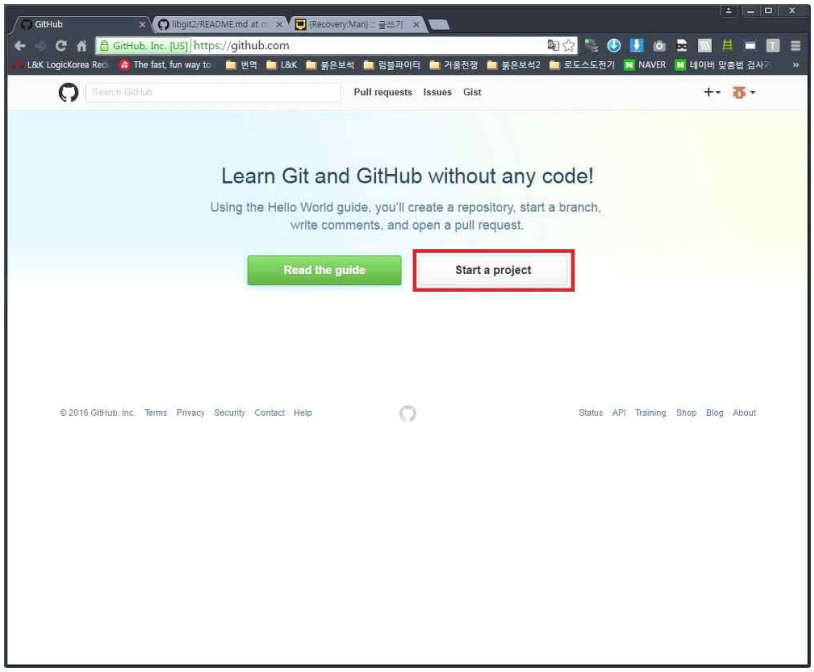
출처 : http://recoveryman.tistory.com/251?category=635733

*** 깃허브 프로젝트 만들어 등록하기 ***

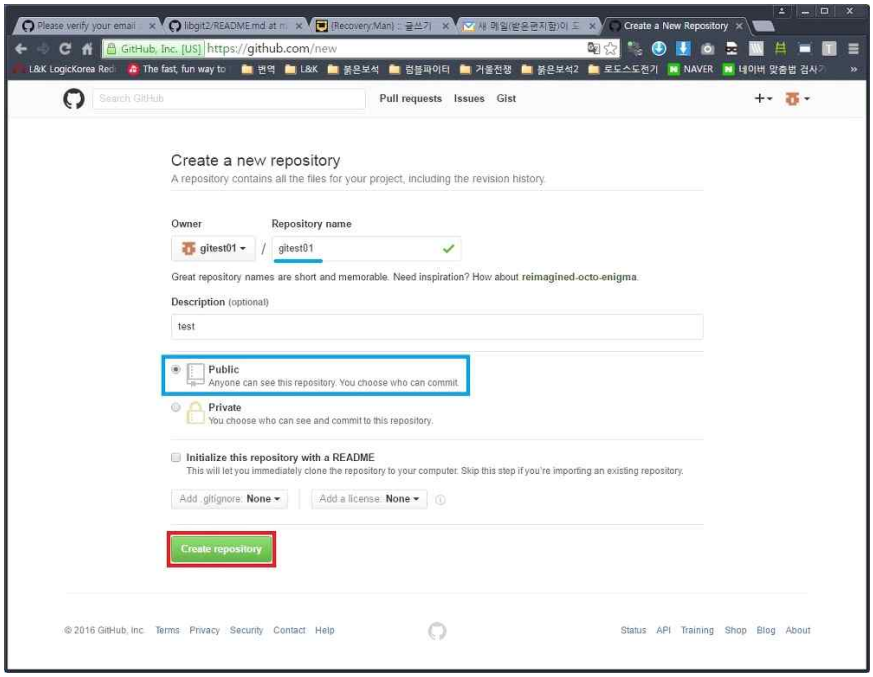
1. 깃 허브 웹 사이트를 방문해서 회원가입을 진행해 주도록 합니다.(들어가기)
https://github.com

