làm việc với remote



Phần 3: Quản lý nhánh và lịch sử

Tạo và gộp nhánh, quản lý nhánh remote, rebase, các workflow phổ biến



Phần 4: Cộng tác qua GitHub

Thiết lập tài khoản, quy trình Fork & Pull Request, quản lý repository, tự động hóa



Phần 5: Tùy chỉnh & mở rộng Git

Cấu hình nâng cao, Git Attributes, Git Hooks, chính sách kiểm soát, Alias & Scripts

Agenda



Phần 1: Giới thiệu hệ thống quản lý phiên bản và Git

Khái niệm VCS, lược sử Git, nguyên lý hoạt động, cài đặt và cấu hình Git



Phần 2: Quy trình làm việc Git cơ bản

Khởi tạo repository, theo dõi thay đổi, xem lịch sử, hoàn tác, làm việc với remote



Phần 3: Quản lý nhánh và lịch sử

Tạo và gộp nhánh, quản lý nhánh remote, rebase, các workflow phổ biến



Phần 4: Cộng tác qua GitHub

Thiết lập tài khoản, quy trình Fork & Pull Request, quản lý repository, tự động hóa



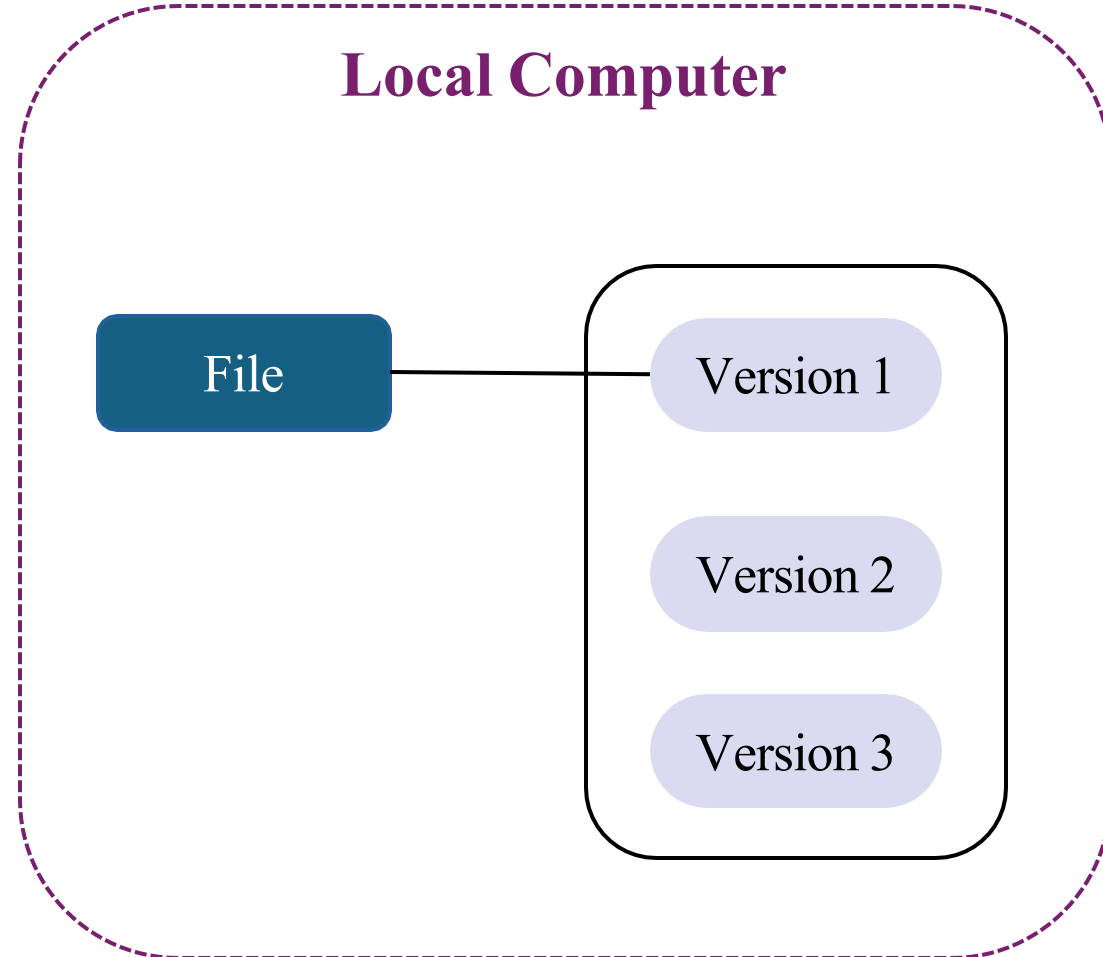
Phần 5: Tùy chỉnh & mở rộng Git

Cấu hình nâng cao, Git Attributes, Git Hooks, chính sách kiểm soát, Alias & Scripts

Khái niệm Version Control System (VCS)

Version Control System là **hệ thống theo dõi và lưu lại các thay đổi của tập tin theo thời gian**. VCS giúp ghi lại ai thực hiện thay đổi, thay đổi gì và khi nào.

Local Version Control Diagram



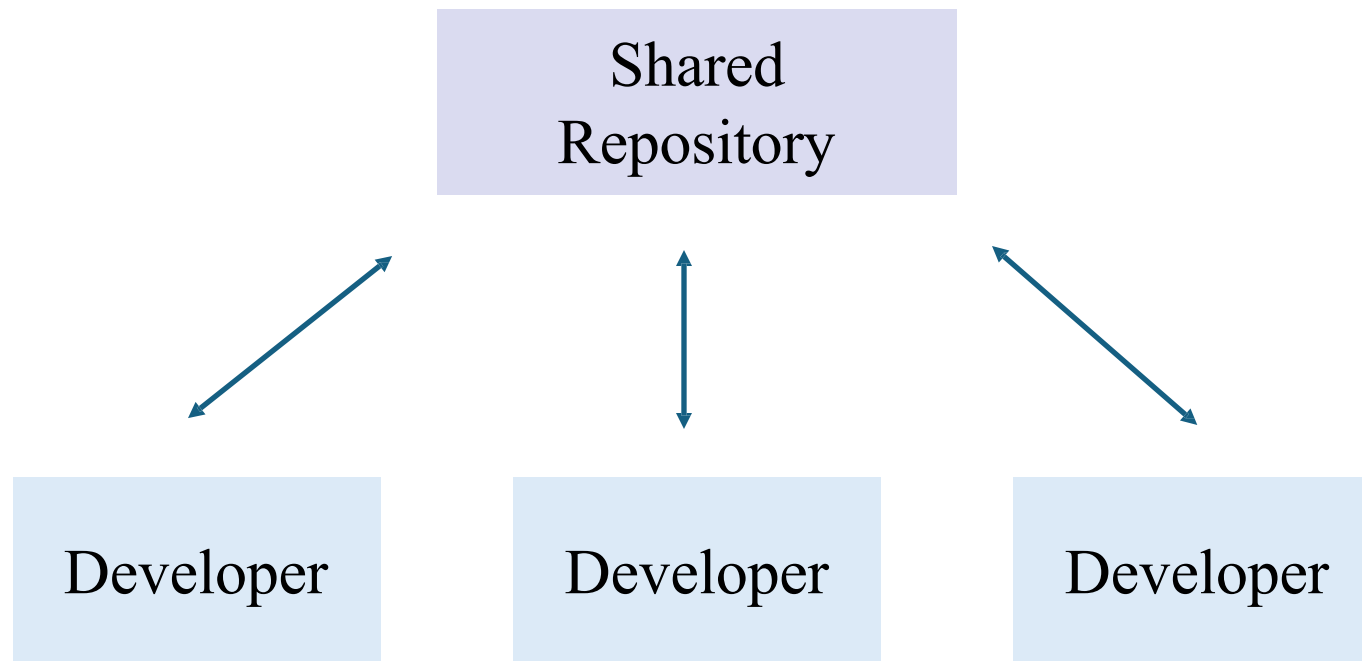
Lợi ích

- ✓ Khôi phục phiên bản cũ khi có lỗi
- ✓ So sánh sự khác biệt giữa các phiên bản
- ✓ Biết rõ ai thay đổi, thay đổi gì và khi nào
- ✓ Bảo vệ dữ liệu và hỗ trợ làm việc nhóm

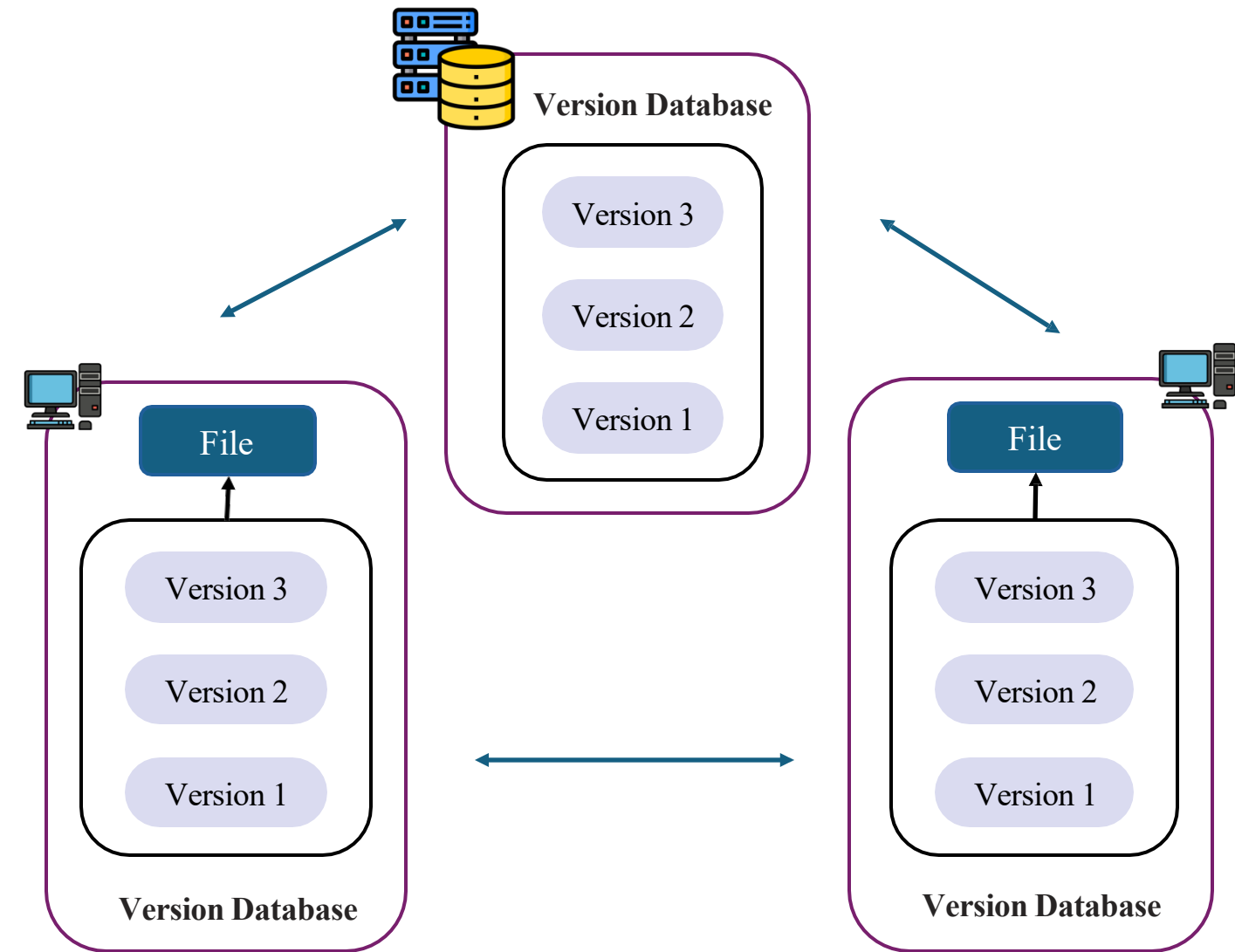
Image source: <https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control>

Các loại VCS

Bên cạnh local VCS có 2 loại VCS chính là **Centralized VCS (CVCS)** và **Distributed VCS (DVCS)**. Mỗi loại VCS đều có đặc trưng riêng biệt.



