

Quản lý source code với GIT

Nội dung

1	Git là gì?	2
1.1	Lịch sử quản lý mã nguồn	2
1.2	Mô hình hoạt động của Git.....	4
2	Sử dụng Git	8
2.1	Cài đặt môi trường.....	8
2.2	Thiết lập cấu hình môi trường	9
2.3	Các thao tác trên local repository	10
2.4	Các thao tác trên remote repository	12
2.5	Branching	14
3	Sử dụng Git bằng công cụ đồ họa	15
4	Bài tập.....	15
4.1	Bài tập 1.....	15
4.2	Bài tập 2.....	16
4.3	Bài tập 3.....	16
4.4	Bài tập 4.....	16
5	Tóm tắt	17
6	Tài liệu tham khảo.....	18

1 Git là gì?

Git là một hệ thống quản lý source code phân tán (distributed revision/version control system) được phát triển bởi Linus Torvalds vào năm 2005. Một số đặc điểm chính của Git:

- Mã nguồn mở, miễn phí.
- Tốc độ nhanh (dữ liệu được nén trước khi gửi).
- Kiến trúc đơn giản, dễ học, dễ sử dụng.
- Phân tán hoàn toàn.

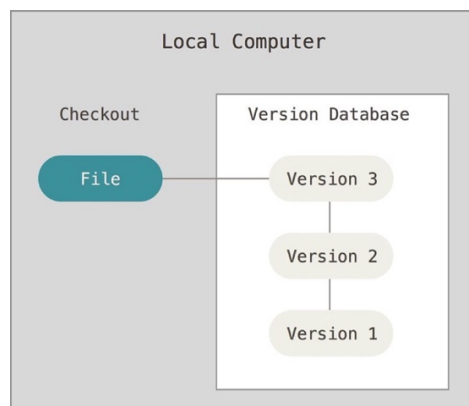
Tại sao phải dùng Git?

- Các dự án thường có nhiều thành viên phát triển song song với nhau nên cần phải có 1 hệ thống *kiểm soát phiên bản* để đảm bảo không có xung đột giữa các thành viên.
- Các yêu cầu trong một dự án có thể thay đổi thường xuyên, do đó một hệ thống quản lý phiên bản cho phép các nhà phát triển có thể *quay lại phiên bản cũ* của source của họ.
- Mỗi thành viên trong dự án có thể phát triển một tác vụ, chức năng riêng lẻ. Do đó, họ cần có một hệ thống cho phép *phân nhánh các công việc* và *tích hợp* vào công việc chung khi hoàn thành.

Một hệ thống Git sẽ hỗ trợ tất cả các tính năng trên, đáp ứng các nhu cầu quản lý mã nguồn cho các dự án.

1.1 Lịch sử quản lý mã nguồn

1.1.1 Local version control systems



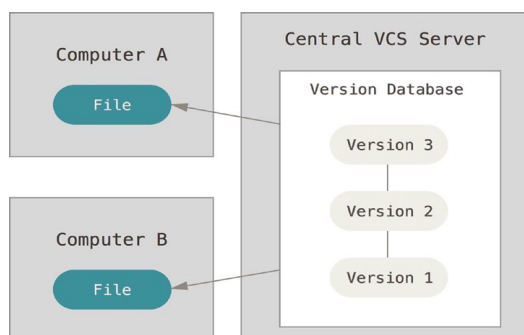
Hình 1. Mô hình Local version control (Nguồn: Pro Git)

Một số đặc điểm của mô hình Local version control:

- Các phiên bản chỉ lưu cục bộ trên máy người dùng.
- Sử dụng lệnh rcs để tạo các phiên bản.

- Rất khó phối hợp trong nhóm nhiều người.
- Hiện đã lỗi thời và hầu như không còn được sử dụng.

1.1.2 Centralized version control systems

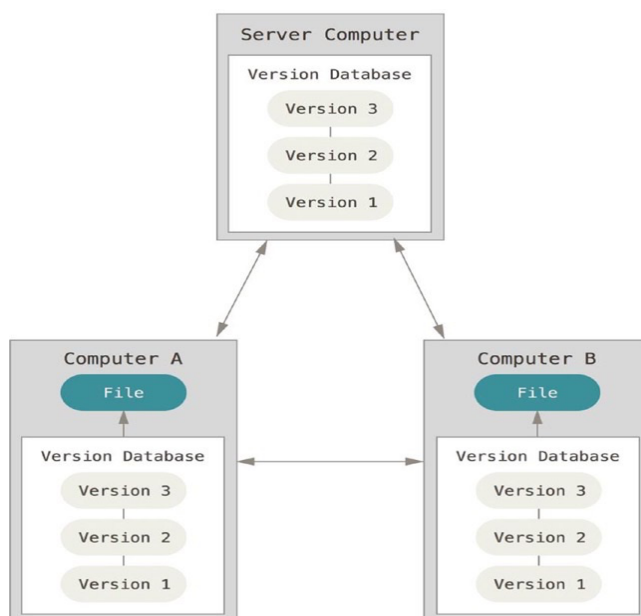


Hình 2. Mô hình Centralized version control system

Trong mô hình này, lịch sử các version, thay đổi được lưu trữ ở 1 server. Một trong những hệ thống phổ biến sử dụng mô hình này là SVN (subversion). Mô hình này khắc phục được nhược điểm của mô hình Local version control như tất cả mọi người đều biết được tiến độ của dự án, biết được các thành viên khác đang làm gì,... nhưng vẫn còn 1 số hạn chế sau:

- Khi server bị hỏng thì sẽ bị mất dữ liệu.
- Không thể hoạt động nếu không có nối kết đến máy chủ.
- Tốc độ chậm.

