STEP

디지털 기초

Git 기본 퀵 가이드

3차시 : 브랜치 관련 명령의 동작 원리

브랜치 조작 명령

1. 브랜치

(1) 브랜치
- 브랜치는 폴더 사용 방식과 유사
- 프로젝트 진행 시 새로운 기능의 시작은 브랜치 생성
- Git을 사용한 협업 시 브랜치 생성은 필수

(2) 브랜치 생성
- Git branch [브랜치 이름] [커밋 체크섬] # 브랜치 생성
- Git switch [브랜치 이름] # 브랜치 전환,
  작업 디렉토리 덮어쓰기

(3) Git에서 명령은 작업 디렉토리, 스테이지, 커밋과
커밋 히스토리에 영향

(4) 작업 디렉토리 확인 방법
- 소스트리에서 탐색기 버튼 사용
- CLI에서 ls 명령 사용

2. 소스트리로 브랜치 생성

(1) 대상 커밋 선택

(2) 마우스 우클릭하여 브랜치 선택

(3) 브랜치 이름 입력→브랜치 생성 버튼 클릭→브랜치 생성
'새 브랜치 체크아웃' 체크 시 브랜치 생성과 동시에
브랜치 및 작업 폴더 내용 변경

(4) Git 체크아웃 명령의 기능
- 현재 작업 중인 브랜치 변경: 브랜치 변경 명령 switch
- 커밋으로부터 특정 파일들을 꺼내 파일 복구
: 파일 복구 기능 restore

브랜치 조작 명령

3. 브랜치를 변경한 상태에서 새로운 커밋 생성

(1) 새로운 feature2.py 파일 생성

(2) 기존 hello.py도 내용 변경

(3) 변경한 내용을 다시 스테이지에 추가, 새로운 커밋 생성

4. 로그 확인

(1) Git bash와 소스트리로 히스토리 그래프 확인 시
신규 가지 생성 확인

(2) 메인 브랜치에서는 Git을 사용한 브랜치 관리는
하지 않는 것이 일반적

(3) 새로운 기능 구현 시
- 기능 이름으로 브랜치 생성
- 브랜치로부터 새로운 커밋 생성

5. 브랜칭(워크플로우) 전략

(1) 브랜치를 사용해 작업을 관리·협업하는 방식

(2) Git-flow, GitHub flow, GitLab flow 등

브랜치 조작 명령

6. 커밋의 빈도와 크기

(1) 커밋은 원자적으로 수행
- 원자적: 커밋의 제목을 쓸 때 작은 내용을 담아 커밋하는 것이 좋다는 의미
- 최대한 작게, 여러 번 커밋 생성

(2) 작업이 하나의 의미를 담고 있다면 커밋 내 파일 개수는 무관

(3) 브랜치 생성은 작업 폴더 전체의 사본을 만드는 것과 유사
- Git의 브랜치는 필요한 만큼 생성해도 저장 공간, 속도에 영향을 주지 않음

7. merge 명령

(1) 새로운 기능 완성 후 항상 main 브랜치로 통합

(2) merge 명령: 두 브랜치를 합치는 명령
- 각 브랜치의 최신 커밋을 합쳐서 새로운 커밋(merge 커밋) 생성

(3) merge 명령의 특징
- 단순하고 어렵지 않은 명령
- 두 커밋에서 같은 파일을 작업 시 충돌 발생
- 충돌은 당연한 현상: GUI 프로그램, 코드 에디터 대부분 충돌 해결 기능 보유

브랜치 조작 명령

8. 신규 feature2 브랜치 커밋 생성 및 병합

(1) 커밋 생성→작업 브랜치를 main으로 변경→merge 명령 수행
- 'Git switch main' 명령: 메인 브랜치로 전환
- 'Git merge feature2'명령: 병합 시도

(2) 왼쪽 메뉴 > main branch 이름 더블 클릭

(3) 마우스 우클릭 > checkout main 브랜치 선택

(4) 히스토리에서 feature2 브랜치 선택 > 병합 클릭

(5) 병합 시도 시 충돌 발생
- 해결방법: 변경사항 둘 중 하나만 활용
  ① 소스트리: 소스트리에서 충돌이 발생한 파일 우클릭 > 충돌 해결 > 저장소 것을 사용하여 해결 클릭
  ② 코드에디터: 에디터를 열어 불필요한 내용 제거 > 직접 필요한 내용 수정 > 저장 > 충돌 내용 변경·확인 후 스테이지에 올리기 > 커밋 선택해 새로운 merge 커밋 생성

9. 커밋을 되돌리는 명령

(1) revert

(2) hard reset

① 완전히 되돌리고 싶을 때
② 터미널에서는 Git reset —hard HEAD~1 입력

(3) 소스트리를 이용한 초기화

① 머지 커밋 직전 커밋 선택 후 마우스 우클릭
② '이 커밋까지 현재 브랜치를 초기화 – hard' 선택
③ 머지 커밋이 생성 이전 상태로 복구

브랜치 조작 명령

10. GitHub의 Pull Request를 이용한 병합

(1) Pull Request(PR) 이용
- Git과 GitHub으로 협업 시 대부분 Pull Request 이용해 병합
'feature2'브랜치로 변경

