1. Git 설치

* <https://git-scm.com/>로 가서 다운
* <https://youtu.be/JKT9laOAPIs>를 참조

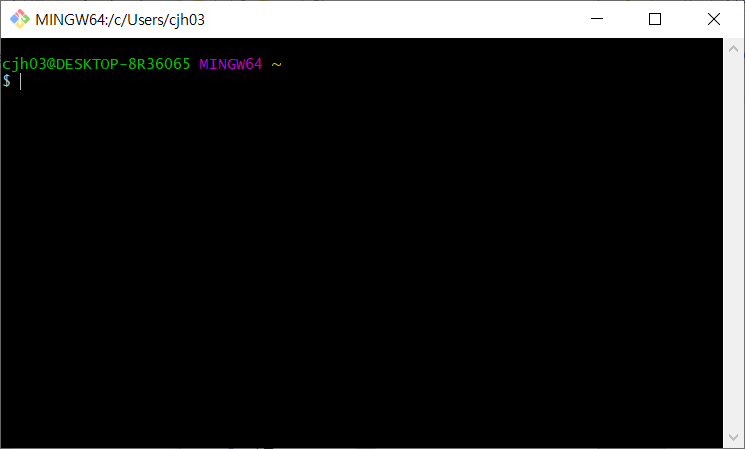
1. Git 저장소 생성

* ***자신이 사용하기 편한 폴더*** [[1]](#footnote-1)생성

1. 단, 안드로이드 프로젝트를 위해 ***경로상*** unicode가 없어야 함 (***한글 x***)

* 윈도우 검색창에서 Git Bash 실행

 <- 아이콘 이렇게 생겼음

 <-초기 실행창

* 자신이 사용할 폴더로 이동 (아까 생성한 폴더[[2]](#footnote-2))

1. Git Bash는 기본적으로 Linux 명령을 사용함

A1. ***cd “폴더경로”*** 로 이동”

* 저장소 생성 방법 2가지

1. ***clone 명령어로 원격 저장소를 내 로컬 저장소에 복사***

a1***.*** 우리의 경우 현재 Toworld repository가 있으므로 a를 사용

a2***. git clone Toworld주소 .*** 을 사용해 Toworld의 저장소를 내 로컬에 복사

a3. 맨 마지막에 **.** 은 현재 디렉토리에 생성하겠다는 의미

a4. Toworld주소: ***https://github.com/kpuce2021/ToWorld.git***

1. init 명령어로 현재 디렉토리에서 git 작업을 진행

b1. ***git init*** 명령으로 현재 디렉토리를 로컬 저장소로 사용

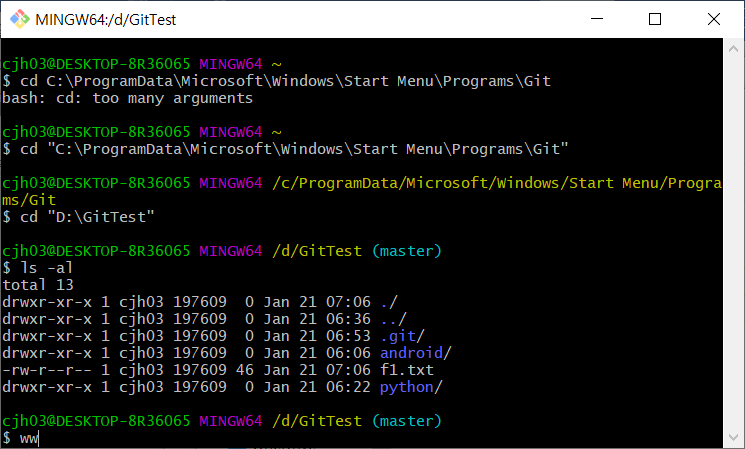
1. 생성 후

C1. ***ls -al*** 명령으로 ***.git*** 이라는 디렉토리가 생성되었는지 확인

C2. ***.git 디렉토리는 버전 관리의 모든 정보가 있으므로 절대 삭제해서는 안 됨***

1. Git을 이용한 파일 버전 관리 개념

* ***우선적으로 Git으로 관리 하고싶은 파일은 모두 Git저장소 내부에 있어야 함***



1. python, android 모두 GitTest라는 저장소 내부에 존재함

* ***파일의 구분 4단계***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Local directory | Status | Repository | Github |
| 주요 키워드 |  | Untracked  Tracked | Commit  log | Push  Pull |

