

합 업 도 구

깃&깃허브



깃

깃이란?

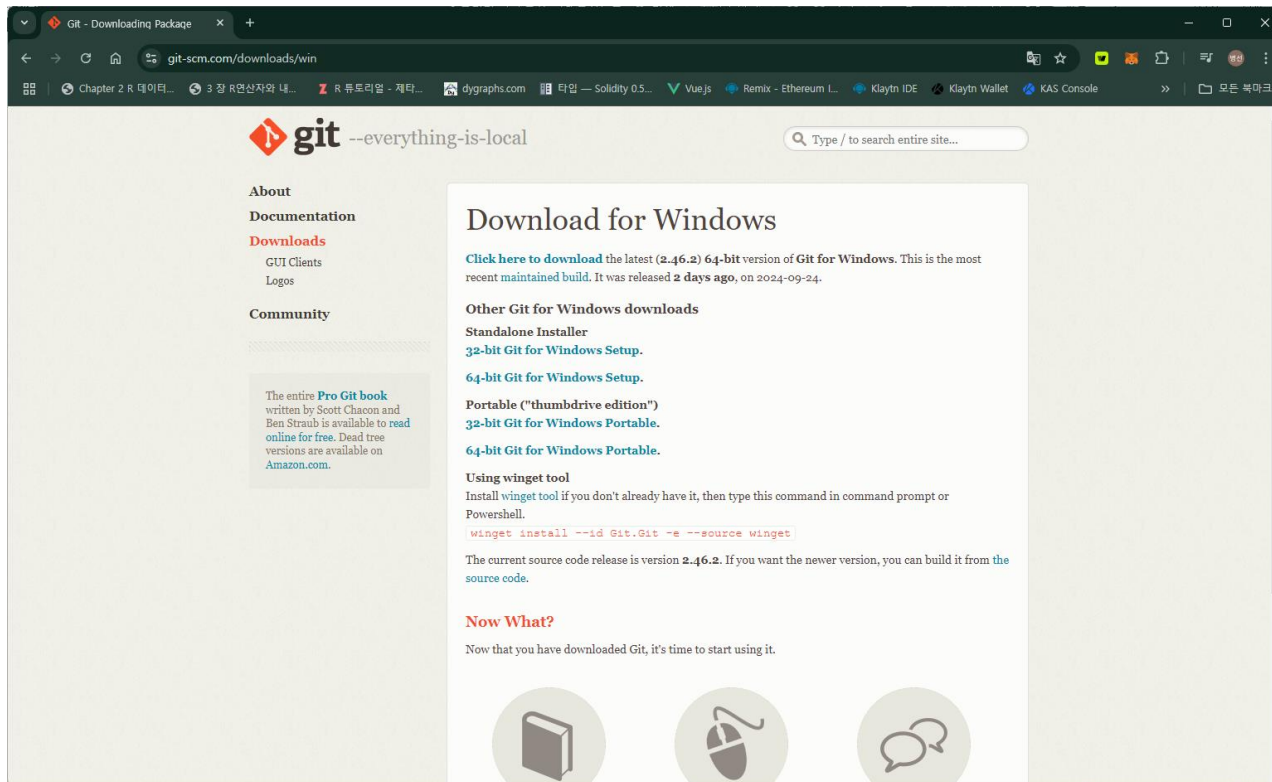
Git은 소스 코드 버전 관리 시스템으로, 개발자가 소스 코드를 효율적으로 관리하고 협업할 수 있도록 도와주는 도구입니다. Git은 파일의 변경 사항을 추적하고, 여러 명의 개발자가 동시에 같은 프로젝트에서 작업할 때 충돌 없이 작업을 진행할 수 있게 해줍니다.

깃의 주요 기능

1. 버전 관리
 - 문서를 수정할 때마다 언제 수정했는지 어떤 것을 변경했는지 등을 구체적으로 기록하는 기능
2. 백업
 - 현재 컴퓨터에 있는 자료들을 다른 컴퓨터에 복제하는 기능
3. 협업
 - 팀원들끼리 파일을 편하게 주고받으면서 일을 할 수 있는 기능

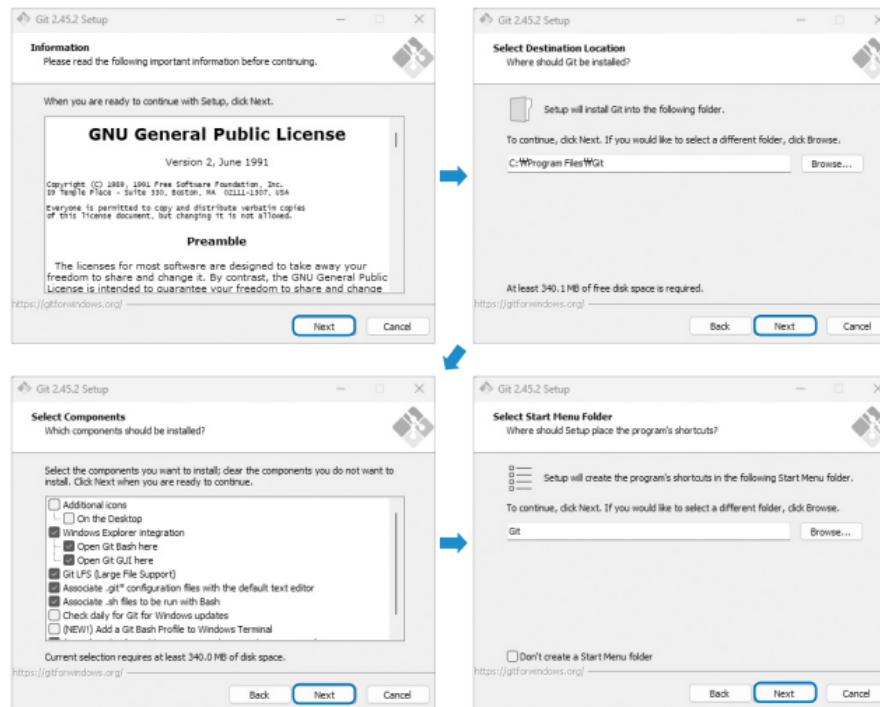
깃 설치(Windows)

- <https://git-scm.com/downloads/win> 해당 사이트에서 깃을 다운로드 후 실행



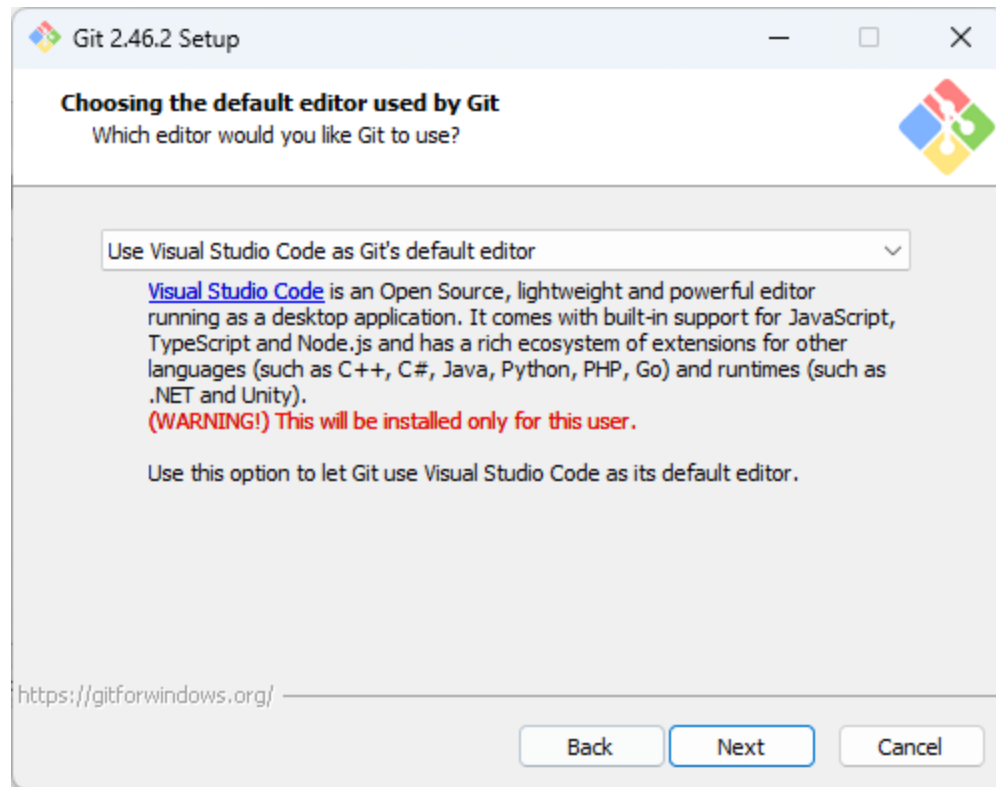
깃 설치(Windows)

- 파일을 실행 시킨 후 기본적인 설정은 Next를 눌러준다.



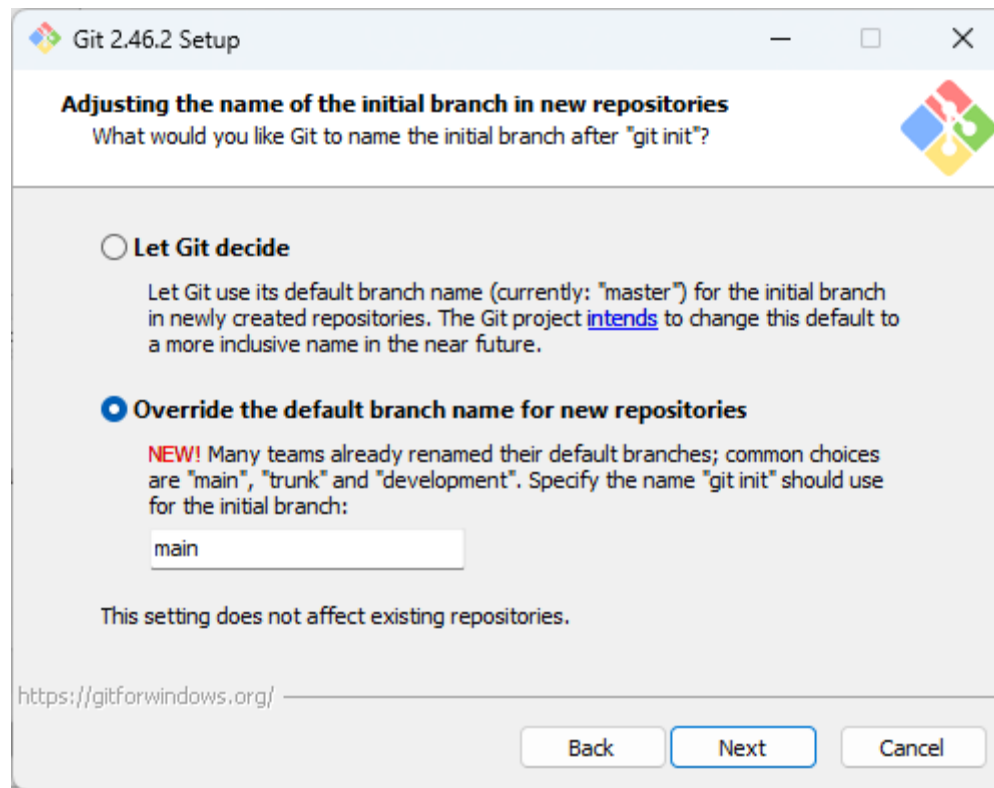
깃 설치(Windows)

- 깃에서 사용할 편집기는 Vscode로 설정을 변경한다.



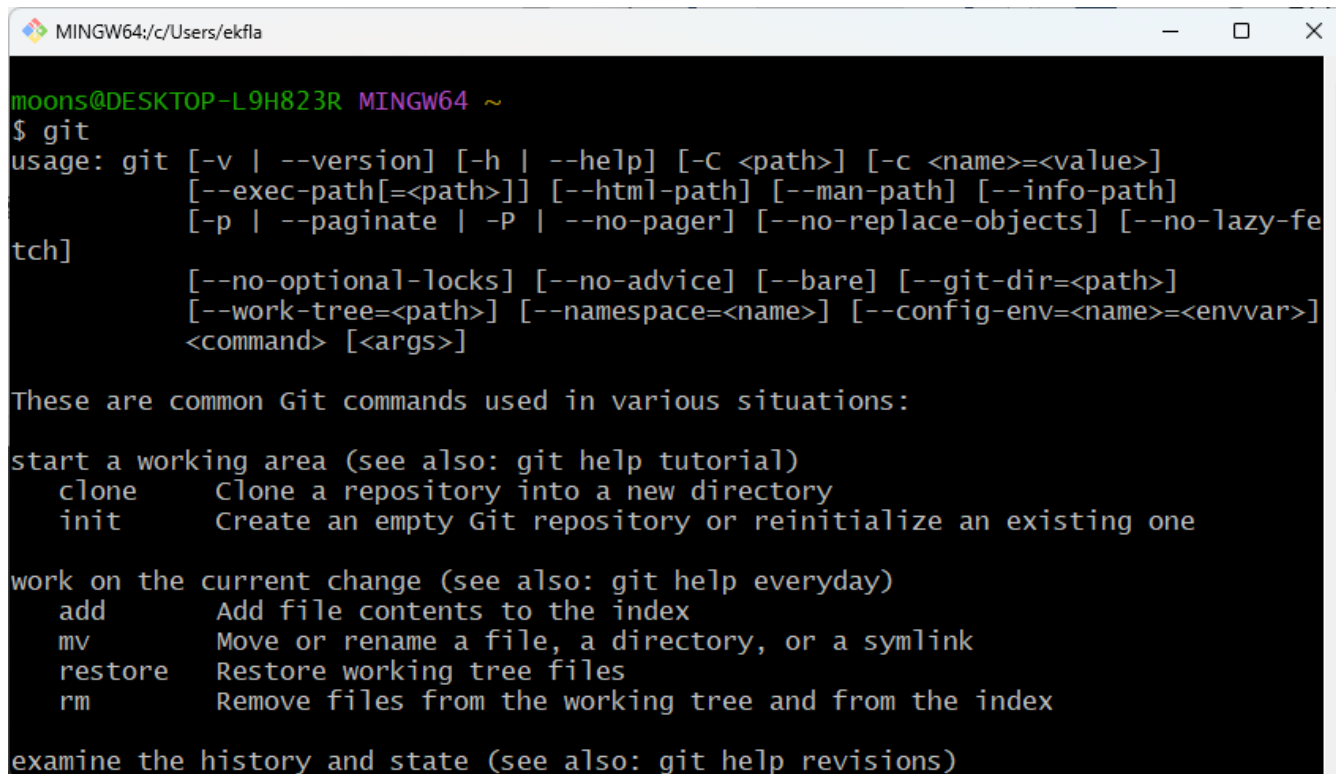
깃 설치(Windows)

- 옵션 중 두번째 옵션을 선택한다. (브랜치의 이름은 그대로 유지)



깃 설치(Windows)

- 윈도우 작업표시줄에서 'git'이라고 입력하면 'git bash'를 선택한다.
- 실행된 창에서 'git'이라는 명령어를 입력 후 ENTER키를 누른다.



```
moons@DESKTOP-L9H823R MINGW64 ~
$ git
usage: git [-v | --version] [-h | --help] [-C <path>] [-c <name>=<value>]
          [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
          [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--no-lazy-fetch]
          [--no-optional-locks] [--no-advice] [--bare] [--git-dir=<path>]
          [--work-tree=<path>] [--namespace=<name>] [--config-env=<name>=<envvar>]
          <command> [<args>]

These are common Git commands used in various situations:

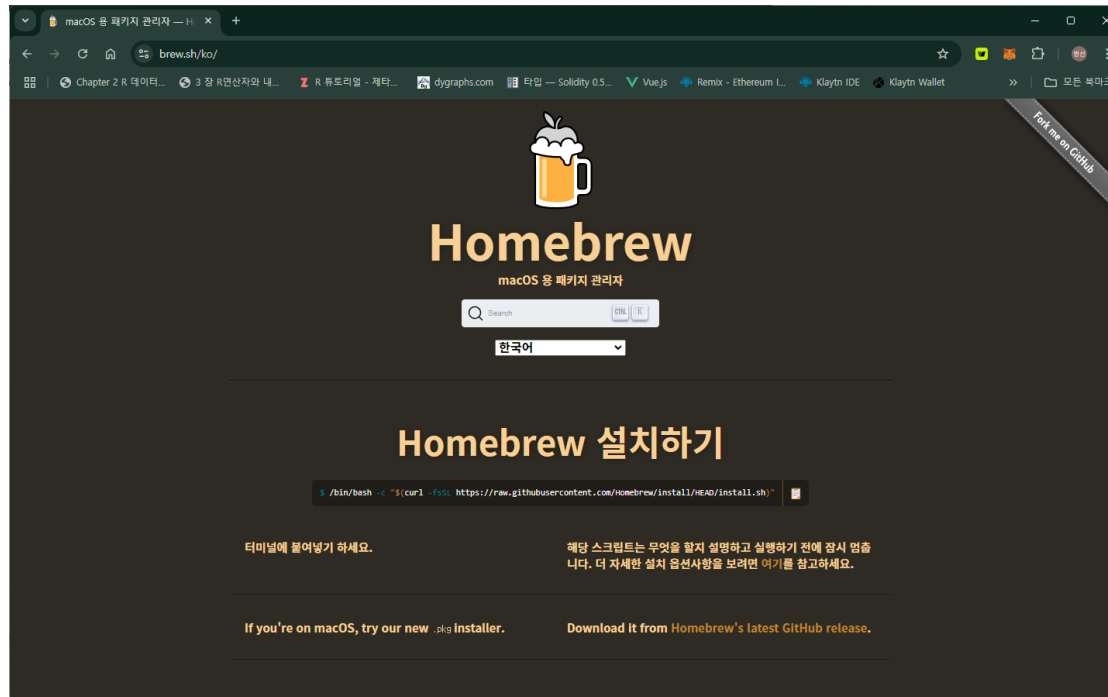
start a working area (see also: git help tutorial)
  clone      Clone a repository into a new directory
  init       Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
  add        Add file contents to the index
  mv         Move or rename a file, a directory, or a symlink
  restore    Restore working tree files
  rm         Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)
```

깃 설치(MAC)

- MAC 맥에서 깃을 설치하기 위해서는 홈브류(<https://brew.sh/ko/>) 설치가 필요
- 사이트에 접속하여 터미널을 이용하여 설치



깃 설치(MAC)

- brew 명령어인 install을 이용하여 git을 설치

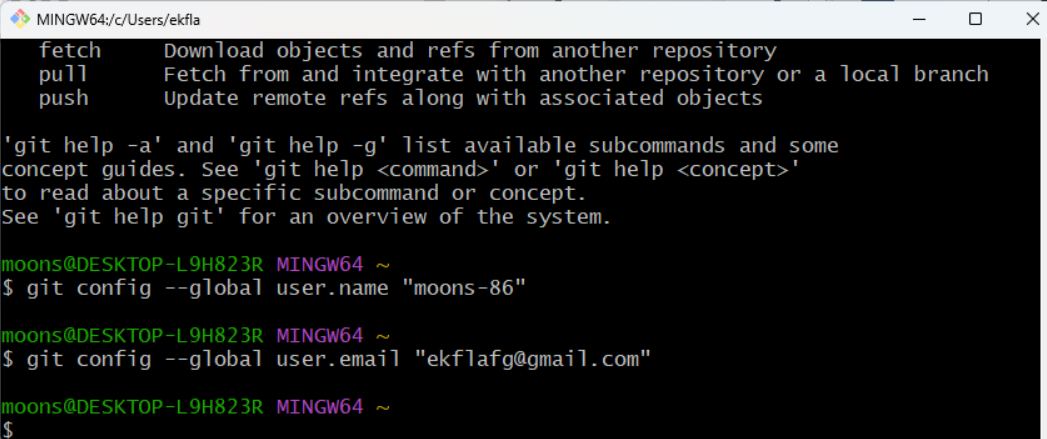
```
munbyeongseon@munbyeongseon-ui-MacBookAir Desktop % brew install git
==> Auto-updating Homebrew...
Adjust how often this is run with HOMEBREW_AUTO_UPDATE_SECS or disable with
HOMEBREW_NO_AUTO_UPDATE. Hide these hints with HOMEBREW_NO_ENV_HINTS (see `man b
rew`).
==> Downloading https://ghcr.io/v2/homebrew/portable-ruby/portable-ruby/blobs/sh
a256:e7340e4a1d7cc0f113686e461b93114270848cb14676e9037a1a2ff3b1a0ff32
##### 100.0%
==> Pouring portable-ruby-3.3.5.arm64_big_sur.bottle.tar.gz
==> Auto-updated Homebrew!
Updated 2 taps (homebrew/core and homebrew/cask).
==> New Formulae
aicommit      flang          maeparser      probe-rs-tools  wthrr
binsider      inchi          mbpoll         repopack
coordgen      keep-sorted   polkit         roxctl
facad         lld           postgresql@17  rsgain
==> New Casks
ccstudio              localsend              synology-image-assistant
excalidrawz           mininstaller          vienna-assistant
fujifilm-tether-app   rouvy                 windows-app
label-live            silhouette-studio
==> Downloading https://ghcr.io/v2/homebrew/core/git/manifests/2.46.2
```

깃 환경 설정

- 윈도우라면 git bash를 MAC이라면 터미널 창을 열어준다.

```
git config --global user.name "이름"  
git config --global user.email "메일주소"
```

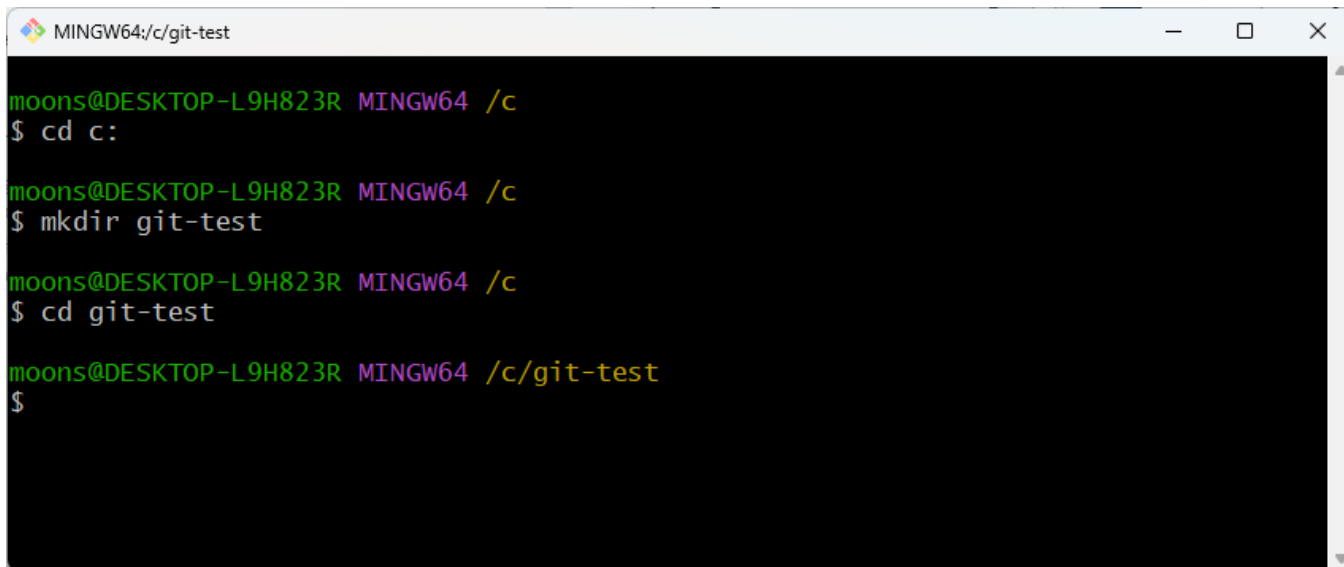
- --global 옵션은 현재 컴퓨터에 있는 모든 저장소에서 같은 사용자 정보를 사용하도록 설정



```
MINGW64/c/Users/ekfla  
fetch      Download objects and refs from another repository  
pull       Fetch from and integrate with another repository or a local branch  
push       Update remote refs along with associated objects  
  
'git help -a' and 'git help -g' list available subcommands and some  
concept guides. See 'git help <command>' or 'git help <concept>'  
to read about a specific subcommand or concept.  
See 'git help git' for an overview of the system.  
  
moons@DESKTOP-L9H823R MINGW64 ~  
$ git config --global user.name "moons-86"  
  
moons@DESKTOP-L9H823R MINGW64 ~  
$ git config --global user.email "ekflafg@gmail.com"  
  
moons@DESKTOP-L9H823R MINGW64 ~  
$
```

