

# Git / GitHub Tutorial





# INDEX

1 Git, GitHub?

2 Git의 사용법

3 GitHub의 사용법

4 GitHub Desktop 사용법

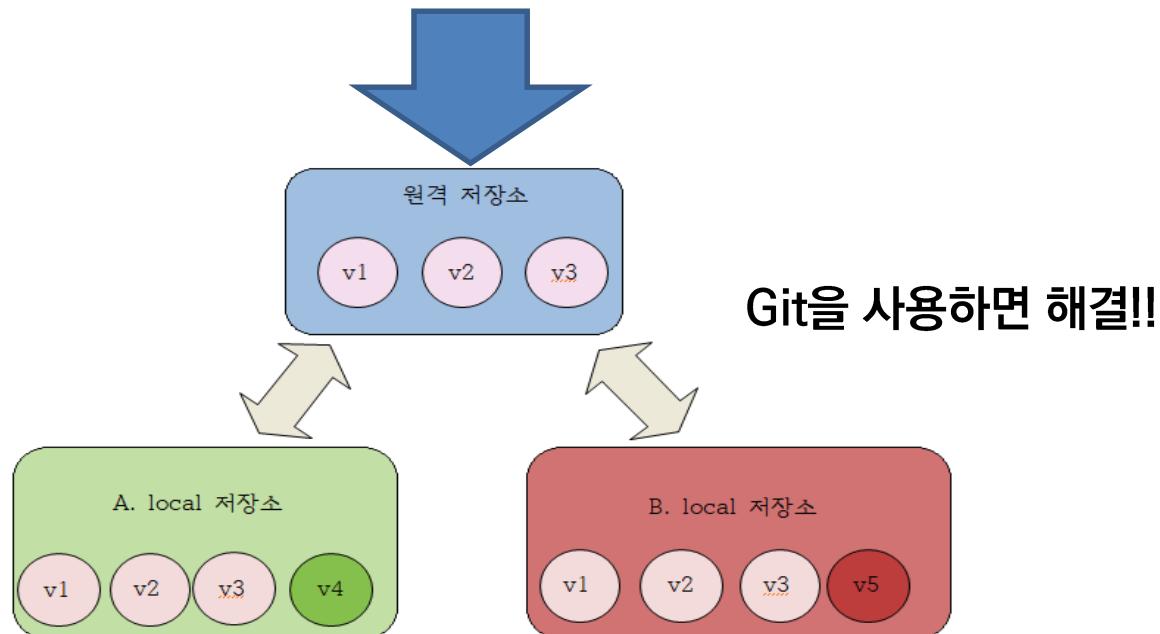
# 1.1 Git은 무엇인가?

▶ 버전관리를 위한 프로그램이자 형상관리 도구

- 컴퓨터 파일의 변경사항을 추적하고 여러 명의 사용자들 간에 해당 파일들의 작업을 조율하기 위한 분산 버전 관리 시스템
- 소스를 버전 별로 관리할 수 있어서 개발 과정에서 실수로 소스를 삭제하거나 수정하기 이전으로 돌아가는 경우 유용하게 사용되는 Tool
- Git은 Git Bash라는 커맨드에서 동작하며 윈도우는 명령 프롬프트(cmd)에서 맥os는 터미널에서도 사용할 수 있다.
- 설치주소는 <https://git-scm.com/downloads>로 자신에 맞는 운영체제로 설치할 수 있음

# 1.1 Git은 무엇인가?

- ① 한글 문서를 빠르게 작업하고자 팀원들이 동시 작업을한다면 이 문서는 어떻게 관리가 될까??
- ② 팀원이 동시에 작업하므로 이 문서는 하루가 다르게 업데이트가 될 것이다. 매일 또는 시간 단위로 업데이트되는 파일을 어떻게 팀원과 공유할까??
- ③ 이전 버전의 문서로 작업을 했을 경우를 대비하여 어느 시기에 백업을 해둬야 할까??
- ④ 팀원 중 누가 어느 부분을 작성했는지 어떻게 알까??

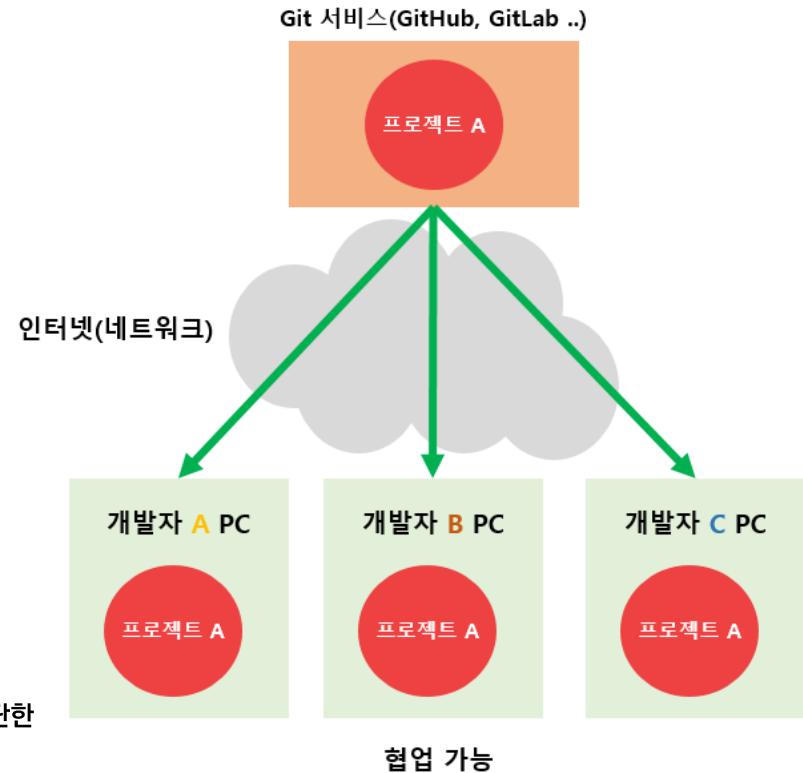


# 1.2 GitHub은 무엇인가?

▶ Git의 원격 저장소 서비스 → Git을 호스팅해주는 웹 서비스

- Git 저장소 서버를 대신 유지 및 관리해줌
- 백업이 기본 기능이지만 단순히 저장 뿐 아니라 다른 유저들과 함께 코드를 공유하고 온라인으로 하나의 프로그램을 같이 제작 가능
- Git을 통해 협업하는 프로젝트를 공유 할 수 있는 네트워크상의 저장공간이 필요  
→ 네트워크상 저장공간을 제공해주는 서비스 중 하나가 GitHub!

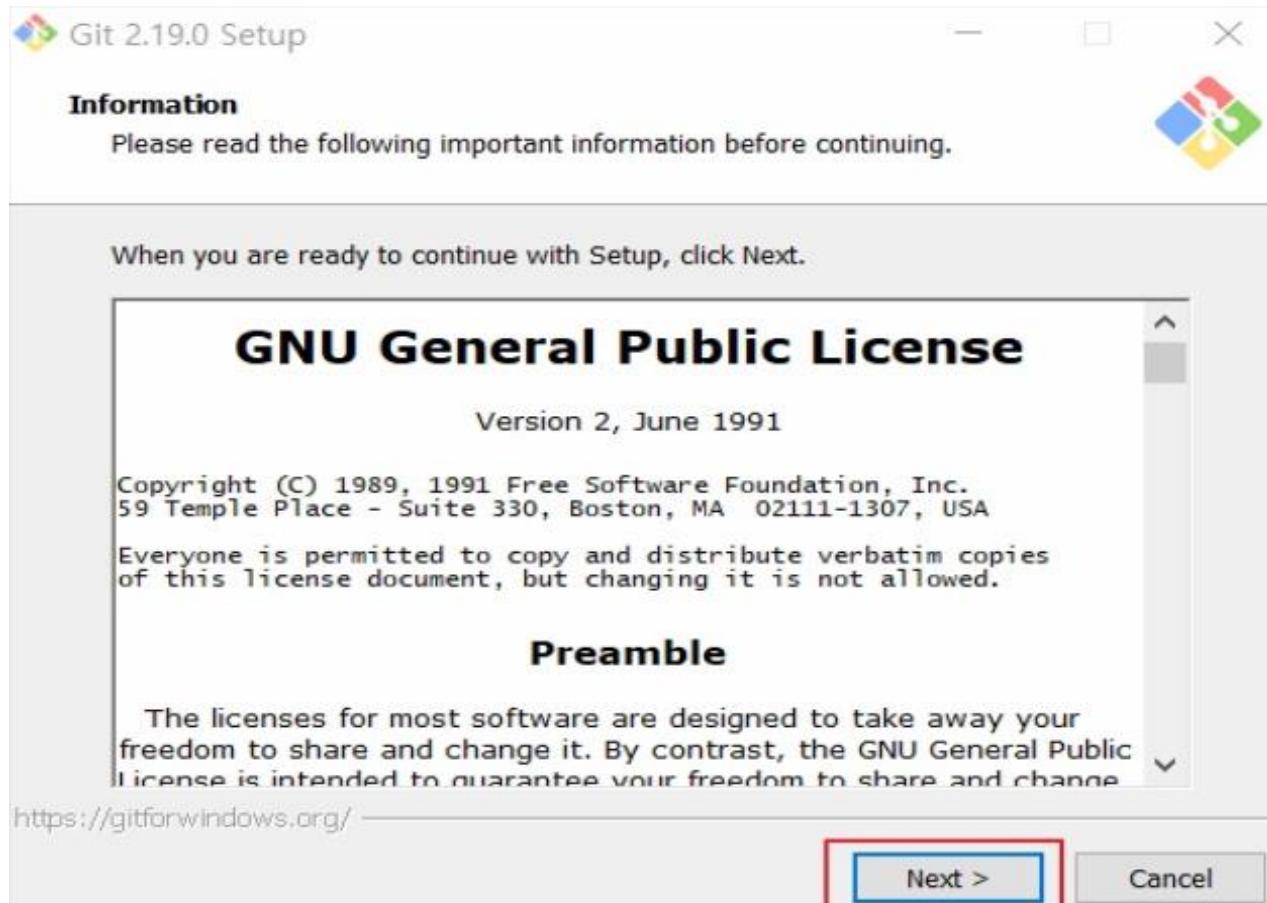
※ GitHub는 Internet Explorer 지원을 2018년 7월에 종단한다고 합니다. 지원을 종단한다고 해서 접속이 안되는 않겠으나, 버그를 마주할 수도 있으니 Microsoft Edge, Google Chrome 또는 Firefox를 이용할 것을 추천함



