

형상관리도구



Git & GitHub 기본 활용

first
coding

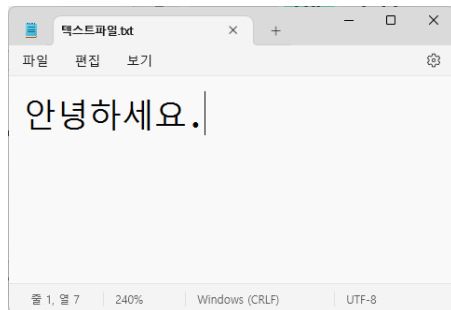
Git & GitHub

1. 버전관리시스템 (Version Control System)

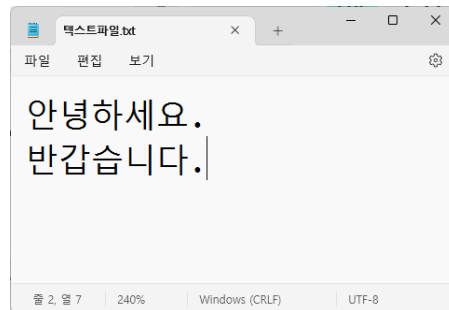
- 버전이란
- 버전관리
- 버전관리의 필요성
- 버전관리시스템 (Version Control System)
- Git

- 버전이란?

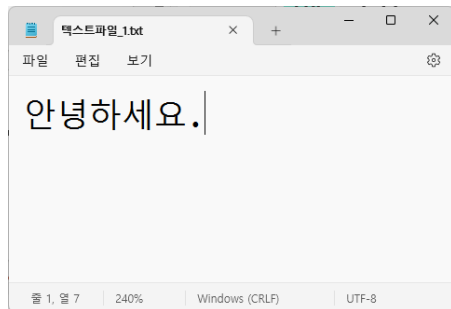
- 이전과 다른 “변화”를 구분하는 표시방법
- 수정이 끝난 상태를 가르킴



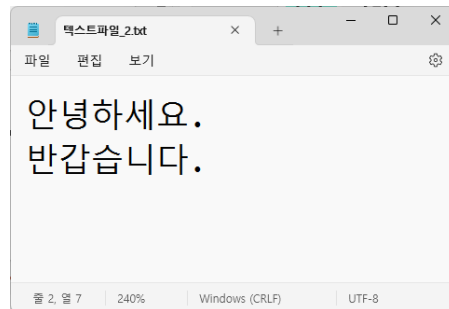
저장



변경 이전 내용을 확인할 수 없음



다른이름으로
저장



변경 이전 내용을 확인할 수 있음

- 버전관리

- 버전을 관리하는 것

- 동일한 정보 또는 파일에 대한 여러 버전을 관리하는 것

- 소프트웨어 개발 관점에서는 "소스코드를 관리"하는 것을 버전 관리라고 함.













index_old.html

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
</body>
</html>
```

index_new.html

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <h1>Git Test page</h1>
</body>
</html>
```

- 버전관리의 필요성

 index.html index_new.html index_new_1.html index_new_2.html index_new_3.html index_new_20230708.html index_new_20230708_수정완료.html index_new_20230708_수정중.html index_new_20230709_수정요청처리완료.html index_new_20230709_수정요청처리중.html index_new_20230710_완성.html index_old.html

지속적인 수정본 생성되었을 때의 문제점

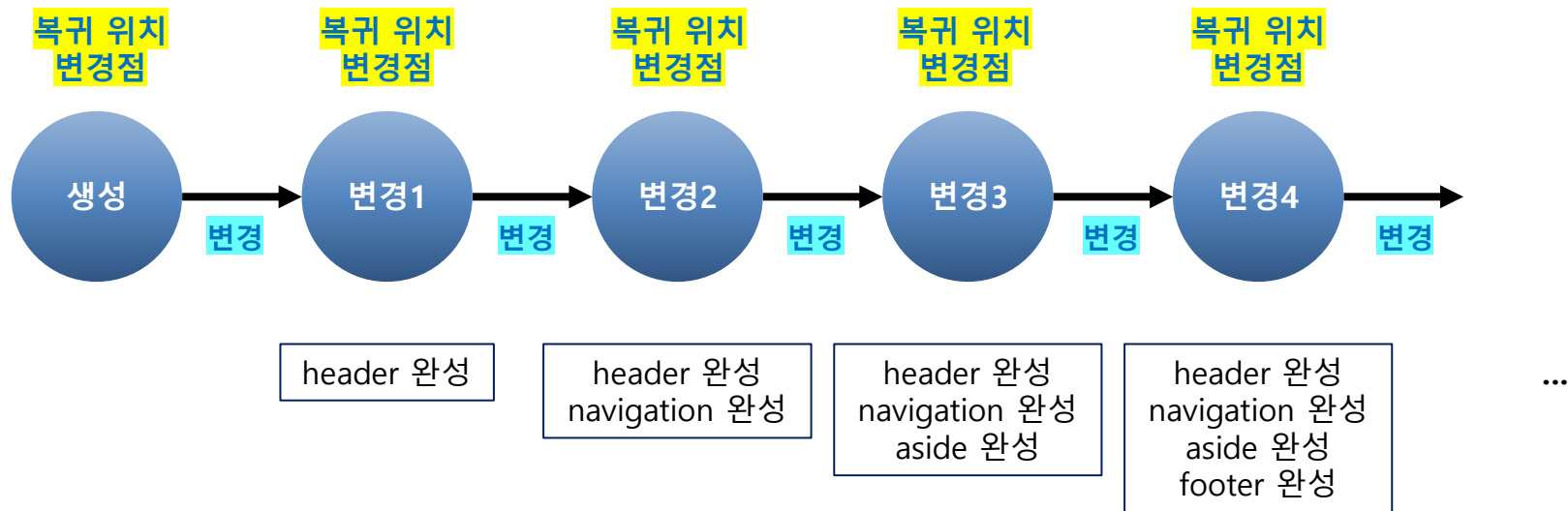
1. 파일에서 무엇이 변경되었는가?
2. 어떤 파일이 삭제되었는가?
3. 삭제된 파일을 복구할 수 있는가?
4. 계속해서 생성되는 파일을 보관할 수 있는가?
5. 파일의 수정 작업을 협업하는 경우 누가 무엇을 변경했는지 확인할 수 있는가?

- 버전관리의 필요성

- 프로그램 개발 중에는 많은 기능(요구사항)이 추가/변경 되고, 이에 따라 수많은 코드가 변경
- 기능 구현을 위한 코드를 작성하는 동안은 완성된 상태가 아니고, 코드 작성 완료 후 테스트를 통한 정상 동작을 확인하고 나면 안정된 상태가 됨
- 개발 과정에서 때에 따라 이전 상태로 돌아가 다시 시작할 수 있는 코드의 복귀(포인트) 지점이 필요함

01 버전관리시스템 (Version Control System)

- 버전관리의 필요성



01 버전관리시스템 (Version Control System)

- 버전관리시스템 (Version Control System)
 - 문서나 소스코드, 콘텐츠의 변경내용을 관리하고 추적하는 소프트웨어

버전관리 시스템에서 제공하는 기능

- 변경점 관리
- 버전 관리
- 백업 및 복구
- 협업

- 버전관리시스템 (Version Control System)

버전관리 시스템 종류



로컬 VCS

서버 없이
작업 컴퓨터 내에서
버전관리



중앙집중식
VCS

하나의 서버 존재
수정하고자 하는 파일을 받아
수정 후 서버로 업로드
서버가 버전 관리를 함.

대표 프로그램
Subversion



분산 VCS

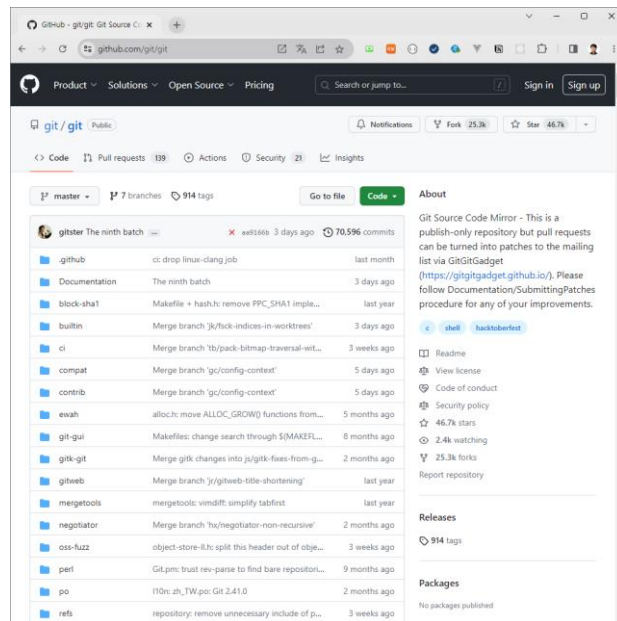
파일들을 저장하는 서버 존재
위해 프로젝트 전체를 받아
수정 후 변경 내용을 서버에 업로드
작업 컴퓨터와 서버
둘 다 버전 관리를 함.

