



ROKEY 스터디 2차시 자료_송서영

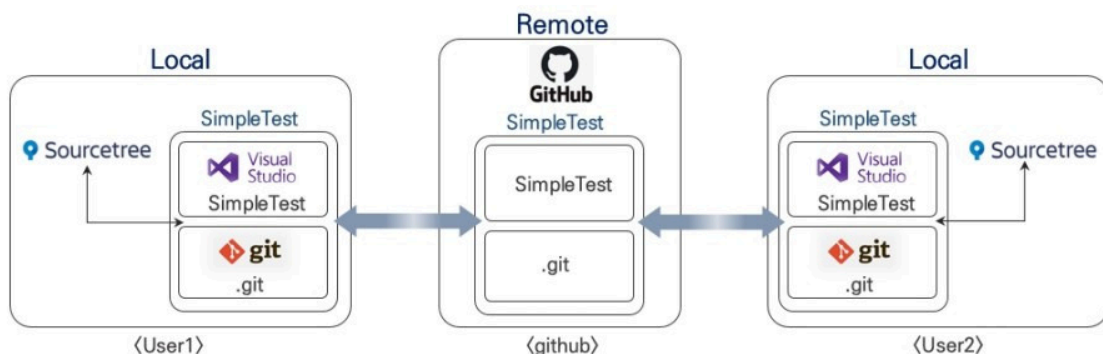
@ ROKEY 6기 스터디 2반 1조 (2025.10.01)

- 입사 후 동료들과 협업 프로젝트 진행..
 - 결과물은 하나여야 하는데 각자 짜 온 코드가 전부 다름
 - 또 내 컴퓨터에 저장된 코드를 공유하고싶는데 알집으로 압축해서 보내기 번거로움→ 실무적인 문제들이 너무 많음...

→ 이런 걸 해결해 주는 것이 깃, 깃허브다!

- 내 소스 코드 저장
- 협업 중 소스 코드 공유 및 버전 관리

1. 깃, 깃허브 개념



1. Git (깃) 개념

- 내 컴퓨터(로컬 환경)에서** 소스 코드의 변경 이력을 관리하는 파일 버전 관리 프로그램
- 코드를 수정할 때마다 기록해둬서, 언제든지 원하는 과거 버전의 코드로 돌아갈 수 있다!
- 비유:** 나만의 주방에서 한 요리법이 담긴 개인 레시피 북.
- 하지만 깃에는 이 개인 레시피 북을 온라인 요리 플랫폼에 올릴 수 있는 기능도 있다!

→ 그 온라인 요리 플랫폼이 GitHub.

2. GitHub (깃허브) 개념

- Git으로 관리하는 코드를 **인터넷에 올려** 다른 사람들과 공유하고 협업할 수 있도록 만든 **온라인 코드 저장소 플랫폼**
- **비유**: 내 레시피를 여러 셰프들과 공유하고, 피드백을 주고받으며 함께 더 멋진 요리를 만들어나가는 **온라인 요리 공유 플랫폼**

3. Git 핵심 기능 및 주요 용어

- **핵심 기능 3가지**

1. Commit (커밋)

: 코드의 변경 사항을 하나의 버전(체크포인트)으로 **저장하는 행위**

- "회원가입 기능 추가"와 같이 새로운 버전에 대한 메시지를 남겨 어떤 변경이 있었는지 기록

2. Branch (브랜치)

: 원본 코드에 영향을 주지 않고 새로운 기능을 개발하거나 버그를 수정할 수 있는 **독립적인 개발 공간(가지)**

3. Merge (머지)

: 브랜치에서 완료된 작업을 원본 코드(또는 다른 브랜치)에 **하나로 합치는 기능**

- **주요 용어**

◦ Repository (저장소)

: 소스 코드가 저장되는 공간

- 내 컴퓨터에 있으면 **로컬 저장소(Git으로 관리)**
- GitHub처럼 인터넷에 있으면 **원격 저장소**

◦ Push (푸시)