- **Centralized VCS (CVCS):** Như Subversion, CVS - dùng máy chủ trung tâm, gặp rủi ro khi server hỏng



- **Distributed VCS (DVCS):** Như Git, Mercurial - mỗi người có bản sao đầy đủ, an toàn và linh hoạt hơn

Lược sử Git



git

Bối cảnh ra đời (2005)

Mâu thuẫn về quyền sử dụng giữa nhóm Linux và công cụ BitKeeper

Người sáng lập

Linus Torvalds tạo ra Git để quản lý mã nguồn Linux

Mục tiêu thiết kế

Nhanh, đơn giản, dễ phân nhánh, làm việc phân tán

- ❑ Năm 2005, sau khi có tranh chấp giữa nhóm phát triển Linux và công ty BitKeeper về quyền sử dụng, Linus Torvalds quyết định tạo hệ thống quản lý phiên bản mới cho dự án Linux.
- ❑ Linus đã thiết kế Git dựa trên kinh nghiệm từ các hệ thống cũ và sửa những nhược điểm của chúng. Chỉ sau vài tháng, Git đã trở thành công cụ chính thức của Linux và được cộng đồng mã nguồn mở nhanh chóng sử dụng rộng rãi.

Nguyên lý hoạt động của Git

1 Snapshot thay vì Diff

- Khác với các hệ thống VCS cũ chỉ lưu các thay đổi, **Git lưu "bản chụp" toàn bộ dự án mỗi khi commit.**
- Nếu file không thay đổi, Git chỉ tạo liên kết đến bản chụp cũ, giúp tiết kiệm dung lượng nhưng vẫn nhanh.

2 Toàn vẹn dữ liệu với SHA-1

- **Git bảo vệ dữ liệu bằng hàm băm SHA-1.** Mỗi thứ trong Git đều có mã định danh 40 ký tự.
- **Khi nội dung thay đổi, mã hash cũng thay đổi hoàn toàn,** giúp đảm bảo dữ liệu không bị hỏng.

3 Nguyên lý "Offline-first"

- Git cho phép làm việc mà không cần kết nối mạng.
- Hầu hết các thao tác như commit, tạo nhánh, gộp nhánh đều thực hiện trên máy tính của bạn.
- Chỉ khi cần chia sẻ code (push/pull), bạn mới cần Internet.

Cài đặt và cấu hình Git



Cài đặt Git

- Linux (Debian/Ubuntu): `sudo apt install git`
- Linux (Fedora/RHEL): `sudo dnf install git`
- macOS: `xcode-select --install` hoặc tải từ git-scm.com
- Windows: cài đặt từ git-scm.com hoặc `choco install git`



Cấu hình cơ bản

Thiết lập thông tin người dùng và các tùy chọn căn bản:

- `git config --global user.name "Tên Của Bạn"`
- `git config --global user.email "email@example.com"`
- `git config --global core.editor "code --wait"`
- `git config --global color.ui auto`



Thiết lập Alias hữu ích

Tạo lệnh tắt cho các câu lệnh Git thường dùng:

- `git config --global alias.st status`
- `git config --global alias.co checkout`
- `git config --global alias.br branch`
- `git config --global alias.cm "commit -m"`

Hỗ trợ và trợ giúp



Tài liệu chính thức

Trang web git-scm.com cung cấp tài liệu đầy đủ về Git. Bạn sẽ tìm thấy hướng dẫn cơ bản, nâng cao và danh sách các lệnh hữu ích.



Trợ giúp từ dòng lệnh

Git có sẵn trợ giúp ngay trong dòng lệnh. Gõ `"git help"`, `"git --help"` hoặc `"man git-{lệnh}"` để xem hướng dẫn về bất kỳ lệnh nào.



Cộng đồng hỗ trợ

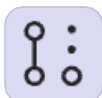
Khi cần giúp đỡ, hãy tìm đến Stack Overflow, GitHub Discussions và các diễn đàn Git. Cộng đồng Git luôn sẵn sàng hỗ trợ người mới.



Sách và tài nguyên học tập

Sách "Pro Git" của Scott Chacon là tài liệu miễn phí và đầy đủ nhất về Git, có tại git-scm.com/book. Bạn cũng có thể học qua các khóa trực tuyến và video hướng dẫn.

Tổng kết phần 1



Khái niệm VCS

Hệ thống quản lý phiên bản giúp theo dõi và kiểm soát các thay đổi trong mã nguồn



Lược sử Git

Git được phát triển bởi Linus Torvalds năm 2005, là hệ thống phân tán (DVCS)



Nguyên lý hoạt động

Git lưu trữ dữ liệu dưới dạng ảnh chụp (snapshot) thay vì lưu trữ sự thay đổi



Cài đặt và cấu hình

Cài đặt dễ dàng trên các nền tảng, cấu hình cơ bản với git config



Hỗ trợ và trợ giúp

Tài liệu phong phú, trợ giúp dòng lệnh và cộng đồng năng động

Agenda



Phần 1: Giới thiệu hệ thống quản lý phiên bản và Git

Khái niệm VCS, lược sử Git, nguyên lý hoạt động, cài đặt và cấu hình Git



Phần 2: Quy trình làm việc Git cơ bản

Khởi tạo repository, theo dõi thay đổi, xem lịch sử, hoàn tác, làm việc với remote



Phần 3: Quản lý nhánh và lịch sử

Tạo và gộp nhánh, quản lý nhánh remote, rebase, các workflow phổ biến



Phần 4: Cộng tác qua GitHub

Thiết lập tài khoản, quy trình Fork & Pull Request, quản lý repository, tự động hóa



Phần 5: Tùy chỉnh & mở rộng Git



Cấu hình nâng cao, Git Attributes, Git Hooks, chính sách kiểm soát, Alias & Scripts

Khởi tạo và sao chép kho (Repository)

Khởi tạo repository mới

Để bắt đầu dùng Git cho dự án mới, tạo repository trong thư mục dự án. Lệnh **git init** sẽ tạo thư mục ẩn **.git/** để Git lưu trữ dữ liệu và theo dõi dự án.

```
mkdir <new_project_name>
cd <new_project_name>
git init
```

Name	Status	Date modified	Type	Size
 .git		6/4/2025 11:17 PM	File folder	

Sao chép repository từ xa

Để làm việc với dự án có sẵn, cần sao chép (clone) repository từ máy chủ về máy tính. Lệnh **git clone** sẽ tải về toàn bộ lịch sử và các nhánh của dự án.

```
git clone https://github.com/username/repository.git
git clone git@github.com:username/repository.git
```

Khi clone, Git tạo một thư mục mới với tên giống với tên repository. Có thể đặt tên thư mục khác như sau:

```
git clone https://github.com/username/repository.git
<new_name>
```

Theo dõi và lưu trữ thay đổi (1/2)



Trạng thái file trong Git

Trong Git, mỗi file trong thư mục làm việc thuộc một trong bốn trạng thái:

- **Untracked:** File mới chưa được Git theo dõi, chưa có trong lần lưu trước
- **Modified:** File đã thay đổi so với phiên bản trước, nhưng chưa sẵn sàng để lưu
- **Staged:** File đã được đưa vào vùng chuẩn bị, sẵn sàng cho lần lưu tiếp theo
- **Committed:** File đã được lưu trữ an toàn trong Git



Thao tác cơ bản

Để xem trạng thái hiện tại của các file, dùng lệnh:

```
git status
```

Để đưa file vào vùng chuẩn bị (staging area):

```
git add tên-file.txt # Thêm một file
git add .             # Thêm tất cả file đã thay đổi
```

Để lưu các thay đổi đã chuẩn bị:

```
git commit -m "Mô tả ngắn về thay đổi"
```



Sử dụng .gitignore

File **.gitignore** liệt kê các file và thư mục mà Git sẽ bỏ qua, không theo dõi. Thường là:

- File log và dữ liệu tạm
- Thư mục build
- File cấu hình cá nhân (như .env)
- Thư mục chứa thư viện (node_modules, vendor)





Ví dụ về nội dung file .gitignore:

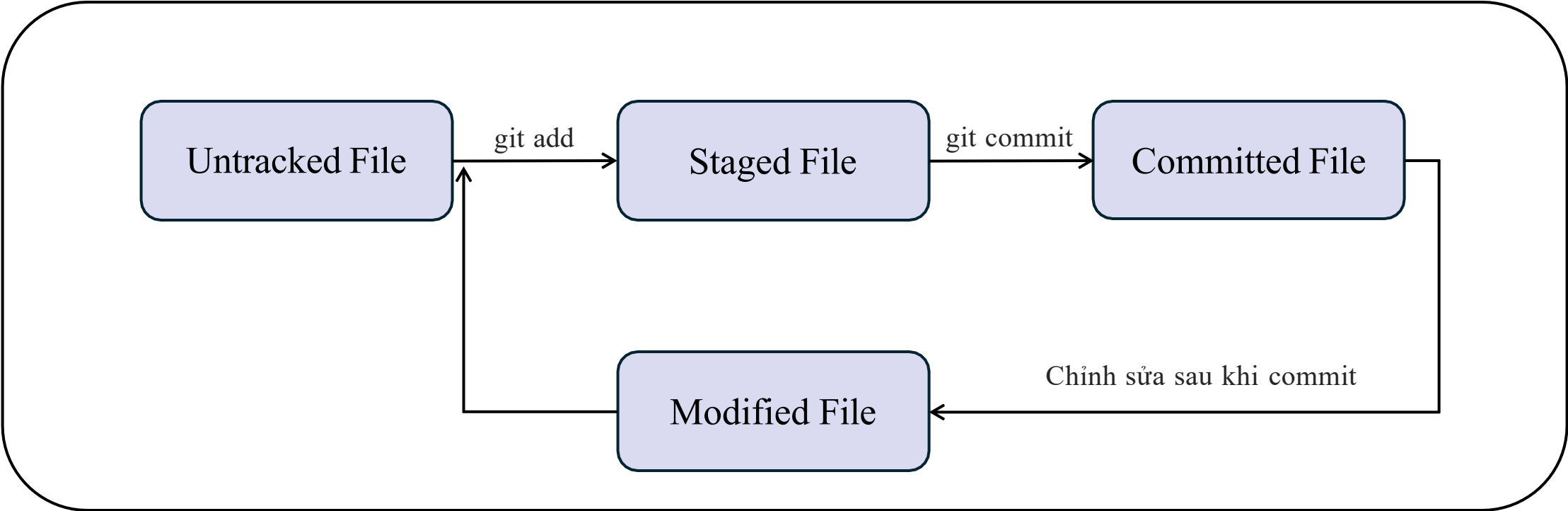
```
# Bỏ qua thư mục build
/build/
```

```
# Bỏ qua tất cả file có đuôi .log
*.log
```

```
# Bỏ qua file cấu hình cụ thể
config.ini
.env
```

Theo dõi và lưu trữ thay đổi (2/2)

Trạng thái	Mô tả ngắn	Xuất hiện trong “git status”	Hành động tiếp theo
 Untracked	File mới chưa được Git theo dõi, chưa có trong lần lưu trước	Untracked files:	git add
 Modified	File đã thay đổi so với phiên bản trước, nhưng chưa sẵn sàng để lưu	Changes not staged for commit:	git add
 Staged	File đã được đưa vào vùng chuẩn bị, sẵn sàng cho lần lưu tiếp theo (commit)	Changes to be committed:	git commit
 Committed	File đã được ghi lại (commit) trong repository, nằm trong lịch sử Git.	Không hiển thị trong git status	Không cần làm gì thêm



Xem lịch sử commit

Lệnh **git log** hiển thị lịch sử commit với mã hash, tác giả, thời gian và nội dung. Giúp theo dõi và hiểu rõ các thay đổi trong dự án.

git log -p

Hiển thị sự khác biệt của mỗi commit.

git log --stat

Hiện số file và dòng thay đổi trong mỗi commit.

git log --since="2 weeks ago"

Lọc commit trong 2 tuần gần đây.

git log --author="Tên"

Lọc commit theo tên người tạo.

