



GIT이 뭐길래 도대체 왜 우리를 괴롭히는가?

Git이 뭔가요?



Git이란 버전 관리 프로그램의 하나로써 현재 세계에서 가장 인기있는 버전 관리 툴
버전관리란 파일의 변화를 시간에 따라 추적하고 관리하는 것을 의미

개발을 하다가 작성한 프로그램이 동작하지 않아서 예전의 코드로 되돌리고 싶을 때, 테스트 코드를 짜보고 싶을 때, 다른 작업자들과 파일을 함께 수정할 때 등 다양한 상황에서 우리는 버전 관리 프로그램이 필요하게 된다.

다양한 코딩 프로그래밍을 진행하다 보면, 이 모든 파일들을 수정할 때마다 앞서 했던 방식처럼 새 파일을 만들어서 수정 사항을 관리한다면 쉽지 않은 일이 될 것이다.

이렇게 다양한 확장자의 파일들이 서로 복잡하게 연결되어있는 파일들을 관리하기 위해서는 기존의 파일 시스템(파일, 폴더)으로는 한계가 명확하며, 바로 이러한 이유로 수많은 개발자들이 코드 작업을 할 때 git을 통해 버전 관리를 하게 된다.

Github는 뭔가요?



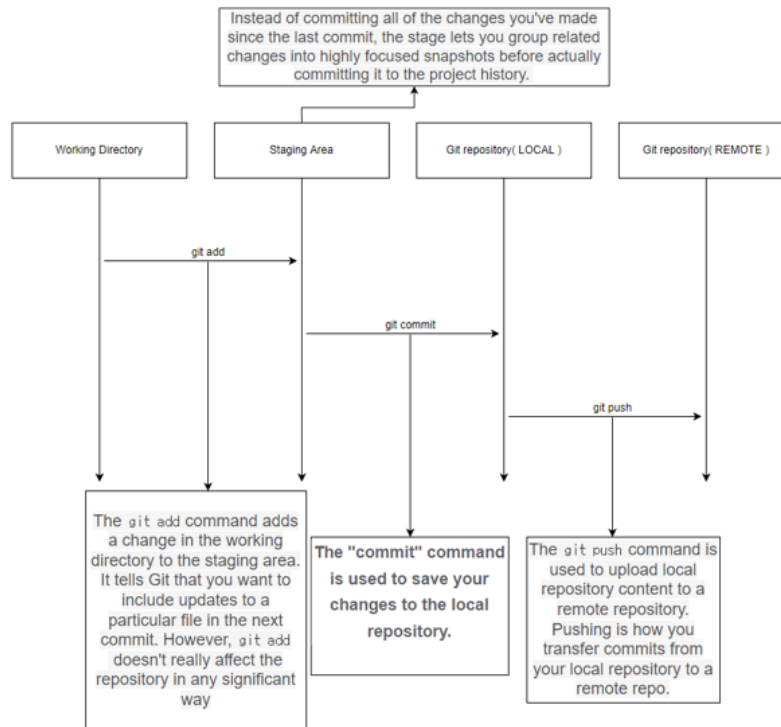
깃허브는 Git이라는 툴을 통해 관리되는 프로젝트들을 저장할 수 있는 저장소 서비스
Git 프로젝트를 위한 구글 드라이브, iCloud같은 서비스와 비슷한 개념

내 컴퓨터에 저장된 git을 사용하는 프로젝트를 깃허브에 업로드 한 뒤 동기화를 해주면 어느 PC에서도 손쉽게 소스코드를 받아서 작업할 수 있고, 나아가 팀 프로젝트를 진행할 때도 번거롭게 파일을 주고 받을 필요없이 깃허브를 통해서 수월한 협업이 가능하다. Git을 통해 관리되는 소스코드이기 때문에 수정 내역과 작성자 등 협업에 필요한 내용이 모두 추적된다.

Git의 작동 구조 → 필수 아님



GIT이 작동하는 구조는 다음과 같다.



Working Directory: 사용자의 작업 공간으로써, 로컬 저장소에 접근할 수 있으며, 실제 파일을 수정하거나 생성하는 공간

💡 untracked 상태: untracked

저장소 내에서 새로 만들어진 모든 파일들은 untracked 상태로 시작한다.

untracked 상태에서는 git이 코드 변경 이력을 추적하지 않는다.

💡 tracked 상태: unmodified, modified, staged

tracked 상태에서의 파일들은 git에 의해서 파일의 변경 이력이 추적된다.

모든 파일들의 변경 이력을 추적할 경우 시스템 부하가 커지며, 따라서 git에서 요청받는 파일들만 변경 이력을 추적하게 된다.

Staging Area: Working Directory에서 제출된 tracked 상태의 파일들을 관리 및 임시로 저장하는 공간

Working Directory와 실제로 저장하고 기록하는 공간 사이의 임시 공간으로 실제로 저장하고 기록하는 공간에서는 Stage Area에서의 파일 내용을 기반으로 변경된 차이점만 기록하게 된다.