1. Local directory

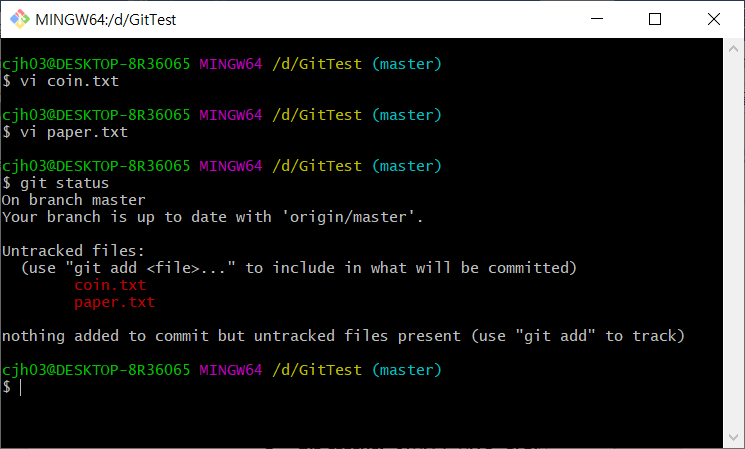
* 윈도우 내의 파일 탐색기로도 볼 수 있는 평범한 폴더
* 여기서 파일을 수정하면 Git Bash에서도 확인 가능

1. Status

* 상태 개념

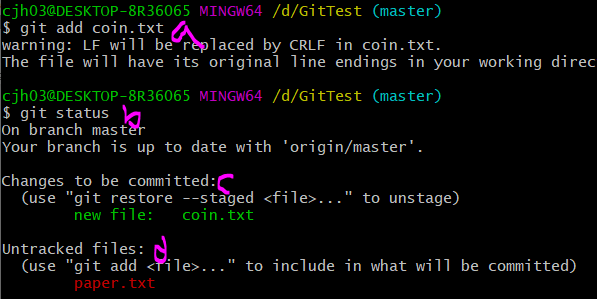
1. Untracked(붉은색): 파일이 Git으로부터 추적되지 않음 = 관리되지 않음
2. Tracked[[3]](#footnote-3)(녹색): 파일이 Git으로부터 추적됨 = 관리됨

* ***git status***: 상태를 확인할 수 있는 명령어

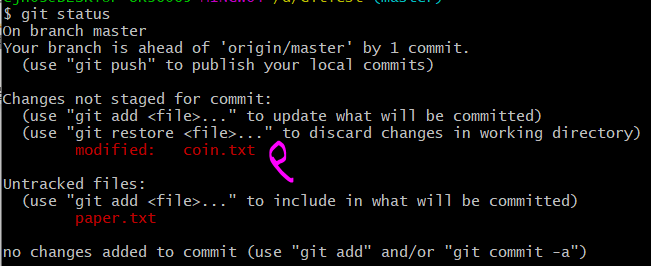


1. coin.txt와 paper.txt 모두 Untracked files: 에 속함
2. 수정된 파일 혹은 처음 생성된 파일은 untracked가 됨

* ***git add*** ***파일명***: untracked 파일을 tracked 파일로 변경하는 명령어



1. git add로 coin.txt를 Tracked파일로 변경
2. git status로 git 로컬 저장소 내의 상태를 확인
3. ***add 된 coin.txt는 tracked = commit 대기 상태***
4. add 되지 않은 paper.txt는 untracked = commit 대기 상태 아님



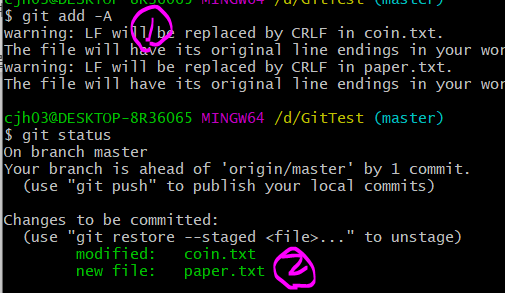
1. add 로 commit 대기 상태를 만들어줘도 수정하면 untracked