Kết hợp các tùy chọn để tối ưu cách xem. Ví dụ: **git log --oneline --graph --all** hiển thị lịch sử đồ họa ngắn gọn cho tất cả nhánh.

Xem lịch sử commit

git log -p

```
commit 1cbb0ad849bf85837bb0adfdc1bf5e99cc5942ee (HEAD -> master, origin/master)
Author: dangnha <nhand.21it@vku.udn.vn>
Date: Sat May 31 02:35:49 2025 +0700

    lab2.3 commit 2 at main branch

diff --git a/update-master.py b/update-master.py
index 58aaab5..830133a 100644
--- a/update-master.py
+++ b/update-master.py
@@ -1,2 @@
-print("commit1")
\ No newline at end of file
+print("commit1")
+print("commit2")

commit caa8787c621a771c533613b0fcb80952a2c440d7
Author: dangnha <nhand.21it@vku.udn.vn>
Date: Sat May 31 02:35:34 2025 +0700

    lab2.3 commit 1 at main branch

diff --git a/update-master.py b/update-master.py
new file mode 100644
index 0000000..58aaab5
--- /dev/null
+++ b/update-master.py
@@ -0,0 +1 @@
+print("commit1")
\ No newline at end of file
```

git log --stat

```
commit 1cbb0ad849bf85837bb0adfdc1bf5e99cc5942ee (HEAD -> master, origin/master)
Author: dangnha <nhand.21it@vku.udn.vn>
Date: Sat May 31 02:35:49 2025 +0700

    lab2.3 commit 2 at main branch

update-master.py | 3 ++-
1 file changed, 2 insertions(+), 1 deletion(-)

commit caa8787c621a771c533613b0fcb80952a2c440d7
Author: dangnha <nhand.21it@vku.udn.vn>
Date: Sat May 31 02:35:34 2025 +0700

    lab2.3 commit 1 at main branch

update-master.py | 1 +
1 file changed, 1 insertion(+)

commit 4c6fc7b8e7a3119beda6d01cb9d19bea3801d040
Merge: 040690c 5ff28fa
Author: dangnha <nhand.21it@vku.udn.vn>
Date: Sat May 31 02:20:50 2025 +0700

    Merge and solve conflict
```

Xem lịch sử commit

git log --since="1 weeks ago"

```
commit 1cbb0ad849bf85837bb0adfdc1bf5e99cc5942ee (HEAD -> master, origin/master)
Author: dangnha <nhand.21it@vku.udn.vn>
Date: Sat May 31 02:35:49 2025 +0700
```

lab2.3 commit 2 at main branch

```
commit caa8787c621a771c533613b0fcb80952a2c440d7
Author: dangnha <nhand.21it@vku.udn.vn>
Date: Sat May 31 02:35:34 2025 +0700
```

lab2.3 commit 1 at main branch

```
commit 4c6fc7b8e7a3119beda6d01cb9d19bea3801d040
Merge: 040690c 5ff28fa
Author: dangnha <nhand.21it@vku.udn.vn>
Date: Sat May 31 02:20:50 2025 +0700
```

Merge and solve conflict

```
commit 5ff28fa2c9e53e10a9f7d941577c87531fa212b7 (origin/feature-a, feature-a)
Author: dangnha <nhand.21it@vku.udn.vn>
Date: Sat May 31 02:18:45 2025 +0700
```

add common file at feature a

```
commit 040690c3551e470cc2e7ad20ffd463b432a9becf
Author: dangnha <nhand.21it@vku.udn.vn>
Date: Sat May 31 02:18:08 2025 +0700
```

add common file

```
commit cb7fe675eabb7c38ec295e3de4066c2d7f39364c
Author: dangnha <nhand.21it@vku.udn.vn>
Date: Sat May 31 02:13:42 2025 +0700
```

Create txt feature a

git log --author="dangnha"

```
commit 1cbb0ad849bf85837bb0adfdc1bf5e99cc5942ee (HEAD -> master, origin/master)
Author: dangnha <nhand.21it@vku.udn.vn>
Date: Sat May 31 02:35:49 2025 +0700
```

lab2.3 commit 2 at main branch

```
commit caa8787c621a771c533613b0fcb80952a2c440d7
Author: dangnha <nhand.21it@vku.udn.vn>
Date: Sat May 31 02:35:34 2025 +0700
```

lab2.3 commit 1 at main branch

```
commit 4c6fc7b8e7a3119beda6d01cb9d19bea3801d040
Merge: 040690c 5ff28fa
Author: dangnha <nhand.21it@vku.udn.vn>
Date: Sat May 31 02:20:50 2025 +0700
```

Merge and solve conflict

```
commit 5ff28fa2c9e53e10a9f7d941577c87531fa212b7 (origin/feature-a, feature-a)
Author: dangnha <nhand.21it@vku.udn.vn>
Date: Sat May 31 02:18:45 2025 +0700
```

add common file at feature a

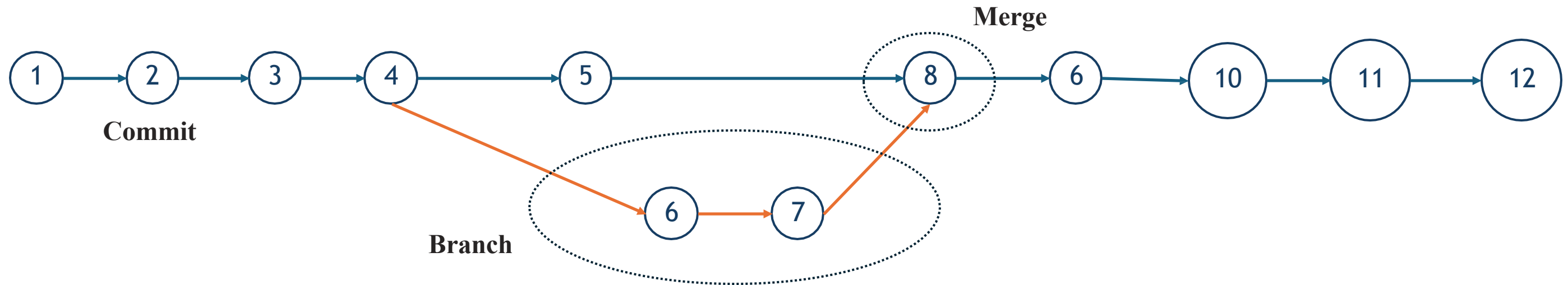
```
commit 040690c3551e470cc2e7ad20ffd463b432a9becf
Author: dangnha <nhand.21it@vku.udn.vn>
Date: Sat May 31 02:18:08 2025 +0700
```

add common file

```
commit cb7fe675eabb7c38ec295e3de4066c2d7f39364c
Author: dangnha <nhand.21it@vku.udn.vn>
Date: Sat May 31 02:13:42 2025 +0700
```

Create txt feature a

Xem lịch sử commit



```
* d8206a9 (origin/feature-b, feature-b) lab3.2 add file feature-b
* ae2535a lab2.3 update common file
* 30d72fa lab2.3 commit 1
* 1cbb0ad (HEAD -> master, origin/master) lab2.3 commit 2 at main branch
* caa8787 lab2.3 commit 1 at main branch
* 4c6fc7b Merge and solve conflict
| \
| * 5ff28fa (origin/feature-a, feature-a) add common file at feature a
| * cb7fe67 Create txt feature a
* | 040690c add common file
| /
* 4a3be32 Change code to wrong
* 246bf52 Change readme
* a8f5090 add read data
* 335ee7c Init project - add readme
```

Kết hợp các tùy chọn để tối ưu cách xem. Ví dụ: `git log --oneline --graph --all` hiển thị lịch sử đồ họa ngắn gọn cho tất cả nhánh.

Hoàn tác thay đổi

Git cho phép **hoàn tác và sửa đổi thay đổi ở các trạng thái khác nhau**. Hiểu các lệnh này giúp bạn xử lý tình huống khi mắc lỗi hoặc cần điều chỉnh.

Tình huống	Lệnh	Mô tả
Đã sửa file nhưng chưa staged	<code>git restore <file></code>	Khôi phục file về trạng thái của commit trước
Đã staged nhưng muốn bỏ stage	<code>git restore --staged <file></code>	Đưa file từ staged về modified
Muốn sửa commit gần nhất	<code>git commit --amend</code>	Sửa message hoặc thêm file vào commit cuối
Muốn hoàn tác commit	<code>git revert <commit-hash></code>	Tạo commit mới để hoàn tác thay đổi

 **Cảnh báo:** `git restore` sẽ xóa hoàn toàn thay đổi chưa commit và không thể khôi phục!

Để an toàn hơn, sử dụng `git stash`:

```
git stash      # Lưu tạm thời các thay đổi
git stash pop  # Áp dụng lại thay đổi đã lưu
```

Với `git reset`, bạn có thể quay lại phiên bản trước đó, nhưng hãy cẩn trọng vì lệnh này ảnh hưởng đến lịch sử commit.

Làm việc với remote repository

Remote repository là **phiên bản lưu trữ dự án từ xa trên máy chủ** như GitHub, GitLab hoặc Bitbucket, cho phép cộng tác và đồng bộ hóa thay đổi.

