GitHub 공동작업을 위한 Fork 사용(1)

Intro …

GitHub에서 공동작업을 할 경우, conflict 방지를 위해 대부분 각자의 branch를 이용하여 작업을 하게 된다.

branch를 사용하면, 로컬 작업 후 master 브랜치로 바로 push하지 않고 원격지의 branch로 먼저 push한 후에, pull request를 하여 안전하게 merge 작업을 함으로써 안전한 공동 집필작업을 할 수 있을 것이다.

Pull vs Pull Request

pull request는 Pull과 반대의 의미를 갖고 있는데, pull은 원격 리파지토리의 변경사항을 로컬로 당겨오는 반면, pull request는 원격 리파지토리가 내 branch의 변경사항을 당겨가도록 요청을 하는 것이다.

궁극적으로 내 branch에서 발생한 변경사항을 remote 리파지토리가 알아차리고 반영하게 한다는 점에서 push와 비슷하지만 주요한 차이가 있다.

|  |  |  |
| --- | --- | --- |
|  | push | Pull request |
| 사용가능한 위치 | 로컬 🡪 원격으로만 push가 가능  branch간 push는 불가능 | 원격 branch 사이에서만 pull request가 가능 |
| mergy여부 | push와 동시에 merge가 발생 | 유보된 merge  pull request를 승인하는 과정 중에 merge가 발생하며, pull request만으로는 merge되지 않는다. |

pull request가 생성되고 나면, 요청받은 리파지토리에서 요청받은 pull request를 승인해야 최종적으로 merge가 되면서 마무리된다.

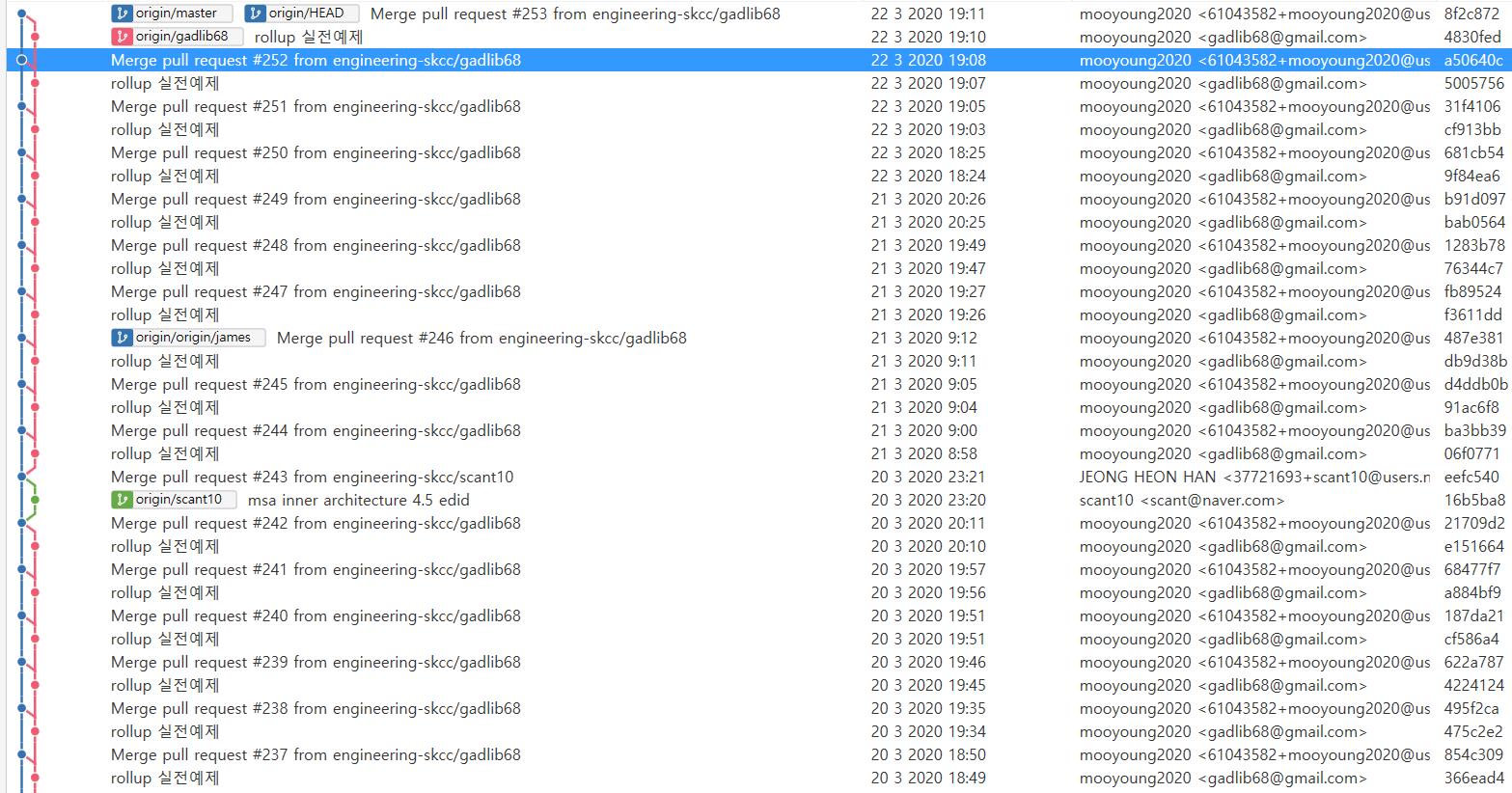
Pull request를 하게 되면 두 단계에 걸쳐 소스 반영이 되므로 약간 번거로운 측면이 있지만, 우아하고 안정적으로 소스를 merge할 수 있다.

그런데, GitHub에 단순히 소스를 올리는 작업이 아니라, 블로그 포스팅을 해야 한다면 각자의 로컬에서 작업한 소스파일(주로 MD 또는 image파일들)을 GitHub의 원격 리파지토리의 해당 branch에 올리고, master branch쪽으로 pull request를 생성한 후에 다시 master branch에서 해당 pull request를 승인하고 나서요 최종적으로 소스 merge가 되고 나서야, 실제 브라우저에서 작동되어 나온 블로그 결과물을 확인할 수 있을 것이다.

|  |  |
| --- | --- |
|  |  |

만약 merge 작업까지 모두 마치고 나서 결과물을 확인했는데 오타 또는 이미지 파일 깨짐과 같은 현상이 있다면, 이미 대중에게 공개된 상황에서 급히 오타를 수정하고 이미지를 고치느라 위와 같은 작업으로 수십번 삽질을 하게 되는 상황이 올 수도 있다.

(반복된 삽질의 흔적)



또한 잘못된 글귀를 고치느라 소요되는 시간 동안 많은 방문객이 이미 왔다갔다면 수십번의 삽질에도 불구하고, 이미 널리 공개되어 난감한 상황이 될 수도 있다.

자, 이런 경우를 어떻게 해결해야 할까?

