Git căn bản

Các bạn chỉ cần biết được Git căn bản là có thể tự quản lý code của bản thân mình rồi. Trong chương này, mình sẽ hướng dẫn mọi người Git căn bản và nâng cao.

Bắt buộc các bạn cần phải hiểu và thao tác được với Git căn bản để có thể push source code lên github phục vụ mục đích deploy cũng như review fix lỗi. Còn Git nâng cao thì sẽ áp dụng khi các bạn muốn tùy chỉnh nhiều hơn hoặc sau này đi làm, nên là nếu bạn muốn xin vào các vị trí Junior, Senior thì Git nâng cao là yêu cầu cần thiết phải biết.

1. Config name và email cho git với `git config`

- Git có 2 loại config đó là `local` và `global`. Local là dành riêng cho dự án, còn Global là cho cả máy tính, nếu không config local thì Git sẽ tự động dùng config của global.
- Để xem các config ở local

```
git config --local --list
```

- Để xem các config ở global

```
git config --global --list
```

- Đầu tiên thì chúng ta cần config cho git biết chúng ta là ai khi mà đưa code lên

```
git config --global user.name "Du Thanh Duoc"
git config --global user.email duthanhduoc@gmail.com
```

> **Lưu ý**: Với terminal sử dụng Git bash thì khi dùng tiếng Việt với Unikey nó sẽ dễ bị lỗi. Nên chuyển Unikey sang tiếng Anh nhé.

- Github nó sẽ dựa vào email mà bạn config ở local để định danh bạn khi bạn thay đổi code trên Github. Vậy nên khi bạn config một email không giống email mà bạn đăng ký trên Github thì khi gửi code lên Github bạn sẽ thấy một user khác chứ không phải bạn!

2. Khởi tạo Repository trên máy tính với `git init`

- Repository (Repo) là nơi chứa mã nguồn dự án của bạn. Để khởi tạo một Git Repo trên local thì dùng câu lệnh

```
git init
```

- Sau khi khởi tạo thì bạn sẽ thấy một thư mục ẩn tên là ".git".

> **Lưu ý**: Code của bạn phải nằm chung thư mục chứa folder ".git" thì code của bạn mới được quản lý bởi Git!

Có 2 loại repository:

1. **Local repository**: Repo trên máy tính
2. **Remote repository**: Repo trên server như Github

3. Hiển thị trạng thái với `git status`

Câu lệnh này sẽ hiển thị

- Bạn đang ở branch nào
- Trạng thái branch của bạn so với origin như thế nào (cái này nhiều lúc không chính xác vì dữ liệu đã được thay đổi trên origin, muốn chính xác thì phải `git fetch` để tải về dữ liệu mới nhất)
- Trạng thái các file trong dự án, file nào đang được git track (theo dõi)

```
git status
```

4 Các khu vực làm việc với Git

Chúng ta sẽ có các khu vực theo thứ tự dưới đây

1. **Khu vực làm việc**: Chính là nơi chúng ta đang code, vẫn ở local
2. **Khu vực staging**: Sau khi dùng `git add` thì file sẽ được đưa lên khu vực này, vẫn ở trên local
3. **Khu vực committed**: Sau khi dùng `git commit` thì file từ staging sẽ được đưa lên đây, cũng vẫn ở trên local
4. **Khu vực remote (gọi origin cũng được)**: Sau khi dùng `git push` sẽ file ở commited lên đây, bây giờ file của bạn đã đưa lên trên server

5. Thêm file vào khu vực Staging với `git add`

Câu lệnh dưới đây sẽ thêm một hoặc nhiều file (đã thay đổi) vào khu vực **Staging**.

Thêm 1 file

```
git add index.html
```

> **Lưu ý**: Các file của bạn phải thay đổi gì đó thì mới add vào được nha Thêm nhiều file

```
git add index.html app.js
```

Thêm tất cả các file.

```
git add .
```

> **Lưu ý** nếu bạn ở thư mục con thì nó chỉ thêm tất cả các file ở thư mục con thôi.