# 2.1 Git 설치

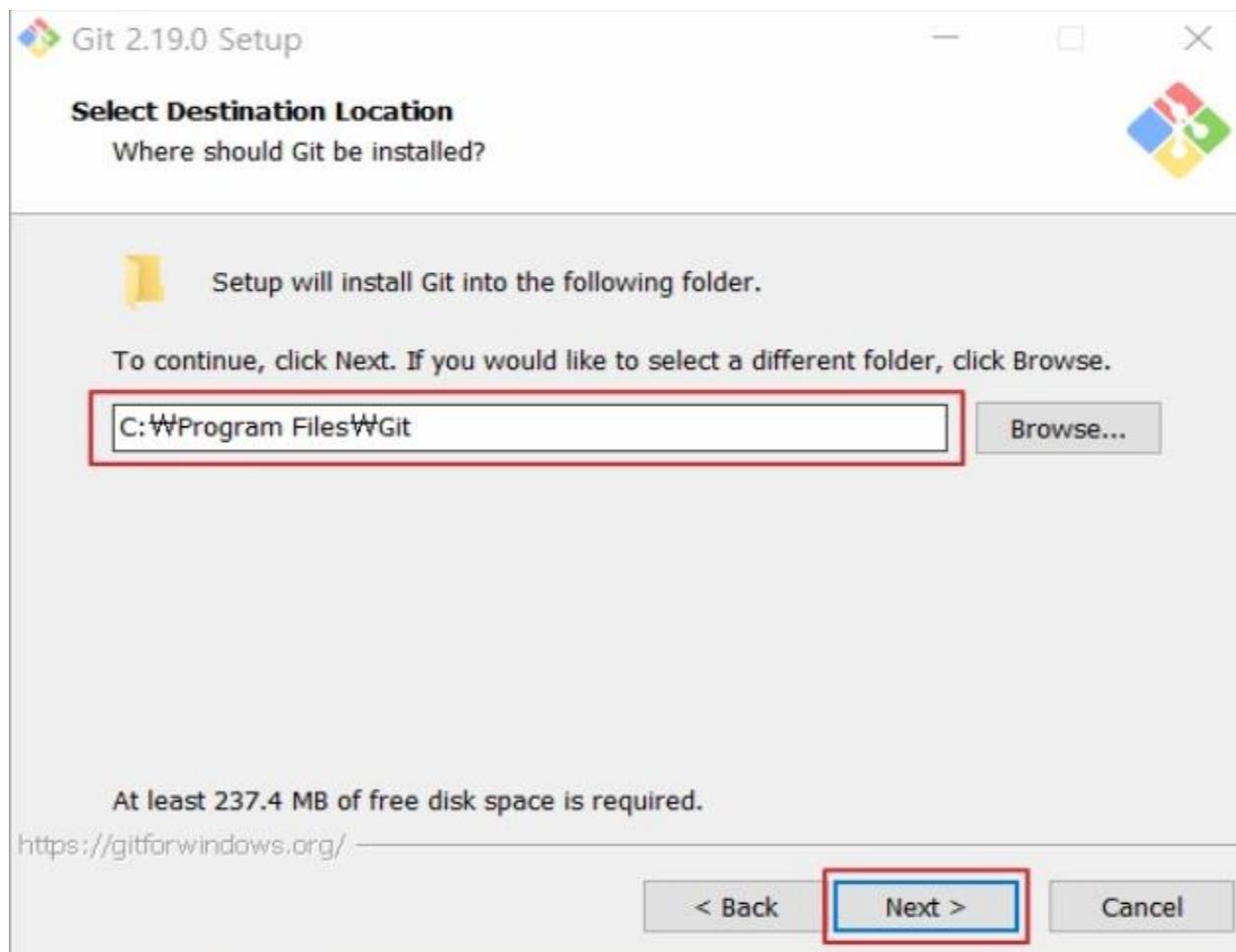
## 1. 약관을 읽어주고 Next를 눌러줍니다

※ <https://git-scm.com/downloads>로 자신에 맞는 운영체제로 설치할 수 있음



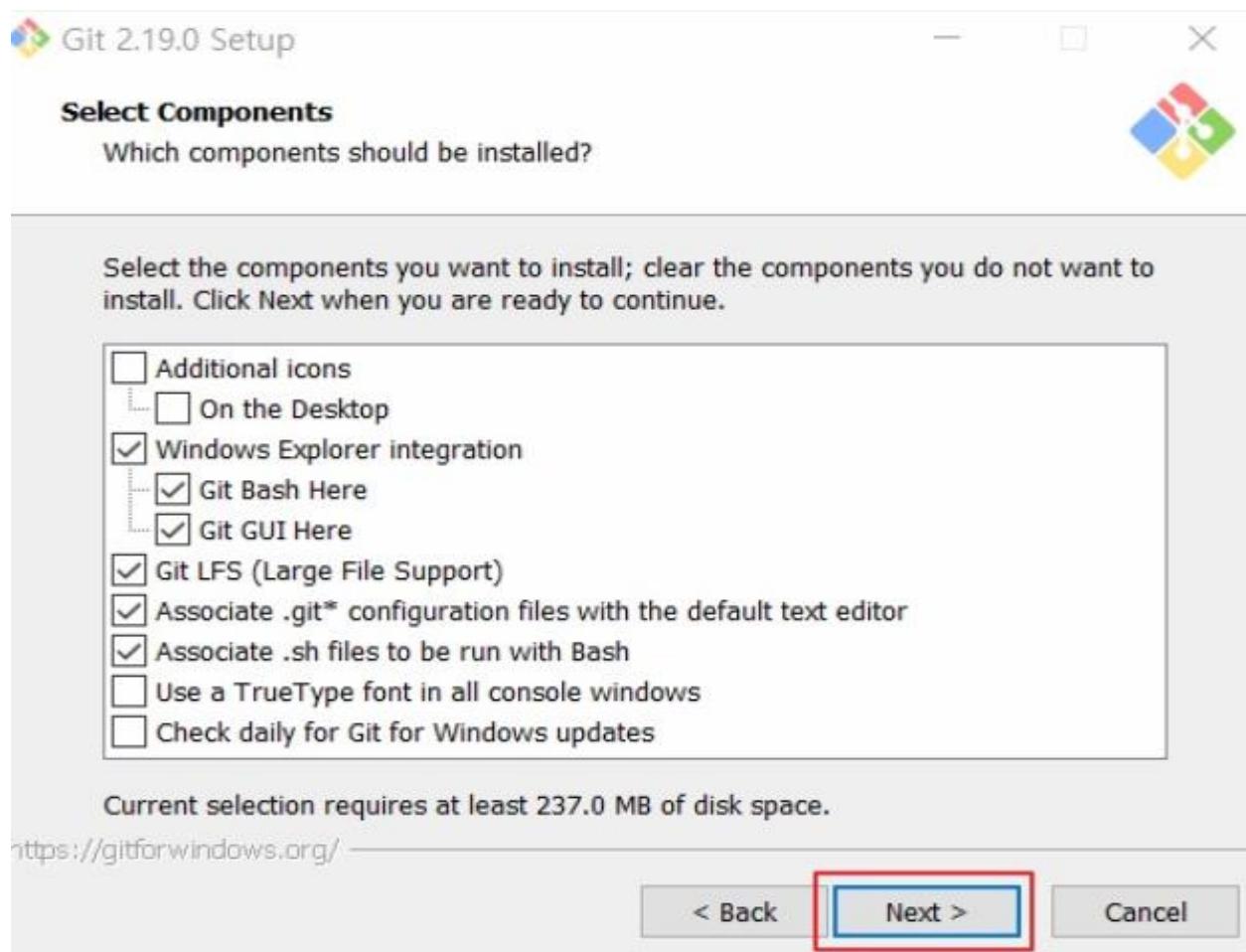
## 2.1 Git 설치

2. 설치경로를 선택하고 Next를 눌러줍니다



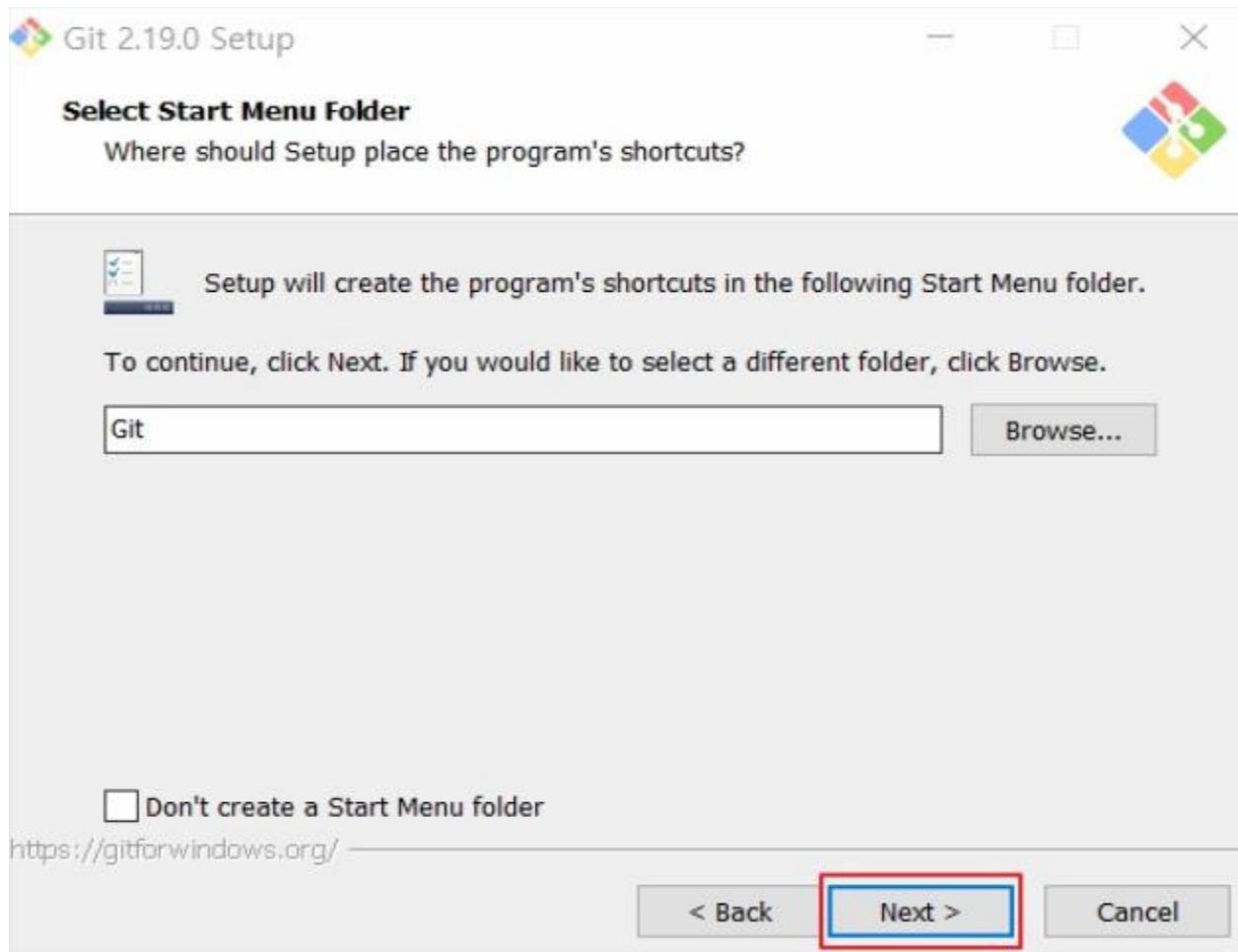
## 2.1 Git 설치

### 3. 설치한 Components를 선택합니다



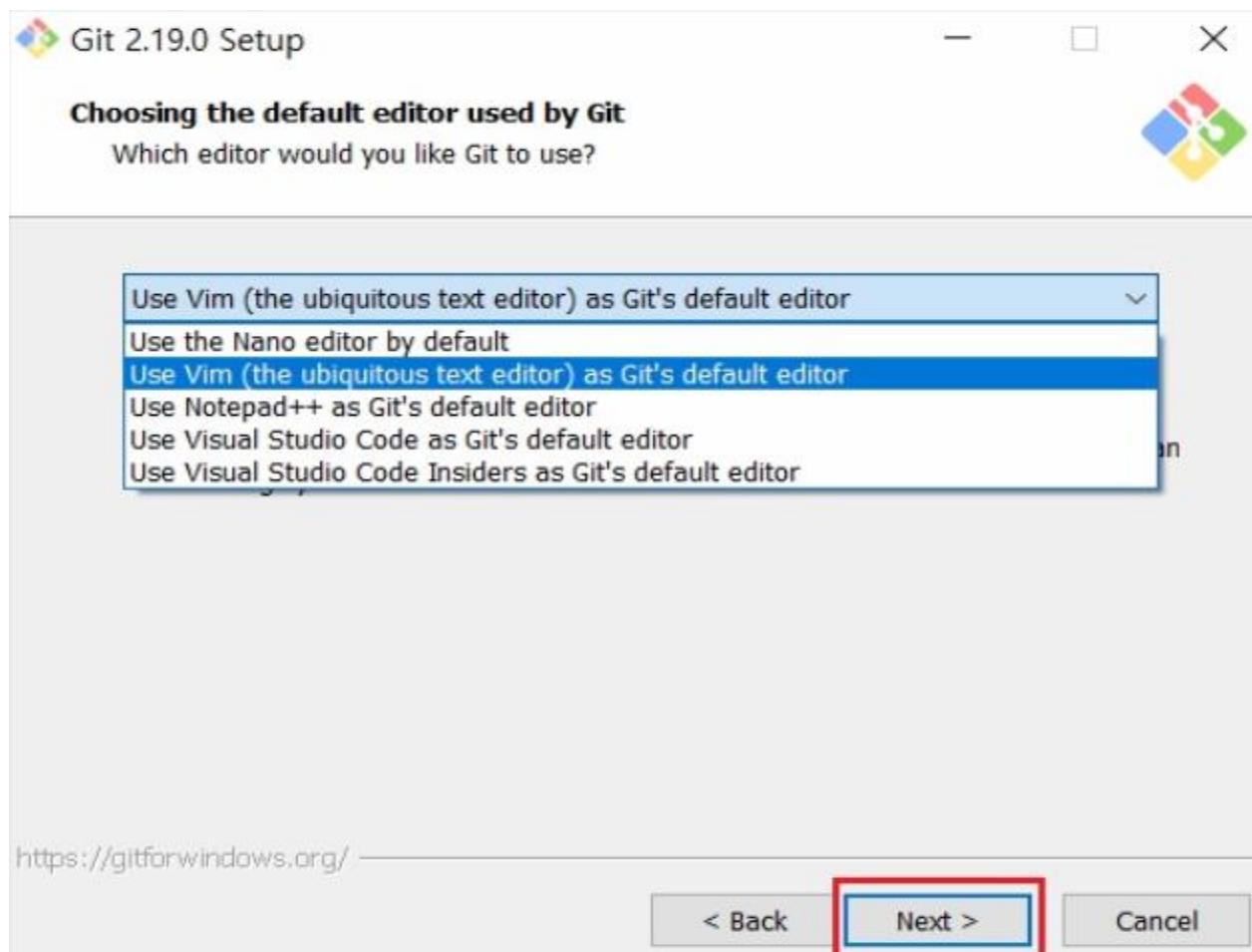
## 2.1 Git의 설치

### 4. 시작 메뉴에 폴더를 만듭니다 (추가를 원하지 않을 경우 Don't create a Start Menu folder 체크)



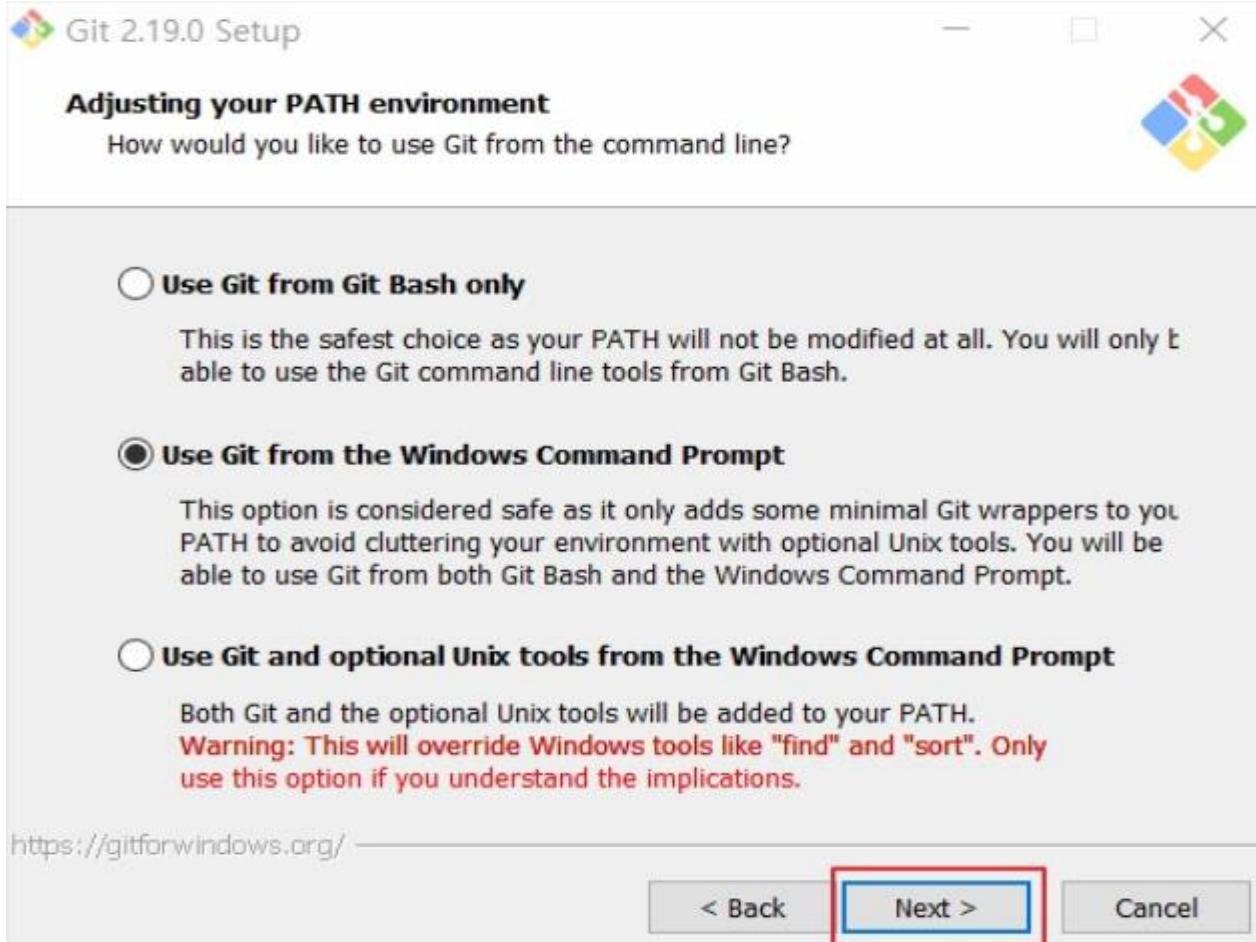
## 2.1 Git의 설치

### 5. Git의 기본 editor를 설정



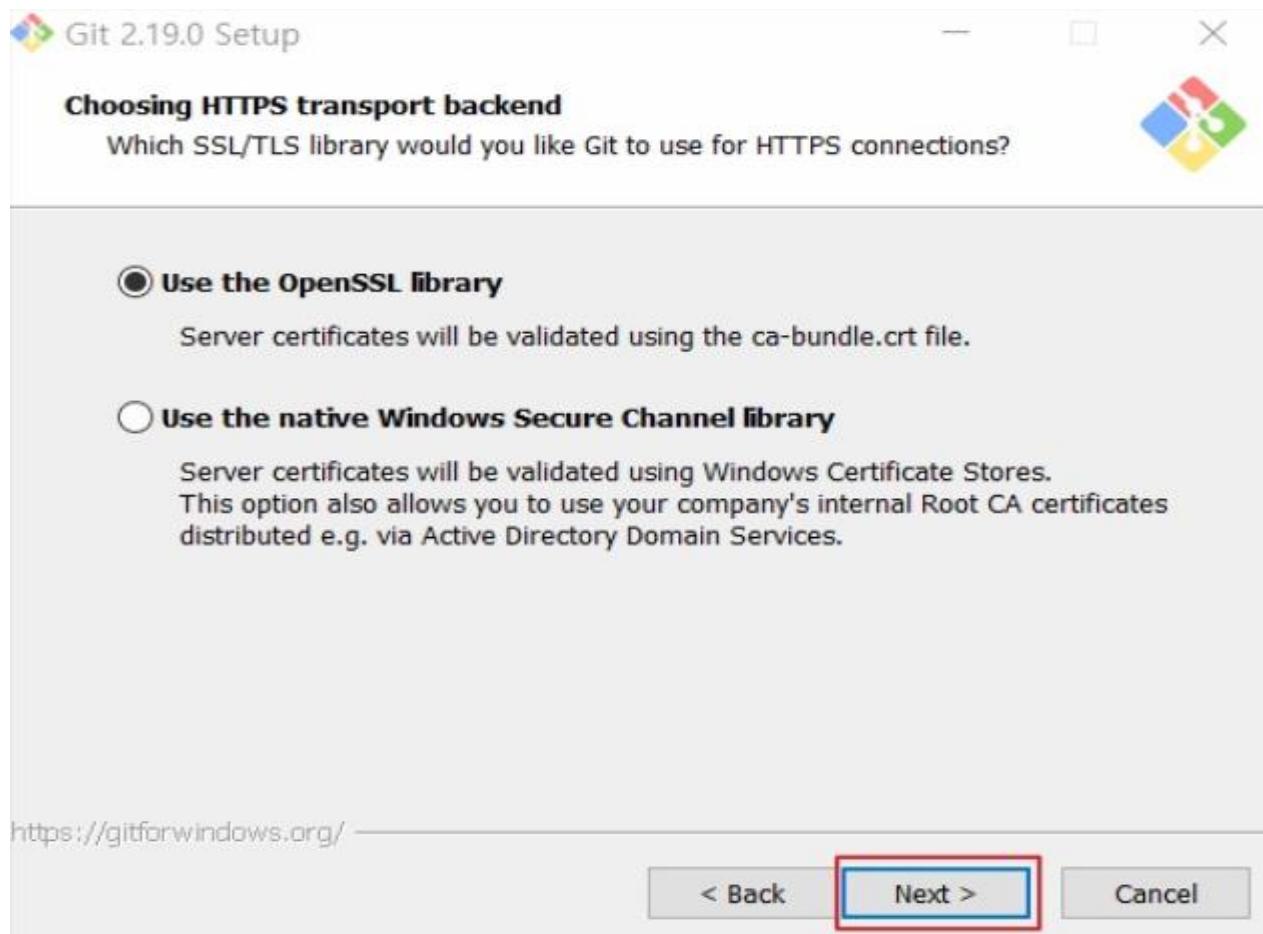
## 2.1 Git의 설치

### 6. Git커맨드의 설정 부분(일반적으로 cmd를 사용)



# 2.1 Git의 설치

## 7. 디폴트로 되어 있는 Use the OpenSSL library를 선택

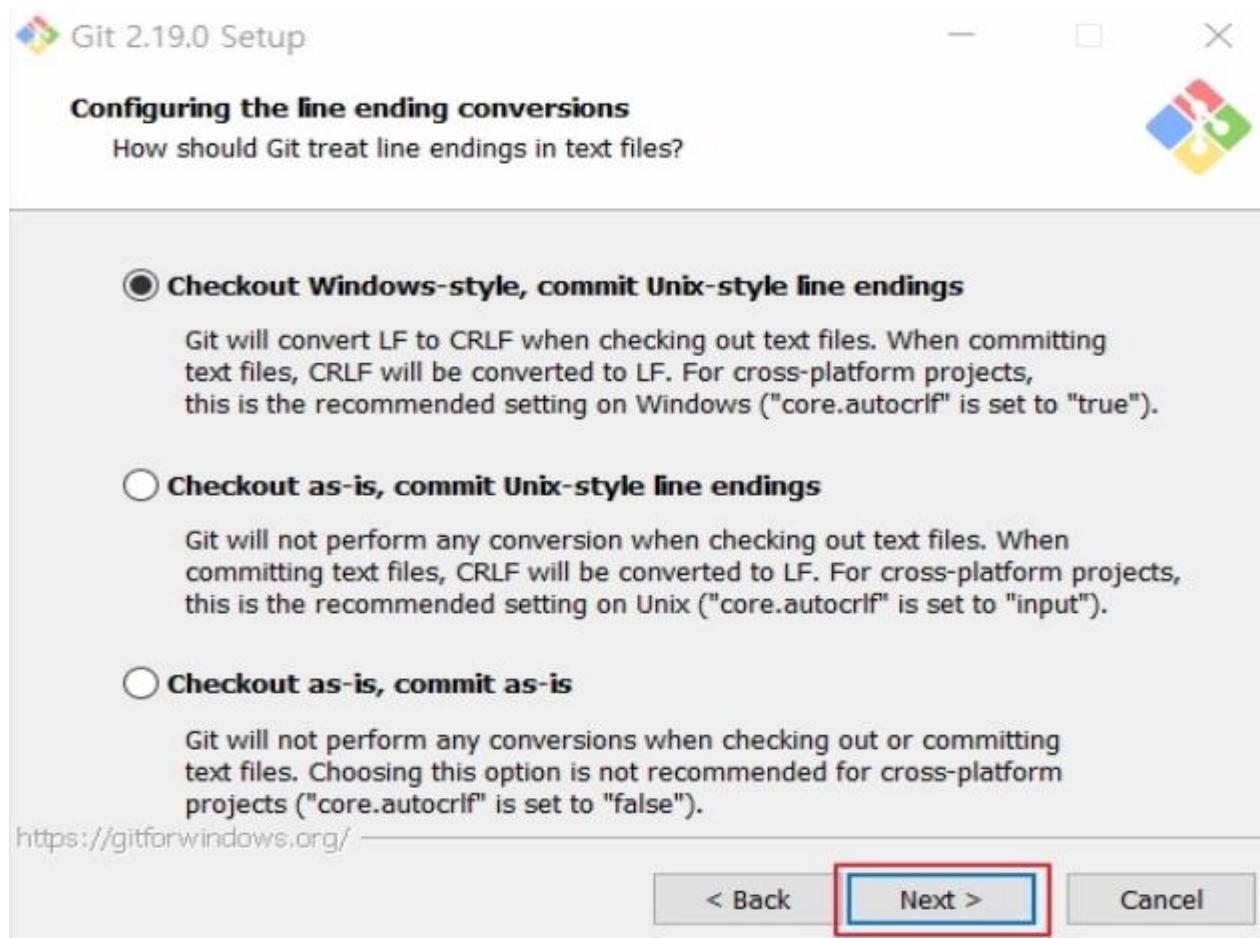


● **Use the OpenSSL library :**  
OpenSSL 라이브러리 사용하며 서버인증서는 ca-bundle.crt 파일을 사용하여 유효성 검사

● **Use the native Windows Secure Channel library :**  
Windows 인증서 저장소를 사용하여 서버 인증서의 유효성 검사