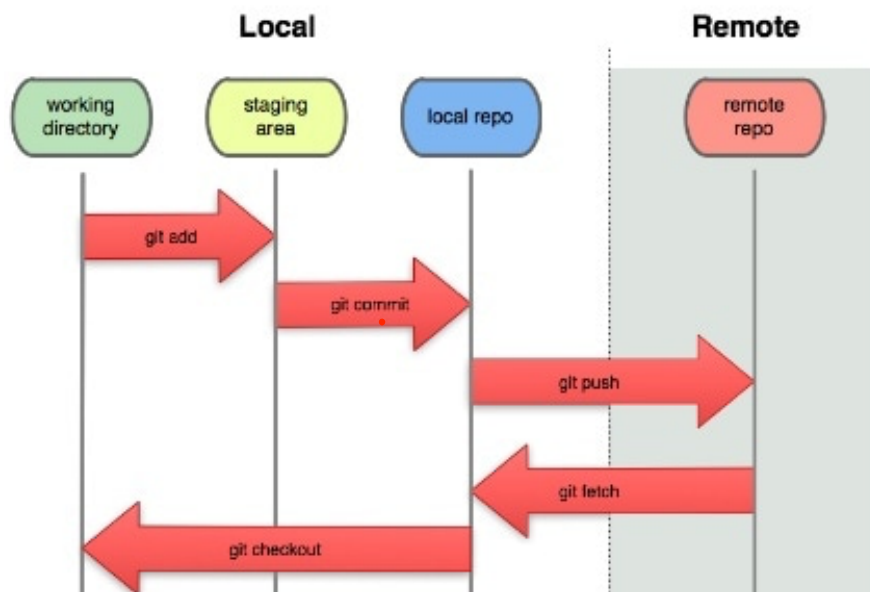
1.1.3 Distributed version control systems



Hình 3. Mô hình Distributed version control system

Trong mô hình này thì các client không chỉ lưu trữ snapshot mới nhất của dữ liệu mà còn sao chép toàn bộ repository của server. Do đó, nếu server bị sự cố thì có thể được phục hồi dựa trên bản sao lưu ở các client. Thao tác checkout sẽ thực hiện sao lưu toàn bộ repository. Một số hệ thống thông dụng sử dụng mô hình này GitHub, GitLab, BitBucket,...

1.2 Mô hình hoạt động của Git



Hình 4. Mô hình hoạt động của Git

Working directory: là nơi chứa các file mới được tạo, các file cũ bị xóa hoặc các file đang được cập nhật. Sau khi các thay đổi được thực hiện, chúng sẽ được thêm vào staging area. Working directory còn được xem là một single checkout của một phiên bản của project. Mỗi khi project được checkout thì Git sẽ kéo các file trong repository sẽ được kéo về lưu trong working directory, sẵn sàng cho việc truy xuất, thay đổi.

Staging area: là một tập tin, chứa các thông tin về các thay đổi đã thực hiện trên working directory và các thao tác sẽ được thực hiện trong lần commit kế tiếp. Tập tin này được đặt trong thư mục Git và còn được gọi là chỉ mục (index) của Git.

Local repository: chứa tất cả các tập tin đã được commit trên máy tính cục bộ. Thông thường, một local repository sẽ được chứa trong một thư mục ẩn có tên `.git`. Mỗi commit có thể coi như một checkpoint. Do đó, mỗi khi commit thì Git sẽ kiểm tra các file sẽ được commit với các file trong lần commit trước đó để chỉ lưu trữ những tập tin có thay đổi.

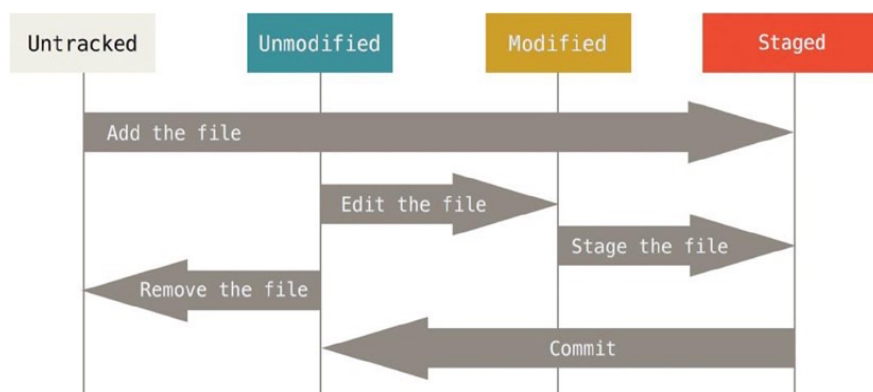
Remote repository: là một kho đơn thuần chứa trên máy tính ở xa, không có working directory. Remote repository được sử dụng như là một điểm cộng tác (collaboration point) nên không cần lưu trữ working directory. Hiểu một cách đơn giản, nội dung của remote repository chính là một bản sao của thư mục `.git`.

1.2.1 Làm việc với local repository

Mô hình hoạt động của Git trên *local repository* có thể được mô tả như sau:

- 1) Khởi tạo từ 1 repository từ 1 repository có sẵn hoặc khởi tạo 1 local repository rỗng.
- 2) Thêm các tập tin vào working directory. Các tập tin này được gọi là trong trạng thái *untracked* (không được theo dõi). Khi này, các thay đổi trên file sẽ không được theo dõi cũng như sẽ không được thêm vào repository khi ta thực hiện thao tác commit.
- 3) Để các tập tin, thư mục được thêm vào repository thì đầu tiên phải đưa các tập tin, thư mục mới này vào trạng thái *tracked* (được theo dõi bởi Git) và sau đó là đưa các tập tin này vào trạng thái *staging*.
- 4) Một tập tin đang trong trạng thái *tracked* hoặc *staging* mà có thay đổi nội dung thì sẽ chuyển sang trạng thái *modified*. Các tập tin này cần được đưa trở vào staging trước khi commit lên repository.

Trạng thái của một tập tin trên *local repository* được mô tả trong Hình 5. Để xem trạng thái của các tập tin trong local repository, ta sử dụng lệnh `git status`.



Hình 5. Các trạng thái của tập tin trên local repository

1.2.2 Làm việc với remote repository

Remote repository là repository được lưu trữ từ xa trên internet. Đây là 1 phương tiện để cộng tác với nhau trong việc thực hiện 1 dự án. Cộng tác với những thành viên khác trong 1 dự án bao gồm việc đẩy (push) và kéo (pull) dữ liệu đến/từ remote repository khi chúng ta cần chia sẻ hay cộng tác. Quản lý một remote repository bao gồm một số thao tác: thêm/xóa 1 remote repository, quản lý các branches và xác định nhánh nào cần theo dõi,...

1.2.3 Git branching

Như đã đề cập trong phần trên, Git không lưu trữ các thay đổi trong repository mà lưu trữ các snapshot của repository. Khi ta thực hiện commit, Git sẽ lưu trữ một đối tượng commit chứa một con trỏ tới bản snapshot của nội dung đã được staged. Đối tượng này cũng tên và email của người

đã thực hiện commit cùng với một số thông tin khác như: nội dung của commit message, con trỏ tới commit trước đó (được gọi là parent commit). Một commit “bình thường” sẽ có 1 parent commit, còn một commit được merged từ nhiều nhánh khác nhau.