2. start project

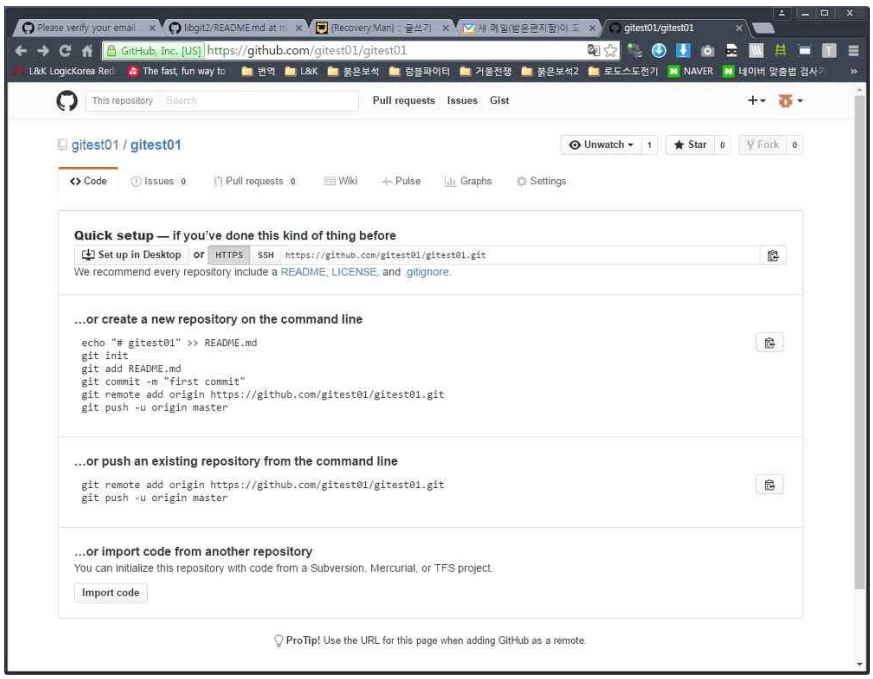


3. 일단 저장소를 만들어야 이 GitHub를 통해 소스 공유가 가능해 집니다.

- Owner 는 소유자를 말합니다. 회원가입 할 때 Username 부분에 기입했던 닉네임이 그대로 들어가 있네요.
- Repository name 는 저장소의 이름을 뜻합니다.
- Description 은 이 저장소는 무슨 저장소인지 입력하는 부분 입니다.
기본적으로 'Public'에 체크가 되어 있음. 이건 무료 저장소를 사용하는 겁니다.
Create repository 를 눌러 주시면 저장소가 만들어 집니다.