Xem danh sách remote:

```
git remote -v, git remote add, git remote remove
```

Thêm remote mới:

```
git remote add origin https://github.com/username/repository.git
```

"origin" là tên quy ước cho remote mặc định, có thể đặt tên khác như "upstream" hoặc "production"

Đẩy và kéo thay đổi

Đẩy commit lên remote:

```
git push origin main, git push --force-with-lease
```

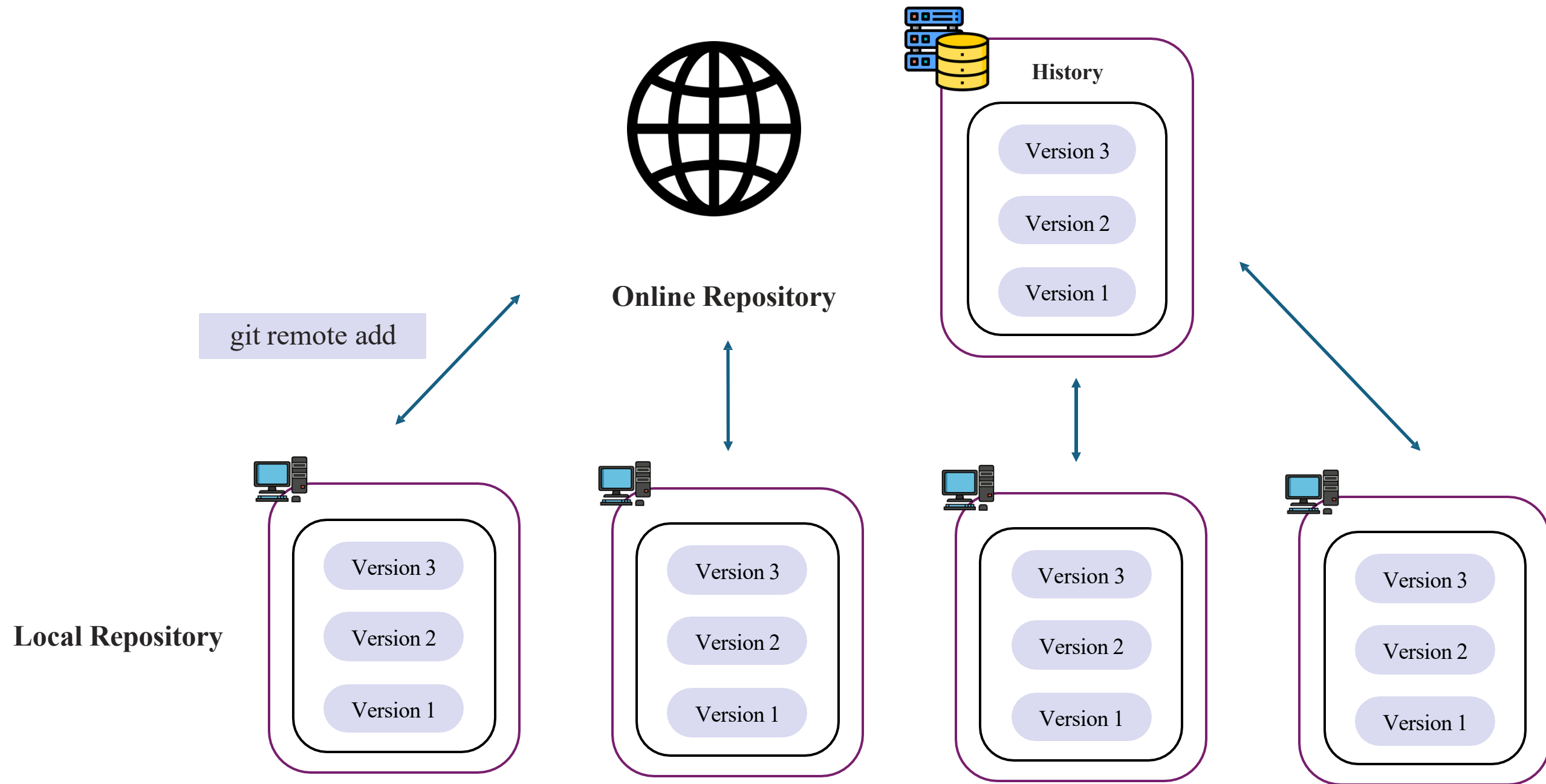
Kéo thay đổi từ remote về local:

```
git pull origin main, git pull --rebase
```

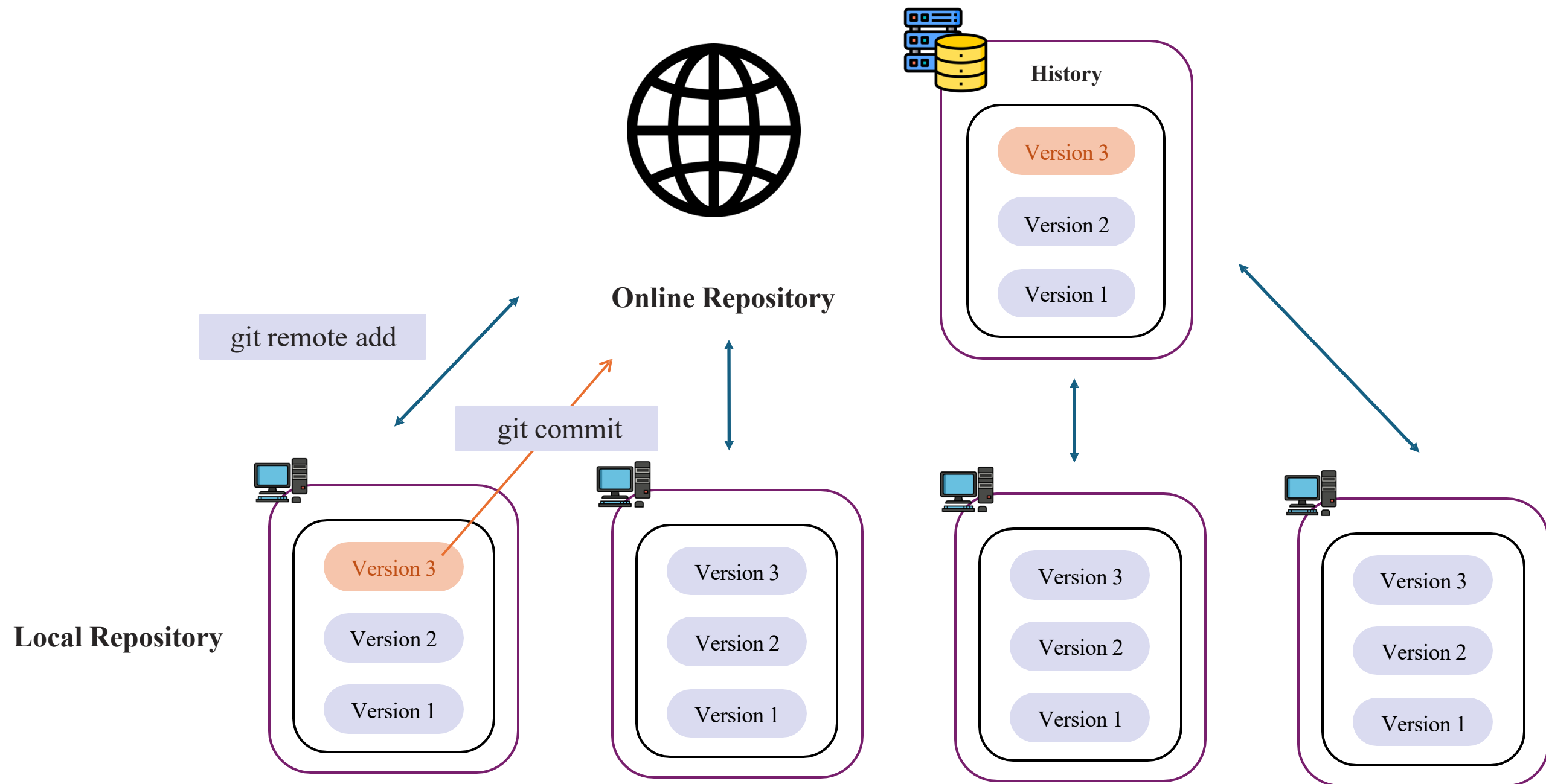
Pull là kết hợp của fetch và merge. Để kiểm soát chi tiết hơn:

```
git fetch, git fetch --all  
git merge origin/main
```

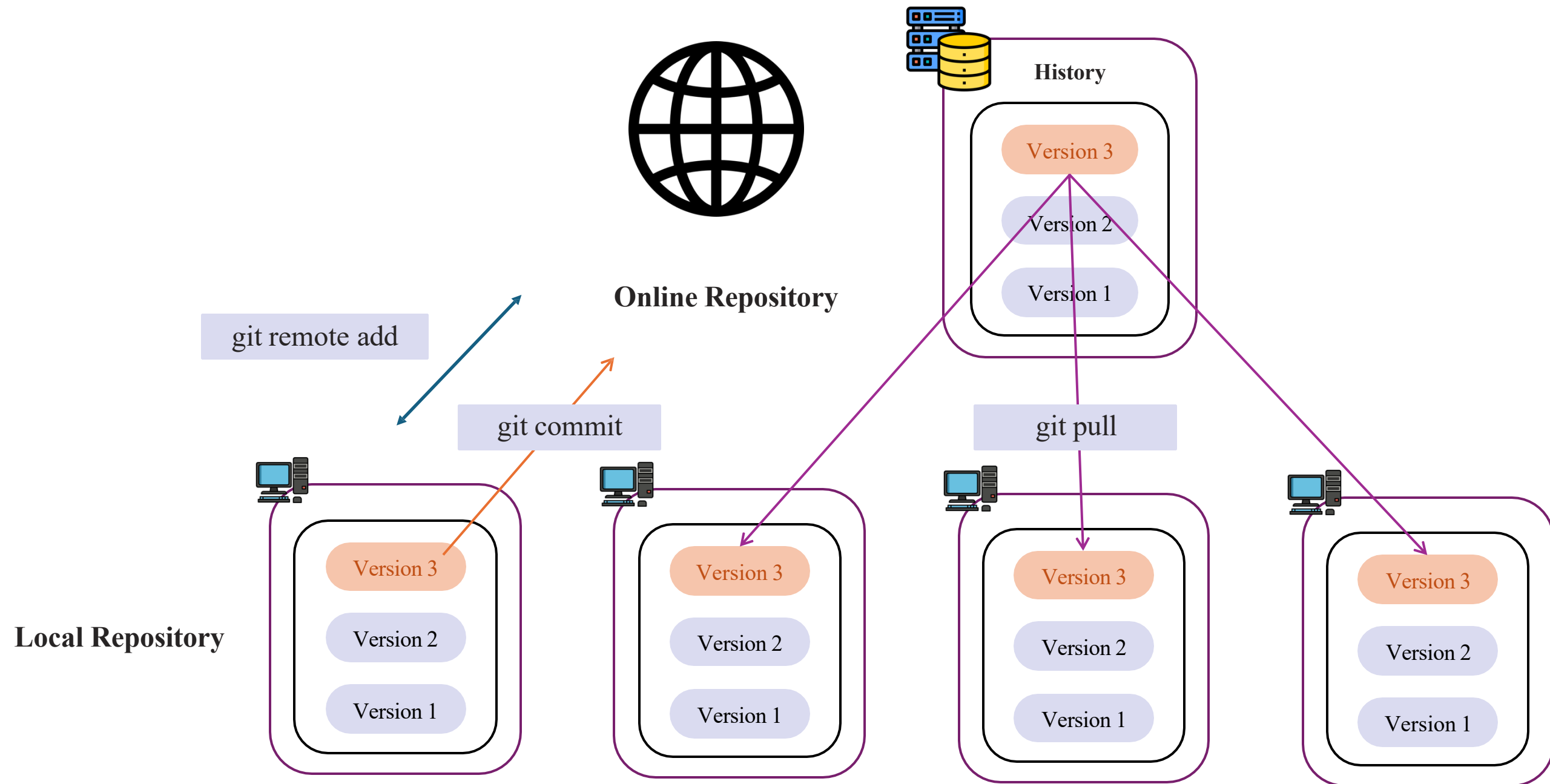
Làm việc với remote repository



Làm việc với remote repository



Làm việc với remote repository



Tagging – Đánh dấu phiên bản

Tag đánh dấu các điểm quan trọng trong lịch sử repository, thường là các phiên bản phát hành. Tag giúp đặt tên cho commit cụ thể, dễ dàng tham chiếu sau này.

Annotated tag

- Là tag đầy đủ với thông tin người tạo, ngày tạo và message.
- Được khuyến nghị cho các phiên bản quan trọng như release version.