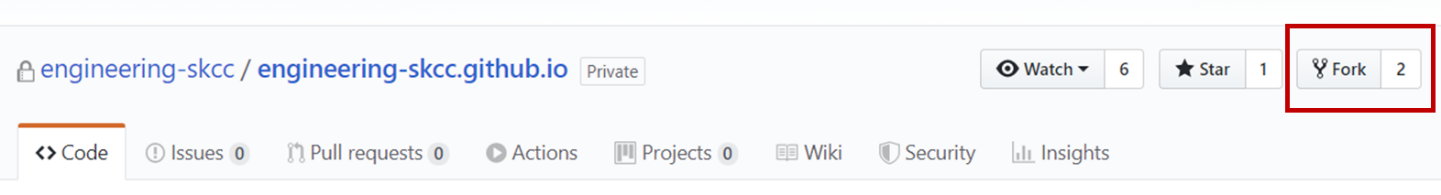
이미 branch를 이용한 pull, push, pull request 기능을 마스터했다면 이제부터 fork를 이용한 “따로또같이”의 공동작업이 가능하다.

fork라고? UNIX에서 사용하던 그 fork와 같은 명령어다. UNIX에서 fork를 배울 때 이론만으로는 이해가 안되고, 직접 실습을 통해서 이해가 되었던 적이 있다.

GitHub의 fork도 비슷하다. 직접 해봐야 안다.

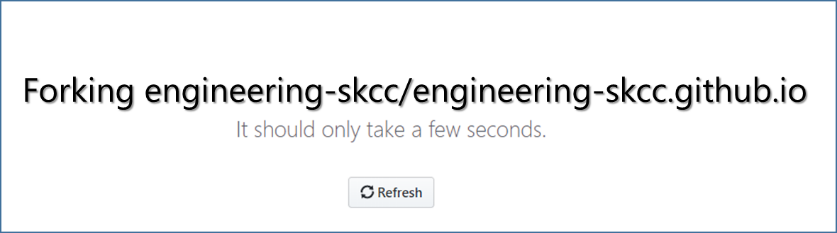
일단 Fork를 해보자.

대상 리파지토리 우측에 보면 fork라는 조그만 네모박스가 있다.



오른쪽에 숫자가 있는데, 이것은 이미 fork를 해간 숫자이다.

눌러서 누가누가 fork했는지 살짝 들여다본 후, 과감하게 Fork를 누르면…. 바로 Fork가 진행된다. (놀라지 마시라)

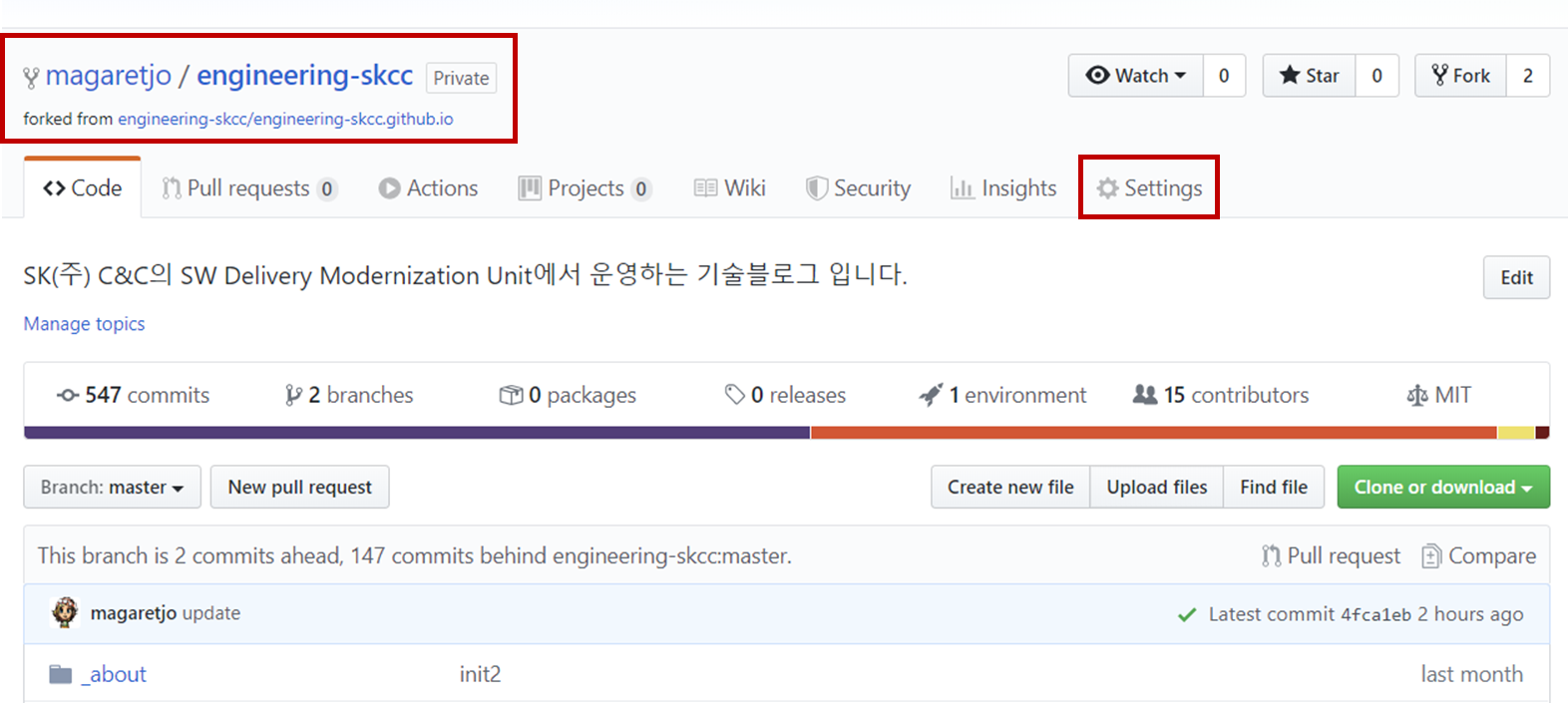


순식간에 자신의 리파지토리로 복제가 된다. 원래의 리파지토리 안에 있던 모든 소스와 로그가 고스란히 내 리파지토리로 생겨난 셈이다.

차이점은 아래 그림과 같이 상단에 GitHub 계정명이 다르게 나타난다는 점이다.

(원래는 magaretjo/engineering-skcc.GitHub.io로 생기는데, 아래와 같이 자신이 선호하는 이름으로 변경이 가능하다.)

또 하나의 차이점은 상단 메뉴에 Settings가 나타난다. Original 리파지토리와 달리 fork해온 리파지토리는 내 소유의 리파지토리이므로 Settings를 통해 마음껏 변경이 가능하다.



자, 이제 순식간에 뚝닥 원본과 동일한 리파지토리가 만들어졌으며, 이제 이 리파지토리를 통해 내맘대로 테스트해볼 수 있는 블로그 포스팅도 가능할 것이다.

Fork 리파지토리를 사용하면 뭐가 좋을까?

fork의 긴 여정을 떠나기 전에 우선 뭐가 좋은지부터 알아두는 게 좋을 거 같다.

* 별도의 Branch 없이 편하게 작업 가능

일단, original 리파지토리와 별도로 fork한 리파지토리에서 작업을 하니, 잘못 혼돈하여 master 브랜치를 건드릴 염려가 없다.