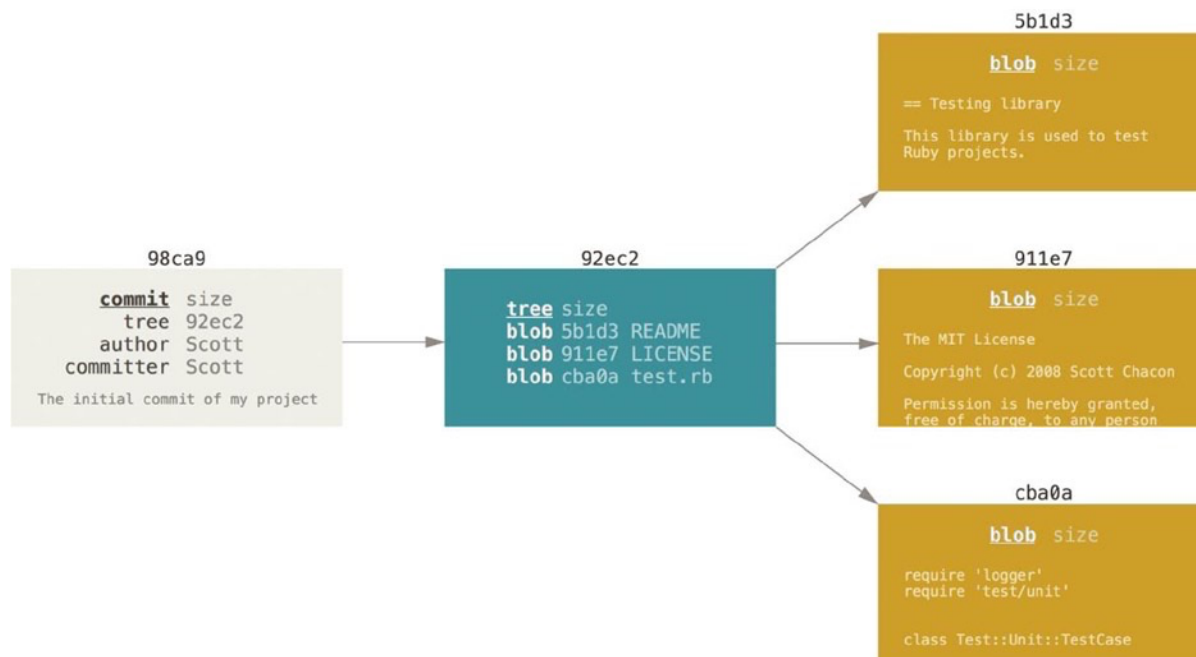
Hình 6 mô tả một commit tree khi ta thêm commit 3 tập tin, sau đó stage và commit các file này lên repository.

```
$ git add README test.rb LICENSE
```

```
$ git commit -m 'initial commit of my project'
```

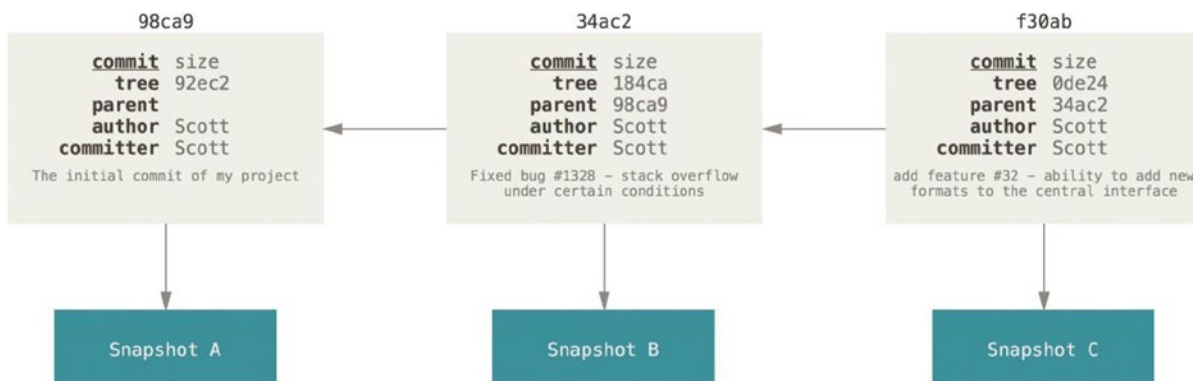
Khi stage một tập tin (`git add`), tập tin sẽ được checksum bằng thuật toán SHA-1, sau đó lưu trữ phiên bản của tập tin vào Git repository (được gọi là blob) và thêm checksum đó vào vùng staging. Khi thực hiện một commit (`git commit`), Git sẽ checksum các thư mục con (trong trường hợp này là thư mục root của project) và lưu trữ các *tree objects* đó vào Git repository. Sau đó, Git tạo một *commit object* chứa commit metadata và một con trỏ trỏ đến root project tree. Như vậy, sau khi thêm 3 tập tin như trên thì có 5 đối tượng được tạo ra và lưu trữ trong Git repository, bao gồm:

- 3 blobs objects, mỗi blob chứa nội dung của 1 tập tin được commit (5b1d3, 911e7, cba0a).
- 1 tree object (92ec2) chứa danh sách các tập tin trong thư mục và tên tập tin tương ứng với từng blob.
- 1 commit object (98ca9) chứa các thông tin như được mô tả ở trên.



Hình 6. Hình ảnh của một commit tree. Nguồn: Pro Git

Nếu có sự thay đổi trong project và commit được thực hiện thì một commit tree khác được tạo ra và commit object mới sẽ chứa 1 con trỏ trỏ tới commit object trước đó như được minh họa trong Hình 7.