대표 프로그램
Git

- Git

- 2005년에 리눅스 개발자인 리누스 베네딕트 토르발스(Linus Benedict Torvalds)가 개발
- 오픈소스인 분산형 버전 관리 시스템
 - GitHub 저장소에 코드를 공개하고 있음.
- 현재 버전 관리 시스템 중 가장 많이 사용되고 있음.
- 여러 IDE에 Git 클라이언트를 내장 또는 사용 가능
 - VSCode, Eclipse, IntelliJ 등

<https://github.com/git/git>



- Git으로 할 수 있는 것



버전 관리

특정 파일을 수정할 때마다
변경내용(언제, 어떤 것을, 누가)을
구체적으로 기록하는 버전 관리

네트워크나 인터넷이 연결되어 있
지 않은 상태에서도 로컬 컴퓨터의
소스 코드만으로 버전을 관리



백업

버전 관리하는 파일들을
원격저장소(온라인 저장소)에
복제(저장)을 할 수 있음

온라인 저장소를 제공하는
대표적인 서비스 → GitHub

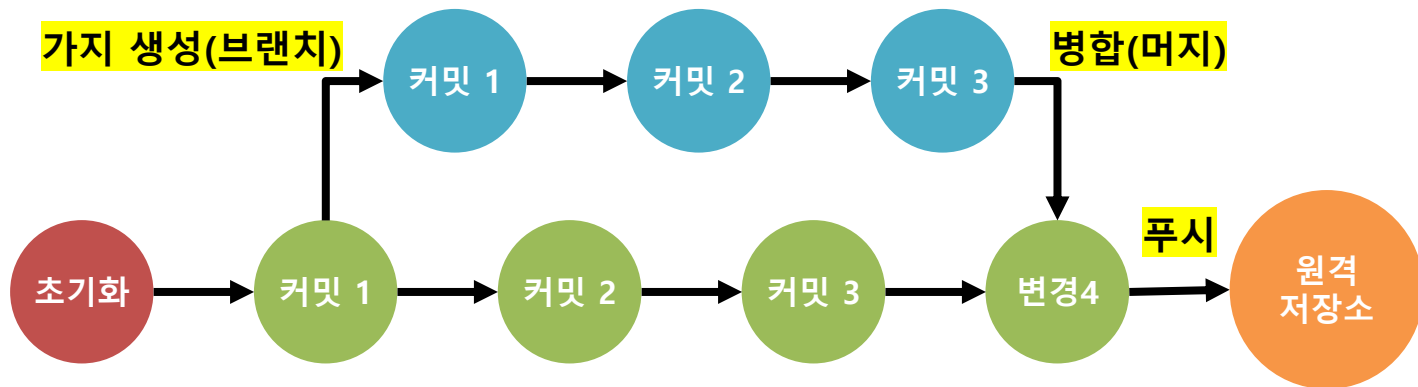


협업

온라인 저장소 서비스를 이용하면
프로그램 개발 참여자들과
관리 파일과 변경내용을 공유하며
협업이 가능

01 버전관리시스템 (Version Control System)

- Git의 관리 흐름



초기화 (init) → 작업 하는 파일들이 있는 폴더를 깃 저장소로 변경하는 것

커밋 (commit) → 변경된 내용의 이력을 기록하는 것

브랜치 (branch) → 메인 관리 파일들과 분리 격리된 코드 이력을 기록하는 것

병합 (merge) → 메인 변경 이력과 분리된 변경 이력을 통합하는 것

푸시 (push) → 작업 컴퓨터에 존재하는 로컬 저장소의 이력을 원격저장소 전송 및 공유

- Git GUI 프로그램

- 깃헙 데스크톱 (GitHub Desktop)

- GitHub 에서 제공하는 GUI 프로그램 (<https://desktop.github.com>)
 - 자주 쓰는 기본적인 기능 제공

- 소스트리

- Git GUI 중 대표적인 툴로 Atlassian 에서 제공하는 프로그램 (<https://www.sourcetreeapp.com>)
 - 기본 기능부터 고급 기능까지 사용할 수 있는 기능 제공

- Git 커맨드 라인 인터페이스 (Command Line Interface, CLI)
 - 터미널 창에 직접 명령을 입력해서 깃을 사용하는 방식
 - 리눅스의 기본 명령어를 사용 (기본 명령어 숙지)
 - 깃 명령(옵션)어를 사용 (git 명령어 숙지)
 - 반복할 일을 자동화하거나 서버 환경에서 깃을 사
 - github에 있는 코드를 내려 받고, 코드를 빌드하고, 빌드 된 프로그램 파일을 배포하는 등의 작업이 가능
 - 많은 개발자들이 커맨드 라인 인터페이스(CLI)로 깃을 사용

- 버전이란

변경사항을 구분하는 방법

- 버전관리

하나의 정보 또는 파일의 버전들을 관리하는 것, 소프트웨어 개발 관점에서는 소스코드를 관리하는 것

- 버전관리의 필요성

여러 상태의 파일들이 필요한 시점이 존재하고, 변경 내용의 자세한 정보가 필요

- 버전관리시스템 (Version Control System)

문서나 소스코드의 변경내용을 관리하고 추적하는 소프트웨어로 버전관리, 변경 점 관리, 협업 등의 기능을 제공

- Git

최근 가장 많이 사용되는 버전관리시스템

Git & GitHub

2. Git 설치

- Window에 Git 설치
- MacOS에 Git 설치
- Git 환경 설정

- Window에 Git 설치

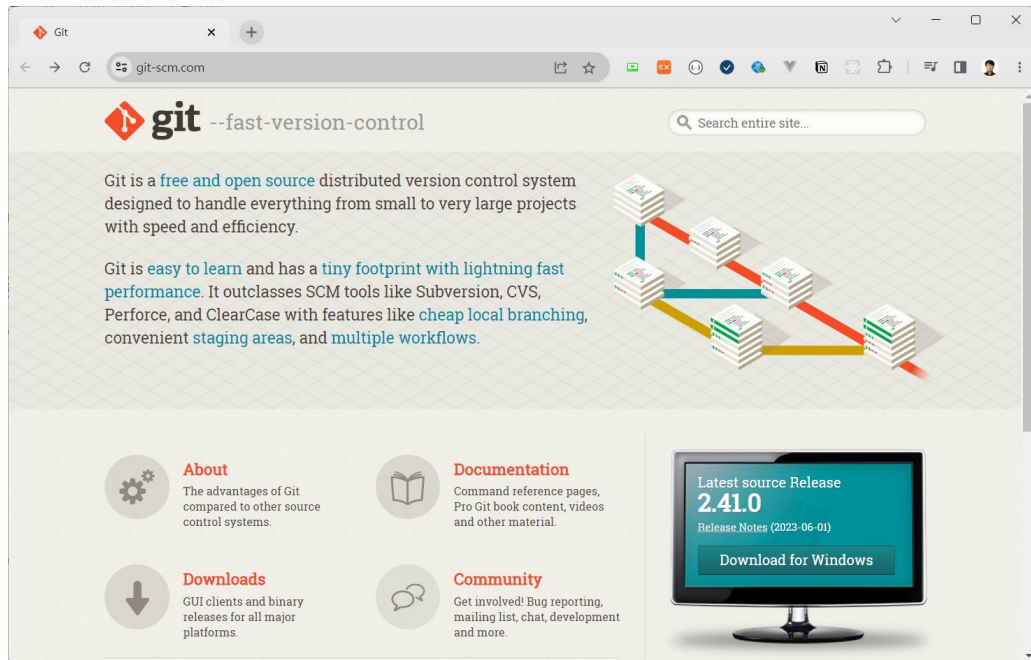
1. <https://git-scm.com> 접속 → 화면 오른쪽 아래에서 [Download 2.xx.x for Windows]를 클릭
2. 파일 목록이 있는 화면으로 이동 → 위에 있는 **[Click here to download]**를 클릭 → **다운로드 파일을 실행**
3. 화면마다 [Next]를 클릭해 설치할 경로와 구성 요소, 시작 메뉴에 표시할 메뉴 이름 등은 기본값 그대로 사용
4. Git 에서 사용할 기본 편집기를 선택
→ 기본값 **'Use Vim (the ubiquitous text editor) as Git's default editor'** 선택 → [Next]를 클릭
5. 기존 깃에서는 저장소를 만들 때 **master**라는 브랜치 이름을 사용, 최신 깃에서는 **main** 브랜치를 사용
2개의 옵션 중 **'Override the default branch name for new repositories'** 선택 → [Next]를 클릭
6. 커맨드 라인에서 어떤 방법으로 깃을 사용할지 선택
기본값 **'Git from the commandline and also from 3rd-party software'** 선택 → [Next]를 클릭
7. 보안 서버에 접속하는 방법을 선택
기본값 **'Use bundled OpenSSH'**가 선택 → [Next]를 클릭