자신의 branch로 체크아웃하지 않고 master에서 작업하다가 종종 소스가 꼬이는 경우가 있는데, fork해온 리파지토리는 온전히 자신만의 것이므로 굳이 another branch를 만들 필요없이 default branch에서 작업해도 된다.

* Private 리파지토리를 유지하면서 publish도 가능

이게 무슨 말이냐….

GitHub에서는 리파지토리를 만들 때마다 private인지 public인지를 결정해야 한다. 유료라고 한다면 걱정할 필요가 없는데, 무료 계정을 사용할 경우 private 리파지토리에서는 아쉽게도 웹사이트처럼 publishing이 불가능하다.

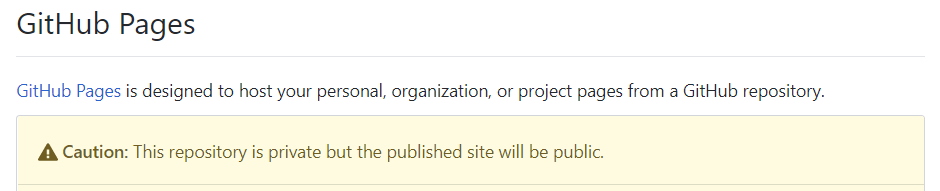
웹브라우저에서 블로그를 확인하려면 Publising이 필수이므로 어쩔 수 없이 public 옵션을 선택해야 하는데, public이라 함은 리파지토리 소스가 외부에 공개될 뿐 아니라 아무나 다 내 리파지토리에 커밋을 할 수 있게 된다. 헐…

따라서, 아무나 내 리파지토리를 못 쓰게 하려면 private으로 잠가야 한다.

그런데, fork를 하게 되면 original 리파지토리의 속성이 유지된다.

우리의 original 리파지토리가 private이었기 때문에, private하게 유지가 되면서 publishing도 가능한 것이다.

Settings > Options메뉴에서 하단으로 죽 이동하면 다음과 같은 메시지를 확인할 수 있다.

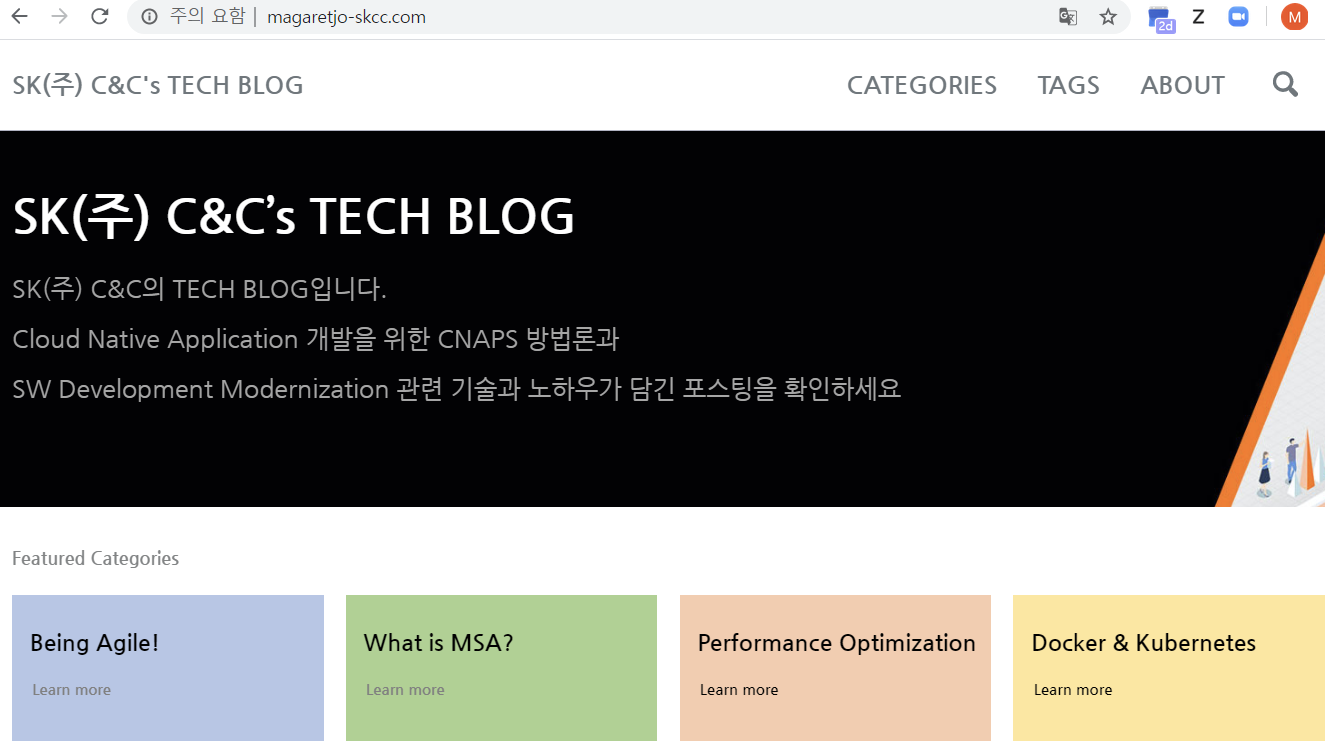


만약, fork를 하지 않고 그냥 복제를 해온다면 어떻게 될까? 이 경우에는 original 리파지토리의 속성을 따르는게 아니라, 위에서 언급했듯이 그냥 내 계정의 유료 여부에 따라 사용범위가 제한된다.

* Original 페이지와 다른 나만의 페이지 확인 가능

Fork를 권장하는 궁극적 목적이 바로 여기에 있다.

Original 리파지토리에서는 master와 merge까지 해야 확인할 수 있었던 포스팅 페이지들을 이제 나만의 fork 리파지토리에서 변경된 사항을 push만 하고, 바로 확인이 가능하다. 짠~~



Original 사이트 같은데, 저 위의 URL을 보면, original 사이트와 다른 주소가 보인다.

짝퉁 사이트인 셈이다.

이제, Fork를 사용하면 하고 있는 작업들이 좀 편해질 것 같은 생각이 들었다면 세부적으로 들어가보자.

위와 같이 되기 위해서는 해야 할 일들이 좀 있다.

구체적인 작업사항에 대해서는 (2)편에서 소개하겠다.

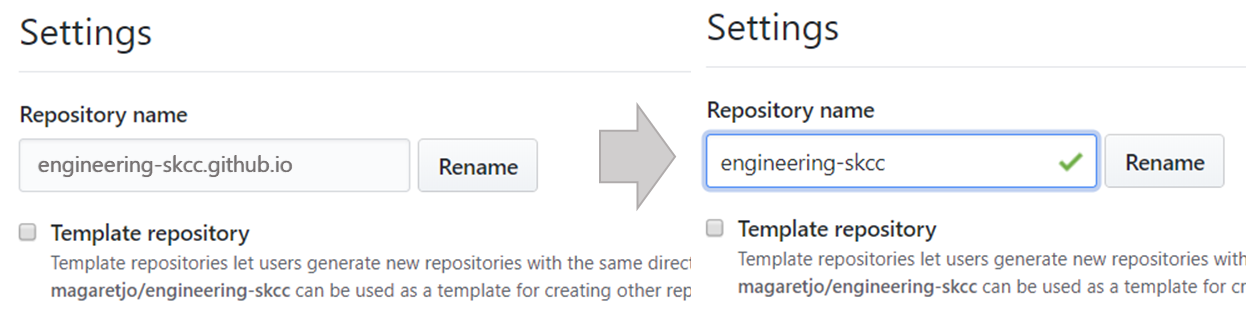
GitHub 공동작업을 위한 Fork 사용(2)

지난 포스팅에서는 GitHub를 이용한 공동 작업시, Fork를 사용하게 될 경우 어떤 점이 좋은지를 설명하였다.

