

실습03주차_Git_(최종)



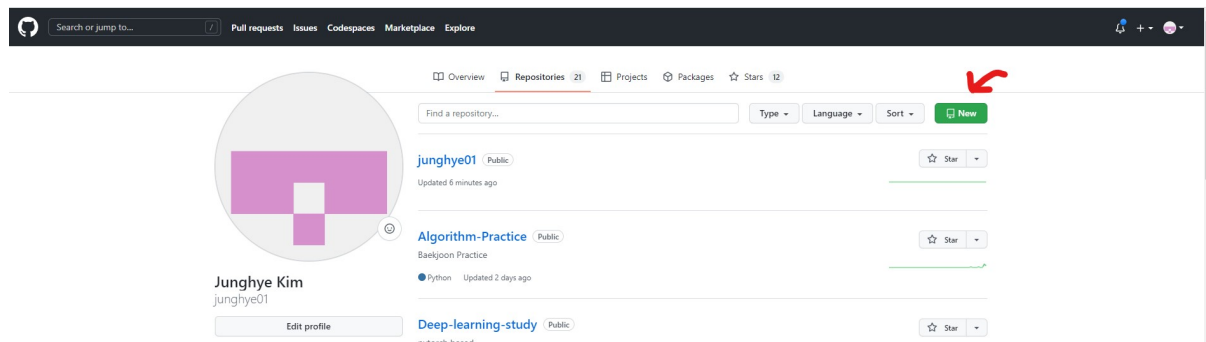
목차

1. Git Repository 생성
2. Git Workflow 명령어
3. Git Fetch & Pull
4. Git branch
5. Git 실습

1. Git Repository 생성

<https://github.com/> 에 방문하여

우측 상단 프로필 > Your repositories > new를 누른 뒤,




아래의 정보들을 모두 작성하여 레포지토리를 생성함.


- Repository name: 현재 생성하는 저장소 이름.
- Description: 레포지토리의 설명란으로 선택사항.
- Public / Private: 레포지토리의 공개/비공개 여부 결정.
- README file: 레포지토리를 설명을 자세하게 작성할 수 있는 파일로, 레포지토리를 잘 구성하고 싶다면, 생성해야함.
- .gitignore: 레포지토리에 파일들을 업로드할 때, 해당 gitignore에 적힌 파일들은 제외하고 업로드됨. 보통은 프로그램 실행과 관계 없는 파일들로 설정해둠.
- license: 이론시간에 배운 오픈소스 라이선스와 관련된 설정.

Owner *

Repository name *

 junghye01 ▾


/


ossprac 

Great repository names are short and memorable. Need inspiration? How about [laughing-disco](#)?

Description (optional)

for ossprac hw

☒  **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.](#)

.gitignore template: Python ▾

Choose a license
A license tells others what they can and can't do with your code. [Learn more.](#)

