Git và GitHub — liệu bạn đã thật sự biết cách sử dụng?

Biến Git và GitHub trở thành công cụ đắc lực cùng Software Engineer nhà Got It!

[](https://medium.com/@ducminh.phan?source=post_page-----ff5cc723edfd----------------------)

[Duc-Minh Phan](https://medium.com/@ducminh.phan?source=post_page-----ff5cc723edfd----------------------)

Follow

[Mar 26](https://medium.com/got-it-vietnam/git-v%C3%A0-github-li%E1%BB%87u-b%E1%BA%A1n-%C4%91%C3%A3-th%E1%BA%ADt-s%E1%BB%B1-bi%E1%BA%BFt-c%C3%A1ch-s%E1%BB%AD-d%E1%BB%A5ng-ff5cc723edfd?source=post_page-----ff5cc723edfd----------------------) · 16 min read

Như đã hứa ở bài viết [***LinkedIn — vũ khí bí mật khi tìm việc online***](https://medium.com/got-it-vietnam/linkedin-v%C5%A9-kh%C3%AD-b%C3%AD-m%E1%BA%ADt-khi-t%C3%ACm-vi%E1%BB%87c-online-aec9663e52d6), ở bài viết tiếp theo này, hãy cùng Got It bàn về **Git và GitHub** — hai công cụ mà dân dev ai cũng nghe tên, thậm chí còn sử dụng hàng ngày. Thế nhưng, bạn có biết thế nào là một GitHub đủ “đẹp” để đưa vào CV? Hay làm thế nào để tối đa hoá hiệu suất công việc trên hai công cụ này?

Hãy đọc bài viết nóng hổi được chúng mình đặt hàng từ một kỹ sư lập trình nhà Got It— người chuyên review GitHub cho các bạn ứng viên nhé!

https://miro.medium.com/max/60/1*6X9Sw2qcseL_Ty9-TTnf0Q.jpeg?q=20



**1. Git**

Khi viết code, sẽ có rất nhiều lúc bạn muốn, hoặc cần phải quay lại đoạn code mình đã viết tại một thời điểm trước đây. Một cách đơn giản, ta hoàn toàn có thể dùng tính năng undo/redo của trình biên soạn (editor) mình đang dùng. Tuy nhiên, editor có thể sẽ không lưu lại lịch sử sau khi đóng, và bạn cũng có thể nhấn nhầm sang một phím khác thay vì nhấn redo, và thế là lịch sử trạng thái được editor lưu lại đã bị ghi đè khiến bạn không thể redo được nữa.

Ngoài ra, khi nhiều người cùng làm việc trong một dự án, các thành viên cần tìm ra cách chia sẻ các thay đổi của mình để đảm bảo rằng tất cả mọi người đang làm việc trên cùng một phiên bản của dự án. Trước đây, lập trình viên thường chia sẻ các thay đổi code bằng cách xuất ra file patch và gửi cho nhau qua email. Khi nhận được một bản patch, các thành viên sẽ áp dụng nó vào trong phiên bản code hiện tại của mình trên máy cá nhân.

Dưới đây là một ví dụ về file patch, được trích ra từ <https://bit.ly/34r4SVU>.

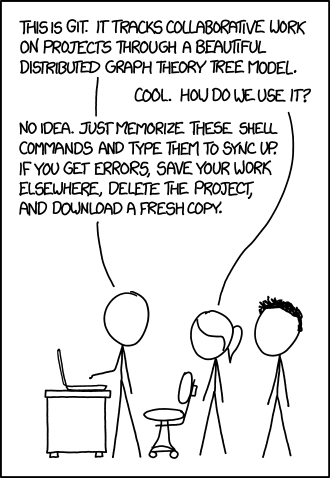
From 06f2790b5ca3fea45515e33c9660ad6265120a5a Mon Sep 17 00:00:00 2001  
From: =?UTF-8?q?=C5=81ukasz=20Langa?= <[lukasz@langa.pl](mailto:lukasz@langa.pl)>  
Date: Wed, 4 Mar 2020 23:16:55 +0100  
Subject: [PATCH] Rename FileMode into just ModeThe mode was never just about files to begin with. There are no other modes in  
Black, this can be the default one.  
---  
 black.py | 40 +++++++++++++++++++---------------------  
 1 file changed, 19 insertions(+), 21 deletions(-)diff --git a/black.py b/black.py  
index 52096819..31859d1a 100644  
--- a/black.py  
+++ b/black.py  
@@ -186,7 +186,7 @@ class Feature(Enum):  
   
   
 [@dataclass](http://twitter.com/dataclass)  
-class FileMode:  
+class Mode:  
 target\_versions: Set[TargetVersion] = field(default\_factory=set)  
 line\_length: int = DEFAULT\_LINE\_LENGTH  
 string\_normalization: bool = True  
@@ -209,6 +209,10 @@ def get\_cache\_key(self) -> str:  
 return ".".join(parts)  
   