**git tag -a v1.0 -m "Version 1.0
- Phát hành chính thức"**

Lightweight tag

- Là con trỏ đơn giản đến một commit cụ thể, không lưu thông tin bổ sung.
- Thích hợp cho đánh dấu tạm thời hoặc cá nhân.

git tag v1.0-beta

Đẩy tag lên remote

Git push mặc định không đẩy tag.
Để đẩy tag cụ thể:

git push origin v1.0

Đẩy tất cả tag:

git push origin --tags

Tag đánh dấu các điểm quan trọng trong lịch sử repository, thường là các phiên bản phát hành. Tag giúp đặt tên cho commit cụ thể, dễ dàng tham chiếu sau này.

Xem nội dung tag: **git show v1.0**. Liệt kê tất cả tag: **git tag**. Tìm kiếm tag theo mẫu: **git tag -l "v1.*"**.

Tổng kết phần 2



Khởi tạo và sao chép kho

Tạo mới (git init) hoặc sao chép (git clone) repository để bắt đầu làm việc



Theo dõi và lưu trữ thay đổi

Sử dụng git add để đưa file vào staging area và git commit để lưu lại



Xem lịch sử và hoàn tác

Kiểm tra lịch sử (git log), hoàn tác thay đổi khi cần thiết



Làm việc với remote repository

Đồng bộ hóa thay đổi giữa local và remote với git push/pull



Tagging - Đánh dấu phiên bản

Sử dụng tag để đánh dấu các phiên bản quan trọng (lightweight/annotated)

Agenda



Phần 1: Giới thiệu hệ thống quản lý phiên bản và Git

Khái niệm VCS, lược sử Git, nguyên lý hoạt động, cài đặt và cấu hình Git



Phần 2: Quy trình làm việc Git cơ bản

Khởi tạo repository, theo dõi thay đổi, xem lịch sử, hoàn tác, làm việc với remote



Phần 3: Quản lý nhánh và lịch sử

Tạo và gộp nhánh, quản lý nhánh remote, rebase, các workflow phổ biến



Phần 4: Cộng tác qua GitHub

Thiết lập tài khoản, quy trình Fork & Pull Request, quản lý repository, tự động hóa



Phần 5: Tùy chỉnh & mở rộng Git

Cấu hình nâng cao, Git Attributes, Git Hooks, chính sách kiểm soát, Alias & Scripts

Khái niệm nhánh (Branch)



Nhánh là Con trỏ

- Nhánh trong Git là **con trỏ đến một commit** cụ thể.
- Khi tạo commit mới, con trỏ tự động cập nhật. Vì chỉ là file nhỏ chứa mã SHA-1, việc tạo và xóa nhánh diễn ra rất nhanh chóng.



Phát triển Song song

- **Nhánh tạo môi trường làm việc riêng** để phát triển tính năng mới, sửa lỗi, hoặc thử nghiệm mà không ảnh hưởng mã nguồn chính.
- Giúp nhiều người **làm việc đồng thời trên cùng codebase mà không xung đột**.



Lịch sử Phân nhánh

- Lịch sử Git như một cây với nhiều nhánh phát triển riêng biệt.
- Sau đó các **nhánh được gộp (merge) hoặc tái cơ sở (rebase) tạo lịch sử thống nhất**, giúp theo dõi quá trình phát triển.

❖ Git tự động tạo nhánh "master" hoặc "main" khi khởi tạo repository. Đây là nhánh chính chứa phiên bản ổn định. Khi phát triển tính năng mới hoặc sửa lỗi, hãy tạo nhánh mới từ nhánh chính.

Tạo, chuyển và gộp nhánh



Tạo và chuyển nhánh

Tạo và chuyển đến nhánh mới với một lệnh:

```
git checkout -b  
feature/login
```

Tương đương với:

```
git branch feature/login  
git checkout feature/login
```



Phát triển tính năng

- Trên nhánh mới, mọi commit sẽ chỉ ảnh hưởng đến nhánh hiện tại, giữ nguyên nhánh main.
- Điều này cho phép bạn thử nghiệm tự do mà không ảnh hưởng đến code chính.



Gộp nhánh

Khi hoàn thành tính năng và muốn đưa thay đổi vào nhánh chính:

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

Git sẽ tự động hợp nhất các thay đổi nếu không có xung đột.

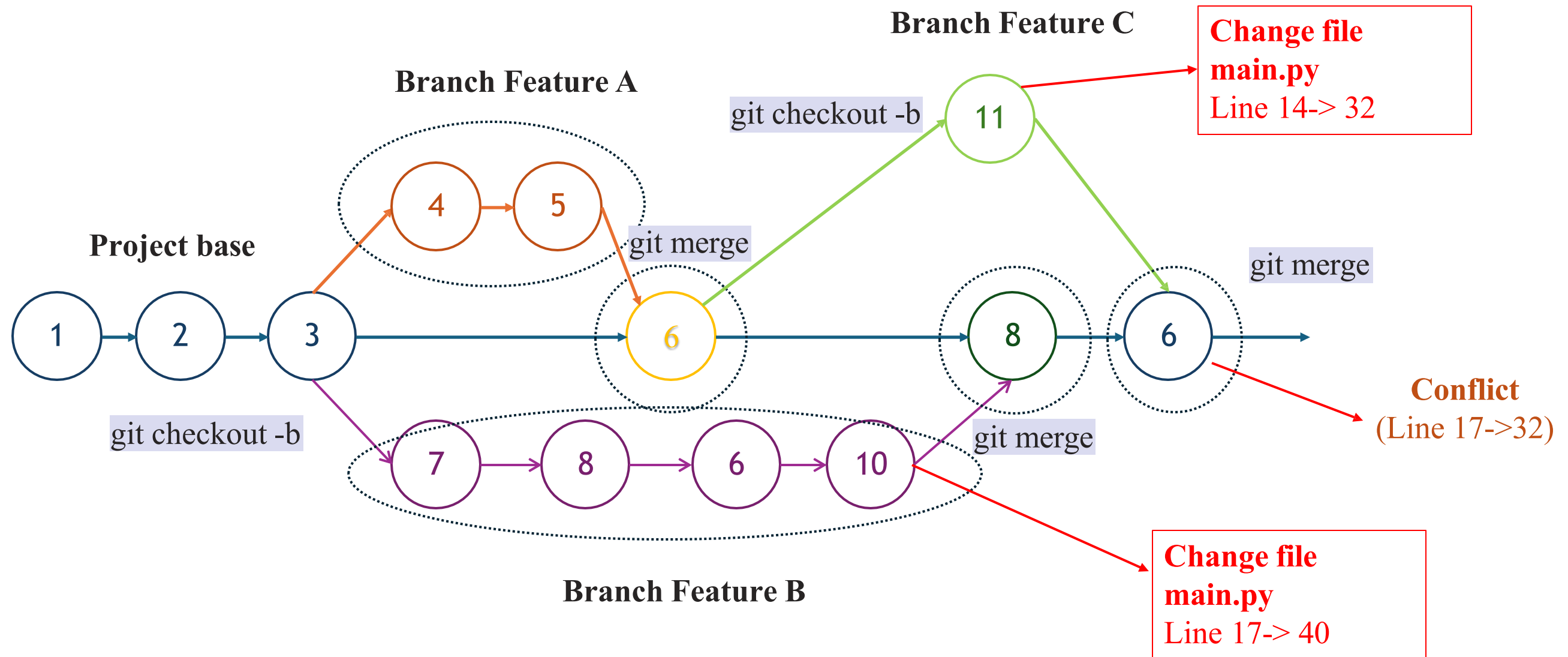


Giải quyết xung đột

Khi xung đột xảy ra (cùng dòng bị thay đổi ở cả hai nhánh), Git sẽ đánh dấu các phần xung đột. Giải quyết bằng cách chỉnh sửa file, sau đó:

```
git add .  
git commit
```

Tạo, chuyển và gộp nhánh



Quản lý nhánh



Liệt kê nhánh

Các lệnh xem danh sách nhánh:

- **git branch**: Nhánh local, đánh dấu nhánh hiện tại bằng *
- **git branch -r**: Nhánh remote-tracking
- **git branch -a**: Tất cả nhánh (local và remote)
- **git branch -vv**: Chi tiết trạng thái theo dõi remote



Đổi tên nhánh

Đổi tên nhánh:

- **git branch -m tên-cũ tên-mới**: Đổi tên nhánh chỉ định
- **git branch -m tên-mới**: Đổi tên nhánh hiện tại

Cập nhật lên remote:

```
git push origin :tên-cũ tên-mới
```



Xóa nhánh

Xóa nhánh không cần thiết:

- **git branch -d feature/login**: Xóa nhánh đã merge
- **git branch -D feature/login**: Xóa nhánh chưa merge (cẩn trọng)
- **git push origin --delete feature/login**: Xóa nhánh trên remote



Nhánh không có lịch sử

Tạo nhánh mới không có lịch sử chung:

```
git checkout --orphan tên-nhánh-mới
```

Hữu ích khi muốn bắt đầu lại từ đầu nhưng giữ nguyên mã nguồn.

Nhánh remote

Remote-tracking branches

Remote-tracking branch là tham chiếu đến trạng thái nhánh trên repository từ xa, có dạng:

[remote-name]/[branch-name]

(ví dụ: origin/main). Bạn không thể trực tiếp chỉnh sửa - chúng tự động cập nhật khi giao tiếp với remote.

Lệnh **git fetch** cập nhật tất cả remote-tracking branch theo trạng thái mới nhất của remote repository mà không thay đổi nhánh local.

Theo dõi nhánh từ xa

Thiết lập quan hệ theo dõi giữa nhánh local và remote:

```
git branch --track branch-name  
origin/branch-name
```

Khi clone repository, Git tự động thiết lập nhánh main local theo dõi origin/main, cho phép **git pull** và **git push** không cần chỉ định nhánh.

Thay đổi nhánh remote mà nhánh local theo dõi:

```
git branch -u origin/dev-feature
```

Tạo nhánh local từ remote

Tạo nhánh local mới từ nhánh remote:

```
git fetch origin  
git checkout -b mybranch  
origin/feature-x
```

Hoặc cú pháp rút gọn:

```
git checkout feature-x
```

Nếu nhánh local chưa tồn tại nhưng có nhánh remote-tracking cùng tên, Git tự tạo nhánh local và thiết lập theo dõi.

Push nhánh mới lên remote lần đầu:

```
git push -u origin feature-branch
```


Rebase

Rebase là tính năng mạnh mẽ của Git, **tạo lịch sử commit gọn gàng và tuyến tính bằng** cách "tái áp dụng" các commit từ nhánh hiện tại lên đầu nhánh đích, thay vì tạo commit merge.

Thực hiện Rebase cơ bản

```
git checkout feature-branch  
git rebase main
```

Lệnh này gỡ bỏ tạm thời các commit từ feature-branch, áp dụng commit từ main, sau đó áp dụng lại commit của feature-branch. Kết quả là lịch sử tuyến tính như thể các thay đổi được thực hiện liên tiếp.

Rebase tương tác

```
git rebase -i HEAD~3
```

Rebase tương tác cho phép điều chỉnh, kết hợp hoặc xóa commit trong quá trình rebase, giúp "dọn dẹp" lịch sử trước khi chia sẻ.

Lưu ý quan trọng: Không rebase các nhánh đã chia sẻ công khai. Rebase thay đổi lịch sử commit và có thể gây rắc rối cho người khác. Chỉ rebase nhánh local hoặc nhánh bạn làm việc một mình.