6. Khôi phục những file ở khu vực Staging về khu vực code với `git reset`

Ngược lại với `git add` thì `git reset` sẽ đưa một hoặc nhiều file ở khu vực **Staging** trở về khu vực code.

Khôi phục 1 file từ staging về khu vực code

```
git reset index.html
```

Khôi phục nhiều file

```
git reset index.html app.js
```

Khôi phục tất cả các file.

```
git reset .
```

7. Commit code với `git commit`

Câu lệnh dưới đây sẽ đính kèm các title khi bạn thực hiện những thay đổi trong dự án. Nó có tác dụng với những file trong khu vực **Staging**.

```
git commit -m "Tôi thay đổi dòng này vì nó xảy ra tình trạng vòng lặp vô tận"
```

> **Lưu ý** có một số terminal bắt buộc bạn phải dùng dấu nháy kép "" chứ không được dùng dấu nháy đơn ''. Còn một số terminal thì dùng cả 2 đều được.

Trong trường hợp bạn muốn đính kèm thêm description để mô tả thêm cho title thì bạn có thể dùng 2 cách

Cách 1:

```
git commit -m "Title" -m "Description"
```

Cách 2: Bạn gõ thiếu dấu `"` và nhấn Enter nó sẽ cho bạn xuống dòng, tiếp theo bạn sẽ gõ tiếp mô tả và kết thúc bằng dấu `"` là được.

```
git commit -m "Title
> Description"
```

8. Tạo và clone remote repo với HTTPS và SSH

8.1. Clone với Https

1. Mở remote repo trên github lên, click vào "Code" chọn "HTTPS" và copy đường link
2. Mở terminal lên paste đường link đó kèm với câu lệnh `git clone`, ví dụ:

```
git clone https://github.com/dreact04072022/git-can-ban.git
```

Trong trường hợp repo đó ở chế độ public thì không sao, nếu repo đó ở chế độ private hoặc ở public nhưng bạn muốn có quyền push code lên remote repo đó thì bạn phải nhập username và mật khẩu vào khi clone. Ví dụ:

```
git clone https://username:password@github.com/dreact04072022/git-can-
ban.git
```

8.2. Clone với SSH

Clone với SSH thì bạn cần làm 2 thứ

1. Tạo SSH key và add vào github
2. Copy đường link SSH trên Github repo rồi chạy câu lệnh `git clone` là được

```
git clone git@github.com:dreact04072022/git-can-ban.git
```

Clone với SSH tiện lợi hơn giao thức HTTPS vì bạn không cần nhập username và mật khẩu mỗi khi clone, dễ làm lộ thông tin. Bạn cũng chỉ cần tạo SSH key và add vào 1 lần duy nhất, những lần sau chỉ cần chạy `git clone` là được rồi.

8.3. Tạo SSH key

Để tạo SSH key thì có thể tham khảo: https://docs.github.com/en/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent

1. Đầu tiên mở terminal Gish Bash lên
2. Paste text bên dưới, thay thế email là địa chỉ email Github của bạn

```
ssh-keygen -t ed25519 -C "your_email@example.com"
```

Điều này sẽ tạo một SSH key mới, sử dụng email đã được cung cấp như một nhãn trong SSH key (bạn có thể mở public key của SSH key lên sẽ thấy email trong đó)

Nó sẽ yêu cầu bạn nhập tên file để lưu, nếu bạn enter thì nó sẽ lấy tên file mặc định như trong dấu (). Lưu ý là khi nhập tên file phải nhập đầy đủ đường dẫn lưu file tương tự như trong dấu () nhé.

```
Enter file in which to save the key (/c/Users/dutha/.ssh/id_ed25519):
```

Tiếp theo nó sẽ yêu cầu bạn nhập passphrase (tương tự password thôi). Cá nhân mình thì không nhập, cứ Enter thôi vì khi nhập sau này mỗi khi làm việc với Git phải nhập passphrase khá mệt

```
Enter passphrase (empty for no passphrase):
```

```
Enter same passphrase again:
```

