커밋 - 변경된 내용을 저장

readme - 프로젝트 내용 설명하는 파일이라고 생각하기.

branch - 분기

fork - 원본을 복사해감

pull request - fork로 복사해간 사람이 수정해서 pull request로 알리면 원주인이 그걸 받아볼 수 있음

issue - 버그, 오류등을 알리고 소통하는 일종의 소규모 커뮤니티

wiki - 프로젝트를 위해 알아야 하는 지식이나 메뉴얼 등을 저장

graph - 저장소와 관련된 정보를 시각적으로 표시

watch - 팔로우 비슷한 기능

.ignore - 해당 파일들은 업로드 되지 않는다.

push - 원격 저장소에 저장한다.

pull - 원격 저장소에서 가져온다.

---git 명령어---

git config --global init.defaultBranch main

github에서 master를 사용하지 않기 때문에 main으로 기본설정을 바꾸어 주어야 한다.

git init . - 현재 위치 버전관리 시작(현재위치 .)

repository - 저장소 버전별로 저장되어있다.

working tree - 버전으로 만들어지기 전 단계

staging area - 버전으로 만들려고 하는 파일들

파일 생성 nano 파일명.확장자

git status - 현재 상태

git add 파일명 - working tree에서 staging area로 올린다.

git add . - 현 디렉토리 이하의 모든 파일을 add한다.

git commit -m "설명" - staging area에서 저장소로 커밋된다.

git commit -am "설명" - add 와 commit을 한번에 수행 //모든 파일들이 최초한번은 add를 했어야함

git log - 로그보기

git log --stat -관련된 파일 리스트 보기

git log -p 파일간의 차이를 보여주는 로그

git log --all - 모든 branch를 보여줌

git log --graph - 시각적으로 표현

git log --oneline - 버전이 한줄로 나옴

git diff - 마지막 버전과 workig tree 사이의 차이점을 보여줌

git reset --hard - 수정내용삭제/취소 -> 마지막 버전으로 되돌림.

git reset --hard commit id - 해당 버전으로 리셋된다(해당버전이 되고 그 이후버전은 제거된다. 해당 버전이 마스터가 된다)

log에서 복사한 commit id

git checkout commit id - 해당 커밋 버전으로 돌아감.(head를 옮김)

git checkout master - 최신버전으로 되돌아감

git revert commit id - 해당 버전의 수정사항을 취소하여 커밋함.(해당 버전은 보존함.)

버전 1,2,3,4(최신) 에서 git revert commit id4 하면 4의 수정사항이 되돌아간 3의 상태로 가되, 4의 버전은 유지한다.

단,2나 1의 버전으로 가기 위해서 역순으로 4,3,2를 revert해야한다. "해당id"의 수정사항만 되돌리기 때문에 4상태에서 2를 revert하면 3,4의 수정사항이 충돌이 일어난다.

diff tool - 차이 더 잘 보고싶은 경우 찾아보기

.gitignore - 버전관리 하기 싫은 경우 알아보기

tag - commit id 대신 사용 가능

---

git commit --amend -commit한 메세지 수정

git branch - 현재 모든 branch보여줌

git branch 이름 - "이름" branch 생성

git checkout branch이름 - head를 해당 branch로 이동

merge하려는 두 커밋의 공통조상 base

merge된 커밋 - merge commit

c1의 내용에 c2를 병합하려면 c1에서 병합 수행

git merge c2

다른파일은 복사되고

같은 이름의 파일은 다른 부분은 합쳐지고 같은 부분이 수정된 경우 수동으로 수정할것을 알림. - 충돌

git status 로 상태 확인 가능하고

nano로 열어보면 >>>> 와 <<<<사이에 내용만 수정하면 된다. (수정후 꺽쇠 부분은 지워주어야 함.)

이후 add,commit해주어야 함

3way merge

base를 기준으로 한쪽이 수정된 부분은 수정된 내용으로 합병. 양쪽이 수정된 경우 충돌이 난다.

공통조상을 같는 두 commit에 대해 merge시 자동 수행됨.

git mergetool - 병합시 사용 할 수 있는 툴을 사용함.

프로그램 종료시 이전 내용을 .orig로 저장해주고 병합한 내용을 add까지 해줌.

cherry-pick - 병합시 이전 버전의 것을 골라서 병합/알아보기

rebase - 병합과 동시에 부모를 합침/어려움 알아만두기/알아보기

원격 저장소에 지역 저장소를 연결

git remote add origin https://github.com/qkrtndh/my-repo.git(저장소 주소)

git remote - 연결 저장소확인

git push 하면 원격저장소로 저장됨.

처음 연결하는 경우 git push --set-upstream origin master 뜨는데 그냥 복붙해주면 됨.

기본저장소 origin에 master를 연결하겠다는 의미정도.

깃헙 아이디 비번 넣고 새로고침

원격 저장소로부터 복사해오기

git clone https://github.com/qkrtndh/my-repo.git

현 위치에 해당 저장소를 my-repo(폴더)생성

git pull 하면 원격 저장소로부터 다운받아옴

pull

작업

add,commit,push

협업자를 자신의 github colaborator에 초대해야 함.

수락측은 메일에서 수락해서 참여

충돌 해결 후 add 하기

기존

git clone / git pull -> 작업 -> add -> commit -> push

추천

git clone / git fetch -> git merge FETCH_HEAD(origin/main) -> 작업 -> add -> commit -> push

git pull = git fetch + git merge FETCH_HEAD(origin/main)

git fetch하면 원격 저장소를 업데이트하고

FETCH_HAED라는 곳에 정보를 저장.

git merge FETCH_HEAD는 알아서 합쳐준다고 생각하면 된다.

Issue 발급하고 해결해서 closing하면 안보임 – 필터로 볼 수 있긴함.

Reopen issue 하면 다시 열 수 있음

Assignees 하면 협업자에게 이슈 할당 가능

Labels는 해당 issue의 속성을 나타냄(어떤 종류인지)

@하면 협업자들 언급가능 – 본인이 언급된 이슈 필터 가능

#하면 다른 이슈들 나옴 (중복 이슈등에 사용), 이슈 닫을 수 있음

다른 중복된 쪽에 언급됨.

커밋 아이디를 붙여넣으면 자동으로 링크가 걸린다.

Issue_template.md 라는 파일을 깃 허브 내에 만들어 두면

해당 저장소의 이슈들을 생성할 때 기본적인 템플릿을 지정할 수 있음.

Github page를 통해 깃허브상에서 관리중인 페이지 소스를 실제 페이지로 만들어 여러가지 목적으로 사용할 수 있다. (일종의 무료 도메인처럼 사용할 수 있다)