깃 저장소 만들기

- 깃 저장소는 컴퓨터 저장소 어디서든 생성이 가능
- C 드라이브에 간단한 저장소를 생성
 - 터미널에서 C 드라이브로 이동 (`cd c:`)
 - `git-test` 폴더를 생성 (`mkdir git-test`)
 - 생성한 폴더로 이동 (`cd git-test`)



```
MINGW64:/c/git-test
moons@DESKTOP-L9H823R MINGW64 /c
$ cd c:

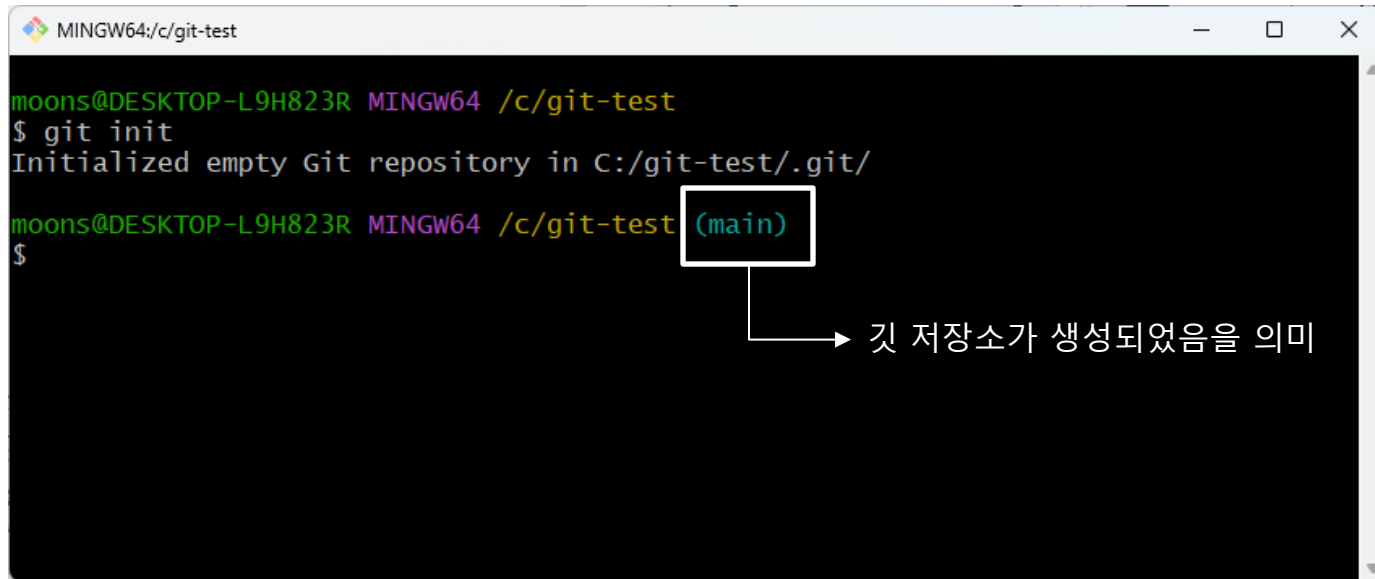
moons@DESKTOP-L9H823R MINGW64 /c
$ mkdir git-test

moons@DESKTOP-L9H823R MINGW64 /c
$ cd git-test

moons@DESKTOP-L9H823R MINGW64 /c/git-test
$
```

깃 저장소 만들기

- 해당 폴더를 깃 저장소로 초기화 (git init)
 - init 명령어는 initialize'의 줄임말



```
MINGW64:/c/git-test
moons@DESKTOP-L9H823R MINGW64 /c/git-test
$ git init
Initialized empty Git repository in C:/git-test/.git/
moons@DESKTOP-L9H823R MINGW64 /c/git-test (main)
$
```

→ 깃 저장소가 생성되었음을 의미

깃 버전 관리

- 깃에서 가장 기본적인 기능으로 파일에 대한 버전들을 관리하는 기능이다.
- 깃에서는 버전을 관리하면서 원래 파일 이름은 그대로 유지하면서 파일에서 무엇을 변경했는지를 변경 시점마다 저장할 수 있다.
- 버전마다 작업한 내용을 확인할 수 있고 그 버전으로 되돌리기가 가능하다.

깃 작업 영역

- 작업 트리는 생성한 git-test 디렉토리 영역
- 스테이지와 저장소는 init 명령어를 이용하여 생성된 .git 디렉토리 영역

작업 트리

현재 작업중인
디렉토리
파일의 수정, 저장 등의 작업을
하는 공간

스테이지

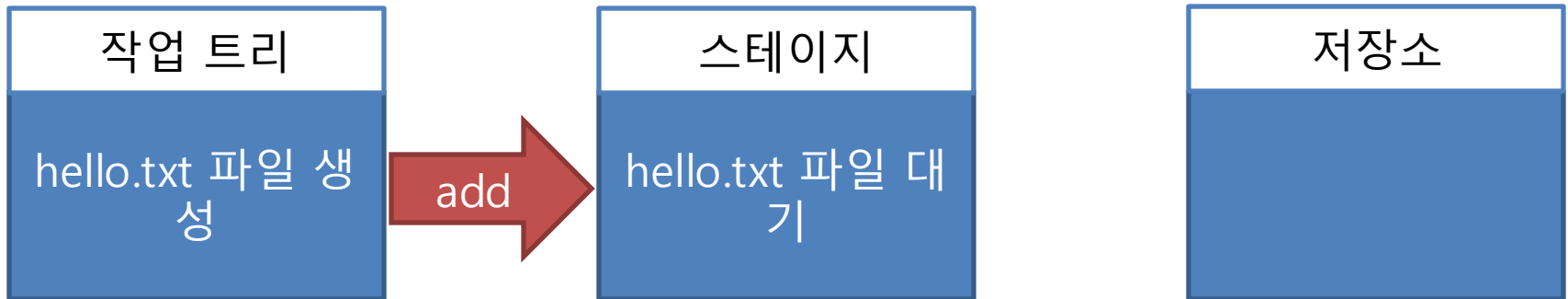
버전으로 만들
파일이 대기하는
장소

저장소

스테이지에 대기
하던 파일들을
버전으로 만들어
저장하는 장소

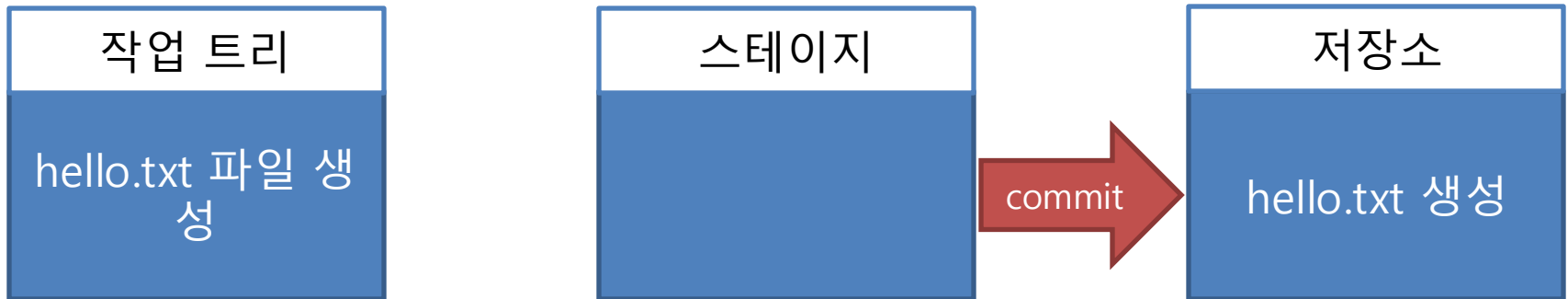
add와 commit 이해

- 스테이지와 저장소는 숨김 폴더로 지정되어 있다.
- 스테이지에 버전으로 사용할 파일을 작업 트리에서 생성하고 수정 후 저장
- 해당 파일을 스테이지로 이동 시키기 위해서는 add 라는 명령어를 사용



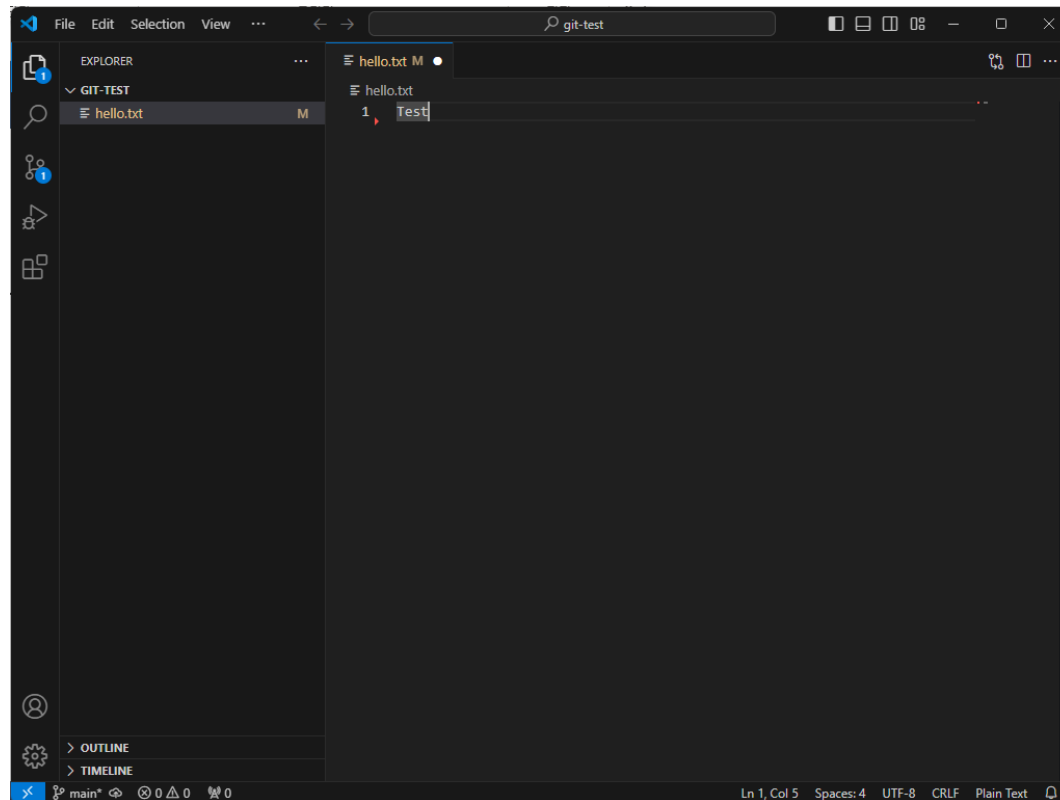
add와 commit 이해

- 스테이지에 있는 대기 파일들을 버전으로 만드는 명령어는 commit 이라는 명령어를 이용
- 스테이지에 대기하던 파일은 사라지고 저장소에 버전이 생성



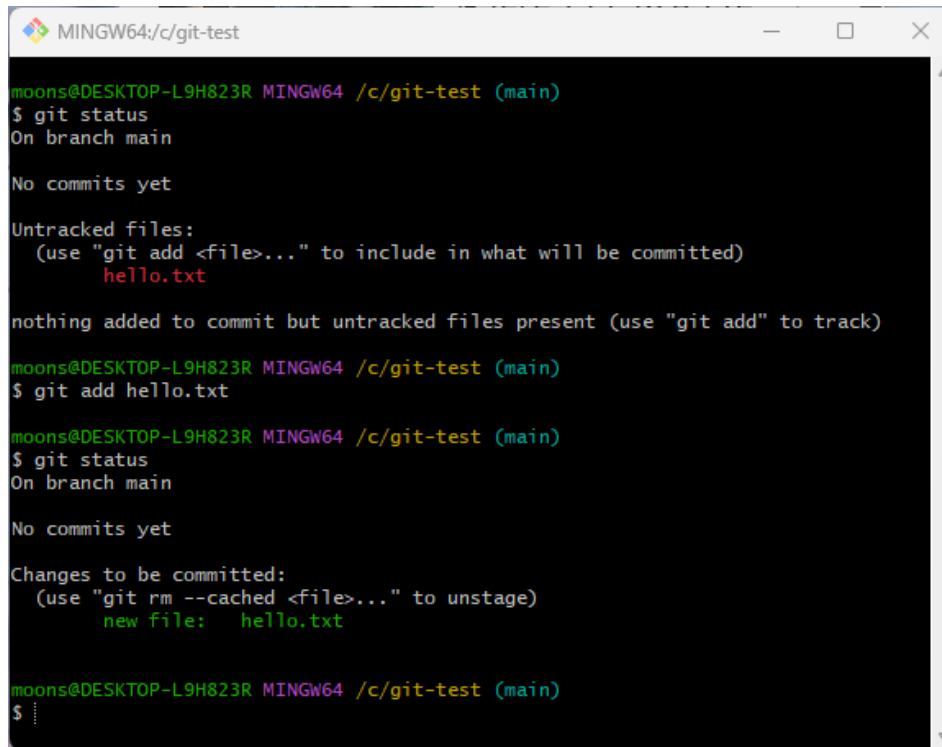
add와 commit 이해

- Vscode를 이용하여 작업 트리에 hello.txt 파일을 생성하고 Test라는 문구를 입력하고 저장



add와 commit 이해

- git status 는 현재 깃의 상태를 출력한다.
- git add <파일이름> 파일을 스테이지에 등록한다.
- git status 를 다시 확인해보면 스테이지에 추가되었다.



```
MINGW64:/c/git-test
moons@DESKTOP-L9H823R MINGW64 /c/git-test (main)
$ git status
On branch main

No commits yet

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

nothing added to commit but untracked files present (use "git add" to track)

moons@DESKTOP-L9H823R MINGW64 /c/git-test (main)
$ git add hello.txt

moons@DESKTOP-L9H823R MINGW64 /c/git-test (main)
$ git status
On branch main

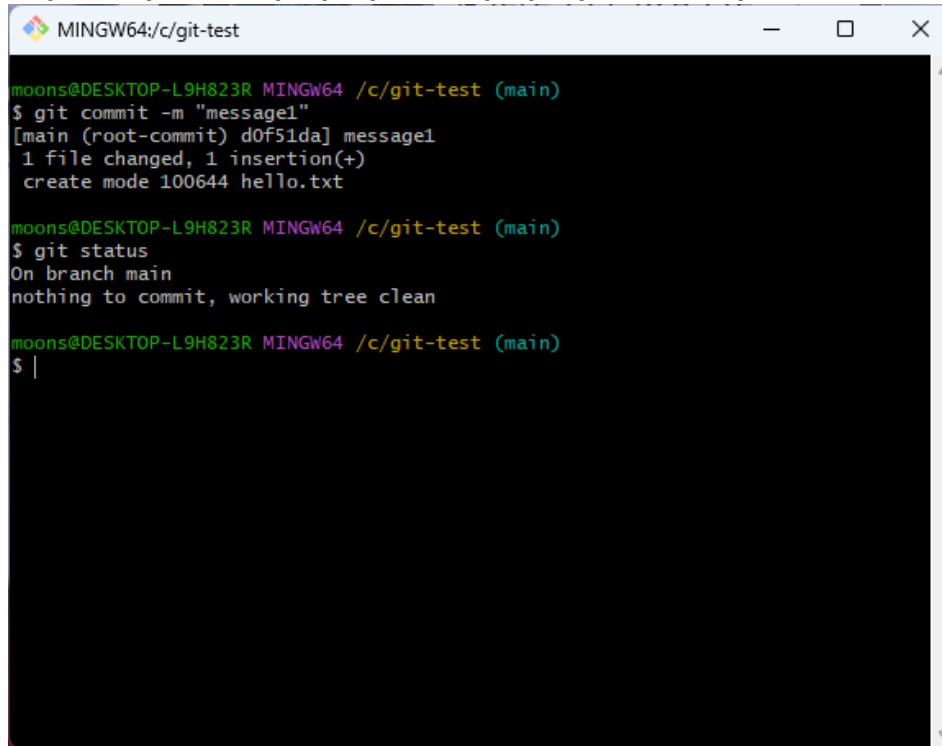
No commits yet

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

moons@DESKTOP-L9H823R MINGW64 /c/git-test (main)
$
```

add와 commit 이해

- `git commit -m "저장할 메시지명"` 을 사용하면 스테이지에서 저장소로 이동하며 버전이 생성된다.
- `git status` 를 다시 확인해보면 버전으로 만들 파일도 존재하지 않고 작업 트리도 수정 내역이 존재하지 않는다

A screenshot of a Windows terminal window titled 'MINGW64:/c/git-test'. The terminal shows a user named 'moons' at a desktop named 'DESKTOP-L9H823R' running several git commands. First, they run 'git commit -m "message1"', which outputs the commit hash 'd0f51da', the message 'message1', and details about a file change and insertion. Then, they run 'git status', which outputs 'On branch main' and 'nothing to commit, working tree clean'.

```
MINGW64:/c/git-test

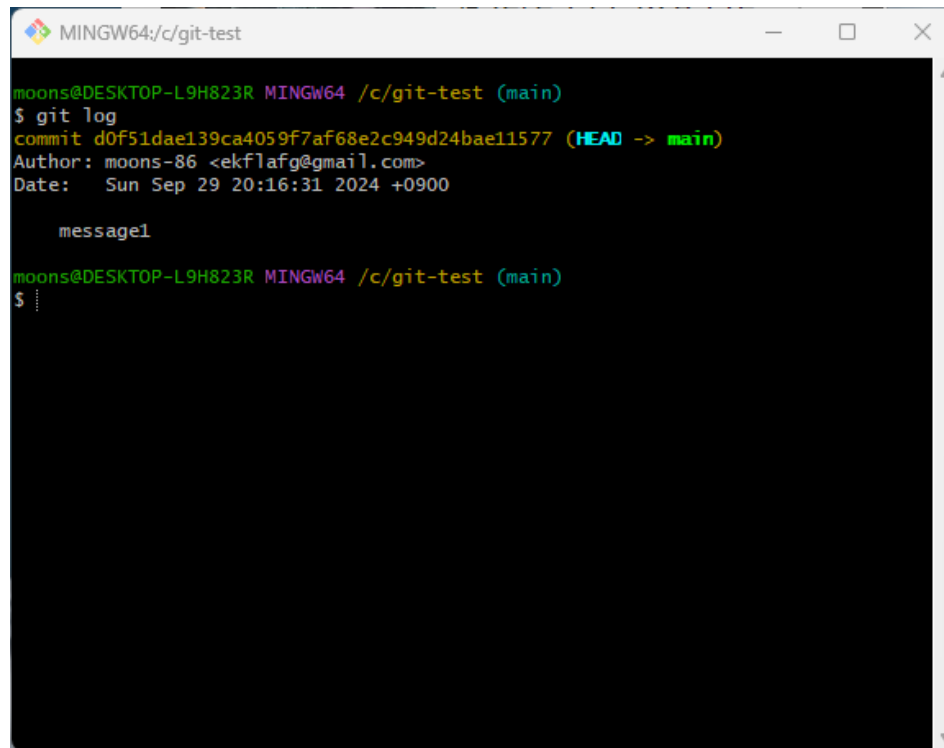
moons@DESKTOP-L9H823R MINGW64 /c/git-test (main)
$ git commit -m "message1"
[main (root-commit) d0f51da] message1
1 file changed, 1 insertion(+)
create mode 100644 hello.txt

moons@DESKTOP-L9H823R MINGW64 /c/git-test (main)
$ git status
On branch main
nothing to commit, working tree clean

moons@DESKTOP-L9H823R MINGW64 /c/git-test (main)
$ |
```

add와 commit 이해

- git log 를 이용하여 저장소의 버전을 확인한다.