* ***결론: commit을 하기 위해선 git add라는 명령으로 파일들을 commit 대기 상태로 만들어줘야 함***
* 시나리오

1. 파일 생성 혹은 변경 -> 무조건 untracked 상태
2. ***git status*** (로컬 저장소 상태 확인)으로 파일 상태 확인
3. ***git add*** (commit 가능 상태로 변경)로 untracked(붉은색) -> tracked(녹색)
4. 이 후 commit -> 추후에 다룰 예정

* 옵션

1. ***git add -A***: 모든 untracked 파일을 add



1. ***git add --help***: 도움말
2. 앞으로 ***명령어 --help***는 그냥 ***도움말***이라고 생각하면 됨

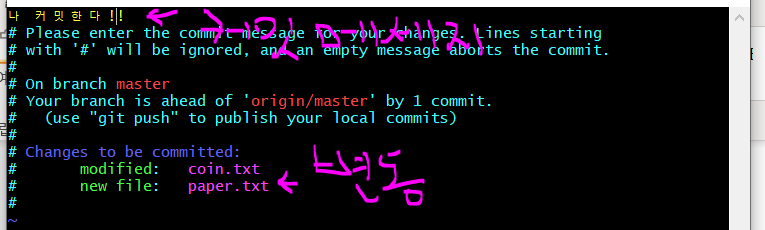
1. commit 이전 “자기 자신이 누구인지 밝히기” = ***한 번만 수행하면 됨***

* ***git config --global user.name 자신의 닉네임***
* ***git config –global user.email 자신의 이메일***
* 위의 두 명령을 사용하면 이후 commit시 닉네임, 이메일이 같이 나옴

1. Repository

* ***git commit***: commit을 실행하는 명령어

1. 실행 시 vi편집기가 실행됨, #은 무시되는 내용, 편집기에 commit 메시지를 [[4]](#footnote-4)작성



1. vi 편집기이기 때문에 저장: w, 종료: q (리눅스 명령)
2. 옵션

***git commit -a***: untracked 파일을 모두 add후 commit

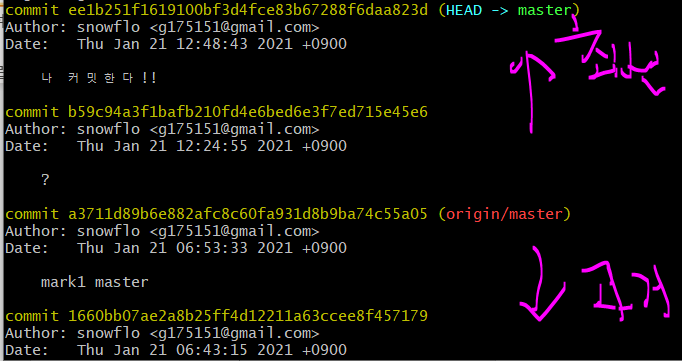
* 주의사항: new file은 add 되지않음
* 위의 그림 new file: paper.txt, modified: coin.txt

***git commit -m “commit 메시지”***: vi 편집기에 들어가지 않고 commit 메시지 작성

***git commit -am “commit 메시지”: 2가지 동시 사용***

***git commit --amend: commit 메시지 변경***

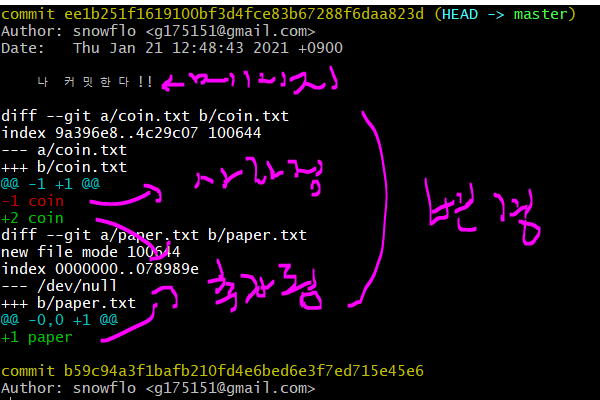
* ***git log***: 로그를 확인하는 명령어



1. 위가 최신 commit, 아래가 과거 commit, ***q로 종료***
2. Author: 작성자, Date: 날짜, commit 메시지로 구성