   
+# Legacy name, left for integrations.  
+FileMode = Mode  
+  
+  
 def supports\_feature(target\_versions: Set[TargetVersion], feature: Feature) -> bool:  
 return all(feature in VERSION\_TO\_FEATURES[version] for version in target\_versions)

Mỗi file patch chỉ biết đến hai trạng thái của dự án, là trước và sau khi được trích xuất ra. Như vậy, khi có nhiều file patch, ta cần phải biết thứ tự của chúng để có thể xây dựng lại chính xác một phiên bản của dự án tại bất kì thời điểm nào. Ngoài ra, vì file patch cũng chỉ là file thông thường được lưu trong ổ cứng, nên ta cần một cách để sao lưu lại chúng. Khi có nhiều người cùng làm việc trên một file, ta cần thêm một cơ chế đồng bộ các file patch với nhau, vì rất có thể sẽ có những thay đổi chồng chéo lên nhau.

Để giải quyết các vấn đề nêu trên, ta cần đến một hệ thống quản lý phiên bản. Phần một của bài viết này sẽ giới thiệu đến các bạn Git, một trong những hệ thống quản lý phiên bản phổ biến nhất hiện nay.

**1.1.**Git**là gì?**

https://miro.medium.com/max/42/1*0o9GZUzXiNnI4poEvxvy8g.png?q=20



<https://xkcd.com/1597/>

Git (<https://git-scm.com/>) là một hệ thống quản lý phiên bản phân tán (distributed version control system), được viết bởi Linus Torvalds (người tạo ra và phát triển chính của Linux kernel) vào năm 2005. Mục đích ban đầu của Git là để hỗ trợ việc phát triển Linux kernel. Ngoài Git còn có các hệ thống khác như Subversion và Mercurial. Các bạn có thể tham khảo bài viết sau về mức độ phổ biến của các hệ thống quản lý phiên bản vào năm 2016: <https://rhodecode.com/insights/version-control-systems-2016>; hoặc dữ liệu từ Google Trends: <https://trends.google.com/trends/explore?date=all&q=git,svn,mercurial>.

Để hiểu rõ hơn về Git, ta sẽ tìm hiểu về từng khía cạnh trong tên gọi hệ thống quản lý phiên bản phân tán.

**Hệ thống quản lý phiên bản (version control system, VCS)**

https://miro.medium.com/max/60/1*kYZttNrJldnOlH3zAXSCUw.png?q=20



Nguồn: [reddit](https://www.reddit.com/r/ProgrammerHumor/comments/72rki5/the_real_version_control/)

Trong cuốn sách kinh điển The Pragmatic Programmer [1], hai tác giả David Thomas và Andrew Hunt đã dành hẳn một phần (Topic 19) để nói về việc quản lý phiên bản. Còn trong một quyển sách khác nổi tiếng không kém [2], tác giả Robert C. Martin cũng đề cập đến việc sử dụng một hệ thống quản lý phiên bản ở ngay đầu phần phụ lục Tooling, và ông khuyên dùng Git để quản lý code.

Git hỗ trợ các lập trình viên bằng cách lưu lại toàn bộ các thay đổi với dự án, được chia nhỏ thành các commit. Mỗi commit là một đơn vị thay đổi, tương tự như một file patch. Tại mỗi commit, ta có một phiên bản của dự án. Bằng việc lưu lại toàn bộ thay đổi trong một hệ thống riêng biệt, ta hoàn toàn có thể quay lại bất cứ phiên bản nào tại thời điểm mà commit được tạo ra. Ngoài ra, Git cũng hỗ trợ làm việc nhóm thông qua tính phân tán.

**Tính phân tán (distributed)**

Khi làm việc với một dự án, ta sẽ tạo ra một Git repository (kho chứa), thường được gọi tắt là repo. Đây là nơi chứa toàn bộ các thông tin cần thiết để quản lý dự án, ví dụ như các thay đổi đã được nhắc đến ở trên.

