✨

Git과 버전 관리 시스템 요약

학습목표

- Git의 기본 개념과 사용법을 이해할 수 있습니다.
- 버전 관리 시스템의 필요성을 이해할 수 있습니다.
- Git 환경설정, GitHub/GitLab 과 GIt의 관계를 이해할 수 있습니다.
- 원격 저장소(Remote Repository)와 로컬 저장소(Local Repository)의 차이를 이해할 수 있습니다.
- Git의 세 가지 영역 및 상태를 이해할 수 있습니다.(Committed, modified, staged)
- 하나의 Remote Repository에서 협업할 수 있습니다.
- 충돌이 발생했을 경우 해결할 수 있습니다.
- Git branch에 대해 이해하고 브랜치를 생성, 전환, 머지, 삭제할 수 있습니다.

반드시 알아야 할 기본 GitHub의 기능과 Git 명령어

- Fork
- clone
- status
- restore
- add
- commit
- reset
- log
- pull
- push
- init
- remote add
- remote -v

요약

Git의 기본 개념과 사용법을 이해할 수 있습니다.

버전 관리 시스템의 필요성을 이해할 수 있습니다.

- 버전 관리 시스템이란 파일이나 코드의 변경사항을 기록하고 특정 시점의 버전을 되돌릴 수 있는 시스템입니다. 소프트웨어 개발에서는 여러 사람이 협업하거나 코드를 수정하거나 테스트할 때 버전 관리 시스템이 필요합니다.

Git 환경설정, GitHub/GitLab 과 GIt의 관계를 이해할 수 있습니다.

- Git은 분산형 버전 관리 시스템으로, 인터넷에 연결되지 않아도 로컬에서 작업할 수 있고, 다른 사람과 쉽게 공유할 수 있습니다. GitHub과 GitLab은 웹 기반의 호스팅 서비스로, Git 저장소를 온라인에 만들고 관리할 수 있습니다.

원격 저장소(Remote Repository)와 로컬 저장소(Local Repository)의 차이를 이해할 수 있습니다.

- 원격 저장소(Remote Repository)는 인터넷에 있는 공개된 또는 비공개된 저장소로, 여러 사람이 협업하거나 백업하는 용도로 사용됩니다. 로컬 저장소(Local Repository)는 개인 컴퓨터에 있는 저장소로, 자신만 접근할 수 있습니다.

Git의 세 가지 영역 및 상태를 이해할 수 있습니다.(Committed, modified, staged)

- Git은 작업 디렉토리(Working Directory), 스테이징 영역(Staging Area), 커밋 영역(Commit Area)라는 세 가지 영역으로 구분됩니다. 작업 디렉토리는 실제 파일들이 있는 곳으로, 수정된(Modified) 상태라고 합니다. 스테이징 영역은 커밋하기 전에 임시로 저장하는 곳으로, 추가된(Staged) 상태라고 합니다. 커밋 영역은 스테이징 영역에서 확정된 변경사항들을 저장하는 곳으로, 커밋된(Committed) 상태라고 합니다.

하나의 Remote Repository에서 협업할 수 있습니다.

- 하나의 원격 저장소에서 여러 사람이 협업하려면 먼저 자신의 로컬 저장소와 원격 저장소를 연결해야 합니다. 이때 git remote add 명령어를 사용합니다. 예를 들어 원격 저장소의 주소가 https://github.com/user/repo.git 라면 다음과 같이 입력합니다.

```
git remote add origin https://github.com/user/repo.git
```

  - origin은 원격 저장소의 별칭입니다. 별칭은 다른 것으로 바꿀 수 있습니다.
- 그 다음 git push 명령어로 로컬 저장소의 커밋들을 원격 저장소에 업로드합니다. 예를 들어 master 브랜치를 업로드하려면 다음과 같이 입력합니다.

```
git push origin master
```

  - master는 브랜치 이름입니다.
- 반대로 git pull 명령어로 원격 저장소의 최신 커밋들을 로컬 저장소에 다운로드합니다. 예를 들어 origin의 master 브랜치를 다운로드하려면 다음과 같이 입력합니다.

```
git pull origin master
```

충돌이 발생했을 경우 해결할 수 있습니다.

- 충돌이란 두 개 이상의 브랜치에서 같은 파일의 같은 부분을 수정하고 병합하려고 할 때 발생하는 문제입니다. 충돌이 발생하면 Git은 충돌 부분을 표시해줍니다. 예를 들어 다음과 같은 형식으로 나타납니다.

```
<<<<<<< HEAD
This is some content from the current branch.
=======
This is some content from another branch.
>>>>>>> another-branch
```

  - <<<<<<< HEAD와 ======= 사이에 있는 내용은 현재 브랜치에서 수정한 내용입니다. =======와 >>>>>>> another-branch 사이에 있는 내용은 다른 브랜치에서 수정한 내용입니다. another-branch는 병합하려는 브랜치의 이름입니다.
- 충돌을 해결하려면 원하는 내용을 선택하거나 새로운 내용을 작성해야 합니다. 그리고 git add 명령어로 스테이징 영역에 추가하고 git commit 명령어로 커밋해야 합니다. 예를 들어 다음과 같이 입력합니다.

```
git add conflicted-file.txt
git commit -m "Resolved conflict in conflicted-file.txt"
```

  - conflicted-file.txt는 충돌이 발생한 파일의 이름입니다. -m 옵션은 커밋 메시지를 입력하는 옵션입니다.

Git branch에 대해 이해하고 브랜치를 생성, 전환, 머지, 삭제할 수 있습니다.

- Git의 브랜치란 독립적으로 어떤 작업을 진행하기 위한 개념입니다. 브랜치를 만들면 Git은 'HEAD'라는 특수한 포인터를 사용하여 지금 작업하는 로컬 브랜치를 가리킵니다.
- git branch 명령어로 브랜치를 만들 수 있습니다. 예를 들어 새로운 브랜치를 testing이라고 이름짓고 싶다면 다음과 같이 입력합니다.

```
git branch testing
```

- git checkout 명령어로 브랜치를 옮길 수 있습니다. 예를 들어 testing 브랜치로 이동하려면 다음과 같이 입력합니다.

```
git checkout testing
```

- git checkout 명령어에 -b 옵션을 붙이면 브랜치를 생성하면서 옮길 수 있습니다. 예를 들어 testing 브랜치를 생성하면서 이동하려면 다음과 같이 입력합니다.

```
git checkout -b testing
```

- git merge 명령어로 두 개의 브랜치를 병합할 수 있습니다. 예를 들어 master 브랜치와 testing 브랜치를 병합하려면 다음과 같이 입력합니다.

```
git checkout master
git merge testing
```

- git branch -d 명령어로 사용이 완료된 브랜치를 삭제할 수 있습니다. 예를 들어 testing 브랜치를 삭제하려면 다음과 같이 입력합니다.

```
git branch -d testing
```