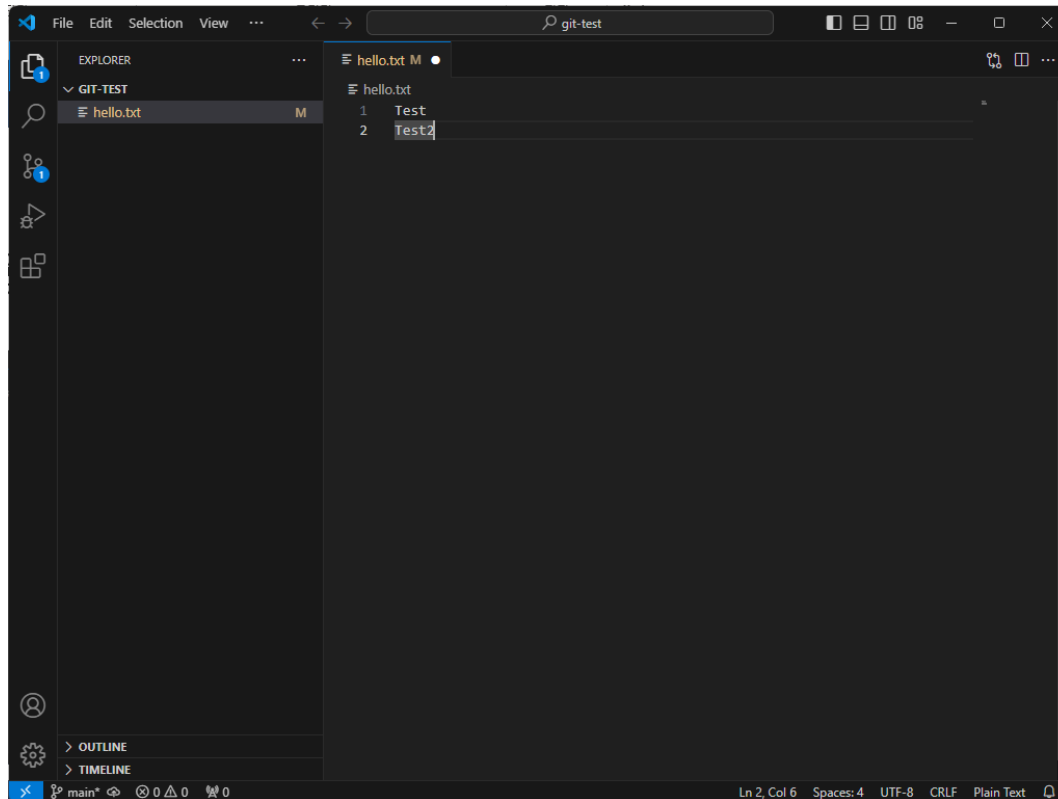
```
MINGW64:/c/git-test
moons@DESKTOP-L9H823R MINGW64 /c/git-test (main)
$ git log
commit d0f51dae139ca4059f7af68e2c949d24bae11577 (HEAD -> main)
Author: moons-86 <ekflafg@gmail.com>
Date: Sun Sep 29 20:16:31 2024 +0900

    message1

moons@DESKTOP-L9H823R MINGW64 /c/git-test (main)
$
```

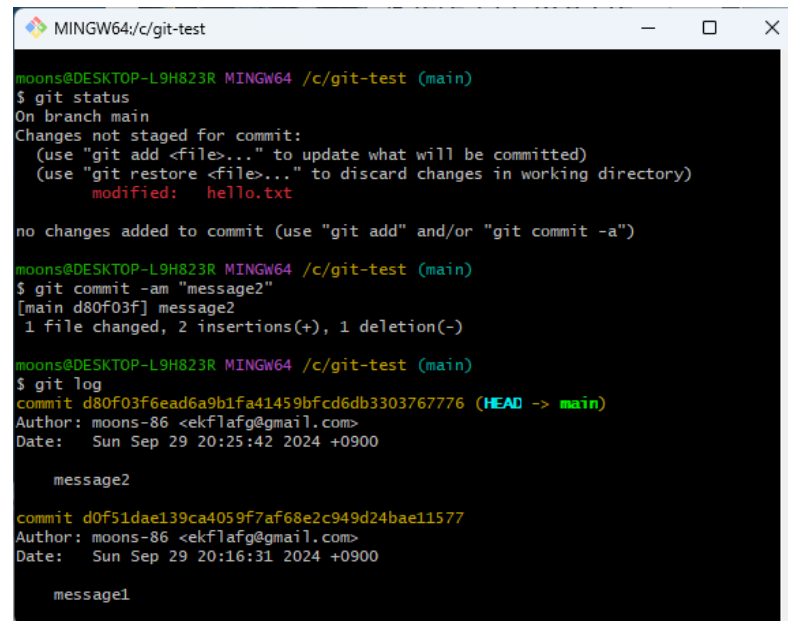
add와 commit 이해

- hello.txt 파일 내용을 수정하고 저장



add와 commit 이해

- hello.txt 파일을 수정 한 뒤 add와 commit를 한꺼번에 처리하려면 `git commit -am "메시지명"`을 사용하면 된다.
- commit을 완료한 뒤 log를 확인하면 message1과 message2 두개의 버전을 확인 할 수 있다.
(파일을 새로 생성하는 경우에는 add, commit을 사용)



```
MINGW64/c/git-test
moons@DESKTOP-L9H823R MINGW64 /c/git-test (main)
$ git status
On branch 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:   hello.txt

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

moons@DESKTOP-L9H823R MINGW64 /c/git-test (main)
$ git commit -am "message2"
[main d80f03f] message2
1 file changed, 2 insertions(+), 1 deletion(-)

moons@DESKTOP-L9H823R MINGW64 /c/git-test (main)
$ git log
commit d80f03f6ead6a9b1fa41459bFcd6db3303767776 (HEAD -> main)
Author: moons-86 <ekflafg@gmail.com>
Date:   Sun Sep 29 20:25:42 2024 +0900

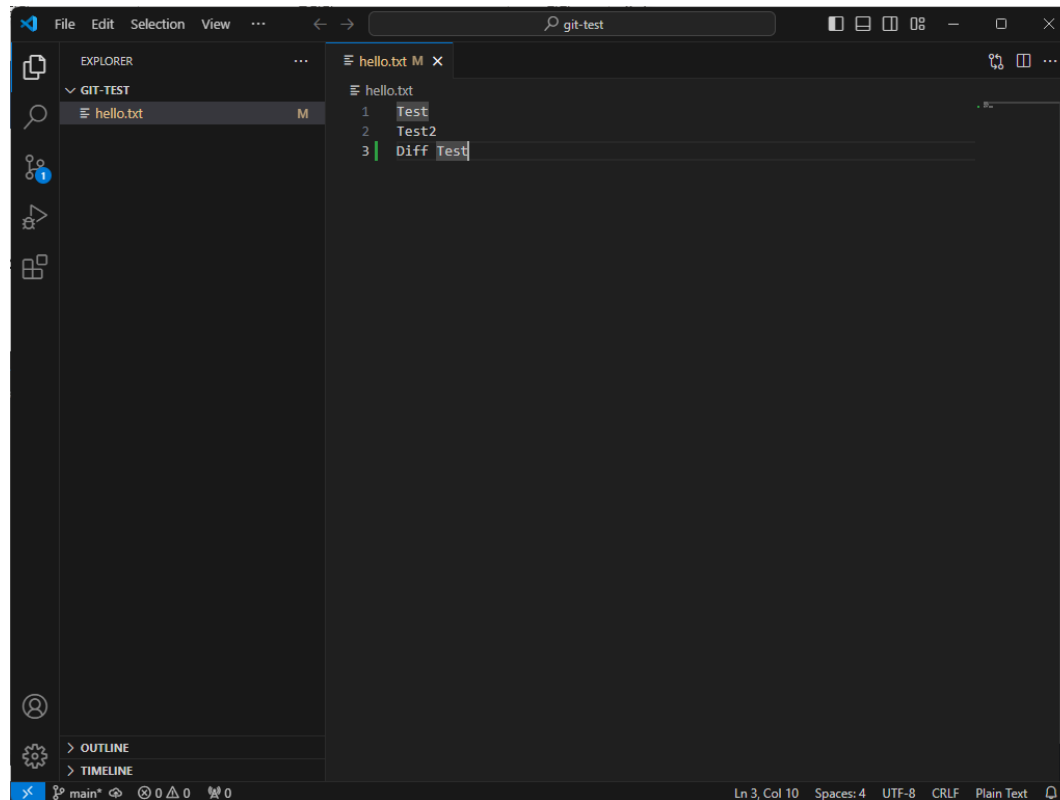
    message2

commit d0f51dae139ca4059f7af68e2c949d24bae11577
Author: moons-86 <ekflafg@gmail.com>
Date:   Sun Sep 29 20:16:31 2024 +0900

    message1
```

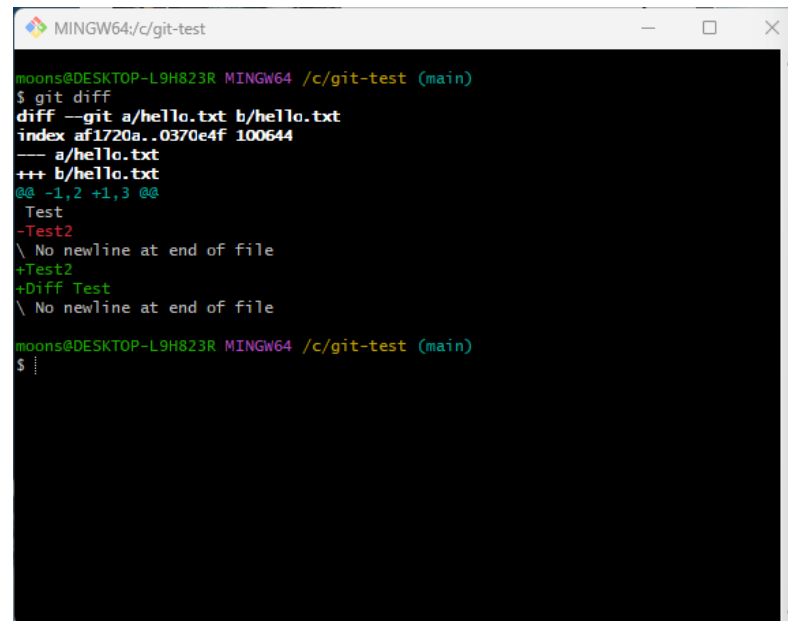

add와 commit 이해

- hello.txt 파일 내용을 수정하고 저장



변경 사항 확인하기

- hello.txt 파일을 수정 한 뒤 git diff를 사용하면 저장소에 있는 파일과 작업 트리에 있는 파일의 내용의 변경 사항을 확인 할 수 있다.

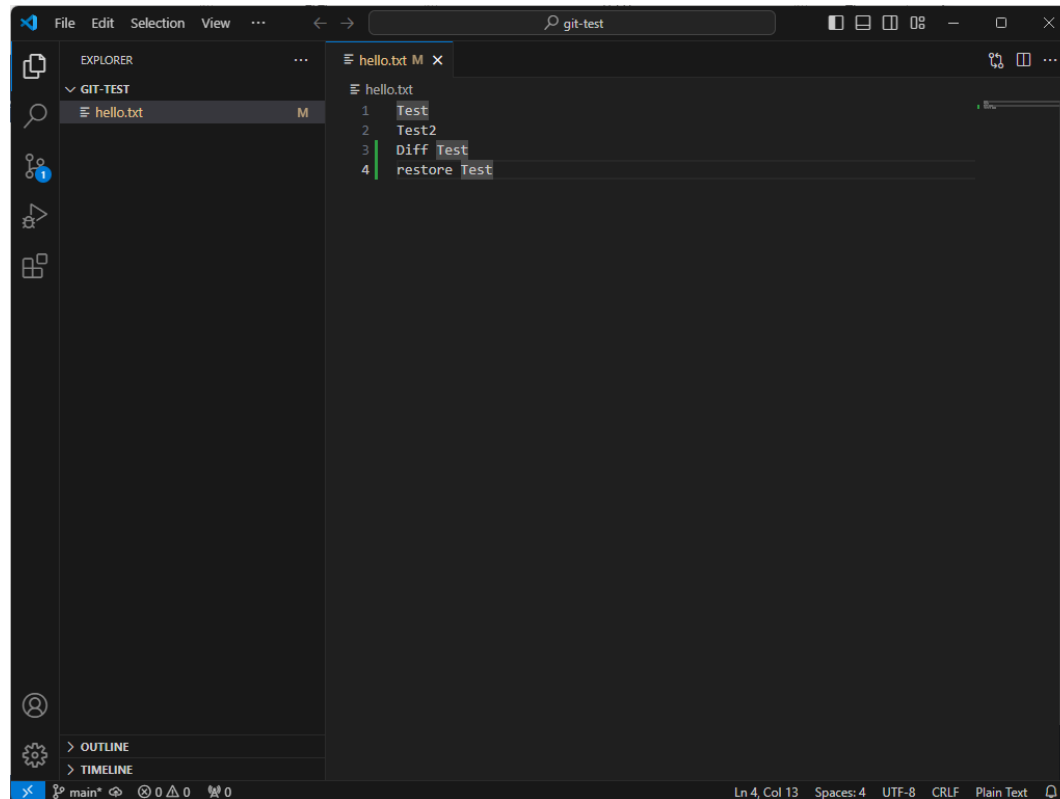


```
moons@DESKTOP-L9H823R MINGW64 /c/git-test (main)
$ git diff
diff --git a/hello.txt b/hello.txt
index af1720a..0370e4f 100644
--- a/hello.txt
+++ b/hello.txt
@@ -1,2 +1,3 @@
 Test
-Test2
\ No newline at end of file
+Test2
+Diff Test
\ No newline at end of file

moons@DESKTOP-L9H823R MINGW64 /c/git-test (main)
$
```

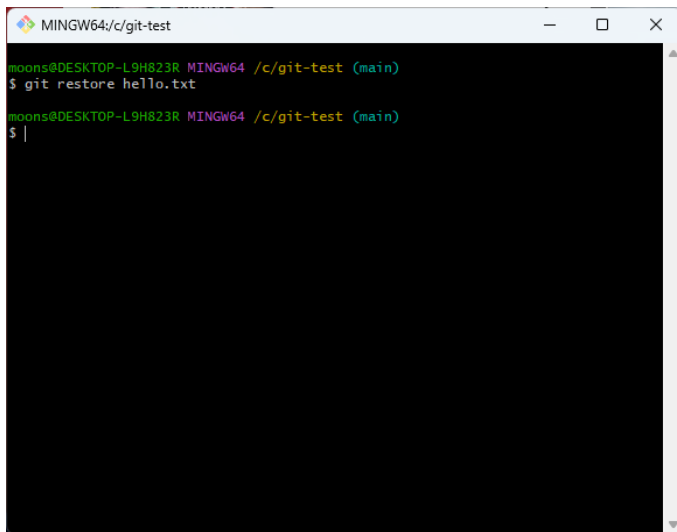
작업 되돌리기

- hello.txt 파일 내용을 수정하고 저장

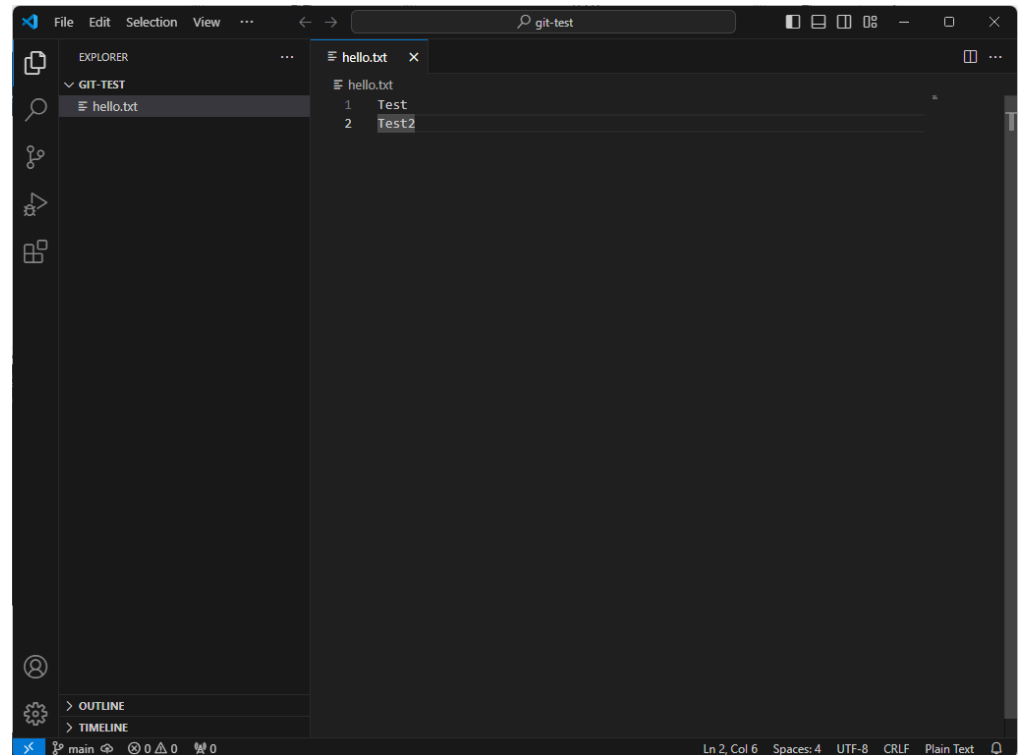


작업 되돌리기

- `git restore hello.txt` 명령어를 입력하면 수정사항들을 취소하고 저장소에 있는 파일로 되돌려준다.

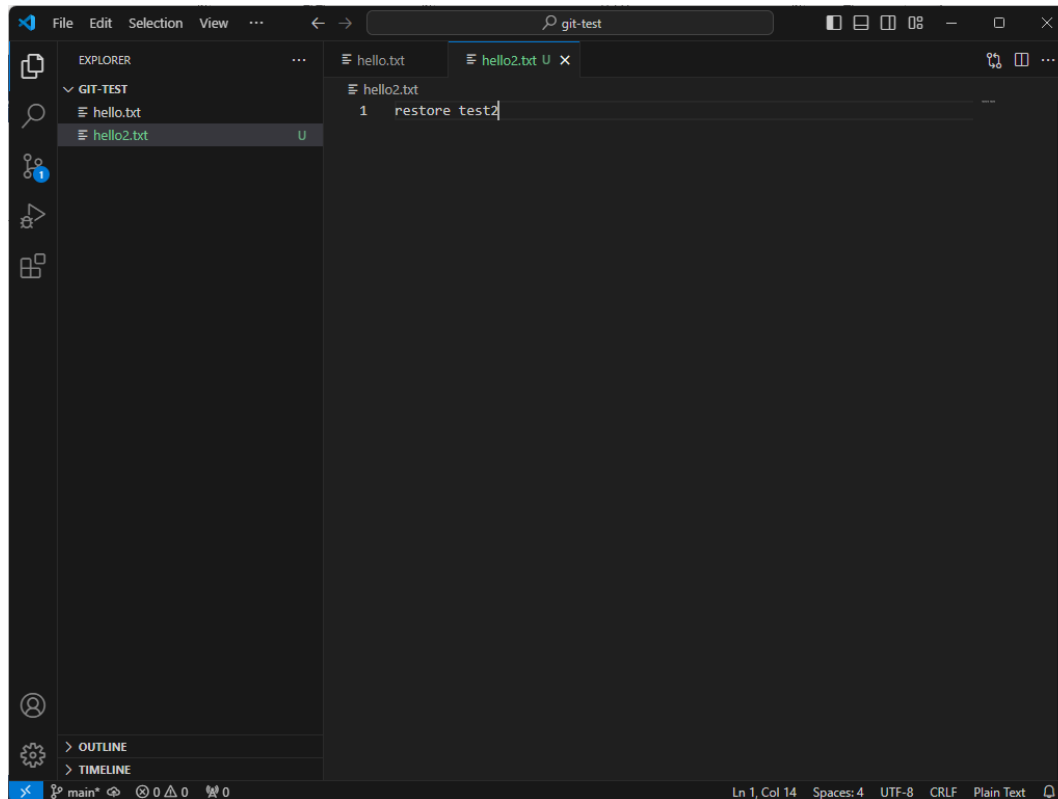


```
MINGW64/c/git-test
moons@DESKTOP-L9H823R MINGW64 /c/git-test (main)
$ git restore hello.txt
moons@DESKTOP-L9H823R MINGW64 /c/git-test (main)
$ |
```



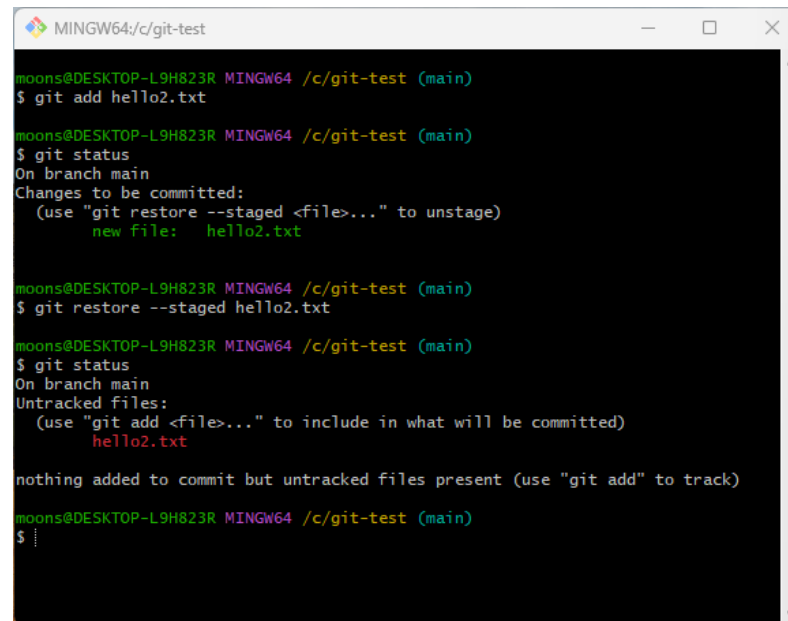
스테이지 되돌리기

- hello2.txt 파일을 생성하고 문구를 작성 뒤 저장해 준다.