License: Apache License 2.0 ▾

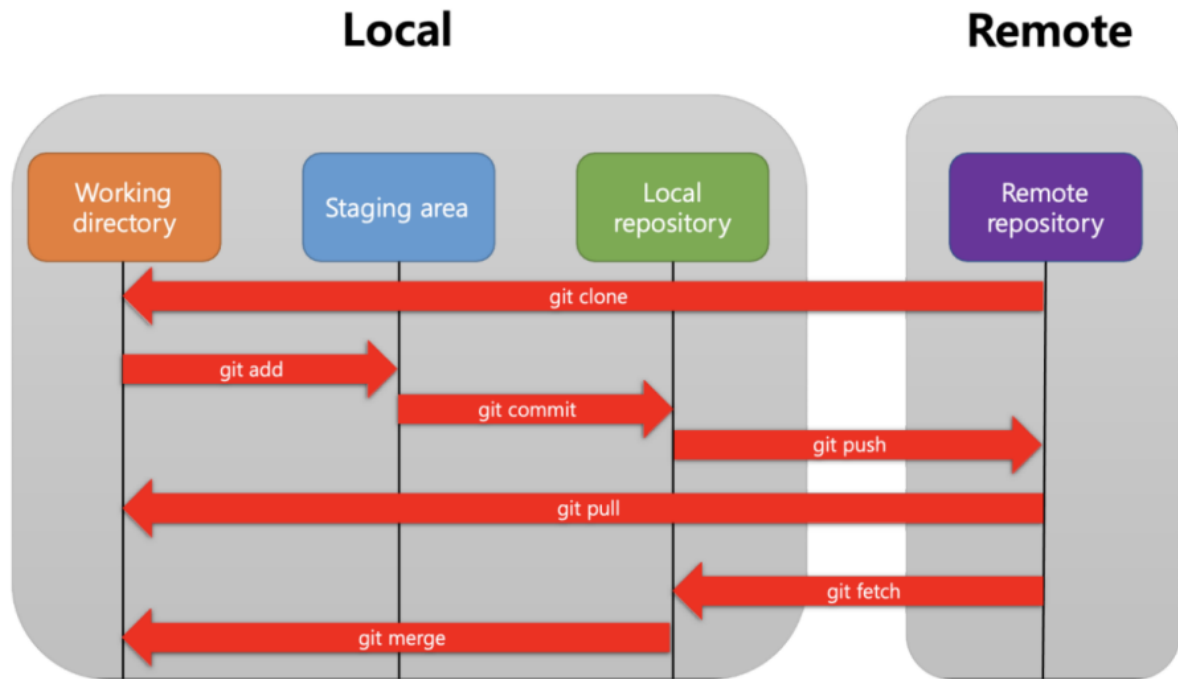
This will set  **main** as the default branch. Change the default name in your [settings](#).

 You are creating a public repository in your personal account.

Create repository

2. Git Workflow 명령어

add → commit → push를 사용해서 local에서 remote git 서버로 코드 파일을 저장하게 된다.



working directory : 실제로 개발자가 작업하는 공간

staging area : git add를 통해 변경사항이 1차로 저장되는 공간

local repository : 1차로 변경된 사항들이 git commit을 통해 최종적으로 저장되는 공간이며 Local repository에 있는 commit들이 git push를 통해서 remote repository로 이동됨

remote repository : 네트워크 상의 다른 위치에 존재하며 여러 사람이 함께 공유할 수 있는 공간

git clone

: remote repository에 저장된 저장소(레포지토리)를 내 working directory(노트북)으로 다운로드 받는 명령어.

```
git clone [복사할 레포지토리 URL]
```

e.g. `git clone https://github.com/junghye01/ossprac.git`

이 명령은 “ossprac” 라는 디렉토리를 만들고 그 안에 .git 디렉토리를 만들고
그리고 저장소의 모든 데이터를 가장 최신 버전으로 가져옴

만약, local 폴더 이름을 다르게 저장하고 싶을 경우

```
git clone https://github.com/junghye01/ossprac.git ossprac2
```

이렇게 하면 “ossprac2” 라는 이름의 폴더로 local에 저장됨

git add

: 기존 서버에 업로드된 코드와 비교해서 현재 working directory에서 일어난 변경 사항을 staging area로 옮기는 명령어. 1차 저장.

```
git add [추가할 파일명]
```

e.g. `git add .` : 현재 경로의 모든 파일을 저장하는 명령어

e.g. `git add main.py` : 이렇게 파일명을 지정하여서 저장 가능

git commit

: staging area에 있는 변경 사항을 local에 존재하는 git의 저장소로 옮기는 명령어.

```
git commit -m "[커밋 메세지]"
```

e.g. `git commit -m "ossprac에 커밋하는 메세지입니다."`

git push

: local 저장소에 저장된 변경 사항들을 최종적으로 remote 저장소로 옮기는 명령어. github 웹사이트에서 변경사항을 확인할 수 있음.

```
git push [remote repo 이름] [branch 이름]
```

e.g. `git push origin main`

git pull

: 현재 존재하는 working directory에 존재하는 파일들과 remote repository에 존재하는 파일들의 변경 사항을 확인하고 업데이트하는 명령어.

```
git pull [remote repo 이름] [branch 이름]
```

e.g. `git pull origin main`



git clone VS git pull

git clone은 현재 내 working directory(노트북)에 작업할 repository와 관련된 파일이 아무것도 없을 때, 폴더를 통째로 다운받는 것. 레포지토리를 통채로 zip 파일로 다운받는 것과 유사함.