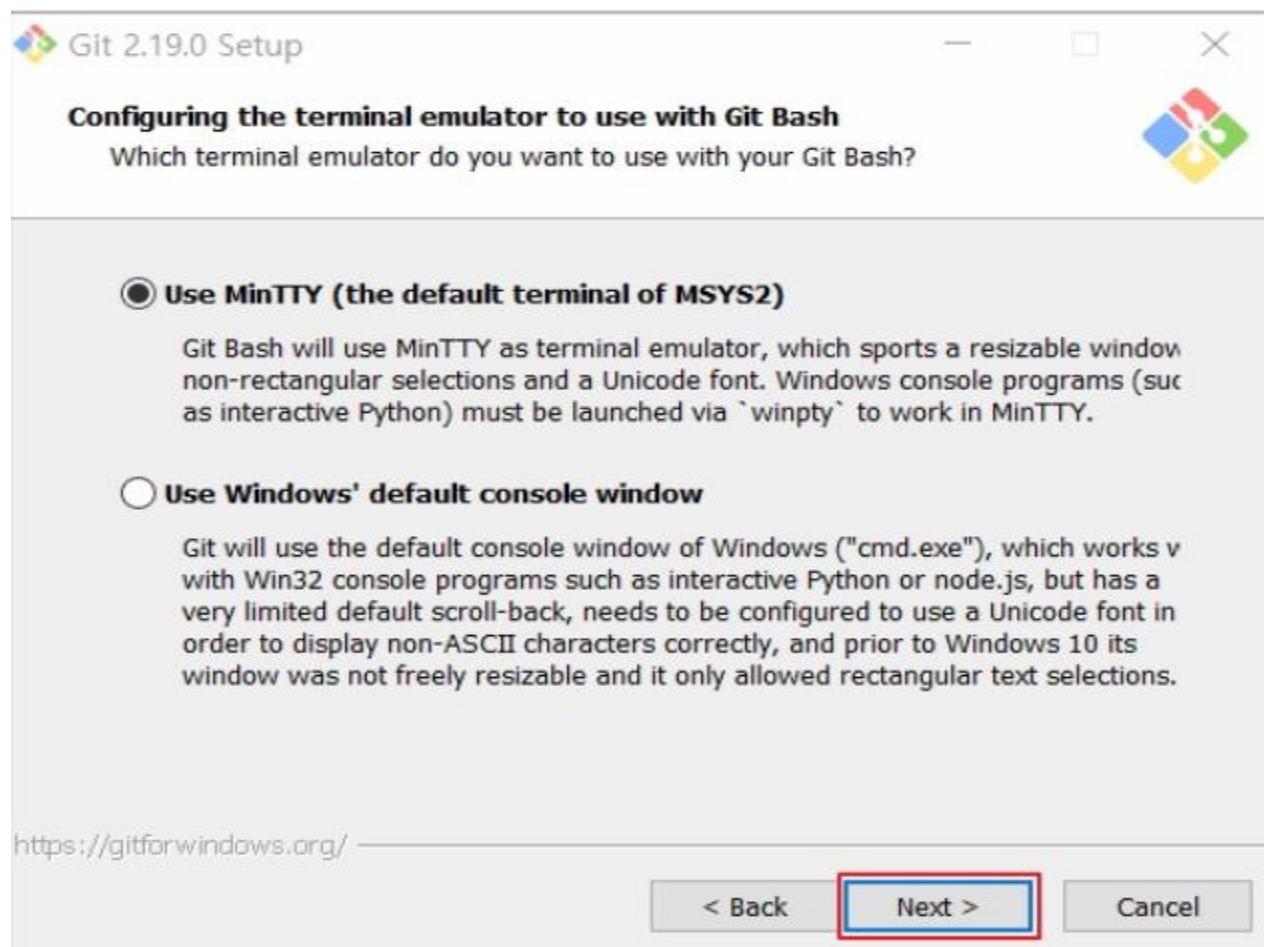
## 2.1 Git의 설치

### 8. Checkout, Commit할 때의 텍스트라인 엔딩을 선택



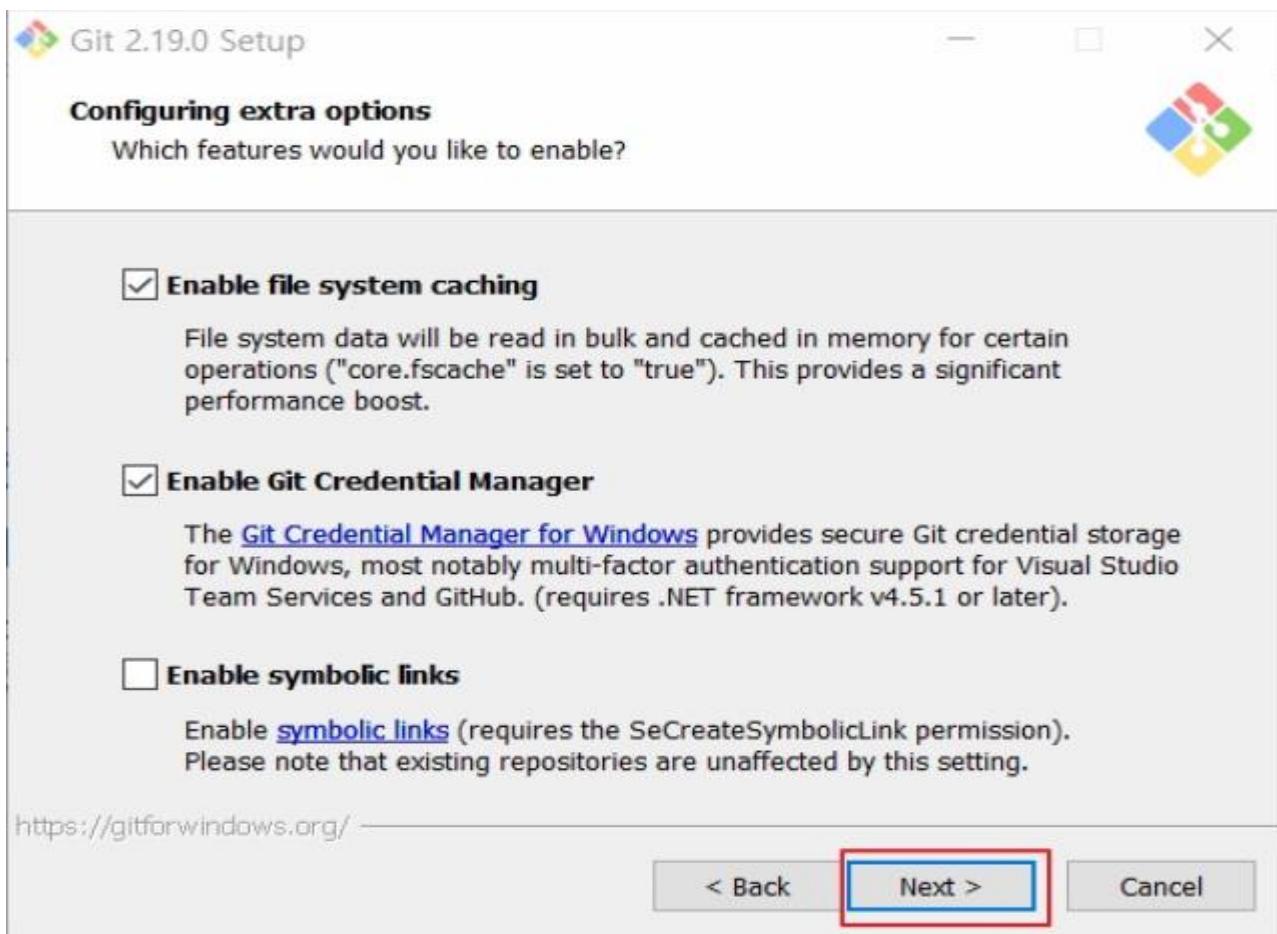
# 2.1 Git의 설치

## 9. Git Bash 터미널 형식을 선택



# 2.1 Git의 설치

## 10. 옵션 선택



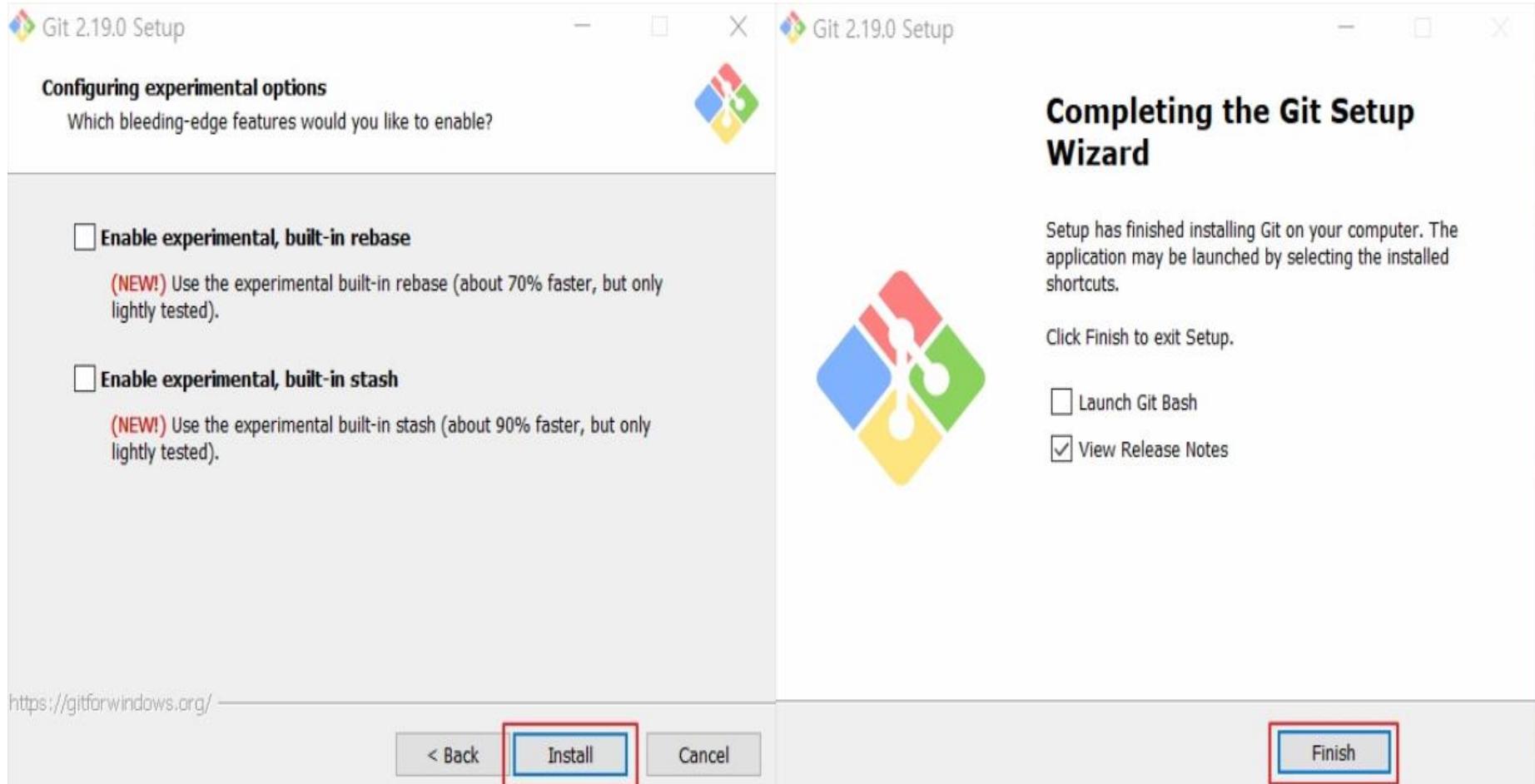
● **Enable file system caching :**  
성능향상을 위해 파일 시스템 데이터를 메모리에 캐시함

● **Enable Git Credential Manager :** Windows 용 보안 Git 자격증명 저장소를 사용하기 위해 Git Credential Manager 활성화함

● **Enable symbolic links :**  
symbolic links 활성화함(기존 저장소는 영향을 받지 않음)

# 2.1 Git의 설치

## 11. Install



## 2.2 Git초기 세팅

### ▶ 사용자 이름과 이메일을 등록

The screenshot shows a terminal window titled 'MINGW64:/c/Users/MASTER'. It displays the following command-line session:

```
MASTER@DESKTOP-4GKBQ15 MINGW64 ~
$ git config --global user.name Ham
git: 'comfig' is not a git command. See 'git --help'.

The most similar command is
  config

MASTER@DESKTOP-4GKBQ15 MINGW64 ~
$ git config --global user.name Ham

MASTER@DESKTOP-4GKBQ15 MINGW64 ~
$ git config --global user.email ii8858@naver.com

MASTER@DESKTOP-4GKBQ15 MINGW64 ~
$ git config --global --list
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
user.name=Ham
user.email=ii8858@naver.com

MASTER@DESKTOP-4GKBQ15 MINGW64 ~
$ |
```

1. git config --global user.name 사용자 이름  
→ 작업자 이름 설정

2. git config --global user.email 사용자 이메일  
→ 작업자 이메일 설정  
3. git config --global --list  
--> 설정값 확인

## 2.3 Git저장소 만들기 1

### ▶ 버전관리를 하지 않는 로컬 디렉토리(폴더) 선택하여 적용

```
이제운@sec MINGW64 ~
$ cd c:/Gitpratice2

이제운@sec MINGW64 /c/Gitpratice2
$ git init
Initialized empty Git repository in C:/Gitpratice2/.git/

이제운@sec MINGW64 /c/Gitpratice2 (master)
$ git add hello.txt

이제운@sec MINGW64 /c/Gitpratice2 (master)
$ |
```

※ git init --> 여러 파일을 추적하는 .git 폴더가 생성. 이것을 입력한 후에야 추가적인 git 명령어들을 줄 수 있음  
(init 명령어를 실행한 직후에는 프로젝트 디렉터리 내부에 있는 어떤 파일도 Git에게 관리되지는 않음)  
(git init 이후에 해당 프로젝트 디렉터리 내부의 파일들을 Git의 관리 대상으로 등록하고 싶은 경우에는 git add 명령어를 통해 목록에 추가하고 git commit 명령어를 통해 커밋해야 함)

※ git add --> working directory의 변경된 작업 파일을 staging area로 추가  
(ex 작업 트리에 a.txt파일이 추가되면 git add hello.txt 명령을 통해 Index에 hello.txt 파일의 변경 사항을 추적할 수 있게 함)

1. cd [등록할 프로젝트 디렉터리 경로] 를 통해 기존에 존재하는 디렉터리로 이동
2. 경로에 이동했다면 git init 명령어를 통해 Git 저장소로 등록
3. git add 명령어로 파일을 추가 (git 관리대상으로 추적을 시작 )
4. add 후에 아래와 같이 git status명령어로 현재 상태 확인

(어떤 파일이 저장소 안에 있는지, commit이 필요한 변경사항이 있는지, 현재 저장소의 어떤 Branch에서 작업하고 있는지 등을 볼 수 있음)

```
이제운@sec MINGW64 /c/Gitpratice2 (master)
$ git status
On branch master

No commits yet

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

## 2.3 Git저장소 만들기 1

### ▶ 버전관리를 하지 않는 로컬 디렉토리(폴더) 선택하여 적용

```
이제 윈 @sec MINGW64 /c/Gitpratice2 (master)
$ git commit -m "new file!!"
git commit -m "new file"
git status
[master (root-commit) 43f5db5] new file
  1 file changed, 0 insertions(+), 0 deletions(-)
   create mode 100644 hello.txt
```

```
이제 윈 @sec MINGW64 /c/Gitpratice2 (master)
$ git status
On branch master
nothing to commit, working tree clean
```

```
이제 윈 @sec MINGW64 /c/Gitpratice2 (master)
$ |
```

hello.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움(I)

version2

git gitgit gitgit git git git

7. 이후 해당 디렉터리의 파일 내용을 변경 후 다시 git status를 사용해 현재 상태를 확인하면 변경된 사실을 알 수 있음

6. 추가 되었지만 저장소에 확정된 것이 아님으로  
git commit -m “내용 설명 등” 명령어로 저장소에  
확정을 하고 status로 상태를 확인함

※ git commit --> 어떤 변경사항이라도 만든 후, 저장소의 “스냅샷”을  
찍기 위해 이것을 입력한다.  
--> staging area의 내용을 local repository에 확정 짓는다.

```
이제 윈 @sec MINGW64 /c/Gitpratice2 (master)
$ git status
On branch master
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:   hello.txt

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

## 2.3 Git저장소 만들기 1

### ▶ 버전관리를 하지 않는 로컬 디렉토리(폴더) 선택하여 적용

```
이제 윈 @sec MINGW64 /c/Gitpratice2 (master)
$ git add hello.txt

이제 윈 @sec MINGW64 /c/Gitpratice2 (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   hello.txt
```

```
이제 윈 @sec MINGW64 /c/Gitpratice2 (master)
$ git commit -m "version2 file"
[master ae9f628] version2 file
 1 file changed, 2 insertions(+)
```

```
이제 윈 @sec MINGW64 /c/Gitpratice2 (master)
$ git log
commit ae9f6288f54a5bc4437005b046afc40448e48c7f (HEAD -> master)
Author: yoon <happysunlee7@naver.com>
Date:   Mon Oct 14 15:37:49 2019 +0900

  version2 file

commit 43f5db5342ff65e14ba1c60c709403f31cbce60d
Author: yoon <happysunlee7@naver.com>
Date:   Mon Oct 14 15:23:26 2019 +0900

  new filegit status

이제 윈 @sec MINGW64 /c/Gitpratice2 (master)
$ |
```

8. 내용이 변경되었으니 다시 git add 명령어로 추가하는 작업과 commit하도록 함

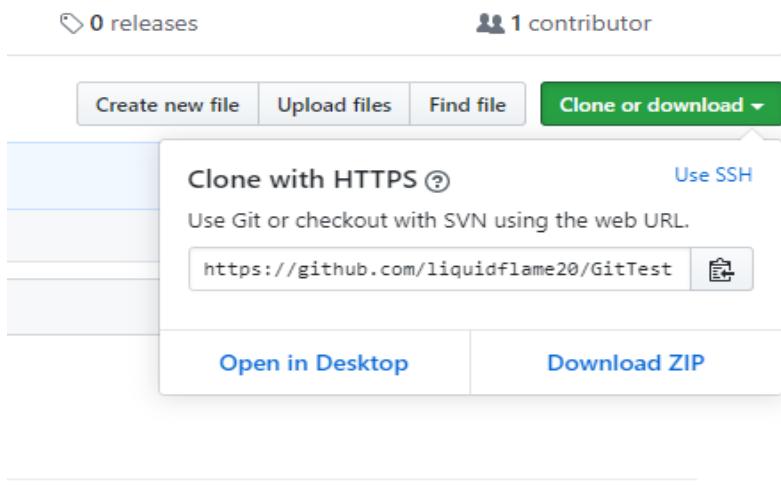
9. git log라는 명령어를 통해 이전에 했던 commit과 버전들에 대해 알 수 있음

※ git log

--> 커밋 기록을 조회, 지금까지의 커밋 기록들이 쭉 출력되며 가장 위에 나오는 내역이 가장 최근 내역임을 알 수 있다. 기록에는 SHA-1 체크섬, 저자 이름, 저자 이메일, 커밋 날짜와 시간, 커밋 메시지가 포함되어 있다.

## 2.3 Git저장소 만들기 2

### ▶ GitHub 등 원격저장소의 데이터를 로컬저장소로 복제



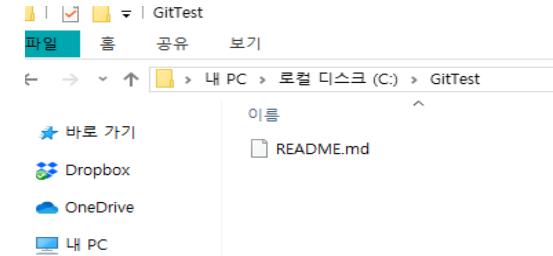
1. 원격저장소인 GitHub에 들어가서 저장소를 하나 만들고 만든 저장소의 주소를 복사함

```
이제 온 @sec MINGW64 /c/Gitpractice2 (master)
$ cd c:

이제 온 @sec MINGW64 /c
$ git clone https://github.com/liquidflame20/GitTest.git
Cloning into 'GitTest'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.

이제 온 @sec MINGW64 /c
$ |
```

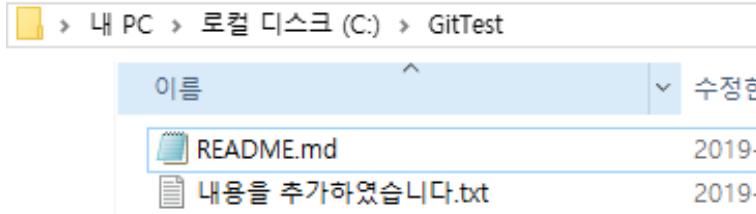
2. 복사할 경로로 이동한 뒤 git clone이라는 명령어를 사용하여 원격저장소를 복제함



(해당 파일이 복제된 것을 확인 할 수 있다)

## 2.3 Git저장소 만들기 2

### ▶ GitHub 등 원격저장소의 데이터를 로컬저장소로 복제



#### 3. 복제한 파일에 내용을 수정, 추가하는 작업을 함

```
이제 콘솔 @sec MINGW64 /c  
$ cd c:GitTest  
  
이제 콘솔 @sec MINGW64 /c/GitTest (master)  
$ git status  
On branch master  
Your branch is up to date with 'origin/master'.  
  
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)  
    ".\353\202\264\354\232\251\354\235\204 \354\266\224\352\260\200\355\225\230\354\230\200\354\212\265\353\213\210\353\213\244.txt"  
  
no changes added to commit (use "git add" and/or "git commit -a")  
  
이제 콘솔 @sec MINGW64 /c/GitTest (master)  
$ |
```

```
이제 콘솔 @sec MINGW64 /c/GitTest (master)  
$ git add *  
  
이제 콘솔 @sec MINGW64 /c/GitTest (master)  
$ git status  
On branch master  
Your branch is up to date with 'origin/master'.  
  
Changes to be committed:  
  (use "git restore --staged <file>..." to unstage)  
    modified: README.md  
    new file: ".\353\202\264\354\232\251\354\235\204 \354\266\224\352\260\200\355\225\230\354\230\200\354\212\265\353\213\210\353\213\244.txt"  
  
이제 콘솔 @sec MINGW64 /c/GitTest (master)  
$ git commit -m "new version file"  
[master 1015362] new version file  
 2 files changed, 3 insertions(+), 1 deletion(-)  
  create mode 100644 ".\353\202\264\354\232\251\354\235\204 \354\266\224\352\260\200\355\225\230\354\230\200\354\212\265\353\213\210\353\213\244.txt"  
  
이제 콘솔 @sec MINGW64 /c/GitTest (master)  
$ |
```

#### 4. 해당 파일에서 git status 명령어를 수행하면 파일의 수정과 추가됨을 확인 할 수 있다. git add를 통해 추가하고 git commit을 통해 확정하도록 함

( git add \* 명령어를 통해 두 개 이상 수정된 내용에 대해 모두 추가 할 수 있다.)

( 수정하고 추가한 내용이 commit 된 것을 확인할 수 있다.)

## 2.3 Git저장소 만들기 2

### ▶ GitHub 등 원격저장소의 데이터를 로컬저장소로 복제

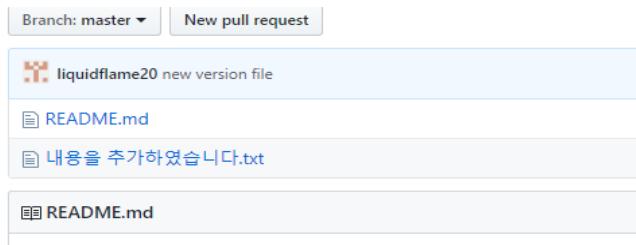
```
이제은@sec MINGW64 /c/GitTest (master)
$ git add *

이제은@sec MINGW64 /c/GitTest (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified: README.md
    new file: "\353\202\264\354\232\251\354\235\204 \354\266\224\352\260\
200\355\225\230\354\230\200\354\212\265\353\213\210\353\213\244.txt"

이제은@sec MINGW64 /c/GitTest (master)
$ git commit -m "new version file"
[master 1015362] new version file
 2 files changed, 3 insertions(+), 1 deletion(-)
 create mode 100644 "\353\202\264\354\232\251\354\235\204 \354\266\224\352\260\
200\355\225\230\354\230\200\354\212\265\353\213\210\353\213\244.txt"

이제은@sec MINGW64 /c/GitTest (master)
$
```



5. 현재 local 저장소에만 확정된것이며 GitHub의 원격저장소에는 추가된 내용이 없다. git push 명령어로 GitHub에도 내용을 갱신하도록 함

(GitHub 아이디, 패스워드를 입력하는 창이 뜨면 로그인 하도록 함)

※ git push

--> local repository의 내용을 remote repository(원격저장소 : GitHub)로 업로드 함

6. GitHub에 push한 내용이 업데이트 된 것을 확인 할 수 있음

## 2.4 Git 명령어

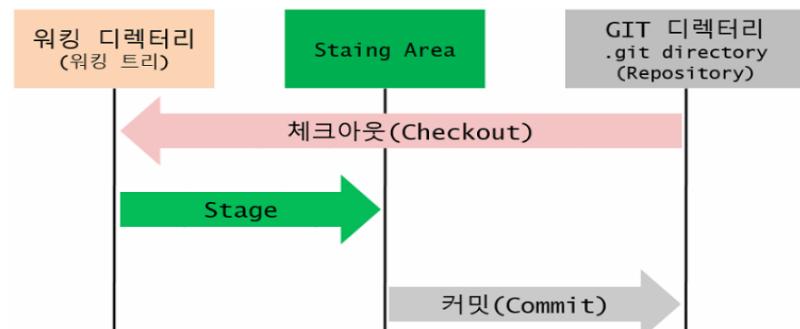
- working directory( working tree(작업 트리) , working space )

- 현재 작업하고 있는 공간, git에서 작업하는 폴더
- git이 관리하고 있지만, 아직 추적(track)하고 있지 않은 상태
- 작업 트리, working directory, working space라 불림

- index (staging area): stage 또는 staging area라고 하며, 준비 공간을 뜻함

- git이 추적하고 있으며, 버전으로 등록되기 전 상태
- 저장소에 커밋하기 전에 Commit을 준비하는 위치

Git 에서 세 가지 영역  
Git 프로젝트는 Git 디렉터리, 워킹 트리, Staging Area 라는 세 가지 영역을 갖게 됩니다.  
Git 프로젝트에서 파일들은 아래 세 가지 영역별로 다양한 상태를 가지게 됩니다.