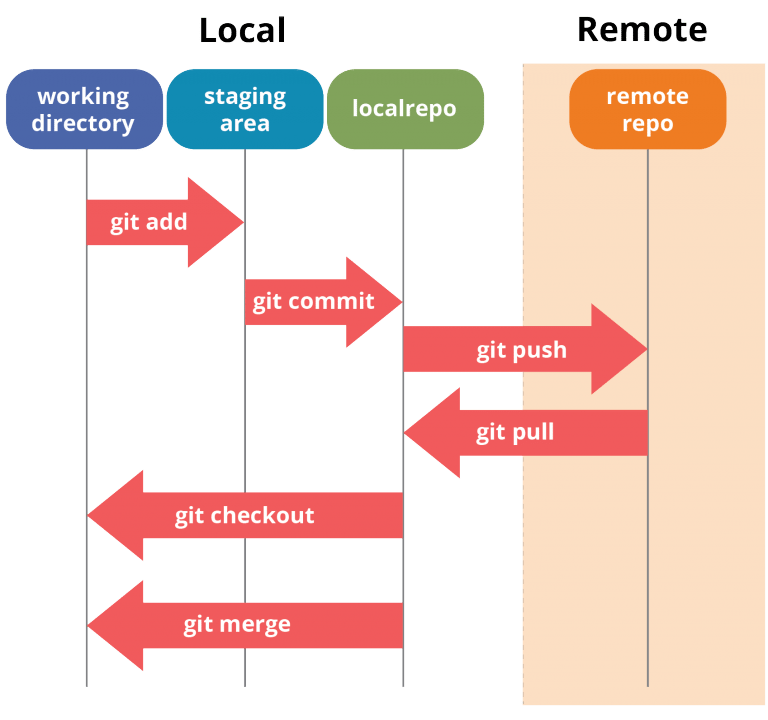
Khi làm việc với một dự án được quản lý bởi Git, mỗi lập trình viên sẽ có một bản sao của Git repo trên máy cá nhân, và được đồng bộ với nhau thông qua một repo được lưu trữ trên máy chủ riêng biệt (remote repository). Thay vì chia sẻ các bản patch, lập trình viên sẽ tạo ra các commit, và gửi các commit này lên remote repository. Các lập trình viên khác có thể đồng bộ repo trên máy cá nhân với remote repo để thấy được các commit này, và thêm vào phiên bản của dự án hiện tại trên máy của mình nếu muốn.

**1.2. Sử dụng**Git**như thế nào?**

Khi bắt đầu sử dụng Git cho một dự án, ta cần khởi tạo một Git repository trên máy cá nhân, bắt đầu với câu lệnh git init. Sau khi được khởi tạo, mọi thay đổi sẽ được Git theo dõi và quản lý. Trong trường hợp bắt đầu làm việc với một dự án đã và đang sử dụng Git, và đã có một remote repository, ta có thể tạo ra một bản sao trên máy cá nhân bằng câu lệnh git clone.