그리고, 일단 Original 리파지토리에서 일단 Fork를 하는 방법이 매우 간단하다는 것도 배웠다. 이제 Fork해온 리파지토리를 가지고, 나만의 작업할 수 있는 환경을 만들어보도록 하자.

리파지토리 이름 변경

Settings > Options 메뉴로 들어가면 가장 먼지 리파지토리 이름이 나온다. 기존 명칭과 다르게 나만의 이름을 지어보자. Rename 버튼을 눌러야 실제로 반영이 된다.

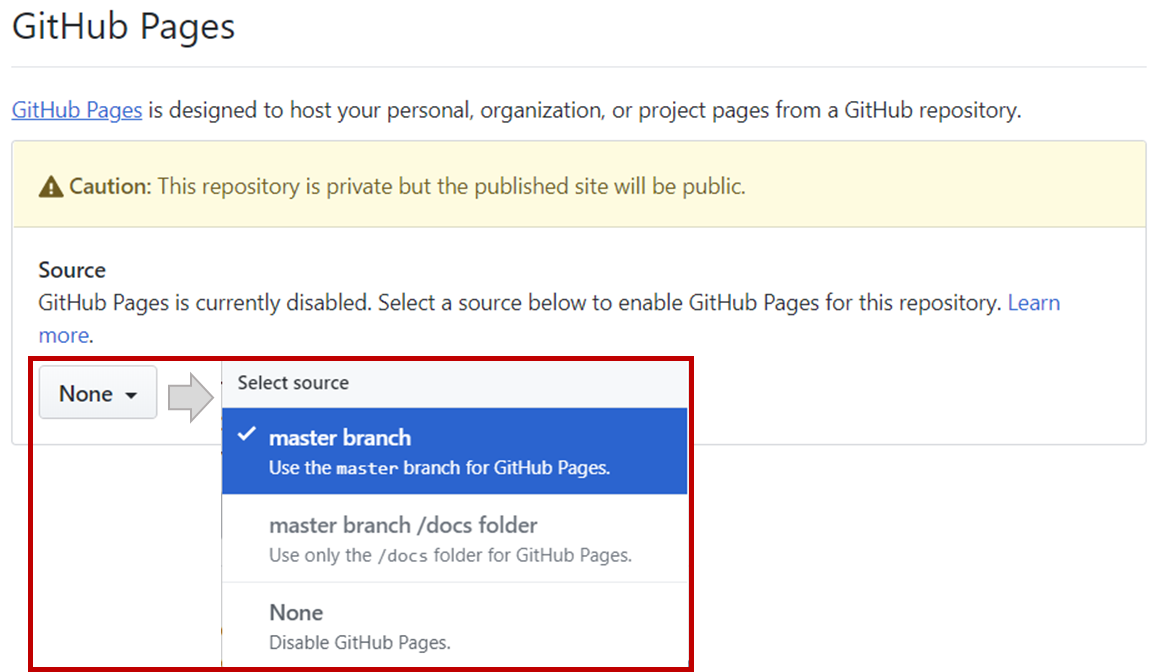


위와 같이 하여 fork해온 리파지토리의 이름을 바꿀 수 있다.

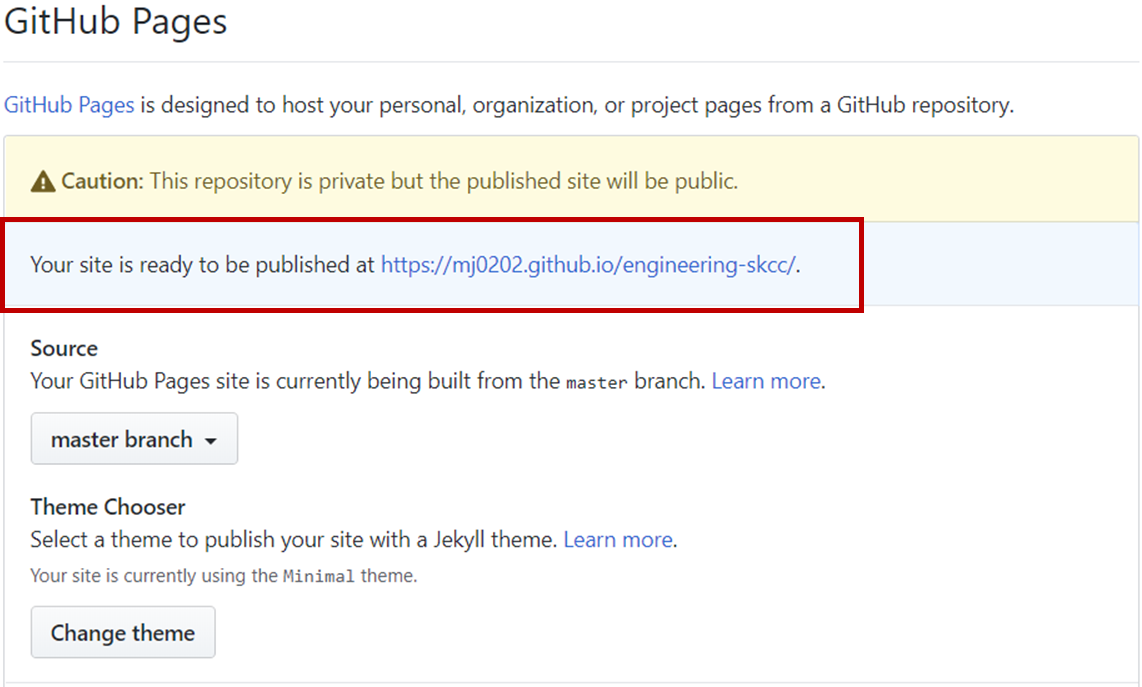
Publishing을 위한 설정

리파지토리 이름을 변경한 후에 화면 아래로 스크롤을 해서 내려가면 다음과 같이 GitHub Pages 세팅 화면이 나타난다.

Source 부분의 None을 눌러서 포스팅할 재료를 선택하도록 하자.



master 브랜치를 선택하면 다음과 같이 화면이 바뀔 것이다.



자 이제 여기까지 했으면, 90%의 준비가 끝났다.

절대 경로의 문제

Publishing URL을 눌러 짝퉁 사이트가 생겼는지 확인해보자.

아마도 뭔가 되는 것 같기는 한데… 뭔가가 좀 이상한 것 같기도 하고…

이상한 점을 발견했다면 당신은 좀 대단한 사람?