Hình 7. Dây chuyền các commit objects. Nguồn: Pro Git

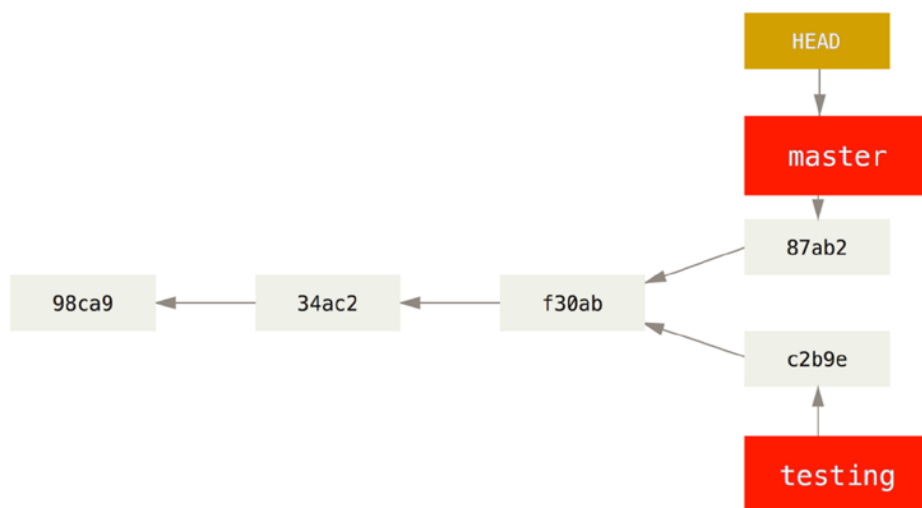
Một branch là một con trỏ (pointer) trỏ đến một commit và nó có thể di chuyển giữa các snapshot (movable pointer). Khi tạo một repository với lệnh `git init` thì Git sẽ tạo một branch mặc định với tên là `master`. Mỗi khi một commit được thực hiện, con trỏ này sẽ tự động di chuyển đến commit object mới.

Khi tạo một branch mới, một con trỏ với tên tương ứng sẽ được tạo ra và trỏ đến commit hiện tại. Hình 8 mô tả trạng thái của Git sau khi một branch mới với tên là `testing`. Hai branch này sẽ trỏ tới cùng commit.



Hình 8. Trạng thái của Git sau khi thêm 1 branch mới. Nguồn: Pro Git

Để biết người dùng làm việc trên branch nào, Git có 1 con trỏ đặc biệt với tên là `HEAD`, trỏ đến branch mà người dùng đang làm việc. Mỗi khi một commit được thực hiện thì một snapshot sẽ được tạo ra và con trỏ của branch hiện tại sẽ di chuyển đến commit mới. Như vậy, với cơ chế branching thì một project có thể có nhiều nhánh lịch sử commit khác nhau như minh họa trong Hình 9.



Hình 9. Git với nhiều nhánh history commit. Nguồn: Pro Git

Muốn làm việc trên nhánh nào thì người dùng phải chuyển (switch) đến nhánh tương ứng. Lưu ý rằng, khi chuyển branch thì các tập tin trong working directory có thể bị thay đổi. Working directory sẽ được cập nhật bởi các tập tin trong bản commit mới nhất của branch đó.

2 Sử dụng Git

2.1 Cài đặt môi trường

2.1.1 Đối với hệ điều hành Windows

Có nhiều các cài đặt Git trên hệ điều hành Windows. Phổ biến nhất là download bộ cài đặt Github for Windows tại địa chỉ <http://windows.github.com>. Bộ cài đặt này bao gồm cả phiên bản giao diện dòng lệnh và đồ họa.

2.1.2 Đối với hệ điều hành Linux

Cách thông dụng nhất để cài đặt Git trên Linux là dùng các công cụ quản lý package như **yum** hoặc **apt-get**:

```
yum install git
```

hoặc:

```
apt-get install git
```

Để kiểm tra kết quả cài đặt, ta gõ vào lệnh:

```
git --version
```

Nếu hiển thị phiên bản của Git thì việc cài đặt là thành công.

2.1.3 Trên hệ điều hành Mac OS

Có nhiều cách cài đặt Git trong Mac OS. Phương pháp dễ dàng nhất là sử dụng bộ cài đặt Git download tại địa chỉ <http://git-scm.com/download/mac>.



Hình 10. Giao diện của bộ cài đặt Git

Tương tự như việc cài đặt trên Linux, muốn kiểm tra việc cài đặt thì ta gõ lệnh `git --version` trong cửa sổ lệnh (Terminal). Nếu hiển thị phiên bản của Git thì có nghĩa là việc cài đặt thành công.

2.2 Thiết lập cấu hình môi trường

Sau khi cài đặt Git xong, ta phải thiết lập môi trường bằng lệnh `git config`.

- Thiết lập *định danh* (identity): bao gồm tên người dùng và địa chỉ email.

```
git config --global user.name <username>
```

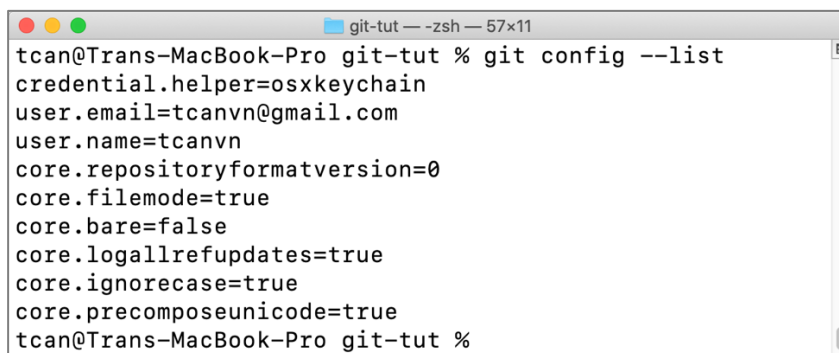
```
git config --global user.email <email>
```

- Thiết lập *trình soạn thảo* mặc định (editor): đây là trình soạn thảo sẽ được sử dụng khi Git muốn yêu cầu người dùng nhập vào 1 message.

```
git config --global core.editor emacs
```

- Kiểm tra cấu hình:

```
git config --list
```



```
tcan@Trans-MacBook-Pro git-tut % git config --list
credential.helper=osxkeychain
user.email=tcanvn@gmail.com
user.name=tcanvn
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
core.ignorecase=true
core.precomposeunicode=true
tcan@Trans-MacBook-Pro git-tut %
```

Hình 11. Hiển thị cấu hình Git với `git config`

2.3 Các thao tác trên local repository

Repository là 1 kho chứa, nơi lưu trữ mã nguồn cho phép nhiều người có thể sao chép (clone) mã nguồn đó để cùng phát triển. Có hai loại kho chứa là Local repository (kho chứa cục bộ) và Remote repository (kho chứa trên máy chủ từ xa). Kho chứa cục bộ sẽ được tạo trên máy tính cục bộ của người dùng còn kho chứa từ xa sẽ được tạo trên 1 máy chủ ở xa.