Sau khi thiết lập môi trường làm việc, ta có thể bắt đầu thao tác với Git như sơ đồ bên dưới. Local repository chính là bản sao của Git repo trên máy cá nhân, được đồng bộ với những local repo khác thông qua remote repo. Cũng trên máy cá nhân, ta còn có hai môi trường nữa là thư mục làm việc (working directory) và khu vực chờ (staging area).

https://miro.medium.com/max/60/1*DFl91eNu6QYrWj73n0QX_w.png?q=20



Nguồn: <https://www.edureka.co/blog/git-tutorial/>

Tất cả các thao tác thêm/sửa/xoá để thay đổi code sẽ được thực hiện trên thư mục làm việc. Khi muốn lưu lại những thay đổi này, ta sẽ dùng câu lệnh git add để thêm chúng từ thư mục làm việc sang khu vực chờ. Sau đó sử dụng câu lệnh git commit để lưu từ khu vực chờ vào local repository.

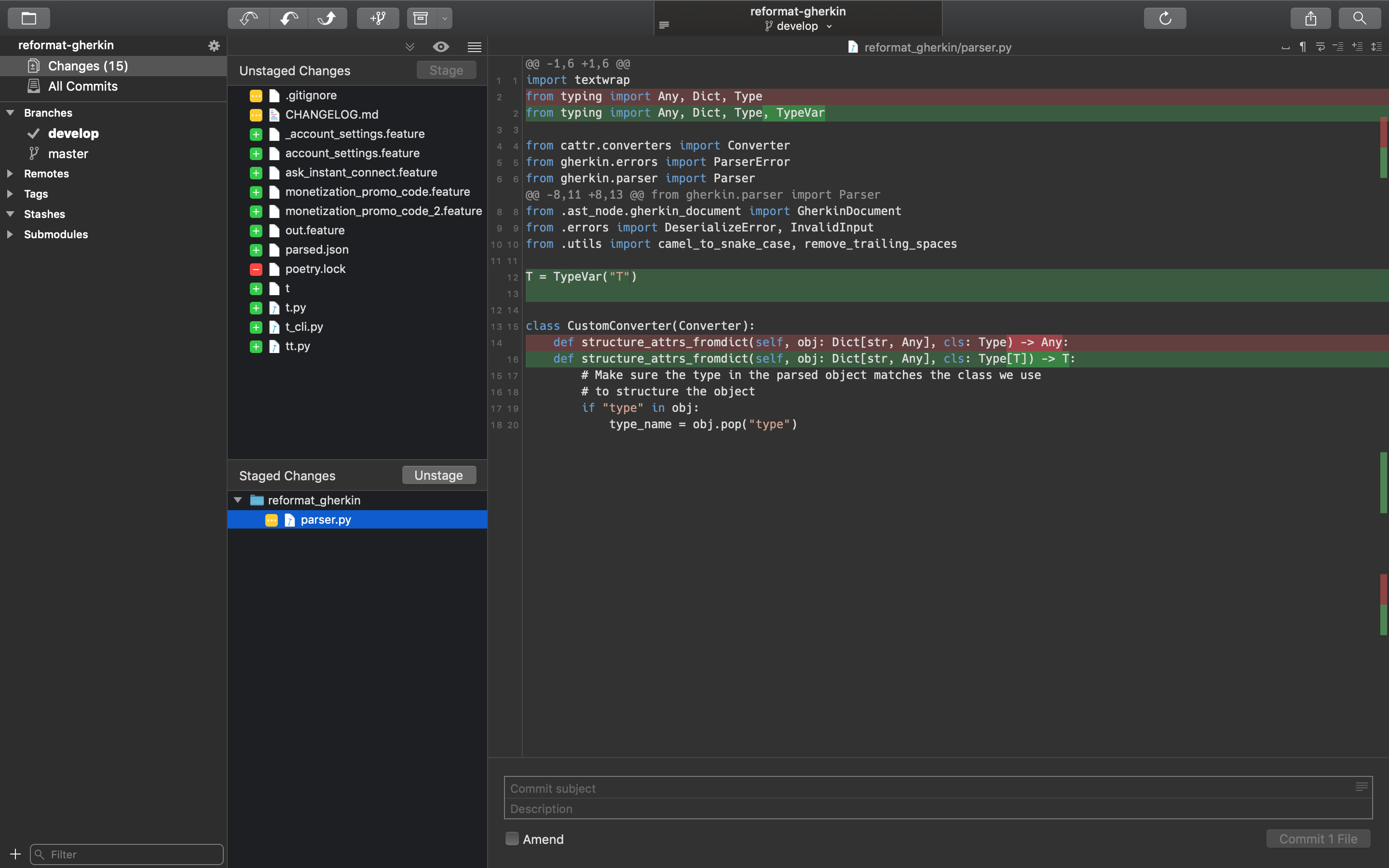
Khu vực chờ là nơi để chuẩn bị cho những thay đổi trước khi chúng được commit. Ví dụ như ta thay đổi ba file: a.py, b.py, và c.py. Tuy nhiên thay đổi ở file c.py tương đối độc lập với hai file còn lại nên ta muốn tách ra một commit riêng. Để làm được việc đó, đầu tiên ta thêm file a và b vào khu vực chờ và commit trước:

> git add a.py  
> git add b.py  
> git commit -m "Thay doi file a va b"

Sau đó mới thêm file c vào khu vực chờ và commit:

> git add c.py  
> git commit -m "Thay doi file c"

https://miro.medium.com/max/60/1*CLdq1aGOJXcH_xEO8G3wzg.png?q=20

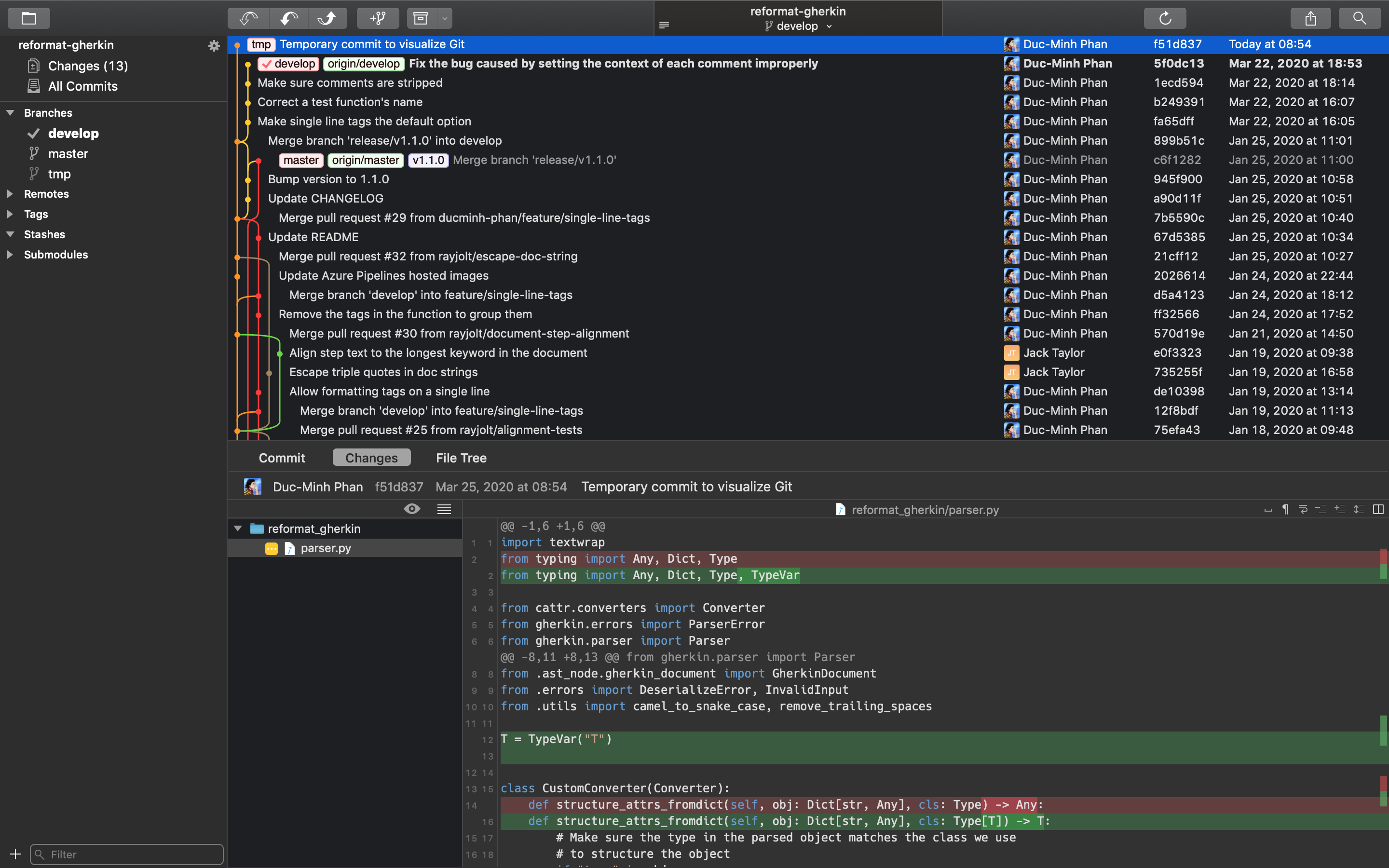


Thao tác với Git sử dụng [Fork](https://git-fork.com/): duyệt lại thay đổi ở khu vực chờ trước khi commit

Tuy nhiên lúc này, những thay đổi mới chỉ đang ở local repository, tức là những lập trình viên khác không biết đến sự tồn tại của chúng. Để đưa những thay đổi này lên remote repository, ta dùng lệnh git push. Và ngược lại, dùng lệnh git pull để cập nhật những thay đổi mới nhất của người khác từ remote repository về máy cá nhân.

Một tính năng rất quan trọng khi làm việc nhóm với Git là tách nhánh (branching). Tất cả Git repository đều có một nhánh mặc định là master, nơi các commit sẽ được lưu lại ở thời điểm ban đầu. Khả năng tách nhánh của Git giúp cho các thành viên của dự án có thể làm việc độc lập vì nếu mỗi người có một nhánh riêng, các thay đổi khi được push lên remote repository sẽ không gây ra xung đột. Ta có thể kiểm tra xem nhánh hiện tại đang làm việc là gì với câu lệnh git branch, và tạo một nhánh mới với câu lệnh git checkout.

https://miro.medium.com/max/60/1*4S0Be5R5BX_A3hoJz_Jm0g.png?q=20



Thao tác với Git sử dụng [Fork](https://git-fork.com/): lịch sử commit và các nhánh

Sau khi đã tạo ra một nhánh mới, ta có thể bắt đầu commit những thay đổi lên nhánh này. Khi hoàn thành phần việc của mình trên nhánh cá nhân, ta cần tích hợp các thành phần lại với nhau thông qua việc hợp nhất các nhánh với câu lệnh git merge. Các thay đổi sau khi đã được hợp nhất lại sẽ được các thành viên pull về local repository của mình. Trong hình minh hoạ bên trên, ta có thể thấy các merge commit như Merge branch 'release/v1.1.0' into develop, hoặc Merge branch 'release/v1.1.0'.

**1.3. Các nguyên tắc khi làm việc với**Git

Có một số nguyên tắc nhất định giúp cải thiện hiệu suất và nâng cao khả năng làm việc nhóm khi sử dụng Git để quản lý phiên bản cho dự án.

* **Tách nhánh mới để làm việc.** Như đã nói ở trên, việc tách nhánh mới giúp các thành viên có thể làm việc độc lập với nhau. Ngoài ra, bằng cách làm việc trên một nhánh riêng biệt, ta có thể đảm bảo rằng code trên nhánh mặc định (master) luôn là code ổn định và có thể được đưa lên môi trường production.
* **Viết commit message rõ ràng, dễ hiểu.** Mỗi commit đều có một message, là nơi để lập trình viên tóm tắt những thay đổi được lưu lại trong commit này. Commit message rõ ràng và tường minh giúp cho các thành viên khác có thể dễ hình dung về những thay đổi hơn khi nhìn vào git log hoặc git tree. [Conventional Commits](https://www.conventionalcommits.org/en/v1.0.0/) là một quy ước viết commit message thường được dùng.