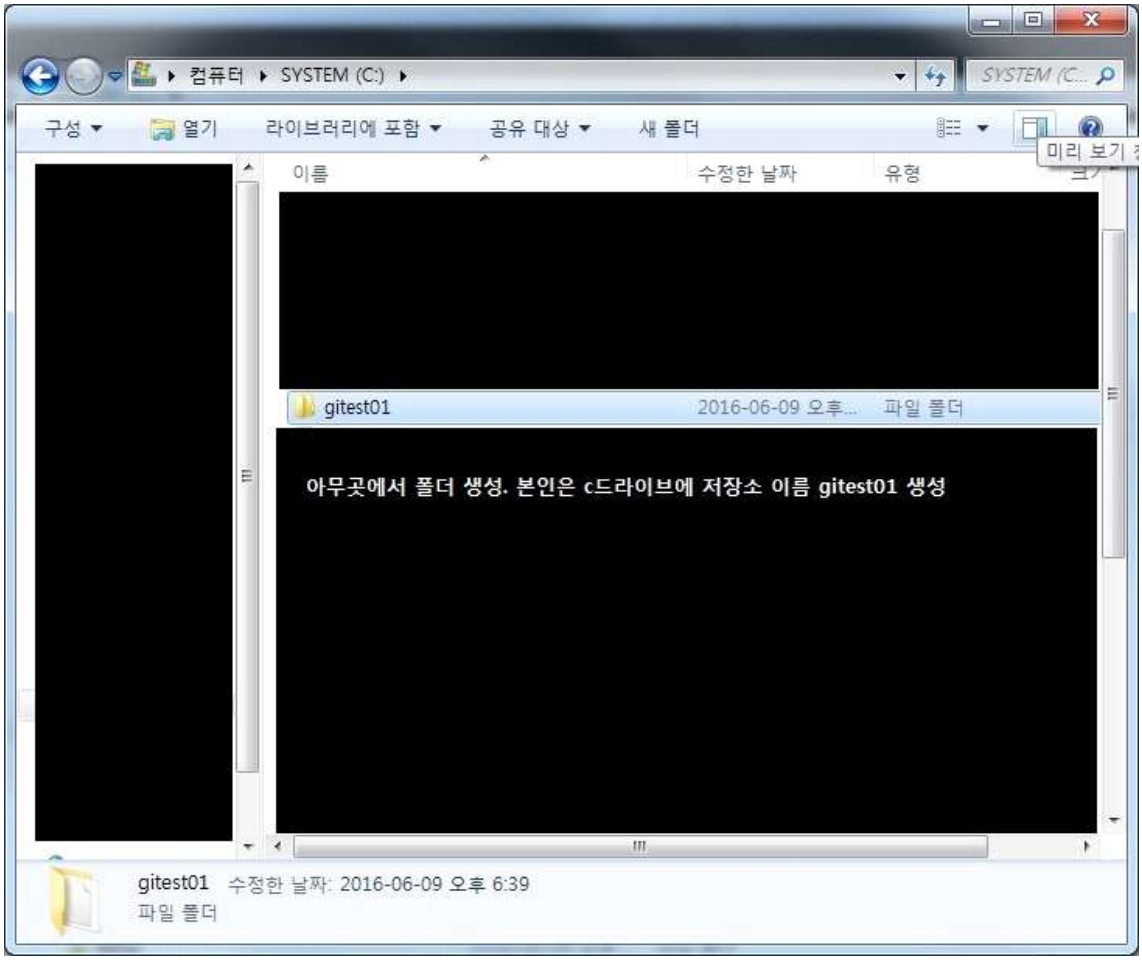


4. 그러면 이렇게 테스트 한번 해보라는 페이지가 뜹니다.



여기서 bash 창을 켭니다.

5. 폴더를 만들어주고 여기에 등록하기 위한 준비



bash 창을 켜서 만든 폴더의 경로를 찾아 들어갑니다.
그다음 여기에 적혀있는 것 중 첫 번째를 입력합니다.
...or create a new repository on the command line
echo 어쩌고 저쩌고 등등등

그렇게 하면 성공

6. 사용자 이름과 메일 주소 입력(경고문고와 임의의 사용자 계정 이름 사용 방지)
- git config --global user.name "사용자이름"
- git config --global user.email 메일주소

7. 저장소 만들기
- mkdir 폴더명
* 초기화
- git init

8. 파일 만들고 link 시키기
* 이름을 날짜로 만든다. (그냥 만든는 것도 가능)
- echo "test md" >> "$(date +%y)$(date +%m)$(date +%d)_$(date +%H)$(date +%M)_"README.md
* 파일 link 상태 확인
- git status
- git add <파일이름> or git add *

- git push -u origin master
(마지막 명령어를 해줘야지 올라감)

9. [깃허브(Github)] 6. Git 프로젝트 세가지 단계, 파일의 라이프 사이클

아직 커밋까진 안했지만 이러저러한걸 조금씩 해봤습니다. 일단 지금까지 해왔던걸 바탕으로 공식사이트에 있는 워킹 디렉토리, Staging Area, Git 디렉토리 등에 대해 조금 살펴보겠습니다.