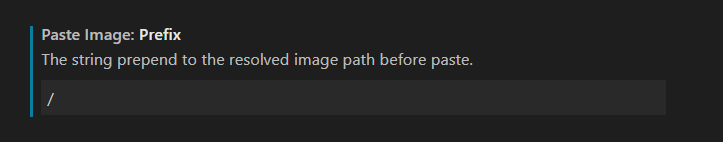
짝퉁사이트에 포스팅된 글의 링크나 이미지들의 속성을 확인해보자.

아마도 “[https://engineering-skcc.github.io/~”](https://engineering-skcc.github.io/~)로 시작하는 주소를 달고 있을 것이다.

이는 포스팅 소스 안에 링크나 이미지 주소를 작성하면서 절대경로, 즉 “https://”로 시작하는 full path를 지정했기 때문일 것이다.

Original 리파지토리에 이미 해당 이미지를 올린 상태에서 복제해왔기 때문에 지금 상태에서는 문제가 없겠지만, 만약 fork 리파지토리에서 새로운 글과 이미지를 올리게 된다면 해당 이미지는 모두 확인할 수가 없다.

따라서, 새로 생긴 사이트에 맞게 Path가 적용되게 하기 위해서는, 포스팅용 MD 파일에서 “[https://engineering-skcc.github.io/~”](https://engineering-skcc.github.io/~)의 절대경로대신 “/”를 사용해야 한다.



약간 귀찮을 수 있지만, 요새는 에디터가 워낙 좋아서 순삭이다.

무엇보다, 우리가 포스팅한 “[https://engineering-skcc.github.io/~”](https://engineering-skcc.github.io/~)의 경로는 언제든 다른 곳으로 바뀔 수 있다. 따라서 저렇게 절대 경로를 사용하는 것은 바람직하지 않다.

망설이지 말고 본인의 소스에서 저런 링크를 찾아 모두 바꾸도록 하자.

단, 바꿀 땐 바꾸더라도, 본인이 작성한 MD 파일만 바꾸고, 남의 글은 건드리지 않도록 한다. 나중에는 merge시 남의 소스와 충돌이 나지 않으려면…

Fork 리파지토리에서 Path를 모두 바꾸는 작업을 한 뒤, commit / push하고 제대로 반영되었는지 다시한번 확인해 보자. 짜자…. 잔…. 어라?

루트 경로의 문제

아마도 아까보다 더 이상해진 상황을 접하게 될 것이다.

그나마 Original 사이트에서 끌고 오던 이미지조차 보이지 않게 되었다.

이번엔 무엇이 문제일까?

애석하게도 GitHub에 포스팅된 모든 리소스 파일의 루트 디렉토리는 “https:// … github.io/”이다. 어떤 리파지토리를 만들더라도 포스팅하는 모든 웹페이지의 루트 디렉토리는 본인의 GitHub계정 주소가 된다.

이에 따라 내부의 다른 페이지 링크나 이미지 참조시에는 “https://... github.io/ (repository name) /” 또는 “/ (repository name) /”을 Prefix로 붙여야 한다.

하지만, 우리가 원본에서 이러한 repository name을 생략할 수 있었던 것은 GitHub가 계정명과 동일한 대표 리파지토리일 경우 repository name이 들어간 경로는 생략할 수 있도록 해주기 때문이다. (음…. 이해시키기가 대략 난감)

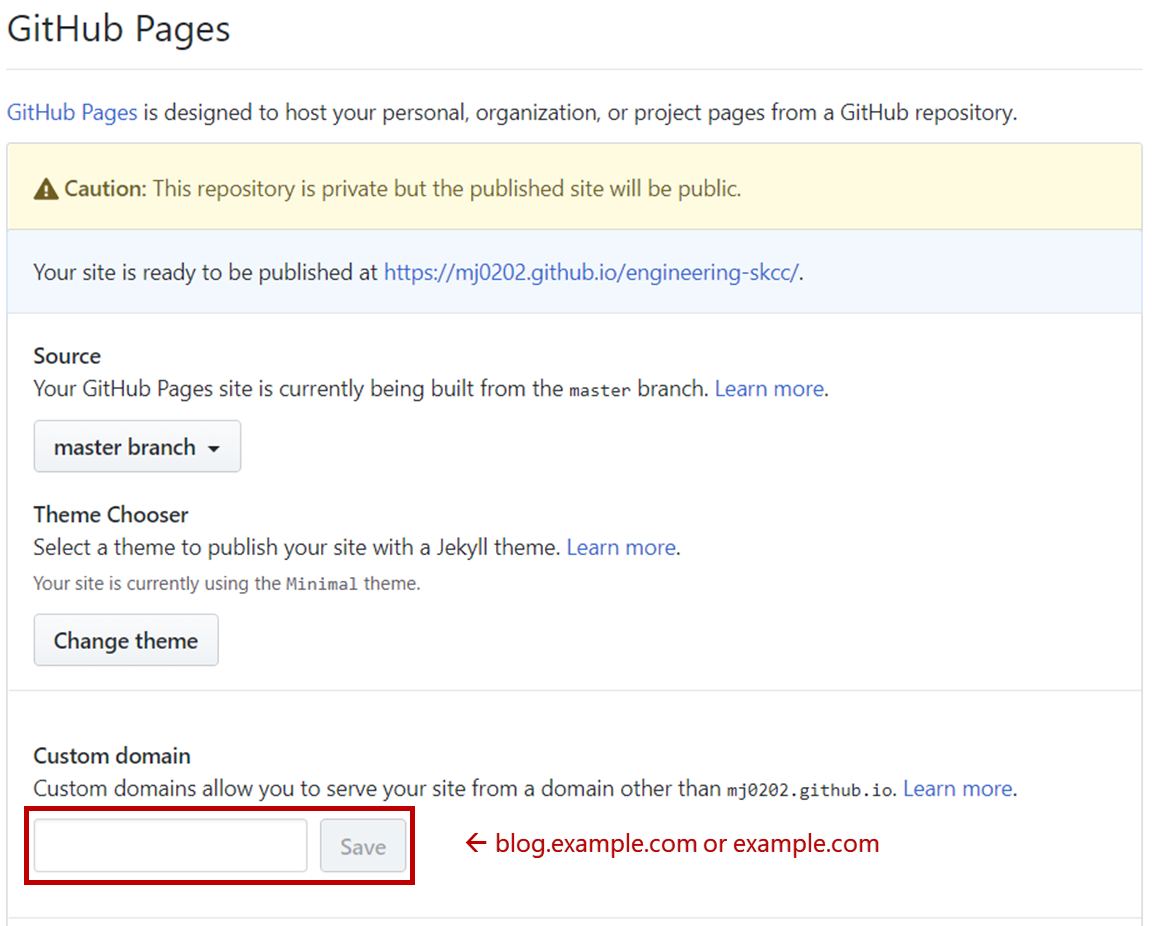
즉, Original 리파지토리에서는 홈페이지 주소가 “https://engineering-skcc.github.io/engineering-skcc/”가 되어야 하지만 GitHub가 융통성 있게 “https://engineering-skcc.github.io/”을 홈페이지로 인식하게 해준다. 따라서 내부 리소스 참조시 “/”만 사용해도 리소스는 대표 리파지토리 밑에 있는 것으로 인식한다.