https://miro.medium.com/max/60/1*bLtPTIsKUeAQHPo2eGrKpw.png?q=20



<https://xkcd.com/1296/>

* **Mỗi commit chỉ nên có một trách nhiệm duy nhất.** Nguyên tắc Single Responsibility (S trong [SOLID](https://en.wikipedia.org/wiki/SOLID)) là một nguyên tắc cơ bản trong thiết kế phần mềm. Tạo Git commit là một trong nhiều tính huống khác ta có thể áp dụng nguyên tắc này. Mỗi commit chỉ nên lưu lại những thay đổi nhỏ, có liên quan trực tiếp đến nhau. Thay vì tạo ra một commit với rất nhiều thay đổi (hàng trăm dòng code và hàng chục file được thay đổi), ta nên chia ra thành nhiều commit nhỏ. Điều này giúp dễ quan sát các thay đổi, và việc quay lại một trạng thái trước đây cũng dễ dàng hơn vì lịch sử đã được chia nhỏ.
* **Tránh commit những thay đổi không cần thiết, hoặc không nên được lưu lại.** Giả sử rằng trong dự án của bạn cần lưu lại những cấu hình cho việc kết nối vào cơ sở dữ liệu, hoặc là khoá bí mật dùng trong việc mã hoá thông tin,… Những thông tin này tuyệt đối không được lưu lại với Git, vì một khi bạn đưa những thông tin này lên một remote repository công khai, bất cứ ai cũng có thể xem được và truy cập vào cơ sở dữ liệu của bạn,... Ngoài ra, còn những thông tin khác như các thành phần phụ thuộc của dự án cũng không nên được lưu lại, ví dụ như npm\_modules khi làm việc với JavaScript, hoặc venv khi làm việc với Python.

**1.4. Các tính năng khác của**Git

* hooks. Hooks là một tính năng rất mạnh mẽ của Git. Với tính năng này, ta có thể thêm các hành động tuỳ chỉnh vào từng giai đoạn làm việc với Git, ví dụ như trước và sau khi commit/push,... Thường dùng nhất là pre-commit hook, được chạy trước mỗi commit. Thậm chí còn có một [framework](https://pre-commit.com/) chuyên để quản lý chúng. Pre-commit hooks thường được sử dụng để format code theo một chuẩn chung ([black](https://github.com/psf/black)/[prettier](https://prettier.io/)), chạy các chương trình phân tích tĩnh (static analysis) như [mypy](http://mypy-lang.org/), [eslint](https://eslint.org/),... để giảm thiểu lỗi.
* annotate/blame. Khi sử dụng câu lệnh này, với mỗi dòng trong một file, ta có thể biết được người gần nhất sửa dòng này là ai, và sửa vào lúc nào. Các kỹ sư ở Got It rất hay dùng để blame nhau, đúng như tên gọi của câu lệnh, ví dụ như: Chỗ này chú J, chú K viết khó đọc thế, chỗ này ai viết mà tinh tế thế, hoá ra là anh M.

https://miro.medium.com/max/60/1*LnFcc318VX8gvSmnAK_u6g.jpeg?q=20



Nguồn: [reddit](https://www.reddit.com/r/ProgrammerHumor/comments/6p9syt/the_problem_with_git_blame/)

* cherry pick. Khi dùng câu lệnh git merge để hợp nhất hai nhánh lại làm một, ta sẽ mang tất cả commit của nhánh tính năng vào nhánh chính. Tuy nhiên, không phải lúc nào ta cũng muốn làm vậy, chẳng hạn như khi cần deploy và test trước một số thay đổi chọn lọc từ nhánh tính năng. Đây là lúc ta sử dụng đến câu lệnh cherry-pick để copy thay đổi từ những commit cần thiết sang một nhánh khác. Đơn vị nhỏ nhất có thể cherry pick là một commit, nên nếu commit quá lớn thì ta sẽ không thể chọn ra những thay đổi cần thiết.
* Ngoài các tính năng kể trên, bạn đọc có thể tham khảo thêm các tính năng khác như submodule, stash, rebase, status, diff,…

**1.5. Tìm hiểu thêm về**Git

Trên đây chỉ là những hướng dẫn rất cơ bản về Git. Bạn đọc muốn tìm hiểu sâu hơn có thể tham khảo tại những liên kết sau:

* Giới thiệu tổng quan về các tính năng của Git: <https://git-scm.com/about>.
* Tài liệu trên trang web chính thức của Git: <https://git-scm.com/doc>.
* Hướng dẫn sử dụng Git: <https://www.atlassian.com/git/tutorials>.
* Khoá học về Git trên Codecademy: <https://www.codecademy.com/learn/learn-git>.
* Git cheatsheet: <https://ndpsoftware.com/git-cheatsheet.html>.

**2. GitHub**

**2.1. Tổng quan về GitHub**

Dựa vào tên gọi thì các bạn cũng có thể đoán được rằng Git và GitHub có liên quan đến nhau. Trong phần giới thiệu về Git bên trên, ta biết rằng các thành viên của một dự án đồng bộ code với nhau thông qua một remote repository được lưu trữ trên máy chủ riêng biệt. GitHub là một công ty cung cấp dịch vụ lưu trữ các dự án được kiểm soát phiên bản bởi Git, có trang chủ tại địa chỉ <https://github.com/>. Tất cả các dự án của Got It được kiểm soát phiên bản bởi Git và lưu trữ trên GitHub.