공식 사이트에 더 자세한 설명과 사용방법 등이 나와 있습니다.

https://git-scm.com/book/ko/v2

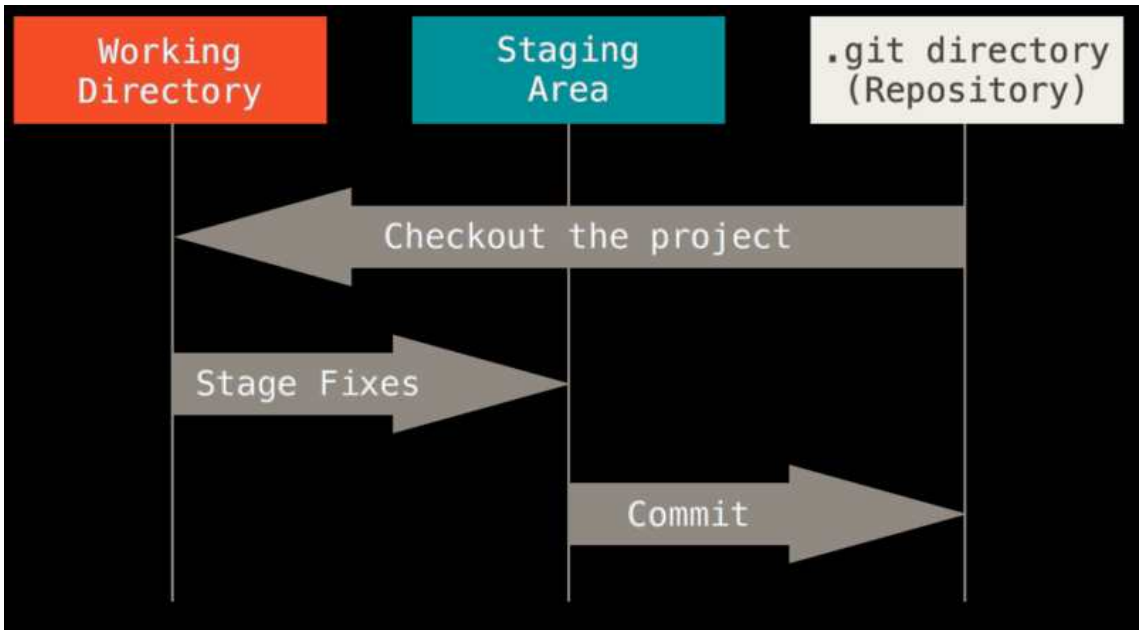
1. 워킹 디렉토리, 스테이징 에리어, 깃 디렉토리

1-1 Working Directory

처음에 저희들이 작업하게 되는 공간은 바로 'Working Directory' 입니다. 아래 그림의 주황색 부분에 해당되죠.
공식 사이트에서의 설명은
'워킹 디렉토리는 프로젝트의 특정 버전을 Checkout 한 것이다. Git 디렉토리는 지금 작업하는 디스크에 있고 그 디렉토리 안에 압축된 데이터베이스에서 파일을 가져와서 워킹 디렉토리를 만든다.'
라고 나와 있습니다.
여기서의 Checkout란 예를 들어 A라는 장소에 있는 것을 B라는 곳으로 복사 하는 것으로 생각하시면 됩니다.
여기서 A에 해당하는건 두가지가 될 수 있습니다.

첫째는 중앙 저장소, 둘째는 로컬 저장소 입니다.
Git은 저장소가 SVN과는 다르게 local에서도 있습니다.
SVN같은 경우는 작업을 하고 나서 다이렉트로 중앙저장소로 넘겨지지만

Git 같은 경우는 local에서 한번 세이브가 되고 이 세이브 된걸 중앙으로 한번 더 세이브 합니다.
(원래 이 의미가 맞을지 모르겠으나 처음 시작하는 사람의 입장에서 이해해본 설명중에 적합하다 판단하였습니다.)
Git 시스템은 장점은 중앙 저장소가 뻥! 터져도 local에 있는 기록을 넘겨주면 설정만을 제외한 기록들이 복구가 됩니다.
그리고 인터넷이 없어도 언제든지 로컬에서 버전관리를 하고 나중에 인터넷이 연결 되면 로컬에서 버전관리된 것을 중앙 저장소로 넘겨주면 됩니다.
아무튼..;;
Working Directory 는 다른 저장소의 파일을 다운 받은 것도 되고..
내가 신규로 뭐 작업을 시작해도 됩니다. 프로젝트의 시작을 여기서 해도 상관 없어요.