Để quản lý mã nguồn dự án trên local repository, ta thực hiện theo các bước sau:

- 1) Tạo thư mục để chứa Local repository:

```
mkdir <tên thư mục dùng làm kho chứa>
```

- 2) Khởi tạo kho chứa:

Di chuyển vào thư mục vừa tạo và gõ vào lệnh sau:

```
git init
```

Sau khi thực hiện lệnh trên thì Git sẽ tạo 1 thư mục ẩn **.git** chứa tất cả những thiết lập cũng như toàn bộ thông tin về kho chứa. Thư mục này dành riêng cho Git và người dùng sẽ không cần truy cập vào thư mục này. Tất cả các tập tin, thư mục của dự án sẽ lưu vào trong thư mục chứa Local repository.

- 3) Thực hiện dự án: thêm các tập tin, thư mục vào thư mục của dự án. Các tập tin, thư mục mới thêm vào sẽ ở trạng thái **untracked**.
- 4) Chuyển các tập tin, thư mục của dự án từ trạng thái **untracked** về trạng thái **tracked**:

```
git add <tên tập tin | tên thư mục | *>
```

Sau đó, ta có thể xem lại trạng thái của các tập tin trong dự án:

```
git status
```

Lưu ý: Sau khi sử dụng lệnh `git add` thì tập tin vừa được thêm vào sẽ được chuyển vào vùng *staging* (index), sẵn sàng được commit. Những tập tin đang trong vùng *staging*, nếu

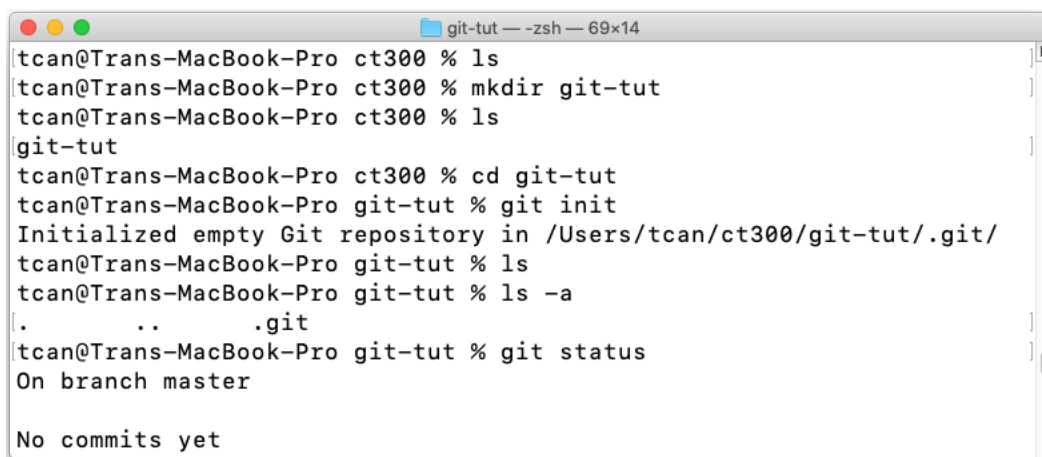
bị thay đổi thì sẽ bị cho ra khỏi vùng *staging*. Muốn đưa những tập tin này trở lại vùng *staging* thì phải sử dụng lại lệnh `git add`.

- 5) Tiến hành commit để lưu toàn bộ nội dung trong vùng *staging* vào CSDL của Git:

```
git commit -m <ghi chú về commit>
```

Lệnh này có thêm tham số `-a` cho phép đưa các tập tin đang trong trạng thái *tracked* vào vùng *staging* một cách tự động trước khi commit (tương đương với việc thực hiện lệnh `git add` cho tất cả các tập tin trước khi commit).

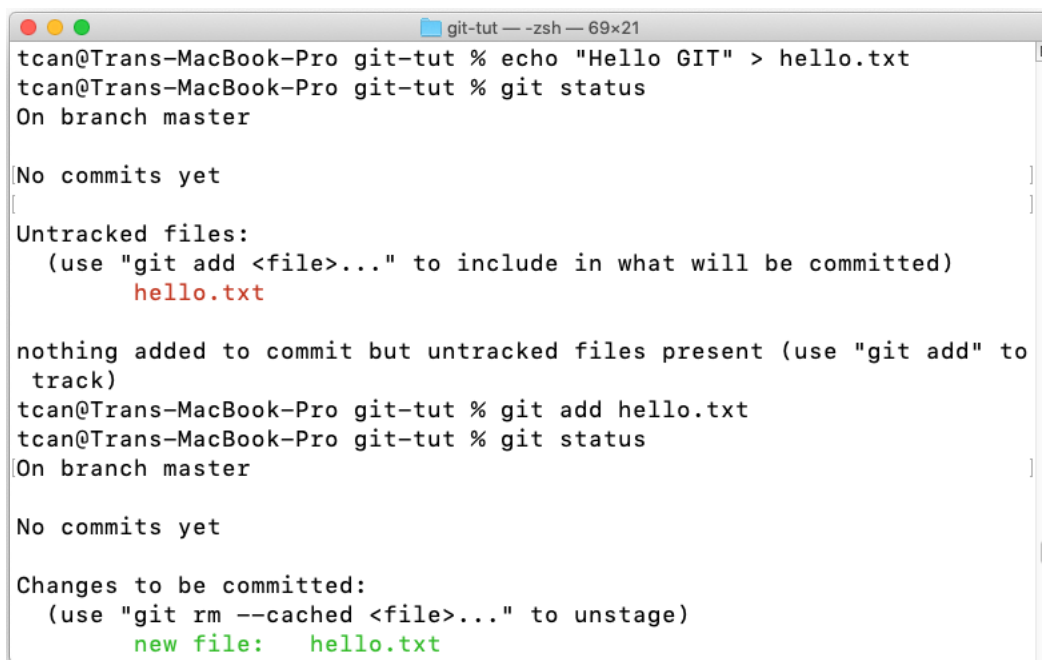
Các hình sau minh họa cho các lệnh được giới thiệu bên trên:



```
git-tut -- -zsh -- 69x14
tcan@Trans-MacBook-Pro ct300 % ls
tcan@Trans-MacBook-Pro ct300 % mkdir git-tut
tcan@Trans-MacBook-Pro ct300 % ls
git-tut
tcan@Trans-MacBook-Pro ct300 % cd git-tut
tcan@Trans-MacBook-Pro git-tut % git init
Initialized empty Git repository in /Users/tcan/ct300/git-tut/.git/
tcan@Trans-MacBook-Pro git-tut % ls
tcan@Trans-MacBook-Pro git-tut % ls -a
.  ..  .git
tcan@Trans-MacBook-Pro git-tut % git status
On branch master

No commits yet
```