커밋하려면 파일의 추적 상태 정보들만 기록, 즉 빠른 커밋을 하기 위해서 만들어진 공간이라고 볼 수 있다.

💡 stage 상태:

stage 영역에 들어온 파일들은 모두 tracked 상태의 파일

💡 unstage 상태:

파일 내에 변화가 있을 경우 해당 파일은 unstage상태라고 부른다.

stage 영역에서 나간 파일들은 모두 untracked 상태의 파일 다시 정리하자면, 깃이 버전 기록을 하기 위해서는 파일의 최종 상태가 반드시 stage 여야 한다.

Working Directory에서의 파일과 Stage Area에서의 파일 간 내용 차이가 존재하면, 해당 파일은 unstage 상태이다.

Local Git Repository: 파일들의 변경 내역을 저장하는 저장소 중에서 local에 위치하고 있는 저장소

Remote Git Repository: Local Git Repository에 있는 저장 내용들을 push 할 경우에 저장되는 원격 저장소

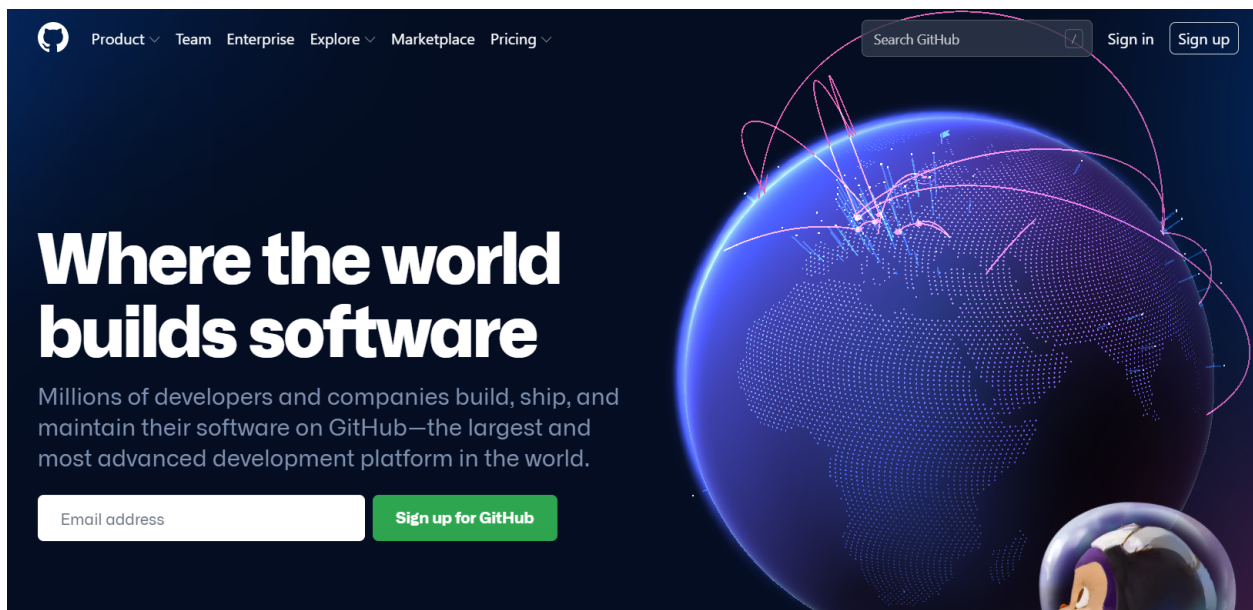
Git 설치 및 Github 계정 생성

💡 git bash 설치하기 및 유저 정보 설정

하단 링크에서 본인 운영체제에 맞는 git bash 설치 방법을 따라 설치한다.

<https://git-scm.com/book/ko/v2/시작하기-Git-설치>

💡 GitHub 가입하기!



우측 상단의 Sign in 눌러서 로그인 또는 Sign up 눌러서 회원가입!

Sign up : 회원 가입

- 원하는 편한 이메일 이용해서 회원가입!

Sign in : 로그인

- 이미 계정이 있다면 가볍게 로그인!



레포지토리 만들기

new 또는 start a project 버튼을 눌러서 레포지토리를 생성한다.

Recent Repositories

New

Find a repository...

New 눌러서 레포지토리 생성

Learn Git and GitHub without any code!

Using the Hello World guide, you'll create a repository, start a branch, write comments, and open a pull request.

Read the guide

Start a project

start a project 버튼 눌러서 레포지토리 생성

이름을 지정하고, public으로 생성한다. (**private**일 경우에는 사용자들만이 볼 수 있다.)

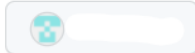
Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?

[Import a repository.](#)

Owner *

Repository name *



/

likelion_2022_git



Great repository names are short and memorable. Need inspiration? How about **shiny-succotash**?

Description (optional)



Public

Anyone on the internet can see this repository. You choose who can commit.