* git log 옵션

1. git log -p: 소스와 소스상의 차이점 파악 가능



1. ***git diff 최근커밋아이디..과거커밋아이디***: 두 commit 사이의 변동 사항 파악

커밋아이디: commit마다 붙는 고유의 id



1. ***git diff: commit 이전 마지막으로 나의 파일을 검토할 수 있는 기회***

***최근 commit과 나의 파일의 변동 사항 보여줌***

***단, git add 하면 사용할 수 없음***

1. ***git log -reverse: commit을 최신순이 아닌 과거순으로 보는 명령어***

* ***git reset 특정 커밋아이디 --(hard, mixed, soft = 3중 1택, ()는 쓰는거 아님)***

1. ***예시***

***git reset 13lk1h3k13 --hard***

***git reset 13lk1h3k13 --mixed***

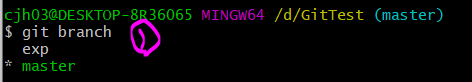
***git reset 13lk1h3k13 --soft***

1. ***hard, mixed, soft 범위***

|  |  |  |
| --- | --- | --- |
| working directory  working tree  working copy  로컬 작업 디렉토리 | Index  Staging area  Cache  커밋 대기 상태 | Repository  History  Tree  커밋 완료 상태 |
|  |  | git reset --soft |
|  | git reset --mixed | |
| git reset --hard | | |

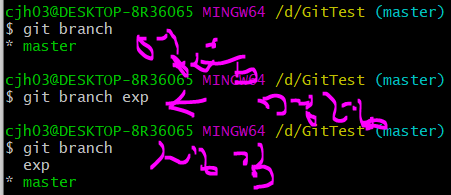
1. ***ORIG\_HEAD: 위험한 명령 – “정보를 잃어버릴 수도 있는”을 기록***
2. ***branch***

* ***git branch: 현재 생성된 branch 확인***



1. ***\*: 현재 checkout된 branch를 의미***

* ***git branch branch이름: branch이름을 가진 branch를 생성***



1. branch는 기존의 branch의 모든 commit 내용을 가지고 생성됨
2. ***현재 위치한 branch를 기점으로 생성됨***

* ***git checkout*** ***branch이름***: branch이름을 가진 branch로 checkout

1. 옵션

***git checkout -b*** ***branch이름***: branch를 생성하며 생성한 branch로 checkout

***git checkout 커밋아이디: 특정 커밋을 가리킴 -> 크게 쓸 일 없음***

* ***git log --branches***: 생성된 모든 branch의 로그를 확인

1. 이렇게만 쓰지는 않음
2. ***git log --branches --decorate --graph***: graph 옵션으로 생성된 branch를 시각적으로 볼 수 있음



1. HEAD는 현재 branch를 의미
2. ***git log --branches --decorate --graph --oneline: b를 한 줄로 보여줌***
3. ***git log 브랜치a..브랜치b: 브랜치a에 없고, 브랜치b에 있는 것을 보여줌***
4. ***git log -p 브랜치a..브랜치b: e에 추가로 코드까지 보여줌***
5. ***git diff 브랜치a..브랜치b: 현재 브랜치a와 브랜치b의 상태를 비교***

* ***여담***

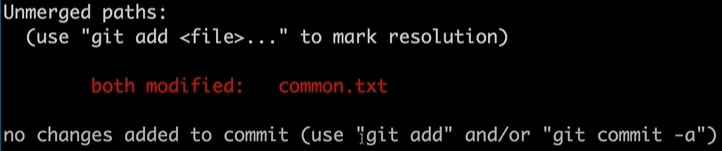
1. Branch를 변경하면 window내의 보이는 파일도 변경됨 (로컬 저장소 파일 탐색기)
2. ***Source tree라는 git을 그래프로 표현해주는 tool이 존재***
3. ***Branch 병합***

* 모든 병합은 기존의 branch로 병합

1. ***git merge 병합할branch***: branch 병합
2. ***git branch -d branch이름***: branch 삭제 -> 깔끔한 branch 관리가 가능함

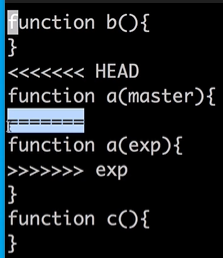
* ***conflict***

1. git merge를 시도했을 때, ***automatic merge failed 메시지와 함께 충돌*** 알림
2. ***git status***로 어떤 상황인지 파악 가능