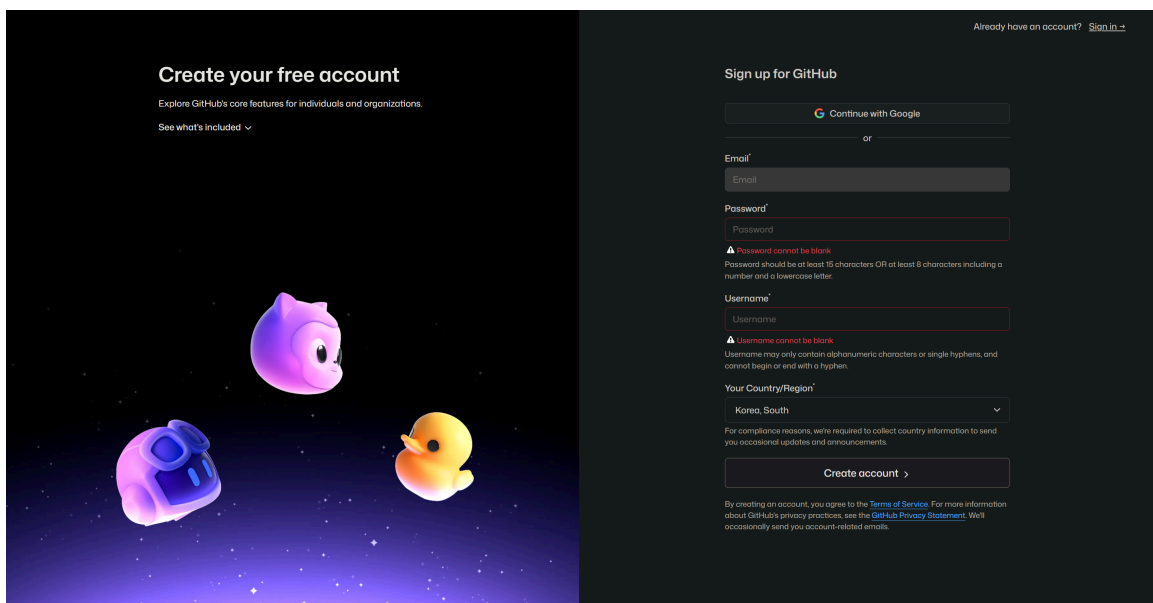
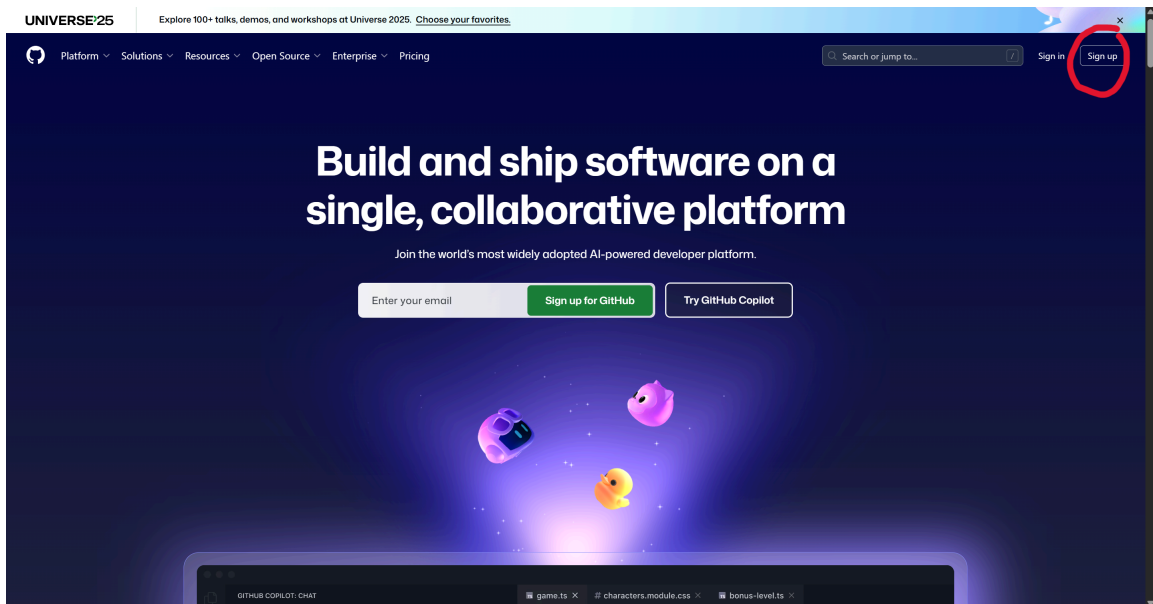
: 내 컴퓨터의 코드 변경 내역을 GitHub와 같은 **원격 저장소로 밀어 올려서 공유(업로드)하는 명령어**