Private

You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

☐ Add a README file

This is where you can write a long description for your project. [Learn more.](#)

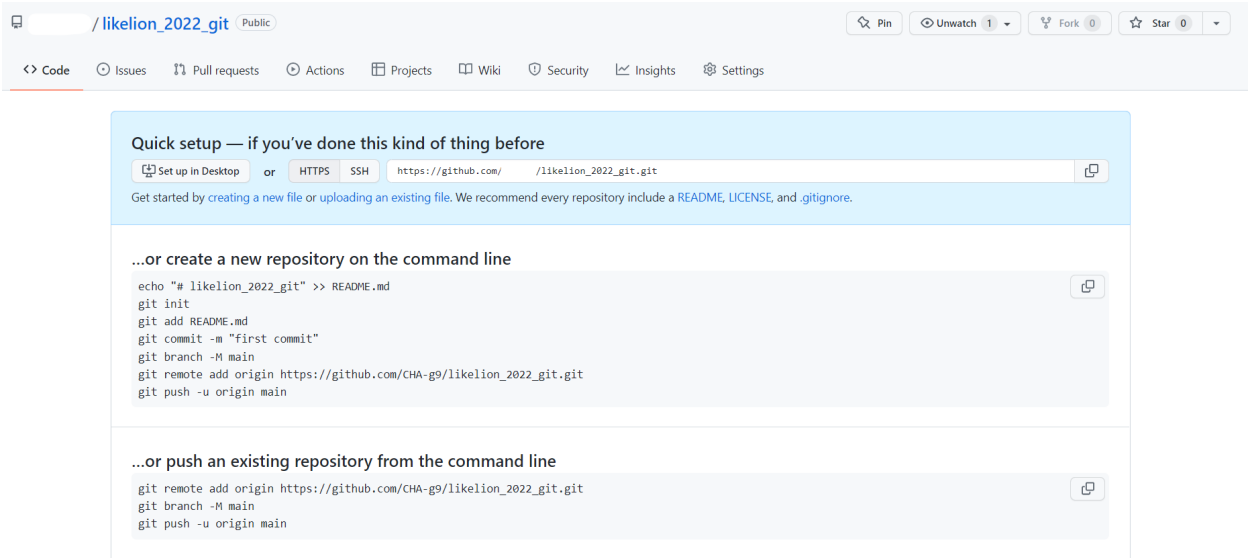
☐ Add .gitignore

Choose which files not to track from a list of templates. [Learn more.](#)

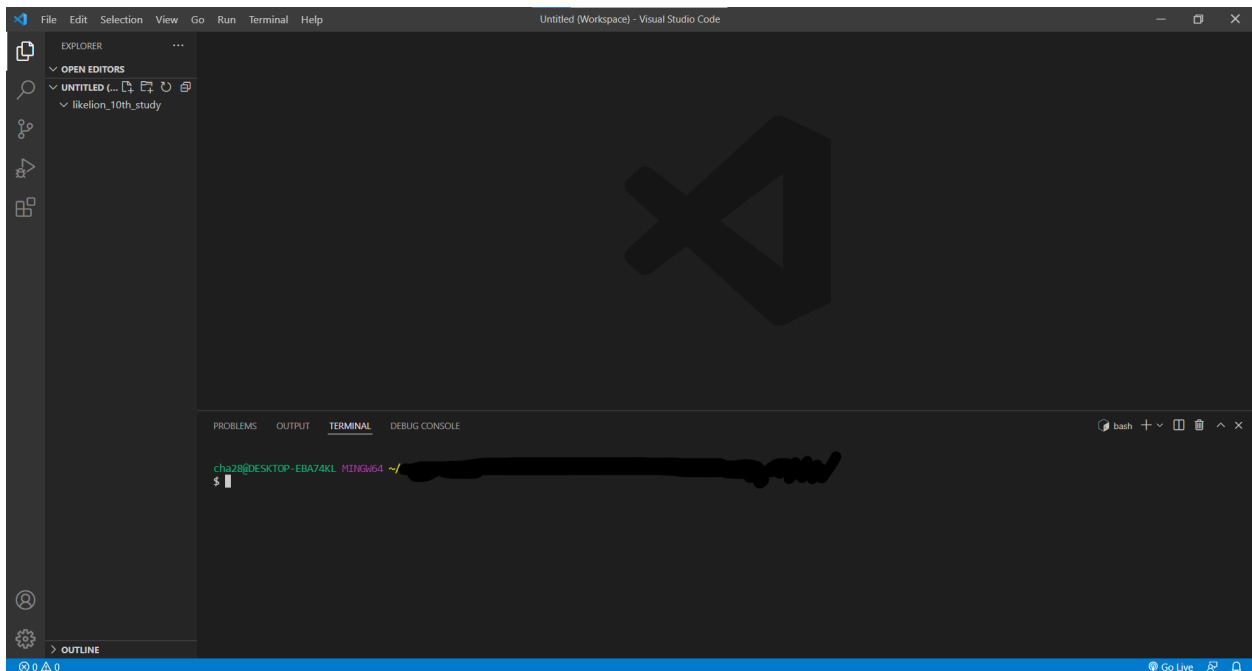


저장소 연결하기

레포지토리가 열리면 아래와 같은 화면이 뜨게되면, 이제 이 창을 띄워둔 상태에서 VSCode를 실행한다.