- Window에 Git 설치

8. HTTPS처럼 보안이 추가된 연결에 어떻게 연결할 것인지 선택
기본값 'Use the OpenSSL Library' 선택 → [Next]를 클릭
9. 이어서 텍스트 파일에서 줄 끝부분을 어떻게 처리할 것인지 선택
기본값 'Checkout Windows-style, commit Unix-style line endings' 선택 → [Next]를 클릭
10. 터미널 에뮬레이터를 선택
기본값 'Use MinTTY' 선택 → [Next]를 클릭
11. 깃의 pull 명령을 어떻게 처리할 것인지 선택
기본값 'Default' 선택 → [Next]를 클릭
12. 기본값 'Git Credential Manager' 선택 → [Next]를 클릭
파일 시스템을 캐싱하도록 설정하면 버전 관리를 좀 더 빠르게 실행할 수 있음
13. 'Enable file system caching' 선택 → [Next]를 클릭
14. 제시된 옵션을 시험 삼아 사용해 볼 것인지 확인 → 아무것도 선택하지 말고 → [Install]을 클릭
15. [Finish] 클릭해 깃 설치를 완료

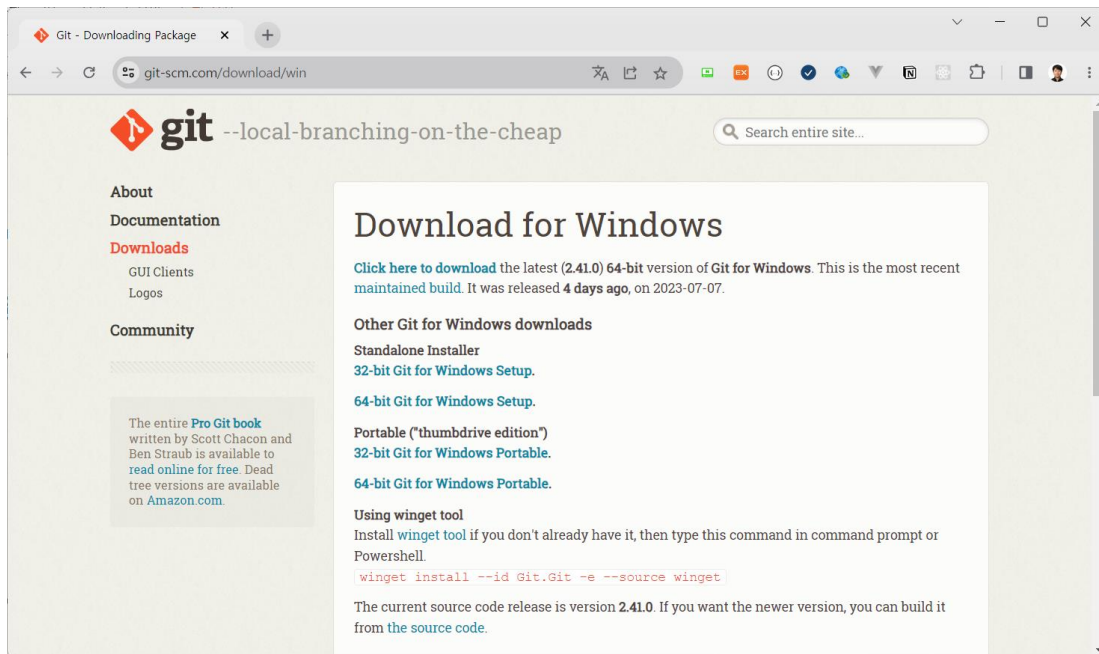
- Window에 Git 설치

1. <https://git-scm.com> 접속 → 화면 오른쪽 아래에서 [Download 2.xx.x for Windows]를 클릭



- Window에 Git 설치

2. 파일 목록이 있는 화면으로 이동 → 위에 있는 **[Click here to download]**를 클릭 → **다운로드 파일을 실행**

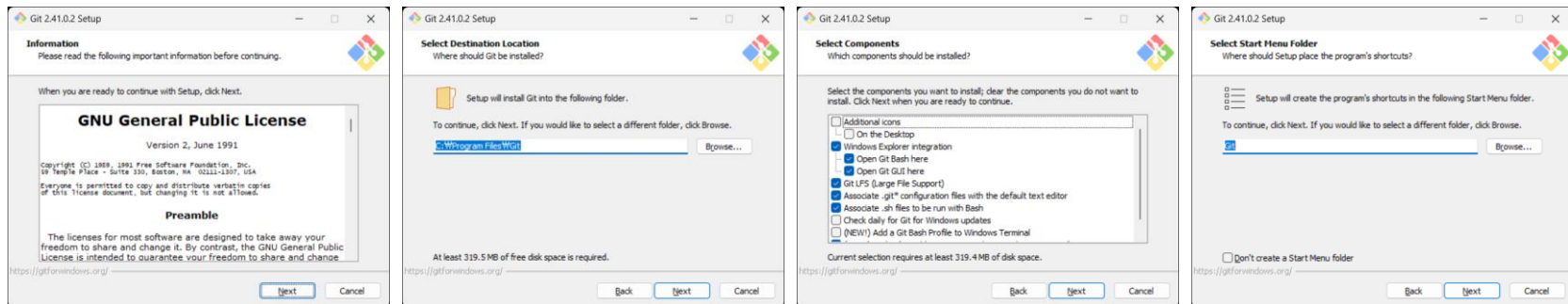


Git-2.41.0.2-64-bit.exe

58.4MB • 완료

- Window에 Git 설치

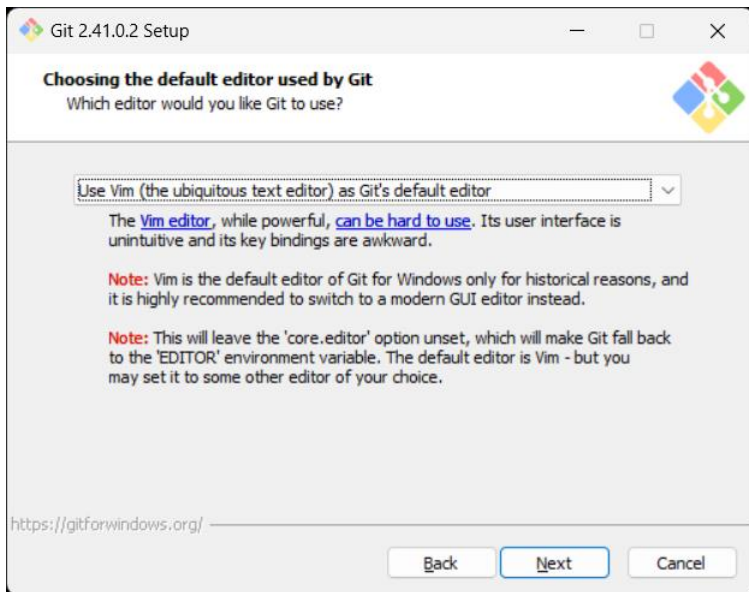
3. 화면마다 [Next]를 클릭해 설치할 경로와 구성 요소, 시작 메뉴에 표시할 메뉴 이름 등은 기본값 그대로 사용