- repository(저장소): 저장소를 의미

- 본인 컴퓨터의 local repository와 GitHub, Gitlab같은 원격 저장소인 remote repository 가 있음

## 2.4 Git 명령어

---

- branch: 개발의 한 갈래, 분기
  - > 깃 저장소를 만들면 기본적으로 main branch가 생성됨
  - > main branch 파일 내용을 그대로 복사한 새로운 branch를 만들어서 내용을 수정, 추가 작업을 한 뒤 main branch와 병합 시킬 수 있음
  - > 독립된 working directory를 의미 (master로부터 분리된)
  - > branch를 통해 프로젝트 참여자마다 branch를 가져서 독립된 작업공간을 갖음
- checkout: 현재 작업중인 워킹 디렉터리(워킹 트리)의 일부 혹은 전체를 업데이트 하는 것
  - > 작업자의 작업트리(작업중인 폴더)를 저장소(repository)의 특정 시점과 일치하도록 변경하는 작업
- commit(커밋): 어떤 순간 작업공간의 상태를 저장한 것.
  - > 작업공간 안에 있는 모든 파일과 파일의 데이터를 사진 찍듯이 복사해서 저장소에 보존함  
즉, commit은 작업공간의 어떤 시점의 스냅샷이라고 할 수 있음
  - > git에서는 파일별로 추적(track)을 하는데 이를 위해서는 한번이라도 commit을 하여야 함
- Fetch: 원격저장소의 변경사항을 가져옴
- Pull: 원격저장소의 변경사항을 가져와 지역저장소에 병합 (Fetch + Merge)

## 2.4 Git 명령어

---

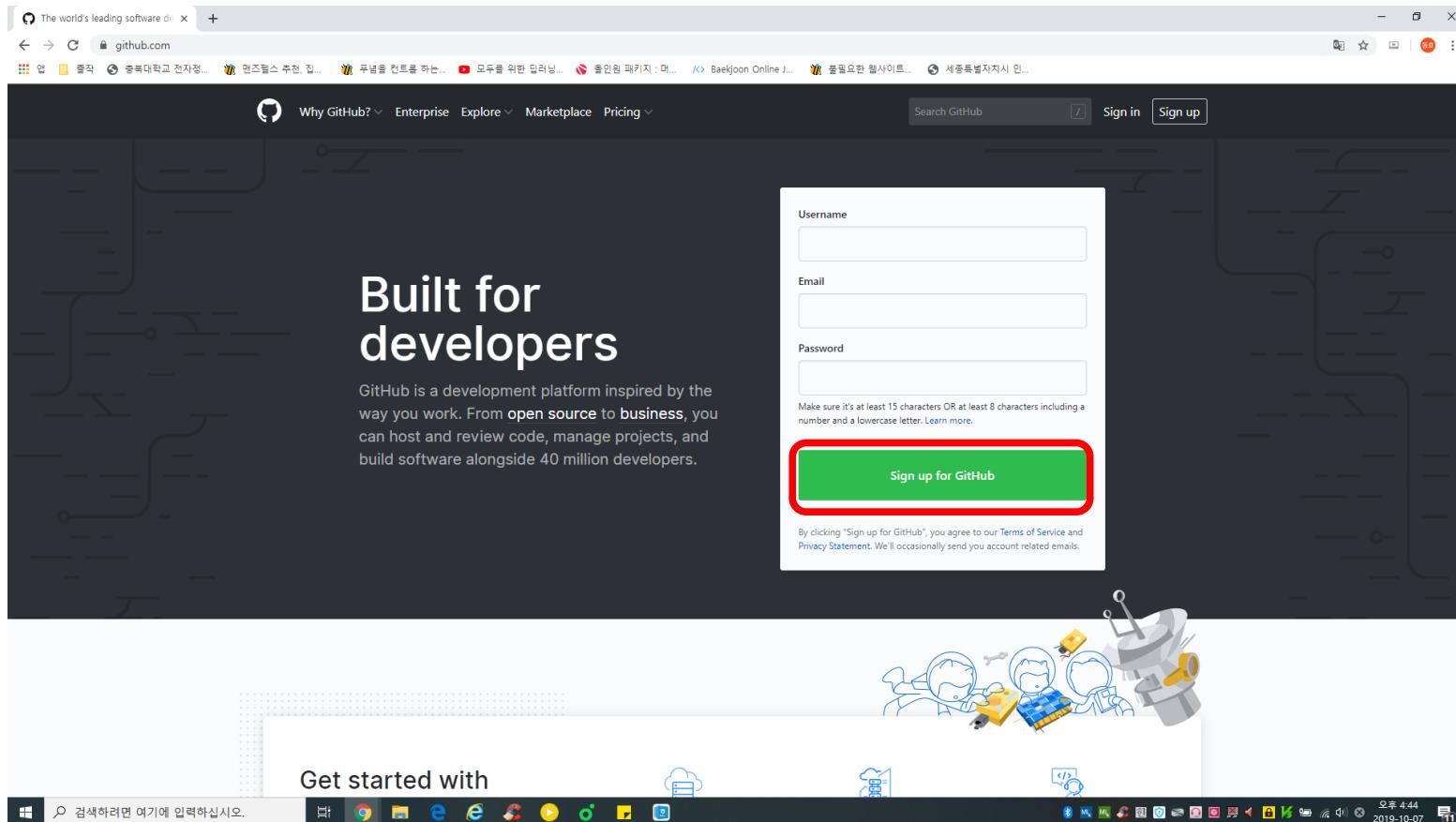
- Push: 확정된 변경 사항을 원격 저장소에 게시하는 것
- HEAD(대문자): 간단히 말해 현재 branch를 말함. HEAD는 현재 체크 아웃된 commit을 가리킴 (현재 작업중인 commit)
  - > 'HEAD'란 현재 사용 중인 branch의 선두 부분을 나타내는 이름으로 기본적 'master'의 선두 부분을 나타냄. 'HEAD'를 이동하면, 사용하는 Branch가 변경됨
  - > branch는 여러 개가 있지만 HEAD는 단 하나의 branch만을 가리킴  
(git checkout 명령으로 새로 만든 branch로 이동할 수 있음)  
※ testing 브랜치로 이동하려면 오른쪽과 같이함 \$ git checkout
- merge: 2개의 branch에서 작업한 다른 내용을 하나로 합치는 것을 의미, 현재 branch를 기준으로 병합됨
  - > 만약 두 branch가 같은 파일의 같은 곳을 수정했다면, 충돌(merge conflict)이 발생하므로 해결해야함

※ 대부분의 용어들이 뒷 부분 GitHub에서도 같은 의미의 용어들입니다. 참조하시기 바랍니다.

# 3.1 GitHub 회원가입 및 로그인

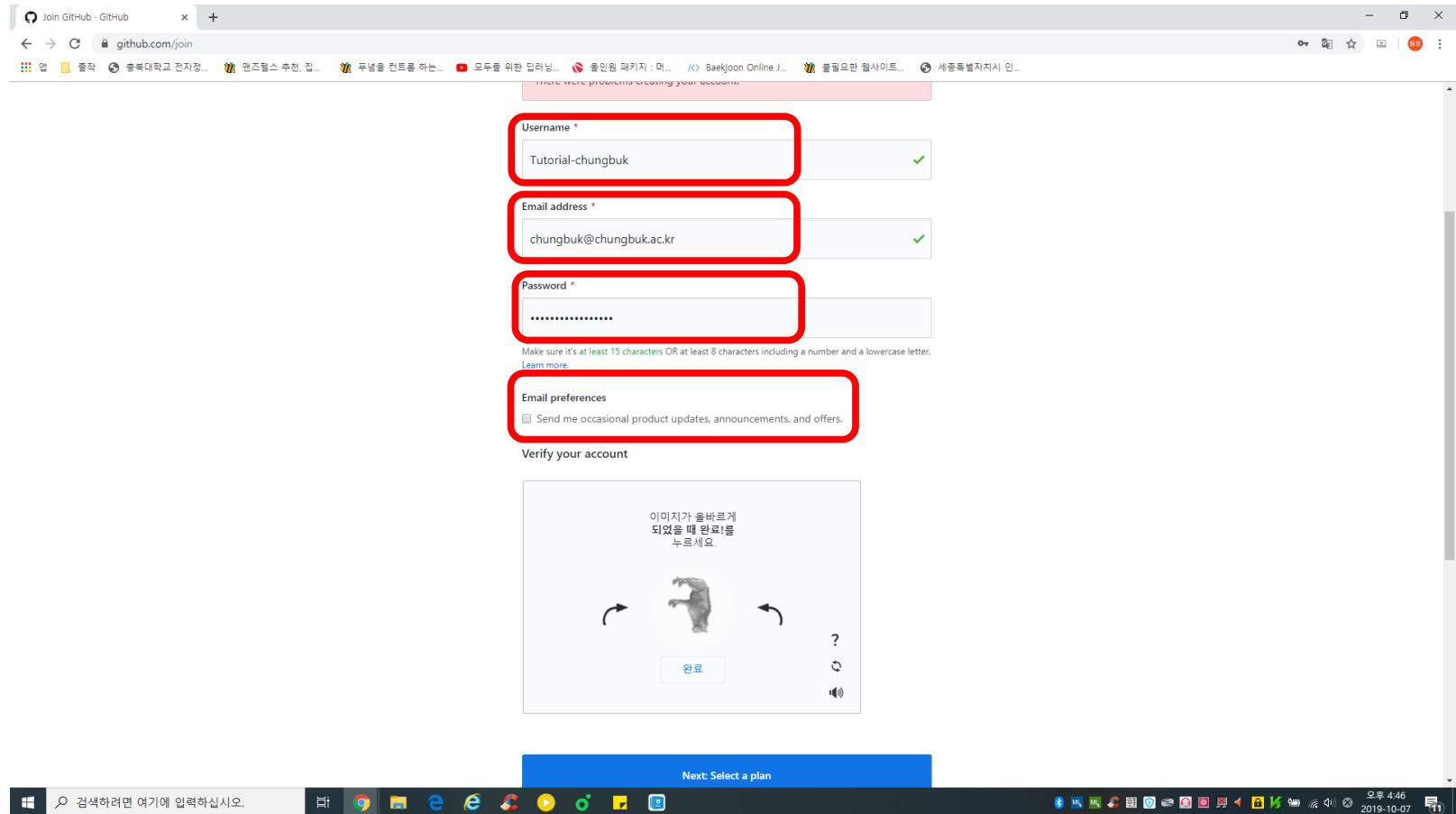
## 1. 회원가입을 위해 Sign up for GitHub를 클릭

※ github.com 접속할 것!



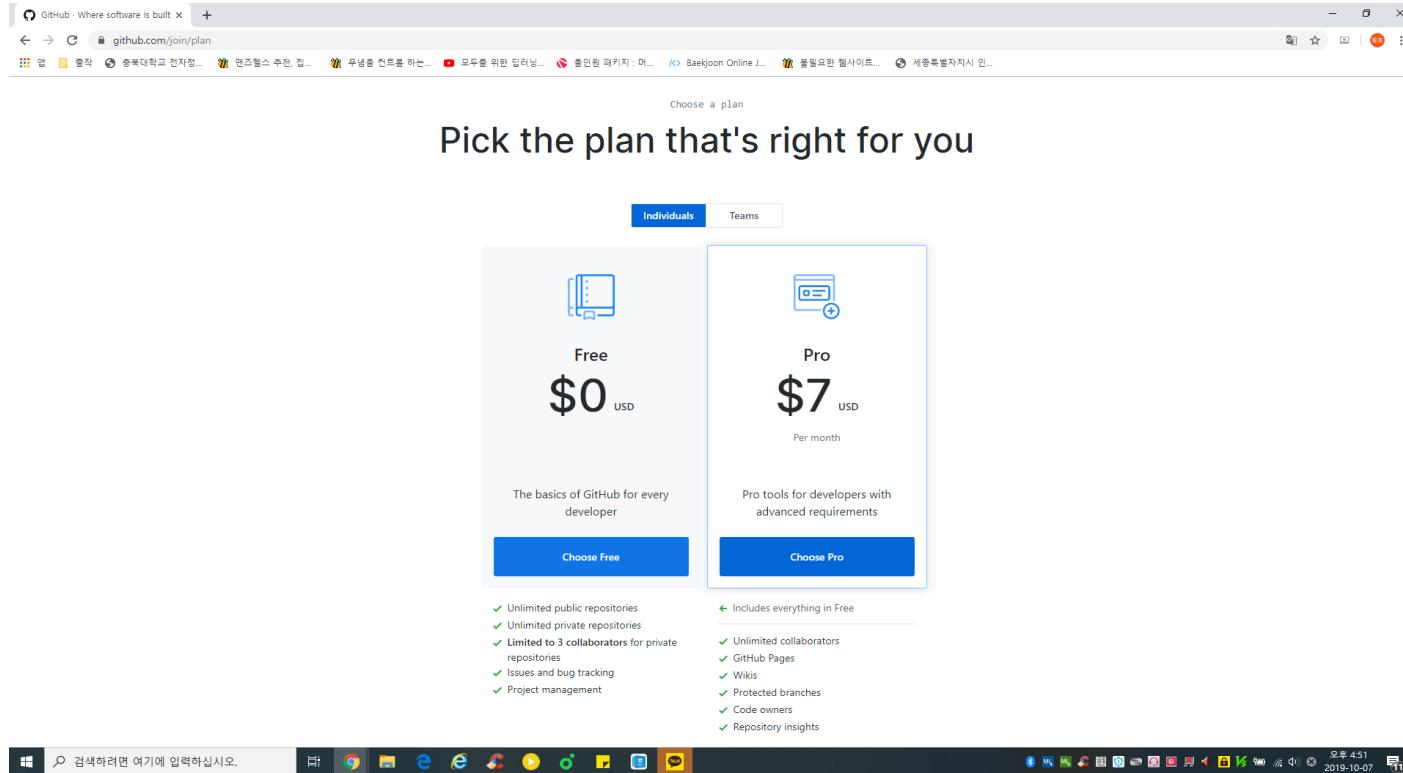
# 3.1 GitHub 회원가입 및 로그인

## 2. 기본사항 입력(업데이트 등 기타 메일 수신 등의 시 박스 체크할 것)



# 3.1 GitHub 회원가입 및 로그인

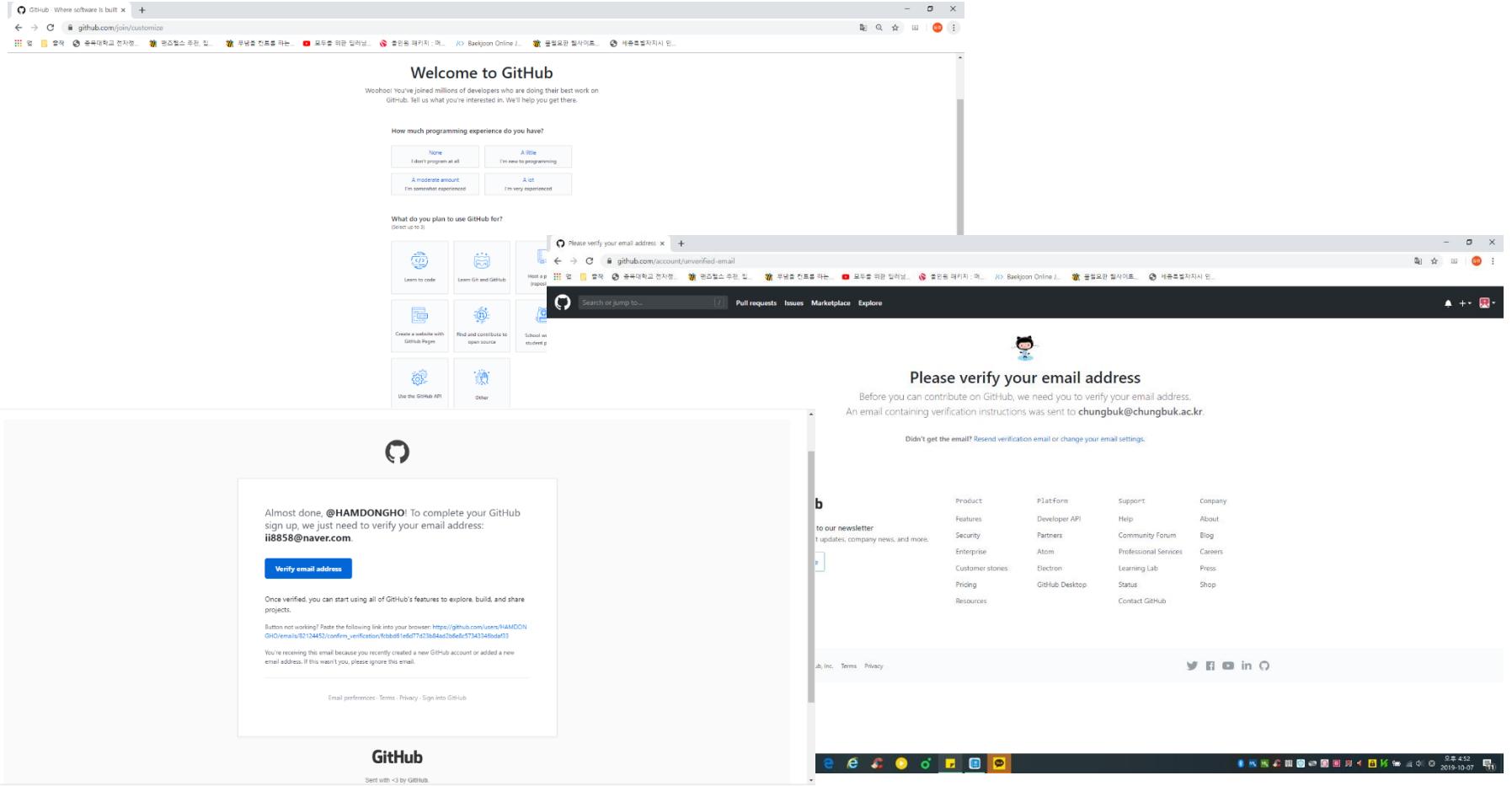
## 3. 무료, 유료 사용자 구분



- 공개 저장소로만 사용하는 경우 무료이며, 비밀 저장소로 사용하는 경우 비용을 지불해야 함  
→ 무료 저장소로 이용하는 경우 Unlimited public repositories for free에 체크하고 Continue를 클릭