VSC를 열고, 터미널을 연다. 이 때 터미널이 bash 창인지 꼭 확인해야한다!



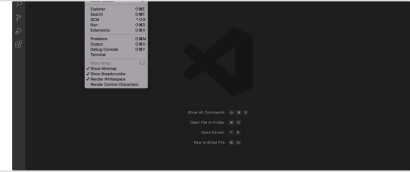
bash가 안 열린다면, 아래 토글 속 링크를 참고해서 열어주세요!

▼ Mac

[VSCode] 비주얼 스튜디오 코드에서 터미널 실행하기 for Mac OS

비주얼 스튜디오 코드(vscode)는 기본적으로 터미널 기능을 제공한다. 우리는 여기에 코드를 입력하여 원하는 것을 실행한다. 그런데 코드 입력이 실행으로 이어지기 위해서는 우선적으로 기능 설치가 필요하다. 아래 방법을 참고하여 터미널을 정상적으로 실행해보자.


 <https://make-some-wave.tistory.com/entry/vscode-terminal-mac>

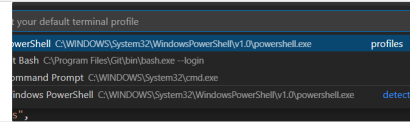


▼ Windows

visual studio code 기본 터미널 변경 방법(vsc 최신버전 반영)

현재 최신 버전의 visual studio code를 기준으로 설명 드리겠습니다. 아주 간단합니다!!

 <https://hoho325.tistory.com/313>



터미널이 열렸다면, 이제 연결하고 싶은 로컬 폴더를 열어주세요. `cd "파일 경로"` 를 터미널에 치면 이동할 수 있습니다.



명령어로 저장소를 연결

지금부터는 명령어로 저장소를 연결해보겠다.

1. **레포지토리 초기화**: 앞서 파일 경로 설정이 완료되었다면, 이제 `git init` 명령어를 입력해주세요. 이 명령어의 역할은 레포지토리를 초기화하는 것이다. 이 명령어를 입력한 후에는 연결해둔 로컬 폴더에 .git 형태의 파일이 생겨나는 것을 볼 수 있다.


```
cha28@DESKTOP-EBA74KL MINGW64 ~/OneDrive/바탕 화면/대학교/멋쟁이 사자처럼/10기 운영진/likelion_10th_study
$ git init
Initialized empty Git repository in C:/Users/cha28/OneDrive/바탕 화면/대학교/멋쟁이 사자처럼/10기 운영진/likelion_10th_study/.git/
```

2. **유저 설정**: `git config --global user.name "아이디"`, `git config --global user.email "GitHub 가입한 이메일 주소"` 를 이용해 유저를 설정해준다. GitHub에 만들어둔 내 레포지토리를 찾아내기 위한 과정이라고 생각하시면 된다.
3. **저장소 연결**: `git remote add origin "레포지토리 주소"` 를 이용해 로컬 저장소와 GitHub 상의 레포지토리를 연결해주는 과정이다. 이 때 레포지토리 주소는 앞서 만들어 둔 나의 레포지토리에서 복사, 붙여넣기 하는 것이 좋다. 하나라도 틀리면 오류가 날 수도 있기 때문이다.
4. **브랜치 생성**: 필수는 아니지만, 브랜치를 생성할 줄 알아야 한다. 팀프로젝트에서는 거의 필수적이라고 볼 수 있다. `git branch -M main` 명령어를 입력해주세요! 처음 연결했을 때와 달리 뒤 괄호 부분에 쓰인 말이 달라지는 것을 볼 수 있습니다. 내 레포지토리와 연결되는 새로운 브랜치가 생성되었다는 뜻입니다.

▼ 여기서 잠깐, 브랜치가 뭔가요?

GitHub는 세이프포인트 생성 및 저장이 목적이기도 하지만 협업의 목적도 가진 저장소이다. 다시 말하면 여러 사람이 작성한 코드를 한 곳에 모아 쉽게 관리하고자 하는 것이 목적인 시스템이라는 의미다.

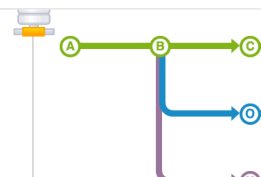
같은 코드를 버전관리 할 때, 업로드 할 때처럼 다양한 협업 상황의 혼선을 방지하기 위해 GitHub는 사람마다 업로드 하는 길을 달리 만들어주는데, 이 길이 바로 **‘브랜치(Branch)’**이다!