2. 깃허브 세팅법

1. 깃허브 사이트 들어가기

- 구글에 깃허브 또는 github 검색
- github.com

2. Sign Up으로 회원가입



구글 계정으로 가입 권장

3. 깃 환경설정(configuration)

(깃 설치 과정은 1차시에 진행했으므로 생략, 깃 사이트 들어가서 os에 맞는 깃 설치 하면 됩니다)

Git Bash 열고

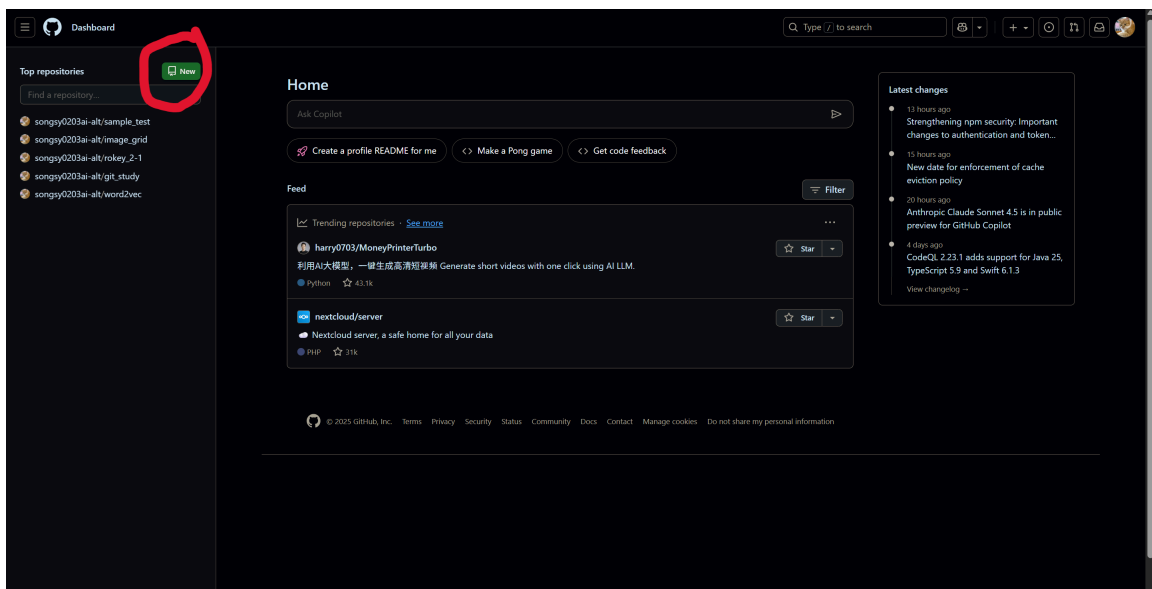
1. 유저명 지정 : `git config --global user.name "내_이름"`
2. 깃허브 가입 이메일 주소 입력 : `git config --global user.email "email"`
3. 설정 제대로 됐는지 확인 : `git config --list`