Hình 12. Tạo 1 repository rỗng



```
git-tut -- -zsh -- 69x21
tcan@Trans-MacBook-Pro git-tut % echo "Hello GIT" > hello.txt
tcan@Trans-MacBook-Pro git-tut % git status
On branch master

No commits yet

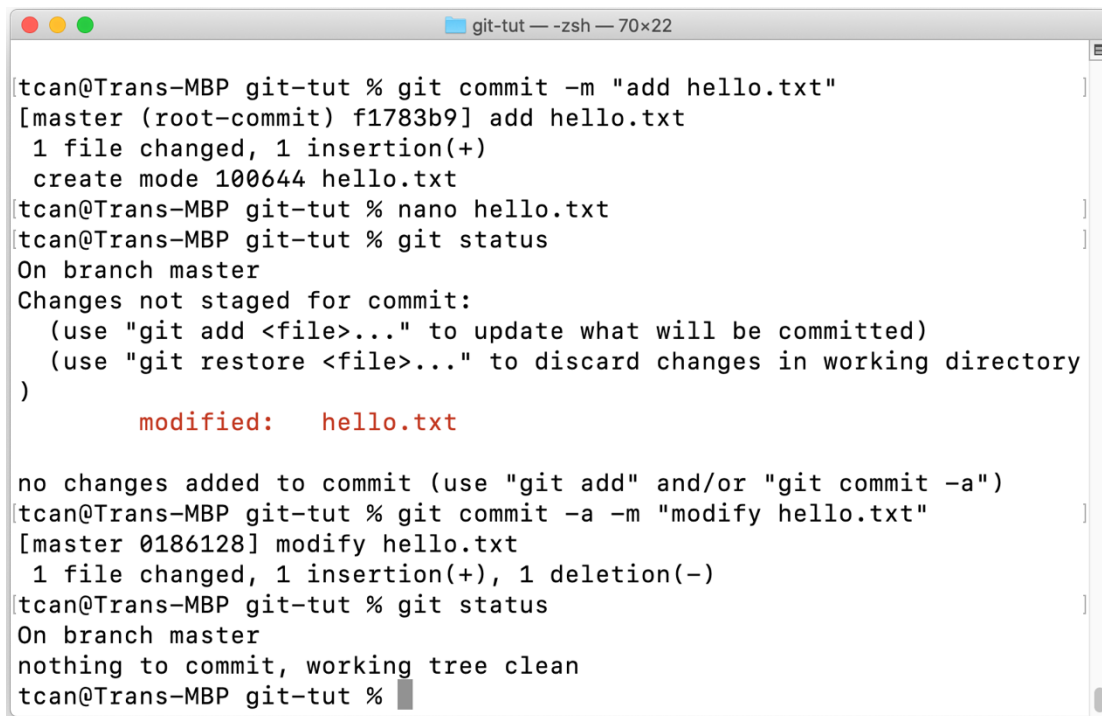
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        hello.txt

nothing added to commit but untracked files present (use "git add" to track)
tcan@Trans-MacBook-Pro git-tut % git add hello.txt
tcan@Trans-MacBook-Pro git-tut % git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   hello.txt
```

Hình 13. Thêm 1 file vào thư mục chứa code và chuyển file sang trạng thái *staged*



```
git-tut -- zsh -- 70x22

tcan@Trans-MBP git-tut % git commit -m "add hello.txt"
[master (root-commit) f1783b9] add hello.txt
 1 file changed, 1 insertion(+)
 create mode 100644 hello.txt
tcan@Trans-MBP git-tut % nano hello.txt
tcan@Trans-MBP git-tut % git status
On branch master
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:   hello.txt

no changes added to commit (use "git add" and/or "git commit -a")
tcan@Trans-MBP git-tut % git commit -a -m "modify hello.txt"
[master 0186128] modify hello.txt
 1 file changed, 1 insertion(+), 1 deletion(-)
tcan@Trans-MBP git-tut % git status
On branch master
nothing to commit, working tree clean
tcan@Trans-MBP git-tut %
```