스테이지 되돌리기

- 생성된 파일을 add 명령어로 스테이지에 저장
- status 를 이용하여 상태를 확인하면 스테이지에 파일이 있는 것이 확인
- restore --staged hello2.txt를 해주면 스테이지에 올라가기 전 상태로 돌아온다



```
MINGW64/c/git-test
moons@DESKTOP-L9H823R MINGW64 /c/git-test (main)
$ git add hello2.txt

moons@DESKTOP-L9H823R MINGW64 /c/git-test (main)
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   hello2.txt

moons@DESKTOP-L9H823R MINGW64 /c/git-test (main)
$ git restore --staged hello2.txt

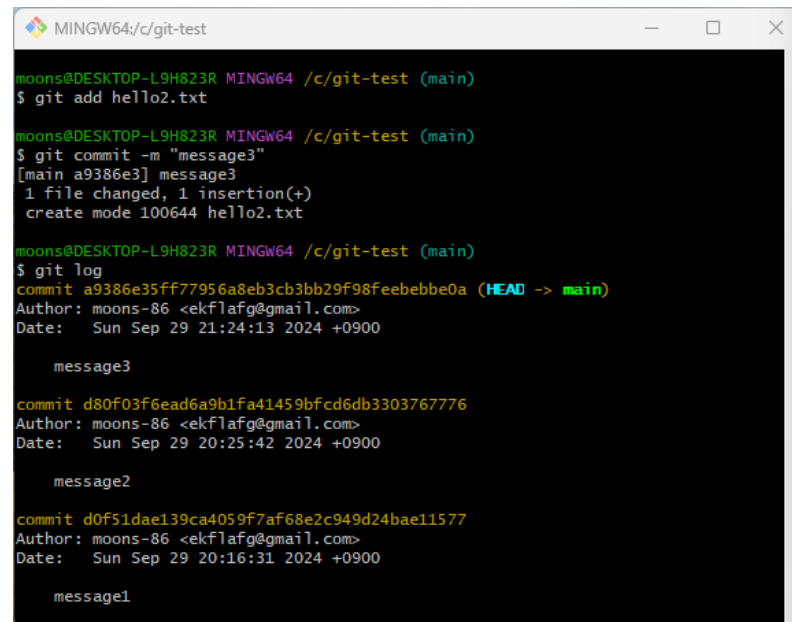
moons@DESKTOP-L9H823R MINGW64 /c/git-test (main)
$ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    hello2.txt

nothing added to commit but untracked files present (use "git add" to track)

moons@DESKTOP-L9H823R MINGW64 /c/git-test (main)
$ |
```

최근 commit 되돌리기

- 최근의 commit을 되돌리기 위해 hello2.txt를 저장소에 저장
- git log를 이용하여 현재 버전들을 확인

A terminal window titled 'MINGW64:/c/git-test' showing a sequence of git commands and their outputs. The user adds 'hello2.txt', commits with message3, and then runs 'git log' which displays three commits: message3 (HEAD -> main), message2, and message1.

```
MINGW64:/c/git-test
moons@DESKTOP-L9H823R MINGW64 /c/git-test (main)
$ git add hello2.txt

moons@DESKTOP-L9H823R MINGW64 /c/git-test (main)
$ git commit -m "message3"
[main a9386e3] message3
1 file changed, 1 insertion(+)
create mode 100644 hello2.txt

moons@DESKTOP-L9H823R MINGW64 /c/git-test (main)
$ git log
commit a9386e35ff77956a8eb3cb3bb29f98feebebb0a (HEAD -> main)
Author: moons-86 <ekflafg@gmail.com>
Date: Sun Sep 29 21:24:13 2024 +0900

    message3

commit d80f03f6ead6a9b1fa41459bfcd6db3303767776
Author: moons-86 <ekflafg@gmail.com>
Date: Sun Sep 29 20:25:42 2024 +0900

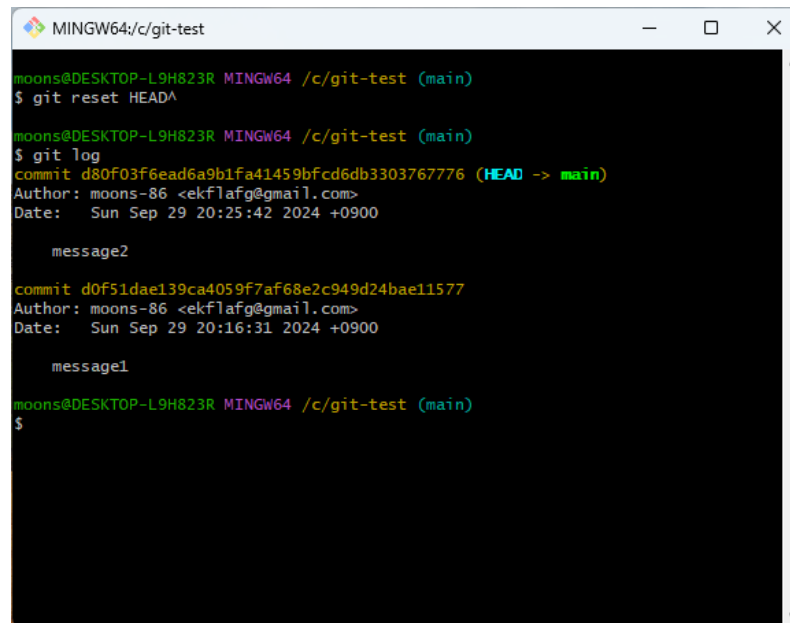
    message2

commit d0f51dae139ca4059f7af68e2c949d24bae11577
Author: moons-86 <ekflafg@gmail.com>
Date: Sun Sep 29 20:16:31 2024 +0900

    message1
```

최근 commit 되돌리기

- 최근의 commit을 되돌리기 위해서는 git에서 `reset HEAD^` 명령어를 이용
- 명령어를 이용한 뒤 log를 확인하면 message3 로그가 제거
- `hello2.txt`는 작업 트리에서만 존재하고 저장소에는 존재하지 않는다.



```
MINGW64:/c/git-test
moons@DESKTOP-L9H823R MINGW64 /c/git-test (main)
$ git reset HEAD^

moons@DESKTOP-L9H823R MINGW64 /c/git-test (main)
$ git log
commit d80f03f6ead6a9b1fa41459bfcd6db3303767776 (HEAD -> main)
Author: moons-86 <ekflafg@gmail.com>
Date:   Sun Sep 29 20:25:42 2024 +0900

    message2

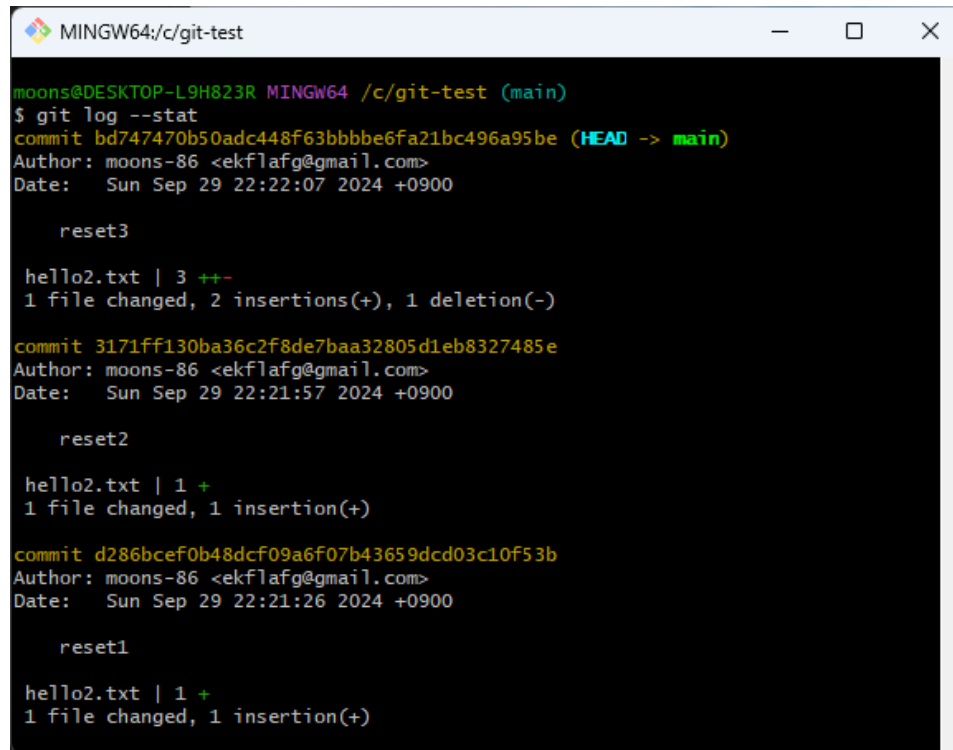
commit d0f51dae139ca4059f7af68e2c949d24bae11577
Author: moons-86 <ekflafg@gmail.com>
Date:   Sun Sep 29 20:16:31 2024 +0900

    message1

moons@DESKTOP-L9H823R MINGW64 /c/git-test (main)
$
```


특정 commit 되돌리기

- 특정 commit으로 되돌리기 위해 여러 개의 commit을 생성
- commit 들은 특정 hash 값 (d286bcef0b48dcf09a6f07b43659dcd03c10f53b)을 가지고 있다.
- 해당 hash 값을 이용하여 특정 commit으로 되돌리기 가능



```
MINGW64:/c/git-test
moons@DESKTOP-L9H823R MINGW64 /c/git-test (main)
$ git log --stat
commit bd747470b50adc448f63bbbbe6fa21bc496a95be (HEAD -> main)
Author: moons-86 <ekflafg@gmail.com>
Date: Sun Sep 29 22:22:07 2024 +0900

    reset3

    hello2.txt | 3 ++-
    1 file changed, 2 insertions(+), 1 deletion(-)

commit 3171ff130ba36c2f8de7baa32805d1eb8327485e
Author: moons-86 <ekflafg@gmail.com>
Date: Sun Sep 29 22:21:57 2024 +0900

    reset2

    hello2.txt | 1 +
    1 file changed, 1 insertion(+)

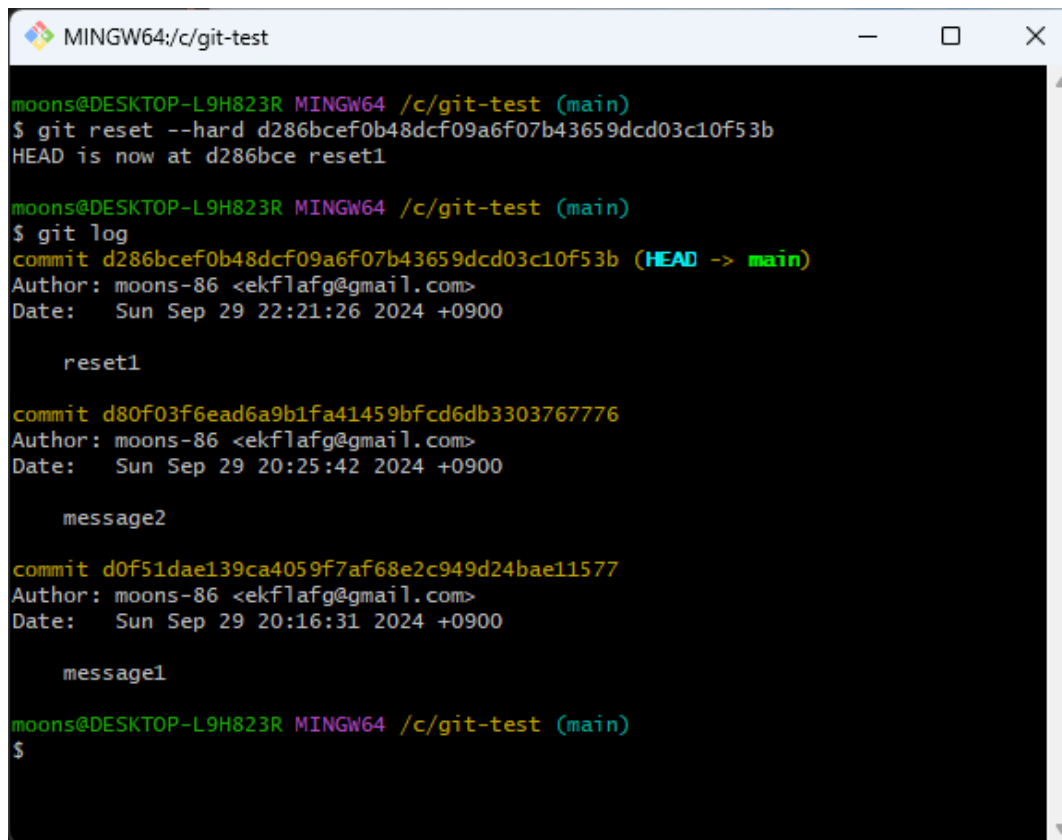
commit d286bcef0b48dcf09a6f07b43659dcd03c10f53b
Author: moons-86 <ekflafg@gmail.com>
Date: Sun Sep 29 22:21:26 2024 +0900

    reset1

    hello2.txt | 1 +
    1 file changed, 1 insertion(+)
```

특정 commit 되돌리기

- git에서 `reset --hard <hash값>` 을 이용하면 특정 commit으로 변경



```
MINGW64:/c/git-test

moons@DESKTOP-L9H823R MINGW64 /c/git-test (main)
$ git reset --hard d286bcef0b48dcf09a6f07b43659dcd03c10f53b
HEAD is now at d286bce reset1

moons@DESKTOP-L9H823R MINGW64 /c/git-test (main)
$ git log
commit d286bcef0b48dcf09a6f07b43659dcd03c10f53b (HEAD -> main)
Author: moons-86 <ekflafg@gmail.com>
Date: Sun Sep 29 22:21:26 2024 +0900

    reset1

commit d80f03f6ead6a9b1fa41459bfcd6db3303767776
Author: moons-86 <ekflafg@gmail.com>
Date: Sun Sep 29 20:25:42 2024 +0900

    message2

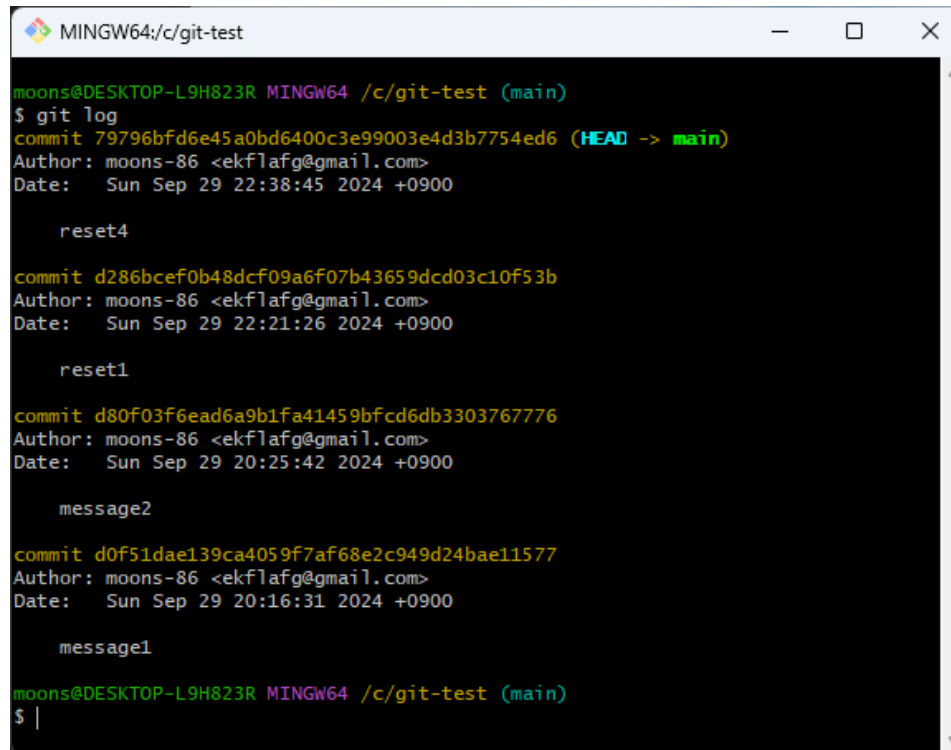
commit d0f51dae139ca4059f7af68e2c949d24bae11577
Author: moons-86 <ekflafg@gmail.com>
Date: Sun Sep 29 20:16:31 2024 +0900

    message1

moons@DESKTOP-L9H823R MINGW64 /c/git-test (main)
$
```

특정 commit 취소하기

- 특정 commit을 취소 하기 위해 하나의 commit을 추가
- 되돌리기와 취소하기의 차이는 log에 기록을 남기기 위함이다.



```
MINGW64:/c/git-test
moons@DESKTOP-L9H823R MINGW64 /c/git-test (main)
$ git log
commit 79796bfd6e45a0bd6400c3e99003e4d3b7754ed6 (HEAD -> main)
Author: moons-86 <ekflafg@gmail.com>
Date: Sun Sep 29 22:38:45 2024 +0900

    reset4

commit d286bcef0b48dcf09a6f07b43659dcd03c10f53b
Author: moons-86 <ekflafg@gmail.com>
Date: Sun Sep 29 22:21:26 2024 +0900

    reset1

commit d80f03f6ead6a9b1fa41459bfcd6db3303767776
Author: moons-86 <ekflafg@gmail.com>
Date: Sun Sep 29 20:25:42 2024 +0900

    message2

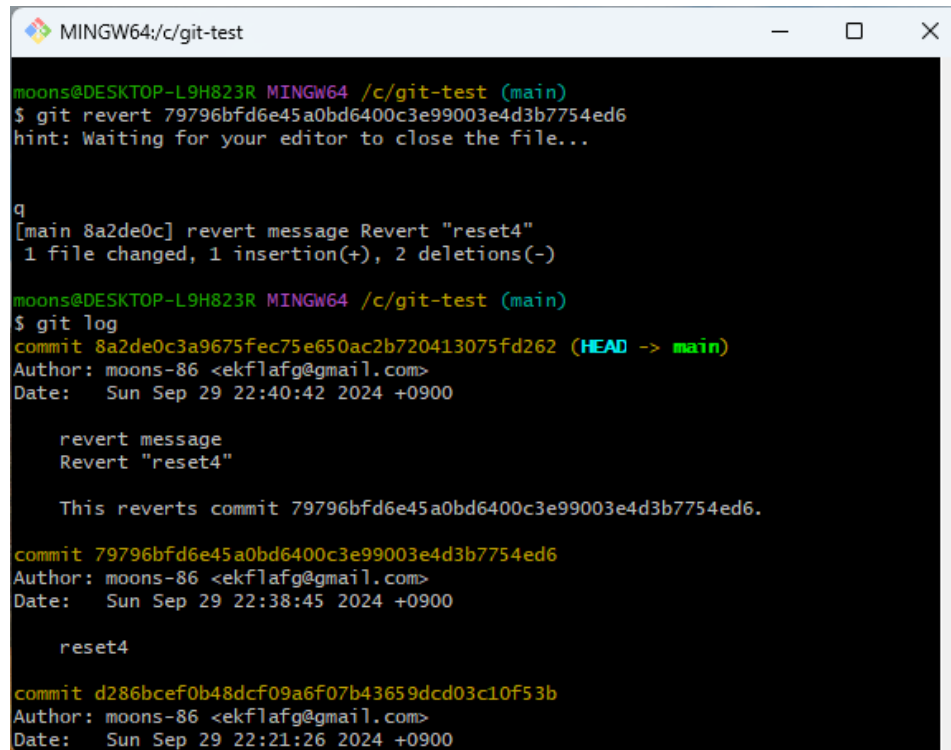
commit d0f51dae139ca4059f7af68e2c949d24bae11577
Author: moons-86 <ekflafg@gmail.com>
Date: Sun Sep 29 20:16:31 2024 +0900

    message1

moons@DESKTOP-L9H823R MINGW64 /c/git-test (main)
$ |
```

특정 commit 취소하기

- git에서 revert <hash값> 을 이용하면 특정 commit 취소가 가능
- log을 이용해서 확인해보면 취소 이력이 남아있다.



```
MINGW64:/c/git-test

moons@DESKTOP-L9H823R MINGW64 /c/git-test (main)
$ git revert 79796bfd6e45a0bd6400c3e99003e4d3b7754ed6
hint: Waiting for your editor to close the file...

q
[main 8a2de0c] revert message Revert "reset4"
1 file changed, 1 insertion(+), 2 deletions(-)

moons@DESKTOP-L9H823R MINGW64 /c/git-test (main)
$ git log
commit 8a2de0c3a9675fec75e650ac2b720413075fd262 (HEAD -> main)
Author: moons-86 <ekflafg@gmail.com>
Date: Sun Sep 29 22:40:42 2024 +0900

    revert message
    Revert "reset4"

    This reverts commit 79796bfd6e45a0bd6400c3e99003e4d3b7754ed6.

commit 79796bfd6e45a0bd6400c3e99003e4d3b7754ed6
Author: moons-86 <ekflafg@gmail.com>
Date: Sun Sep 29 22:38:45 2024 +0900

    reset4

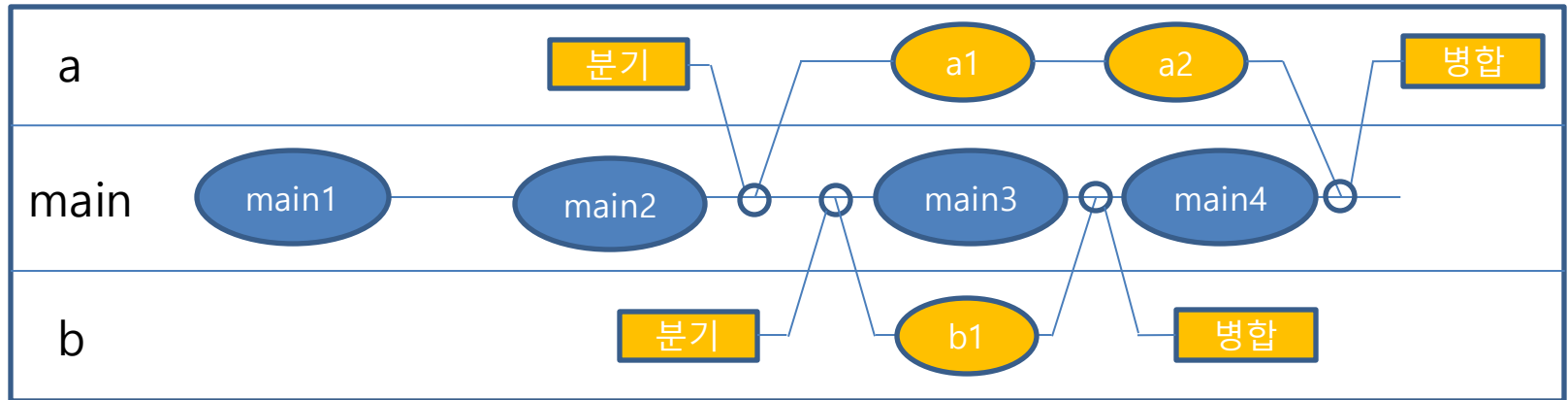
commit d286bcef0b48dcf09a6f07b43659dcd03c10f53b
Author: moons-86 <ekflafg@gmail.com>
Date: Sun Sep 29 22:21:26 2024 +0900
```

브랜치

- 브랜치는 원래 나뭇가지라는 뜻
- 나무가 가지에서 새로운 줄기를 뻗듯이 여러 갈래로 퍼지는 데이터의 흐름을 뜻한다.
- 기존의 메인 코드에서 브랜치를 따로 생성하여 기능을 추가하거나 오류를 수정 한 뒤 완료가 되면 메인 코드에 병합을 하는 방식
- 브랜치는 버전 관리 시스템에서 핵심 기능
- 협업 시 굉장히 유용하게 사용

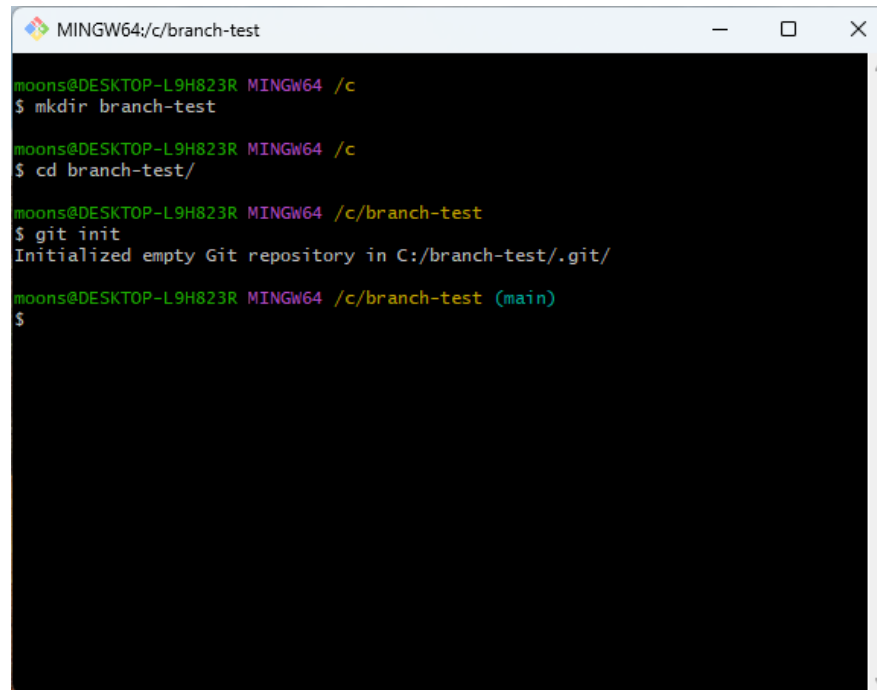
브랜치

- 깃은 처음에 시작하면 main 브랜치가 생성
- main2에서 새로운 브랜치 a, b를 생성
- 새로운 브랜치에서 작업이 끝나면 main 브랜치에 병합이 가능
- 기존의 파일은 유지하면서 새로운 공간에서 작업을 한 뒤 문제가 없을 때 기존의 파일에 병합을 하는 시스템