- Window에 Git 설치

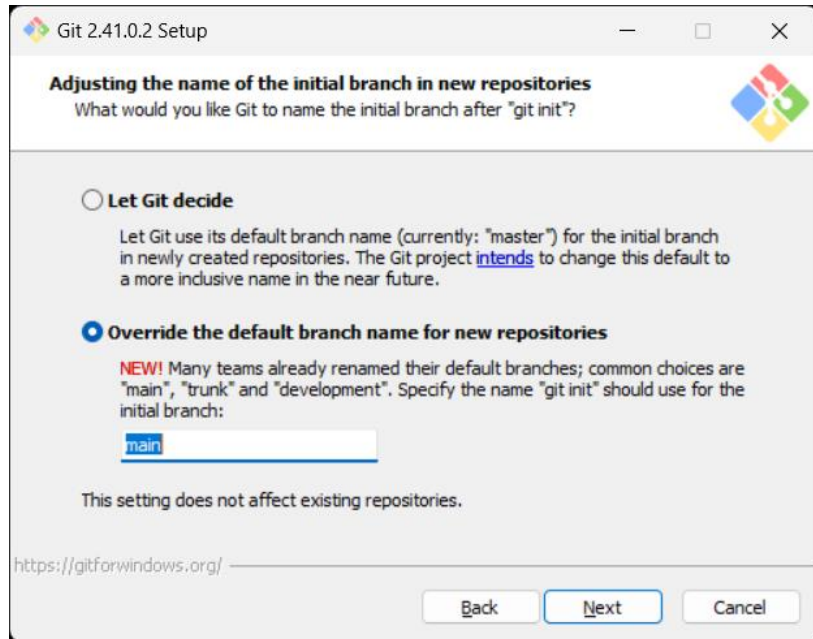
- 4. Git 에서 사용할 기본 편집기를 선택

→ 기본값 'Use Vim (the ubiquitous text editor) as Git's default editor' 선택 → [Next]를 클릭



- Window에 Git 설치

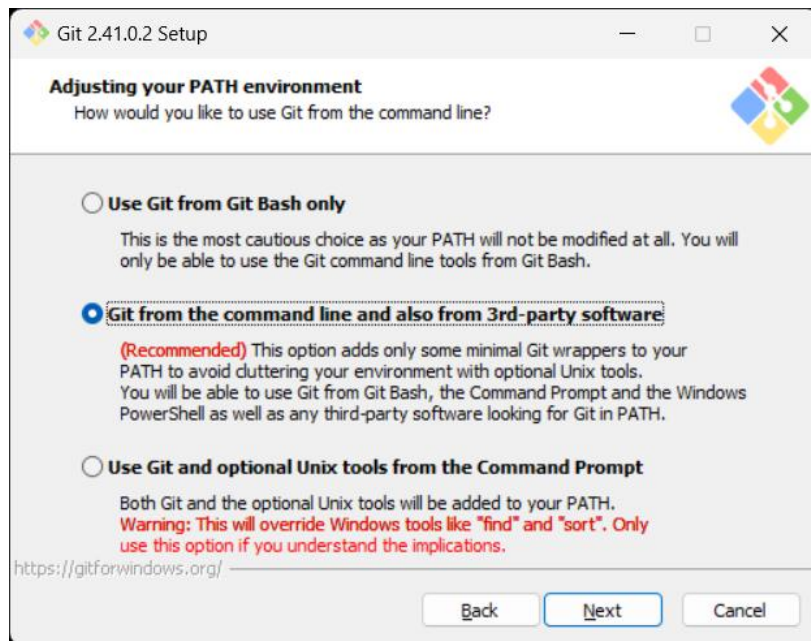
5. 기존 깃에서는 저장소를 만들 때 **master**라는 브랜치 이름을 사용, 최신 깃에서는 **main** 브랜치를 사용
- 2개의 옵션 중 '**Override the default branch name for new repositories**' 선택 → [Next]를 클릭



- Window에 Git 설치

6. 커맨드 라인에서 어떤 방법으로 깃을 사용할지 선택

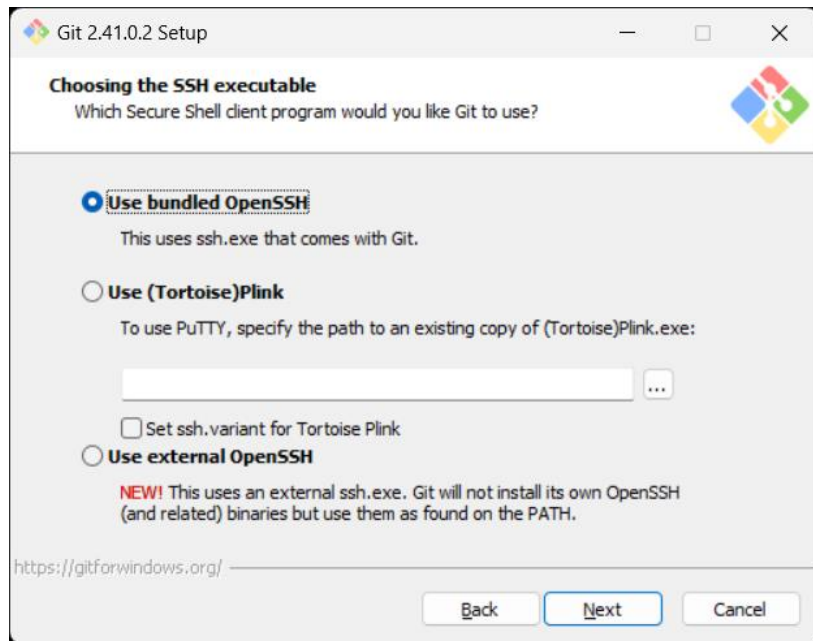
기본값 'Git from the commandline and also from 3rd-party software' 선택 → [Next]를 클릭



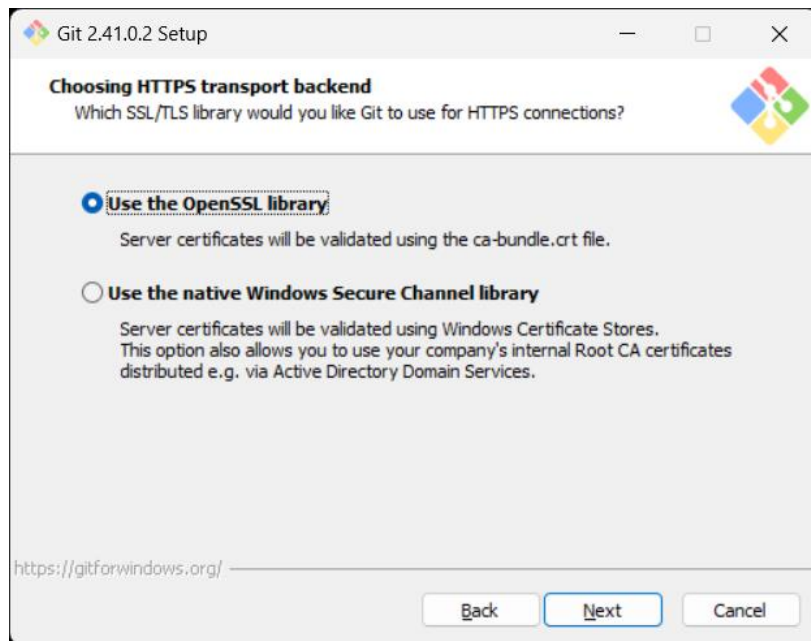
- Window에 Git 설치

7. 보안 서버에 접속하는 방법을 선택

기본값 **'Use bundled OpenSSH'**가 선택 → [Next]를 클릭



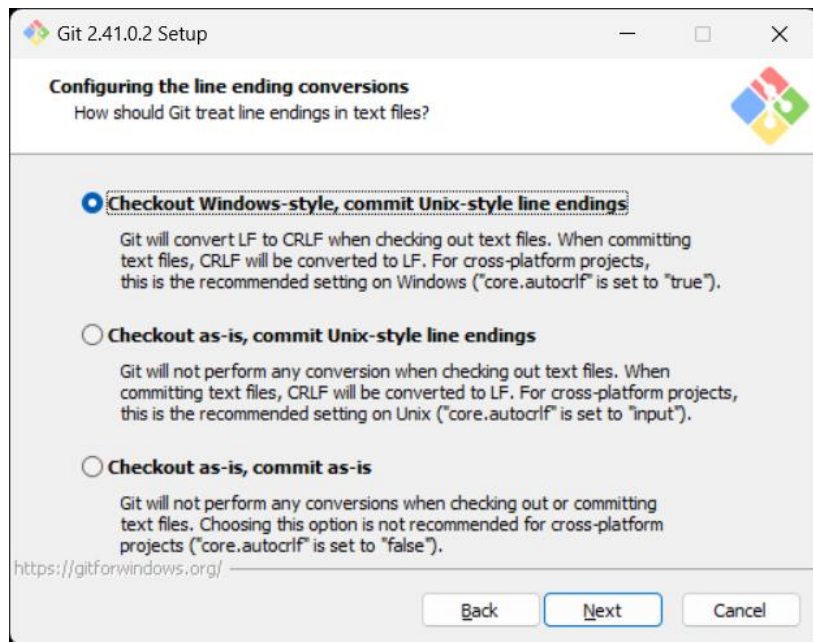
- Window에 Git 설치
 - 8. HTTPS처럼 보안이 추가된 연결에 어떻게 연결할 것인지 선택
기본값 'Use the OpenSSL Library' 선택 → [Next]를 클릭