Các workflow & mô hình nhánh phổ biến



Tầm quan trọng của mô hình nhánh

Mô hình nhánh là quy ước sử dụng nhánh giúp đội phát triển làm việc nhất quán và hiệu quả. Việc lựa chọn mô hình phù hợp phụ thuộc vào quy mô dự án, số lượng thành viên và chu kỳ phát hành.



So sánh các mô hình

Mỗi mô hình có ưu điểm riêng: GitFlow phù hợp với chu kỳ phát hành dài, GitHub Flow đơn giản hơn cho triển khai liên tục, Trunk-based thúc đẩy tích hợp liên tục, và GitLab Flow cân bằng giữa đơn giản và kiểm soát.



Áp dụng cho dự án

Khi chọn mô hình, cân nhắc văn hóa đội, quy trình CI/CD và yêu cầu kinh doanh. Có thể bắt đầu đơn giản với GitHub Flow và phát triển theo thời gian. Quan trọng là mọi thành viên hiểu và tuân thủ mô hình đã chọn.

GitFlow Branching Model

GitFlow được Vincent Driessen giới thiệu năm 2010, **định nghĩa các loại nhánh và quy tắc tương tác**. Mô hình này phù hợp với dự án có chu kỳ phát hành rõ ràng, nhưng có thể phức tạp cho dự án nhỏ hoặc triển khai liên tục



Main branch

Chứa mã nguồn sẵn sàng triển khai. Mỗi commit được gắn tag số phiên bản và đại diện cho bản phát hành chính thức.



Develop branch

Tạo từ main, là nhánh tích hợp cho tất cả tính năng mới đã hoàn thành nhưng chưa sẵn sàng phát hành.



Feature branches

Tách từ develop (feature/*), mỗi tính năng một nhánh. Sau khi hoàn thành, merge lại vào develop.



Release branches

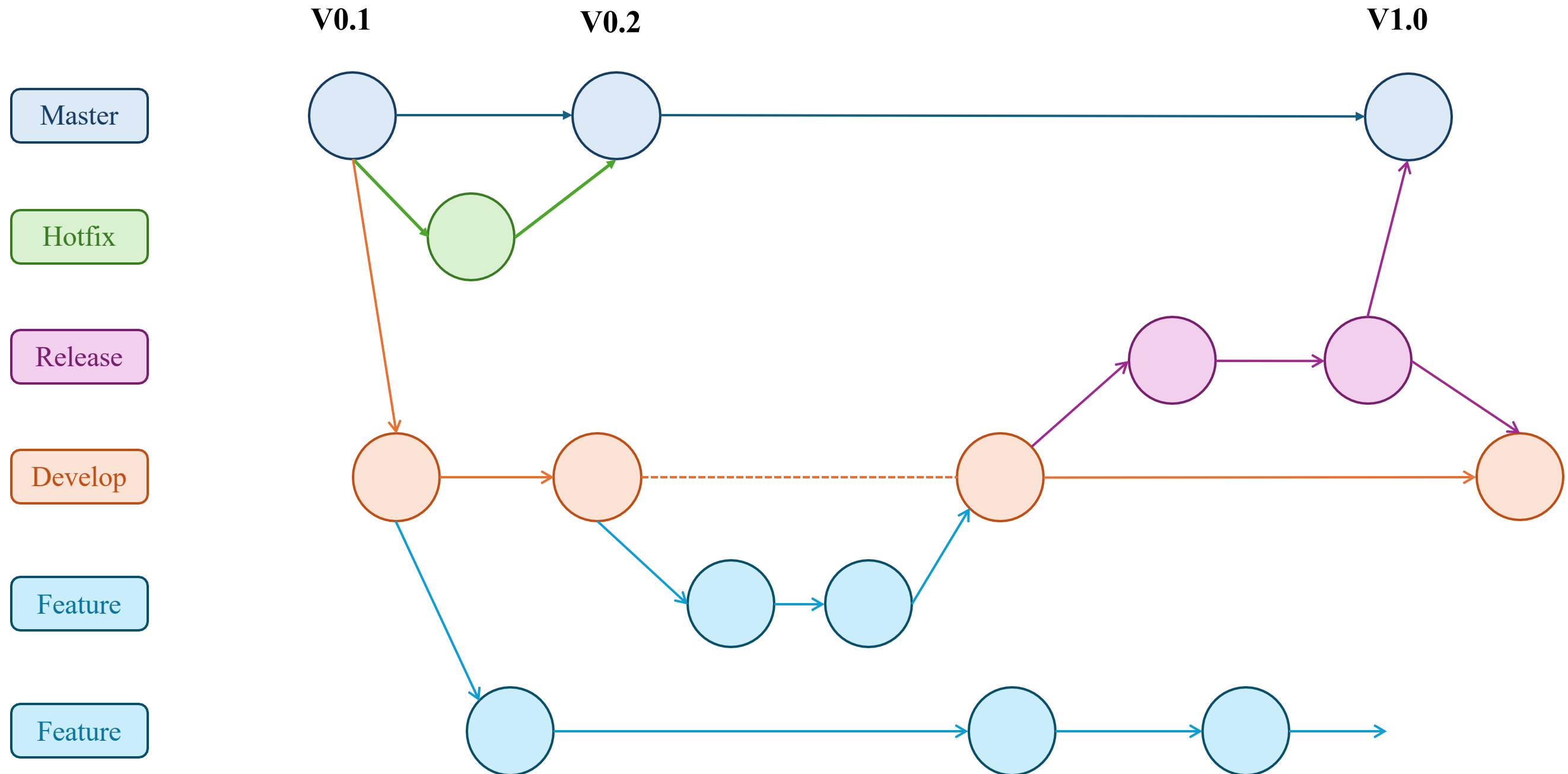
Tạo từ develop (release/*) khi chuẩn bị phát hành. Chỉ sửa lỗi, không thêm tính năng. Sau đó merge vào main và develop.



Hotfix branches

Tạo từ main (hotfix/*) để sửa lỗi khẩn cấp. Sau khi sửa, merge vào main và develop.

GitFlow Branching Model



Các workflow hiện đại

Chọn workflow dựa trên yêu cầu dự án và văn hóa đội. Xu hướng hiện nay là sử dụng workflow đơn giản hỗ trợ tốt cho triển khai và tích hợp liên tục.

Trunk-based Development

Phương pháp phát triển đơn giản với công việc diễn ra trực tiếp trên nhánh "trunk" (main):

- Commit trực tiếp vào main hoặc qua nhánh tính năng ngắn hạn
- Sử dụng feature flags kiểm soát tính năng mới
- Yêu cầu tích hợp liên tục (CI)
- Phù hợp với DevOps và triển khai nhiều lần mỗi ngày

GitHub Flow

Quy trình đơn giản cho triển khai liên tục:

1. Tạo nhánh từ main cho mỗi tính năng/lỗi
2. Commit và push thường xuyên
3. Mở Pull Request để review
4. Merge sau khi pass kiểm tra và được approve
5. Triển khai ngay sau khi merge

Lý tưởng cho các dự án web và ứng dụng có chu kỳ phát hành liên tục.

GitLab Flow

Kết hợp GitHub Flow với quản lý môi trường:

- Sử dụng nhánh production thay vì chỉ dùng tags
- Thêm các nhánh môi trường (staging, pre-production)
- Thay đổi di chuyển: main → môi trường kiểm thử → production
- Hai biến thể: nhánh môi trường hoặc nhánh phát hành

Giải quyết hạn chế của GitHub Flow với nhiều môi trường/phiên bản, đơn giản hơn GitFlow.

Tổng kết phần 3



Phân nhánh (Branch)

Hiểu được khái niệm, quy trình tạo, chuyển đổi và gộp nhánh. Phương pháp quản lý nhánh hiệu quả và làm việc với nhánh từ xa.



Rebase & Lịch sử

Nắm vững kỹ thuật rebase để tái cấu trúc lịch sử commit và tạo lịch sử dự án sạch hơn.



Mô hình nhánh

So sánh GitFlow và các workflow hiện đại như Trunk-based, GitHub Flow và GitLab Flow, hiểu ưu nhược điểm của từng loại.



Hợp tác hiệu quả

Áp dụng các mô hình phù hợp với quy mô dự án và văn hóa đội, cải thiện quy trình làm việc nhóm.

Agenda



Phần 1: Giới thiệu hệ thống quản lý phiên bản và Git

Khái niệm VCS, lược sử Git, nguyên lý hoạt động, cài đặt và cấu hình Git



Phần 2: Quy trình làm việc Git cơ bản

Khởi tạo repository, theo dõi thay đổi, xem lịch sử, hoàn tác, làm việc với remote



Phần 3: Quản lý nhánh và lịch sử

Tạo và gộp nhánh, quản lý nhánh remote, rebase, các workflow phổ biến



Phần 4: Cộng tác qua GitHub

Thiết lập tài khoản, quy trình Fork & Pull Request, quản lý repository, tự động hóa



Phần 5: Tùy chỉnh & mở rộng Git

Cấu hình nâng cao, Git Attributes, Git Hooks, chính sách kiểm soát, Alias & Scripts

Thiết lập tài khoản & bảo mật

1

Đăng ký tài khoản

Truy cập github.com để tạo tài khoản. Chọn tên người dùng phù hợp vì sẽ xuất hiện trong URL dự án. Xác minh email để kích hoạt đầy đủ tính năng.

2

Thiết lập phương thức xác thực

GitHub hỗ trợ HTTPS và SSH. Nên dùng SSH vì bảo mật hơn và không cần nhập mật khẩu liên tục. Tạo SSH key bằng lệnh `ssh-keygen -t ed25519 -C "email@example.com"` và thêm vào tài khoản GitHub.

3

Tùy chỉnh hồ sơ cá nhân

Cập nhật ảnh đại diện và thông tin liên hệ.

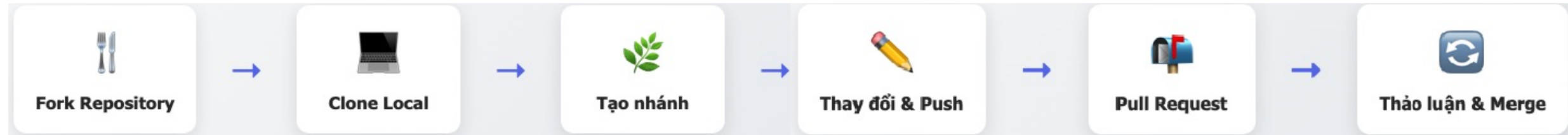
4

Bảo mật tài khoản

Bật xác thực hai yếu tố (2FA) qua ứng dụng, SMS hoặc khóa bảo mật vật lý. Kiểm soát quyền truy cập các ứng dụng bên thứ ba trong phần "Applications" trong cài đặt.

Quy trình Fork & Pull Request

Hướng dẫn chi tiết về quy trình đóng góp mã nguồn mở:



1 Fork repository

Tạo bản sao (fork) repository về tài khoản GitHub của bạn để có quyền chỉnh sửa độc lập. Nhấn nút "Fork" ở góc trên bên phải trang repository.

2 Clone về máy local

Clone repository đã fork về máy tính:

```
git clone https://github.com/username-của-bạn/tên-repository.git
cd tên-repository
```

Thiết lập upstream remote để đồng bộ với repo gốc:

```
git remote add upstream https://github.com/owner-gốc/tên-repository.git
```

3 Tạo nhánh chủ đề

Tạo nhánh mới từ nhánh chính (main/master):

```
git checkout -b tên-nhánh-mới
```

Đặt tên nhánh mô tả ngắn gọn về tính năng hoặc sửa lỗi, ví dụ: "fix-login-bug".