Ngoài mục đích chính là nơi để lưu trữ các dự án sử dụng Git, GitHub là một nơi có thể giúp các lập trình viên cải thiện kĩ năng của bản thân. Hiện nay, GitHub là một trong những dịch vụ phổ biến nhất để lưu trữ các dự án mã nguồn mở. Các lập trình viên khác có thể tham gia đóng góp vào các dự án này thông qua việc sử dụng Git và GitHub. Nếu như bạn chưa đủ khả năng để đóng góp vào dự án mà bạn muốn tham gia, thì bạn cũng có thể học hỏi kinh nghiệm từ các lập trình viên khác bằng cách theo dõi các đóng góp của họ. Ngoài việc theo dõi các dự án mà bạn đã biết, bạn cũng có thể khám phá các dự án mới đang được nhiều người chú ý đến thông qua các tính năng [Explore](https://github.com/explore) và [Trending](https://github.com/trending). GitHub cũng rất thân thiện với người dùng mới bằng việc có những hướng dẫn rất rõ ràng và trực quan tại <https://guides.github.com/>.

**2.2. GitHub và tuyển dụng**

Không chỉ dừng lại ở việc cải thiện kĩ năng cá nhân, sử dụng GitHub hợp lý cũng là một cách để các lập trình viên có thể gây ấn tượng với nhà tuyển dụng. Sau khi tạo tài khoản GitHub, bạn có thể lưu trữ các dự án cá nhân của mình. Thông qua đó, nhà tuyển dụng sẽ có một hình dung rõ ràng hơn về khả năng của bạn, thay vì chỉ dựa vào những gì được viết trong CV.

Lịch sử hoạt động trên GitHub thể hiện nhiều điều về lập trình viên đó, ví dụ như các công nghệ đã sử dụng, khả năng cộng tác với các lập trình viên khác, khả năng quản lý dự án,… Tại Got It, các ứng viên có tài khoản GitHub (hoặc các dịch vụ tương tự) “đẹp” sẽ được đánh giá cao trong bước CV Review (các bạn có thể tìm hiểu kĩ hơn về quy trình tuyển dụng của Got It tại [đây](https://medium.com/got-it-vietnam/recruitment-got-it-tuy%E1%BB%83n-d%E1%BB%A5ng-nh%C6%B0-th%E1%BA%BF-n%C3%A0o-ph%E1%BA%A7n-1-a1cb0665a22f). Có thể nói, Github của một lập trình viên cũng giống như portfolio của một designer.

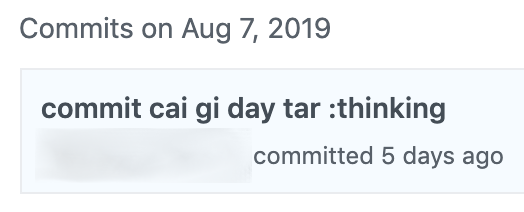
Câu hỏi được đặt ra là: Làm thế nào để có một tài khoản GitHub “đẹp”? Vì GitHub là nơi để lưu trữ các dự án sử dụng Git, nên sử dụng Git đúng cách là một tiêu chí để đánh giá ứng viên. Các sai lầm thường gặp khi sử dụng Git có thể nhắc tới việc dồn quá nhiều thay đổi vào một commit, hoặc viết commit message không rõ ràng, thiếu nghiêm túc,…