Hình 14. Commit và sửa đổi tập tin

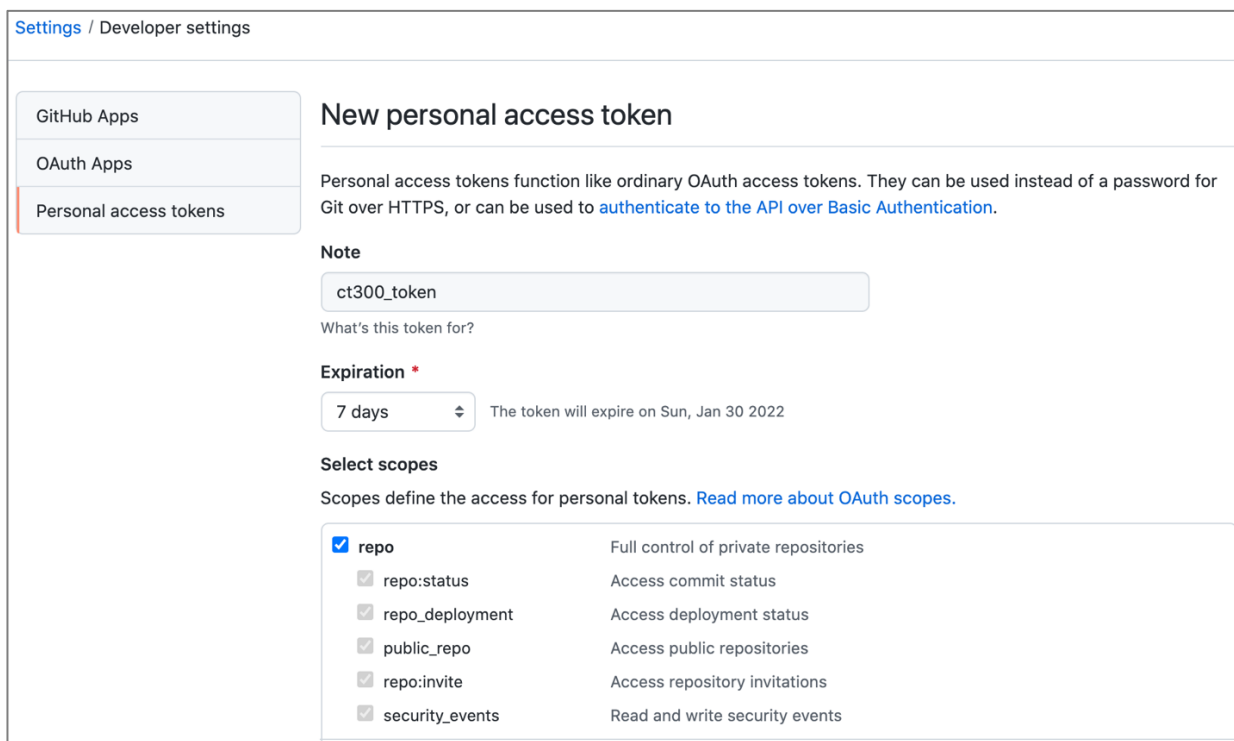
2.4 Các thao tác trên remote repository

Để đẩy (push) code từ 1 Local repository lên 1 remote repository, ta dùng câu lệnh `git push` với cú pháp như sau:

```
git push <remote_repository> <branch>
```

Trong đó, `remote_repository` là địa chỉ của remote server, `branch` là nhánh của repository mà ta muốn push code lên. Trong bước này, Git sẽ yêu cầu ta *chứng thực* để có thể push code lên remote repository. Phương pháp chứng thực sẽ tùy vào remote repository. Ví dụ, Github thì dùng PAT (Personal Access Token). Để lấy token để truy cập vào Github thì ta thực hiện các bước sau:

- 1) Chọn: Setting => Developer Settings => Personal Access Token => Generate New Token.
- 2) Chọn các quyền sẽ cấp cho token này.
- 3) Chọn Generate token. Đây là giá trị mà ta sẽ nhập vào khi Git yêu cầu nhập mật khẩu.



Hình 15. Tạo token trên Github

Sau khi đăng nhập và push code thành công, ta có thể lưu lại thông tin đăng nhập vào cache để các lần push code lần sau ta sẽ không cần phải nhập thông tin đăng nhập nữa.

```
git config --global credential.helper cache
```

Muốn xóa cache thì ta dùng lệnh sau:

```
git config --global --unset credential.helper
```

Ta có thể đặt tên cho remote repository để dễ nhớ và thuận tiện cho việc push code lên trong những lần sau bằng lệnh sau:

```
git remote add <remote_repository_name> <remote_repository_link>
```

Hình 16 mô tả quá trình push code lên một remote repository.



Hình 16. Push code lên remote repository

2.5 Branching

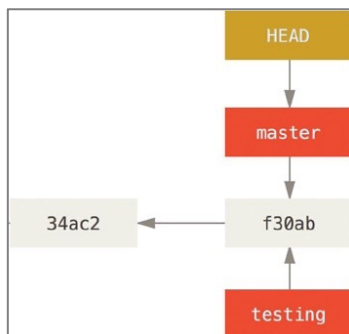
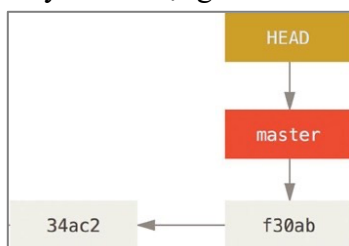
Để tạo 1 nhánh mới, ta dùng lệnh `git branch` như sau:

```
git branch <branch name>
```

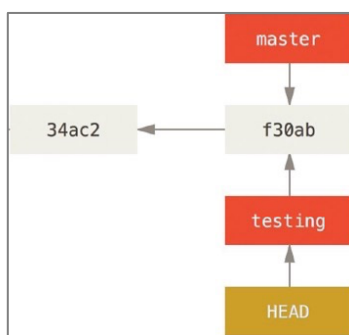
Sau khi tạo nhánh mới thì con trỏ HEAD (xem Phần 1.2.3) vẫn còn trỏ đến nhánh hiện tại (current branch). Nếu muốn chuyển sang nhánh mới thì ta dùng lệnh `git checkout` như sau:

```
git checkout <branch name>
```

Các hình sau minh họa quá trình chuyển đổi trạng thái của Git khi sử dụng các lệnh trên:

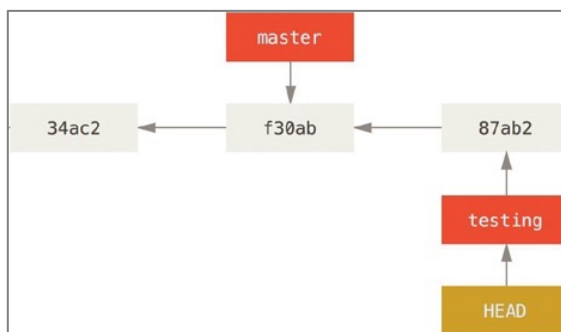


`git branch testing:`



`git checkout testing:`

```
git commit -a -m "...":
```



Muốn ghép (merge) một nhánh vào nhánh hiện hành, ta dùng lệnh `git merge` như sau:

```
git merge <branch name>
```

Trong quá trình merge, có thể xảy ra xung đột (conflict) giữa các tập tin trong nhánh hiện hành với nhánh cần ghép. Khi đó, ta cần phải giải quyết xung đột trước khi merge.

3 Sử dụng Git bằng công cụ đồ họa

Hiện ngoài công cụ dòng lệnh thì có một số công cụ đồ họa cho phép thao tác trên các Git repository thuận tiện hơn. Sau đây là một số công cụ Git với công cụ đồ họa:

- Windows: GitHub Desktop, GitKraken, Sourcetree, Tortoise Git, SmartGit, GitForce, Git Cola, Aurees, Magit, Fork,...
- MacOS: GitHub Desktop, GitKraken, Sourcetree, SmartGit, GitForce, Sublime Merge, Fork, GitUp,...

Ngoài ra, một số IDE thông dụng như Eclipse, Visual Studio Code, Netbean,... cũng có hỗ trợ các tính năng liên quan đến Git.