하지만, 짝퉁 사이트의 경우에는 홈페이지가 “https://(본인의 GitHub계정).github.io/(fork repository 명칭)/”과 같이 되며, 내부 리소스 참조시 “/(fork repository 명칭)을 사용해야 한다.

따라서, 이를 해결하는 방법은 “https://(본인의 GitHub계정).github.io/(fork repository 명칭)/”으로 된 홈페이지 주소를 “https:// ~.com/” 과 같이 바꾸어 주고, 이 주소를 루트로 인식하게 만들면 된다.

Custom domain 설정하기

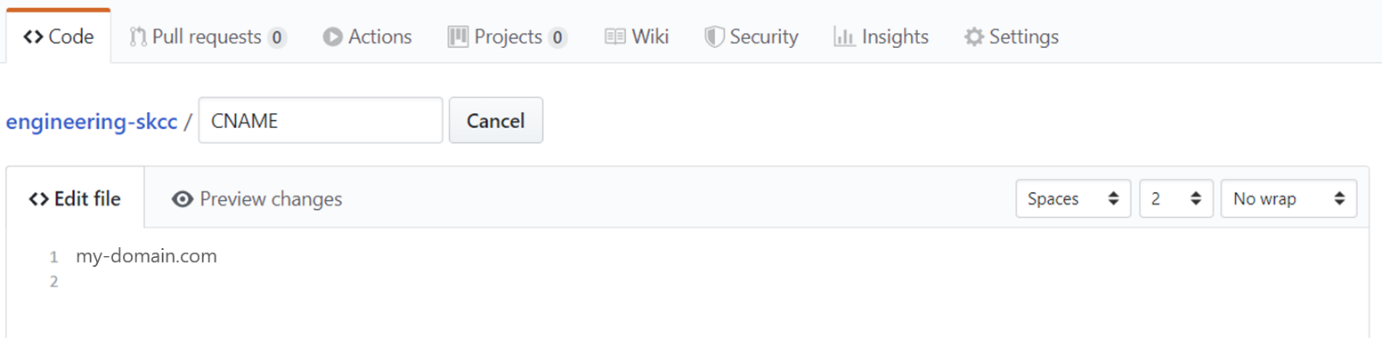
GitHub에서는 default로 설정되는 퍼블리싱 URL을 다음의 설정 기능을 이용하여 바꿀 수 있다.



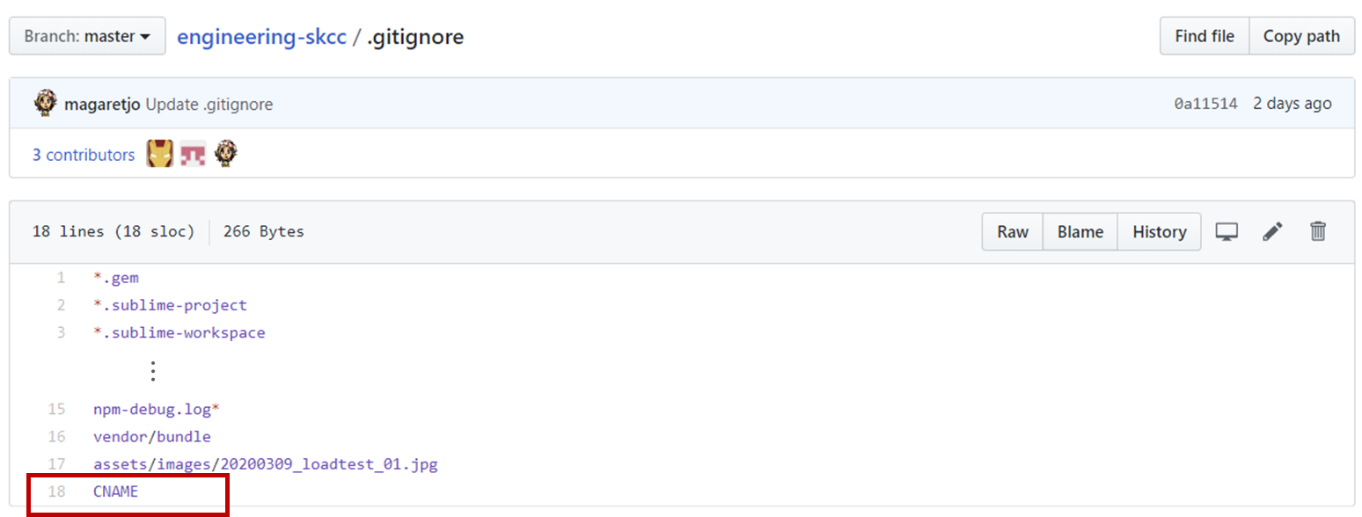
blog.example.com 또는 example.com과 같은 식으로 나만의 도메인 이름을 지어보자.

그리고, Save 버튼을 누르면 다음과 같이 fork 리파지토리의 루트 아래에

CNAME file이 생성된다.



그리고 이와 같이 생긴 CNAME 파일이 Original 리파지토리에 병합되지 않도록 하기 위해 .gitnore 파일을 열어 그 안에 CNAME 파일명을 반드시 추가하자.



이제 브라우저에서 새로 생긴 주소로 접속해보자.



애석하게도 연결이 거부될 것이다.

마지막으로 한 스텝이 남아있다. 정말 마지막 스텝이다.

hosts 파일에 domain 주소 추가하기

우리가 새로 만든 도메인 주소는 공식적인 DNS에 등록되어 있지 않기 때문에 서비스되지 않는다. DNS에 등록하지 않고도 나오게 하려면 hosts 파일에 ip와 해당 주소를 등록하면 된다. 다음과 같이 hosts 파일에 등록하자.

185.199.111.153 my-domain.com

185.199.110.153 my-domain.com

185.199.109.153 my-domain.com

185.199.108.153 my-domain.com

185.199.xxx.153의 IP주소는 GitHub 사이트의 IP주소이다.

이제 내가 설정한 domain 주소로 접속을 해보자. 모든 것이 제대로 동작되는 것을 확인할 수 있을 것이다.

짝퉁 사이트가 무난히 동작되는 것을 확인했으니, 이제 (3)편에서는 Fork 리파지토리에서 변경한 소스를 Original로 다시 어떻게 반영해야 하는지에 대해서 알아보도록 하자.

GitHub 공동작업을 위한 Fork 사용(3)

Fork 리파지토리를 Original에 반영하려면 ?