https://miro.medium.com/max/60/1*TceafScNVzYx56tF0jSx2A.png?q=20



Dồn quá nhiều thay đổi vào một commit

https://miro.medium.com/max/60/1*polADOxZFCAPL52OGDMknA.png?q=20

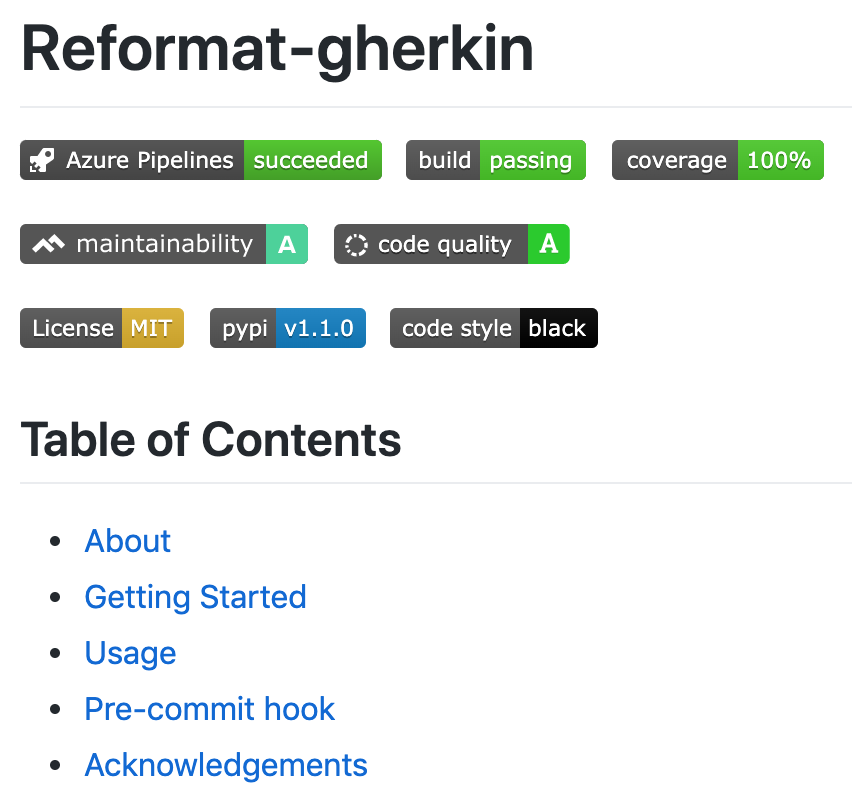


Commit message thiếu nghiêm túc

Ngoài việc sử dụng Git hợp lý, những điều nên làm để có thể gây ấn tượng thông qua GitHub có thể kể tới:

* **Code rõ ràng và dễ đọc, theo quy chuẩn.** Với các dự án mã nguồn mở, bất kì ai cũng có thể xem toàn bộ mã nguồn, qua đó có thể đánh giá chất lượng của dự án. Có thể nói rằng chất lượng code là tiêu chí quan trọng nhất khi đánh giá ứng viên thông qua các dự án cá nhân. Việc viết code dễ đọc có thể bắt đầu bằng cách làm theo các quy ước thông dụng như PEP8 với Python và ESLint với JavaScript. Cài đặt các pre-commit hooks để đảm bảo rằng các quy ước được tuân thủ cũng là một điểm cộng.
* **Sử dụng các công cụ hỗ trợ.**Ngoài cách đảm bảo chất lượng code thông qua việc làm theo các quy ước, viết test giúp ta chắc chắn rằng code đã được viết đúng logic. Viết test đúng và đủ cũng là một phần không thể thiếu trong các dự án được đầu tư nghiêm túc. Ta có thể cài đặt sao cho test được tự động chạy mỗi khi push code lên GitHub với các công cụ như [Travis CI](https://travis-ci.com/), [CircleCI](https://circleci.com/), [Azure Pipelines](https://azure.microsoft.com/en-us/services/devops/pipelines/); cùng với đó là theo dõi tỉ lệ code đã được kiểm tra bởi test với các công cụ như [Coveralls](https://coveralls.io/), [Codecov](https://codecov.io/),... Thông qua khả năng sử dụng các công cụ một cách thích hợp, ta có thể đánh giá được xem tác giả có quan tâm và đảm bảo chất lượng của dự án hay không, cùng với đó là khả năng tìm hiểu và áp dụng các công nghệ, kĩ thuật phổ biến.

https://miro.medium.com/max/60/1*2WlGeW4EManEjaQUQTOE3A.png?q=20



Một ví dụ về việc sử dụng các công cụ hỗ trợ như CI, coverage,…

* **Cung cấp mô tả rõ ràng về các dự án.** Các dự án cá nhân cần có một tiêu đề và mô tả ngắn gọn nhưng vẫn đầy đủ. Ngoài ra, mỗi dự án nên có một file README chứa những thông tin cơ bản về dự án. File README này sẽ được hiển thị tại trang chủ của dự án, ví dụ như <https://github.com/psf/black>, nó sẽ là ấn tượng đầu tiên khi truy cập vào dự án của bạn trên GitHub. Vì vậy, có một file README rõ ràng, dễ đọc, mô tả chi tiết về dự án là một điểm cộng rất lớn trong quá trình review GitHub. Các bạn có thể tham khảo các bài viết sau để tìm hiểu thêm về cách viết README: <https://www.makeareadme.com/>, <https://github.com/kylelobo/The-Documentation-Compendium>.
* **Quản lý dự án.** GitHub cung cấp khả năng quản lý dự án với tính năng [Issues](https://guides.github.com/features/issues/). Với Issues, bạn có thể theo dõi bug và các yêu cầu tính năng mới cho mỗi dự án. Tận dụng hợp lý tính năng này là một cách để thể hiện kĩ năng quản lý dự án và làm việc nhóm.
* **Giấy phép (License).** Có thể bạn chưa để ý đến license, nhưng đây là một điều rất quan trọng với các dự án mã nguồn mở. Chúng cho phép lập trình viên khác sử dụng, cũng như đóng góp các thay đổi, nâng cấp cho dự án của bạn. Một số loại giấy phép phổ biến phải kể đến MIT, Apache License 2.0,… Bạn có thể tìm hiểu thêm tại <https://choosealicense.com/>, hoặc tham khảo một ví dụ về giấy phép MIT tại <https://github.com/psf/black/blob/master/LICENSE>.
* **Tạo trang cá nhân với GitHub.** Có một trang web cá nhân là một trong những điều gây ấn tượng mạnh với nhà tuyển dụng. GitHub cho phép bạn tạo trang web cá nhân và trang web cho các dự án của bạn. Hướng dẫn chi tiết có tại <https://pages.github.com/>.