오픈소스 - 기말고사

I. 오픈소스 소프트웨어와 라이선스 (19차시)

구분	내용 요약
OSS 정의	**소스코드**가 공개되어 누구나 자유롭게 **확인, 수정, 제작, 재배포**가 가능한 라이선스를 만족하는 소프트웨어.
출발 개념	리처드 스톨먼의 **자유 소프트웨어 (Free Software)**와 **카피레프트 (Copyleft)** 사상.
개발 서버	소스코드 협업 및 공유 플랫폼: **GitHub, GitLab, Bitbucket** 등.
대표 라이선스	1. **GPL (강한 카피레프트)**: 2차 저작물 제작 시 **소스코드 전체 공개 의무** (가장 강력). 2. **LGPL (약한 카피레프트)**: 사용된 **라이브러리 부분만** 소스코드 공개 의무. 3. **MIT/Apache (비 카피레프트)**: **소스코드 공개 의무 없음** (가장 느슨함), 라이선스와 저작권 관련 명시 의무만 있음.

II. Git 작업 공간 및 임시 저장 (20~21차시)

구분	내용 요약
Git 4 영역	**작업 디렉토리**, **스테이징 영역**, **깃 (지역) 저장소**에 **임시 저장소 (stash)**가 추가된 개념.
`stash` **기능**	**커밋하지 않은** 작업 디렉토리와 스테이징 영역의 변경 사항을 **임시 저장**하고, 해당 영역을 **깨끗한 상태**로 되돌림.
필요성	작업 중 브랜치 이동(`checkout` , `switch`)을 위해 **작업 디렉토리를 정리**해야 할 때 사용.
주요 명령어	1. `git stash` (또는 `git stash save`): 변경 내용 임시 저장. 2. `git stash list` : 저장된 목록 확인. 3. `git stash apply` : 최신 내용을 반영하되 **목록에 유지**. 4. `git stash pop` : 최신 내용을 반영하고 **목록에서 삭제**. 5. `git clean -f` : **Untracked 파일**을 **무조건 삭제**.

III. 브랜치 병합 및 충돌 해결 (22~24차시)

병합 유형	조건	결과 (커밋 이력)	명령어
Fast-forward	현재 브랜치가 대상 커밋의 **직접 조상**일 때 (일렬 상태).	**새 커밋 없이** HEAD 포인터만 이동.	`$ git merge [브랜치명]` (기본)
3-way Merge	두 브랜치가 **공통 조상에서 분기**했을 때.	**새로운 병합 커밋 (Merge Commit)**	`$ git merge [브랜치명]` (기본)

		생성.	
Non Fast-forward	일렬 상태에서도 **강제로 3-way 병합 커밋** 생성.	**선형 이력**에 병합 커밋을 명확히 남김.	`$ git merge --no-ff [브랜치명]`
Squash Merge	대상 브랜치의 모든 커밋을 **하나로 합쳐** 현재 브랜치에 반영.	병합된 브랜치의 **개별 이력은 남지 않음.**	`$ git merge --squash [브랜치명]` (이후 `commit` 필요)
병합 충돌	**동일 파일의 동일 부분**을 두 브랜치에서 모두 수정했을 때 발생.		
충돌 해결 절차	1. 충돌 파일에서 `<<<<<<<`, `=======`, `>>>>>>>` **충돌 표시를 직접 제거**하며 수정. 2. `$ git add <파일>`. 3. `$ git commit`으로 병합 완료.		
병합 취소	충돌 발생 후 병합을 취소하고 이전 상태로 돌아가려면 `$ git merge --abort` 명령 사용.		

IV. 고급 이력 관리 (25~27차시)

1. 리베이스 (Rebase) (25차시)

- **기능**: 현재 브랜치의 **Base (공통 조상)**를 다른 브랜치의 최신 커밋으로 **재배치**하여 합치는 방식.
- **특징**: 커밋 이력이 분기 없이 **선형적**으로 깔끔하게 정리됨. 단, **원래 커밋 이력이 변경**되므로 주의 필요.
- **충돌 해결**: 파일 수정 후 `$ git add <파일>`, 그리고 `$ git rebase --continue`로 진행.

2. 커밋 이력 수정 (26차시)

- **최신 커밋 수정**:
  - `$ git commit --amend -m "new message"` : **최신 커밋 메시지**만 수정하고 새 ID 부여.
  - `$ git commit --amend --no-edit` : 내용만 수정한 후 **메시지 수정 없이** 최신 커밋에 반영하고 새 ID 부여.
- **이전 여러 커밋 수정 (대화형 리베이스)**:
  - `$ git rebase --interactive HEAD~n` 형태로 사용 (HEAD~n의 직전 커밋부터 수정 가능).
  - `r(eword)` : 개별 커밋 메시지 수정.
  - `s(quash)` : 이후 커밋을 이전 커밋에 결합하여 **하나의 커밋**으로 뭉침.

- `d(rop)` : 해당 커밋 삭제.

3. VS Code에서 Git 활용 (27차시)

- **영역 표시**:
  - **Changes 영역**: **작업 디렉토리**의 변경 사항.
  - **Staged Changes 영역**: **스테이징 영역**의 변경 사항.
- **Diff 기능 (CLI)**:
  - `$ git diff` : **스테이징 영역**을 기준으로 **작업 디렉토리**와 비교.
  - `$ git diff --staged` : **HEAD (최근 커밋)**를 기준으로 **스테이징 영역**과 비교.
- **변경 취소/복구 (restore)**:
  - `$ git restore {파일}` : 스테이징 영역 내용으로 **작업 디렉토리** 수정 취소.
  - `$ git restore --staged {파일}` : 깃 저장소(HEAD) 내용으로 **스테이징 영역** 수정 취소 (unstage).

V. 버전 되돌리기 (reset vs revert) (28~30차시)

구분	git reset (되돌리기/재설정)	git revert (취소)
기능	특정 커밋으로 **이동** (타임 머신). 이후 로그 이력 **모두 삭제**.	특정 커밋을 **취소하는 새로운 커밋 생성** (Undo).
이력 관리	**이전 이력 삭제**되어 깨끗해짐 (공동 작업 시 위험).	**이전 이력 그대로 유지**되며 새로운 취소 커밋 추가 (공동 작업 시 안전).
전제 조건	작업 디렉토리 상태에 따라 옵션 사용.	**작업 영역이 깨끗해야** 수행 가능.
주요 옵션	`--hard` : 3개 영역 모두 대상 커밋 내용으로 복사/수정 (작업 내용 모두 삭제). `--mixed` **(기본)**: 깃 저장소/스테이징 영역만 변경, 작업 디렉토리는 유지. `--soft` : 깃 저장소만 변경, 스테이징/작업 디렉토리 유지.	`--no-edit` : 커밋 메시지 자동 생성.
되돌리기	`$ git reset --hard ORIG_HEAD` 로 직전 `reset` 취소 가능.	`$ git revert --abort` 로 취소 가능.