How to use Git

 Git 기본 개념

- **저장소(repository)**repository(저장소)는 파일이나 디렉토리를 저장하는 장소입니다. Git repository의 장점은 변경 이력이 존재하는 파일별로 구분되어 저장된다는 점입니다. 저장소는 내 컴퓨터에 있는 '로컬 저장소(local repository)'와 전용 서버에서 관리되는 '원격 저장소(remote repository)'의 2개가 존재합니다. 기본적으로 로컬 저장소에서 작업을 진행하고 그 결과를 원격 저장소에 저장할 수 있습니다.
- **브랜치(branch)**소프트웨어가 출시된 이후 버전의 지속적인 유지 보수를 통해 기능을 추가하거나 버그를 수정하게 됩니다. 이처럼 여러 버전을 관리하기 위해 브랜치(branch)라는 기능을 사용합니다.
- **커밋(commit)**파일을 추가하거나 변경사항을 저장소에 기록하기 위해 커밋 메시지를 남기는 작업을 말합니다.변경한 사람, 시간, 변경 내용 등을 기록해 과거 변경 이력과 내용을 손쉽게 파악할 수 있습니다.
- 리눅스 기반 → 리눅스 명령어에 익숙할 시 깃에도 익숙할 수 있습니다.

 필요한 프로그램

- Git Bash(https://git-scm.com/)

 Git 폴더 생성 방법

```
# 1. 원하는 디렉토리로 이동한 후 깃에 업로드할 로컬 폴더 생성

cd Desktop/sunjunglee/directory
mkdir git_hub/

# 2. 생성한 폴더로 이동

cd git_hub/

# 3. 깃 저장소 지정(깃 초기화)

git init

### 다른 방법

# 1. 원하는 디렉토리로 이동

cd Desktop/
```

```
# 2. 깃 저장소 생성

git init git_folder/
```

💡 Git Clone 하기

```
#1. 원하는 디렉토리로 이동

cd Desktop/sunjunglee/directory

#2. clone 명령어 입력

git clone [https://클론할원격저장소주소]
```

💡 Git 업로드 방법

```
# 1. git 계정 로그인

git config --global user.name "username"
git config --global user.email "user@email.com"

# 2. git 원하는 폴더 이동

cd /디렉토리/디렉토리/디렉토리/업로드할폴더

# 3. 폴더 깃에 추가하기 위한 명령어 입력

git add .
git add 디렉토리/ #디렉토리 추가
git add ~~.ipynb #파일 추가

# 4. 커밋 메세지 입력

git commit -m "Upload a Folder" #메세지는 예시문

### 수정한 작업물 커밋

git commit -am "message"

## 커밋한 메세지 수정

git commit --amend

# 5. git 로컬 저장소와 원격 저장소 연결

git remote add origin 복사한주소붙여넣기

# 5-2. 만약 error: remote origin already exists.에러 발생하면

git remote remove origin

# 5-4. 원격 저장소 연결되었는지 확인하기
```

```
git remote -v

# 6. 원격 저장소에 파일 올리기 - git push

git push -u origin master # 만약 error: failed to push some refs to ~ 오류 발생시 master가 아닌 main으로 설정되어 있는지 확인해보기

# 7. 원격 저장소의 내용 master 브랜치로 가져오기

git pull origin master # 오류 발생시 master가 아닌 main 등 확인해 입력
```

💡 깃 로그인 정보 확인하기

```
git config --list
```

💡 진행상황 확인하기

```
# 깃 상태확인

git status

# 변경사항 확인

git diff

# 깃 커밋 확인

git log # (HEAD -> master)라고 적힌 부분이 가장 최신 부분임

# 한 줄로 커밋내역 확인하기

git log --oneline

# 각 브랜치 커밋 확인

git log --oneline --branches

# 그래프로 각 브랜치 커밋 확인하기

git log --oneline --branches --graph

# 변경사항 확인

git diff
```

💡 Git에 있는 폴더를 로컬로 덮어쓰기

```
git fetch --all
git reset --hard origin/master
```

💡 커밋 취소(되돌리기)

```
# 최신 커밋 취소

git reset HEAD^

# 최근 5개 커밋 취소

git rest HEAD~5
```

💡 vscode 파일 열기

```
code . [파일명].[파일확장자명]
```

브랜치 명령어 정리

💡 새로운 브랜치에서 파일 생성 후 병합

```
# 기존 마스터 브랜치에서 작업한 작업물이 있어야 브랜치 생성 가능

# 새 브랜치 생성 및 브랜치 확인 명령어

git branch # master -> 처음 실행시 master에서 작업하고 있는 것 확인 가능

# 새로운 브랜치 생성

git branch [브랜치명]

# 브랜치 사이 이동하기

git checkout [브랜치명]

git log --oneline # master에서 apple로 이동한 것 확인 가능

# 생성한 브랜치에서 새로운 작업물 생성후 커밋
```

```
git add [새로운 작업물]
git commit -m "message"

# 생성한 브랜치에서 작업 후 master(main)으로 병합

git checkout master # master 브랜치로 체크아웃해야함

git merge [생성한 브랜치명] # 작업물 병합

ls -al # master 브랜치 안에 있는 변경된 작업물 확인하기
```

💡 브랜치에서 수정 후 마스터 병합

```
# 생성한 브랜치로 이동

git checkout [브랜치명]

# 생성한 브랜치 내에서 마스터 브랜치의 작업물 수정 후 커밋

git commit -am "message"

# master 브랜치로 이동

git checkout master

# 브랜치에서 수정한 내용 병합

git merge [브랜치명]
```

💡 master branch와 같은 부분 수정 후 병합

```
# 생성한 브랜치로 이동

git checkout [브랜치명]

# 생성한 브랜치 내에서 마스터 브랜치 작업물과 같은 부분 수정 후 커밋

git commit -am "message"

# master 브랜치로 이동

git checkout master

# 브랜치내에서 수정한 내용 병합 -> 오류 발생

git merge [브랜치명] # 오류발생

# 충돌 부분 확인 후 직접 수정한 후 커밋

git commit -am "message"
```

💡 브랜치 삭제

```
git branch -d [브랜치명]
```

💡 수정한 파일 숨기는 명령어

```
# 숨기기

git stash

# 숨긴 항목 되돌리기

git stash pop

# 숨긴 항목 삭제하기

git stash drop
```

타 계정 충돌 해결하기

- 제어판 → 사용자계정 → Windows 자격증명 → github 관련한 계정 삭제