1-2 Staging Area

스테이징 에리어 에서는 작업한 내용을 로컬 저장소에 세이브 하기 위해 워킹 디렉토리에서 수정되거나 새로 만들어진 파일을 이 Staging Area로 가져오는 작업을 합니다.
이렇게 가져온 목록들이 바로 5장에서 나왔던 Tracked(관리 대상 파일)를 하려고 하는 겁니다.
바로 그 커밋 이라고 하기 위한 바로 전단계 인것이죠.
commit 란 마무리된 작업에 작업 이력을 기록해서 저장소로 보내는 행위, 즉 Staging Area의 Tracked된 파일들을 저장소에 저장 을 의미합니다. 이건 1-3 Git diractory에서 한번 더 설명하겠습니다.
위의 그림을 보시면 Stage Fixes 이란게 보이실 꺼예요. 이런 행위를 해서 나온걸 '스냅샷' 이라고 합니다.
바로 그걸 4, 5장에서 저희가 git add ~~ 를 통해 해왔던 것입니다. git add ~~~ 은 ~~~ 에 해당하는 파일을 Tracked시키는 것이지. 스냅샷을 만든거야! 라고 생각하시면 이해가 될꺼같

습니다.

1-3 Git directory

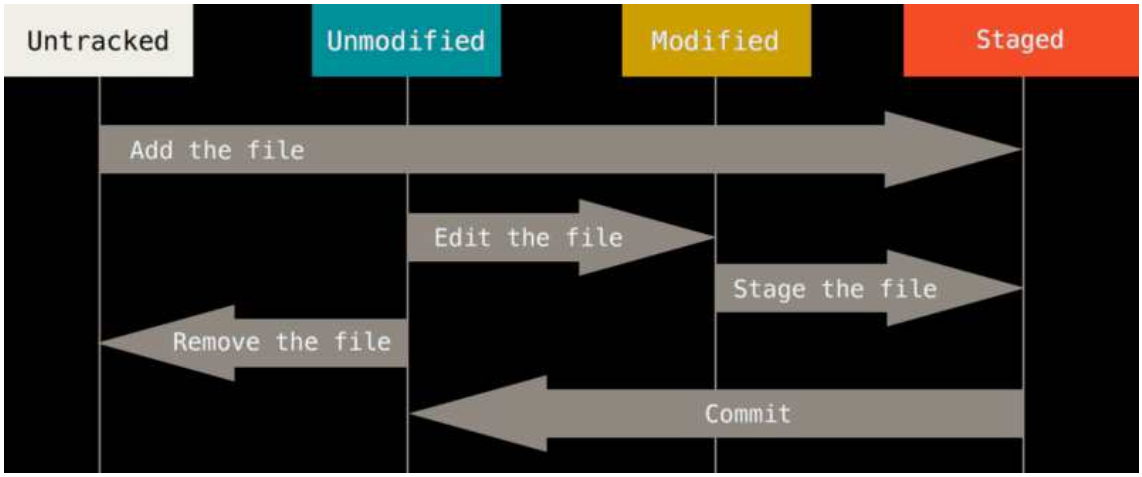
여기서는 commit 란 마무리된 작업에 작업 이력을 기록해서 저장소로 보내는 행위, 즉 Staging Area의 Tracked된 파일들을 저장소에 저장 을 하게 됩니다.
만약 안전하게 잘 저장 됐다면 이게 바로 Committed상태 라고 합니다.
아직 커밋을 안해봤기에 이정도의 설명만 하고 커밋을 해보고 난 후 조금 더 자세한 내용을 보충하겠습니다. 이제 공식 홈페이지에 나온 내용이 조금은 이해되는듯 합니다.
Git으로 하는 일은 기본적으로 아래와 같다.
1. 워킹 디렉토리에서 파일을 수정한다.
2. Staging Area에 파일을 Stage해서 커밋할 스냅샷을 만든다.
3. Staging Area에 있는 파일들을 커밋해서 Git 디렉토리에 영구적인 스냅샷으로 저장한다.

