작성일시	@2022년 10월 28일 오전 9:22
강의날짜	
편집일시	@2022년 10월 28일 오후 4:39
분야	
공부유형	
복습	☐
태그	

Git undoing

개요

- Working Directory 작업 단계
  - working directory에서 수정한 파일 내용을 이전 커밋 상태로 되돌리기
  - `git restore`
- Staging Area 작업 단계
  - Staging Area에 반영된 파일을 Working Directory로 되돌리기
  - `git rm –cached`
  - `git restore –staged`
- Repository 작업 단계
  - 커밋을 완료한 파일을 Staging Area로 되돌리기
  - `git commit –amend`

working directory 작업 단계 되돌리기

git restore

- Working Directory에서 수정한 파일을 수정 전(직전 커밋)으로 되돌리기
- 이미 버전 관리가 되고 있는 파일만 되돌리기 가능

- **git restore를 통해 되돌리면, 해당 내용을 복원할 수 없으니 주의할 것**
- `git restore {파일 이름}`

Staging Area 작업 단계 되돌리기

개요

- Staging Area에서 반영된 파일을 Working Directory로 되돌리기 (==Unstage)
- root-commit 여부에 따라 두 가지 명령어로 나뉨
  - root-commit이 없는 경우 : `git rm --cached`
  - root-commit이 있는 경우 : `git restore --staged`

`git restore --staged`

- **the contents are restored from HEAD**
- root-commit이 있는 경우 사용(Git 저장소에 한 개 이상의 커밋이 있는 경우)
- `git restore --staged {파일 이름}`

`git commit –amend`

- 커밋을 완료한 파일을 Staging Area로 되돌리기
- 상황 별로 두 가지 기능으로 나뉨
  - Staging Area에 새로 올라온 내용이 없다면, **직전 커밋의 메시지만 수정**
  - Staging Area에 새로 올라온 내용이 있다면, **직전 커밋을 덮어쓰기**
- **amend(수정하다) 즉, 이전 커밋을 수정해서 새 커밋으로 남김**
- **커밋 내용을 수정하거나 수정 사항을 새로 커밋에 추가하고 싶을 때 사용**
- **수정 사항을 반영하기 위해 새로운 커밋을 생성하지 않아도 됨**

Git reset

개요

- 시계를 마치 과거로 돌리는 듯한 행위로, 프로젝트를 특정 커밋(버전) 상태로 되돌림
- 특정 커밋으로 되돌아 갔을 때, **해당 커밋 이후로 쌓았던 커밋들은 전부 사라짐**
- `git reset [옵션] {커밋 ID}`
  - 옵션은 soft, mixed, hard 중 하나를 작성

    - 커밋 ID는 되돌아가고 싶은 시점의 커밋 ID를 생성

Git revert

- 과거를 없었던 일로 만드는 행위, 이전 커밋을 취소한다는 새로운 커밋을 생성함
- `gir revert {커밋 ID}`
    - 커밋 ID는 취소하고 싶은 커밋 ID를 작성

Git Branch

- “Branch는 Commit을 가리키는 Pointer임”

장점

- 브랜치는 독립 공간을 형성하기 때문에 원본(master)에 대해 안전함
- 하나의 작업은 하나의 브랜치로 나누어 진행되므로 체계적인 개발이 가능함
- Git은 브랜치를 만드는 속도가 굉장히 빠르고, 적은 용량을 소모함

git branch

- 브랜치의 조회, 생성, 삭제와 관련된 git 명령어
- 조회
    - `git branch`
        - 로컬 저장소의 브랜치 목록 확인
    - `git branch -r`
        - 원격 저장소의 브랜치 목록 확인
- 생성
    - `git branch {브랜치 이름}`
        - 새로운 브랜치 생성
            - 이름은 보통 `git branch feature/signup` 과 같이 작성
    - `git branch {브랜치 이름} {커밋 ID}`
        - 특정 커밋 기준으로 브랜치 생성
- 삭제

- `git branch -d {브랜치 이름}`
  - 병합된 브랜치만 삭제 가능
- `git branch -D {브랜치 이름}`
  - 강제 삭제

Git switch

- 현재 브랜치에서 다른 브랜치로 이동하는 명령어
- `git switch {브랜치 이름}`
- `git switch -c {브랜치 이름}`
- `git switch -c {브랜치 이름} {커밋 ID}`
- switch하기 전에, 해당 브랜치의 변경 사항을 커밋해야함을 주의할 것

Git merge

- `git merge {합칠 브랜치 이름}`
- 병합하기 전에 브랜치를 합치려고 하는, 즉 **메인 브랜치로 switch 해야함**
- 세 종류
  - `Fast-Forwad` : 빨리감기
  - `3-way Merge` : 공통 조상
  - `Merge Conflict`

Git workflow

- 어떤 브랜치 전략을 사용할까는 팀에서 정하는 문제
- 브랜치를 자주 생성해야함