브랜치

- 실습을 하기 위해 새로운 깃 환경을 생성한다.
- c:에서 branch-test 폴더를 생성
- 깃 저장소로 초기화 작업



```
MINGW64:/c/branch-test

moons@DESKTOP-L9H823R MINGW64 /c
$ mkdir branch-test

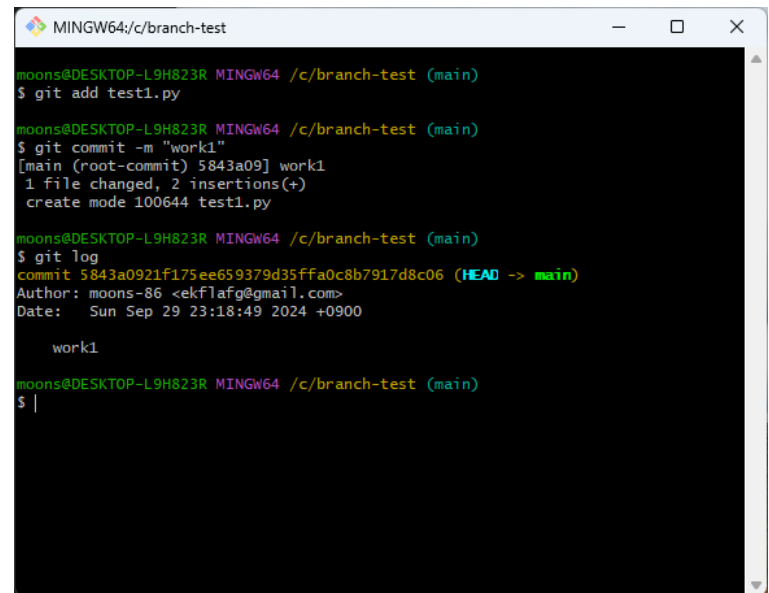
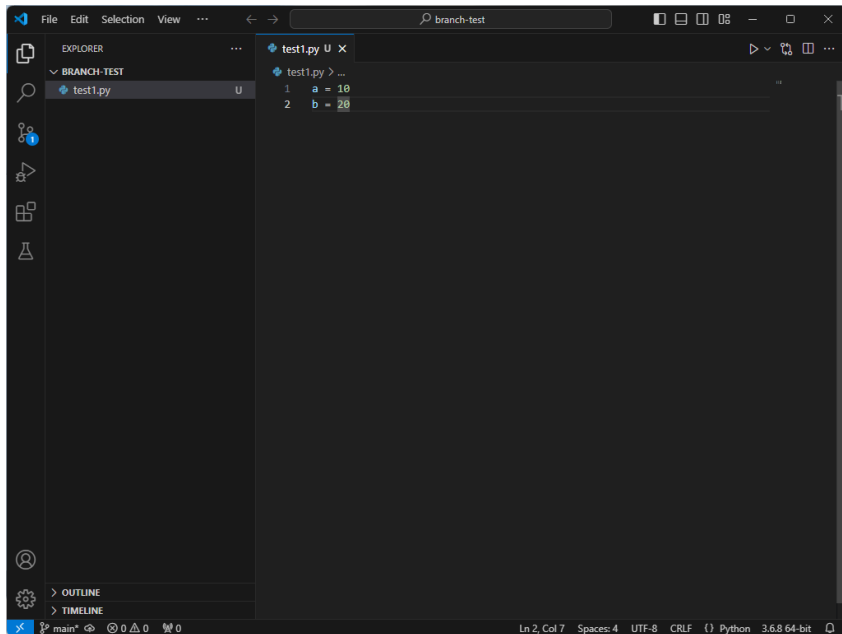
moons@DESKTOP-L9H823R MINGW64 /c
$ cd branch-test/

moons@DESKTOP-L9H823R MINGW64 /c/branch-test
$ git init
Initialized empty Git repository in C:/branch-test/.git/

moons@DESKTOP-L9H823R MINGW64 /c/branch-test (main)
$
```

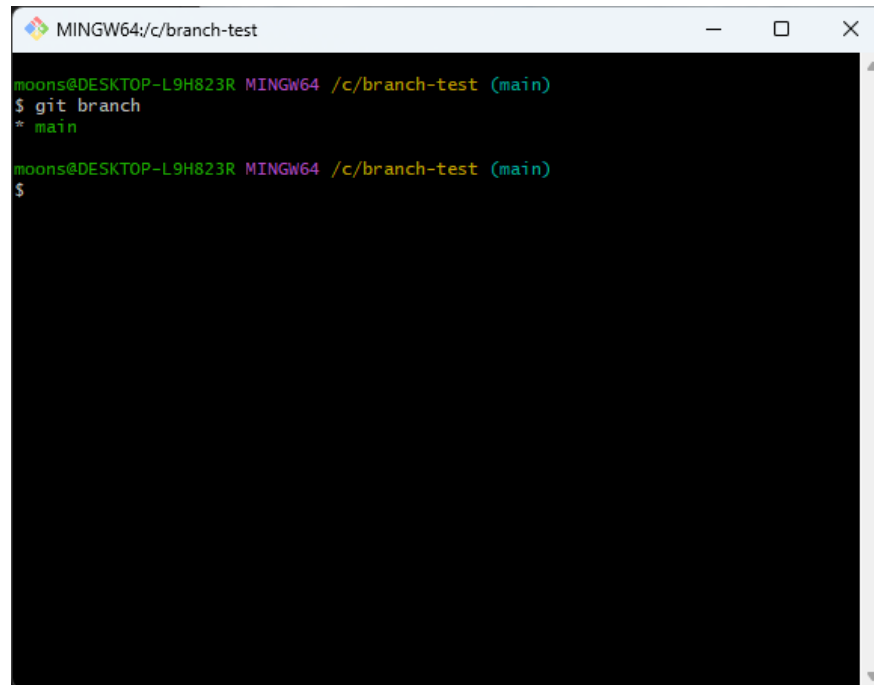
브랜치

- 해당 디렉토리에 파일을 하나 생성하고 코드를 작성 후 저장
- git 저장소에 파일을 저장
- log 메시지에 HEAD -> main 메시지는 main 브랜치를 의미
- 파일을 수정하면서 버전을 여러 개를 생성



브랜치

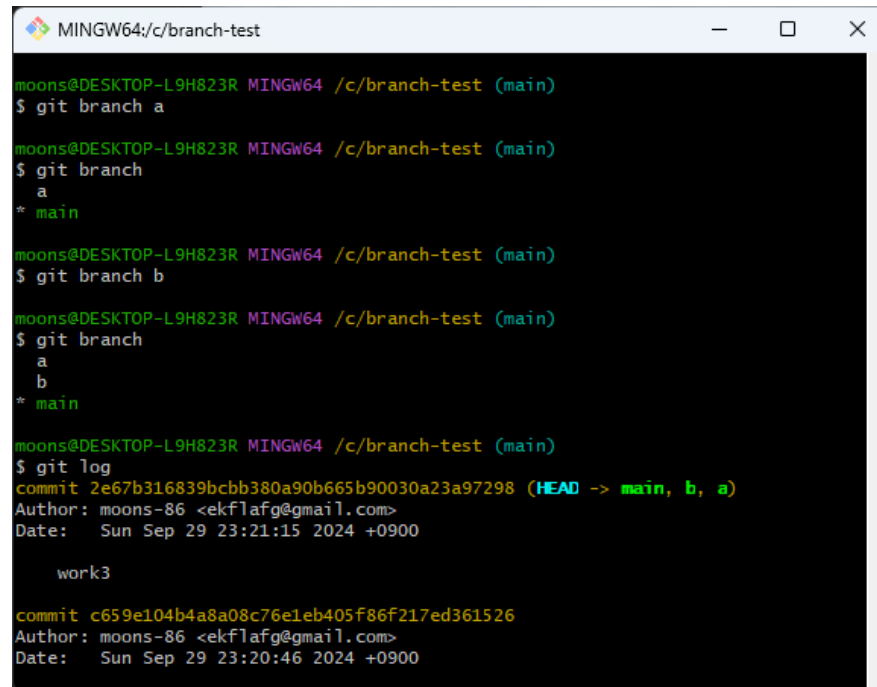
- git에서 branch 명령어를 입력하면 브랜치의 목록 확인이 가능
- 브랜치를 생성하지 않으면 main만 존재

A screenshot of a Windows terminal window titled 'MINGW64:/c/branch-test'. The prompt is 'moons@DESKTOP-L9H823R MINGW64 /c/branch-test (main)'. The user enters the command '\$ git branch'. The output is '* main'. The prompt then changes to 'moons@DESKTOP-L9H823R MINGW64 /c/branch-test (main)' followed by a new '\$' prompt.

```
MINGW64:/c/branch-test  
moons@DESKTOP-L9H823R MINGW64 /c/branch-test (main)  
$ git branch  
* main  
moons@DESKTOP-L9H823R MINGW64 /c/branch-test (main)  
$
```

브랜치

- git branch <브랜치명>을 이용하면 새로운 브랜치 생성 가능
- a, b 브랜치를 생성하고 log를 확인하면 최근 commit이 main, a, b로 변경



```
MINGW64:/c/branch-test

moons@DESKTOP-L9H823R MINGW64 /c/branch-test (main)
$ git branch a

moons@DESKTOP-L9H823R MINGW64 /c/branch-test (main)
$ git branch
a
* main

moons@DESKTOP-L9H823R MINGW64 /c/branch-test (main)
$ git branch b

moons@DESKTOP-L9H823R MINGW64 /c/branch-test (main)
$ git branch
a
b
* main

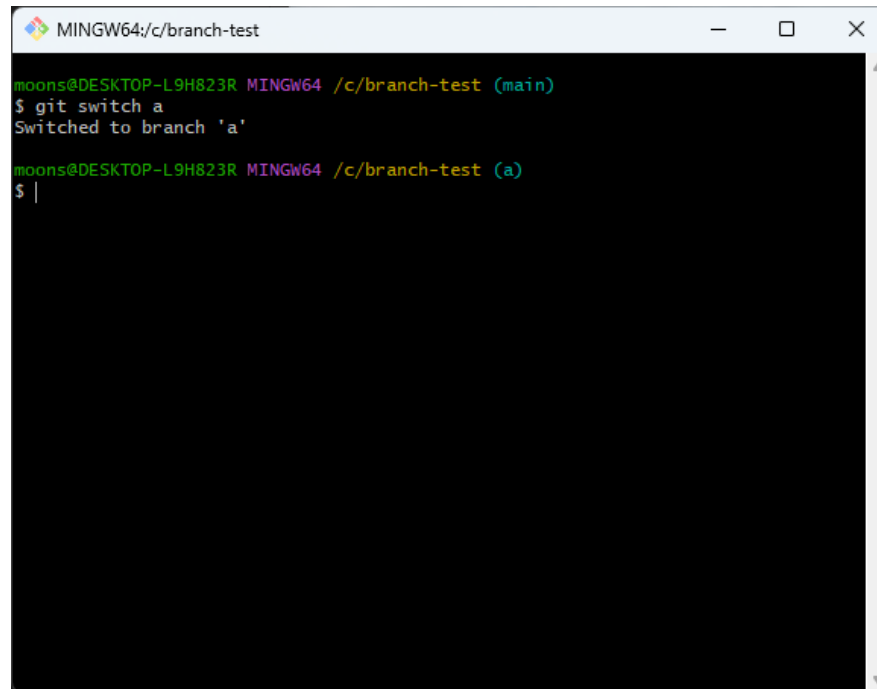
moons@DESKTOP-L9H823R MINGW64 /c/branch-test (main)
$ git log
commit 2e67b316839bcbb380a90b665b90030a23a97298 (HEAD -> main, b, a)
Author: moons-86 <ekflafg@gmail.com>
Date: Sun Sep 29 23:21:15 2024 +0900

    work3

commit c659e104b4a8a08c76e1eb405f86f217ed361526
Author: moons-86 <ekflafg@gmail.com>
Date: Sun Sep 29 23:20:46 2024 +0900
```

브랜치

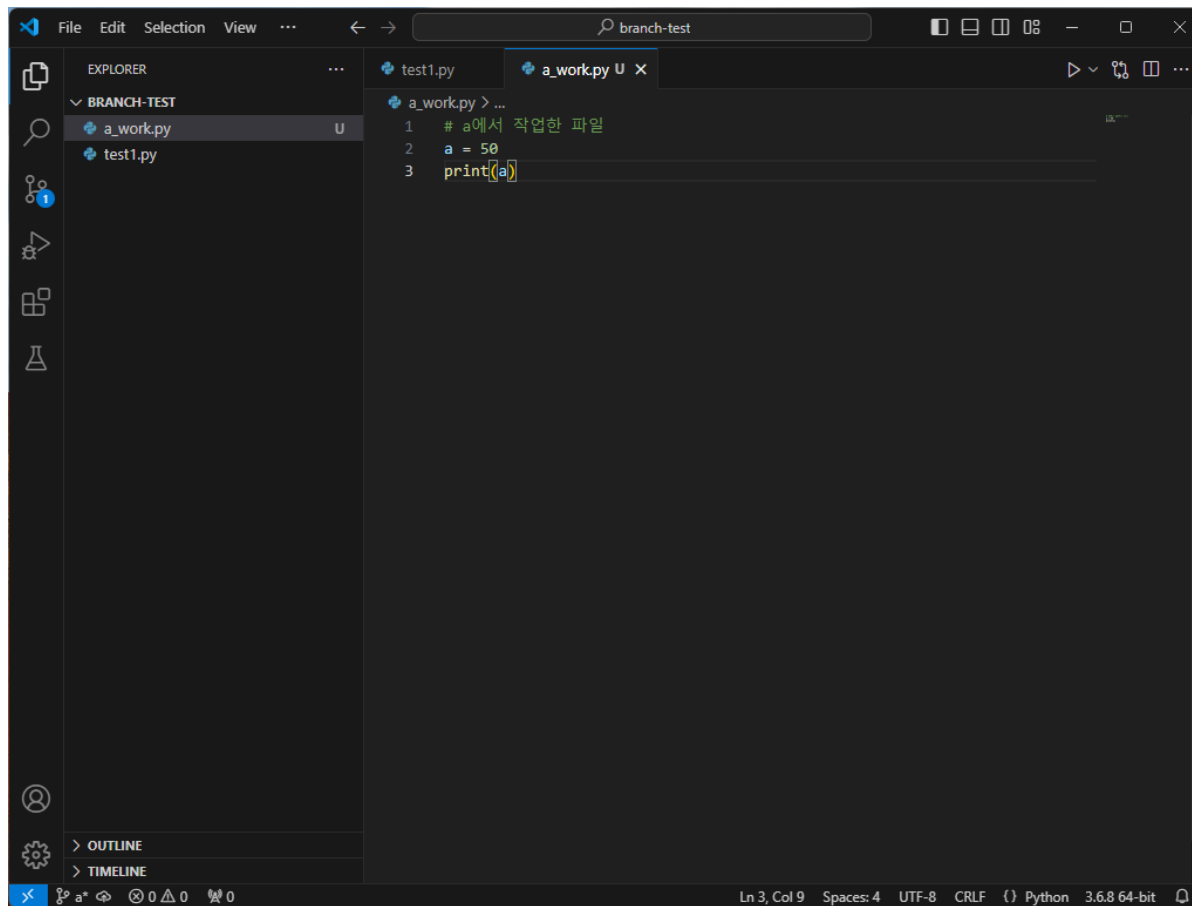
- git switch a 명령어로 브랜치를 변경
- 브랜치를 변경하게 되면 work3 commit과 같은 버전으로 파일이 변경

A screenshot of a Windows terminal window titled 'MINGW64:/c/branch-test'. The prompt is 'moons@DESKTOP-L9H823R MINGW64 /c/branch-test (main)'. The user enters '\$ git switch a', and the terminal responds 'Switched to branch 'a''. The prompt then changes to 'moons@DESKTOP-L9H823R MINGW64 /c/branch-test (a)' followed by a new '\$' prompt.

```
MINGW64:/c/branch-test  
moons@DESKTOP-L9H823R MINGW64 /c/branch-test (main)  
$ git switch a  
Switched to branch 'a'  
moons@DESKTOP-L9H823R MINGW64 /c/branch-test (a)  
$ |
```

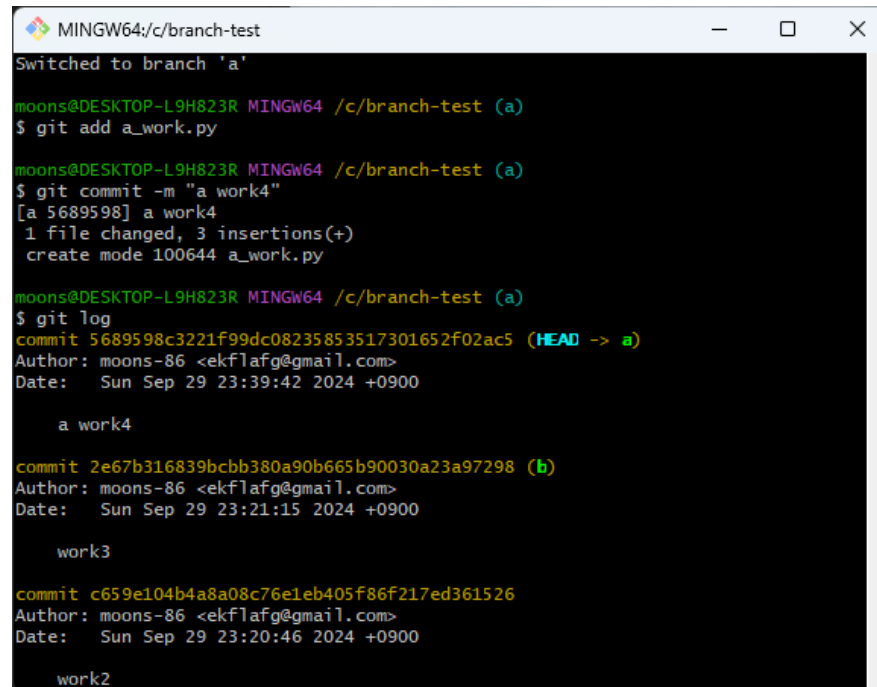
브랜치

- 파일을 하나 생성 후 코드를 입력한 뒤 저장



브랜치

- 브랜치가 a인 상태에서 새로 생성한 파일을 저장소에 버전으로 등록
- log을 확인하면 a의 브랜치로 새로운 commit이 생성



```
MINGW64:/c/branch-test
Switched to branch 'a'

moons@DESKTOP-L9H823R MINGW64 /c/branch-test (a)
$ git add a_work.py

moons@DESKTOP-L9H823R MINGW64 /c/branch-test (a)
$ git commit -m "a work4"
[a 5689598] a work4
1 file changed, 3 insertions(+)
create mode 100644 a_work.py

moons@DESKTOP-L9H823R MINGW64 /c/branch-test (a)
$ git log
commit 5689598c3221f99dc08235853517301652f02ac5 (HEAD -> a)
Author: moons-86 <ekflafg@gmail.com>
Date: Sun Sep 29 23:39:42 2024 +0900

    a work4

commit 2e67b316839bcbb380a90b665b90030a23a97298 (b)
Author: moons-86 <ekflafg@gmail.com>
Date: Sun Sep 29 23:21:15 2024 +0900

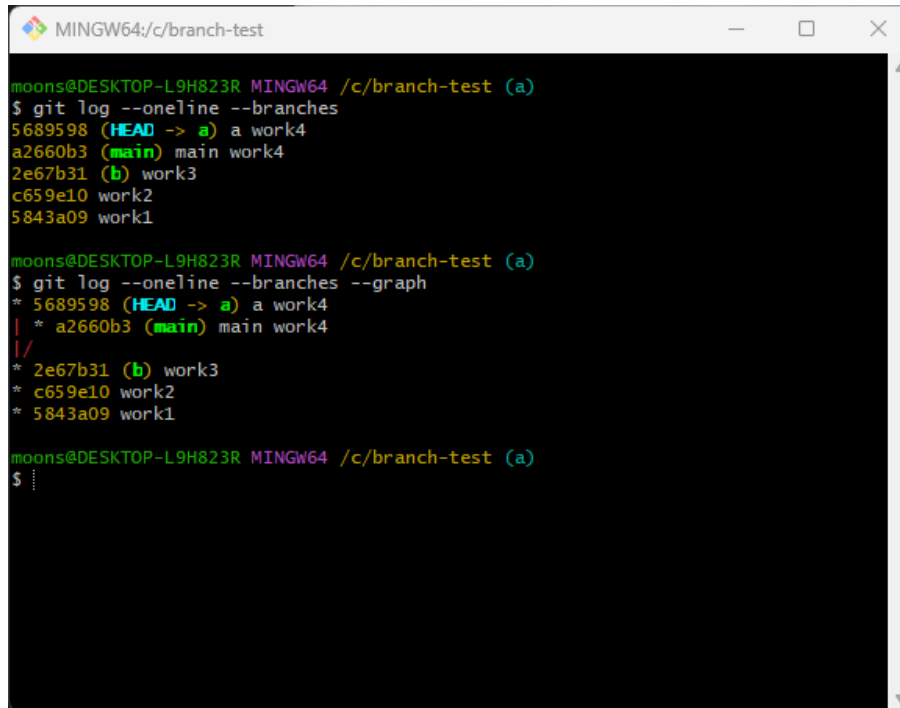
    work3

commit c659e104b4a8a08c76e1eb405f86f217ed361526
Author: moons-86 <ekflafg@gmail.com>
Date: Sun Sep 29 23:20:46 2024 +0900

    work2
```

브랜치

- log에서 --branches 를 이용하면 최신 commit을 한눈에 확인 가능
- --graph 를 이용하면 commit 간의 관계를 보여준다
- work3까지는 main, a, b가 같고 main work4는 main 브랜치의 최신 버전이고 a는 a work4 버전을 최신으로 보여준다



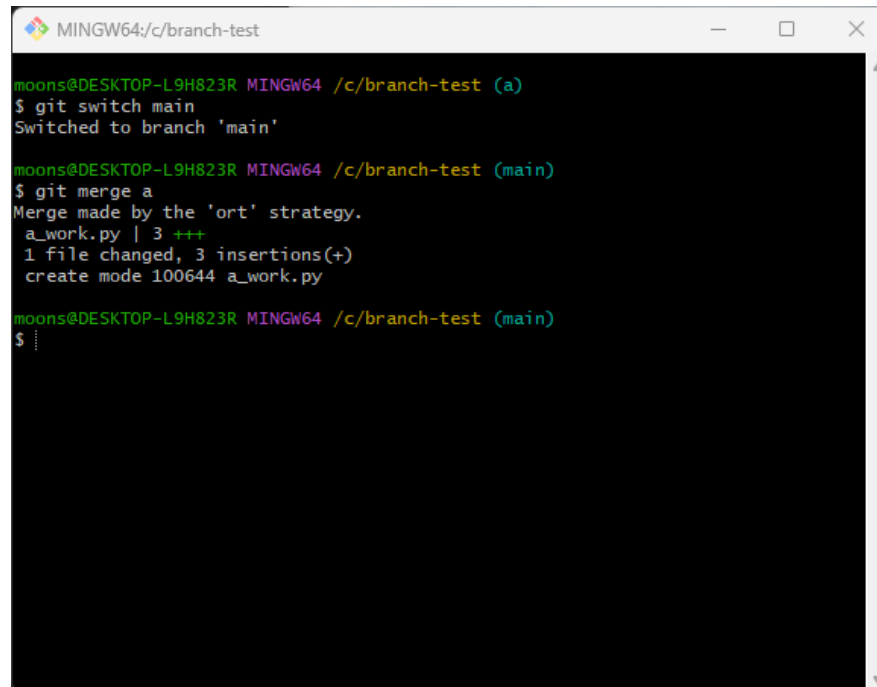
```
MINGW64:/c/branch-test
moons@DESKTOP-L9H823R MINGW64 /c/branch-test (a)
$ git log --oneline --branches
5689598 (HEAD -> a) a work4
a2660b3 (main) main work4
2e67b31 (b) work3
c659e10 work2
5843a09 work1

moons@DESKTOP-L9H823R MINGW64 /c/branch-test (a)
$ git log --oneline --branches --graph
* 5689598 (HEAD -> a) a work4
| * a2660b3 (main) main work4
|/
* 2e67b31 (b) work3
* c659e10 work2
* 5843a09 work1

moons@DESKTOP-L9H823R MINGW64 /c/branch-test (a)
$
```