2. 파일의 라이프 사이클

굉장히 중요한 내용입니다. 하지만 공식 홈페이지에 있는 그림은 역시나 정말 알아보기 쉽게 해놓았습니다. 내용을 그대로 복사해서 넣었습니다. 이보다 더 완벽한 설명은 없다고 생각됩니다.

워킹 디렉토리의 모든 파일은 크게 Tracked(관리대상임)와 Untracked(관리대상이 아님)로 나눈다. Tracked 파일은 이미 스냅샷에 포함돼 있던 파일이다. Tracked 파일은 또 Unmodified(수정하지 않음)와 Modified(수정함) 그리고 Staged(커밋으로 저장소에 기록할) 상태 중 하나이다. 그리고 나머지 파일은 모두 Untracked 파일이다. Untracked 파일은 워킹 디렉토리에 있는 파일 중 스냅샷에도 Staging Area에도 포함되지 않은 파일이다. 처음 저장소를 Clone 하면 모든 파일은 Tracked이면서 Unmodified 상태이다. 파일을 Checkout 하고 나서 아무것도 수정하지 않았기 때문에 그렇다.

마지막 커밋 이후 아직 아무것도 수정하지 않은 상태에서 어떤 파일을 수정하면 Git은 그 파일을 Modified 상태로 인식한다. 실제로 커밋을 하기 위해서는 이 수정한 파일을 Staged 상태로 만들고, Staged 상태의 파일을 커밋한다. 이런 라이프사이클을 계속 반복한다.



*참고 사이트
https://git-scm.com/book/ko/v2/Git%EC%9D%98-%EA%B8%B0%EC%B4%88-%EC%88%98%EC%A0%95%ED%95%98%EA%B3%A0-%EC%A0%80%EC%9E%A5%EC%86%8C%EC%97%90-%EC%A0%80%EC%9E%A5%ED%95%98%EA%B8%B0

* 완전 초보를 위한 깃허브
링크: https://nolboo.kim/blog/2013/10/06/github-for-beginner/

2. 기본 용어
이 튜토리얼에서 반복적으로 사용하려고 하는 몇 개의 용어가 있다. 나도 배우기 시작하기 전에는 들어본 적이 없는 것들이다. 여기 중요한 것들이 있다:

커맨트 라인(Command Line): 깃 명령어를 입력할 때 사용하는 컴퓨터 프로그램. 맥에선 터미널이라고 한다. PC에선 기본적인 프로그램이 아니어서 처음엔 깃을 다운로드해야 한다(다음 섹션에서 다룰 것이다). 두 경우 모두 마우스를 사용하는 것이 아닌 프롬프트로 알려진 텍스트 기반 명령어를 입력한다.

저장소(Repository): 프로젝트가 거주(live)할 수 있는 디렉토리나 저장 공간. 깃허브 사용자는 종종 “repo”로 줄여서 사용한다. 당신의 컴퓨터 안의 로컬 폴더가 될 수도 있고, 깃허브나 다른 온라인 호스트의 저장 공간이 될 수도 있다. 저장소 안에 코드 화일, 텍스트 화일, 이미지 화일을 저장하고, 이름붙일 수 있다.

버전관리(Version Control): 기본적으로, 깃이 서비스되도록 고안된 목적. MS 워드 작업할 때, 저장하면 이전 화일 위에 겹쳐쓰거나 여러 버전으로 나누어 저장한다. 깃을 사용하면 그럴 필요가 없다. 프로젝트 히스토리의 모든 시점의 “스냅샷”을 유지하므로, 결코 잃어버리거나 겹쳐쓰지 않을 수 있다.