- Window에 Git 설치

9. 이어서 텍스트 파일에서 줄 끝부분을 어떻게 처리할 것인지 선택

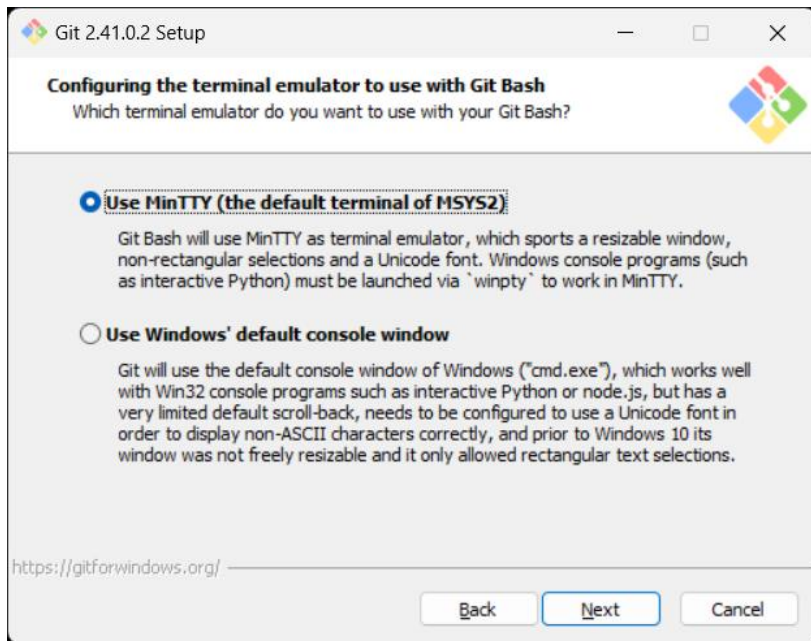
기본값 'Checkout Windows-style, commit Unix-style line endings' 선택 → [Next]를 클릭



- Window에 Git 설치

- 10. 터미널 에뮬레이터를 선택

기본값 'Use MinTTY' 선택 → [Next]를 클릭



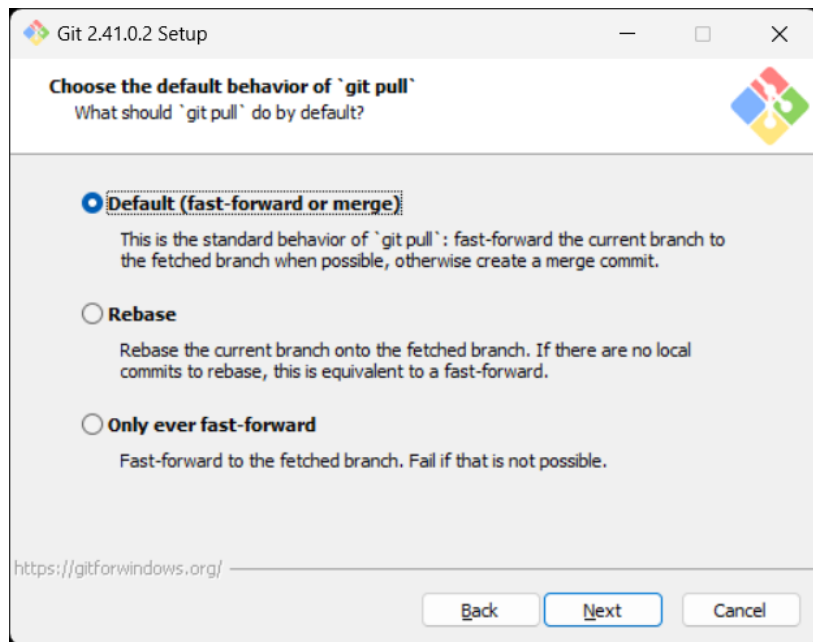
터미널 에뮬레이터

- 가상의 단말기로 tty라고도 함.
- 사용자가 텍스트 단말과 명령 줄 인터페이스(CLI), 텍스트 사용자 인터페이스 응용 프로그램과 같은 모든 응용 프로그램에 접근할 수 있음
- Git에서는 터미널 에뮬레이터를 통해 Git Bash (리눅스 커맨드) 기능을 사용
- 맥 운영체제에는 리눅스 환경이 구축되어 있기 때문에 기존 terminal을 사용

- Window에 Git 설치

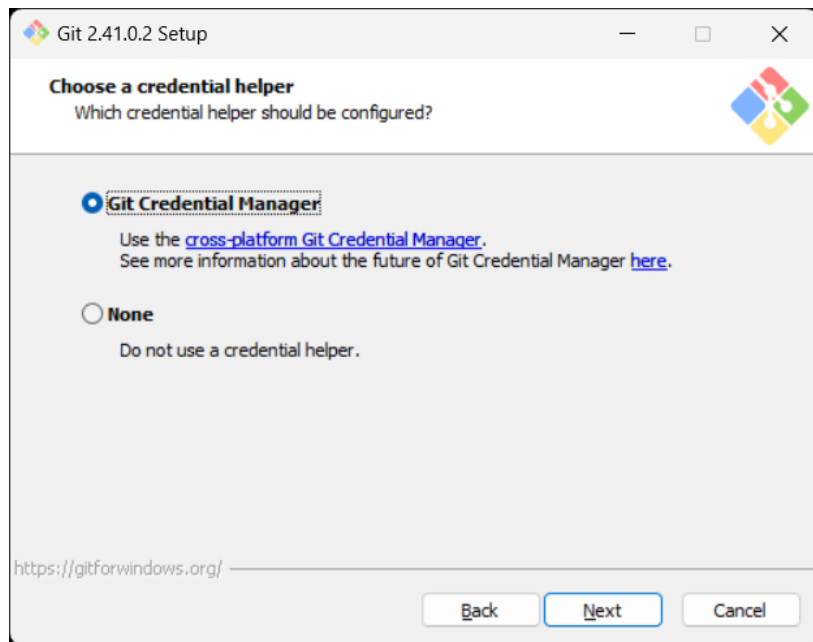
11. 깃의 pull 명령을 어떻게 처리할 것인지 선택

기본값 'Default' 선택 → [Next]를 클릭



- Window에 Git 설치

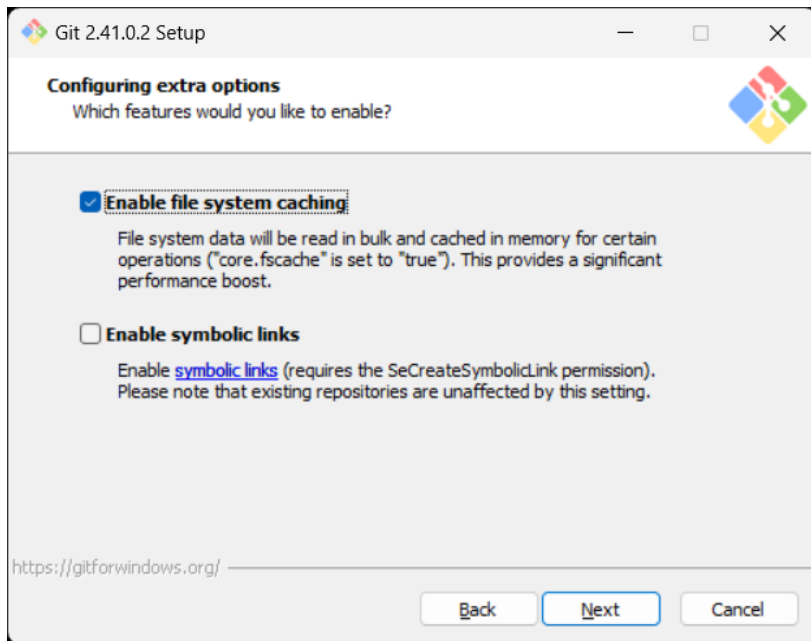
12. 기본값 'Git Credential Manager' 선택 → [Next]를 클릭