브랜치

- 두 개의 브랜치를 main에 병합하려면 merge 명령어를 이용
- main 브랜치로 변경 한 뒤 git merge <브랜치명>를 사용하면
Vscode에 새로운 문서가 생성 되며 해당 문서를 닫으면 병합 완료

A terminal window titled 'MINGW64:/c/branch-test' showing the execution of git merge. The user switches to the 'main' branch and then merges branch 'a'. The output shows the merge was successful using the 'ort' strategy, with 1 file changed and 3 insertions. A new file 'a_work.py' was created.

```
MINGW64:/c/branch-test

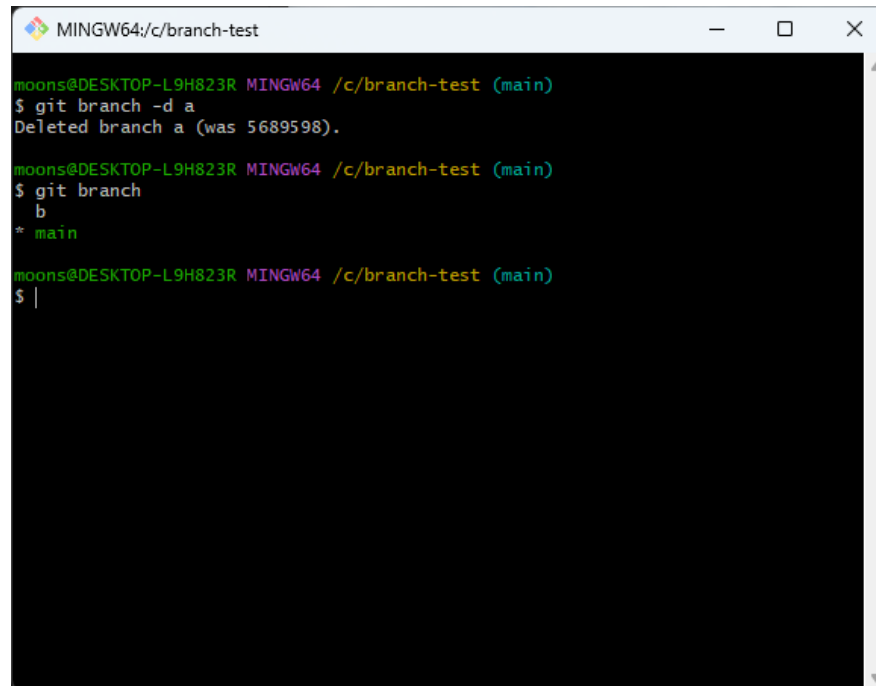
moons@DESKTOP-L9H823R MINGW64 /c/branch-test (a)
$ git switch main
Switched to branch 'main'

moons@DESKTOP-L9H823R MINGW64 /c/branch-test (main)
$ git merge a
Merge made by the 'ort' strategy.
 a_work.py | 3 +++
 1 file changed, 3 insertions(+)
 create mode 100644 a_work.py

moons@DESKTOP-L9H823R MINGW64 /c/branch-test (main)
$ ...
```

브랜치

- 브랜치를 병합 한 뒤 해당 브랜치를 제거하려면 `-d`를 이용하여 브랜치 제거가 가능



```
MINGW64:/c/branch-test

moons@DESKTOP-L9H823R MINGW64 /c/branch-test (main)
$ git branch -d a
Deleted branch a (was 5689598).

moons@DESKTOP-L9H823R MINGW64 /c/branch-test (main)
$ git branch
  b
* main

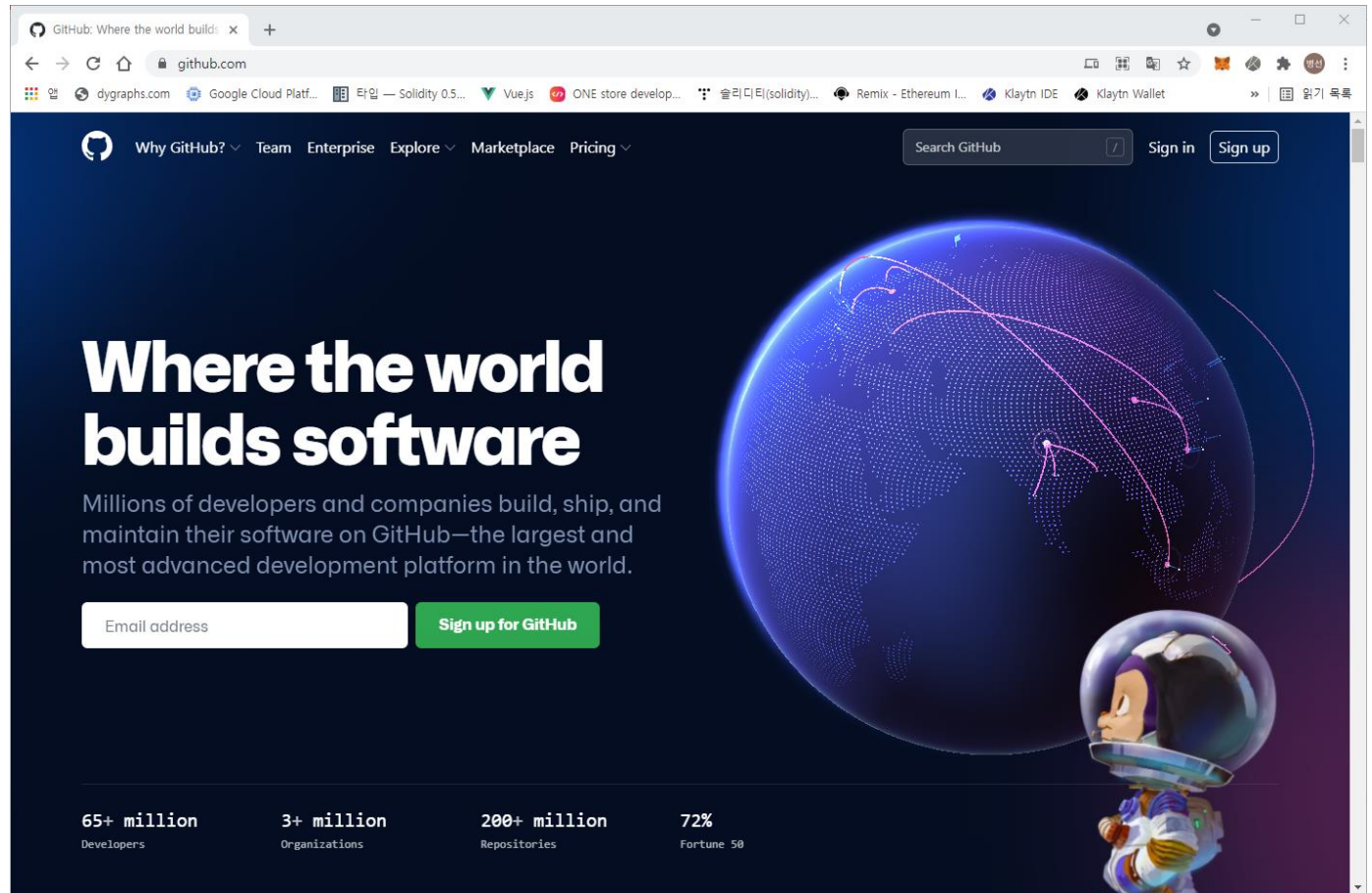
moons@DESKTOP-L9H823R MINGW64 /c/branch-test (main)
$ |
```




깃허브

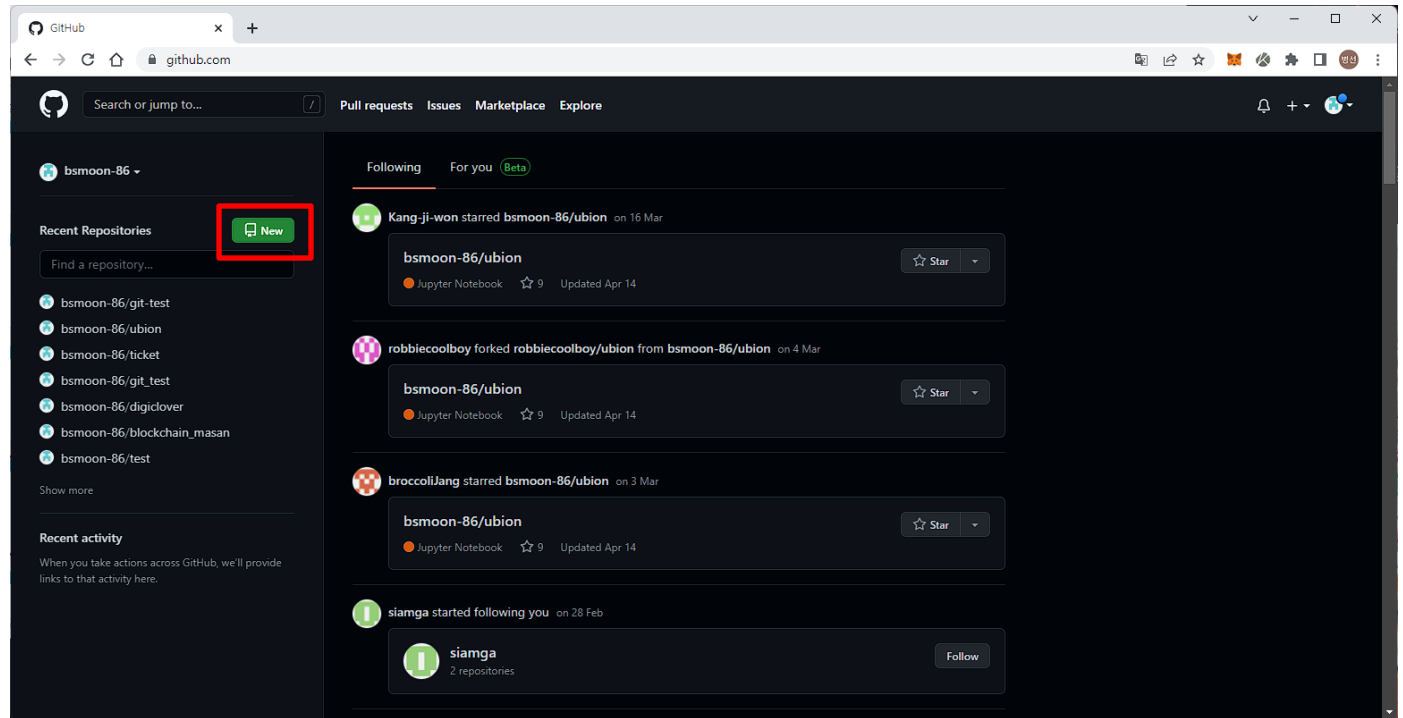
깃허브 회원가입

<https://github.com/>



깃허브 레파지토리 생성

깃허브 로그인 후 좌측 상단의 NEW 버튼 클릭

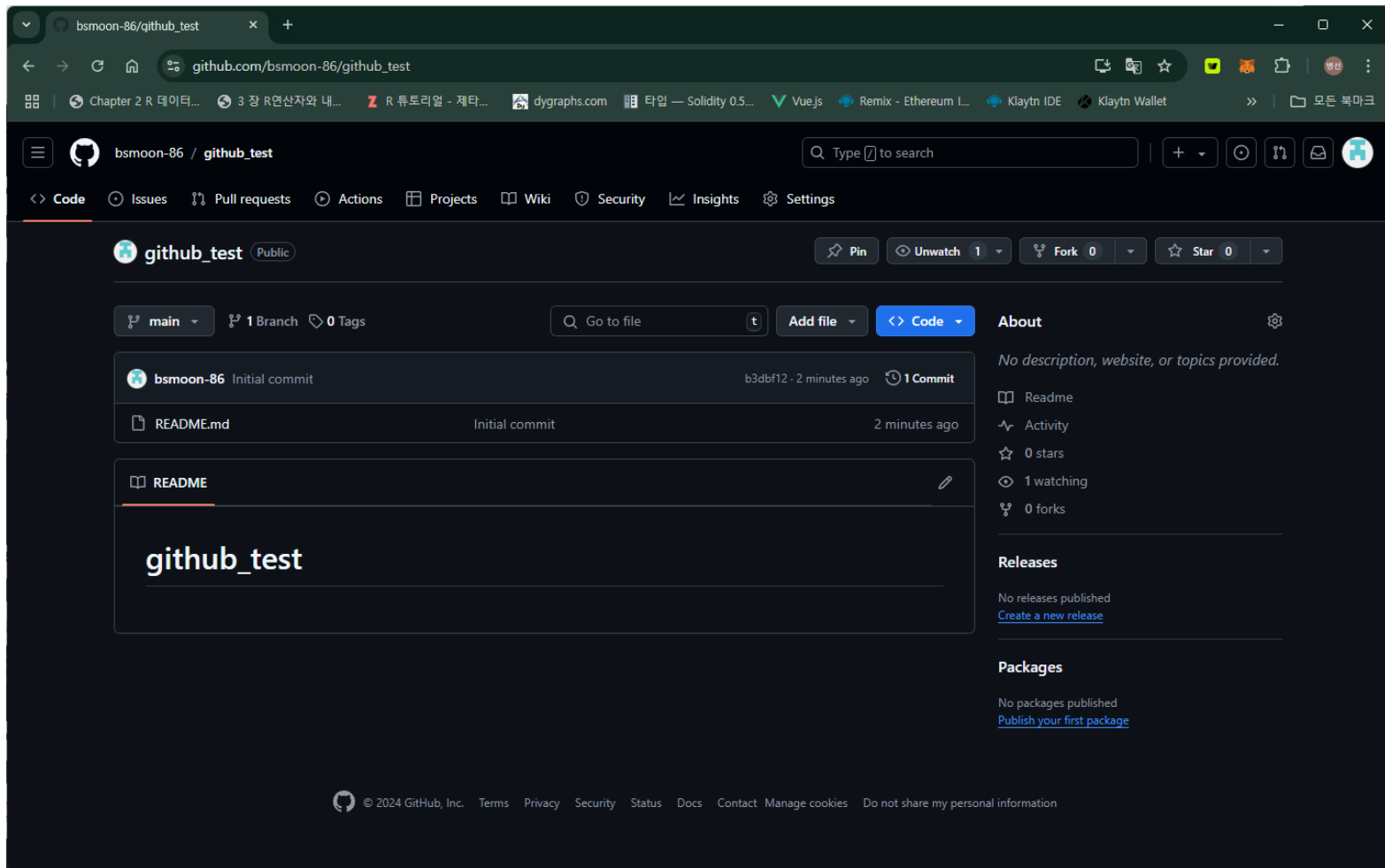


깃허브 레파지토리 생성

- 레파지토리 네임 입력
- Add a README file 체크
- Create repository 클릭
- Repository 생성 완료

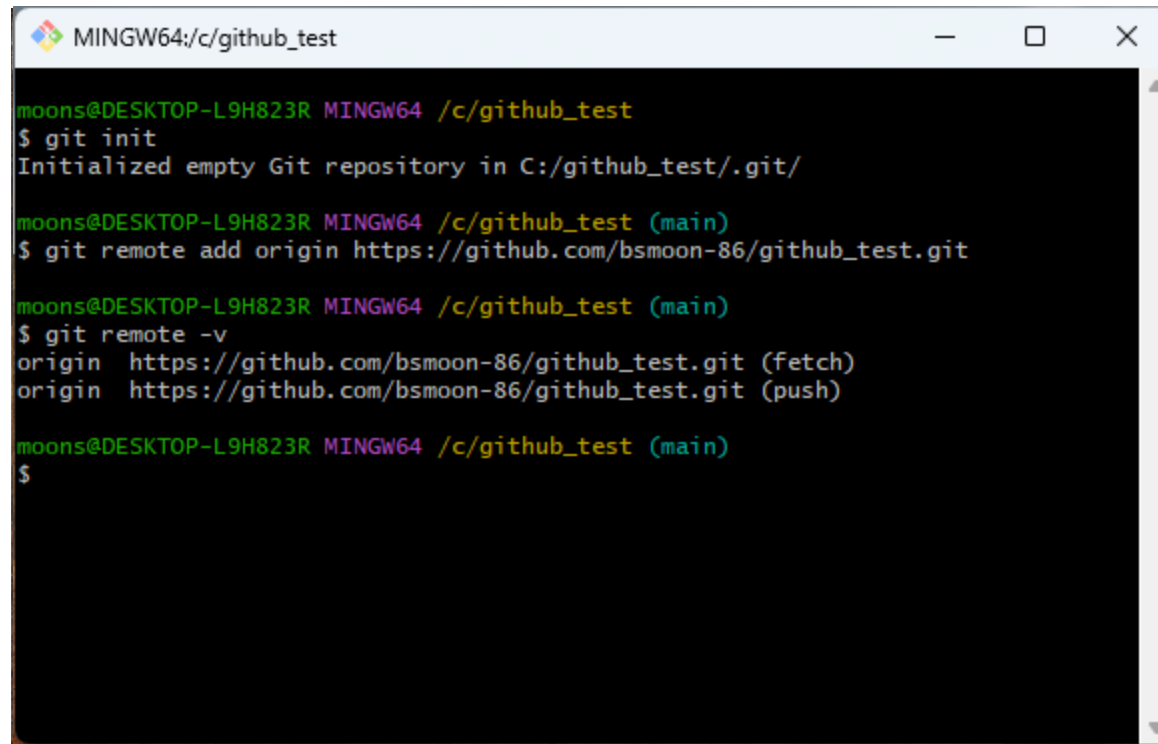
The screenshot shows the GitHub 'Create a new repository' page. The page title is 'Create a new repository' with a subtitle 'A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.' The form includes fields for 'Owner' (set to 'bsmoon-86') and 'Repository name'. Below these is a note: 'Great repository names are short and memorable. Need inspiration? How about [cautious-guide](#)?' There is a 'Description (optional)' text area. The 'Visibility' section has two radio buttons: 'Public' (selected) and 'Private'. Under 'Public', it says 'Anyone on the internet can see this repository. You choose who can commit.' Under 'Private', it says 'You choose who can see and commit to this repository.' The 'Initialize this repository with:' section has a checkbox for 'Add a README file' (checked) with a note 'This is where you can write a long description for your project. [Learn more.](#)'. Below this is the 'Add .gitignore' section with a dropdown menu set to '.gitignore template: None'. The 'Choose a license' section has a dropdown menu set to 'License: None'. At the bottom, there is a blue box with an information icon and the text 'You are creating a public repository in your personal account.' and a green 'Create repository' button.

깃허브 레파지토리 생성 화면



깃허브 사용법 (CMD)

- 내 컴퓨터와 깃허브의 레파지토리를 연결하려면 컴퓨터에 깃 저장소를 생성하고 remote 명령어를 이용하여 저장소를 연결한다.



```
mingw64@DESKTOP-L9H823R MINGW64 /c/github_test
$ git init
Initialized empty Git repository in C:/github_test/.git/

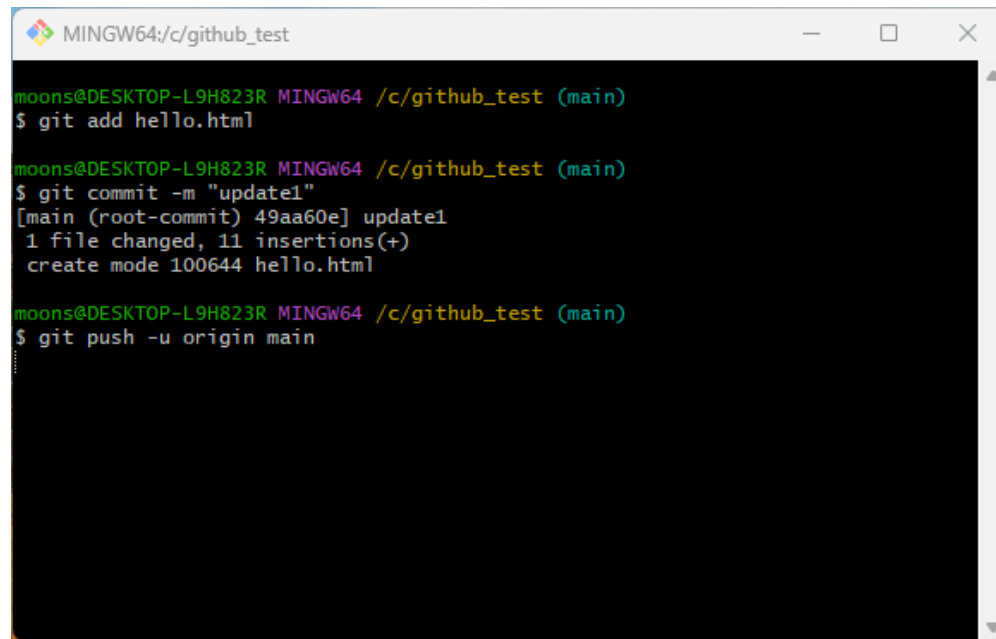
mingw64@DESKTOP-L9H823R MINGW64 /c/github_test (main)
$ git remote add origin https://github.com/bsmoon-86/github_test.git

mingw64@DESKTOP-L9H823R MINGW64 /c/github_test (main)
$ git remote -v
origin  https://github.com/bsmoon-86/github_test.git (fetch)
origin  https://github.com/bsmoon-86/github_test.git (push)

mingw64@DESKTOP-L9H823R MINGW64 /c/github_test (main)
$
```

깃허브 사용법 (CMD)

- 원격 저장소에 commit을 올리려면 깃 저장소에서 add, commit을 실행
- push 명령어를 이용하여 원격 저장소에 저장



```
MINGW64:/c/github_test

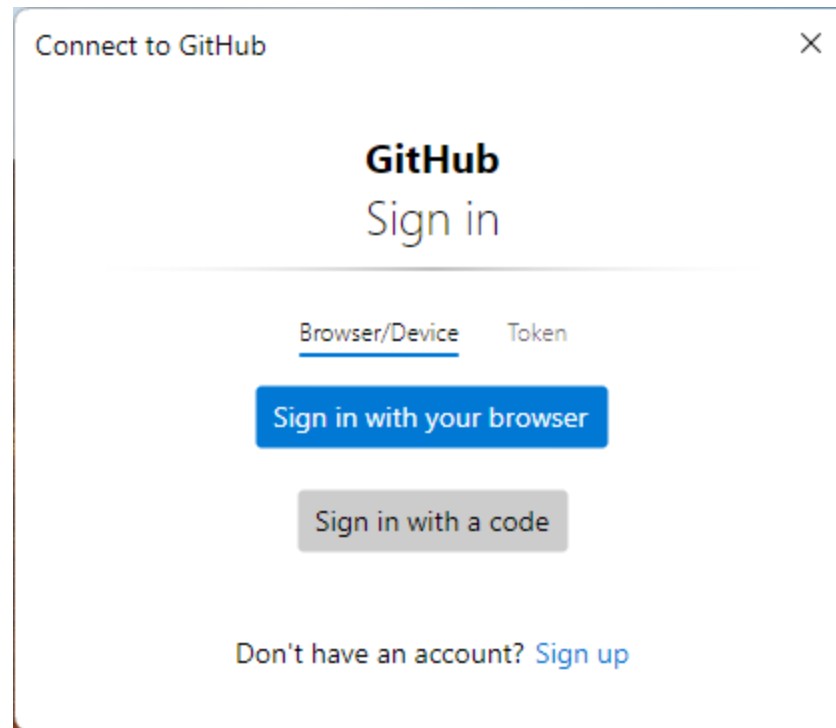
moons@DESKTOP-L9H823R MINGW64 /c/github_test (main)
$ git add hello.html

moons@DESKTOP-L9H823R MINGW64 /c/github_test (main)
$ git commit -m "update1"
[main (root-commit) 49aa60e] update1
1 file changed, 11 insertions(+)
create mode 100644 hello.html

moons@DESKTOP-L9H823R MINGW64 /c/github_test (main)
$ git push -u origin main
```

깃허브 사용법 (CMD)

- 처음으로 push 명령어를 이용하게 되면 깃허브 로그인 화면이 나타난다.