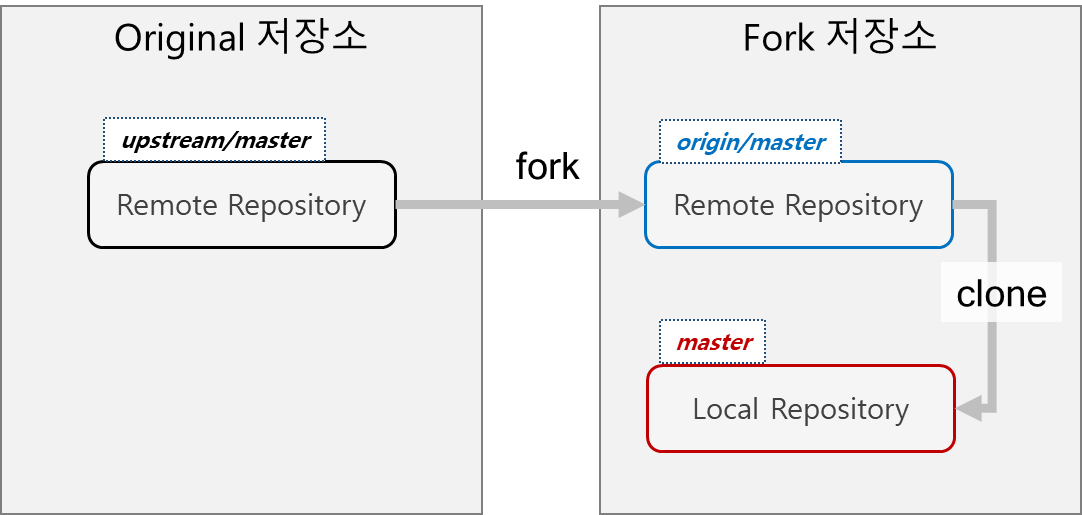
이렇게 만들어진 Fork 리파지토리는 나만의 작업환경이라 하더라도 공동작업을 위해 결국 어느 순간에는 Original 리파지토리로 merge를 해야 할 것이다.

어떤 방법들이 있는지 알아보자.

* 로컬로 다운로드하기

Fork 리파지토리에서 변경작업을 하기 위해서는 기존에 하던대로 본인의 로컬PC로 다운로드 받아야 한다.

$git clone [ fork 리파지토리 URL ]



Fork 리파지토리에서는 master 브랜치를 그대로 사용해도 된다. 나만의 작업용 브랜치를 만들어도 된지만, GitHub Page 페이지에서는 master 브랜치의 소스만을 사용하기도 하고, 자신의 master브랜치와도 병합을 해주어야 하므로 약간의 귀찮음이 발생한다.

그냥 master를 사용하되 Original의 master와 헷갈리지 않도록 하자.

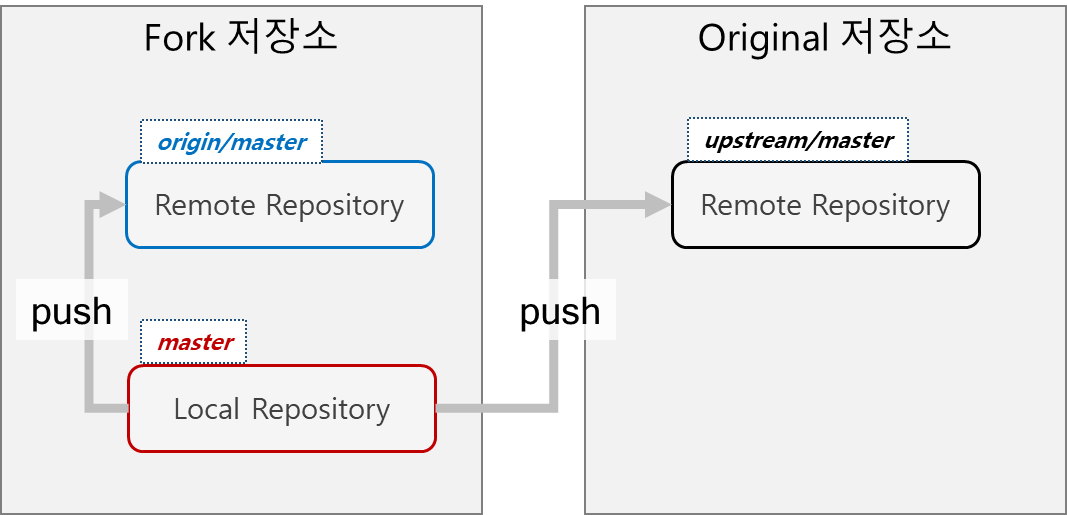
* Upstream에 직접 push하기

Fork 리파지토리는 원본 리파지토리 입장에서는 일종의 브랜치와 동일하다.

따라서, Fork 리파지토리의 변경된 내용들을 Original 리파지토리로 push할 수도 있고, 또는 우아하게 Push request를 할 수도 있다.

즉, 기존에 각각의 브랜치를 이용하여 원본인 master브랜치로 merge하듯이 Fork 리파지토리도 똑같이 작업할 수 있다.

단, 원본 리파지토리를 다운로드한 로컬 PC에서 다시 Remote 리파지토리를 참조하기 위해, origin master라고 썼다면, Fork 리파지토리를 다운로드한 로컬 PC에서는 upstream master를 사용한다. Fork 리파지토리를 다운로드한 로컬 PC 입장에서 origin master는 GitHub에 있는, 즉 Remote에 있는 Fork 리파지토리의 master이다.

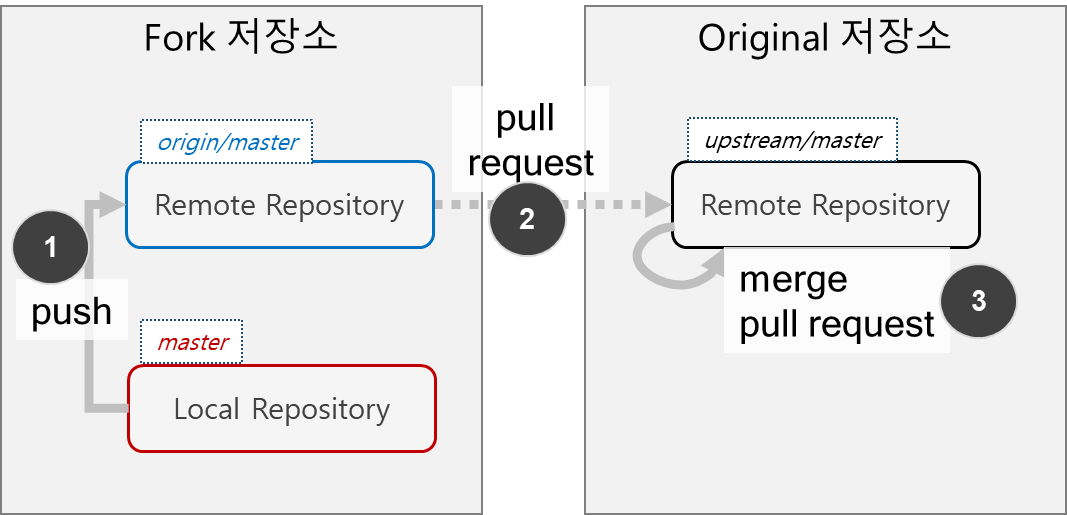


* Pull request 이용하기

하지만, 브랜치 사용시에도 origin master로 직접 push는 지양했듯이 Fork 리파지토리도 upstream으로 직접 push하지 말아야 한다.