Sau khi tạo thành công thì nó sẽ sinh ra cho bạn 2 file là private key và public key theo đường dẫn mà bạn nhập tên file. File chứa public key sẽ có đuôi `.pub` phía sau.

```
Your identification has been saved in /c/Users/dutha/.ssh/id_duthanhduoc10
Your public key has been saved in /c/Users/dutha/.ssh/id_duthanhduoc10.pub
```

Để đọc nội dung public SSH key thì bạn chỉ có khá nhiều cách, bạn dùng cách nào dưới đây cũng được. Ví dụ file public key của mình bên trên là `id_duthanhduoc10.pub`

- Copy đường dẫn này `c:/Users/dutha/.ssh/id_duthanhduoc10.pub` bỏ lên Chrome thì nó sẽ ra nội dung của public key
- Dùng git bash gõ `cat /c/Users/dutha/.ssh/id_duthanhduoc10.pub`

8.4. Tiến hành thêm SSH public key vào Github

Tham khảo đầy đủ tại https://docs.github.com/en/authentication/connecting-to-github-with-ssh/adding-a-new-ssh-key-to-your-github-account

8.5. Kiểm ta SSH key đã kết nối github thành công hay chưa

Tham khảo đầy đủ cho các hệ điều hành tại https://docs.github.com/en/authentication/connecting-to-github-with-ssh/testing-your-ssh-connection

1. Mở Git bash
2. Nhập dòng này vào và enter `ssh -T git@github.com` Có thể bạn sẽ thấy cảnh báo này

```
> The authenticity of host 'github.com (IP ADDRESS)' can't be established.
> RSA key fingerprint is SHA256:nThbg6kXUpJWGl7E1IGOCspRomTxdCARLviKw6E5SY8.
> Are you sure you want to continue connecting (yes/no)?
```

3. Xác nhận fingerprint có giống lúc tạo shh key hay không. Nếu đúng thì gõ `yes`:

```
Hi username You've successfully authenticated, but GitHub does not provide
shell access.
```

4. Kiểm tra lại coi thông báo có đúng username tài khoản github của bạn hay không. Nếu đúng thì đã thành công, nếu sai thì kiểm tra lại public key bạn add vào github có thể bị sai.

8.6. Trường hợp bạn tạo SSH key với passphrases

Với SSH key, nếu ai đó truy cập được vào máy tính của bạn thì có thể truy cập được vào github và thay đổi mã nguồn của bạn. Bạn cũng có thể thêm passphrase vào SSH key của bạn để tăng cường tính bảo mật. Khi bạn thêm như vậy thì mỗi lần bạn push code thì nó Git đều yêu cầu bạn nhập passphrase, bạn cũng có thể dùng `ssh-agent` để chỉ nhập passphrase 1 lần trong phiên làm việc đó thôi.

Thông tin đầy đủ về làm việc với SSH key passphrase tại đây

Thêm hoặc thay đổi passphrase

Nếu bạn tạo SSH key với passphrase thì không cần quan tâm cái này, còn nếu lúc tạo bạn không có passphrase, giờ muốn thêm hoặc thay đổi thì chỉ cần gõ theo lệnh dưới

Lưu ý thay thế tên `id_ed25519` cho đúng tên file private key của bạn nhé

```
$ ssh-keygen -p -f ~/.ssh/id_ed25519
> Enter old passphrase: [Type old passphrase]
> Key has comment 'your_email@example.com'
> Enter new passphrase (empty for no passphrase): [Type new passphrase]
> Enter same passphrase again: [Repeat the new passphrase]
> Your identification has been saved with the new passphrase.
```

Nếu key của bạn có passphrase, bạn sẽ bị yêu cầu enter passphrase cũ trước khi có thể thay đổi

Thêm SSH key vào ssh-agent

Để khởi động ssh-agent thì có 2 cách đó là mỗi lần mở Git bash thì bạn phải khởi động nó hoặc bạn setting cho nó tự khởi động mỗi khi bạn mở Git bash

Khởi động ssh-agent

1. Mở Git bash lên và chạy câu lệnh dưới

```
$ eval "$(ssh-agent -s)"
> Agent pid 59566
```

2. Tiến hành thêm SSH private key vào ssh-agent. Thay thế tên `id_ed25519` cho đúng với tên private key của bạn nhé.

```
ssh-add ~/.ssh/id_ed25519
```