→ 출력 결과 중 user.name과 user.email만 지정한대로 제대로 나오는지 확인

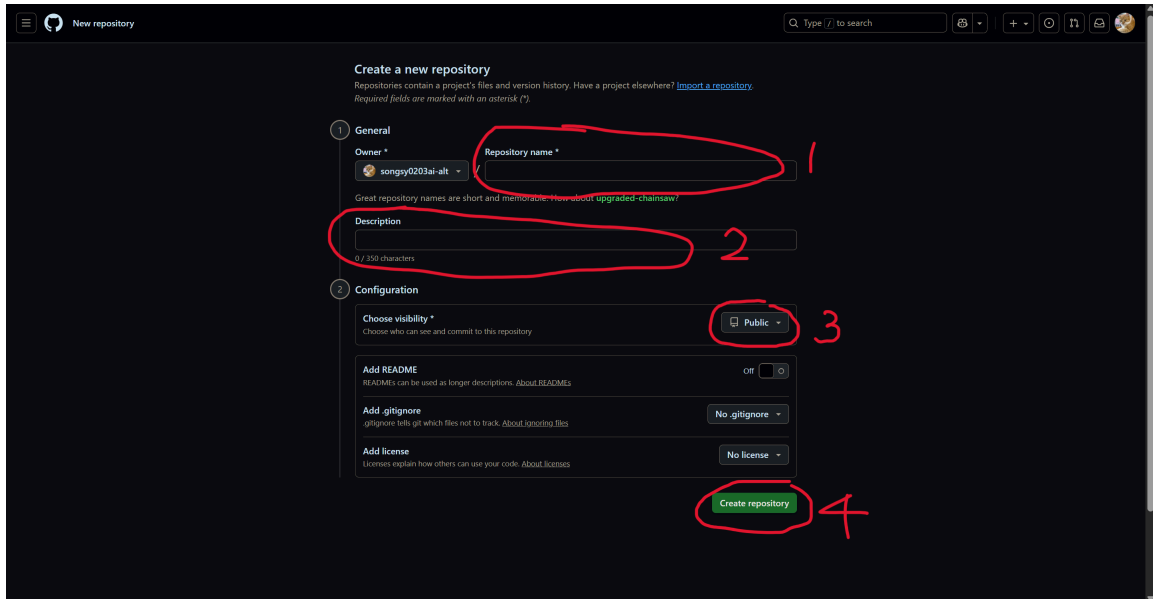
4. 깃허브의 저장소(repository, 레포) 생성

- 깃허브 **저장소(Repository)** : 소스코드를 올리고, 새로운 버전의 소스 코드를 가지 형태(브랜치)로 나누고, 오류나 추가 기능에 대한 의견을 나누기 위한 "공간"

1. New 클릭



2. 저장소의 정보 입력



1. 저장소 이름 설정
2. 어떤 저장소인지 간단 소개
3. 공개/비공개 여부 설정
4. 저장소 생성 버튼 누르면 저장소 생성 완료

→ 이렇게 만든 깃허브 저장소는 어디다 쓰는 거지?

: 내 컴퓨터에서 작업하고 수정하는 소스 코드를 올리는, 남들이 볼 수있는 온라인 공간으로 쓸 수 있다.

5. 깃으로 깃허브 저장소에 내 프로젝트 올리기

사용한 프로젝트(소스 코드) - bfs 알고리즘 소스 코드

(9/29 수업에 작성한 파일이라 모두 갖고 계실겁니다!)

1. 새 폴더 생성

C:\rokey\study\week2

2. C:\rokey\python\ch17 의 bfs.py와 dfs.py를 week2 폴더에 카피하기(예시)

3. VSCode 열고 "새 폴더 열기" → week2 폴더 열기

4. 새 터미널 켜고 `cd C:\rokey\study\week2` 입력

5. 터미널에 아래 명령어 입력

1. 깃 초기화 `git init`

: 깃을 초기화.

맨 처음에 로컬 프로젝트 깃 저장소를 만들 때엔 꼭 git init을 해줘야 함.

- 결과

Initialized empty Git repository in C:/rokey/study/.git/

이름	수정한 날짜	유형	크기
.git	2025-09-30 오후 11:37	파일 폴더	
week1	2025-09-30 오후 11:32	파일 폴더	
week2	2025-09-30 오후 11:34	파일 폴더	

: study 폴더 내에 .git (숨김폴더)가 생성 된다.

- 글씨 초록색?