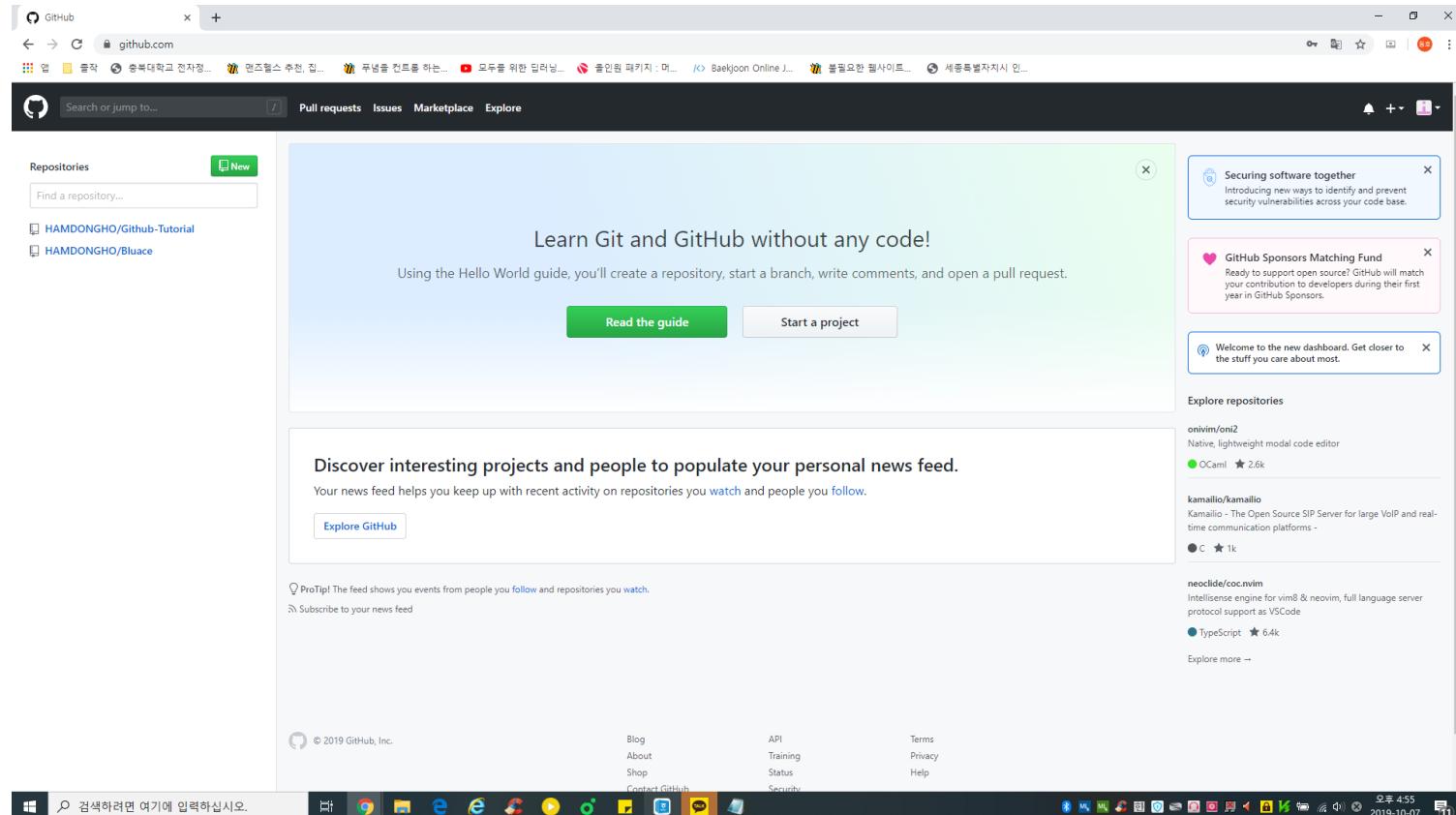
# 3.1 GitHub 회원가입 및 로그인

## 4. 간단한 설문조사(스킵가능) 및 이메일 인증



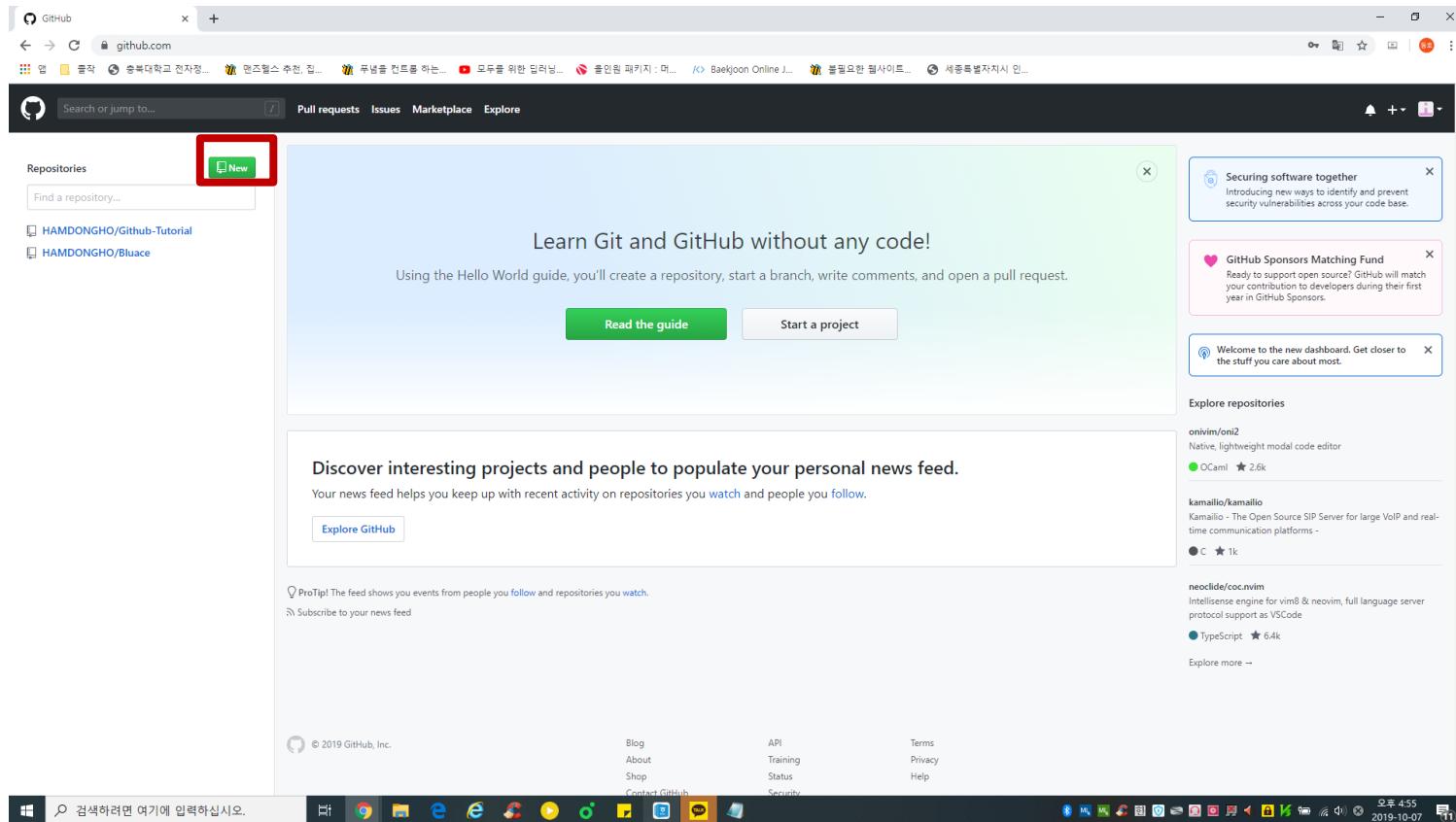
# 3.1 GitHub 회원가입 및 로그인

## 5. 가입완료 후 로그인 된 화면



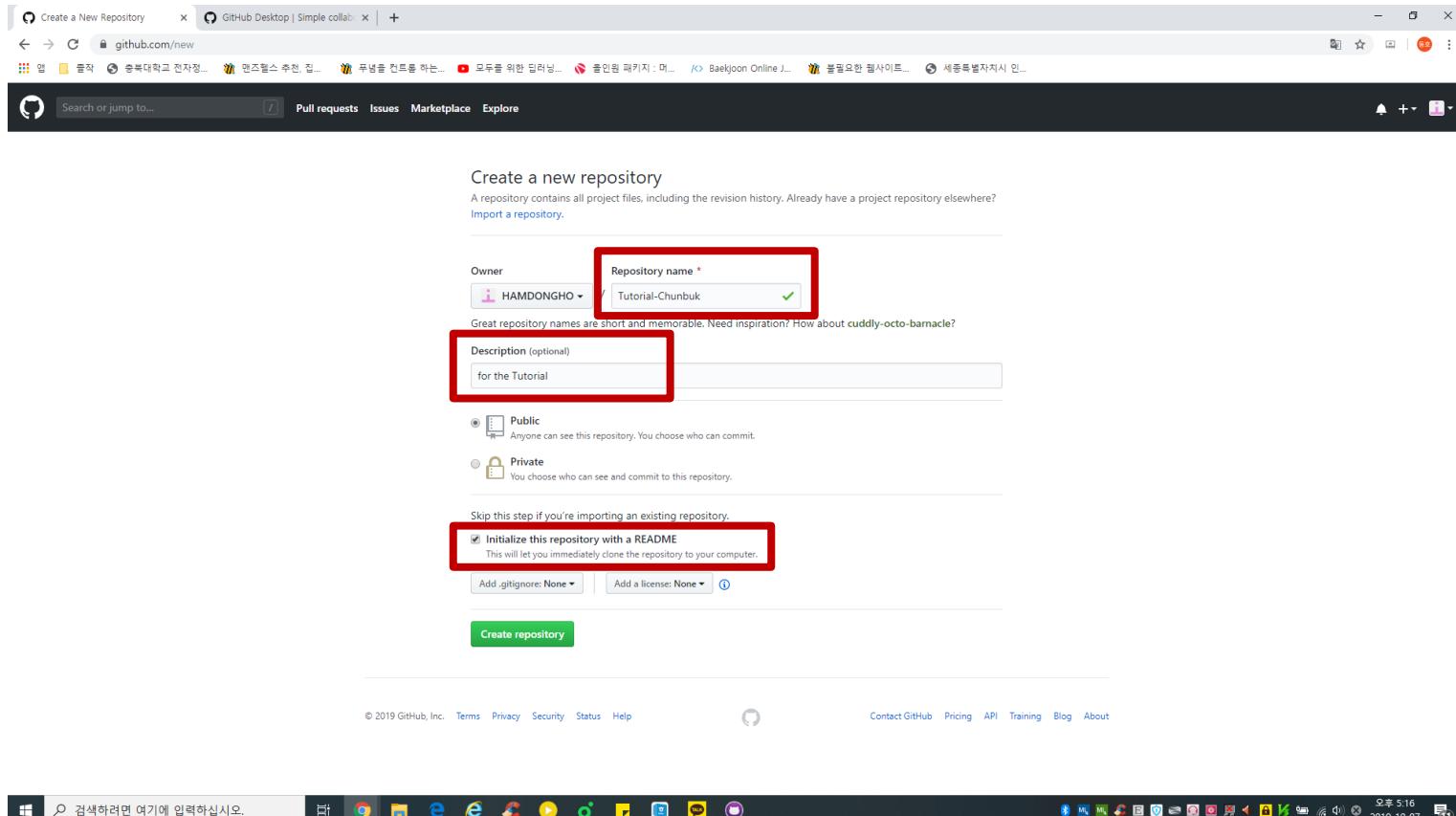
# 3.2 GitHub Repository 생성

## 1. 새로운 저장소를 생성하기 위해 왼쪽 상단의 New 클릭



# 3.2 GitHub Repository 생성

## 2. 저장소 설정

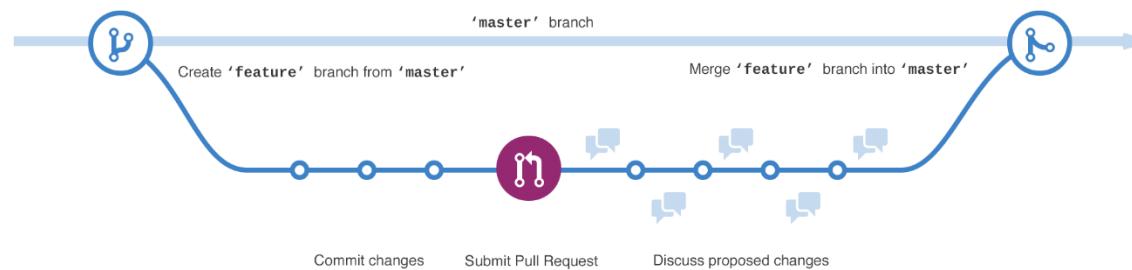


- 저장소 이름 및 간단한 설명을 적고, README를 사용하여 저장소 초기화를 선택함

# 3.3 GitHub Branch 생성

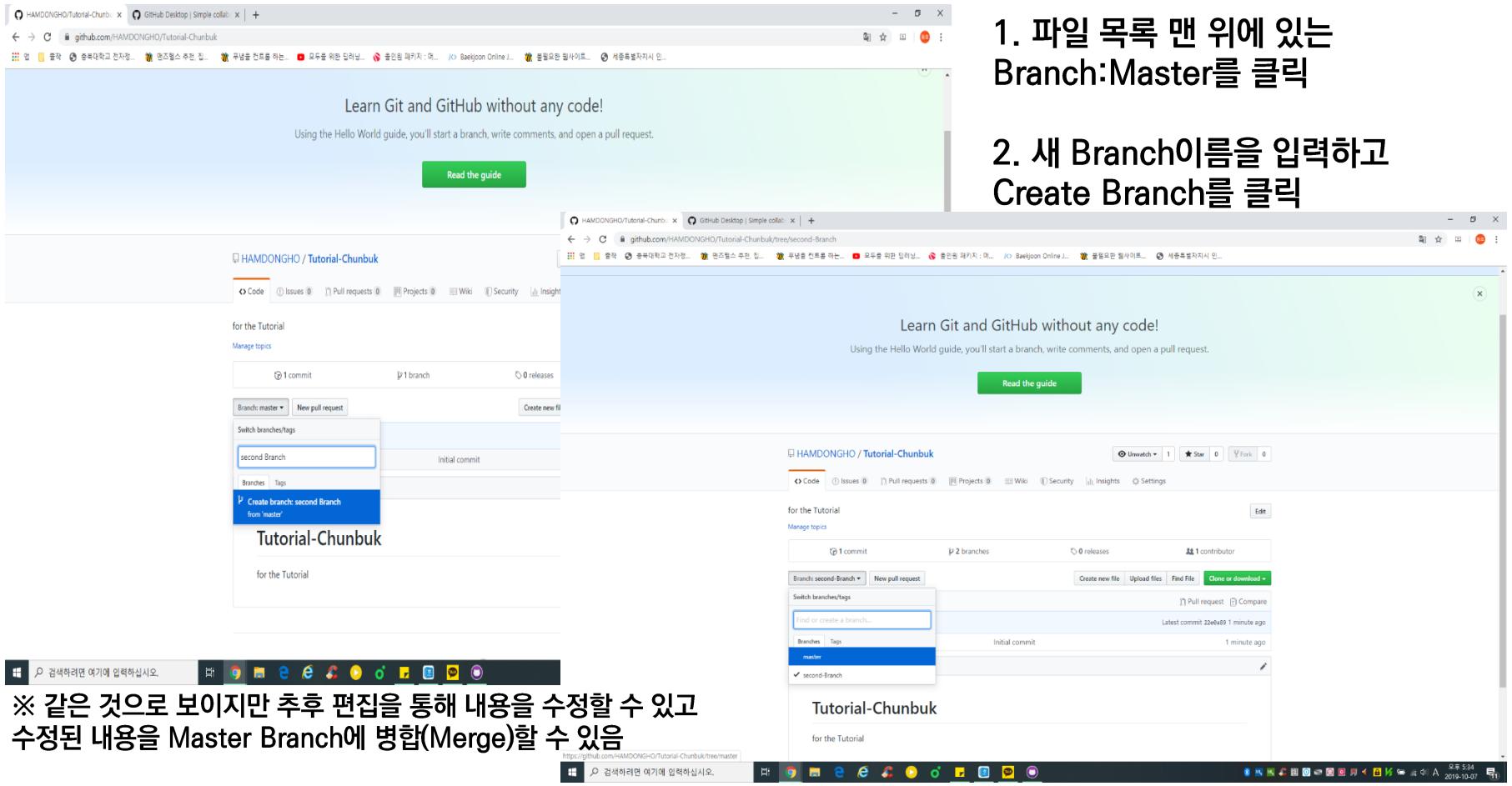
## 1. Branch란??

- 한 번에 서로 다른 버전의 Repository에서 작업하는 방법  
(서로 다른 버전이란, 작업 중 서로 다른 개발자들이 내용을 수정 및 추가하면서 다른 내용을 갖는 파일)
- 기본적으로 저장소에 Master라는 하나의 분기(Master Branch)가 있고, 우리는 Branch를 생성하여 Master에 합치기 전에 코드를 관리
- EX) Chungbuk.txt, OSS senter.txt  
위와 같은 Branch는 GitHub 저장소(Repository)에서 유사한 목적을 갖고 작성된 파일들. 완성된다면, Master Branch로 병합을 할 수 있음
- Master Branch에서 Branch를 생성하면 해당 시점의 Master copy 또는 snap shot이 생성됨
- 특정 Branch에서 작업하는 동안 다른 사람이 Master지점을 변경 한 경우 해당 업데이트를 가져올 수 있음