커밋(Commit): 깃에게 파워를 주는 명령이다. 커밋하면, 그 시점의 당신의 저장소의 “스냅샷”

을 찍어, 프로젝트를 이전의 어떠한 상태로든 재평가하거나 복원할 수 있는 체크포인트를 가질 수 있다.

브랜치(Branch): 여러 명이 하나의 프로젝트에서 깃 없이 작업하는 것이 얼마나 혼란스러울 것인가? 일반적으로, 작업자들은 메인 프로젝트의 브랜치를 따와서(branch off), 자신이 변경하고 싶은 자신만의 버전을 만든다. 작업을 끝낸 후, 프로젝트의 메인 디렉토리인 "master"에 브랜치를 다시 "Merge"한다.

3. 주요 명령어
깃은 리눅스와 같은 큰 프로젝트를 염두에 두고 디자인되었기 때문에, 깃 명령어는 아주 많다. 그러나, 깃의 기본을 사용할 때에는 몇 개의 명령어만 알면된다. 모두 "git"이란 단어로 시작된다.

git init: 깃 저장소를 초기화한다. 저장소나 디렉토리 안에서 이 명령을 실행하기 전까지는 그냥 일반 폴더이다. 이것을 입력한 후에야 추가적인 깃 명령어들을 줄 수 있다.

git config: "configure"의 준말, 처음에 깃을 설정할 때 가장 유용하다.

git help: 명령어를 잊어버렸다? 커맨드 라인에 이걸 타이핑하면 21개의 가장 많이 사용하는 깃 명령어들이 나타난다. 좀 더 자세하게 "git help init"이나 다른 용어를 타이핑하여 특정 깃 명령어를 사용하고 설정하는 법을 이해할 수도 있다.

git status: 저장소 상태를 체크. 어떤 화일이 저장소 안에 있는지, 커밋이 필요한 변경사항이 있는지, 현재 저장소의 어떤 브랜치에서 작업하고 있는지 등을 볼 수 있다.

git add: 이 명령이 저장소에 새 화일들을 추가하진 않는다. 대신, 깃이 새 화일들을 지켜보게 한다. 화일을 추가하면, 깃의 저장소 "스냅샷"에 포함된다.

git commit: 깃의 가장 중요한 명령어. 어떤 변경사항이라도 만든 후, 저장소의 "스냅샷"을 찍기 위해 이것을 입력한다. 보통 "git commit -m "Message hear." 형식으로 사용한다. -m은 명령어의 그 다음 부분을 메시지로 읽어야 한다는 것을 말한다.

git branch: 여러 협업자와 작업하고 자신만의 변경을 원한다? 이 명령어는 새로운 브랜치를 만들고, 자신만의 변경사항과 화일 추가 등의 커밋 타임라인을 만든다. 당신의 제목이 명령어 다음에 온다. 새 브랜치를 "cats"로 부르고 싶으면, git branch cats를 타이핑한다.

git checkout: 글자 그대로, 현재 위치하고 있지 않은 저장소를 "체크아웃"할 수 있다. 이것은 체크하길 원하는 저장소로 옮겨가게 해주는 탐색 명령이다. master 브랜치를 들여다 보고 싶으면, git checkout master를 사용할 수 있고, git checkout cats로 또 다른 브랜치를 들여다 볼 수 있다.

git merge: 브랜치에서 작업을 끝내고, 모든 협업자가 볼 수 있는 master 브랜치로 병합할 수 있다. git merge cats는 “cats” 브랜치에서 만든 모든 변경사항을 master로 추가한다.

git push: 로컬 컴퓨터에서 작업하고 당신의 커밋을 깃허브에서 온라인으로도 볼 수 있기를 원한다면, 이 명령어로 깃허브에 변경사항을 “push”한다.

git pull: 로컬 컴퓨터에서 작업할 때, 작업하고 있는 저장소의 최신 버전을 원하면, 이 명령어로 깃허브로부터 변경사항을 다운로드한다(“pull”).