 더욱 자세한 설명은 아래 링크를 참고!

브랜치란? 【브랜치 (Branch)】 | 누구나 쉽게 이해할 수 있는 Git 입문~버전 관리를 완벽하게 이용해보자~ | Backlog

지금까지 Git의 기본적인 사용법에 대해 알아 보았습니다. 발전 편에서는 브랜치의 사용법에 대해 좀 더 자세히 알아보도록 하겠습니다. 소프트웨어를 개발할 때에 개발자들은 동일한 소스코드를 함께 공유하고 다루게 됩니다. 동일한 소스코드 위에서 어떤 개발자는 버그를 수정하기도 하고 또 다른 개발자는 새로운 기능을 만들어 내기도 하죠.

 https://backlog.com/git-tutorial/kr/stepup/stepup1_1.html



```
cha28@DESKTOP-EBA74KL MINGW64 ~/OneDrive/바탕 화면/대학교/멋쟁이 사자처럼/10기 운영진/likelion_10th_study (master)
$ git config --global user.name "CHA-g9"

cha28@DESKTOP-EBA74KL MINGW64 ~/OneDrive/바탕 화면/대학교/멋쟁이 사자처럼/10기 운영진/likelion_10th_study (master)
$ git config --global user.email "cha2889@naver.com"

cha28@DESKTOP-EBA74KL MINGW64 ~/OneDrive/바탕 화면/대학교/멋쟁이 사자처럼/10기 운영진/likelion_10th_study (master)
$ git remote add origin https://github.com/CHA-g9/likelion_2022_git.git

cha28@DESKTOP-EBA74KL MINGW64 ~/OneDrive/바탕 화면/대학교/멋쟁이 사자처럼/10기 운영진/likelion_10th_study (master)
$ git branch -M main
```

여기까지 따라 왔다면, 이제 파일을 올릴 일만이 남았다.



원하는 파일을 원격 저장소로 올리기

파일을 올리는 단계에서 기억하실 것은, **add - commit - push**의 3단계입니다. 이것까지만 다 성공하시면, 아마 나중엔 여러분도 기계처럼 git을 활용할 줄 알게 되는 것이다. 먼저 간단하게 용어 설명하겠다. (토글을 열어 확인해주세요)

▼ Add란?

더한다는 뜻 그대로, 원격 저장소에 올리고 싶은 파일들을 묶어두는 것이다. 이를 스테이지에 올린다고도 표현한다. 올리고 싶은 파일은 무조건 add로 묶여야한다.

▼ Commit이란?

게임에서 세이프 버튼을 누르는 것과 같다. 묶어둔 파일을 잠시 저장하는 것이다. Commit을 하고나면 언제든지 Commit을 한 시점으로 돌아갈 수 있다.

▼ Push란?

commit은 기본적으로 컴퓨터에 저장되기에 날아가기 전에 더욱 안전한 우리의 원격 저장소, GitHub에 업로드해둘 필요가 있다. 이 업로드 명령을 내리는 명령어가 바로 push이다.

자, 그럼 터미널에 이제 입력해보자!

```
cha28@DESKTOP-EBA74KL MINGW64 ~/OneDrive/바탕 화면/대학교/멋쟁이 사자처럼/10기 운영진/likelion_10th_study (main)
$ git add .

cha28@DESKTOP-EBA74KL MINGW64 ~/OneDrive/바탕 화면/대학교/멋쟁이 사자처럼/10기 운영진/likelion_10th_study (main)
$ git commit -m "first commit"
[main (root-commit) 79caf87] first commit
1 file changed, 14 insertions(+)
create mode 100644 1st.html

cha28@DESKTOP-EBA74KL MINGW64 ~/OneDrive/바탕 화면/대학교/멋쟁이 사자처럼/10기 운영진/likelion_10th_study (main)
$ git push origin main
```

1. **스테이지에 파일 올리기** : `git add .` 으로 원격 저장소에 올리하고자 하는 파일들을 스테이지로 올려준다. 여기 `.` 을 사용한 경우는 모든 파일을 올리겠다는 의미이고, 모든 파일을 올리고 싶지 않은 경우는 파일명을 선택해 add 하면 된다.
2. **Commit 만들기** : `git commit -m 'commit 이름'` 으로 commit을 생성한다. 이 때 'commit 이름'은 원하는대로 지정해도 되지만, 이번 작업의 설명이 될만한 내용을 적는 것이 좋다. 예를 들어, 이번 작업에 로그인 기능을 만들었다면 commit 이름을 'login build' 로 지정하는 것이다. 꼭 그렇게 하실 필요는 없지만 원활한 버전 관리를 위한 작은 팁이다.
3. **원격 저장소에 업로드하기** : `git push -u origin '브랜치 이름'` 을 이용해 나의 GitHub 레포지토리에 파일을 올린다. 이 때 브랜치명은 앞서 브랜치 생성 시에 생성한 브랜치를 의미한다.

