OSS (2학기)

1. OSS 기본 개념 및 라이선스

1) Free Software

Free Software는 소프트웨어를 자유롭게 **사용·복사·수정·배포**할 수 있어야 한다는 철학을 의미한다.

자유(Free)는 가격이 아닌 “사용의 자유”를 의미하며, Free Software Foundation(FSF)이 이를 주도한다.

2) Copyleft

Copyleft는 자유 소프트웨어의 자유가 유지되도록 강제하는 개념이다.

해당 소프트웨어를 수정하여 배포할 경우, **같은 조건(동일한 라이선스)** 아래에서 배포해야 한다.

이를 통해 2차 저작물 역시 자유롭게 유지된다.

3) Copyright

일반적인 저작권으로, 프로그램을 사용할 권리와 수정·배포 권한이 제한된다.

소스 코드 공개 의무가 없으며, 파생 저작물 배포 시 별도의 허가가 필요하다.

4) 주요 라이선스 비교

라이선스	철학	의무 수준	특징
GPL	Copyleft	높음	소스 코드 공개 의무. 수정·배포 시 반드시 GPL 유지
LGPL	약한 Copyleft	중간	라이브러리에 대해 적용. 링크는 허용하고, 수정 시 공개 필요
AGPL	강한 Copyleft	매우 높음	서버·네트워크를 통한 사용도 GPL 조건 강제

라이선스	철학	의무 수준	특징
MIT	Permissive	낮음	자유로운 사용 허가. 소스 공개 의무 없음(저작권 표시만 유지)

2. Stash & Clean

1) Stash 개념

Stash는 작업 디렉토리(Working Directory)와 스테이징 영역(Staging Area)의 현재 변경사항을

임시 저장 공간에 보관하고 작업 공간을 깨끗한 상태로 되돌리는 기능이다.

브랜치를 전환해야 하는 상황 등에서 유용하다.

저장된 stash는 스택(Stack) 구조로 관리된다.

2) Stash 명령어

명령어	설명
`git stash`	WD의 변경사항을 stash에 저장
`git stash --index`	WD + SA 변경사항 모두 stash에 저장
`git stash list`	저장된 stash 목록 확인
`git stash apply`	stash 적용(목록은 유지)
`git stash apply --index`	SA 상태까지 복원
`git stash pop`	stash 적용 후 삭제
`git stash drop`	특정 stash 삭제

WD(Working Directory), SA(Staging Area)가 어떤 수준에서 저장되는지가 중요한 포인트이다.

3) Clean 명령어

명령어	설명
`git clean -i`	untracked 파일을 대화형으로 삭제
`git clean -f`	삭제를 강제로 수행
`git clean -fd`	파일 및 디렉토리 삭제

Clean 명령어는 Git이 추적하지 않는 파일(untracked)만 삭제한다.

---

3. Merge

1) Merge 개념

두 브랜치를 하나로 병합하는 작업이다.

브랜치 간의 관계나 차이에 따라 다양한 병합 방식이 사용된다.

2) Merge 방식

(1) Fast-forward merge

브랜치 간 분기점이 없이 직선 형태로 진행된 경우 사용된다.

HEAD는 단순히 병합 대상 브랜치의 마지막 커밋으로 이동한다.

Merge commit이 생성되지 않는다.

(2) 3-way merge

두 브랜치가 서로 독립적으로 커밋을 가지는 경우 사용된다.

Git은 공통 조상(merge base)을 기준으로 세 방향의 비교(ours, theirs, base)를 수행하여

새로운 병합 커밋을 생성한다.

(3) Squash merge

여러 개의 커밋을 하나의 커밋으로 압축하여 병합한다.

개발 과정에서 커밋이 많을 때 main 브랜치의 히스토리를 간결하게 유지하는 데 유용하다.

3) Merge 옵션

옵션	설명
`--no-ff`	fast-forward 가능해도 merge commit을 강제 생성
`--ff-only`	fast-forward 가능한 경우에만 병합 허용
`--squash`	squash 병합 수행

4) Merge 충돌

두 브랜치에서 동일한 파일의 동일한 부분을 수정하면 충돌이 발생한다.

