git 초기 설정

BASH

```
🔰 # 현재 위치에서 지역 저장소를 생성
$ git init
🔰 # 깃 환경에서 사용자 이름을 [사용자명]으로 지정
$ git config --global user.name "[사용자명]"
🔰 # 깃 환경에서 사용자 이메일을 [사용자이메일명]으로 지정
$ git config --global user.email "[사용자이메일명]"
```

git 스테이징

BASH

```
🔰 # [파일명.확장자명]을 스테이지에 올림
$ git add [파일명.확장자명]
🔰 # 상태를 확인
$ git status
🔰 # 파일 add 취소
$ git restore --staged 파일명
🔰 # 전체 add 취소
$ git reset HEAD
```

git 커밋

BASH

```
🔰 # 커밋 메시지 [메시지명]을 붙여 커밋
$ git commit -m "[메시지명]"
$ git commit -am "[메시지명]" # 스테이징과 커밋을 동시에 진행
🔰 # 최근 순서대로 커밋을 취소
$ git reset HEAD^ # 현재 HEAD의 이전 커밋으로 되돌리기
$ git reset HEAD~n # 현재로 부터 n 번째 이전 커밋으로 되돌리기
🔰 # 롤백할 커밋을 지정
$ git reset [커밋 해시]
# - reset의 3가지 옵션
$ git reset --soft [커밋ID]  # head 만 바뀜
$ git reset --mixed [커밋ID] # staging 도 그 때로 바뀜
$ git reset --hard [커밋ID] # working디렉토리/staging 모두 그 때로 바꿈
🔰 # 커밋을 삭제하지않고 이전 커밋을 새로 복사 추가하는 식으로 롤백
$ git revert [커밋 해시] # 해당 커밋 이전상태로 되돌린다라는 명령
🔰 # 커밋 수정하는 법
# ... 파일 수정 한 뒤
$ git add .
$ git commit --amend # 최신 커밋 수정
```

git 이력 확인

BASH

```
🔰 # 커밋 내역 확인
$ git log           # 전체 이력을 보여준다.
$ git shortlog
$ git log --oneline # 커밋과 커밋 메시지를 조회
$ git log -p        # 이력을 보여줄 때, 변경된 데이터도 보여준다.
$ git log [filename] # 특정 파일에 대한 이력을 보여준다.
$ git log -p [filename] # 특정 파일의 이력과 변경된 데이터를 보여준다.
$ git log --stat    # 커밋마다 파일의 추가/삭제된 통계데이터를 보여줌
$ git log --graph   # 브랜치 분기 및 병합내용을 아스키 그래프로 보여줌
🔰 # 특정 커밋 내역 확인
$ git show [커밋 id]
🔰 # 최근 버전과 작업 폴더의 수정 파일 사이의 차이를 출력
$ git diff              # modified 상태의 file과 마지막 commit 비교
$ git diff --staged     # staged 상태의 file과 마지막 commit 비교
$ git diff --color-words # 변경사항을 color 풀하게 비교
$ git diff [브랜치이름]    # 현재브랜치와 선택된 브랜치와 차이점 비교
$ git diff [이전커밋 id] [이후커밋 id] # 커밋 비교
```

git log 명령어 예시	설명
git log	HEAD와 관련된 commit들이 자세하게 나옴
git log --oneline	간단히 commit 해시와 제목만 보고 싶을 때
git log --oneline --graph --decorate	HEAD와 관련된 commit들을 조금 더 자세히 보고 싶을 때
git log --oneline --graph --all --decorate	모든 branch들을 보고 싶을 때 사용하는 명령어
git log --oneline -n7	내 branch의 최신 commit을 7개만 보고 싶을 때 사용

원격 저장소

BASH

```
🔰 # 원격 저장소에 연결
$ git remote add origin [github 레포지 주소]
🔰 # 옵션 종류 보기
$ git remote --help
🔰 # 추가한 원격저장소의 목록 확인
$ git remote
$ git remote -v # 상세히
🔰 # 특정 원격 저장소의 정보를 확인할 수 있다.
$ git remote show 이름
🔰 # 원격저장소 이름 변경
$ git remote rename 기존이름 변경할이름
🔰 # 원격저장소를 제거
$ git remote rm 이름
```

