Git 버전 관리 실습

1. Git & GitHub 개요

1.1 Git이란?

분산 버전 관리 시스템**(DVCS, Distributed Version Control System)**
코드의 변경 이력을 관리하고 협업을 쉽게 할 수 있도록 지원하는 도구
로컬 저장소**(Local Repository)**와 원격 저장소**(Remote Repository)**로 구성

1.2 GitHub란?

Git을 기반으로 한 클라우드 기반 원격 저장소
코드 백업, 협업, 이슈 관리, 코드 리뷰 등을 지원
주요 기능**:**

- **Pull Request(PR):** 코드 변경 사항을 제안하고 리뷰를 요청
- **Issues:** 버그 및 작업할 내용을 관리
- **Actions:** CI/CD 자동화

---

2. Git 기본 명령어 및 실습

2.2 Git 기본 명령어

명령어	설명
`git init`	새로운 Git 저장소 생성
`git clone <URL>`	기존 원격 저장소 복제
`git status`	변경된 파일 상태 확인
`git add <파일명>`	특정 파일을 스테이징 영역에 추가
`git commit -m "메시지"`	변경 사항을 저장소에 커밋
`git push origin main`	변경 사항을 원격 저장소에 푸시
`git pull origin main`	원격 저장소의 변경 사항을 가져옴
`git branch`	현재 브랜치 확인
`git checkout -b new-feature`	새로운 브랜치 생성 후 전환
`git merge new-feature`	새로운 브랜치를 main에 병합

실습: 간단한 서버 구현 기술인 nodejs를 설치 해서 테스트 했습니다.

먼저 사용 하는 PC에 Git을 설치하고

Github 웹 싸이트에도 계정을 만들어야 합니다.

다음은 CLI 환경에서 작업 합니다. (Windows는 명령프롬프트(CDM)에서 작업)

(실행 하려는 위치에서 Shift + 오른쪽 클릭 > 명령프롬프트, 파워쉘, 터미널 등)

또는 실행 > cmd 열기 합니다.

깃 버전 확인 명령

> git --version

깃 업데이트

> git update-git-for-windows

- - -

맥용 업데이트 (homebrew 이용)

> brew update

> brew upgrade git

git 설치

- brew가 설치 되어 있지 않다면 brew 설치

  /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

brew를 이용해서 git 설치 명령어

> brew install git

Xcode Command Line Tools 업데이트

> xcode-select --install

1. git 사용자 이름과 이메일 등록

> git config --global user.name "사용자 이름"

> git config --global user.email "사용자 이메일"

2. git 사용자 이름과 이메일 확인

> git config user.name

> git config user.email

> git config --global --list

3. nodejs 프로젝트 생성하기

> npm init -y

macOS에서 Node.js 설치

참고: https://nodejs.org/en/download

macOSdp nvm 설치 명령어

Download and install nvm:

curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.1/install.sh | bash

in lieu of restarting the shell

\. "$HOME/.nvm/nvm.sh"

Download and install Node.js:
nvm install 22

Verify the Node.js version:
node -v # Should print "v22.14.0".
nvm current # Should print "v22.14.0".

Verify npm version:
npm -v # Should print "10.9.2".

4. 지역 저장소 git 초기화

새 디렉토리 생성 후 이동

> mkdir mac-git-nodejs

> cd mac-git-nodejs

> pwd

> ls -al

git 레포지토리 초기화

> git init

- 숨김 항목 표시: (단축키) Command + Shift + .

5. gitignore.io를 이용한 .gitignore 생성

주소창에 gitignore.io를 치면 아래 경로로 이동한다.

https://www.toptal.com/developers/gitignore

생성 창에 Node Windows (또는 macOS xcode vs node)를 입력 하고 생성 클릭

생성된 내용을 프로젝트 루트에 .gitignore 파일로 저장한다.

.gitignore 파일에 기록된 파일들은 버전 관리 추적 대상에서 제외된다.