git pull은 내가 현재 작업한 파일들이 내 노트북에 존재할 때, 변경된 사항들만 가져오고 싶을 때 사용하는 명령어.

git status

: 각 과정을 진행하는 중간마다 해당 명령어를 사용하여 git 상태를 확인할 수 있음.

```
git status
```

- git은 파일을 3가지 상태로 관리함
 - modified : 수정했지만 staging area에 저장되지 않은 상태
 - staged : 수정할 파일을 곧 커밋할 것이라고 표시한 상태
 - committed : 데이터가 로컬 git 데이터베이스에 안전하게 저장되어 있다는 뜻

git log

: commit 현황을 보여주는 명령어

```
git log
```

3. Git Fetch & Pull

git fetch

: remote와 local 사이의 변경 사항을 가져오는 것

```
git fetch [remote 이름] [branch 이름]
```

git merge

: fetch의 output(변경 사항)을 사용하여 working directory(local)에 저장된 코드에 반영하여 변경하는 것

```
git merge [remote 이름] [branch 이름]
```

git pull

: pull = fetch + merge. 즉, remote와 local 사이의 변경사항을 가져와 local 코드를 변경하는 것

4. Git branch



Branch는 뭘까?

현재 commit에 대한 코드를 통째로 복사해서 다른 branch의 코드와 상관없이 독립적인 개발 공간을 제공하는 도구!

main branch 는 오류없이 최종 코드가 저장되는 공간

실험적인 코드는 새로운 독립적인 branch에서 작업을 진행하고 문제가 생길 경우 branch를 떼어버리면 되고 문제가 없으면 그 branch를 main에 merge하면 됨

branch 생성

```
git branch [브랜치 이름]
```

branch 이동

```
git checkout [브랜치 이름]
```

branch 생성+이동

```
git checkout -b [브랜치 이름]
```

branch 삭제

```
git branch -d [브랜치 이름]
```

branch 조회

```
git branch
```

5. Git 실습

1. 위에서 만든 레포지토리를 clone합니다.

```
git clone https://github.com/junghye01/ossprac.git ossprac2
```

`cd ossprac2` 로 ossprac2로 디렉토리를 변경해주고

본인이 저장한 폴더 위치에 존재하는 “ossprac2” 폴더를 열어줍니다.

clone 후 git 상태

```
SAMSUNG@DESKTOP-NUC55SE MINGW64 ~/Documents/ossprac2 (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
(base)
```

2. README.md 에는 본인의 이름과 학번을 넣어 수정하고, “hello world”를 출력하는 간단한 main.py 파일을 제작합니다.

- 수정 & 새 파일 main.py 를 생성 후 git status 확인

```
SAMSUNG@DESKTOP-NUC55SE MINGW64 ~/Documents/ossprac2 (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   README.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        main.py

no changes added to commit (use "git add" and/or "git commit -a")
(base)
```

clone할 때 부터 있던 README.md는 modified 상태, 새로 생성한 파일은 untracked 파일로 분류됨!

1. 본인의 이름과 동일한 branch를 생성합니다.

```
git branch junghye
```

```
git checkout junghye
```

or

```
git checkout -b junghye
```

로 생성과 이동을 동시에 하는 방법도 있음

```
SAMSUNG@DESKTOP-NUC55SE MINGW64 ~/Documents/ossprac2 (main)
$ git checkout junghye
Switched to branch 'junghye'
M       README.md
(base)
SAMSUNG@DESKTOP-NUC55SE MINGW64 ~/Documents/ossprac2 (junghye)
$
```

2. 변경 사항을 add → commit → push 합니다. 이때 커밋 메시지는 “이름 학번”의 형태로 합니다.

```
git add .
```

```
SAMSUNG@DESKTOP-NUC55SE MINGW64 ~/Documents/ossprac2 (junghye)
$ git status
On branch junghye
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   README.md
        new file:   main.py

(base)
```

```
git commit -m "김정혜 2020112092"
```

```
git push origin main
```

The screenshot shows the GitHub interface for the repository 'junghye01 / ossprac'. The repository is public. The main branch is 'junghye', which is 1 commit ahead of the 'main' branch. A recent push by 'junghye' is shown, with a 'Compare & pull request' button. Below this, a table lists the files in the commit: .gitignore, LICENSE, README.md, and main.py. The README.md file is expanded, showing the text 'ossprac' and 'for ossprac hw 2020112092 김정혜'.