# 3.3 GitHub Branch 생성

## 2. Branch생성



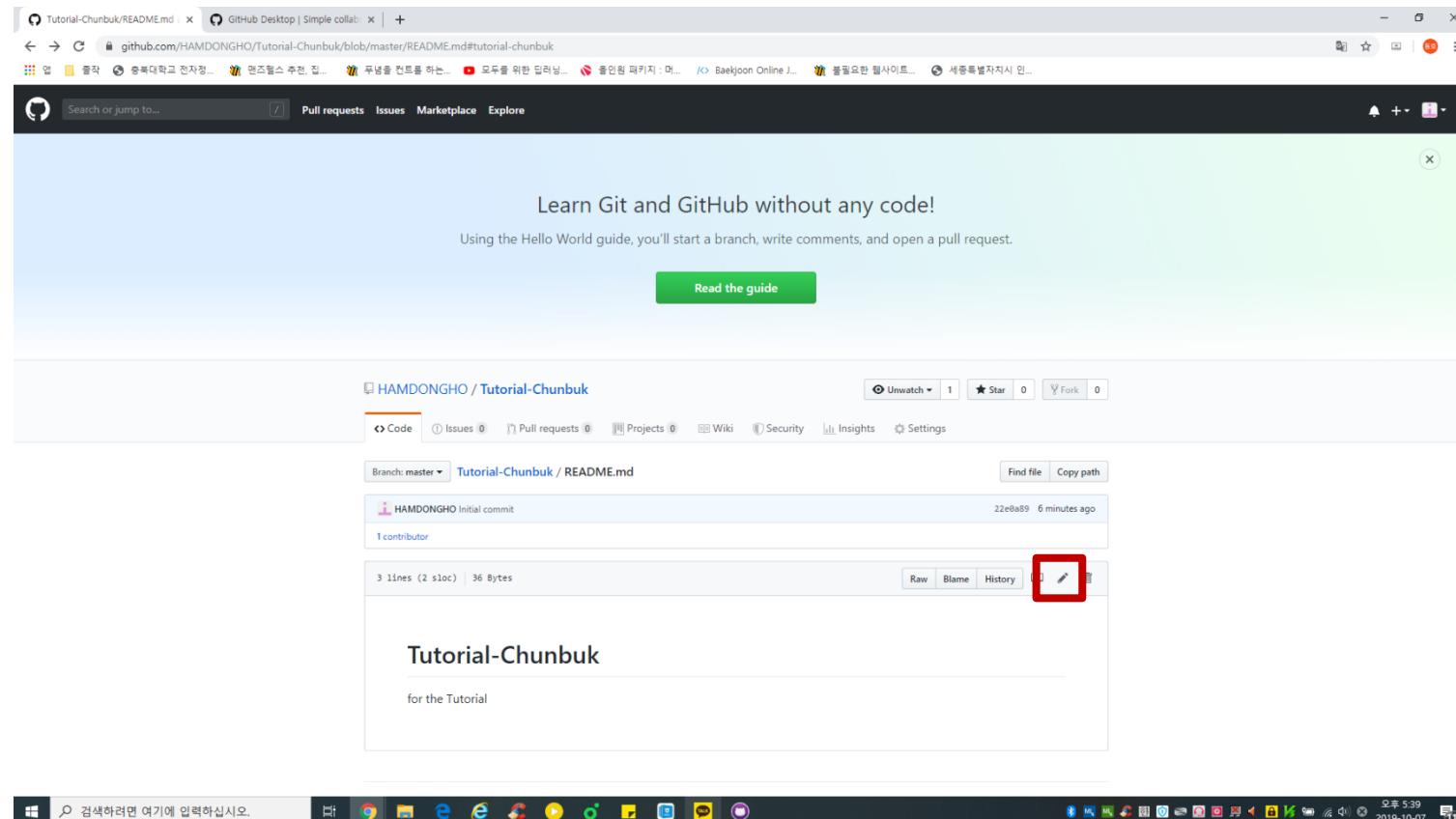
1. 파일 목록 맨 위에 있는 Branch:Master를 클릭

2. 새 Branch이름을 입력하고 Create Branch를 클릭

※ 같은 것으로 보이지만 추후 편집을 통해 내용을 수정할 수 있고 수정된 내용을 Master Branch에 병합(Merge)할 수 있음

# 3.4 GitHub Commit

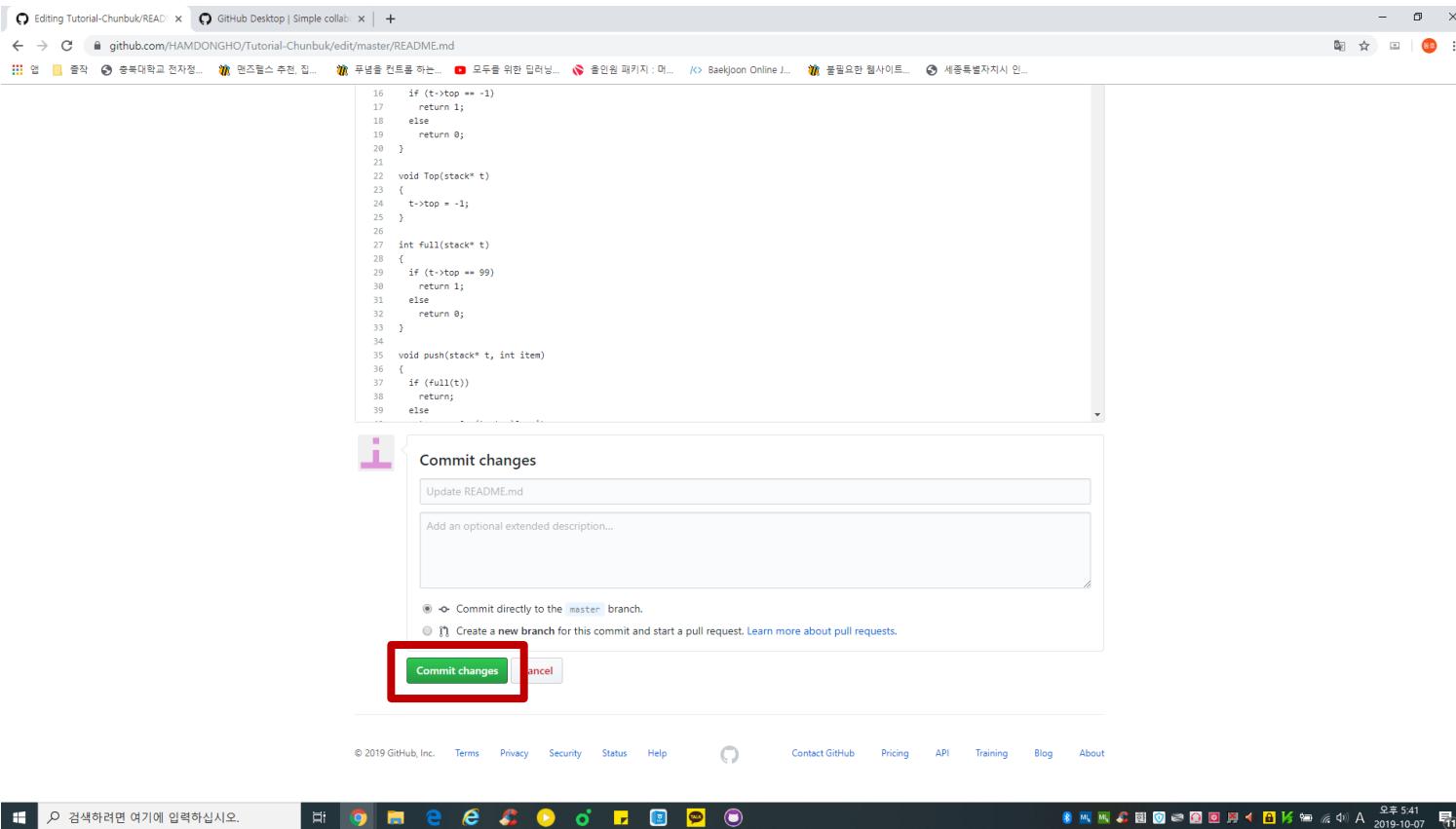
## 1. Commit



1. 수정을 하기 위하여 중앙에서 오른쪽 부근 연필모양을 클릭하고 수정

# 3.4 GitHub Commit

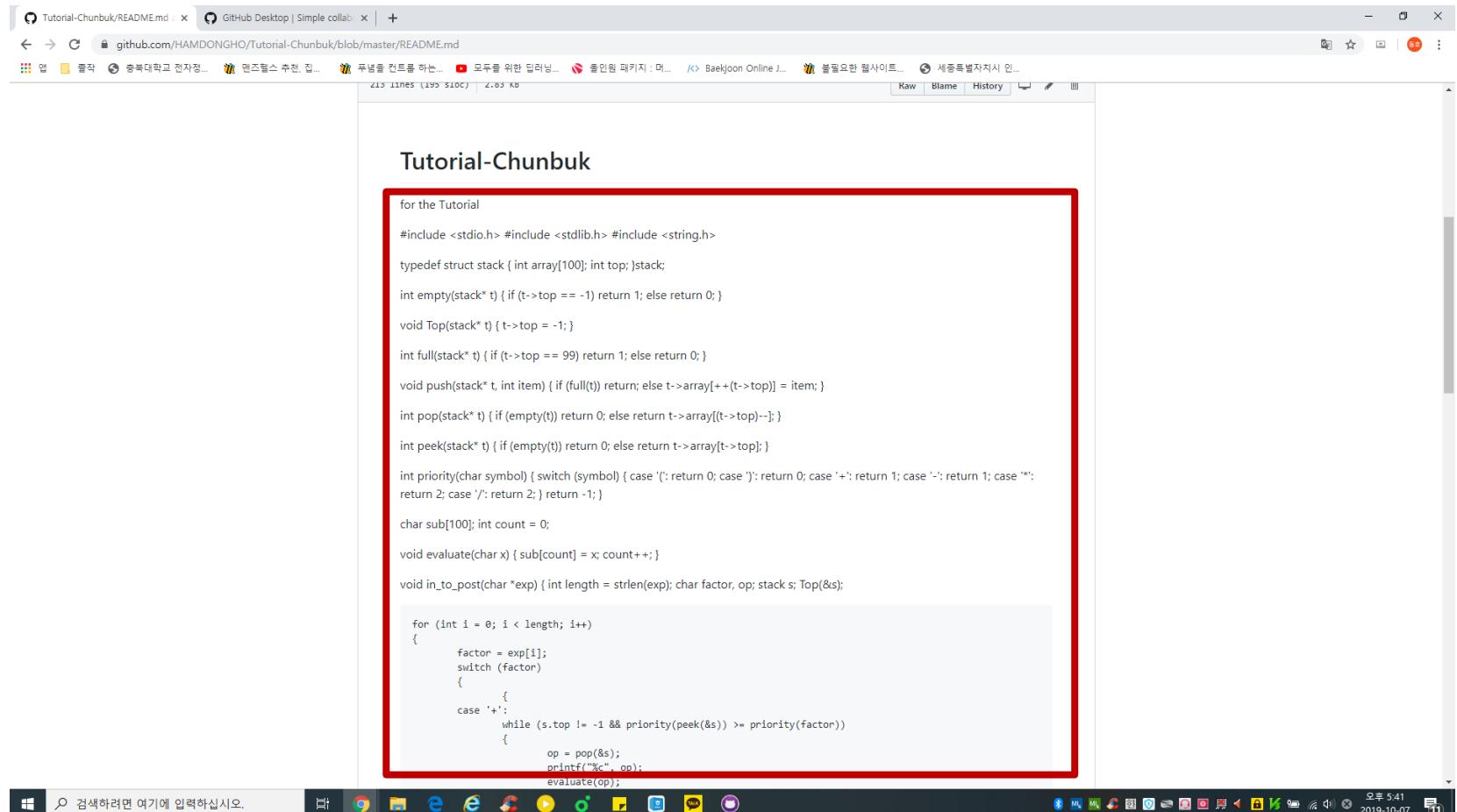
## 1. Commit



2. 수정을 완료했다면 Commit Change를 클릭

# 3.4 GitHub Commit

## 2. Commit Change가 완료되어 적용된 모습



The screenshot shows a Windows desktop environment with a browser window open to a GitHub commit page. The commit has been highlighted with a red border. The code in the commit is as follows:

```
for the Tutorial

#include <stdio.h> #include <stdlib.h> #include <string.h>

typedef struct stack { int array[100]; int top; }stack;

int empty(stack* t) { if (t->top == -1) return 1; else return 0; }

void Top(stack* t) { t->top = -1; }

int full(stack* t) { if (t->top == 99) return 1; else return 0; }

void push(stack* t, int item) { if (full(t)) return; else t->array[++(t->top)] = item; }

int pop(stack* t) { if (empty(t)) return 0; else return t->array[(t->top)--]; }

int peek(stack* t) { if (empty(t)) return 0; else return t->array[t->top]; }

int priority(char symbol) { switch (symbol) { case '(': return 0; case ')': return 0; case '+': return 1; case '-': return 1; case '*': return 2; case '/': return 2; } return -1; }

char sub[100]; int count = 0;

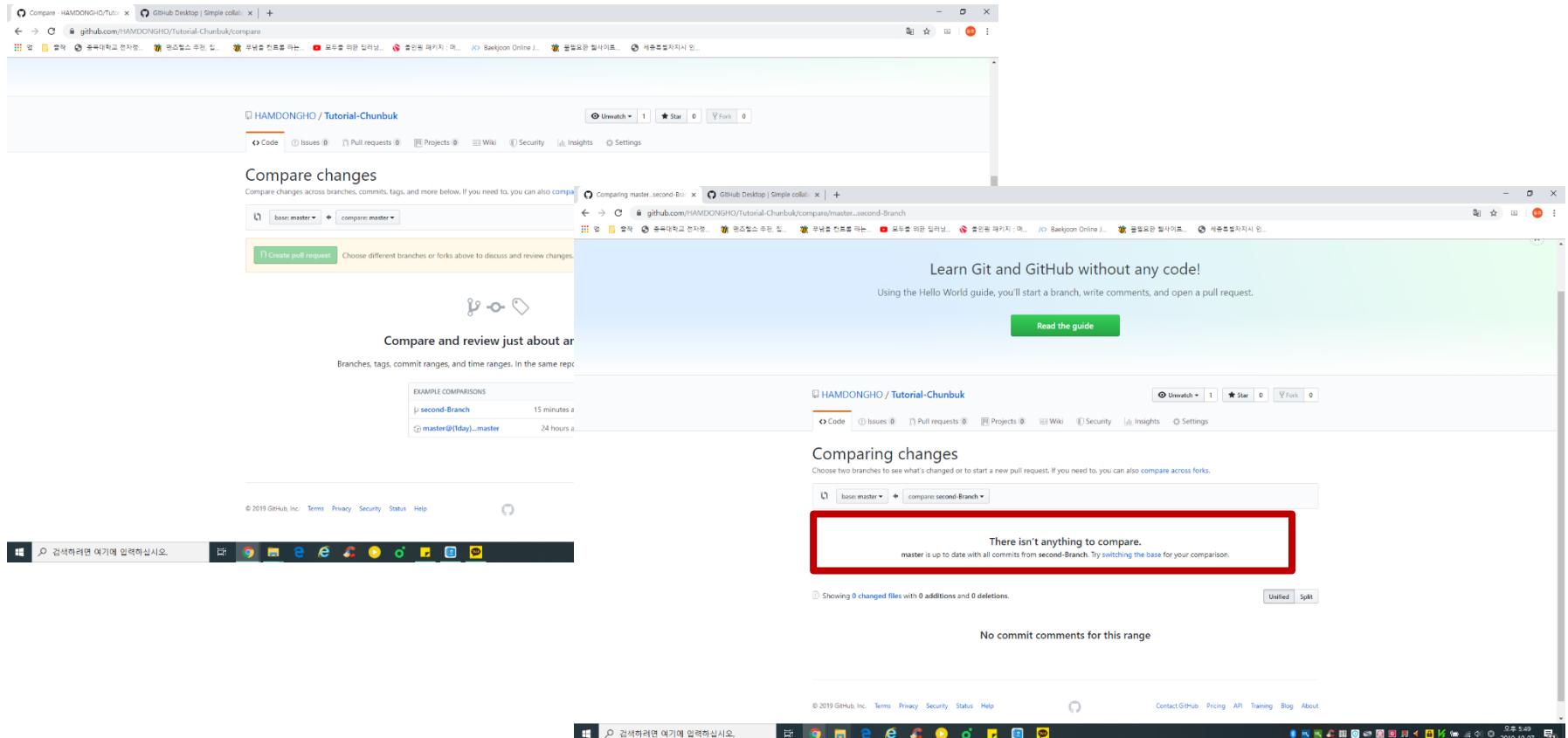
void evaluate(char x) { sub[count] = x; count++; }

void in_to_post(char *exp) { int length = strlen(exp); char factor, op; stack s; Top(&s);

    for (int i = 0; i < length; i++)
    {
        factor = exp[i];
        switch (factor)
        {
            case '+':
                while (s.top != -1 && priority(peek(&s)) >= priority(factor))
                {
                    op = pop(&s);
                    printf("%c", op);
                    evaluate(op);
                }
        }
    }
}
```

# 3.5 GitHub Pull Request, Merge

## 1. Branch에서의 수정부분 없을 경우



- Branch에서 수정이 안되었을 경우 compare하여 바뀐 부분이 없다고 나옴(Merge 불가)

# 3.5 GitHub Pull Request, Merge

## 2. Branch에서의 수정이 있을 경우

The screenshot shows a GitHub comparison page for the repository HAMDONGHO/Tutorial-Chunbuk. The base branch is set to 'master' and the compare branch is 'second-Branch'. A message indicates that automatic merge is not possible. A green button labeled 'Create pull request' is visible. The commit history shows one commit from Oct 07, 2019, by user HAMDONGHO, which updated README.md. The diff view shows the addition of a line: '# Tutorial-Chunbuk for the Tutorial + Im so exciting!'. Below the commit, it says 'No commit comments for this range'. At the bottom, there are links for GitHub's footer and a Windows taskbar showing various open applications.

Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also compare across forks.

base: master ▾ compare: second-Branch X Can't automatically merge. Don't worry, you can still create the pull request.

Create pull request Discuss and review the changes in this comparison with others.

1 commit 1 file changed 0 commit comments 1 contributor

Commits on Oct 07, 2019

HAMDONGHO Update README.md Verified 6bd62e9

Showing 1 changed file with 1 addition and 0 deletions.

Unified Split

1 README.md

... ... @@ -1,2 +1,3 @@  
1 # Tutorial-Chunbuk  
2 for the Tutorial  
3 + Im so exciting!

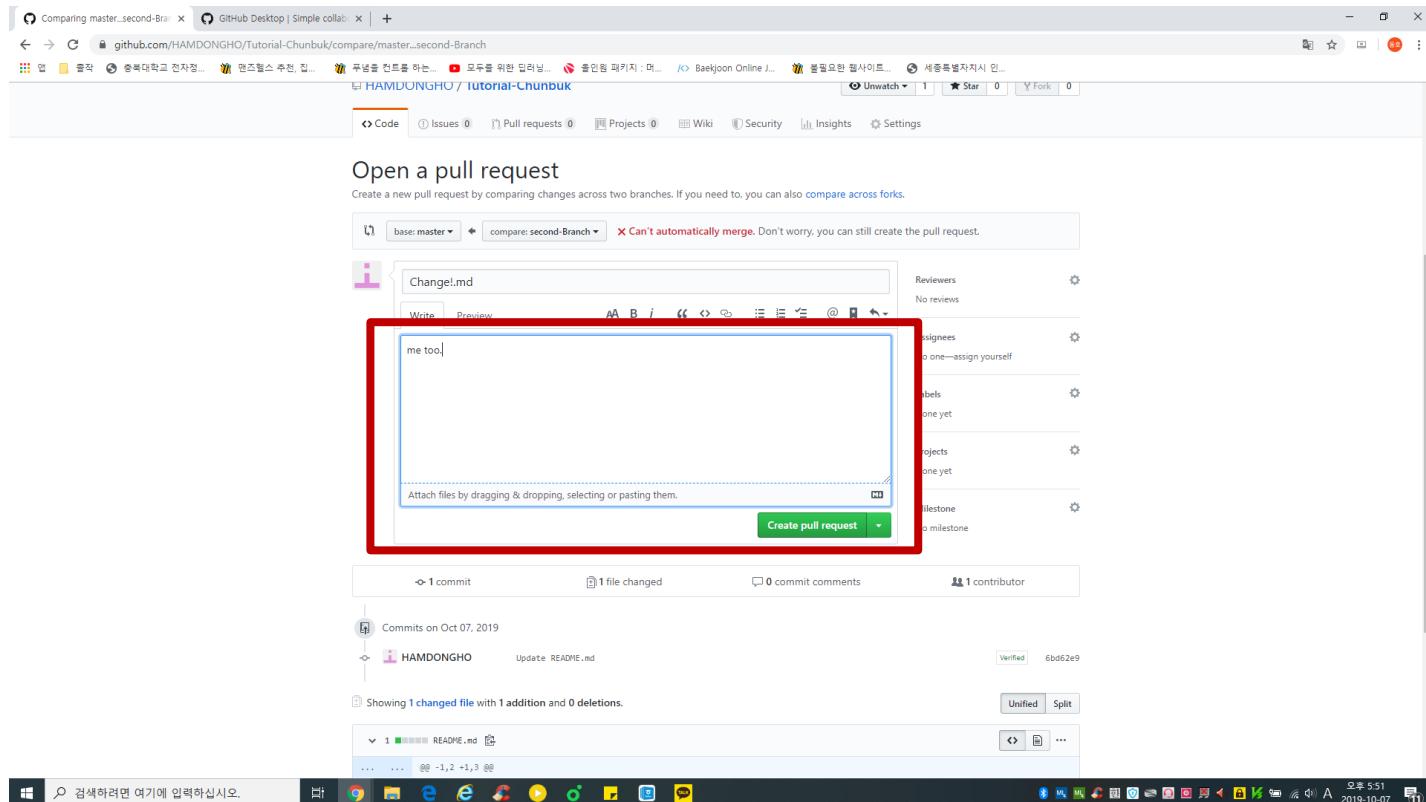
No commit comments for this range

© 2019 GitHub, Inc. Terms Privacy Security Status Help Contact GitHub Pricing API Training Blog About

검색하려면 여기에 입력하십시오. 오전 5:50 2019-10-07

# 3.5 GitHub Pull Request, Merge

## 2. Branch에서의 수정이 있을 경우



- 다른 Branch에서 수정된 부분이 있는 상태에서 Pull Request을 할 경우 아래와 같은 화면으로 넘어가고 Master에서 Requests에 제목을 붙이고 변경사항에 대한 간략한 설명을 쓸 수 있음

# 3.5 GitHub Pull Request , Merge

## 2. Branch에서의 수정이 있을 경우

The screenshot shows a GitHub repository page for 'HAMDONGHO / Tutorial-Chunbuk'. The main header says 'Learn Git and GitHub without any code!' and 'Using the Hello World guide, you'll start a branch, write comments, and open a pull request.' A green button labeled 'Read the guide' is visible. Below the header, the repository name 'HAMDONGHO / Tutorial-Chunbuk' is shown along with statistics: 0 Code, 0 Issues, 1 Pull requests, 0 Projects, 0 Wiki, 0 Security, 0 Insights, and 0 Settings. The 'Pull requests' tab is selected, showing a single pull request titled 'Change!.md #1'. The pull request details show 'HAMDONGHO wants to merge 1 commit into master from second-Branch'. The commit message is 'me too.' and it has been verified. The commit hash is 6bd62e9. The pull request status is '+1 -0' with a green bar. On the right side, there are sections for 'Reviewers' (No reviews), 'Assignees' (No one—assign yourself), 'Labels' (None yet), and 'Projects' (None yet). A warning message at the bottom left says 'This branch has conflicts that must be resolved' with a 'Resolve conflicts' button. The overall interface is light blue and white.

- 설명 추가 후 다음으로 넘어 간 화면

# 3.5 GitHub Pull Request, Merge

## 2. Branch에서의 수정이 있을 경우

Change!.md #1

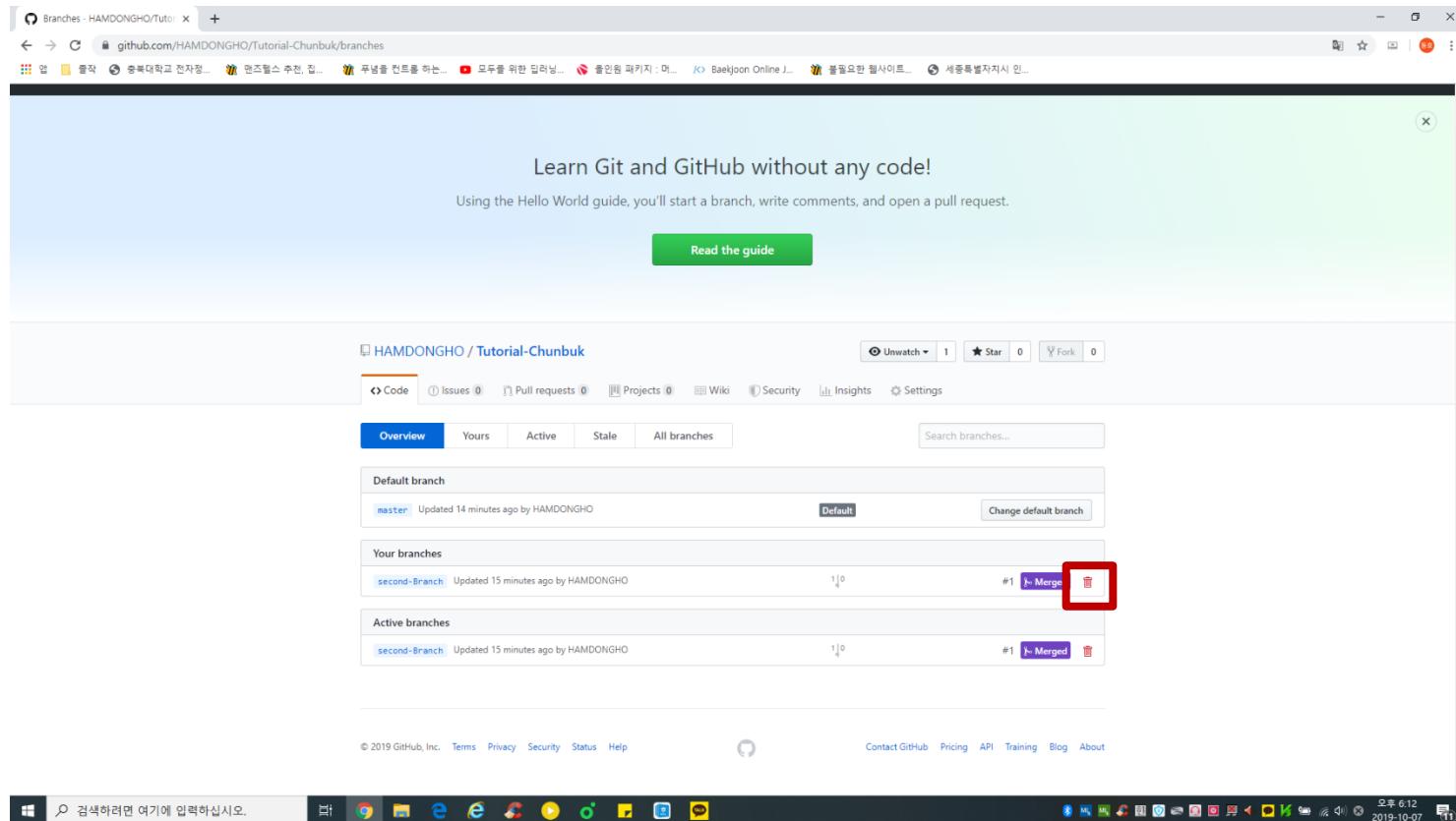
A screenshot of a GitHub pull request page for a file named 'Change!.md'. The pull request is from the branch 'second-Branch' into the 'master' branch. The commit message is 'me too.' There are two commits listed: 'Update README.md' and 'Merge branch 'master' into second-Branch'. A red box highlights the 'Merge pull request' button at the bottom left of the main content area.