깃허브 사용법 (CMD)

- 로그인 화면이 나오지 않는 경우 (MAC)

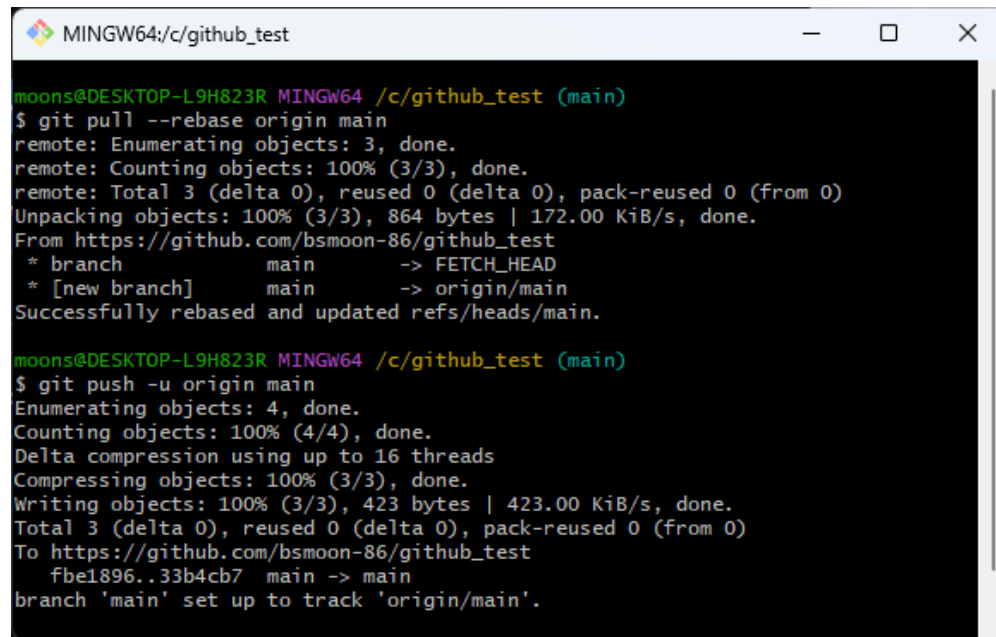
brew install git-credential-manager
(홈브루를 이용한 패키지 설치)

git-credential-manager configure
(터미널 입력)

git push -u origin main

깃허브 사용법 (CMD)

- rejected 에러가 발생한다면 깃 허브의 원격 저장소와 로컬의 깃 저장소의 파일의 목록들이 다르기 때문
- rebase를 이용하여 로컬 저장소 위에 다시 적용 시킨다.
(readme 파일을 생성하지 않으면 rejected 에러 발생X)



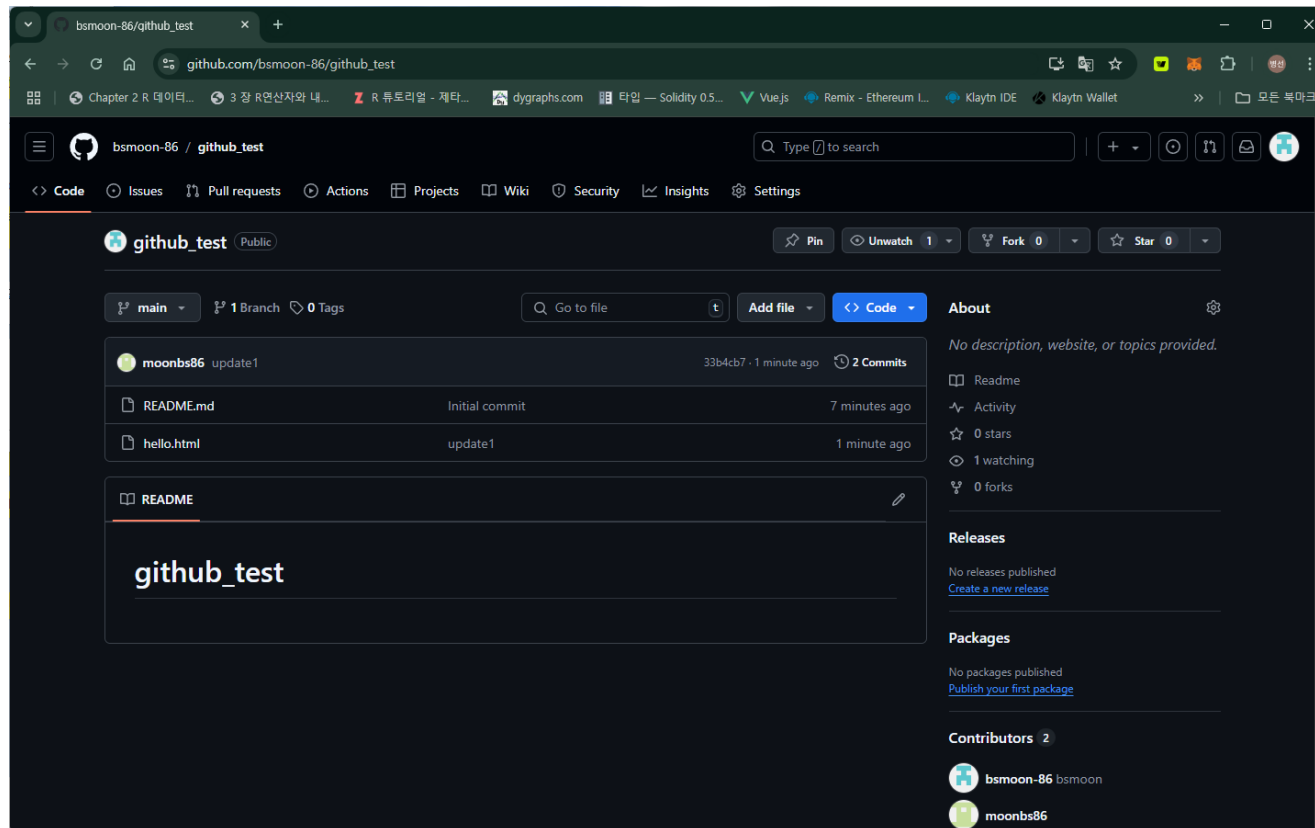
```
MINGW64:/c/github_test

moons@DESKTOP-L9H823R MINGW64 /c/github_test (main)
$ git pull --rebase origin main
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (3/3), 864 bytes | 172.00 KiB/s, done.
From https://github.com/bsmoon-86/github_test
 * branch          main      -> FETCH_HEAD
 * [new branch]     main      -> origin/main
Successfully rebased and updated refs/heads/main.

moons@DESKTOP-L9H823R MINGW64 /c/github_test (main)
$ git push -u origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 16 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 423 bytes | 423.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/bsmoon-86/github_test
 fbe1896..33b4cb7  main -> main
branch 'main' set up to track 'origin/main'.
```

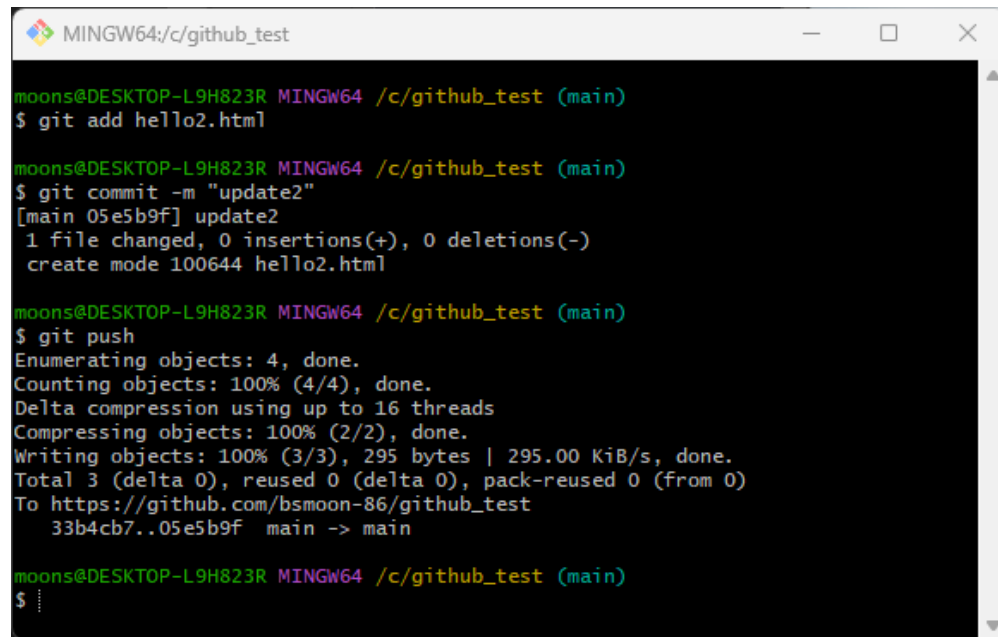
깃허브 사용법 (CMD)

- push가 정상적으로 완료 되면 깃허브 레파지토리에 해당하는 파일이 업로드



깃허브 사용법 (CMD)

- push를 한번 성공 했다면 그 다음부터는 더 간단하게 push 가능
- 파일을 생성하고 add, commit 후 git push 만으로 원격 저장소에 업로드가 가능하다.



```
MINGW64:/c/github_test

moons@DESKTOP-L9H823R MINGW64 /c/github_test (main)
$ git add hello2.html

moons@DESKTOP-L9H823R MINGW64 /c/github_test (main)
$ git commit -m "update2"
[main 05e5b9f] update2
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 hello2.html

moons@DESKTOP-L9H823R MINGW64 /c/github_test (main)
$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 295 bytes | 295.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/bsmoon-86/github_test
33b4cb7..05e5b9f  main -> main

moons@DESKTOP-L9H823R MINGW64 /c/github_test (main)
$
```

마크다운

Markdown은 텍스트 기반의 마크업언어로 2004년 존그루버에 의해 만들어졌으며 쉽게 쓰고 읽을 수 있으며 HTML로 변환이 가능하다. 특수기호와 문자를 이용한 매우 간단한 구조의 문법을 사용하여 웹에서도 보다 빠르게 콘텐츠를 작성하고 보다 직관적으로 인식할 수 있다.

마크다운 장점

간결하다.

별도의 도구없이 작성 가능하다.

다양한 형태로 변환이 가능하다.

텍스트(Text)로 저장되기 때문에 용량이 적어 보관이 용이하다.

텍스트파일이기 때문에 버전관리시스템을 이용하여 변경이력을 관리할 수 있다.

지원하는 프로그램과 플랫폼이 다양하다.

마크다운 단점

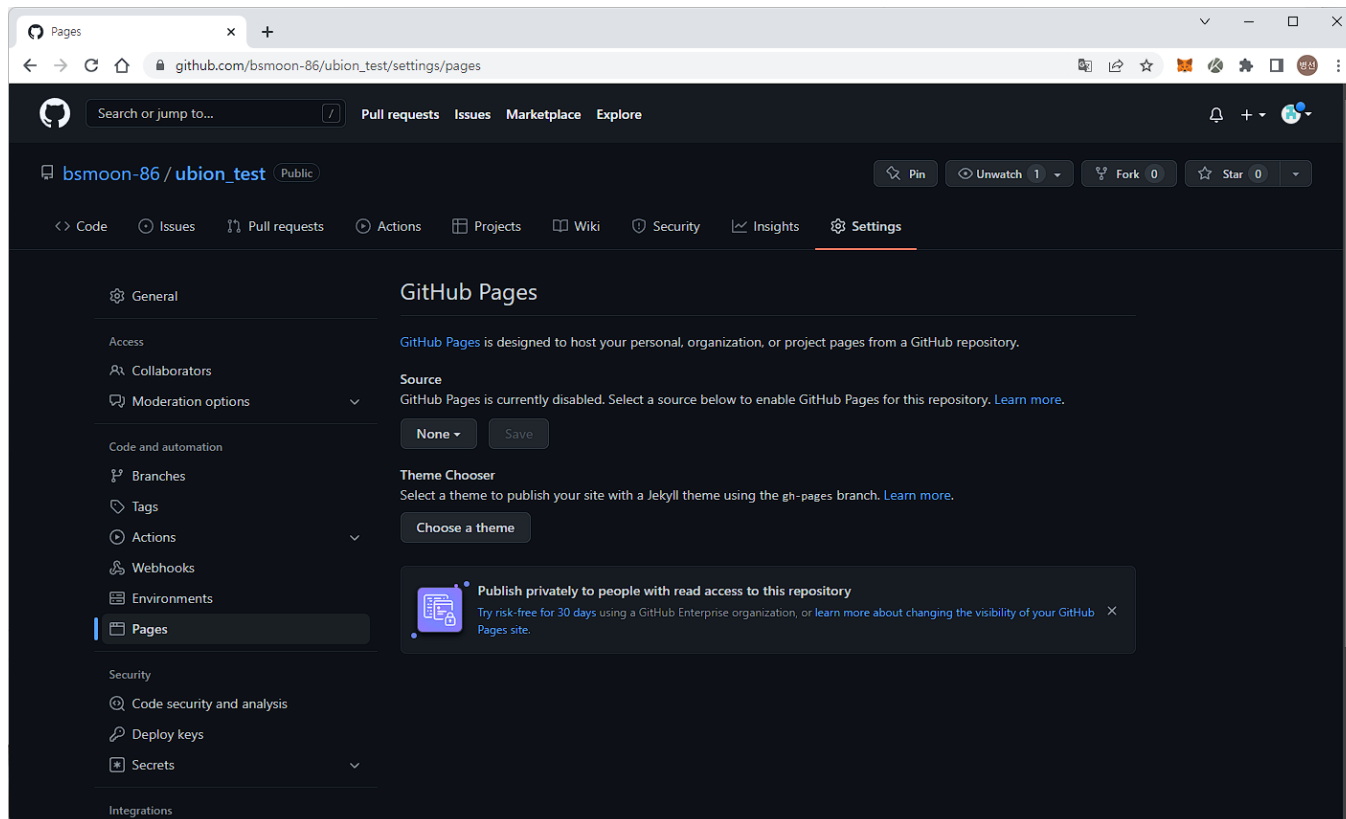
표준이 없다.

표준이 없기 때문에 도구에 따라서 변환방식이나 생성물이 다르다.

모든 HTML 마크업을 대신하지 못한다.