File	Commit Message	Time
.gitignore	Initial commit	1 hour ago
LICENSE	Initial commit	1 hour ago
README.md	김정혜 2020112092	3 minutes ago
main.py	김정혜 2020112092	3 minutes ago

README.md

ossprac

for ossprac hw 2020112092 김정혜

변경사항이 remote repository에 반영된 것을 볼 수 있음

추가 설명

Git과 Github 이론 내용은 교수님께서 수업시간에 설명해주신다고 하셨지만, 실습 이해를 돕기 위해 추가 설명을 남깁니다.

Git과 Github란?

Git

- 버전 관리 시스템 중 하나로, 소프트웨어 개발 등에서 **코드의 버전을 관리**
 - 버전을 관리한다는 것은, 파일의 변경 이력이나 수정 사항등을 기록한다는 것
 - 이전 버전으로 돌아가거나, 시간에 따라 수정된 내용 비교 가능
- 자신이 작업한 코드를 Command Line Interface(CLI) 방식으로 저장 가능

Github

- Git을 기반으로 하는 코드 저장소
- 간편한 Graphic User Interface(GUI) 제공
 - 깃으로 관리되는 프로젝트를 깃허브에 등록하면 웹 인터페이스를 통해 소스코드의 변경된 내용 확인
- 협업 하기 편리함
 - 동시에 하나의 파일을 수정하는 게 가능하기 때문에 병렬적으로 개발할 수 있음
 - 협업 프로젝트를 진행할 때는 다른 사람들이 올린 내역을 다운받고(pull), 내가 기능을 추가하거나 변경해서 작업한 이력들을 전송(push) 하는 방식으로 진행됨



GIT에서 Staging area가 필요한 이유

1. 일부분만 commit하여 세밀한 버전 관리 가능

버전 관리를 세밀하게 하기 위해서이다. git은 파일을 버전으로 관리한다. 버전이 기록되기 위해서는 개발자들이 커밋이라는 걸로 기록을 남겨줘야 한다. e.g. `git commit -m 'ADD 블라블라'` 또는 `git commit -m 'Modify 블라블라'` 이 때 한 번에 커밋을 하면 세밀한 버전 관리가 어렵다.

예를 들어, 프로젝트에 파일 A,B,C 가 있다고 해보자. A,B에는 같은 기능을 추가하고 C에는 다른 기능을 추가했다. staging area는 **일부분만 commit**하는 걸 가능하게 해준다. A,B만 일단 staging area에 올리고 커밋하고, C는 그 다음에 커밋을 해서 기능별로 관리하는 것이다.

이렇게 하면 나중에 파일 C에 있는 기능만 빼버리고 싶을 때 유용하다.

2. 충돌을 해결할 때

충돌이 발생한 일부 파일만 수정하여 그 부분만 add 하여 커밋하면 된다

3. commit을 다시 할 때

`git commit --amend` 를 통해서 최근 커밋 기록을 수정할 수 있다. 이 때, 커밋 기록만 수정하는 게 아니라 파일들도 좀 고칠 때, `commit --amend` 전에 파일들을 고쳐서 Staging area에 add하면 된다.

즉, staging area가 존재하지 않았다면, commit하고 싶은 내용들을 한 번에 결정해야 하고, commit하고 싶은 일부분만 계속 disk의 어떤 공간에 올려두는 행위를 할 수 없게 됨!

실습 추가

지난 수업시간에 진행했던 branch 생성 후 main branch에 합치는 과정입니다.

main branch의 main.py

```
print('hello world')
```

junghye branch의 main.py

```
print('hello world')
name='junghye'
print(f'I\'m junghye')
programming_language='Python'
print('Most used language is {0}'.format(programming_language))
```

junghye 브랜치에는 add commit push까지 마친 상황임!