- Window에 Git 설치

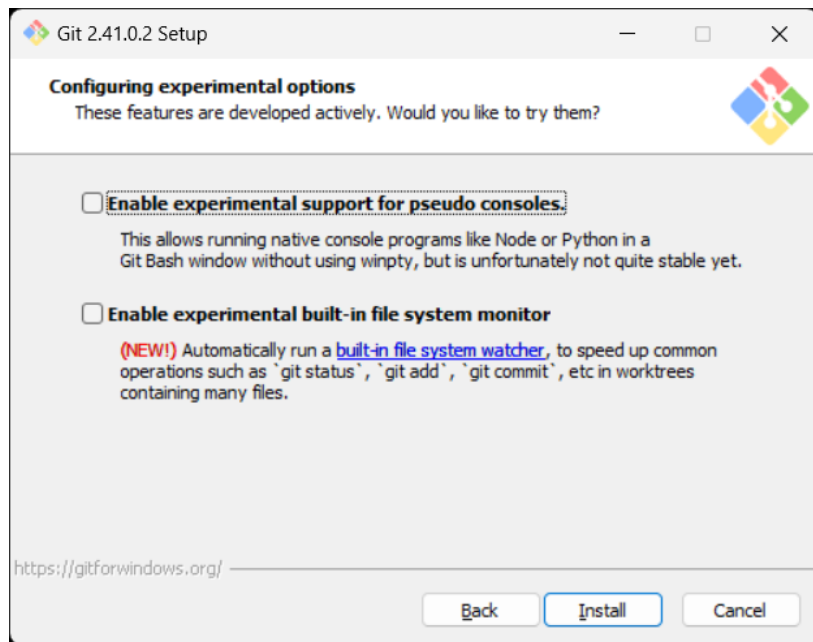
13. 'Enable file system caching' 선택 → [Next]를 클릭

파일 시스템을 캐싱 하도록 설정하면 버전 관리를 좀 더 빠르게 실행할 수 있음



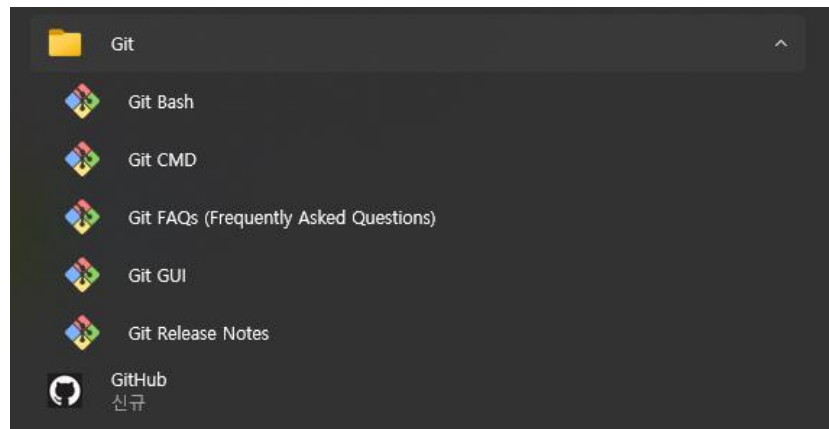
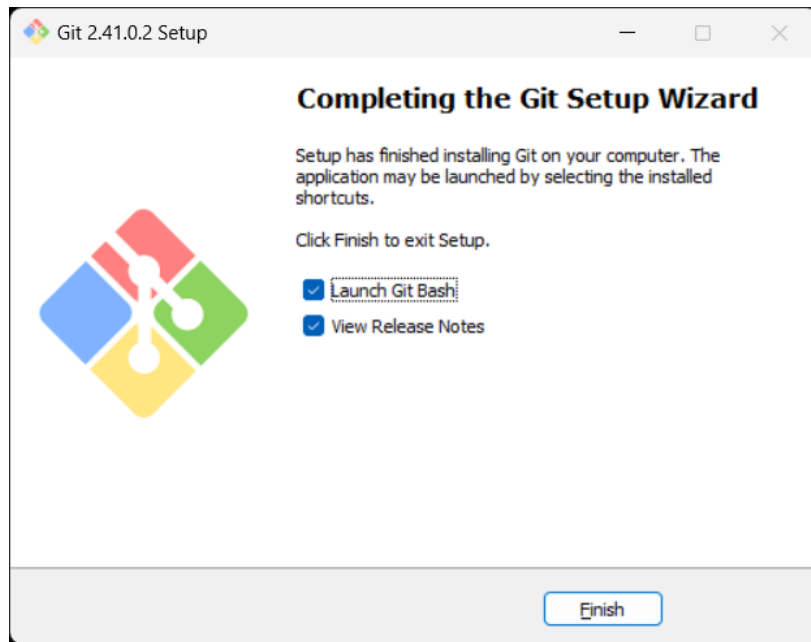
- Window에 Git 설치

14. 제시된 옵션을 시험 삼아 사용해 볼 것인지 확인 → 아무것도 선택하지 말고 → [Install]을 클릭



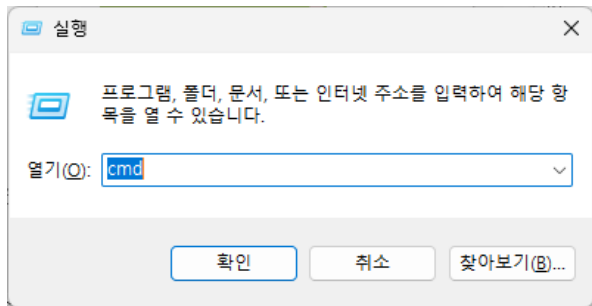
- Window에 Git 설치

15. [Finish] 클릭해 깃 설치를 완료



- Window에 Git 설치 확인

커맨드 환경 실행 (**Windows key + R**) → git --version 명령으로 깃 설치 확인



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.22621.1928]
(c) Microsoft Corporation. All rights reserved.

C:\Users\jin>git --version
git version 2.41.0.windows.2

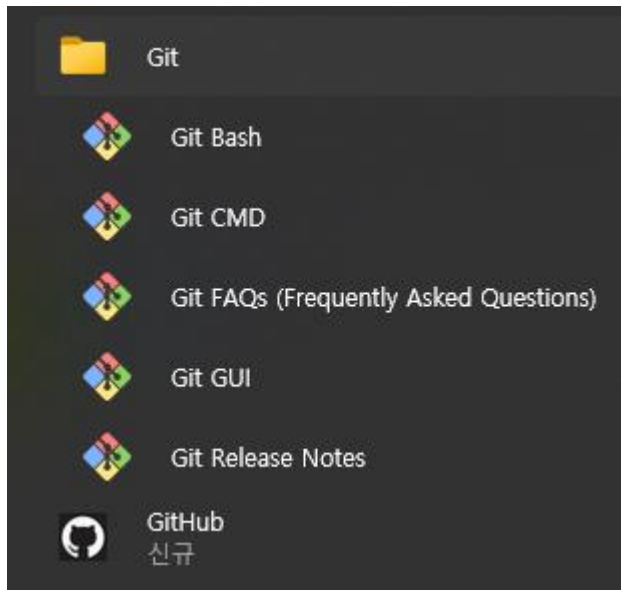
C:\Users\jin>git
usage: git [-v | --version] [-h | --help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<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
```

- Windows에 Git 설치 확인

커맨드 환경 실행 (Windows key + R) → git -version 명령으로 깃 설치 확인



```
MINGW64/c/Users/jin
jin@JIN-Desktop MINGW64 ~
$ git --version
git version 2.41.0.windows.2

jin@JIN-Desktop 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] [--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
```

- MacOS에 Git 설치

1. <https://brew.sh> 홈브류 사이트로 이동 → 언어를 '한국어'로 선택

'Homebrew 설치하기' 아래에 있는 설치 명령을 복사

오른쪽에 있는  를 클릭하면 명령을 간단히 복사할 수 있음

2. 맥에서 터미널을 열고 **Command + V** 를 눌러 복사한 명령을 붙여 넣고 **Enter**

3. 맥에서 사용하는 비밀번호를 입력하고 나면 홈브류가 설치되기 시작

4. 맥 터미널에서 아래와 같이 입력한 후 {{ Enter }}

```
$ brew install git
```

Git 이 설치되며 설치가 끝나면 **\$** 표시가 나타남

5. \$ 옆에 아래와 같이 입력한 후 Enter