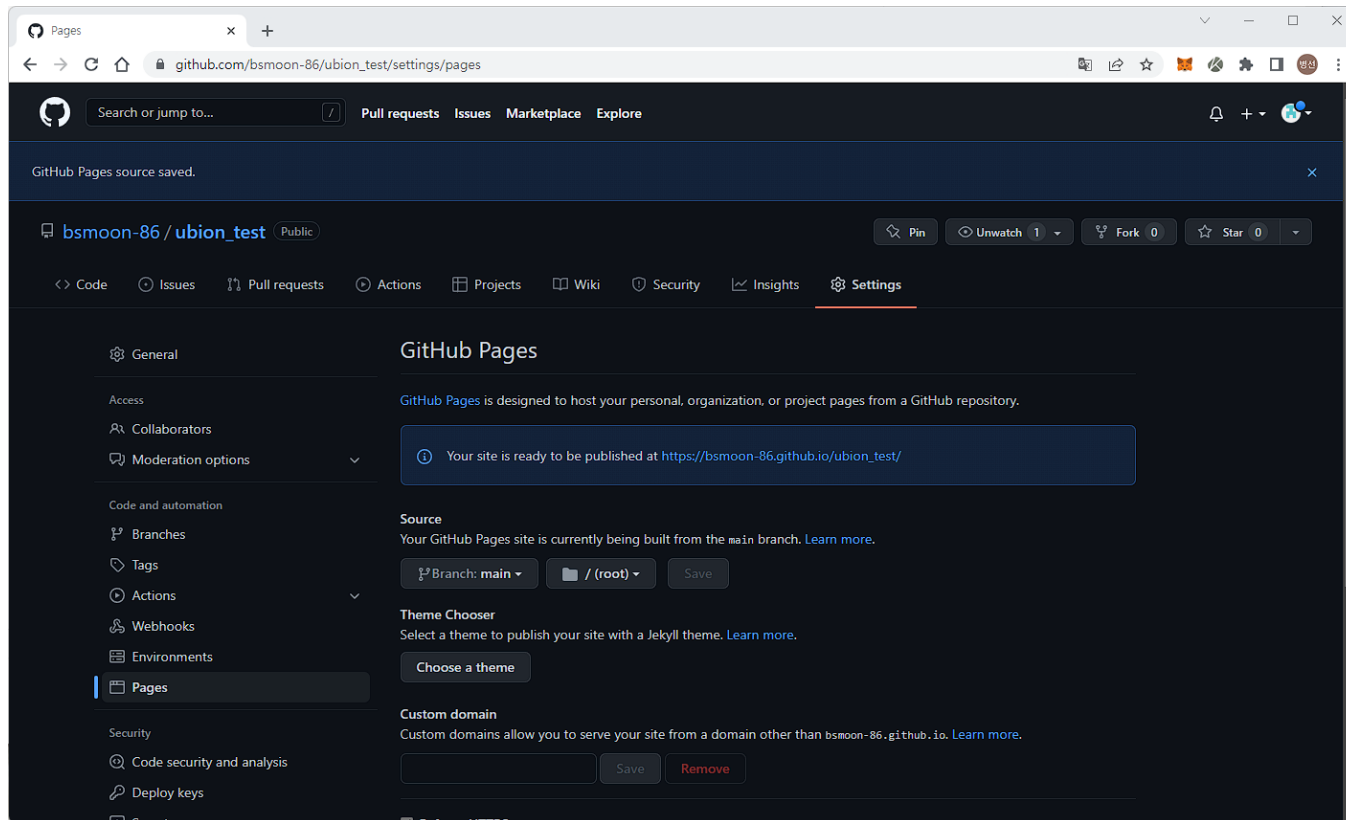
깃허브 웹서버 구축

만들어둔 레파지토리에서 setting 클릭
좌측 메뉴에서 pages 클릭



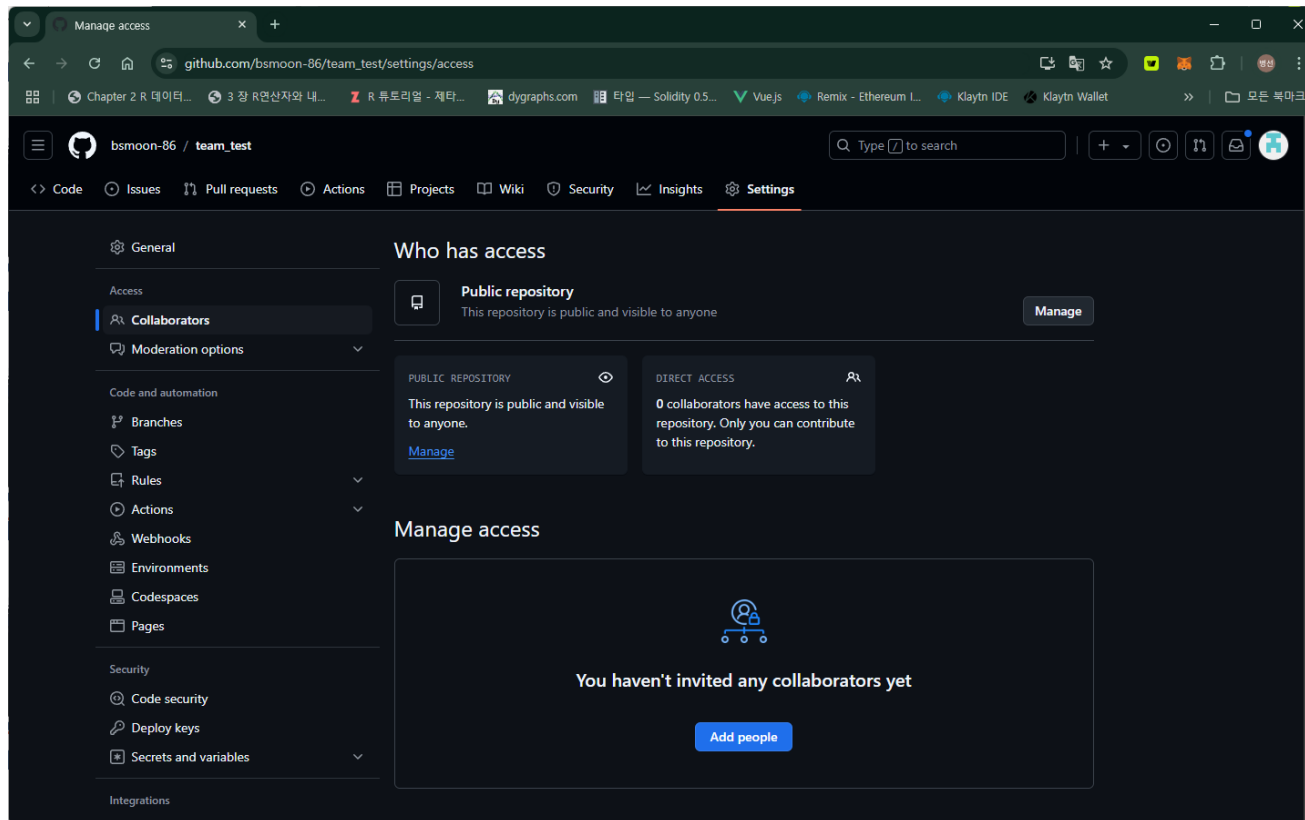
깃허브 웹서버 구축

Source 부분 None으로 되어있는 리스트 박스를
branch main으로 변경 후 Save 클릭



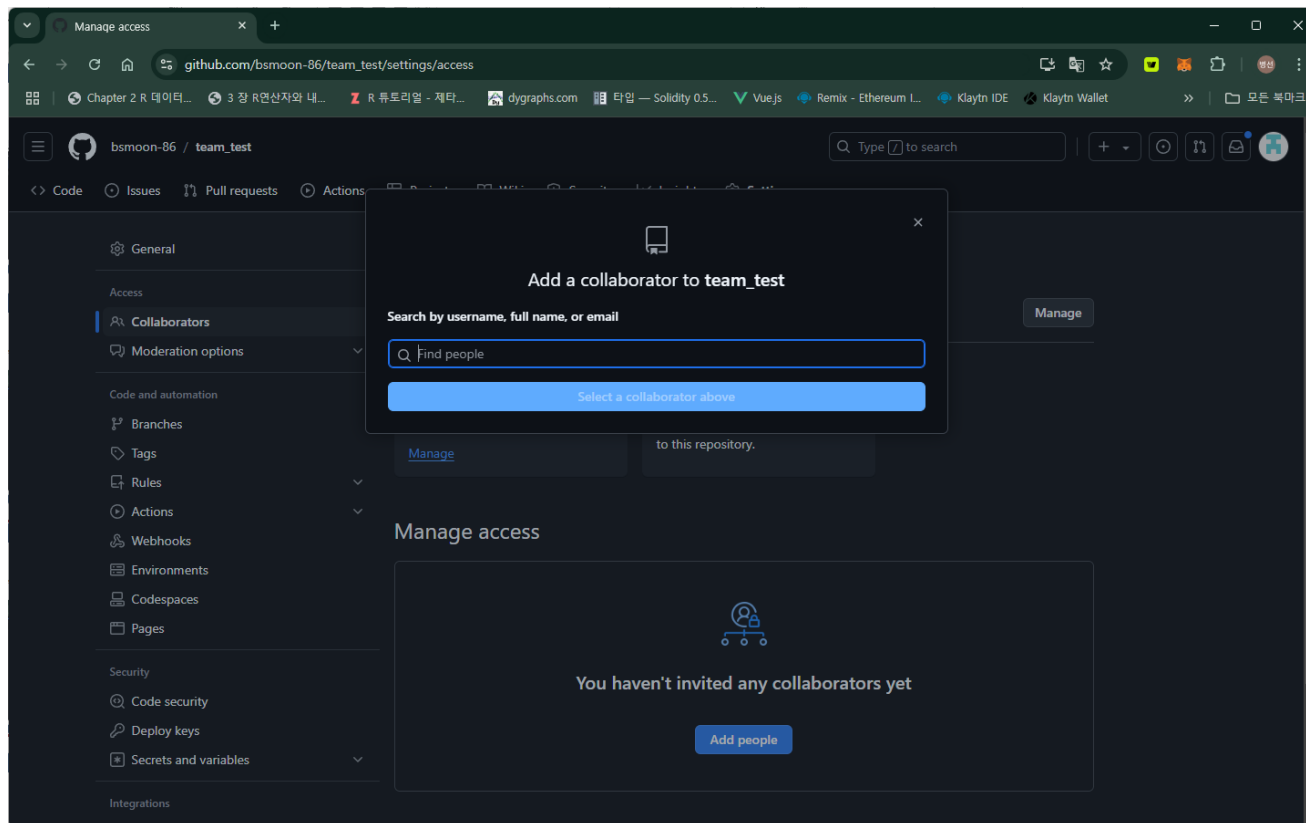
깃허브 원격저장소 협업하기

- repo를 생성 후 setting메뉴로 진입 후 좌측의 Collaborators 로 진입 -> Add people 버튼 클릭



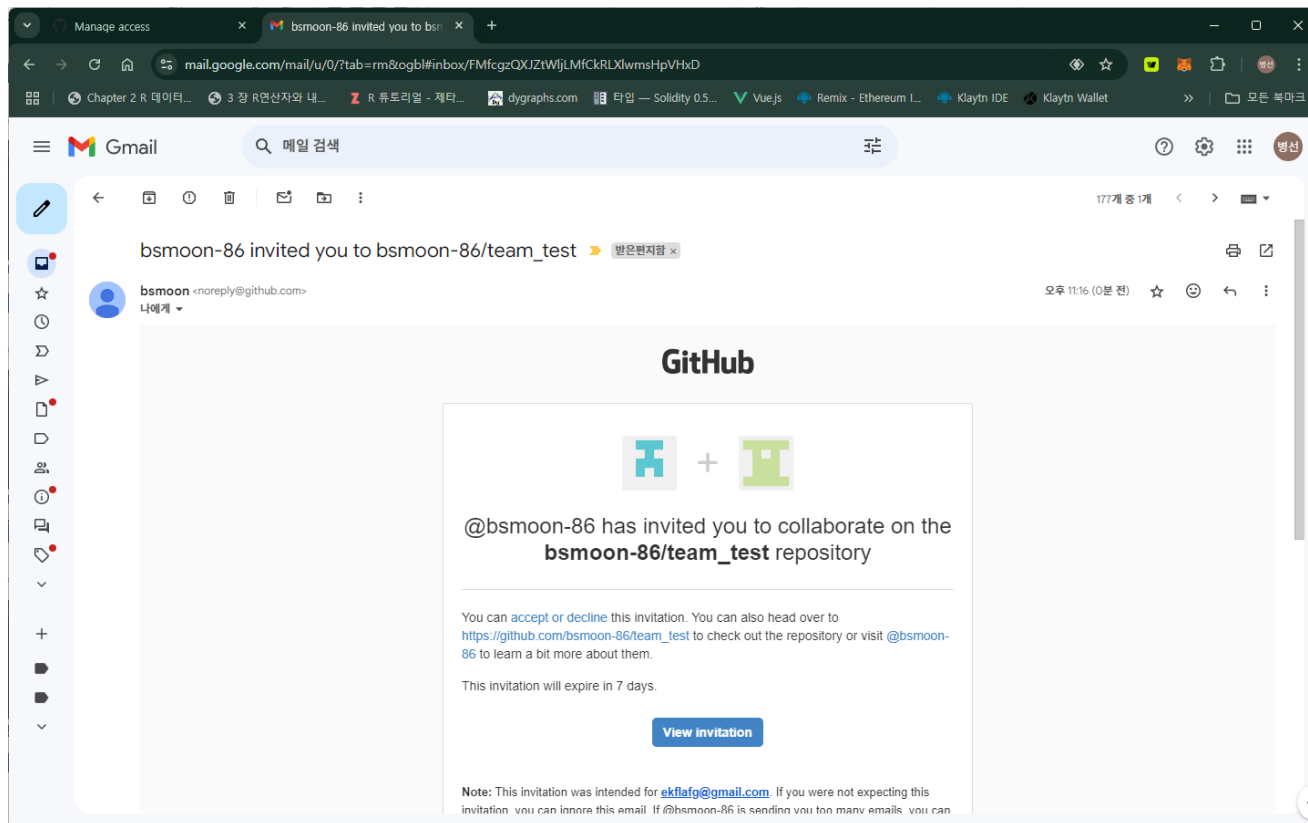
깃허브 원격저장소 협업하기

해당하는 입력 칸에 같이 작업을 하려는 깃허브 유저의 이름이나 이메일을 입력한다.



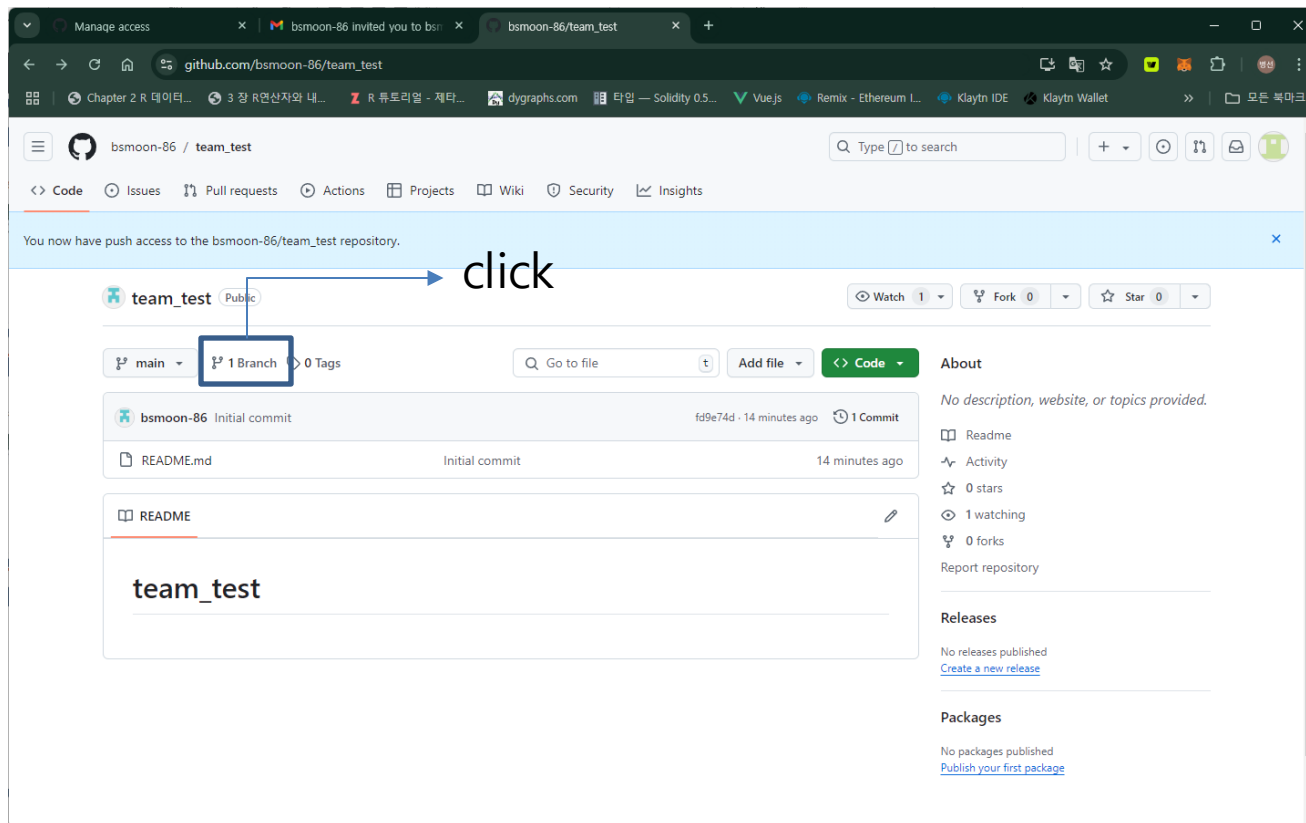
깃허브 원격저장소 협업하기

초대 받은 계정의 이메일을 확인하면 초대 메시지가 메일로 들어온다.



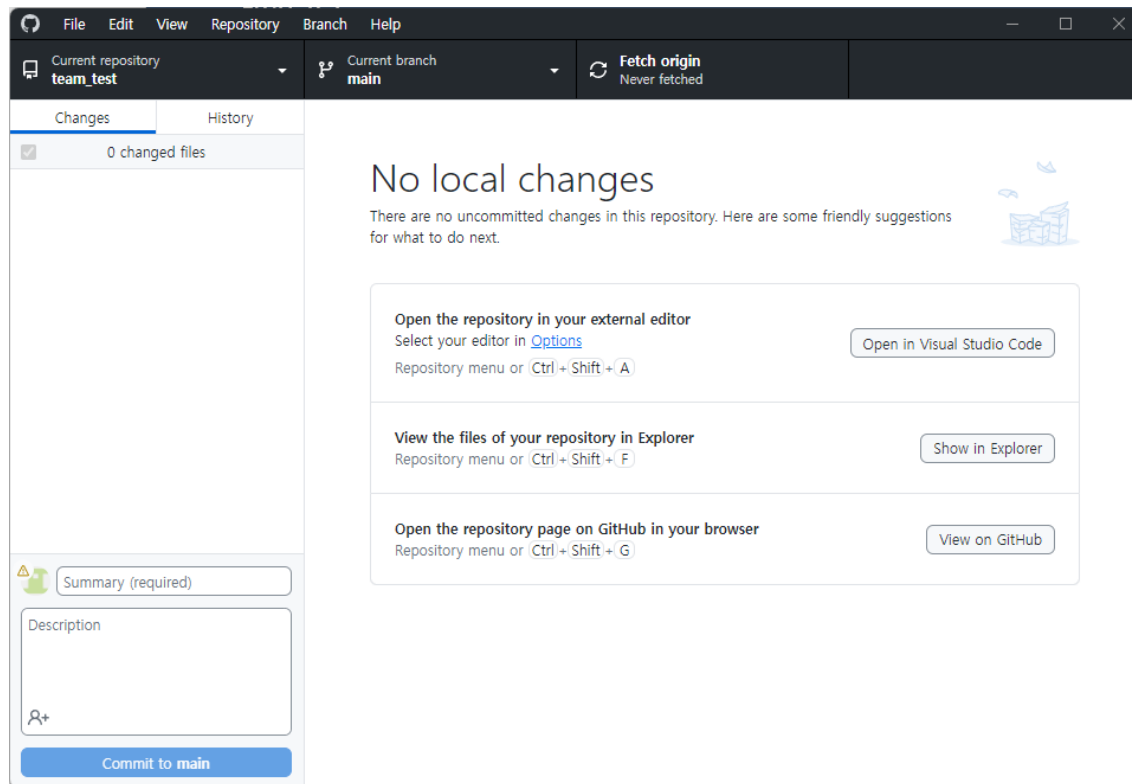
깃허브 원격저장소 협업하기

Branchs 클릭하여 새로운 브랜치를 생성
github desktop를 이용하여 해당 repo를 연결



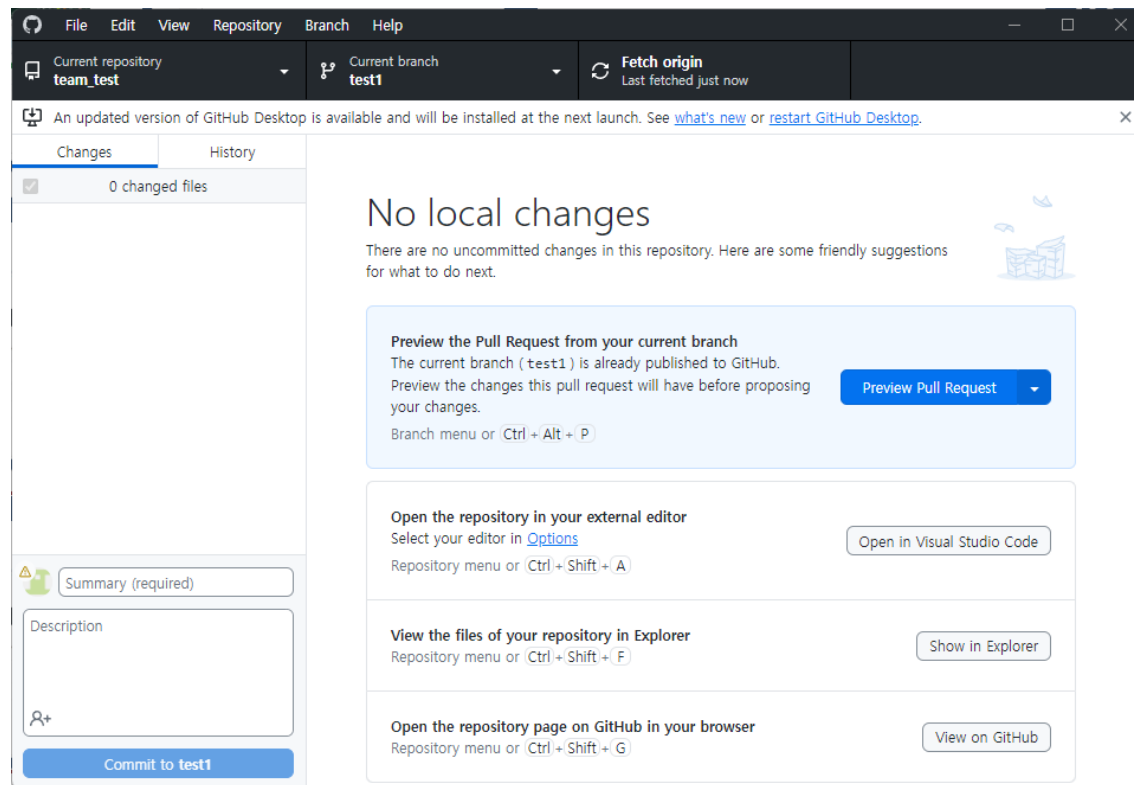
깃허브 원격저장소 협업하기

Current branch를 클릭하여 생성한 branch를 선택 후
Open in Visual Studio Code 클릭 후 파일을 생성하여 commit ->
push를 해준다.



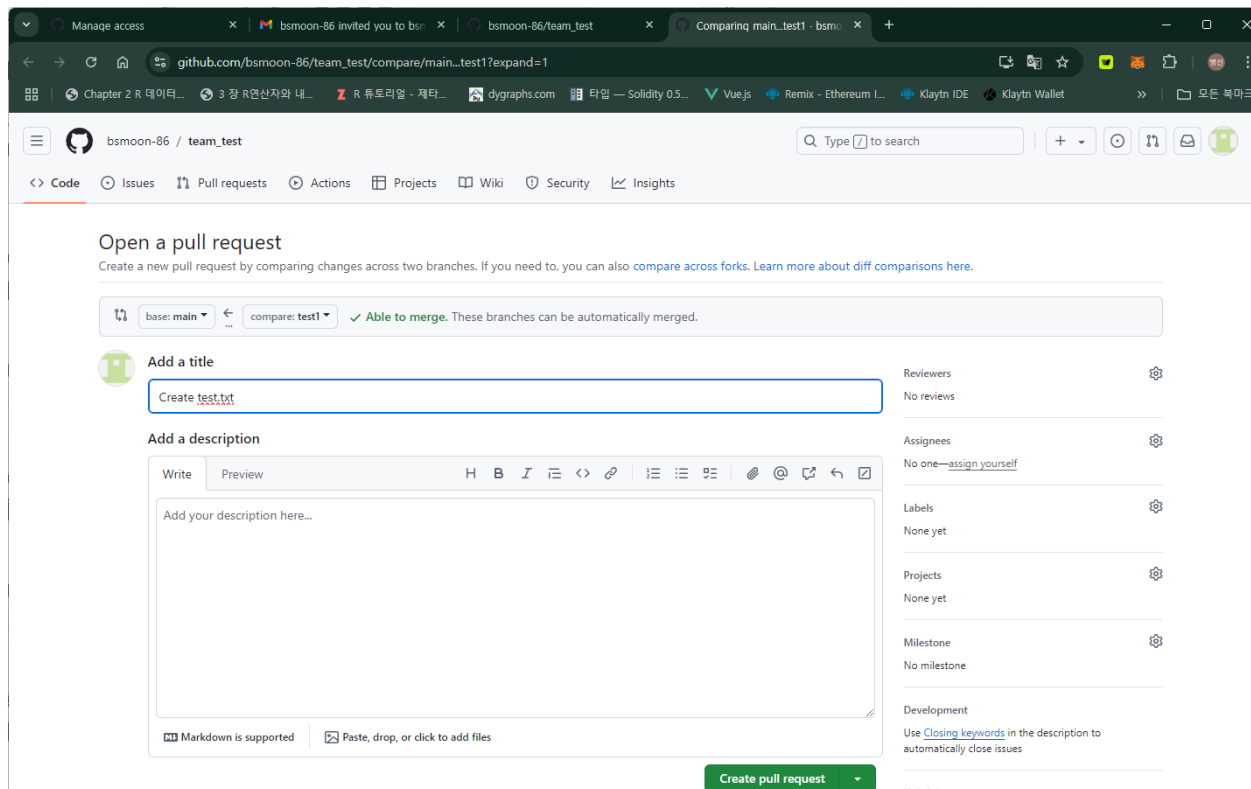
깃허브 원격저장소 협업하기

Preview Pull Request 버튼을 클릭
새로운 창이 나왔을때 Create pull request 버튼을 클릭



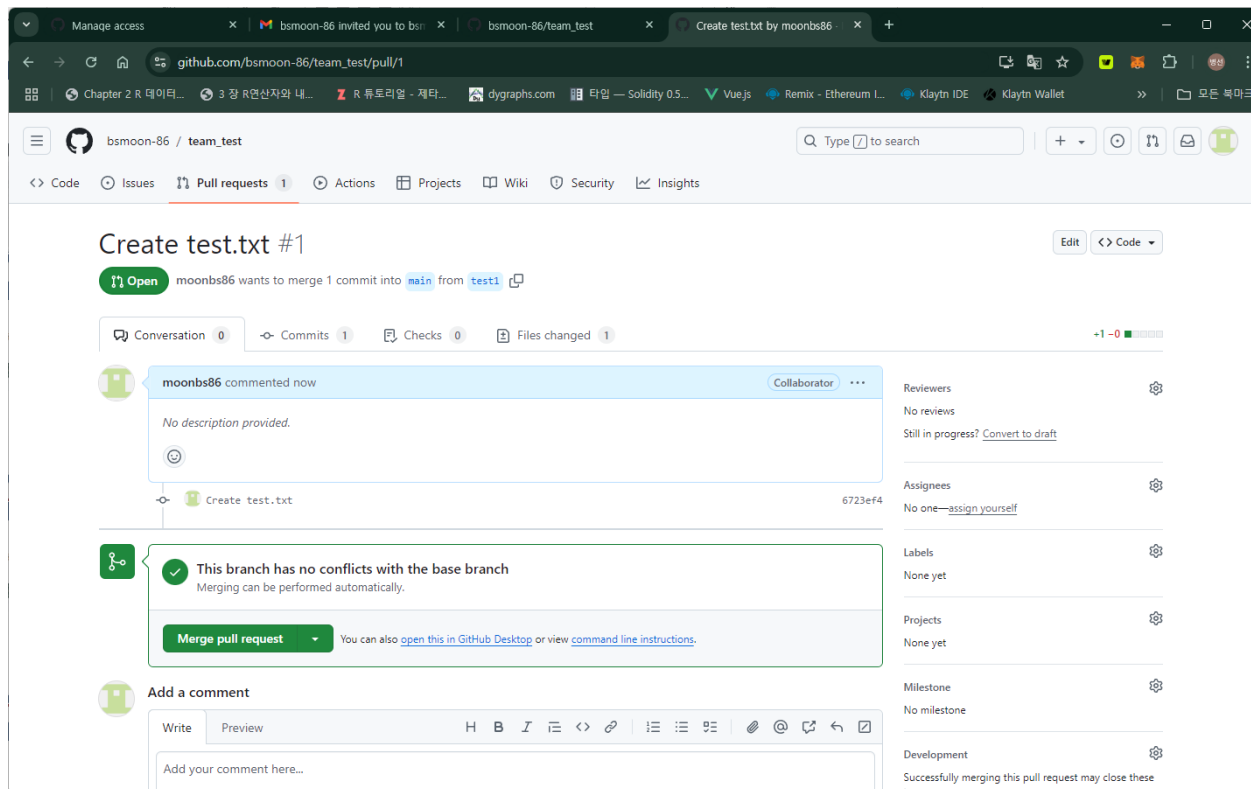
깃허브 원격저장소 협업하기

title과 description를 작성 한 뒤 Create pull request 버튼을 클릭
pull request란 main 브랜치에 merge를 요청하는 작업이다.



깃허브 원격저장소 협업하기

pull request 메뉴를 클릭하면 요청 리스트가 출력이 된다.
해당 리스트를 클릭하면 Merge pull request 버튼이 있으며 해당 버튼을 클릭하면 main branch에 결합이 된다.





**THANK
YOU**