- VSCode 편집기는 폴더 안에 .git 폴더가 있는 것을 보고 "아, 여기는 Git으로 관리되는 곳이구나!"라고 인식
- 또 아직 한 번도 Git에 저장된 적 없는 파일들을 "추적되지 않은 파일 (Untracked files)"이라는 의미로 초록색(U)으로 표시

2. 깃 더하기 `git add .`

- `git add` : 어떤 파일을 깃에 올릴까?(추가할까?) 라는 뜻
- `.` : 현재 프로젝트 폴더(week2)에 있는 모든 파일, 전부(bfs_copy.py와 dfs_copy.py)라는 뜻
- `git add bfs_copy.py` : bfs_copy.py 파일 하나만 깃에 추가

3. 깃 상태 확인 `git status`

```
PS C:\rokey\study\week2> git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   bfs_copy.py
        new file:   dfs_copy.py
```

- git add . 했으므로 깃에 week2 폴더에 든 모든 소스 코드 파일들이 추가 되어있다.

4. 깃 히스토리 생성 `git commit -m "first commit"`

- commit : 코드 작업하고 저장
- `git commit` : 작업하고 저장한 내용에 대한 깃 히스토리 생성
- `-m` : 메세지명 지정 옵션
- `"first commit"` : 히스토리 이름(메세지명). first commit, second commit은 찐최종, 찐찐최종 이런 막 짓는 버전 구분 파일명의 외국 버전이라고 합니다..

5. 현재 내 컴퓨터에 있는 로컬 프로젝트와 깃허브 저장소를 연결하는 깃 명령어 `git remote add origin HTTPS_GITHUB_REPOSIT_LINK`

입력, 엔터 후 아무 일도 생기지 않으면 연결 성공

- `origin` : 로컬 프로젝트를 깃허브 저장소와 처음 연결 시, git은 깃허브 저장소의 주소를 origin이라는 별명으로 저장함.

6. 로컬 프로젝트 - 깃허브 저장소 연결 여부 확인 `git remote -v`

내 깃허브 저장소 주소의 (fetch)와 (push)가 두 줄로 뜨면 연결 성공



7. 연결 된 깃허브 저장소에 푸시 `git push origin master`

: 내 컴퓨터에서 작업하고 저장한(커밋한) 내용을 origin이라는 이름의 원격 저장소(보통 깃허브 저장소)에 있는 master라는 새로운 작업 공간 브랜치(가지)로 밀어 올려서 업로드

- **push** : 코드 변경 내역, 즉 히스토리를 원격 저장소로 밀어 올림
- **origin** : 깃허브 저장소 별명
- **master** : 프로젝트의 가장 중심이 되는 메인 브랜치 (main 브랜치)
- git - github 연결 authentication(인증) 요청될 시 진행하시면 됩니다
- 아래 결과 나오면 푸시 성공

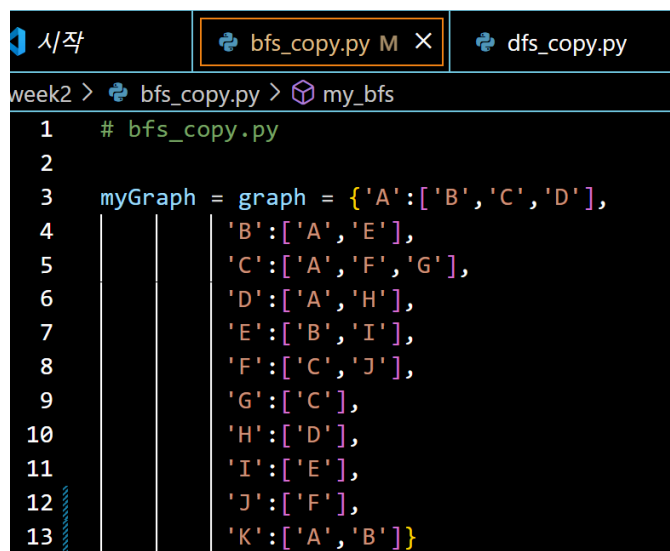
* [new branch] master → master

8. 깃허브 저장소 새로고침하면 로컬 프로젝트에서 다루던 파일들이 아래와 같이 깃허브 저장소에서 보이게 됩니다.

 bfs_copy.py	first commit	33 minutes ago
 dfs_copy.py	first commit	33 minutes ago

6. 만약 로컬 프로젝트의 수정 사항을 깃허브 저장소에 반영하고 싶다면?