Git은 자동으로 병합할 수 없으므로 사용자가 파일을 직접 수정해야 한다.

충돌 해결 관련 명령어

명령어	설명
`git merge --abort`	병합 작업 취소
`git merge --continue`	충돌 해결 후 병합 계속
`git checkout --ours`	현재 브랜치의 버전으로 선택
`git checkout --theirs`	병합 대상 브랜치 버전으로 선택

---

4. Rebase

1) Rebase 개념

한 브랜치의 기반(Base)을 다른 브랜치의 마지막 커밋으로 옮기는 작업이다.

Rebase는 브랜치를 더 “직선적인(Linear)” 형태로 정리하기 위해 사용된다.

특징

- fast-forward 기반 병합 방식
- 히스토리를 재작성하므로 협업에서는 주의 필요
- 충돌 발생 시 수동 해결 후 계속 진행

2) 기본 Rebase 명령어

명령어	설명
`git rebase main`	현재 브랜치의 커밋을 main 위로 재배치
`git rebase master bugfix`	bugfix 브랜치를 master 위로 재배치

3) Interactive Rebase

Interactive rebase는 커밋을 선택·수정·합치기 등 세밀하게 편집할 수 있는 기능이다.

주요 옵션

옵션	설명
`pick`	해당 커밋 사용
`reword`	커밋 메시지 수정
`edit`	커밋 내용 수정
`squash`	이전 커밋과 합침
`fixup`	메시지 없이 합침

4) 커밋 수정 관련 명령어

명령어	설명
`git commit --amend`	마지막 커밋 내용 또는 메시지를 수정
`git commit --amend -m "msg"`	메시지를 바로 수정
`git rebase --abort`	rebase 작업 취소
`git rebase --continue`	충돌 해결 후 rebase 계속

---

5. Reset / Revert / ORIG_HEAD

1) Reset 개념

Reset은 HEAD의 위치와 Staging Area 및 Working Directory 상태를 지정한 커밋으로 되돌리는 명령어이다.

Reset은 히스토리를 변경할 수 있으므로 주의가 필요하다.

2) Reset 옵션

옵션	설명
`--soft`	HEAD만 이동하고 Staging Area, WD는 변경하지 않는다.
`--mixed`	HEAD 이동 + Staging Area 초기화. WD는 유지한다. (기본값)
`--hard`	HEAD, Staging Area, WD 모두 해당 커밋 상태로 되돌린다.

3) 관련 개념

개념	설명
`HEAD~`	바로 이전 커밋

개념	설명
`HEAD~2`	두 단계 이전 커밋
`ORIG_HEAD`	reset 등 위험 작업 직전의 HEAD를 백업하는 포인터
`STASH@{1}`	직전 stash

6. Revert

Revert 개념

Revert는 특정 커밋의 변경사항을 되돌리는 **새로운 커밋을 생성**하는 방식이다.

Reset과 달리 기존의 히스토리를 유지하면서 변경만 상쇄하므로

협업 환경에서 안전하게 되돌리기 작업을 수행할 수 있다.

Revert는 Reset처럼 워킹 디렉토리 전체를 과거 상태로 되돌리지 않고,

지정한 커밋의 변경 내용에 반대되는 작업을 새 커밋으로 추가한다.

따라서 다른 개발자와 협업 시 충돌을 최소화할 수 있다.

Revert 명령어

명령어	설명
`git revert HEAD`	마지막 커밋을 되돌리는 새 커밋 생성
`git revert <커밋ID>`	특정 커밋을 되돌림
`git revert --no-edit`	기본 메시지 사용

Revert 특징

- 기존 히스토리를 유지한다.
- 협업 환경에서 reset보다 훨씬 안전하다.
- 여러 개의 커밋을 연속으로 revert할 수 있다.
- 병합 커밋을 revert할 경우 추가 옵션(`m`)이 필요할 수 있다.