6. 프로젝트 파일들을 지역 저장소에 저장

워킹디렉토리 (빨간색 목록)	> add	**스테이지 영역 (녹색 목록)**	> commit -m	**지역 저장소 (git log)**

커밋 상태 확인

> git log

git add --all 후 파일 상태 확인

(빨간색 목록은 워킹데렉토리, 녹색 목록은 스테이지 영역)

> git status

워킹디렉토리의 모든 작업 파일을 스테이지 영역에 추가

> git add --all

스테이지 영역의 파일을 다시 워킹 디렉토리로 돌려 보내기

> git rm --cached <파일이름>

프로젝트 파일을 지역 저장소에 저장

> git commit -m "커밋 메세지"

7. 새로운 브랜치를 만들어서 파일을 수정하고 지역 저장소에 저장하기

master 브랜치 이름 main으로 변경

> git branch -M main

새 브랜치 만들고 checkout

> git branch 새브랜치 (가장 최근의 커밋을 가리킨다)

> git checkout 새브랜치 (HEAD를 master에서 새브랜치로 변경)

파일 내용을 수정하고 새로 저장소에 commit

> git add --all

> git commit -m "커밋내용"

8. 파일 내용 수정 전으로 돌아가기 (master 브랜치로 돌아가기)

> git checkout master

(브랜치는 완벽한 독립공간이다.)

-- 새 브랜치로 checkout하면 파일 수정했던 내용이 살아난다.

-- master브랜치로 checkout하고 git log 하면 새 커밋이 나타나지 않을 수도 있다.

로컬 브랜치 목록 조회

> git branch

-- 새로운 수정에 대한 분기를 commit하지 않으면 checkout 되지 않음.

9. 브랜치 목록 조회

> git branch (로컬 브랜치 목록 조회)

> git branch -r (원격 브랜치 목록 조회)

> git branch -a (모든 브랜치 목록 조회)

10. 파일 상태 구분

tracked	unmodified	커밋되고 난 후에 아무 것도 수정되지 않은 상태
	modified	커밋되고 난 후에 수정 된 상태
	staged	add를 통해 스테이징 영역에 추가된 상태
untracked	한번도 스테이징 영역에 추가 되지 않은 파일의 상태	

11. 방금 수행했던 커밋 변경하기

커밋한 지점에서 작업 파일을 새로 변경 한 후 기존의 commit에 추가한다.

> git commit --amend

-- vi 편집기가 실행 되면 저장 메세지와 작업 파일 목록을 편집 할 수 있다.

-- i키를 누르면 vi 편집 모드로 들어간다.

-- 편집이 끝나면 vi편집기 저장 명령으로 내용을 저장한다.

-- esc키를 누그로 :wq 명령으로 저장 후 vi 종료.

> git log

-- 커밋 로그를 확인 하면 현재 커밋의 내용이 변경된 것을 확인 할 수 있다.

> git checkout master

-- 수정 전의 내용으로 돌아간다.

> git checkout newbranch

-- 수정 후의 내용으로 다시 돌아온다.

12. 이전의 버전관리 도구인 Subversion과 같은 도구는 변경된 부분만을 저장하는 반면 Git은 스냅샷을 저장한다. Git의 각각 커밋은 40자리 문자열인 SHA-1 해시코드로 표시된다.
-- 일반적으로 master 브랜치에서 새로운 작업을 하진 않고 토픽 브랜치(topic branch)를 만들어 작업한 후에 master 브랜치와 병합(merge)한다. 그것이 더 안전하기 때문이다.

13. 브랜치 병합(merge)

작업한 브랜치를 특정 브랜치와 병합(merge)한다.

(Fast-Forward 병합 - 앞으로 빨리 가기)
병합전에 마스터 브랜치로 전환한다.
> git checkout master (또는 main)

마스터 브랜치로 전환 후 작업 브랜치와 병합한다.
> git merge 작업브랜치 (master브랜치가 작업브랜치로 옮겨지면서 병합 됨)

병합한 후에는 작업했던 브랜치는 삭제 해 준다.
> git branch -D 작업브랜치

Fast-Forward 병합이 아닌 일반 병합은 새로은 커밋을 생성한다.