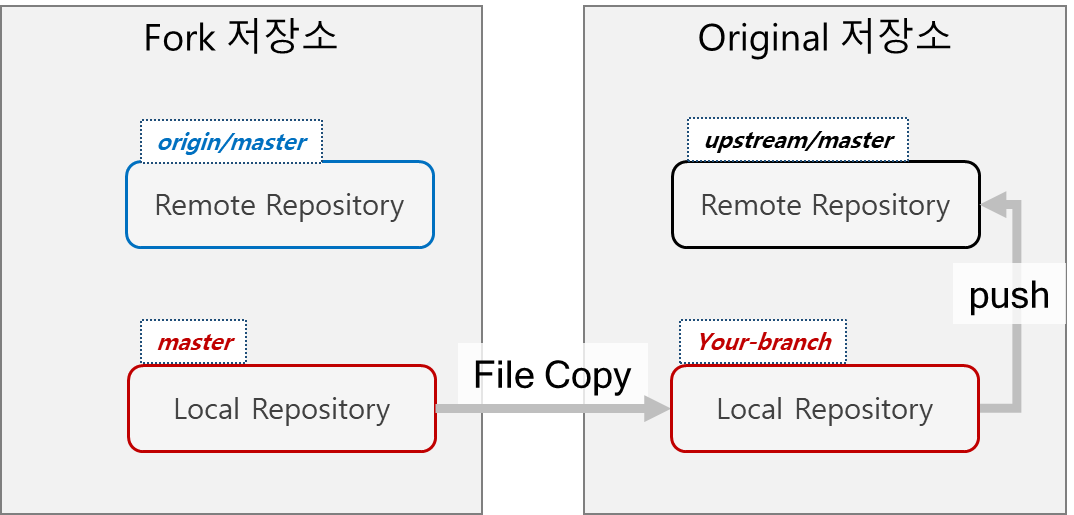
일단, Fork 리파지토리를 만든 목적인 짝퉁 사이트에서 미리 웹페이지를 확인해보기 위해서라도 일단은 자신의 원격 리파지토리로 먼저 push를 하는 게 바람직하다.

웹페이지 결과가 만족스러워서 Original 저장소에 반영하기 원한다면 그 다음에 우리가 아는 Pull Request 방법을 쓰는 것이 제일 안전하다.



* Copy & Paste ?

Fork에서 마음껏 변경하고 난 뒤, 그 흔적들을 Original로 보내지 않고 아름답게 포장된 최종 이미지만을 Original로 보내고 싶다면 아래와 같이 작업을 할 수도 있다. 물론 이 작업은 비추이다.



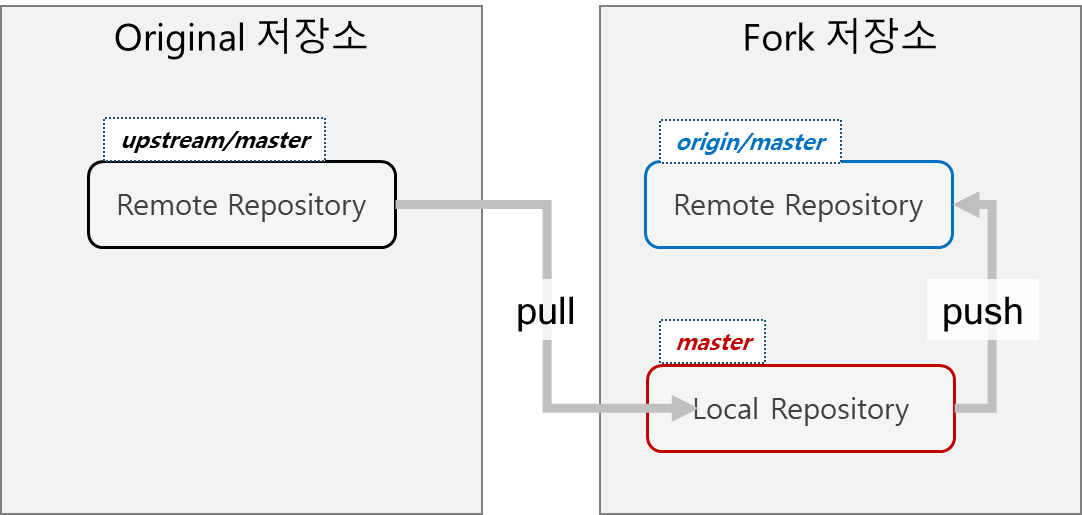
Original 변경 사항을 Fork 리파지토리에 반영하기

역으로 Original에서 변경된 사항을 내 Fork 리파지토리에 반영해야 할 필요가 있는 경우도 있을 것이다. 공통사항이 변경되었거나 최신의 원본 소스가 필요할 경우, 그냥 pull 명령어를 사용하여 upstream으로부터 당겨오면 된다.

$ git pull upstream master

이와 같이 명령어를 사용하면 내 로컬에만 반영이 되고 원격에 있는 Fork 리파지토리에는 반영이 안되었으므로, 다시 push 명령어를 사용하여 원격에도 반영해야 한다.

$ git push origin master



결과적으로 Fork 리파지토리에서도 앞에 upstream이냐 origin이냐에 따라 자유자재로 소스를 동기화시킬 수 있다. 매우 매력적이 아닐 수 없다.

정리 및 당부사항

10여명이 공동 집필진이 구성하는 기술블로그에서 좀 더 작업을 편리하게 하고자 이것저것 시도해보다가 결국에 찾은 방법이 Fork라는 방법이었다.

처음에는 원본 리파지토리를 통째로 개인 리파지토리에 복사하여 작업을 해보았다.

내 리파지토리에서 MD소스파일이 웹페이지 형태로 포스팅된 결과 화면을 아무때나 테스트해볼 수 있는 장점이 있었지만, 원본 파일과의 버전관리가 어려웠고 무엇보다 개인 리파지토리를 public으로 공개해야 하는 문제가 있기 때문에 문을 닫아야 했다.

Fork를 어떻게 사용해야 하는지도 모르는 상태에서 이것저것 눌러보면서 마침내 원하는 답을 찾게 되었다. 원하는 답을 찾게 되어 기쁘다.

몇번의 시행착오가 있었기 때문에 어떤 점을 주의해야 할지도 몸소 체득하였다.

Pull request를 사용하여 Fork 리파지토리를 원본으로 merge할 때, 마지막 스냅샷만 merge되는 것이 아니라 일반 브랜치 merge와 마찬가지로 그동안 발생한 모든 commit 모두가 Original에 반영되어 버린다.

앞에서도 이야기했지만 Fork한 리파지토리를 내 마음대로 작업할 수 있다고 해서 다른 사람이 작업하는 소스 파일까지 마음대로 변경했다가는 merge하는 순간에 함께 Original로 모두 반영이 되어 버릴 수 있기 때문에 각별한 주의가 필요하다.

물론 Git의 구조상 이러한 반영 작업은 기술적으로 문제는 없지만, 다른 사람이 집필한 포스팅 내용을 엉뚱한 것으로 바꾸거나, 특히 본인 입장에서 필요없다고 지워버린 파일이 그대로 원본 리파지토리에서도 삭제되는 과오를 범할 수 있으므로…

이럴 땐 .gitnore를 활용해보는 것도 도움이 될 것이다.

Fork에 대한 이야기는 여기서 마친다.