그러나, 여기서 여러분은 오류를 마주하게 됩니다.(두둥)

```
$ git push origin main
remote: Invalid username or password.
fatal: Authentication failed for 'https://github.com/CHA-g9/likelion_2022_git.git/'
```

바로 위 사진과 같은 오류일 것이다.. 여기서 활용하는 것이 **토큰**이다!



token 생성 방법

▼ GitHub 인증 토큰 생성하기

토큰은 터미널에서 GitHub과 로컬 디렉토리를 연결할 때 인증하는 수단으로 활용한다. 따라서 이 단계에서 생성하신 토큰은 보안과 직결되기 때문에 꼭 안전한 곳에 보관해두셔야한다는 것, 기억하자!

- i. 로그인 후 Settings - Developer settings - Personal access tokens로 접속한다.
- ii. Generate new token 버튼을 눌러 토큰을 생성한다.

New personal access token

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

Note

likelion_token_2022

What's this token for?

Expiration *

30 days

The token will expire on Sun, Apr 24 2022

- 토큰 생성 시에는 토큰 이름과 유효기한을 지정하여야합니다. 이름은 마음대로 선택하셔도 되지만, 유효기한의 경우 무한대로 설정하실 수 없으니 원하시는 적당한 기간을 선택해주세요.

Select scopes

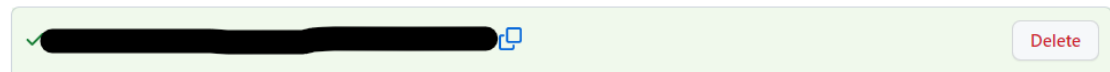
Scopes define the access for personal tokens. [Read more about OAuth scopes.](#)

<input checked="" type="checkbox"/> repo	Full control of private repositories
<input checked="" type="checkbox"/> repo:status	Access commit status
<input checked="" type="checkbox"/> repo:deployment	Access deployment status
<input checked="" type="checkbox"/> public_repo	Access public repositories
<input checked="" type="checkbox"/> repo:invite	Access repository invitations
<input checked="" type="checkbox"/> security_events	Read and write security events
<input checked="" type="checkbox"/> workflow	Update GitHub Action workflows
<input checked="" type="checkbox"/> write:packages	Upload packages to GitHub Package Registry
<input checked="" type="checkbox"/> read:packages	Download packages from GitHub Package Registry
<input checked="" type="checkbox"/> delete:packages	Delete packages from GitHub Package Registry
<input checked="" type="checkbox"/> admin:org	Full control of orgs and teams, read and write org projects
<input checked="" type="checkbox"/> write:org	Read and write org and team membership, read and write org projects
<input checked="" type="checkbox"/> read:org	Read org and team membership, read org projects
<input checked="" type="checkbox"/> admin:public_key	Full control of user public keys
<input checked="" type="checkbox"/> write:public_key	Write user public keys
<input checked="" type="checkbox"/> read:public_key	Read user public keys
<input checked="" type="checkbox"/> admin:repo_hook	Full control of repository hooks

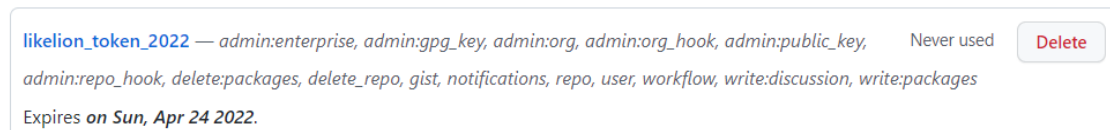
- 하단에 권한을 설정할 수 있는 체크리스트들이 등장하는데요! 우선 모든 것을 다 선택해주시면 된다! (나중에 이 기능들을 일일이 알아보고 나서는 원하는 권한만 선택해주셔도 무방하다!)

Tokens you have generated that can be used to access the [GitHub API](#).

Make sure to copy your personal access token now. You won't be able to see it again!



여기서 네모 2개 겹친 모양을 클릭해서 복사를 하시고!



그 후엔 이렇게 닫아서 다시 볼 수 없습니다!

- 이렇게 생성된 토큰은 복사해서 메모장에 꼭!!!! 붙여넣어 주자!!!!!! 왜냐하면 토큰은 한 번 보여주고 다시는 보여주지 않기 때문이다. 실수로 못 했거나, 안 하신 경우는 기존 토큰을 삭제하고, 새 토큰을 생성하면 되니 걱정하진 말자. (하지만 조금 귀찮겠조....?)

토큰을 아래 방식과 같이 입력하면, 레포지토리에 push가 가능해진다.

▼ Windows

[Github] 윈도우 git credential - access token 적용하기

8월 13일 부터 깃헙의 비밀번호 방식 인증이 사라지고, personal access token을 이용해야 한다는 메시지가 떴다.