4 Thực hiện thay đổi và push

Sau khi hoàn thành thay đổi, commit và push:

```
git add .
git commit -m "Mô tả ngắn gọn về thay đổi"
git push origin tên-nhánh-mới
```

5 Tạo Pull Request

Truy cập repository gốc, nhấn "New pull request". Chọn nhánh của bạn làm nguồn và nhánh chính của repo gốc làm đích. Viết tiêu đề, mô tả chi tiết, sau đó nhấn "Create pull request".

6 Thảo luận và cập nhật

Chủ sở hữu sẽ xem xét và có thể yêu cầu điều chỉnh. Cập nhật theo yêu cầu bằng cách commit và push thêm - Pull Request sẽ tự động cập nhật. Sau khi được chấp thuận, PR sẽ được merge vào repository gốc.

Quản lý repository trên GitHub



Tạo và cấu hình repository

Tạo repository qua nút "+" và "New repository". Đặt tên, mô tả, chọn quyền riêng tư và tùy chọn khởi tạo. Thêm collaborator trong Settings để cấp quyền commit.



Tài liệu và hướng dẫn

Cung cấp README.md mô tả dự án, CONTRIBUTING.md hướng dẫn đóng góp và CODE_OF_CONDUCT.md thiết lập quy tắc. Các tài liệu này làm rõ mục tiêu và quy trình.



Mẫu Issue và Pull Request

Tạo mẫu trong thư mục .github đảm bảo thông tin đầy đủ. Mẫu Issue phân loại lỗi và yêu cầu. Mẫu PR cần mô tả thay đổi, cách kiểm tra và checklist chất lượng.



Thiết lập branch và bảo vệ

Cấu hình nhánh mặc định và quy tắc bảo vệ trong Settings > Branches. Kích hoạt bảo vệ nhánh chính để yêu cầu review, kiểm tra CI/CD và ngăn push trực tiếp.

Organization & Teams

Tạo và quản lý Organization

Organization quản lý nhiều repository và người dùng dưới một tài khoản, phù hợp cho công ty hoặc dự án lớn:

1. Nhấn "+" góc trên phải, chọn "New organization"
2. Chọn gói (Teams/Enterprise)
3. Nhập tên, email và chủ sở hữu

Chuyển repository cá nhân vào Organization: Settings > Options > Transfer ownership.

Quản lý Teams và phân quyền

Teams tổ chức thành viên và phân quyền truy cập:

1. Trong Organization, vào "Teams", nhấn "New team"
2. Đặt tên, mô tả và quyền riêng tư
3. Thêm thành viên
4. Cấp quyền repository:
 - **Read:** Xem và clone
 - **Write:** Push vào repository
 - **Admin:** Quản lý cài đặt

Teams có thể lồng nhau tạo cấu trúc phân cấp, thành viên kế thừa quyền từ team cha.

Nhật ký hoạt động và giám sát

Audit log theo dõi hoạt động trong GitHub Enterprise:

- Ghi lại người thực hiện, hành động và thời gian
- Theo dõi thay đổi quyền và thành viên
- Giám sát truy cập repository
- Xuất nhật ký để phân tích

Truy cập tại Settings > Audit log. Duy trì bảo mật và tuân thủ quy định.

Security alerts và Dependabot phát hiện và khắc phục lỗ hổng trong dependencies.

Mẹo làm việc hiệu quả

1 Sử dụng Draft Pull Request

Dùng Draft Pull Request để nhận phản hồi sớm về code đang phát triển mà không làm mất thời gian reviewer. Khi tạo Pull Request, chọn "Create draft pull request" thay vì "Create pull request".

2 Tạo review checklist

Phát triển checklist review để đảm bảo chất lượng code nhất quán. Checklist nên bao gồm coding standard, test coverage, hiệu suất và bảo mật. Thêm vào template Pull Request để mọi người tự kiểm tra trước khi yêu cầu review.

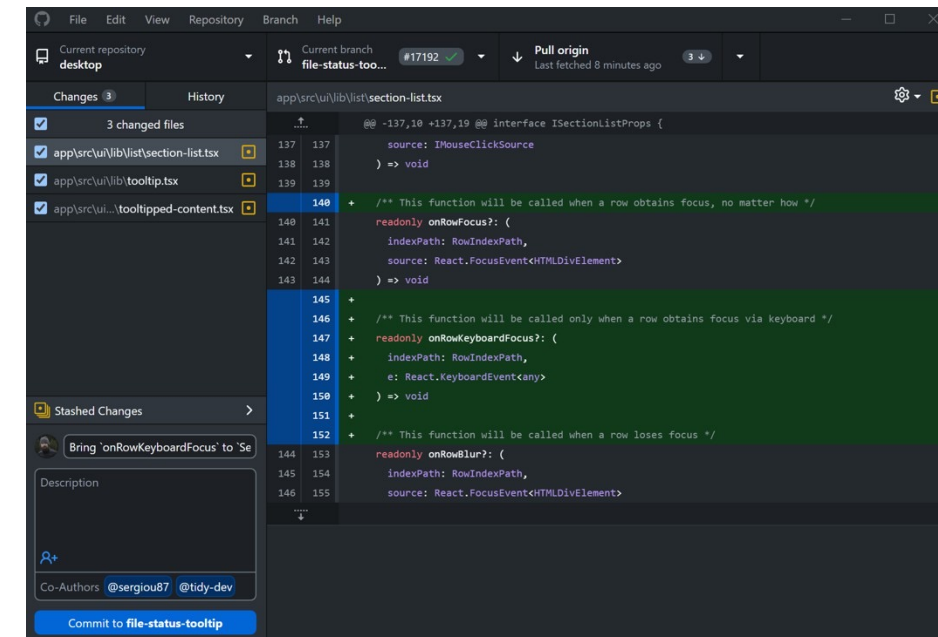
3 Đồng bộ hóa fork thường xuyên

Giữ fork đồng bộ với upstream repository để tránh xung đột và làm việc với code mới nhất:

```
git remote add upstream https://github.com/original/repo.git
git fetch upstream
git checkout main
git merge upstream/main
git push origin main
```

GitHub Desktop

GitHub Desktop là ứng dụng giao diện đồ họa giúp đơn giản hóa quy trình làm việc với Git và GitHub, phù hợp với người mới bắt đầu và người dùng ưa thích GUI.



Tính năng chính

- Giao diện trực quan, dễ sử dụng
- Theo dõi thay đổi trực quan bằng hình ảnh
- Quản lý nhánh và kho lưu trữ đơn giản
- Xử lý xung đột merge dễ dàng
- Tích hợp sẵn với GitHub.com

Cách sử dụng cơ bản

1. Tải và cài đặt từ desktop.github.com
2. Đăng nhập với tài khoản GitHub
3. Clone repository hoặc tạo repository mới
4. Thực hiện thay đổi và commit
5. Push thay đổi lên GitHub

Tổng kết Phần 4



Thiết lập tài khoản & bảo mật

Tạo và cấu hình tài khoản GitHub với xác thực hai yếu tố và quản lý khóa SSH.



Quy trình Fork & Pull Request

Thực hiện việc đóng góp vào dự án mã nguồn mở thông qua fork và gửi pull request.



Quản lý repository trên GitHub

Tùy chỉnh repository với README, license, issue templates và GitHub Pages.



Organization & Teams

Tổ chức cộng tác nhóm qua Organizations, Teams và phân quyền truy cập.



GitHub Desktop

Sử dụng giao diện đồ họa để đơn giản hóa quy trình làm việc với Git và GitHub.

Agenda



Phần 1: Giới thiệu hệ thống quản lý phiên bản và Git

Khái niệm VCS, lược sử Git, nguyên lý hoạt động, cài đặt và cấu hình Git



Phần 2: Quy trình làm việc Git cơ bản

Khởi tạo repository, theo dõi thay đổi, xem lịch sử, hoàn tác, làm việc với remote



Phần 3: Quản lý nhánh và lịch sử

Tạo và gộp nhánh, quản lý nhánh remote, rebase, các workflow phổ biến



Phần 4: Cộng tác qua GitHub

Thiết lập tài khoản, quy trình Fork & Pull Request, quản lý repository, tự động hóa



Phần 5: Tùy chỉnh & mở rộng Git

Cấu hình nâng cao, Git Attributes, Git Hooks, chính sách kiểm soát, Alias & Scripts

Cấu hình nâng cao (.gitconfig)

Thiết lập nhánh mặc định

Từ Git 2.28, thay đổi tên nhánh mặc định khi khởi tạo repository:

```
git config --global init.defaultBranch main
```

Chuyển tên nhánh mặc định từ "master" sang "main" khi chạy git init.

Công cụ xử lý merge và diff

```
git config --global merge.tool vscode
git config --global diff.tool vscode
git config --global difftool.vscode.cmd "code --wait --diff $LOCAL $REMOTE"
git config --global mergetool.vscode.cmd "code --wait $MERGED"
```

Mẫu commit message

Thiết lập mẫu đảm bảo tính nhất quán cho commit message:

```
git config --global commit.template ~/.gitmessage.txt
```

Cấu trúc mẫu:

```
# Tiêu đề: Tóm tắt ngắn gọn (< 50 ký tự)
# Nội dung: Giải thích thay đổi và lý do
# (Tối đa 72 ký tự/dòng)
# Issue liên quan: #123
```

Thiết lập proxy và xác thực

Cấu hình cho mạng doanh nghiệp có proxy:

```
git config --global http.proxy http://proxy.example.com:8080
git config --global https.proxy https://proxy.example.com:8080
```

Lưu thông tin đăng nhập:

```
git config --global credential.helper store # Lưu vĩnh viễn
git config --global credential.helper cache # Lưu tạm trong bộ nhớ
git config --global credential.helper 'cache --timeout=3600' # Lưu theo giây
```


Git Attributes (.gitattributes)

Khái niệm Git Attributes

File .gitattributes quy định cách Git xử lý các loại file. Cú pháp đơn giản:

```
pattern attr1 attr2 ...
```

Pattern là mẫu tên file (*.txt, docs/*.md), attr là hành vi áp dụng cho file.

Ví dụ:

```
# Chuẩn hóa line endings
* text=auto
```

```
# File binary
*.png binary
*.jpg binary
```

```
# Xác định merge
database.xml merge=ours
```

Xử lý kết thúc dòng

Giúp quản lý line endings giữa các hệ điều hành:

```
# Chuẩn hóa thành LF trong repository
*.txt text
*.java text
*.md text
```

```
# Chuyển sang CRLF khi checkout trên Windows
*.bat text eol=crlf
```

```
# Giữ nguyên line ending
*.sh text eol=lf
```

Giải quyết khác biệt giữa Windows (CRLF) và Unix/Mac (LF), tránh thay đổi không cần thiết.

Đánh dấu file binary và chiến lược merge

Thuộc tính binary giữ nguyên nội dung file:

```
*.png binary
*.jpg binary
*.zip binary
```

Quy định cách xử lý khi merge:

```
# Giữ file cấu hình local khi merge
config.xml merge=ours

# Sử dụng phiên bản từ nhánh đang merge
generated-code.js merge=theirs
```

Clean và Smudge Filters

Biến đổi file khi thêm vào hoặc lấy ra từ repository:

```
# Áp dụng filter "indent" cho file C
*.c filter=indent
```

```
# Mở rộng từ khóa cho file nguồn
*.java filter=keyword
```

Cài đặt filter trong .gitconfig:

```
[filter "indent"]
  clean = indent
  smudge = cat

[filter "keyword"]
  clean = sed
  "s/\\$Date[^\$]*\\$\\$Date\\$/"
  smudge = sed
  "s/\\$Date\\$\\$Date: $(date)\\$/"
```

"Clean" chạy khi thêm file vào repository, "smudge" khi lấy file ra.

