**버전 관리의 본질**

**1. 저장소 만들기**

- 디렉토리 생성 -> 디렉토리 이동 -> 현 디렉토리를 git의 저장소로 지정

* git

- git에서 사용할 수 있는 명령어를 모두 볼 수 있음

* 디렉토리 생성

- **mkdir** + 폴더명

* 디렉토리 이동

- **cd** + 이동하고자 하는 폴더명

* 현 디렉토리를 git의 저장소로 지정

**- git init**

**2. git이 관리할 대상으로 파일 등록**

- f1.txt 파일을 생성하고 파일 내에 내용 삽입

- 파일 생성 -> 파일 추적 -> 상태 확인

* 파일 생성 : **vim f1.txt**

- vim이라는 프로그램으로 f1.txt 파일을 편집/생성

- 파일 생성 후 **‘i'** 입력 -> 입력 모드로 전환 -> 입력 완료 후 **‘esc’**

- **:wq** => 파일 저장(w) 후 종료(q)

- **cat** f1.txt or **vim** f1.txt => 파일 내용을 볼 수 있음

* 파일 추적 : **git add f1.txt**

- git에게 버전관리를 명령하지 않은 파일은 무시함

- **새로 추가한 파일은 'add' 명령을 통해 해당 파일을 추적**하도록 해야 O

-> 프로젝트의 핵심적인 파일과 임시적인 파일을 관리할 때 보다 편리하기 위해 필요

- 이미 버전 관리가 되고 있는 **파일을 수정한 후에도 add 명령 필수**

* 프로젝트 폴더의 상태 확인

- **git status**

**3. 버전 만들기 (commit)**

- 버전 : 의미 있는 변화 (작업이 완결된 상태)

* git을 처음 사용한다면 버전을 만들기 전에 **이름을 세팅**해야 함

**1) git config --global user.name + 닉네임**

**2) git config --global user.email + 이메일**

- 다른 사용자가 봤을 때 누가 작업했는지 파악 가능

* **git commit**

- 버전 생성

* **version message (commit message)**

- 파일의 변경 내용을 작성

* **git log**

- 생성된 버전들 확인 ('q' -> 종료)

**4. git stage area**

- **cp** + 복제하고자 하는 파일명 + 복제한 파일의 파일명

* **commit 전에 add 명령을 실행해야 하는 이유**

- **add를 완료한 파일만이 commit 명령 가능** -> 선택적으로 파일 관리가 가능

* git은 **stage**와 **repository**로 분류

- stage : 수정 된 파일이 commit을 기다리는 장소 (이미 add 된 파일)

- repository : stage에 존재하던 파일들이 commit 되어 저장되는 장소

* git add => commit 대기 상태 (이를 stage area라고 함)
* git commit => commit 대기 상태에 있는 파일만 commit

**5. 변경사항 확인하기**

* **git log -p** (모든 로그)

- **각 commit 사이의 소스 차이점**을 확인할 수 있음

- /dev/null : 이전 버전에서 파일이 없었다는 의미

* **git diff '버전 id1'..'버전 id2'**

- 커밋은 각자 고유 ID가 있음

- **버전 간의 차이점을 비교**할 때 사용

* **git diff**

- **git add 하기 전과 후의 파일 내용을 비교**할 때 사용

- commit 이전에 **소스를 마지막으로 검토**할 수 있음

**6. 과거의 버전으로 돌아가기**

* **reset**

**- git reset '버전 id' --hard**

- 돌아가고자 하는 버전으로 돌아가는 명령

- 필요하면 복구 가능 => but, git의 원리를 알아야 함

**- 버전 공유 이후에는 reset 금지**

* **revert**

**- git revert '버전 id'**

- 버전 id를 취소한 내용을 새로운 버전으로 만드는 명령

**- 이미 push 한 상태에서 사용**

* reset과 revert의 차이

- a - b - c - d가 있을 때,

1) git reset c --hard

=> a - b - c

=> c까지 남고, 이후는 취소

2) git revert c

=> a - b - c - d - Revert 'c'

=> c만 취소하고, c가 취소된 이력이 남음

**7. 명령의 빈도와 메뉴얼 보는 방법**

* **git commit --help** : commit messege에 대한 도움말
* **git commit -a** : add 없이도 commit 가능
* **git commit -m "커밋메시지"** : 에디터를 켜지 않고 커밋 메시지 작성
* **git commit -am "커밋메시지"** : 위의 명령어를 한 번에 사용
* **git commit --amend** : 커밋메시지 수정 – push 이후에는 사용하지 X