```
$ git -version
```

```
$ git
```

- MacOS에 Git 설치

1. <https://brew.sh> 홈브류 사이트로 이동 → 언어를 '한국어'로 선택

'Homebrew 설치하기' 아래에 있는 설치 명령을 복사

오른쪽에 있는  를 클릭하면 명령을 간단히 복사할 수 있음

터미널을 실행하고 복사한 코드를 실행

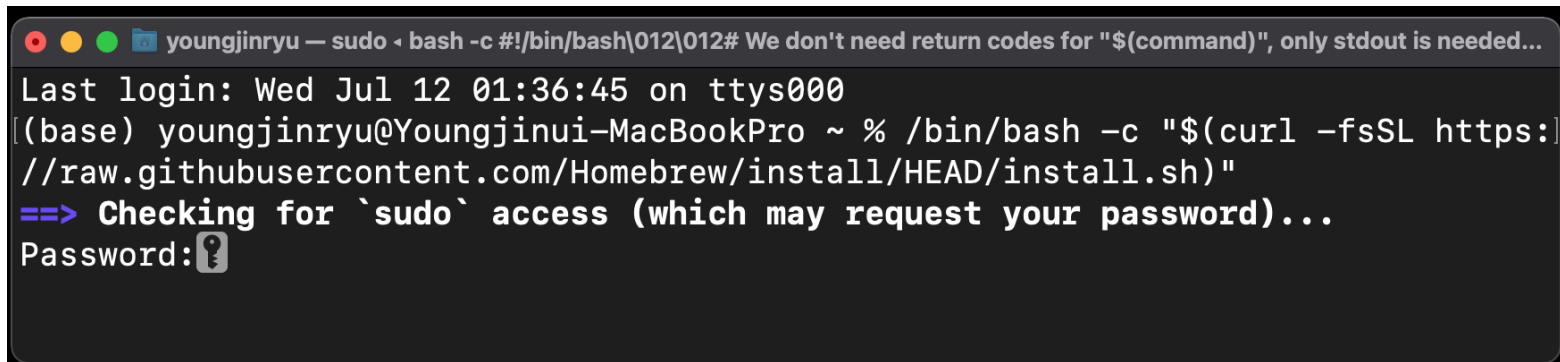
```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

- MacOS에 Git 설치

1. 맥에서 터미널을 열고 **Command + V** 를 눌러 복사한 명령을 붙여 넣고 **Enter**

아래 코드를 순서대로 입력하고 실행

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```



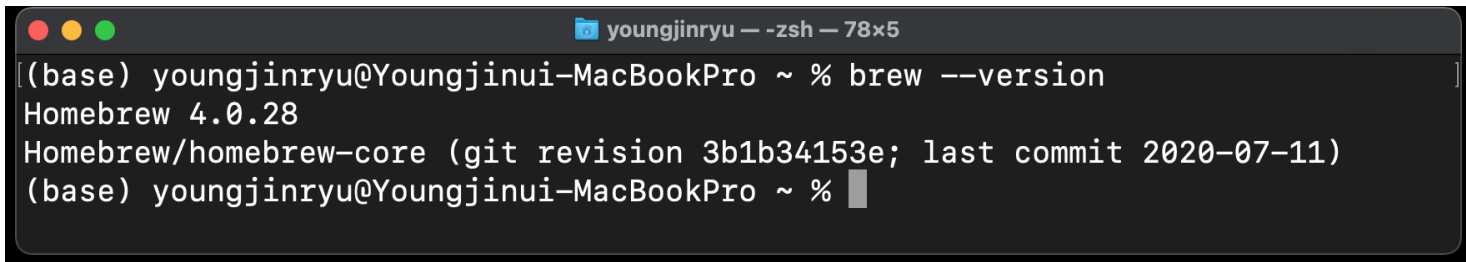
```
youngjinryu — sudo - bash -c #!/bin/bash\012\012# We don't need return codes for "$(command)", only stdout is needed...
Last login: Wed Jul 12 01:36:45 on ttys000
(base) youngjinryu@Youngjinui-MacBookPro ~ % /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
==> Checking for `sudo` access (which may request your password)...
Password: [REDACTED]
```

- MacOS에 Git 설치

1. 맥에서 사용하는 비밀번호를 입력하고 나면 홈브류가 설치되기 시작

설치완료 후 아래 명령으로 버전 확인

brew --version

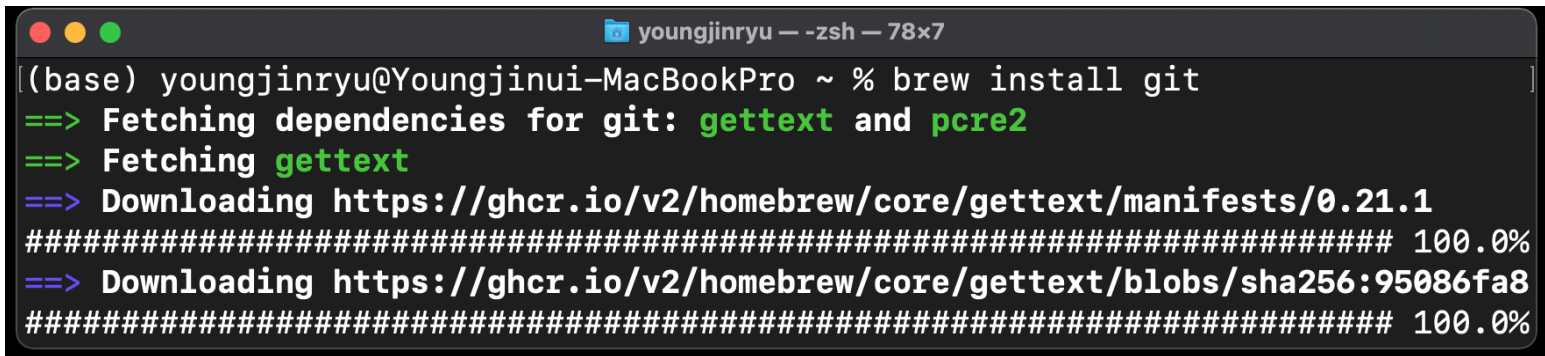


```
youngjinryu — zsh — 78x5
(base) youngjinryu@Youngjinui-MacBookPro ~ % brew --version
Homebrew 4.0.28
Homebrew/homebrew-core (git revision 3b1b34153e; last commit 2020-07-11)
(base) youngjinryu@Youngjinui-MacBookPro ~ %
```

- MacOS에 Git 설치

1. 맥 터미널에서 아래와 같이 입력한 후 {{ Enter }}

brew install git

A screenshot of a macOS terminal window. The title bar shows 'youngjinryu - zsh - 78x7'. The terminal content shows the command '[(base) youngjinryu@Youngjinui-MacBookPro ~ % brew install git]' and its output. The output indicates that dependencies 'gettext' and 'pcre2' are being fetched, followed by the download of 'gettext' from a Homebrew tap, which completes at 100.0%.

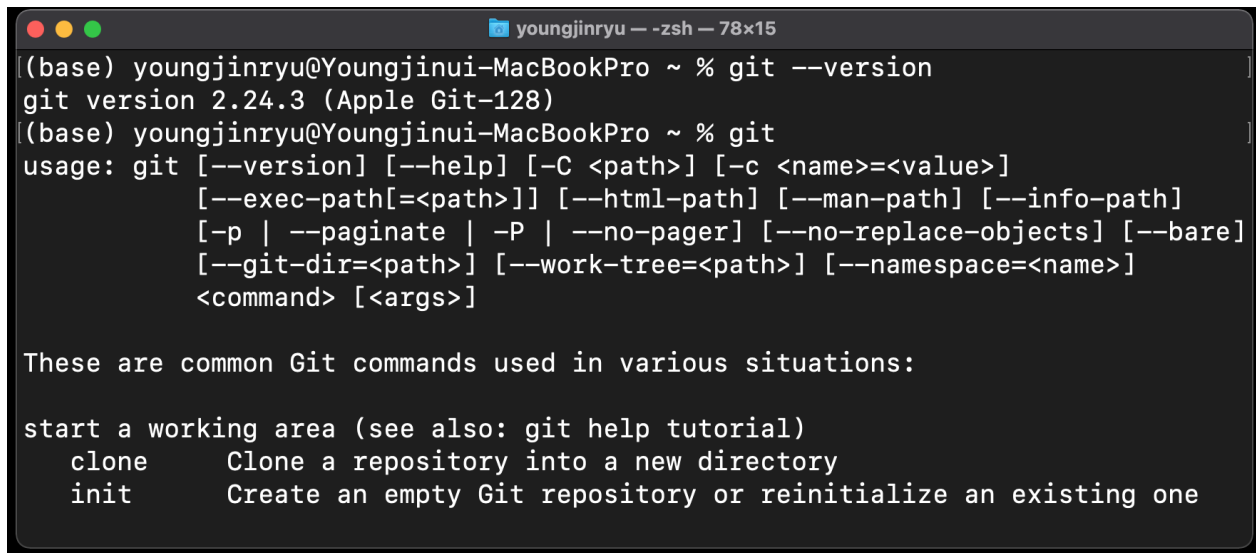
```
youngjinryu@Youngjinui-MacBookPro ~ % brew install git
==> Fetching dependencies for git: gettext and pcre2
==> Fetching gettext
==> Downloading https://ghcr.io/v2/homebrew/core/gettext/manifests/0.21.1
##### 100.0%
==> Downloading https://ghcr.io/v2/homebrew/core/gettext/blobs/sha256:95086fa8
##### 100.0%
```