<https://velog.io/@rimo09/Github-%EC%9C%88%EB%8F%84%EC%9A%B0-git-credential-access-to-ken-%EC%A0%81%EC%9A%A9%ED%95%98%EA%B8%B0#%EC%9C%88%EB%8F%84%EC%9A%B0%EC%97%90-github-%EC%9E%90%EA%B2%A9%EC%A6%9D%EB%AA%85-%ED%86%A0%ED%81%B0-%EC%84%A4%EC%A0%95%ED%95%98%EA%B8%B0>

```
password authentication was removed in favor of a personal access token instead.
https://github.blog/2020-12-15-to-git-operations/ for more information
Access 'https://github.com/amomo-de URL returned error: 403
```

<요약>

1. 제어판 - 사용자 계정 - windows 자격 증명 관리자 탭으로 이동한다.
2. [git:https://github.com](https://github.com) 의 자격 정보를 찾아 편집 버튼을 클릭한다.
3. 암호에 발급된 access token 을 붙여넣고 저장한다. 끝!

▼ Mac

맥 OS: git push에서 The requested URL returned error: 403 해결 방법

Git Hub에서 액세스 토큰 방식으로 바뀌면서 인증에 어려움을 겪을 수 있다. 일단 액세스 토큰 생성 방법은 아래 글을 참고하기 바란다. GitHub 토큰 인증 로그인: Personal Access Token 생성 및 사용 방법 혹시 MAC OS환경에서 git push 실행시, 아래와 같은 로그가 뜨며 잘 안 되는 경우가 발생할 수 있다.

https://curryou.tistory.com/403



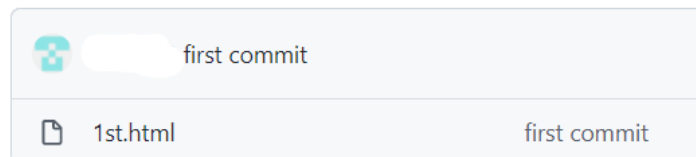
<요약>

1. Spotlight 검색에서 keychain(키체인 접근)을 찾아서 실행
2. "github" 를 검색 후, 종류가 "인터넷암호"인 항목을 더블 클릭
3. 암호보기 항목을 체크후, 기존의 패스워드를 발급받은 Access Token 으로 변경한다.

해결~!

```
cha28@DESKTOP-EBA74KL MINGW64 ~/OneDrive/바탕 화면/대학교/멋쟁이 사자처럼/10기 운영진/likelion_10th_study (main)
$ git push origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 494 bytes | 247.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/CHA-g9/likelion_2022_git.git
* [new branch]      main -> main
```

이렇게 원하는 파일이 올라갔다면, 완료된 것이다.



협업에서 GITHUB를 효율적으로 사용하는 방법

git의 역할은 다름이 아닌, 버전 관리 서비스!! 서로의 작업 버전이 잘 유지되면서, 함께 협업을 할 수 있는 것이 제일 중요하다!!

세가지로 기억하자. GIT을 복제해오고, 자신만의 BRANCH를 제작하고, 변경사항을 PULL해온다.



git clone: repository를 복제해오기

자신이 복제해오고 싶은 repository 주소를 통해서 repository를 자신의 파일에 넣는다.

```
git clone "repository 주소" ## 자신이 가져오고 싶은 git repository 주소를 입력해서 가져온다.
```



git branch: 현재 BRANCH위치 확인과 새로운 BRANCH 제작하기

기본적으로 git branch 명령어를 사용하면, 현재 우리가 어떤 branch에 있는지를 알 수 있다.

main branch에서 시작을 하고 있을텐데, 우리는 자신의 개인 local branch를 만들어 사용해야 한다.

```
git branch ## 현재 어떤 branch가 존재하며, 어떤 branch에 내가 접근해 있는지를 알 수 있다.
```

```
git branch "내가 만들고 싶은 branch 명" ## 새로운 이름으로 branch를 만들 수 있다.
```

위의 코드를 했다고 해서 BRANCH가 변경된 것이 아니다. 반드시 아래 과정까지 진행 이후에 자신의 작업을 진행하도록

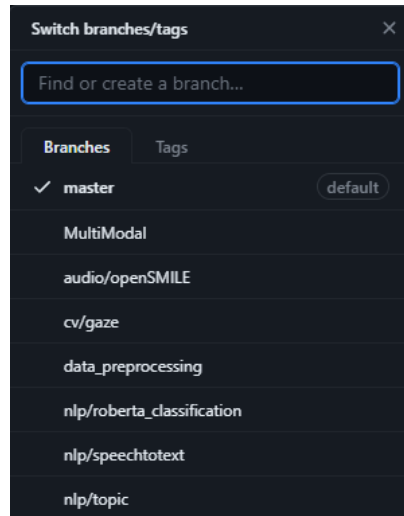


git checkout : 자신이 원하는 BRANCH로 이동하기

git branch를 사용해보자. 아직도 main branch에서 이동하지 않았음을 알 수 있다. 또한 새롭게 보이는 branch(자신이 직전에 만든 branch)가 존재할 것이다. 이제는 그곳으로 이동하도록 하자!!

```
git checkout "이동하고 싶은 브랜치명" ## 이동하고 싶은 브랜치로 이동한다.
```