이 경우는 2개의 branch에서 동시에 수정되어 충돌이 발생한 케이스

1. vi 편집기를 이용해 충돌이 발생한 파일을 염



***========: 경계선, 위: 현재 branch, 아래: 병합할 branch***

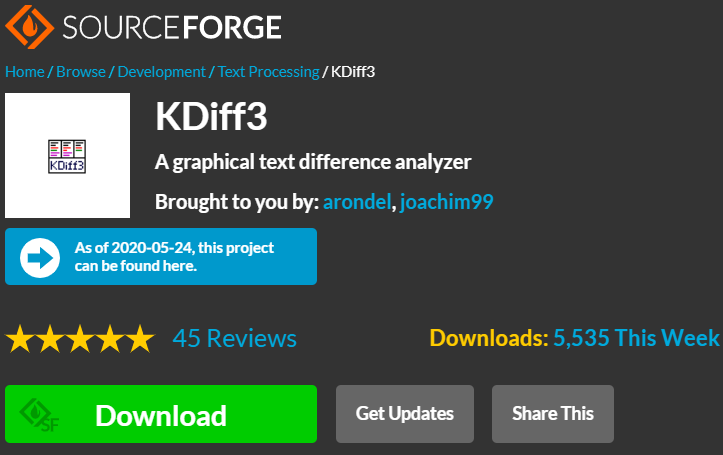
***직접 파일을 열어 판단하고 수정***

* ***mergetool을 이용한 conflict 해결 (kdiff3기준) -> chapter10***

1. ***kdiff3을 이용한 conflict 해결***

* kdiff3 설치: windows10 기준

1. <https://sourceforge.net/projects/kdiff3/> 접속



Download 클릭

1. 설치파일이 모두 다운되면 설치, default로 하되 ***중간에 user 물어보면 current로***

***+ 설치 시 파일경로 무조건 기억할 것 -> 나중에 필요함***

1. ***오류방지를 위한 Git Bash 내부 설정 -> 각각 따로따로 6줄 모두 칠 것***

git config --global --add merge.tool kdiff3

git config --global --add mergetool.kdiff3.path "***kdiff3.exe 경로***"

git config --global --add mergetool.kdiff3.trustExitCode false

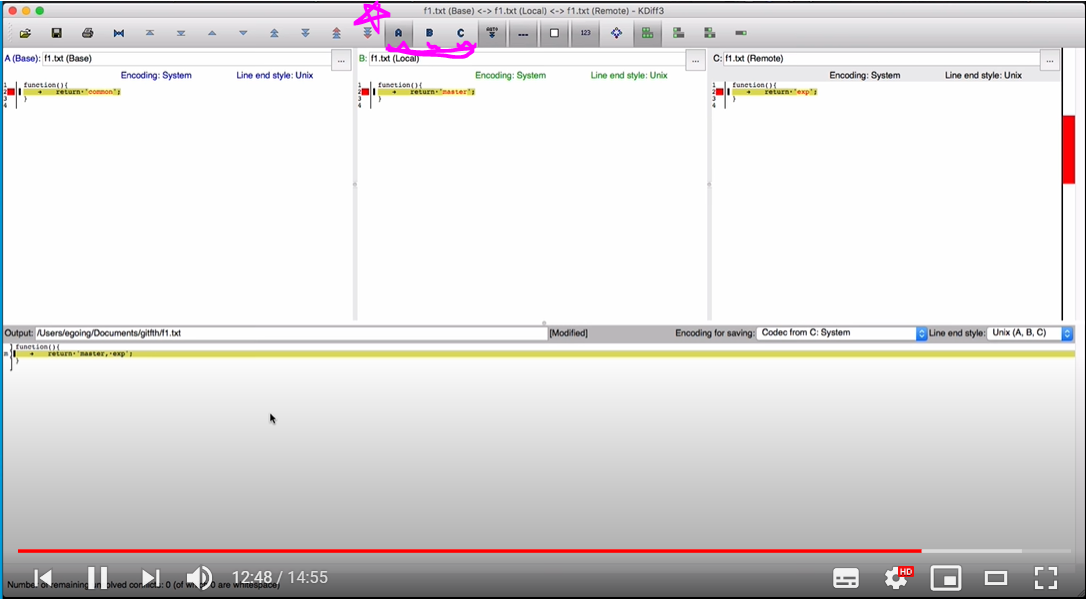
git config --global --add diff.guitool kdiff3

git config --global --add difftool.kdiff3.path " ***kdiff3.exe 경로*** "

git config --global --add difftool.kdiff3.trustExitCode false

* ***사용***

1. ***git merge 병합할branch를 실행 -> conflict 발생했을 경우***
2. ***git mergetool라는 명령 실행***