3. Tiến hành thêm SSH key vào tài khoản Github của bạn. Nếu bạn đã thêm rồi thì thôi.

> **Tip**: Câu lệnh `ssh-add` sẽ tương đương `ssh-add ~/.ssh/id_rsa`. Đôi lúc muốn check ssh-agent có đang chạy hay chưa thì chỉ cần chạy câu lệnh `ssh-add ~/.ssh/id_ed25519` (nhớ thay đường dẫn cho đúng với private key của bạn nhé) nếu kết quả nó là `Could not open a connection to your authentication agent.` thì ssh-agent chưa chạy, còn nếu nó yêu cầu nhập passphrase hay "Identity added..." thì nghĩa là đã chạy rồi.

9. Dùng nhiều tài khoản Github trên cùng một máy tính

Tạo 1 file tên là `config` trong thư mục `~/.ssh/`

Viết nội dung file này tương tự như nội dung bên dưới

```
#Default GitHub
Host github.com
HostName github.com
User git
IdentityFile ~/.ssh/id_rsa

#React-Awesome duthanhduoc06
Host github-duthanhduoc06.com
HostName github.com
User git
IdentityFile ~/.ssh/id_rsa_duthanhduoc06
```

Ở trên thì mình có dùng 2 tài khoản Github, cái đầu tiên dùng với private key là `id_rsa` với Host là `github.com`, cái thứ hai là `id_rsa_duthanhduoc06` với Host là `github-duthanhduoc06.com`.

Khi mình clone những repo thuộc quyền của tài khoản github đầu tiên thì mình vẫn giữ nguyên link github.com nhưng nếu mình clone những repo thuộc quyền sở hữu của tài khoản thứ hai thì mình phải thay đổi thành như thế này

```
git clone git@github-duthanhduoc06.com:dreact04072022/git-can-ban.git
```

Tương tự với những câu lệnh như `git remote add origin git@github-duthanhduoc06.com:dreact04072022/git-can-ban.git`

10. Đẩy code lên git server với `git push`

Muốn đẩy code của bạn lên server thì local repo phải được liên kết với remote repo. Có 2 cách để bạn liên kết:

1. Bạn chưa có local repo, bạn tiến hành clone một remote repo về máy tính thì lúc này bạn sẽ có được local repo
2. Bạn có local repo nhưng chưa có remote repo, lúc này bạn cần tiến hành tạo một remote repo mới hoàn toàn và tiến hành liên kết nó với local repo của bạn.

Nếu bạn đang ở nhánh master, câu lệnh dưới đây sẽ đẩy code của bạn lên nhánh master ở remote repo (hay còn gọi là origin master)

```
git push origin master
```

> Mặc định đối với nhánh master thì bạn chỉ cần `git push` là được rồi vì git nó tự hiểu là bạn đang push lên orgin master.

11. Hiển thị log commit với git log

Hiển thị những thông tin commit gần đây. Các bạn nhân phím "q" để quit (thoát)

```
git log
```

Một số phím chức năng bạn có thể nhập đề điều hướng và tìm kiếm trong log như:

- return - dòng tiếp theo
- w - trang tiếp
- spacebar - trang trước
- q - thoát
- ?pattern - tìm kiếm, với pattern là mẫu tìm kiếm (keyword)
- /pattern - giống ?pattern
- n - đến vị trí tìm kiếm phía dưới
- N - đến kết quả tìm kiếm phía trước

Nếu muốn nhìn thấy thông tin rút gọn thì

```
git log --oneline
```

12 Kéo code từ remote repo về với `git pull`

Trong những trường hợp code trên remote repo có những thay đổi và cập nhật, bạn có thể cập nhật những thay đổi này trên local repo của bạn cho giống như trên remote repo.

Câu lệnh dưới đây sẽ pull code từ nhánh master về nhánh hiện tại của bạn ở local

```
git pull orgin master
```

> Mặc định đối với nhánh master thì bạn chỉ cần `git pull` là được rồi vì git nó tự hiểu là bạn đang pull từ orgin master.