추가적으로 checkout은 중요한 역할을 한다. Clone 이후에 레포지토리는 master branch가 불러와질 것이다. 그런데 git hub에는 다양한 branch가 존재할 수 있다. 아래와 같이 말이다.



master branch가 아닌 다른 branch에 있는 내용을 가져와서 작업하고 싶을 경우에도 또한 checkout 명령어를 사용할 수 있다.

```
# 현재 master branch라고 가정
git branch
## master -> 하나밖에 나오지 않는다. 현재 로컬 내에 브랜치가 하나기 때문

# MultiModal로 이동해보겠다.
git checkout MultiModal ## 디렉토리들이 변환되고, bash에서 현재 위치가 MultiModal임을 표현할 것이다.
```



git pull: 외부에서 변경된 내용 사항을 가져오기

위에서 자신의 결과물을 git에 저장하는 방법은 배웠다. `add - commit - push` 를 사용하면 간단히 올릴 수 있었다.

push를 해서 자신의 branch를 repository에 올린 이후에 이제 다른 사람과의 코드를 합치고(혹은 파트너와 같은 branch에서 작업을 했을 경우에도 포함이다.) 자신이 작업을 하기 이전에 상대방의 코드를 가져와서 최신 버전에서 작업을 진행해야 할 것이다!!

즉 외부에서 변경된 내용 사항을 가져오기 위해서(자신보다 이후의 버전을 가져오기 위해서) 사용하는 코드가 바로 pull이다.

```
git pull ## 간단하게 git pull 커멘드를 통해서 끌어올 수 있다. 그렇게 되면 최신에 저장된 git을 가져갈 수 있다.
```

5. git merge



git merge: 다른사람이 작업한 브랜치와 자신의 브랜치를 합치는 과정

git merge에 대해서는 간단히 설명만 하겠다. 각자의 branch에 있어서 작업을 하게 되면, 그걸 이제는 합쳐야 하는 시간이 발생할 것이다. 그 때 바로 이 git merge를 사용한다. 요즘은 GUI가 많이 발전되어 있어서 Git Kraken과 같은 여러 GUI 툴을 사용하여 작업을 진행한다.

또한, 같은 branch에서 작성하는 merge가 있었을 경우에, 서로 간의 버전이 맞지 않을 수 있는데, 그 경우에는 merge가 자동으로 작동하여, 그 페이지로 넘어가게 된다.

git merge 가 작동하는 방식은 줄 수를 통해서 확인하게 되는데, 같은 줄에 다른 코드가 있을 경우에, 두개를 모두 수용할 것인가, 아니면 둘중에 하나를 수용할 것인가 혹은 둘다 기각할 것인가를 정할 수 있다. 여기서 주의해야 하는 점은 101 - 105 번째 줄이 다른 repository 에서는 99 - 103번째 줄에 존재한다고 하더라도, 이를 인식하지는 못한다. 그래서 각자 이에 대해서 유의하고 작업하는 것이 중요하다.



Merge 여부 필터링하기

```
# Checkout한 브랜치를 기준으로 --merged, --no-merged 옵션을 사용하여 merge가 된 브랜치인지 아닌지 필터링
git branch --merged
git branch --no-merged
```



Merge하기

```
#예를 들어 master브랜치와 test 브랜치가 있다고 했을 경우,
##**git merge test**를 하게되면
#test브랜치에만 있던 코드가 master브랜치에 병합된다.

# master에 체크아웃
git checkout master
# test브랜치의 코드를 master에 합침
git merge test

git merge [브랜치명]
```



Merge 시 충돌 해결하기

merge conflict

merge할 때 발생하는 conflict(충돌)

에러를 안내느냐가 아닌 해결할 수 있느냐가 중요함

충돌이 생길 경우

git은 충돌 내용을 하단과 같이 코드 상에 보여준다.

```
<<<<<<< HEAD
master content -> 현재 브랜치[master]에서 수정된 내용
=====
test content -> 머지할 브랜치[test]에서 수정한 내용
>>>>>>> 충돌나는 브랜치명 또는 commit 아이디

## 즉 여기서 서로다른 부분을 수정해주어야 한다.
## vs code에서는 ui를 통해서 해결이 가능하다.
```

충돌 만나도록 하는 명령어

충돌이 나지 않게 하는 코드들이다. 세부적인 내용은 인터넷에서 찾아보도록 하자.

```
# 대상 브랜치로 이동
git checkout [대상브랜치]

# 대상 브랜치의 로컬 최신화
git pull origin [대상브랜치]

# 다시 내 작업 브랜치로 이동
git checkout {작업 브랜치}

# 머지 요청
```

```
git merge [대상브랜치]
```

```
# 수정 후, add, commit, push 진행
```