(2) 병합 과정
- 변경한 브랜치를 GitHub 원격 저장소로 push

→ 소스트리에서 push 버튼 클릭 후
체크박스에서 feature2 선택

→ feature2 브랜치를 GitHub에 push

→ GitHub 페이지의 프로젝트 저장소로 이동

→ Pull Request 탭

→ New Pull Request 버튼 클릭

→ PR 화면에서 base:main compare: feature2 브랜치 선택

→ 경고 메시지 무시, Create pull request 선택

→ PR 메시지 입력

→ Pull Request 생성

(3) Pull Request 생성 시 merge 버튼 비활성화
- resolve conflict 버튼으로 충돌 해결 가능

브랜치 조작 명령

11. merge 전 리뷰를 통해 수정사항 발견 시

(1) 현재 브랜치에서 수정사항 커밋 후 push 명령

(2) GitHub의 PR이 자동 업데이트

(3) 리뷰 완료 시 merge 버튼 눌러 병합 작업 수행

12. GitHub PR 로컬로 가져오기

(1) 현재 머지는 GitHub에서만 이뤄진 상태

① 소스트리를 이용해 main 브랜치로 기본 브랜치 변경
② pull 선택하여 GitHub의 머지 커밋을 로컬에 가져오기

13. Pull Request의 장점

(1) GitHub의 다양한 부가기능 사용 가능

(2) 작업 내용이 웹 페이지 Pull Request 탭에 기록

(3) PR이 잘못 되었을 경우 GitHub에서는 PR Revert 기능 제공

브랜치 내부 구조

1. 커밋 후 스테이지 상태 확인

(1) 터미널에서 ls 명령 수행
- 작업 디렉토리에 어떤 파일이 있는지 먼저 확인

(2) Git ls-files —stage 명령 수행
- 스테이지 안의 내용 확인

(3) Git ls-tree HEAD 명령 수행
- HEAD 브랜치의 커밋에 들어있는 내용 확인

(4) 커밋 이후 작업 폴더, 스테이지, HEAD 커밋 모두 같은 내용 포함
- 첫 번째 커밋의 README와 LICENSE 파일더 스테이지에 들어 있음

2. 작업 폴더와 커밋의 체크섬 비교

(1) 작업 디렉토리, 스테이지, 커밋 안 파일의 체크섬 값이 모두 같다는 것의 의미
- Git의 커밋에는 항상 파일 전체를 저장한다는 것

(2) Git이 파일 전체를 저장하는 이유는 속도 때문
-모든 커밋 간 변경사항을 비교할 경우 오랜 시간 소요

(3) Git의 특징

① 자신만의 정보로 디렉토리 파일 복구 가능
② 단 하나의 blob만 저장: 해시맵과 유사한 방식으로 관리
- 파일 내용이 같으면 blob의 체크섬 값 동일
- 로컬 저장소 내 체크섬 값이 같은 blob은 하나만 존재

브랜치 내부 구조

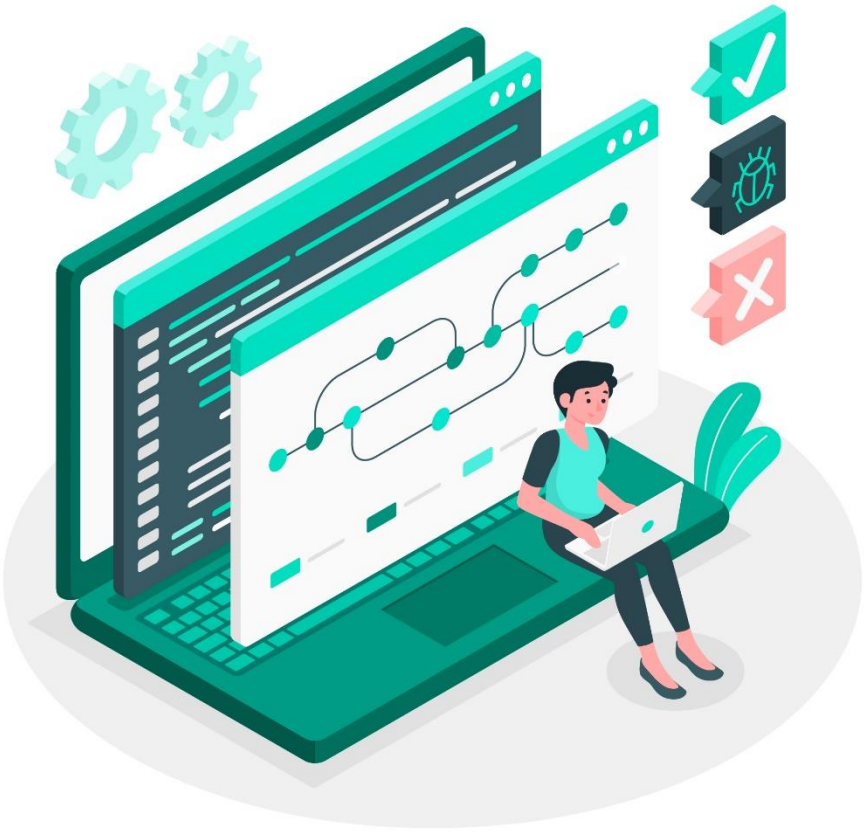
3. 브랜치와 태그 비교

(1) 브랜치

① 체크아웃 후 커밋을 할 때마다 새로운 커밋을 가리킴
② 단방향 연결 리스트 형태: 연결 리스트의 특정 커밋

(2) 태그

① 항상 고정된 커밋을 가리킴
② 프로젝트의 특정 기능 완성 및 목표 도달 시 사용
③ 태그 push 기능
- GitHub의 release 탭에서 태그의 스냅샷 다운로드 가능



오늘의 키워드

브랜치, 소스트리, 로그 확인, 브랜칭 전략, merge 명령,
hard reset, Pull Request, 커밋 내부 구조,
브랜치 내부 구조, 체크섬, 브랜치와 태그

생각할 거리

브랜치 조작 명령에 대해 이해를 기반으로
브랜치의 동작 원리와 내부 구조를 이해하자!