A: base – 기존 commit, B: Local – 현재 위치, C: Remote – 합치려 하는 branch

Output: 결과 -> ABC를 클릭하고 Output을 수정하여 원하는 결과로 바꾸고

Save & quit 할 시 알아서 merge 완료됨

* ***참조링크***

1. <https://sourceforge.net/projects/kdiff3/>
2. <https://opentutorials.org/course/2708/15306>
3. stash

* ***branch에서 작업중 작업이 끝나지 않았음에도 불구하고 checkout을 해야 하는 상황에 사용***
* 작업이 끝나지 않아 ***commit을 하기에는 애매함***. 하지만 ***commit을 하지 않으면 checkout이 불가능***
* ***git stash: 현재 branch의 working directory와 index에 저장 -> 파일을 숨김***

1. ***단 이 기능은 add된 파일에만 사용이 가능***

* ***git stash apply: stash로 숨겨둔 파일을 다시 불러옴***
* ***git stash list: stash된 파일의 목록을 보여줌***
* ***git stash drop: 가장 최근 stash 내역 삭제***
* ***git pop: git stash apply + git stash drop***

1. 원격 저장소 (github)

* 로컬 저장소와 연결되어 동기화되는 저장소
* 2가지 경우

1. ***1. 원격 저장소를 만들고 복제해서 로컬 저장소를 만들어 로컬 저장소에서 작업***

***우리는 toworld repository가 있어서 이 방법을 사용***

**앞에서 설명한 *git clone github주소 .* 명령을 사용**

1. 원격 저장소의 ***이름이 origin***으로 자동 설정되며 연결
2. 2. 로컬 저장소에서 작업하다가, 원격 저장소로 작업 내용 올림

* ***git push***: 원격 저장소로 데이터 업로드

1. ***git push -u 저장소명(origin) master: commit을 master branch에 push***
2. ***처음에 a처럼 하면 그 이후에는 git push로 사용가능***
3. ***단 동일 branch에서만 가능***
4. ***마찬가지로 github에 branch를 생성하고 싶으면 git push -u 저장소명 branch명***
5. ***역시 동일 branch에서는 git push로 사용가능***

* ***git pull***: 원격 저장소에서 데이터 다운로드
* ***기본적인 규칙***

1. ***작업 전: git pull***

***변동 사항이 있을 수 있으니***

1. ***작업 후: git commit 후에 git push***

***로컬 저장소에 먼저 저장 후 원격 저장소로***

* ***Github내 Conflict의 발생***

1. ***기본적으로 conflict가 발생하면 kdiff3이 설치되었다는 가정하에***

***git mergetool 명령 실행***

1. ***git pull***시 발생하는 conflict

내가 기존에 가지고 있는 파일과 다른 사람이 작업해 github에 push한 내용이 다를 경우가 태반일 것임 -> 따라서 항상 작업 전 git pull을 함으로써 최신화

1. ***git push***시 발생하는 conflict

내가 push 하려는 파일과 github에 있는 파일의 내용이 다를 경우에 발생할 확률이 높음 -> git pull을 함으로써 github의 가장 최신 정보를 가져오고: 이 때 conflict가 발생하면 b의 경우로 해결 -> github에서 가져온 최신 정보와 내가 push 하려는 정보의 conflict를 해결하고 commit후 push

1. ***다른 경우: 아직 잘 모르겠음. 해봐야 암***

***원격 저장소 branch 확인 명령***

* ***git branch -r***

***HEAD -> 지금 가리키는 곳***

***나머지: 목록임***

* ***github branch 받아오기***

***git checkout 브랜치명***

***브랜치명 예시: origin/master***

* ***git remote update: 깃 원격저장소 업데이트***
* ***git remote remove 저장소명 e.g origin***

1. Git저장소가 될 폴더 [↑](#footnote-ref-1)
2. Git저장소가 될 폴더 [↑](#footnote-ref-2)
3. Tracked: 사실상 commit 대기 상태, commit 되지는 않았지만 commit이 가능함 [↑](#footnote-ref-3)
4. commit 메시지: commit할 때 변동사항 같은 것을 쓰는 것 [↑](#footnote-ref-4)