이 때 junghye 브랜치에서 수정된 내용을 main branch에 통합시키려 함

통합시킨다 = 따로 분기한 브랜치에서 작업한 내역들(커밋 기록, 변경한 파일들)을 main branch에 합치려 한다는 뜻

sol1) git merge {branch 이름} → add, commit, push 방식

1. main branch로 이동

```
git checkout main
```

2. `git merge {분기한 브랜치명}`

저의 경우에는 git merge junghye

3. `add, commit, push` 진행

sol2) Compare & Pull request

변경사항을 웹 인터페이스를 통해 확인하고, pull request를 진행하는 방식

1. 상단에 'Compare & pull request' 클릭

The screenshot shows the GitHub interface for the repository 'junghye01 / ossprac'. At the top, there's a yellow banner stating 'junghye had recent pushes less than a minute ago' with a green 'Compare & pull request' button. Below this, the 'main' branch is highlighted, with a note that it's not protected. A table lists the repository's files: .gitignore, LICENSE, README.md, and main.py, each with its commit history. On the right, there are sections for 'About', 'Releases', 'Packages', and 'Contributors'.

2. Create pull request 클릭

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

base: main ← compare: junghye ✓ Able to merge. These branches can be automatically merged.

Write Preview H B I T E < > P L I S E @ ↩ ↲

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

Create pull request

Remember, contributions to this repository should follow our [GitHub Community Guidelines](#).

Reviewers: No reviews

Assignees: No one—assign yourself

Labels: None yet

Projects: None yet

Milestone: No milestone

Development: Use [Closing keywords](#) in the description to automatically close issues

able to merge라고 뜨면 merge가능, 아닐경우 충돌 해결해줘야함(히스토리 보고 각 브랜치 내용 비교해서 개발자가 직접 수정)

3. Merge pull request 클릭

Open junghye01 wants to merge 2 commits into main from junghye

Conversation 0 Commits 2 Checks 0 Files changed 0

junghye01 commented now

No description provided.

Owner

junghye01 added 2 commits 4 minutes ago

init 534232f

ADD personal info 4842624

Add more commits by pushing to the junghye branch on junghye01/ossprac.

Require approval from specific reviewers before merging

Branch protection rules ensure specific people approve pull requests before they're merged.

Add rule

Continuous integration has not been set up

GitHub Actions and several other apps can be used to automatically catch bugs and enforce style.

This branch has no conflicts with the base branch

Merging can be performed automatically.

Merge pull request You can also open this in [GitHub Desktop](#) or view [command line instructions](#).

Reviewers: No reviews

Assignees: No one—assign yourself

Labels: None yet

Projects: None yet

Milestone: No milestone

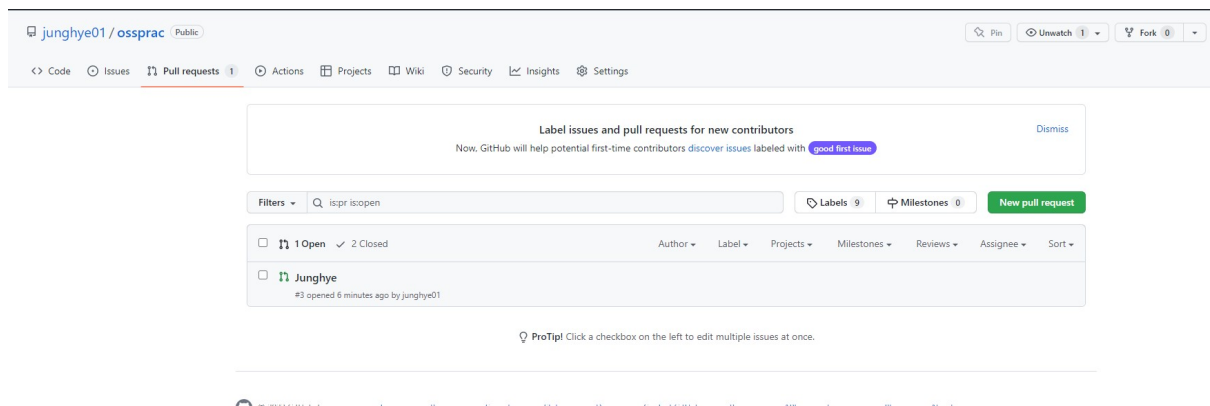
Development: Successfully merging this issues.