병합 시 유의 사항

1) 병합은 브랜치 레벨에서만 병합이 가능하다.
2) 현재 브랜치에 없는 커밋만 병합 가능
3) 변경 대상 브랜치로 전환 후 병합

(주의) 병합할 두 브랜치가 동일 파일의 동일한 곳을 수정 했다면 충돌이 발생해서 병합이 거부된다. 이럴 때는 병합을 먼저 해결하고 커밋을 재 수정 해야 한다.

14. stash에 작업 내용을 임시 저장

stash : 임시 기억 클립보드 (HEAD가 가리키는 어떤 브랜치도 이용 가능)

새 작업 브랜치 생성

> git branch new_work

> git checkout new_work

-- 새로운 작업을 한다.

작업 브랜치에서 stash로 저장

> git stash (작업 하던 내용이 stash에 저장되고 마스터로 전환)

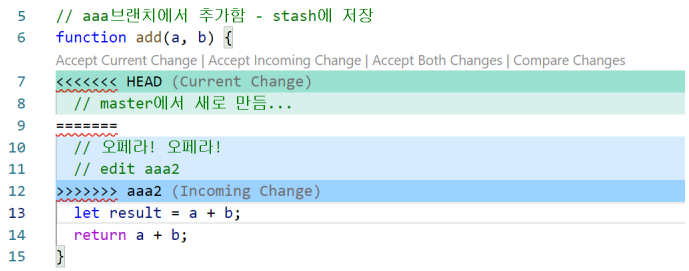
stash에 임시 저장된 내용 반영하기

> git stash pop (적용 후 임시 저장소에서 제거)

> git stash apply (적용)

15. 두 브랜치에서 같은 내용을 수정 했을 때 커밋 후 병합하기

- 두 브랜치에서 같은 위치를 수정하고 모두 커밋 후 병합을 하면 병합은 됨
- 병합 오류가 발생(현재 수정내용과 병합할 커밋의 수정 내용이 모두 보임)
- VS code에서는 "Accept Current Change | Accept Incoming Change | ..." 보임
- 변경된 커밋을 적용 하려면 Accept Incoming Change를 선택하면 바로 수정 됨.



-> 병합 시 충돌되는 부분 수정 후 다시 commit 또는 commit -a 해야 한다.

GitHub 사용하기

1. github 사이트에 회원 가입

https://github.com/

--- 가입메일로 인증 확인 메일이 날아옵니다. 인증 확인 해야 guthub 사용 가능.

--- 인증 완료되면 Settings 메뉴에서 본인 정보 입력.

--- 사용할 프로젝트의 레파지토리 생성. (New Repository)

2. local저장소에 github remote

- (명령어는 깃허브에 새 레파지토리 생성 하면 보여진다.)

> git remote add <저장소별칭> <원격저장소 url>

> git remote add 저장소별칭 https://계정:키@원격저장소/레파지토리.git

3. 지역 저장소 데이터를 원격저장소에 push

> git push -u origin <지역저장소 브랜치>

4. 지역 저장소의 모든 브랜치 push

> git push -u origin --all

5. 개발자가 원격 저장소의 내용을 pull 한 후에 원격 저장소의 내용이 변경되면 Push 거절 된다.
Push 거절 된 경우에는 원격 저장소의 커밋 이력을 가져와 개발자의 커밋 이력을 반영하기 위한 병합을 수행 해야 함.

> git fetch origin

--- 변경된 원격 저장소의 최신 커밋 이력을 가져 온다.

수동으로 지역 저장소와 병합 하기

원격 브랜치 : origin/master의 형식으로 지역 저장소에서 참조.

