6. 깃과깃허브사용법

Git과 GitHub의 차이점

1. **Git**: 로컬에서 버전 관리를 담당하는 도구입니다. 개발 프로젝트의 변경 사항을 추적하고 관리하는 역할을 하며, 각 폴더와 파일의 변경 이력을 기록합니다.
2. **GitHub**: Git을 기반으로 하는 온라인 플랫폼입니다. 개발자들이 Git을 통해 관리하는 프로젝트를 업로드하고 공유할 수 있으며, 협업을 용이하게 합니다.

Git의 기능과 명령어

- **CRUD (Create, Read, Update, Delete)**: Git은 파일 및 폴더에 대한 생성, 조회, 수정, 삭제 기능을 제공합니다. 프로젝트의 이력 관리를 위해 다양한 명령어를 통해 이 기능을 수행합니다.

Git의 주요 용어와 개념

1.CREATE(생성)

1. **Git init**:
    - Git을 사용하려는 폴더에서 `git init` 명령어를 입력하여 Git 저장소를 초기화합니다.
    - 이 명령어를 실행하면 해당 폴더에 `.git` 폴더가 생성되고, Git을 통해 이 폴더의 파일과 폴더들을 관리할 수 있게 됩니다.
2. **Git add**:
    - `git add` 명령어는 변경 사항을 Git에 추가하는 역할을 합니다.
    - 예를 들어 `git add .` 명령어를 실행하면 폴더 내 모든 파일과 폴더의 변경 사항을 스테이징 영역에 추가합니다.
3. **Git commit**:
    - `git commit -m "메시지"`를 통해 변경된 사항을 Git에 저장합니다.

- 메시지는 변경 사항을 설명하는 간단한 내용으로 작성합니다.
- 이 단계는 실제로 변경된 내용을 Git에 기록하고, 해당 변경 사항이 언제 발생했는지에 대한 정보를 저장합니다.

4. **Git push**:
   - `git push` 명령어는 로컬 Git 저장소에 저장된 변경 사항을 GitHub에 업로드합니다.
   - 이를 통해 온라인 저장소에 프로젝트를 백업하거나 공유할 수 있습니다.

브랜치 (Branch)

- 브랜치는 독립적인 개발 환경을 제공합니다. 한 프로젝트에서 여러 버전을 동시에 관리할 수 있어 개발 중인 기능이나 수정 사항을 별도로 테스트하고 병합할 수 있습니다.

예시 흐름

1. `git init` → 저장소 초기화
2. `git add .` → 변경된 모든 파일을 스테이징 영역에 추가
3. `git commit -m "첫 커밋"` → 변경 사항을 Git에 커밋
4. `git push` → GitHub에 변경 사항 업로드

정리

- Git은 로컬에서 프로젝트의 버전 관리를 담당하고, GitHub은 이러한 프로젝트를 온라인에서 관리, 공유, 협업할 수 있는 플랫폼입니다.
- Git을 사용하여 프로젝트의 변경 이력을 관리하고, `init`, `add`, `commit`, `push` 등의 명령어로 프로젝트의 상태를 관리합니다.

2. Read (읽기)

- `git status`: 현재 작업 디렉토리의 상태를 확인합니다. 어떤 변경 사항이 있는지 보여줍니다.
- `git log`: 지금까지 커밋한 내역을 확인할 수 있습니다. 언제, 누가, 어떤 메시지로 커밋했는지를 알 수 있습니다.

3. Update (수정)

- **파일 수정 프로세스**:

1. 파일을 수정한 후 `git add <파일명>` 명령어로 변경 사항을 스테이징 영역에 올립니다.
2. 이후 `git commit -m "커밋 메시지"` 명령어를 사용해 해당 변경 사항을 저장합니다.

- **참고**: 동일한 파일을 여러 번 커밋해도 기존 데이터가 덮어쓰이는 것이 아니라, 시간별로 저장되어 이전 버전으로 되돌아갈 수 있습니다.

4. Delete (삭제)

- `git rm <파일명>`: 해당 파일을 삭제하고, 삭제 내용을 기록하기 위해 커밋해야 합니다. 삭제 후 **`git commit -m "삭제 메시지"`*로 커밋하면 완전한 삭제가 이루어집니다.

5. 주의사항 (Git Rebase)

- `git rebase`: 브랜치의 커밋 기록을 변경하는 명령어입니다. 특정 시점으로 커밋을 되돌릴 수 있지만, 주의하지 않으면 기록이 완전히 사라져 복구가 어려울 수 있습니다. 신중하게 사용하세요.

6. Branch 관리

CRUD 작업은 일반적으로 동일한 브랜치에서 수행되지만, 다른 브랜치에서도 병합(Merge) 기능을 통해 여러 브랜치 간의 작업을 통합할 수 있습니다. 이 부분은 브랜치와 관련된 작업을 배울 때 자세히 다루면 됩니다.

---

1. 브랜치 생성 및 관리

- **브랜치 생성**:
  - 새로운 브랜치를 만들 때 `git branch 브랜치이름` 명령어를 사용합니다.
- **체크아웃 (Checkout)**:
  - `git checkout 브랜치이름`을 통해 브랜치로 이동합니다.
  - `git checkout -b 새로운브랜치이름` 명령어로 브랜치를 생성하면서 동시에 이동합니다.

2. 브랜치 병합 (Merge)

- **병합 과정**:
  - 브랜치들을 하나로 합칠 때는 `git merge 브랜치이름` 명령어를 사용합니다.
  - 병합 시 발생할 수 있는 "충돌"이 있는데, 이는 동일한 파일의 동일한 부분이 다른 브랜치에서 수정되었을 때 발생합니다.

- **충돌 해결**:
  - 충돌이 발생하면 파일을 수동으로 수정하고 `git add .`과 `git commit -m "충돌 해결 메시지"`로 충돌을 해결합니다.

3. 브랜치 삭제 (Delete)

- **병합된 브랜치 삭제**:
  - `git branch -d 브랜치이름` 명령어로 병합된 브랜치를 삭제할 수 있습니다.
- **병합되지 않은 브랜치 삭제**:
  - `git branch -D 브랜치이름`으로 병합되지 않은 브랜치를 강제 삭제할 수 있습니다.

4. GitHub와 원격 저장소 관리

- **GitHub에 푸시 (Push)**:
  - 로컬 브랜치의 변경 사항을 원격 저장소로 푸시할 때 `git push 원격저장소 브랜치이름` 명령어를 사용합니다.
- **GitHub에서의 CRUD 기능**:
  - GitHub에서 브랜치를 삭제할 때는 `git push origin --delete 브랜치이름` 명령어를 사용합니다.

5. README.md와 마크다운 파일

- **README.md 파일 작성**:
  - 프로젝트에 대한 설명을 정리하는 README.md 파일은 포트폴리오의 중요한 요소입니다.
- 마크다운(Markdown) 형식으로 작성하며, 프로젝트에 대한 가이드, 설치 방법, 사용 방법 등을 포함시킵니다.

6. 학습 방법 및 조언

- Git과 GitHub는 반복적으로 학습해야 하며, 함께 공부하면 동기부여와 실력 향상에 도움이 됩니다.
- 학원에서 공부할 때 여러 번 복습하고 실습하면서 이해를 높이는 것이 좋습니다.