Notifications: You're receiving notificat this repository.

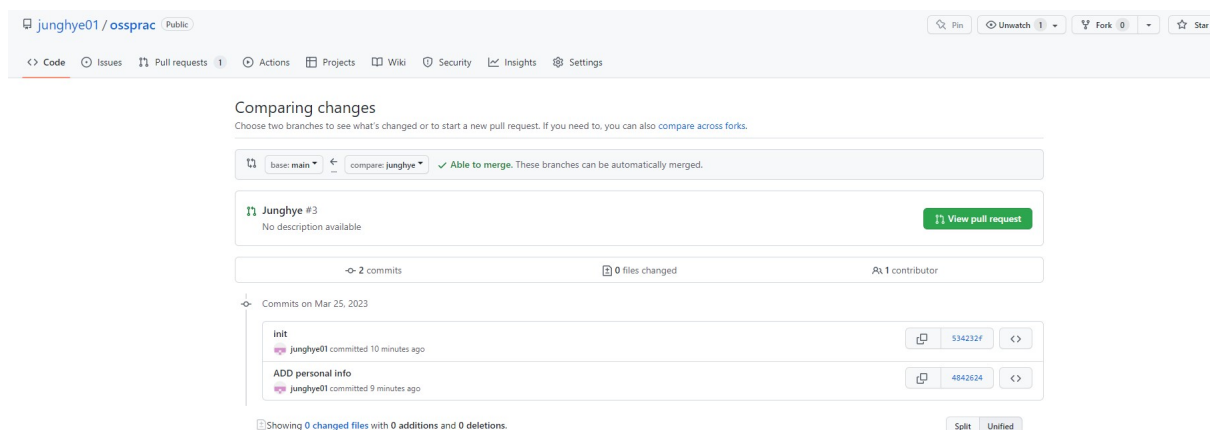
main branch와 비교했을 때 분기한 브랜치에서 작업한 내역들 확인가능

번외) Compare & pull request가 안 뜨시는 분들

1. 상단 바에 'Pull requests' 클릭 → 'New pull request' 클릭



2. 'View pull request' 클릭하고 다음 화면에서 'Merge pull request' 동일하게 진행



이렇게 해도 안 뜬다면 main branch와 분기한 브랜치의 커밋 내역들을 비교해보세요.
커밋 내역들이 동일하면 안 뜰 수도 있습니다.

그 외 질문

Q. `git push origin {branch이름}` 할 때 origin 이 뭔가요?

A. origin은 저희가 클론한 원격저장소를 가리킵니다. 즉, 로컬저장소와 연결된 원격저장소라고 생각하시면 됩니다.

git remote -v를 통해서 현재 로컬 저장소와 연결된 원격저장소를 확인하실 수 있습니다.

```
SAMSUNG@DESKTOP-NUC55SE MINGW64 ~/Downloads/ossprac (main)
$ git remote -v
origin https://github.com/junghye01/ossprac.git (fetch)
origin https://github.com/junghye01/ossprac.git (push)
(base)
```

Q. git workflow 추가 설명

working directory : 작업 폴더

staging area : 커밋하기 전에 1차적으로 수정된 내역이 저장되는 공간

local repository : 로컬의 Git 저장소. commit을 하면 .git 폴더에 staging area에 있는 파일들이 하나의 버전으로 저장된다.

참고로 .git 폴더는 `git clone` 하거나 `git init` (이미 존재하는 폴더에 초기화를 하여 VCS(Version Control System) 관리를 위한 숨겨진 영역을 생성하는 명령어) 를 하면 생성되는 폴더이다. .git폴더는 git에 관련된 모든 사항을 기록함. 숨김 폴더라 `ls -a` 를 통해 확인 가능

```
(base)
SAMSUNG@DESKTOP-NUC55SE MINGW64 ~/Downloads/ossprac (main)
$ ls -a
./ ../ .git/ .gitignore LICENSE main.py README.md
(base)
```

remote repository : push 명령어를 실행하면 로컬저장소의 내용들이 원격저장소로 이동