> git checkout master

> git merge origin/master

> git push origin master

6. Github 사이트에서 내용을 직접 수정 한 후 원격 저장소 내용 Push 해보기

1) 원격 저장소의 소스 수정 (Commit change)

2) 지역 저장소에서 파일 수정 후 커밋

3) 지역 저장소 원격 저장소에 Push (오류)

> git push origin master (Push 거부 됨)

4) 변경이력 가져오기

> git fetch origin

5) 원격 브랜치와 병합

> git checkout master

> git branch

> git merge origin/master

6) 병합 충돌 후 원격 저장소에 Push

> git add --all

> git commit -m "커밋 메세지"

> git push origin master

7. 클론(Clone)

원격 저장소의 내용을 지역 저장소로 복제하는 것.

> git clone 원격저장소url

8. 원격 저장소 변경

git remote set-url origin <변경할 원격 저장소 주소>

9. 자격증명 확인 및 업데이트

git config --global credential.helper cache
git config --global credential.helper 'cache --timeout=3600'

10. 캐시 삭제 및 비활성화

자격 증명 비활성화
git config --global credential.helper
git config --global --unset credential.helper

매번 물어보도록 빈 문자열 설정하기
git config --global credential.helper ""

자격증명 삭제
git credential-cache exit

11. 설정 된 remote 확인 및 재 설정

git config - -global user.name "사용자"
git config - -global user.emmail "사용자이메일"
git remote -v
git remote set-url origin https://github.com/comstudyschool/sw202407day05.git

12. Github에서 토큰 생성

"Developer settings"
-> "Personal access tokens"
-> "Tokens (classic)"으로 이동
-> "Generate new token" 버튼을 클릭.

13. 다른 계정의 레파지토리 가져오기 명령

```
git remote add teacher https://github.com/comstudyschool/sw20240729js.git
# 확인
git remote -v
git fetch teacher/main

# remote repository 에서 필요한 파일만 가져온다.. ( ex) [root]/./day10/images )
git checkout teacher/main -- [가져올 remote repository path]
```

GitHub 협업 방식: 상세 가이드

3.1 기본 협업 흐름

1️⃣ GitHub 저장소 생성 및 팀원 초대

1. **GitHub**에 로그인 후 **New Repository**를 생성합니다.
2. 프로젝트에 적절한 이름, 설명, 공개 여부(공개/비공개) 설정 후 저장소를 생성합니다.
3. **Settings** → **Collaborators** 메뉴에서 팀원들의 **GitHub** 아이디를 입력하여 초대합니다.
4. 초대받은 팀원은 **GitHub** 알림에서 초대 수락 후 협업을 진행할 수 있습니다.

2️⃣ Branch 활용

각 기능을 독립적인 브랜치에서 개발하여 충돌을 방지하고 유지보수성을 높입니다.

📌 새 브랜치 생성 및 전환

git checkout -b feature-branch

- **`feature-branch`**는 새로운 기능을 개발하는 브랜치 이름입니다.
- 브랜치를 생성하면 기존 **`main`**(또는 **`develop`**) 브랜치에서 새로운 분기가 만들어집니다.

📌 기능 개발 후 변경 사항 저장

git add .

git commit -m "새로운 기능 추가**"**

- **`git add .`** → 변경된 모든 파일을 스테이징
- **`git commit -m "새로운 기능 추가"`** → 커밋 메시지를 작성하여 변경 사항을 저장

📌 원격 저장소로 브랜치 푸시

git push origin feature-branch

- 원격 저장소**(GitHub)**에 `feature-branch` 브랜치를 업로드하여 팀원들이 확인할 수 있도록 합니다.

3️⃣ Pull Request(PR) 생성 및 코드 리뷰

> **Pull Request(PR)** 는 팀원들이 변경 내용을 확인하고 코드 리뷰 후 `main` 브랜치로 병합하기 위한 요청입니다.

📌 **GitHub**에서 **PR** 생성

1. **GitHub** 저장소에서 **Pull requests** 탭으로 이동합니다.
2. **New Pull Request** 버튼 클릭 후, `feature-branch` → `main` 병합 요청을 생성합니다.
3. 변경 사항을 설명하는 **PR** 제목과 내용을 입력하고, 리뷰어**(**팀원**)**를 지정합니다.
4. 팀원들은 **PR**을 확인하고 피드백을 남길 수 있습니다.