13 Bỏ qua file với .gitignore

Có những file chúng ta không muốn bị git giám sát thì chỉ cần tạo file `.gitignore` và thêm các file và folder vào file `.gitignore` này.

`.gitignore`

```
# Comment
node_modules/
.logs
```

Những trường hợp phổ biến dùng `.gitignore`

- ignore thư mục node_modules vì thư mục này rất nặng gây tốn thời gian pull và push code. Ngoài ra còn tốn tài nguyên git, ai muốn có node_modules thì chỉ cần chạy npm hoặc yarn là có ngay.
- ignore các thư mục build như `dist`. Ai muốn có các file build thì chỉ cần chạy câu lệnh build là được, không cần gửi lên git thư mục này làm gì
- ignore các file cấu hình trên máy tính cá nhân mà không muốn xuất hiện ở các máy thành viên khác

Cách viết `.gitignore`

- Comment thì dùng `#`
- Ignore file: `example.exe`
- Ignore cả thư mục: `folder/`, tất nhiên là bạn dùng `folder` cũng được nhưng nên thêm `/` để phân biệt với file
- Phủ định thì thêm `!`: `!folder/file.exe`
- Ignore tất cả các file có đuôi là `.exe`: `*.exe`
- Ignore tất cả các file có tên bắt đầu là log: `log*`
- Ignore tất cả các file có đuôi là `.exe` ở theo đường dẫn `folder/file.exe` (các file ở đường dẫn `folder/sub/file.exe` sẽ không bị ignore): `folder/**.exe`
- Ignore tất cả các file có đuôi là `.exe` ở thư mục `folder` dù cho có nằm ở sub-folder đi chăng nữa: `folder/**/**.exe`
- Ignore mọi thứ bên trong thư mục folder: `folder/**`

Xử lý Git cache

Trong một số trường hợp bạn code một thời gian rồi, push pull các kiểu rồi, sau đó bạn mới thêm các file vào `.gitignore`, lúc này những file đó có thể không bị ignore vì nó đã bị git cache từ trước và git nó vẫn quản lý những file này. Cách giải quyết là hãy xóa những file đó ra khỏi cache

```
git rm -r --cached /đường-dẫn-file-hoặc-folder
```

14. Tạo file README.md

ReadMe là file giới thiệu về dự án, cũng là file mô tả cấu trúc hoặc doc của dự án.

Để tạo và sử dụng ReadMe file thì bạn chỉ cần

1. Tạo file có tên là README.md trong thư mục root của repo
2. Thêm nội dung cho file với cú pháp Markdown.
3. Khi hoàn thành thì có thể push lên remote repo và xem thành quả

Markdown là gì?

Markdown là ngôn ngữ đánh dấu với cú pháp khá tương đồng với HTML. Nó có thể dễ dàng chuyển thành HTML và nhiều định dạng khác. Markdown rất thân thiện với developer và viết markdown cũng rất dễ.

Tham khảo cách viết tại: https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax

15. Giải quyết conflict code

Code bị conflict khi code trên local branch khác xung đột với remote branch, vấn đề này xảy ra khi có ai đó đã cập nhật 1 đoạn code trên remote branch và bạn cũng cập nhật đoạn đó trên local branch.

Lúc này git sẽ không cho phép bạn push code lên, bạn phải pull code trên remote branch về và lựa chọn xem nên chọn những đoạn code nào.

Thứ tự thường dùng:

1. Pull code mới nhất trên remote branch về bằng `git pull`
2. Giải quyết conflict bằng tay, thảo luận với những người liên quan đoạn code xung đột nên chọn cái nào.
3. Add file xung đột vào khu vực staging `git add TenFile`. Hoặc có thể `git add .` cũng được
4. Commit cho những file xung đột đó và push code lên

16. Thao tác với Git UI cơ bản trên VS code

- Xem nhánh hiện tại
- Các khu vực Changes / Staged Changes
- Khôi phục Staged Changes về Changes
- Thêm Changes vào Staged Changes
- Add, Commit, Push
- Cài Gitlens extension
- Xem lịch sử các Commit
- Search commit
- Xem author, người đã edit