Git Hooks – Tự động hóa

Giới thiệu về Git Hooks

Git Hooks là script tự động chạy tại các thời điểm cụ thể trong quy trình Git, giúp tự động hóa công việc và thực thi quy tắc. Chúng được lưu trong thư mục `.git/hooks/` dưới dạng file thực thi không đuôi.

Để sử dụng, chỉ cần bỏ đuôi `.sample` từ các file mẫu và cấp quyền thực thi (`chmod +x`).

Client-side Hooks

Hooks chạy trên máy của người phát triển:

- **pre-commit:** Chạy trước commit để kiểm tra code, chạy test
- **prepare-commit-msg:** Chạy trước khi soạn tin nhắn commit
- **commit-msg:** Kiểm tra định dạng tin nhắn commit
- **post-commit:** Chạy sau commit, thường dùng để thông báo
- **pre-push:** Chạy trước khi đẩy code lên remote

Server-side Hooks

Hooks chạy trên máy chủ:

- **pre-receive:** Chạy khi máy chủ nhận yêu cầu push
- **update:** Giống pre-receive nhưng chạy riêng cho mỗi thay đổi
- **post-receive:** Chạy sau khi push xong, kích hoạt CI/CD

Server-side hooks thực thi quy định hiệu quả vì người dùng không thể bỏ qua. Tuy nhiên, cần quyền truy cập máy chủ để cài đặt.

Alias & Scripts

Tạo Git Alias cơ bản

Alias trong Git cho phép bạn tạo lệnh tắt cho các lệnh dài hoặc phức tạp. Cấu hình alias trong file .gitconfig:

```
[alias]
  st = status
  co = checkout
  br = branch -v
  cm = commit -m
  l = log --oneline --graph --decorate
  unstage = restore --staged
  last = log -1 HEAD
  visual = !gitk
```

Sau khi cấu hình, bạn có thể sử dụng **git st** thay vì **git status**, giúp tiết kiệm thời gian đáng kể.

Alias nâng cao và hàm

Bạn có thể tạo alias phức tạp hơn bằng cách kết hợp nhiều lệnh:

```
[alias]
# Tạo nhánh mới và switch sang nhánh đó
nb = "!f() { git checkout -b $1; }; f"

# Xóa tất cả nhánh đã merge vào nhánh hiện tại
cleanup = "!git branch --merged | grep -v '\\*' | xargs -n 1 git branch -d"

# Cập nhật nhánh hiện tại từ upstream
sync = "!f() { git pull upstream $(git rev-parse --abbrev-ref HEAD); }; f"

# Xem lịch sử với giao diện đẹp
graph = "log --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit"
```

Các alias sử dụng **!** cho phép bạn chạy shell command hoặc định nghĩa hàm phức tạp.

Tổng Kết Phần 5



Cấu hình nâng cao (.gitconfig)

Tùy chỉnh sâu các thông số của Git thông qua file .gitconfig, bao gồm cài đặt user, editor và các tùy chọn khác để phù hợp với môi trường làm việc cá nhân.



Git Attributes (.gitattributes)

Quản lý cách Git xử lý các loại file khác nhau, thiết lập quy tắc diff, merge và xuất/nhập cho từng định dạng file cụ thể.



Git Hooks – Tự động hóa

Viết và triển khai các script tự động chạy trước hoặc sau các hành động Git như commit, push, giúp đảm bảo chất lượng mã nguồn và tiêu chuẩn nhóm.



Alias & Scripts

Tạo lệnh tắt cho các thao tác phức tạp, từ alias đơn giản đến các hàm chức năng nâng cao, giúp tối ưu hóa quy trình làm việc hàng ngày với Git.

Hands-on Git&Github

Bài tập thực hành 1

1 Cài đặt và cấu hình Git

Nhiệm vụ: Cài đặt và cấu hình Git cơ bản.

1. Cài đặt Git theo hướng dẫn cho hệ điều hành của bạn.
2. Cấu hình tên và email với git config.
3. Tạo alias cho lệnh status và log.
4. Kiểm tra cấu hình với git config --list.

2 Tạo và chuẩn bị repository

Nhiệm vụ: Khởi tạo repo và commit đầu tiên.

1. Tạo thư mục "git-practice".
2. Khởi tạo Git repository.
3. Tạo file README.md với nội dung giới thiệu.
4. Kiểm tra trạng thái với git status.
5. Đưa file vào staging area và tạo commit.

3 Thực hành với commit và lịch sử

Nhiệm vụ: Tạo commit và xem lịch sử.

1. Tạo file read_data.py cơ bản.
2. Commit file này.
3. Chỉnh sửa README.md, thêm thông tin dự án.
4. Commit thay đổi.
5. Xem lịch sử với git log và git log --oneline.
6. Dùng git show xem chi tiết commit.

4 Thực hành hoàn tác thay đổi

Nhiệm vụ: Học cách hoàn tác trong Git.

1. Sửa đổi nội dung trong read_data.py
2. Dùng git restore để hoàn tác thay đổi chưa staged.
3. Thêm nội dung mới, stage, sau đó dùng git restore --staged để bỏ stage.
4. Tạo commit, dùng git commit --amend sửa message.

Bài tập thực hành 2

1 Thực hành phân nhánh cơ bản

1. Sử dụng repository "git-practice" từ bài trước hoặc tạo mới.
2. Tạo nhánh mới: git branch feature-a
3. Chuyển nhánh: git checkout feature-a
4. Tạo file feature-a.txt với nội dung tùy chọn.
5. Commit file này.
6. Quay lại nhánh main và quan sát sự khác biệt.
7. Thực hành lệnh git checkout -b để tạo và chuyển nhánh đồng thời.

2 Gộp nhánh và xử lý xung đột

1. Ở nhánh main, tạo và commit file common.txt với nội dung "Dòng 1: Nội dung từ main".
2. Chuyển sang nhánh feature-a, tạo và commit file common.txt với nội dung khác.
3. Quay lại main và gộp: git merge feature-a
4. Khi xung đột xảy ra, sửa file để giữ cả hai nội dung.
5. Hoàn tất merge với git add và git commit.

3 Thực hành với rebase

1. Tạo nhánh "feature-b" từ main.
2. Tạo 2-3 commit trên nhánh này.
3. Quay lại main, tạo 1-2 commit khác.
4. Chuyển sang feature-b và chạy git rebase main.
5. Giải quyết xung đột nếu có, tiếp tục với git rebase -continue.
6. Dùng git log --graph --oneline xem cấu trúc lịch sử.
7. So sánh với cấu trúc khi dùng merge ở bài trước.

4 Mô phỏng quy trình GitFlow

1. Tạo repository mới "gitflow-practice".
2. Tạo nhánh develop từ main.
3. Tạo và commit thay đổi trong nhánh feature/login.
4. Gộp feature vào develop: git merge --no-ff feature/login
5. Tạo và commit sửa lỗi trong nhánh release/v1.0.
6. Gộp release vào cả main và develop, tạo tag v1.0.
7. Mô phỏng hotfix: tạo từ main, sửa lỗi, gộp về cả main và develop.

Bài tập thực hành 3

1 Fork và Clone dự án mã nguồn mở

Nhiệm vụ: Thực hành quy trình Fork và Clone dự án.

1. Tìm một dự án mã nguồn mở nhỏ trên GitHub, phù hợp với kỹ năng của bạn (ví dụ: một thư viện JavaScript cơ bản, một template HTML, v.v.)
2. Fork dự án này về tài khoản GitHub của bạn bằng cách nhấn nút "Fork"
3. Clone repository đã fork về máy local
4. Thiết lập remote "upstream" trỏ đến repository gốc
5. Xác minh remote đã thiết lập đúng bằng lệnh `git remote -v`

2 Tạo và đóng góp Pull Request

Nhiệm vụ: Thực hiện một đóng góp nhỏ cho dự án đã fork.

1. Tạo nhánh mới cho tính năng/sửa lỗi của bạn
2. Thực hiện một thay đổi nhỏ (sửa lỗi chính tả, cập nhật tài liệu, thêm comment)
3. Commit và push nhánh lên fork của bạn
4. Tạo Pull Request từ nhánh của bạn vào repository gốc
5. Viết mô tả chi tiết về thay đổi của bạn trong PR

3 Thực hành Code Review

Nhiệm vụ: Học cách review và nhận xét code trên GitHub.

1. Tìm một Pull Request đang mở trong dự án bạn quan tâm
2. Đọc kỹ code và các thay đổi được đề xuất
3. Thêm comment vào các dòng code cụ thể (nếu có nhận xét)
4. Viết một review tổng thể (Approve, Comment, hoặc Request Changes)
5. Nếu làm việc với bạn học, hãy review PR của nhau

4 Quản lý dự án với GitHub

Nhiệm vụ: Thiết lập và quản lý dự án nhỏ trên GitHub.

1. Tạo repository mới cho một "dự án cá nhân" đơn giản
2. Thiết lập README.md với mô tả chi tiết và hướng dẫn cài đặt
3. Tạo CONTRIBUTING.md với hướng dẫn đóng góp
4. Thiết lập Issue templates cho báo lỗi và yêu cầu tính năng
5. Tạo một số Issues mẫu để theo dõi công việc
6. Thử kích hoạt GitHub Pages để xuất bản tài liệu dự án

Tổng Kết



Hiểu về VCS và Git

Nắm vững khái niệm, nguyên lý hoạt động và cấu hình Git cơ bản



Quy trình làm việc cơ bản

Thành thạo commit, push, pull và quản lý các thay đổi



Quản lý nhánh và lịch sử

Sử dụng branch, merge và các workflow chuyên nghiệp



Cộng tác qua GitHub

Fork, Pull Request và quản lý dự án hiệu quả



Tùy chỉnh & mở rộng

Cấu hình nâng cao, Git Hooks và tự động hóa quy trình

Tài liệu tham khảo



Tài liệu chính thức

- Pro Git Book: <https://git-scm.com/book/en/v2>
- Git: <https://git-scm.com/doc>
- GitHub: <https://docs.github.com>
- GitHub Guides: <https://guides.github.com>



Khóa học trực tuyến

- Atlassian Tutorials: <https://www.atlassian.com/git/tutorials>
- GitHub Learning Lab: <https://lab.github.com>
- Codecademy: <https://www.codecademy.com/learn/learn-git>



Blog và bài viết

- Atlassian: <https://www.atlassian.com/blog/git>
- Git Tower: <https://www.git-tower.com/blog>
- GitHub: <https://github.blog>

Công cụ trực quan (GUI)

- GitKraken: <https://www.gitkraken.com>
- Sourcetree: <https://www.sourcetreeapp.com>
- GitHub Desktop: <https://desktop.github.com>
- VS Code + Git Extensions

Công cụ dòng lệnh

- GitHub CLI: <https://cli.github.com>
- Lazygit: <https://github.com/jesseduffield/lazygit>
- tig: <https://jonas.github.io/tig>
- diff-so-fancy: <https://github.com/sofancy/diff-so-fancy>

Tiện ích mở rộng

- Git Flow: <https://github.com/petervanderdoes/gitflow-avh>
- Git LFS: <https://git-lfs.github.com>
- Husky: <https://typicode.github.io/husky>
- Conventional Commits: <https://www.conventionalcommits.org>