📌 코드 리뷰 후 승인 및 병합

1. 팀원들이 **PR**을 확인하고 **Approve(**승인**)** 또는 **Change Request(**수정 요청**)** 합니다.
2. 모든 리뷰가 완료되면 **"Merge pull request"** 버튼을 클릭하여 `main` 브랜치에 병합합니다.
3. 병합이 완료된 후 불필요한 브랜치는 삭제하여 관리합니다.

git branch -d feature-branch # 로컬 브랜치 삭제

git push origin --delete feature-branch # 원격 브랜치 삭제

4️⃣ 병합 및 배포

> 코드 리뷰 및 테스트가 완료된 후 배포를 진행합니다.

📌 `main` 브랜치로 병합

git checkout main

git pull origin main

git merge feature-branch

git push origin main

1. `main` 브랜치로 이동 (**`git checkout main`**)
2. 최신 코드 가져오기 (**`git pull origin main`**)
3. `feature-branch` 병합 (**`git merge feature-branch`**)
4. 원격 저장소에 업데이트 (**`git push origin main`**)

📌 필요하면 배포 진행

- **iOS** 앱 개발자의 경우 `main` 브랜치에서 **Xcode**를 통해 `TestFlight` 또는 `App Store Connect`에 배포합니다.
- 웹 개발의 경우 **GitHub Actions** 또는 **CI/CD** 파이프라인을 활용하여 자동 배포를 진행할 수도 있습니다.

---

4. Git 협업 전략 (Git Flow)

> **Git Flow**는 팀 프로젝트에서 효율적인 협업을 위해 브랜치 전략을 체계적으로 관리하는 방법

4.1 Git Flow 브랜치 구조

브랜치	역할
`main`	제품 릴리스가 이루어지는 안정적인 브랜치
`develop`	새로운 기능 개발이 진행되는 브랜치
`feature/*`	새로운 기능을 개발하는 브랜치 (ex: `feature/login`)
`release/*`	릴리스 준비 브랜치 (ex: `release/v1.0`)
`hotfix/*`	긴급 수정 브랜치 (ex: `hotfix/bug-fix`)

4.2 Git Flow 사용 예제

새로운 기능 개발 과정

1️⃣ `develop` 브랜치로 이동

```
git checkout develop
git pull origin develop
```

2️⃣ 새로운 기능 브랜치 생성

```
git checkout -b feature/login
```

3️⃣ 코드 작업 후 변경 사항 저장

```
git add .
git commit -m "로그인 기능 추가"
```

4️⃣ 원격 저장소에 브랜치 푸시

```
git push origin feature/login
```

5️⃣ GitHub에서 PR 생성 → 코드 리뷰 → `develop`에 병합

4.3 실전 협업 예제

팀 프로젝트 진행

📌 **Step 1.** 프로젝트 시작

```
git clone https://github.com/team/project.git
cd project
git checkout -b develop
```

📌 **Step 2.** 기능 개발

```
git checkout -b feature/user-profile
# 코드 수정 후 커밋
git add .
git commit -m "사용자 프로필 화면 추가"
git push origin feature/user-profile
```

📌 **Step 3.** 코드 리뷰 후 병합

- GitHub에서 PR 생성 후 리뷰 요청

- 리뷰 완료 후 `develop`에 병합

```
git checkout develop
git pull origin develop
git merge feature/user-profile
git push origin develop
```

📌 **Step 4.** 릴리스 준비 (**`release`** 브랜치 생성)

```
git checkout -b release/v1.0
git push origin release/v1.0
```

📌 **Step 5.** **`main`** 브랜치에 병합 후 배포

```
git checkout main
git merge release/v1.0
git push origin main
```

---

5. Git 협업 실습

[실습 1] 기본 Git 사용

Git 저장소 초기화
변경 사항 추가 및 커밋
원격 저장소에 업로드

[실습 2] GitHub 협업

팀원과 함께 브랜치를 생성하고 기능을 개발
PR을 생성하고 코드 리뷰 요청
병합 후 `main` 브랜치에 배포