BASH

```
🔰 # 지역 저장소의 커밋을 맨 처음 원격 저장소에 올리는 경우
$ git push -u origin master
🔰 # -u로 등록한 후에 지역 저장소의 커밋을 원격 저장소에 올리는 경우(업로드)
$ git push
$ git push origin master
🔰 # 원격 저장소의 커밋을 지역 저장소로 가져옴
$ git pull origin master
🔰 # 원격 저장소 복제하기
$ git clone [원격 저장소 주소]
🔰 # 원격 저장소의 커밋을 가져오기만 하고 merge하지 않는다
```

```
$ git fetch
# 이후엔 diff 로 비교
$ git diff test origin/test # 브랜치 이름이 test일 경우 예시
🔰 # 원격저장소 삭제(끊기)
$ git remote remove origin
```

저장소 파일 삭제

BASH

```
🔰 # 로컬저장소 원격저장소 둘다 파일 삭제
$ git rm [filename]
🔰 # 원격저장소 파일만 삭제. 로컬저장소 파일은 냅둠
$ git rm --cached [filename]
```

git branch

BASH

```
🔰 # 브랜치 확인
$ git branch
🔰 # 브랜치 추가 / 삭제
$ git branch [branch name]
$ git branch -d [branch name] # -delete
🔰 # 브랜치로 이동
$ git switch [branch name]
$ git checkout [branch name]
🔰 # 브랜치 추가하고 바로 이동
$ git switch -c [branch name]
$ git checkout -b [branch name]
🔰 # 현재 브랜치에서 다른 브랜치를 merge
$ git merge [branch name]
```

git cherry-pick

BASH

```
🔰 # 만일 내가 X 브랜치에 있고
   # 브랜치 Y의 커밋 중 76ae30ef와 13af32cc만 골라 현재 브랜치인 X에 적용하고 싶을때
$ git cherry-pick 76ae30ef 13af32cc
```

git stash

BASH

```
🔰 # staged, modified 상태의 파일을 저장
$ git stash
🔰 # stash로 저장한 리스트를 출력
```

```
$ git stash list
🔰 # 가장 최근에 저장한 stash를 반영
$ git stash apply
$ git stash apply stash@{1}  # 지정한 stash를 반영
🔰 # 지정한 stash를 삭제
$ git stash drop stash@{1}
🔰 # stash를 반영하고 자동 삭제
$ git stash pop
```

git tag

- light weight 태그 : 단순 커밋 태그용. 커밋을 바로 가리킴
- annotated 태그 : 태그 메세지도 첨부할수 있어 자체 해시값을 가짐. ^{}이 커밋을 가리킴

BASH

```
🔰 # 태그 추가
$ git tag [태그이름]
$ git tag [태그이름] [커밋번호]  # 특정 해쉬에 태그 추가
$ git tag -a [태그이름] [커밋번호] # 특정 해쉬에 annotated 태그 추가
🔰 # 태그 조회
$ git tag
$ git tag -l 'v1.*'  # 와일드카드 패턴으로 검색조회
$ git show [태그이름]  # 특정 태그 조회
$ git show-ref --tags  # 해쉬값과 태그 조회
```

git blame

BASH

```
# git 프로젝트에서 어떤 코드를 누가 수정했는지, 어떤 commit으로 수정이 되었는지 궁금할 때
# 🔰 해당 파일에 대한 모든 수정 내역
$ git blame <파일명>
# 🔰 author name과 timestamp는 출력하지 않음.
$ git blame -s <파일명>
# 🔰 description과 수정사항을 보여줌
$ git show <커밋번호>
# 🔰 파일내용을 start부터 end 라인까지의 수정 내역만 보여줌
$ git blame -L <start,end> <파일명>
```