A screenshot of the GitHub interface showing the merge process. It includes a browser tab for 'Change!.md by HAMDONGHO', a GitHub Desktop window, and a terminal window titled 'github resolve conflicts : 네이버...'. The terminal shows the command 'git merge master'. The GitHub interface shows the merge being completed, with a message 'HAMDONGHO merged 2 commits into master from second-Branch now'. Below it, another message indicates the pull request was successfully merged and closed, with a 'Delete branch' button.

- 수정된 부분에서 기존 Master Branch와의 충돌되는 부분이 없다면, Merge를 클릭함으로 병합을 완료하게 됩니다.

# 3.5 GitHub Pull Request, Merge

## 3. Branch 삭제

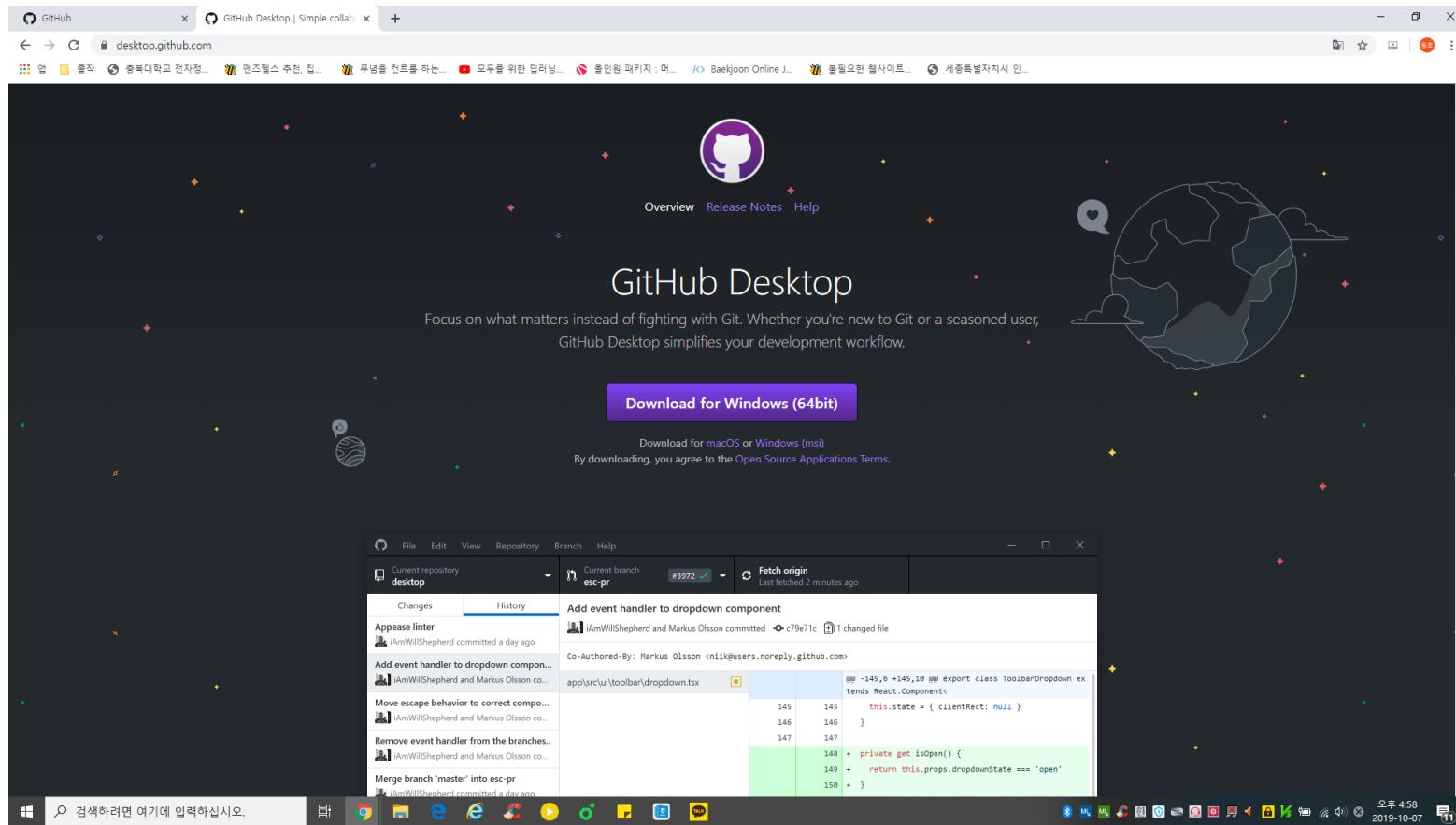


- Code에 들어가서 오른쪽의 붉은색 휴지통 모양을 통해, 특정한 Branch를 삭제할 수 있음

# 4.1 GitHub Desktop 설치 및 계정등록

## 1. GitHub Desktop 설치

※ <https://desktop.github.com/>로 자신에 맞는 운영체제로 설치할 수 있음



# 4.1 GitHub Desktop 설치 및 계정등록

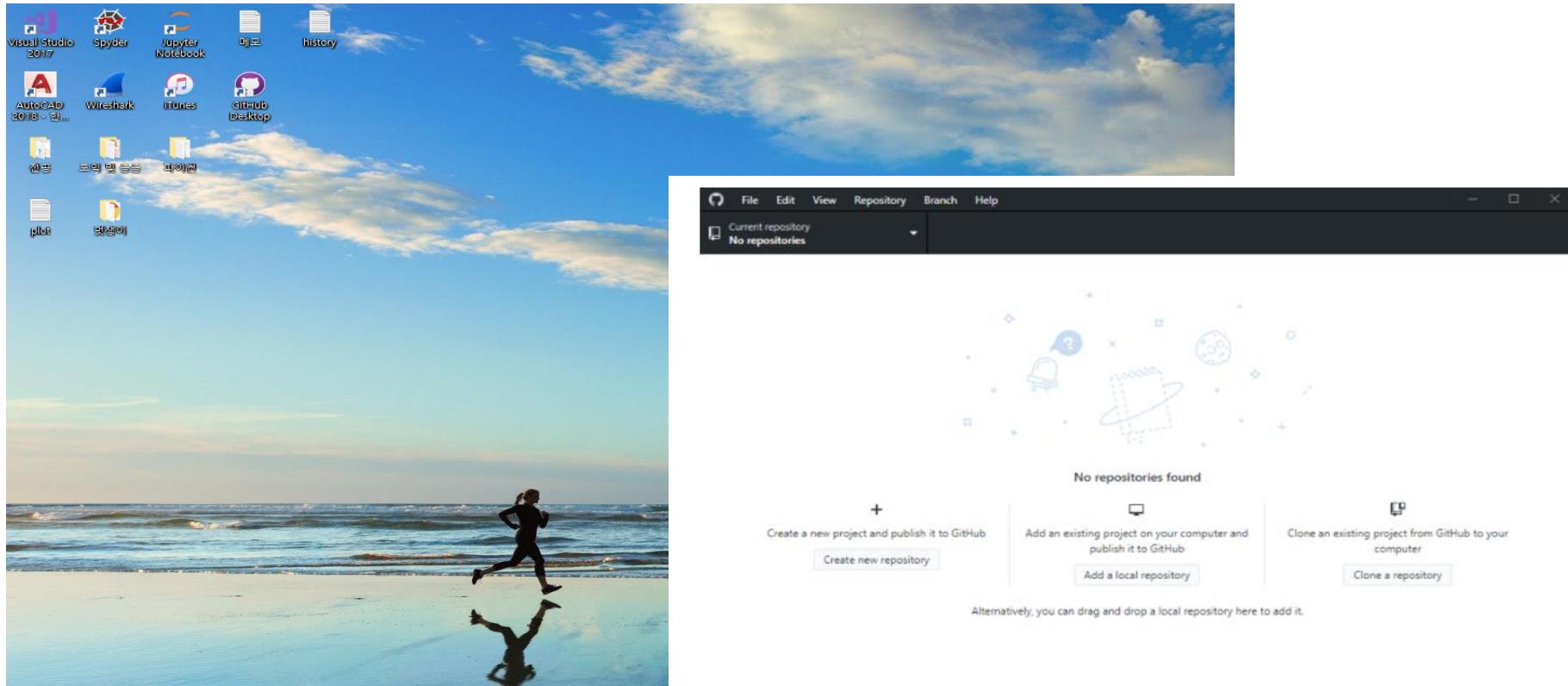
## 1. GitHub Desktop 설치



- GitHub Desktop을 설치하면, 아이콘 생성됨

# 4.1 GitHub Desktop 설치 및 계정등록

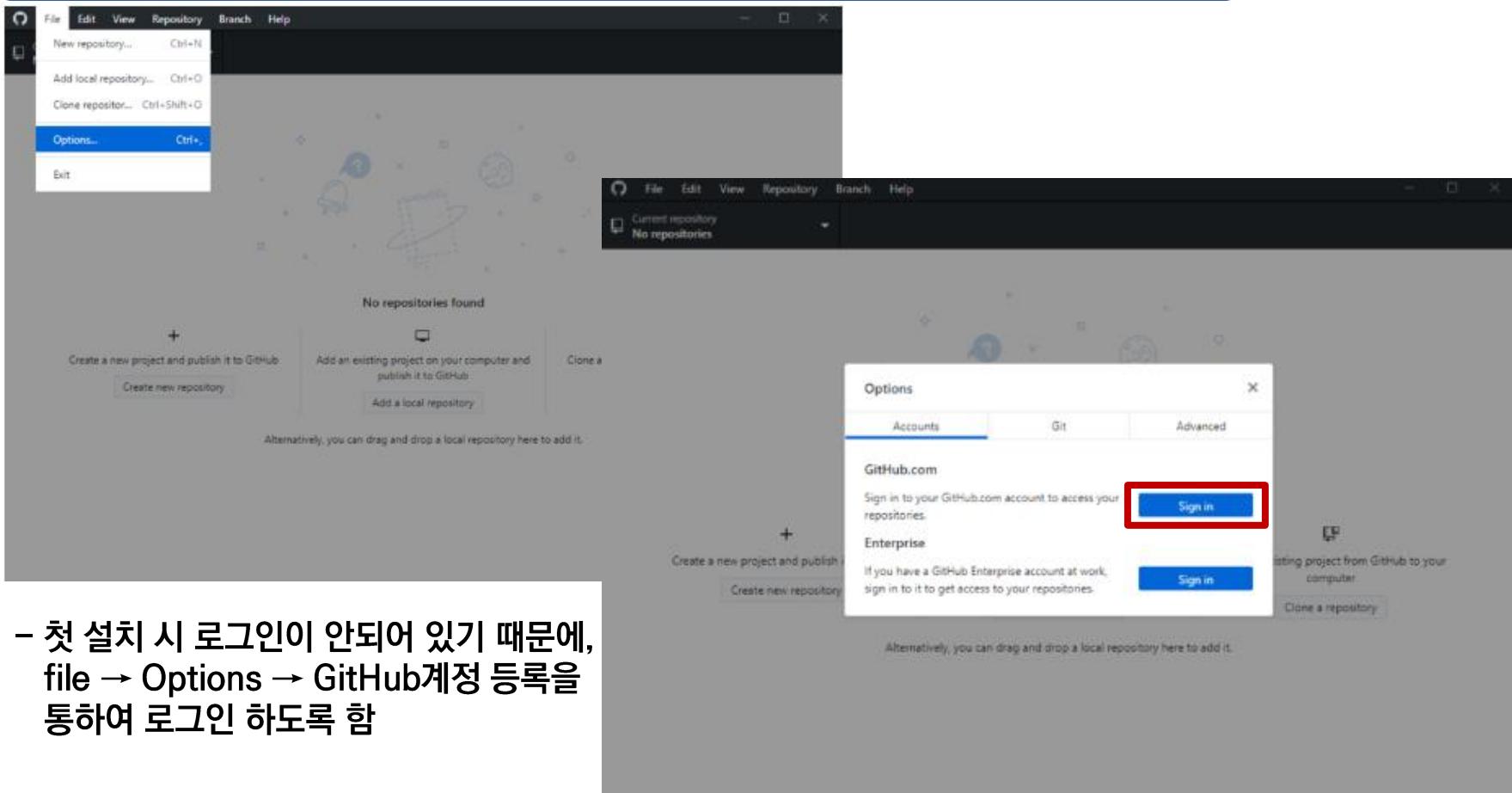
## 1. GitHub Desktop 설치



- GitHub Desktop을 설치하면, 아이콘 생성됨
- 실행화면은 오른쪽과 같음

# 4.1 GitHub Desktop 설치 및 계정등록

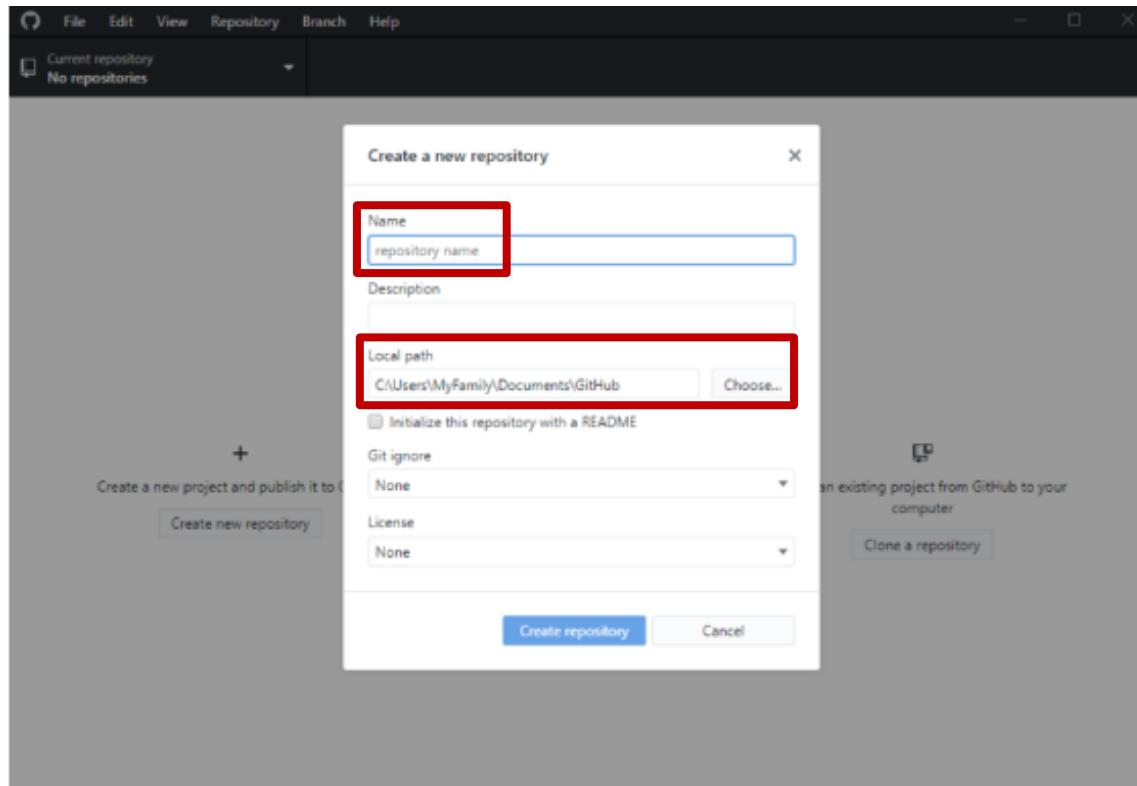
## 2. GitHub Desktop 계정 등록



- 첫 설치 시 로그인이 안되어 있기 때문에, file → Options → GitHub계정 등록을 통하여 로그인 하도록 함

# 4.1 GitHub Desktop 설치 및 계정등록

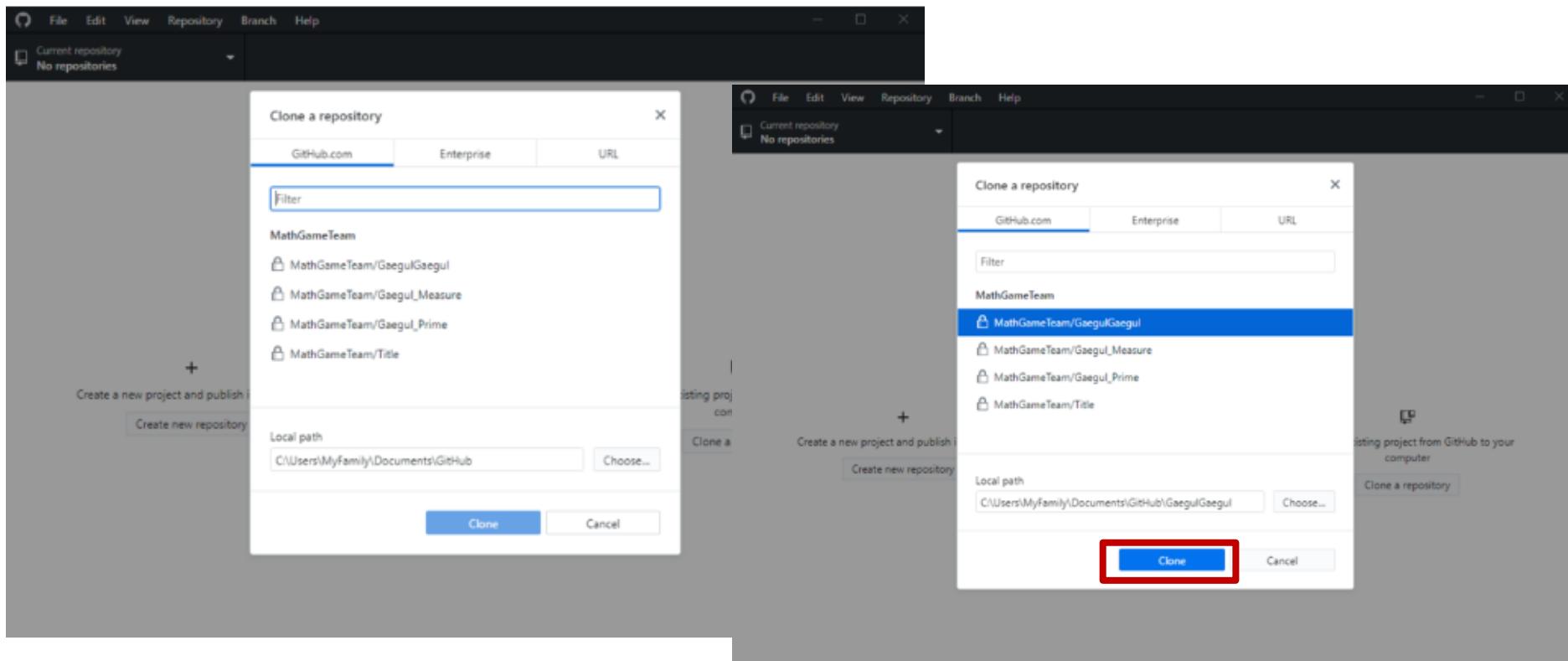
## 3. GitHub Desktop 저장소 생성



- 새로운 저장소(Repository)를 GitHub desktop에서도 생성할 수 있음
- 저장소 이름과 사용자 컴퓨터 내 저장소 위치 설정 후 생성