4 Bài tập

4.1 Bài tập 1

Thực hiện theo những bước sau:

1. Khởi tạo 1 repository mới.
2. Tạo 1 dự án web mới và tạo 1 tập tin `index.html` cho dự án.
3. Commit lên repository.

Giả sử ứng dụng được đưa vào sử dụng và có 1 số lỗi cần được chỉnh sửa. Hãy thực hiện các bước sau để chỉnh sửa các lỗi:

4. Tạo 1 nhánh mới “*hotfix*” trong repository và chuyển sang nhánh mới.
5. Thay đổi nội dung tập tin `index.html`.
6. Commit thay đổi vừa thực hiện.
7. Kiểm tra lại nội dung của tập tin `index.html` trong hai nhánh *hotfix* và *master*.
8. Chuyển sang nhánh *master* và **merge** nhánh *hotfix* vào nhánh này.
9. Kiểm tra lại nội dung của tập tin `index.html` trong hai nhánh sau khi merge.

4.2 Bài tập 2

Hãy thực hiện các bước sau:

1. Tạo 1 repository mới.
2. Viết 1 chương trình `hello.c` hiển thị 1 chuỗi “Hello World” và biên dịch tập tin này.

```
gcc hello.c -o hello
```

3. Dịch và thực thi chương trình.
4. Thêm các tập tin của chương trình vào repository.

Giả sử chương trình đã viết không có xuống dòng sau khi hiển thị chuỗi “Hello World”, làm cho việc hiển thị không được rõ ràng. Do đó, ta cần phải sửa chương trình lại, thêm ký tự vào ký tự xuống dòng. Để thực hiện chỉnh sửa, ta sẽ tạo 1 nhánh riêng theo các bước sau:

5. Tạo 1 nhánh tên là *hotfix*.
6. Xóa tập tin `hello` (tập tin nhị phân).
7. Chỉnh sửa tập tin `hello.c` để thêm ký tự xuống dòng và dịch lại chương trình.
8. Stage và commit tất cả các thay đổi.
9. Kiểm tra lại trạng thái của nhánh *hotfix*.
10. Chuyển sang nhánh chính (*master*) và ghép nhánh *hotfix* vào nhánh này.

4.3 Bài tập 3

Tạo một tài khoản trên Github và thực hiện push các kết quả trong bài tập 1 và 2 lên Github.

4.4 Bài tập 4

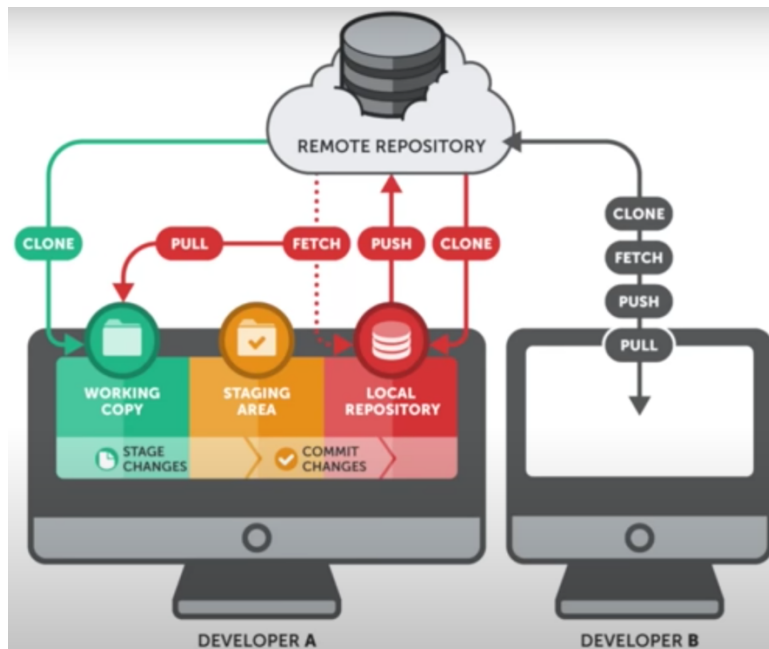
Thực hiện theo các yêu cầu sau:

1. Tạo 1 thư mục mới bên ngoài repository đã thực hiện trong các bài tập 1 và 2.
2. Clone repository trong bài tập 3 và kiểm tra lại kết quả.

3. Viết 1 chương trình `for.c` có 1 vòng lặp hiển thị các số từ 1 đến 10.
4. Dịch và thực thi chương trình này.
5. Commit các sự thay đổi vừa thực hiện.
6. Kiểm tra lại nội dung của repository trên Github và so sánh với repository trên local.
7. Push nội dung của local repository lên Github (remote repository).
8. Kiểm tra lại nội dung của repository trên Github.

5 Tóm tắt

- Version control là gì?
 - Hệ thống lưu trữ mọi thay đổi của source code.
 - Hỗ trợ nhiều người làm việc trên cùng 1 source code cùng lúc.
 - Ghi nhận lại mọi sự thay đổi: ai, đã thay đổi gì?
 - Có thể reverse source code về các phiên bản cũ.
- Git là gì?
 - Một source/version control software/model.
 - Ra đời năm 2005, tác giả là Linus Torvald (sử dụng cho Linux kernel).
 - Toàn bộ source code và history được lưu trữ trên máy người dùng.
 - Các khái niệm quan trọng:
 - Repository: kho chứa source code, thường là 1 project. Có 2 loại repo là local (trên máy tính người dùng) và remote (trên máy tính ở xa/cloud).
 - Commit: lưu lại những thay đổi lên repository (tạo thành 1 phiên bản mới).
 - Branch (nhánh): thường là source code của 1 chức năng, được phát triển riêng. Source code của 1 nhánh sau khi phát triển xong sẽ được merge vào nhánh chính của project.
- Github là gì?
 - Dịch vụ lưu trữ Git (remote) của Microsoft, gần như là lớn nhất hiện nay.
 - Là nơi lưu trữ của rất nhiều dự án phần mềm lớn.
 - Cung cấp thêm 1 số tính năng mở rộng như: xem code trực tiếp, phân quyền,...
- Các lệnh cơ bản: `init`, `clone`, `pull`, `add`, `commit`, `push`, `log`.



6 Tài liệu tham khảo

- [1] Scott Chacon, Ben Straub. Pro Git, Apress,
- [2] <https://thachpham.com/tools/cach-tao-repository-cho-git.html>
- [3] <https://vietnix.vn/git-la-gi/#working-directory-staging-area-va-local-repo>
- [4] <https://nordiccoder.com/blog/git-va-github-la-gi-cach-su-dung-git-cap-nhat/>
- [5] Phạm Huy Hoàng. Tự học Git siêu tốc,
<https://www.youtube.com/watch?v=1JuYQgpbrW0>
- [6] <https://learngitbranching.js.org>
- [7] <https://guides.github.com>