1. 로컬 환경에서 소스 코드 수정 (예시)



```

1  # bfs_copy.py
2
3  myGraph = graph = {'A': ['B', 'C', 'D'],
4                    'B': ['A', 'E'],
5                    'C': ['A', 'F', 'G'],
6                    'D': ['A', 'H'],
7                    'E': ['B', 'I'],
8                    'F': ['C', 'J'],
9                    'G': ['C'],
10                   'H': ['D'],
11                   'I': ['E'],
12                   'J': ['F'],
13                   'K': ['A', 'B']}

```

bfs_copy.py의 라인 13을 아래와 같이 추가함

2. 터미널에 **git add .** 입력

3. `git status` 입력 시 `modified : 수정된 파일명` 으로 뜨면서 수정 사항 안내해줌



```
PS C:\rokey\study\week2> git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   bfs_copy.py
```

4. `git commit -m "second commit"` 으로 새롭게 수정된 히스토리 생성

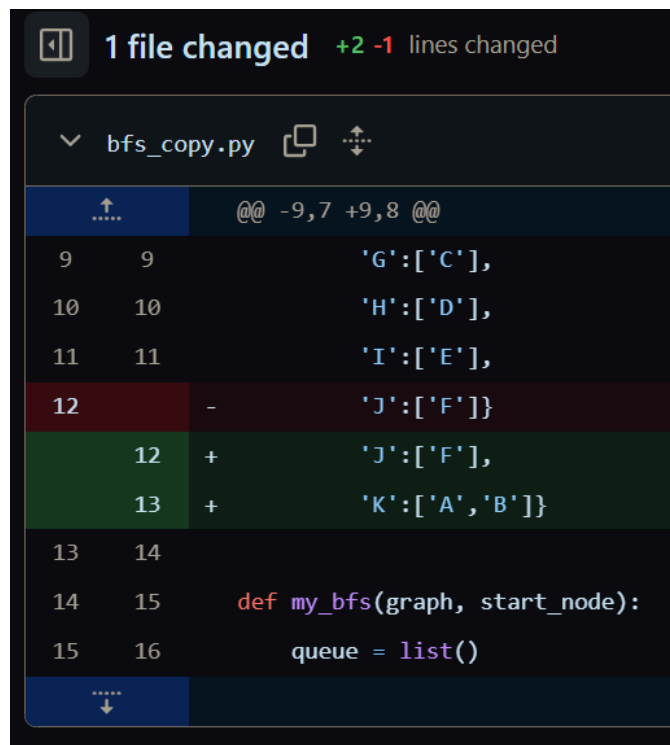
```
PS C:\rokey\study\week2> git commit -m "second commit"
[master 4c8e02d] second commit
1 file changed, 2 insertions(+), 1 deletion(-)
```

5. `git push origin master` 으로 로컬 프로젝트 상의 수정 사항을 깃허브 저장소에 반영(푸시)

6. 깃허브 저장소 새로고침하면 해당 수정 사항 히스토리 반영 됨.

 bfs_copy.py	second commit	5 minutes ago
 dfs_copy.py	first commit	44 minutes ago

수정 사항이 있었던 bfs_copy.py는 first commit 에서 second commit으로 히스토리 이름이 바뀌어 있다.(잘 반영 됨.)



second commit 클릭 시 변경 사항을 상세히 알려줍니다.

참고) VSCode 파일 색깔의 의미 🎨

: VSCode는 파일의 Git 상태에 따라 색상과 문자로 상태를 표시

- **U (초록색, Untracked)**

: 완전히 새로운 파일. Git이 "어? 이런 파일은 처음 보는데?"라고 인식하는 상태.

- **A (초록색, Added)**

: 새로운 파일을 이제 곧 commit 하겠다고 Staging Area에 추가한 상태.

`git add` 명령어를 사용하면 'U' 상태의 파일이 'A' 상태로 바뀜

- **M (노란색, Modified)**

: 기존에 있던 파일(commit 이력이 있는 파일)의 내용이 수정된 상태.

`bfs_copy.py` 파일의 코드를 한 줄이라도 바꾸고 저장하면 이 상태가 됨.

- **색깔 없음 (기본 색)**

: 변경 사항 없는 깔끔한 상태.

마지막 commit 이후로 아무런 수정도 일어나지 않은, Git이 이미 추적하고 있는 파일.