# 4.1 GitHub Desktop 설치 및 인증

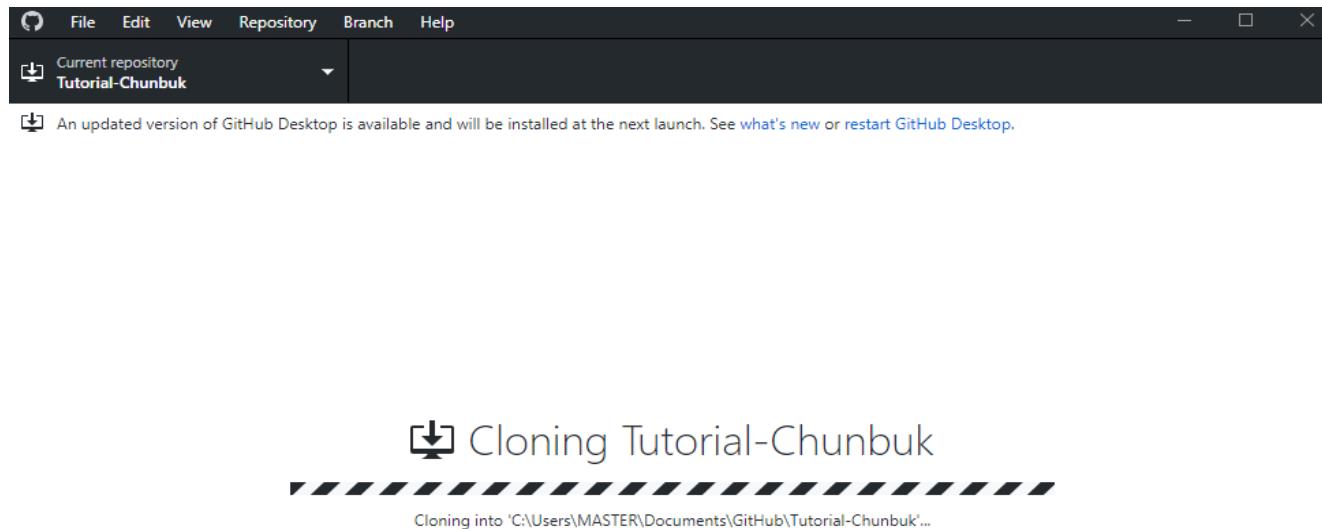
## 4. GitHub Desktop Clone



- GitHub desktop에서 Clone을 통하여 웹에서 사용하는 작업을 불러 올 수 있음

# 4.2 GitHub Desktop 사용

## 1. Clone하여 가져옴



# 4.2 GitHub Desktop 사용

## 2. GitHub Desktop History

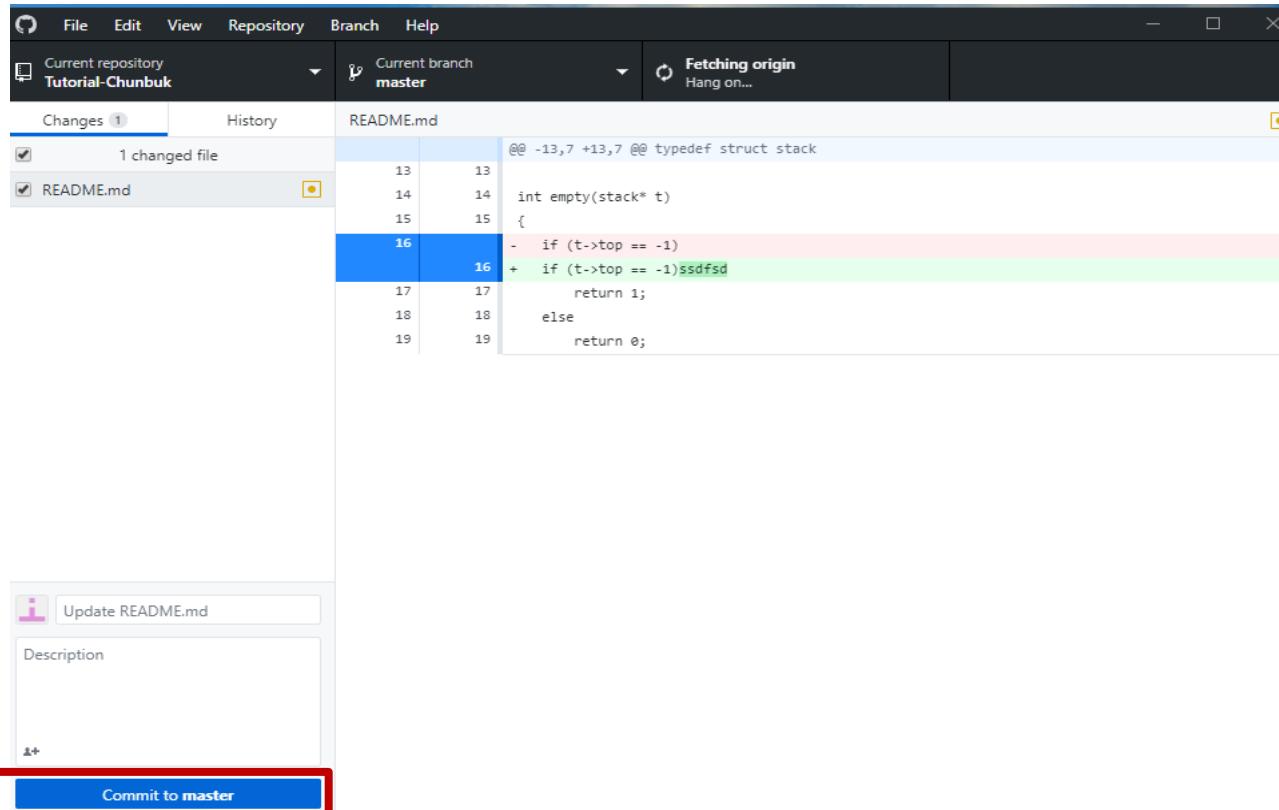
The screenshot shows the GitHub Desktop application interface. The top bar includes File, Edit, View, Repository, Branch, and Help menus. The status bar indicates the current repository is 'Tutorial-Chunbuk', the current branch is 'master', and the last fetch was 'just now'. A message at the bottom left通知有一个更新版本的GitHub Desktop可用。The main window has two tabs: 'Changes' and 'History'. The 'History' tab is active, displaying a list of commits. The first commit is 'Update README.md' by 'HAMDONGHO' (committed 4 minutes ago). The second commit is 'Initial commit' by 'HAMDONGHO' (committed 13 minutes ago). The detailed view for the 'Update README.md' commit shows the file 'README.md' with the following diff:

```
@@ -1,2 +1,212 @@
 1 # Tutorial-Chunbuk
 2 for the Tutorial
 3 +
 4 +#include <stdio.h>
 5 +#include <stdlib.h>
 6 +#include <string.h>
 7 +
 8 +typedef struct stack
 9 +{
10 +    int array[100];
11 +    int top;
12 +}stack;
13 +
14 +int empty(stack* t)
15 +{
16 +    if (t->top == -1)
17 +        return 1;
18 +    else
19 +        return 0;
20 +}
21 +
22 +void Top(stack* t)
23 +{
24 +    t->top = -1;
```

- Clone을 하게 되면, History로 들어가 제대로 Clone이 되었는지 확인할 수 있음

# 4.2 GitHub Desktop 사용

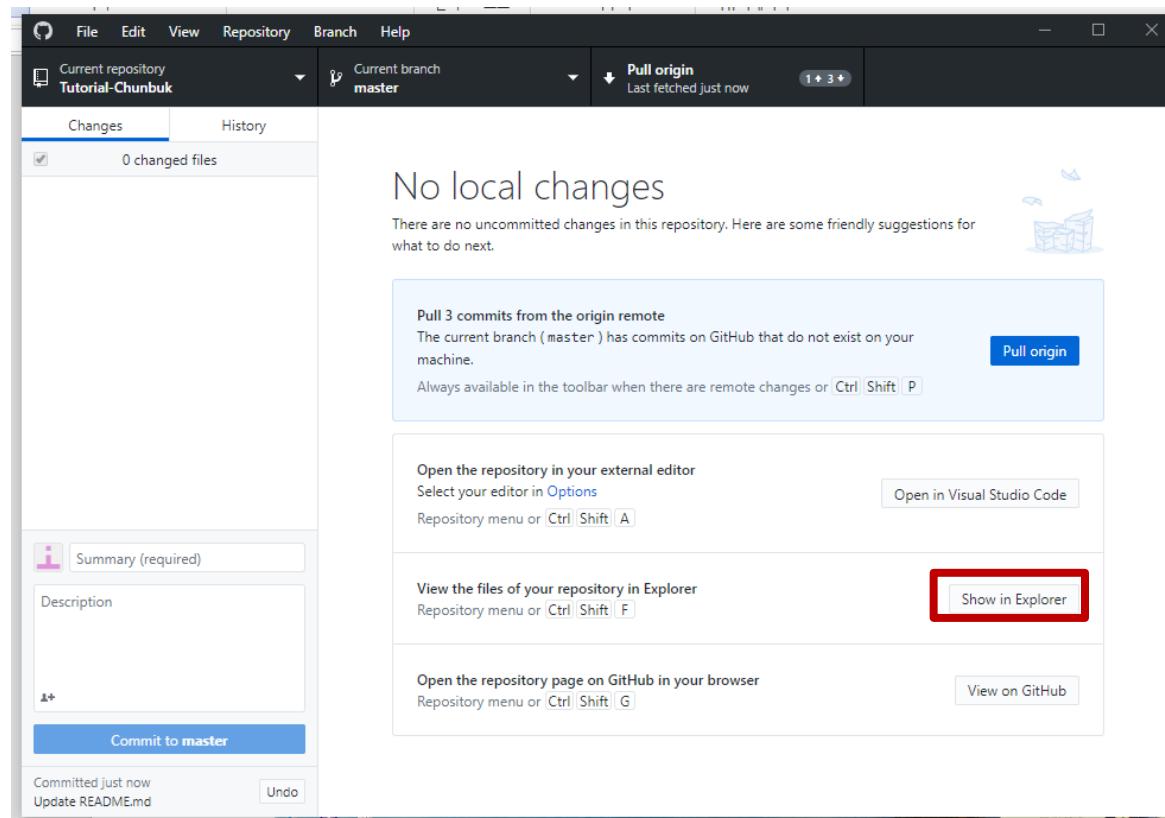
## 3. GitHub Desktop Commit



- Web에서 수정이 있을 경우 아래와 같이 Change가 있다고 알림 해주고, Commit to Master를 클릭하여 Merge하여 Desktop에도 적용 가능

# 4.2 GitHub Desktop 사용

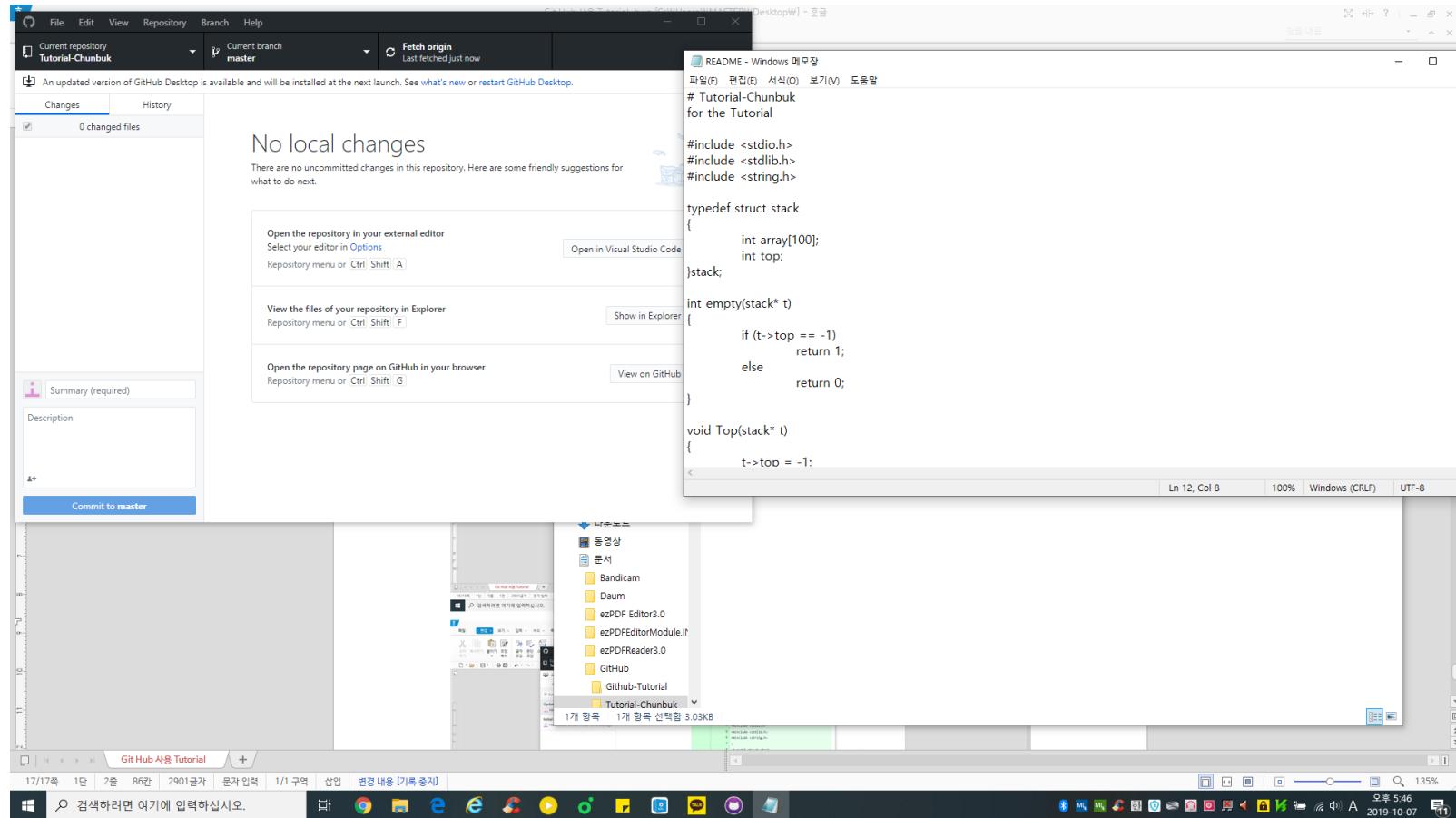
## 4. GitHub Desktop에서의 작업



- Clone한 Desktop Repository에서 편집을 하고 싶을 경우, 첫 화면의 Show in Explorer를 클릭

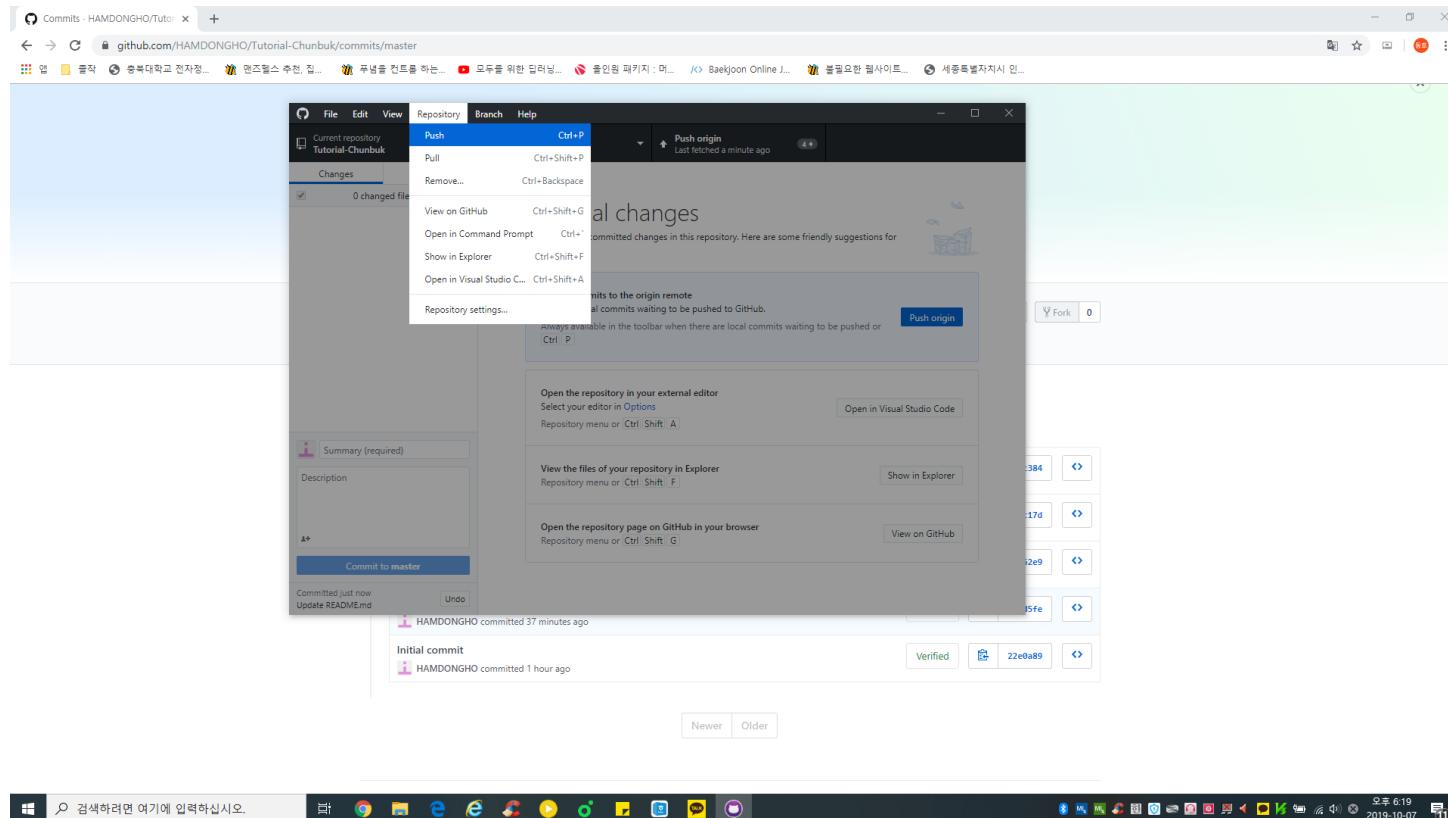
# 4.2 GitHub Desktop 사용

## 4. GitHub Desktop에서의 작업



# 4.2 GitHub Desktop 사용

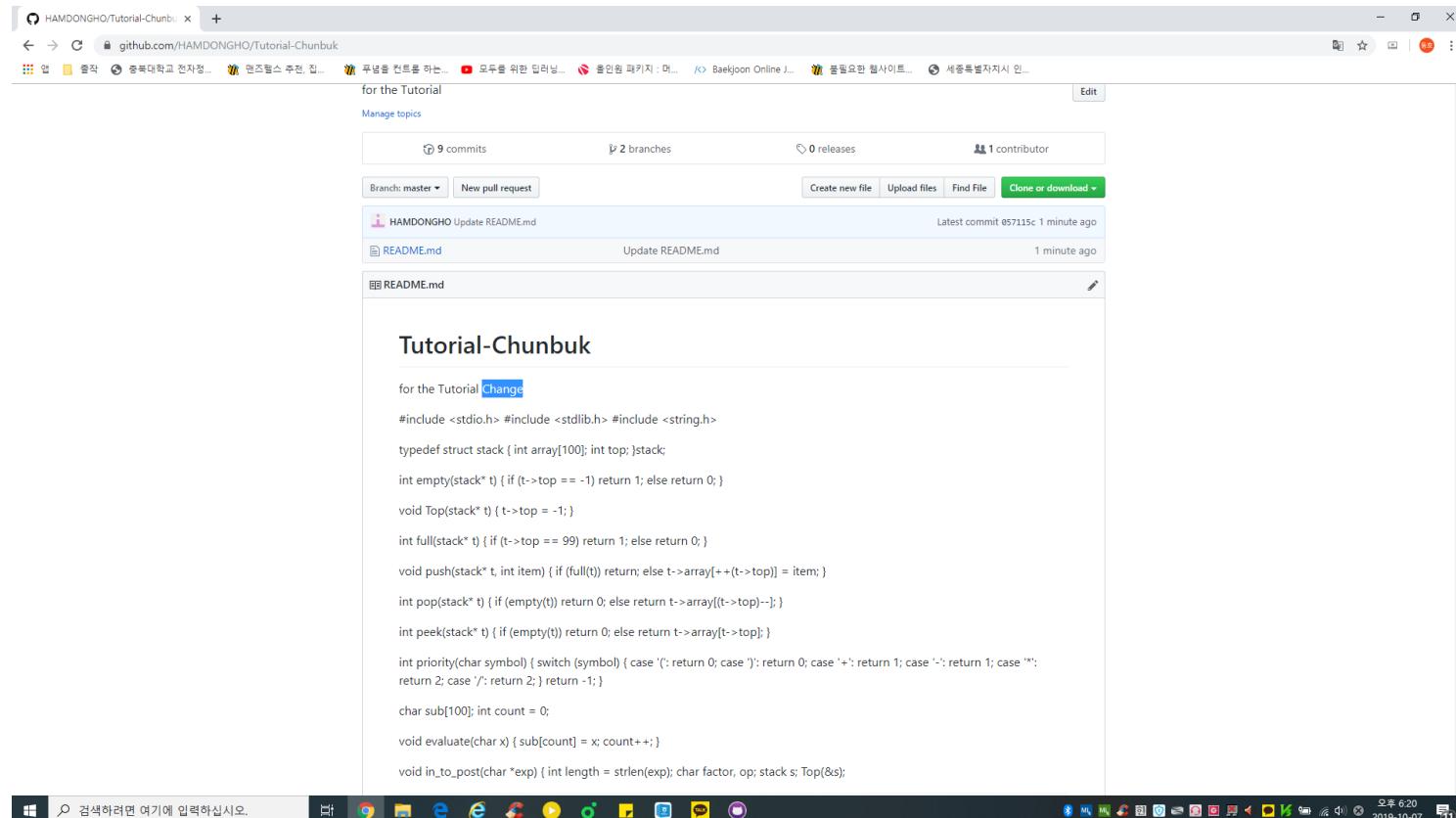
## 5. GitHub Desktop Push



- 수정 후, 저장하고 Push를 통하여 원격저장소(GitHub)에 업로드 할 수 있습니다.

# 4.2 GitHub Desktop 사용

## 6. Web에서 확인



- GitHub Desktop에서 변경하고 Push를 통해, Web에도 적용된 모습

# 충북대학교 소프트웨어중심사업단 오픈소스SW센터