- MacOS에 Git 설치 확인

1. \$ 옆에 아래와 같이 입력한 후 Enter

`git --version`

`git`



```
youngjinryu ~ % git --version
git version 2.24.3 (Apple Git-128)
youngjinryu ~ % git
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
        [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
        [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
        [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
        <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
```

- Git 환경 설정

- Git을 사용하기 전에 먼저 사용자 정보를 입력해야 함
깃은 버전을 저장할 때마다 그 버전을 만든 사용자 정보도 함께 저장
→ 각각의 버전을 누가 언제 만들었는지 쉽게 파악할 수 있음
- 환경 설정의 명령은 운영체제와 상관없이 리눅스 방식의 명령을 사용
windows → **git Bash** , MacOS → **터미널 이용**

git config 명령을 사용 : 아래와 같은 명령으로 사용자의 이름과 이메일 주소를 저장

```
$ git config --global user.name "사용자 이름"
```

```
$ git config --global user.email "사용자 이메일"
```

--global 옵션을 사용하면 현재 컴퓨터에 있는 모든 저장소에서 같은 사용자 정보를 사용하도록 설정됨(**권장**)

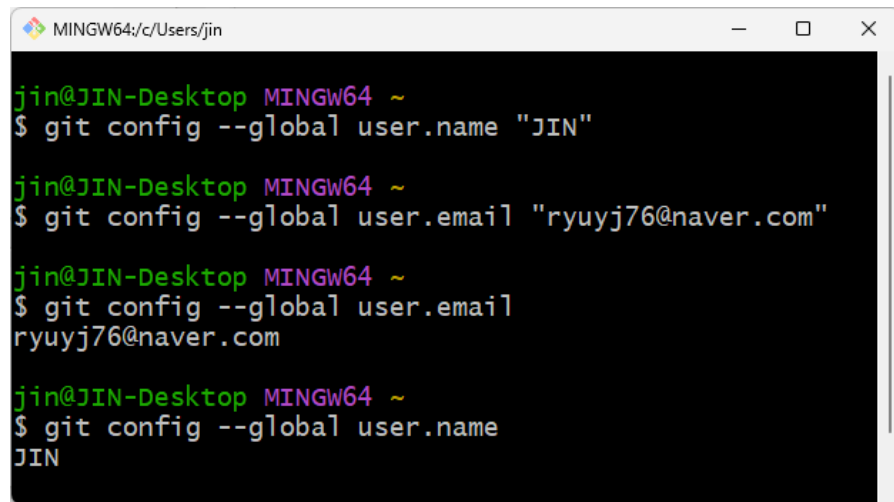
- Git 환경 설정

```
$ git config --global user.name "JIN"
```

```
$ git config --global user.email "test@gmail.com"
```

```
$ git config --global user.email
```

```
$ git config --global user.name
```



```
MINGW64/c:/Users/jin
jin@JIN-Desktop MINGW64 ~
$ git config --global user.name "JIN"

jin@JIN-Desktop MINGW64 ~
$ git config --global user.email "ryuyj76@naver.com"

jin@JIN-Desktop MINGW64 ~
$ git config --global user.email
ryuyj76@naver.com

jin@JIN-Desktop MINGW64 ~
$ git config --global user.name
JIN
```

- Window에 Git 설치

공식사이트에서 설치파일을 다운받아 설치

- MacOS에 Git 설치

홈브류를 이용해서 설치

- Git 환경 설정

CLI 기반으로 변경사항 관리를 위한 사용자 정보 설정,

windows → **git Bash** , MacOS → **터미널 이용**

Git & GitHub

3. 리눅스 기본 명령어

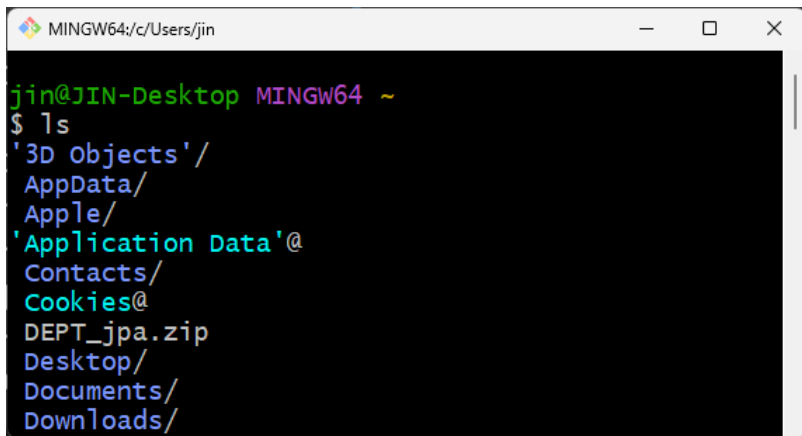
- 현재 디렉터리 확인
- 현재 디렉터리의 파일과 하위 디렉터리 확인
- 터미널 창 내용 지우기
- 디렉터리 이동
- 디렉터리 생성 및 삭제
- 터미널 종료
- 텍스트 문서 생성
- 텍스트 문서 확인

- 현재 디렉터리 확인
 - Git Bash를 실행하고 명령줄을 보면 "~" 표시가 있음
 - 현재 위치가 홈 디렉터리임을 의미
 - `$ pwd`
 - 현재 위치의 경로가 출력됨



```
MINGW64:/c/Users/jin
jin@JIN-Desktop MINGW64 ~
$ pwd
/c/Users/jin
```

- 현재 디렉터리의 파일과 하위 디렉터리 확인
 - `$ ls`
 - 현재 위치의 하위 디렉터리와 파일 이름이 출력

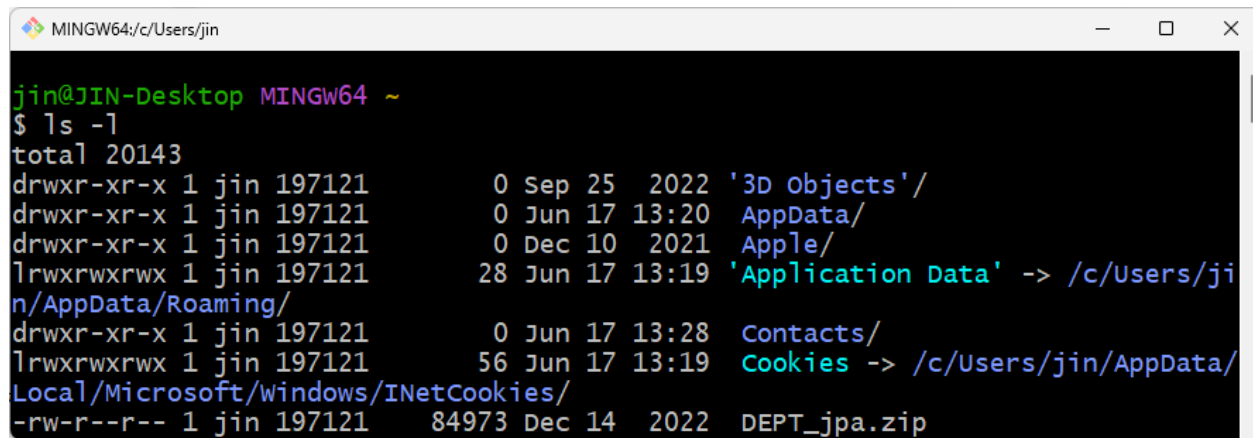


```
MINGW64; c:/Users/jin
jin@JIN-Desktop MINGW64 ~
$ ls
'3D objects'/
AppData/
Apple/
'Application Data'@
Contacts/
Cookies@
DEPT_jpa.zip
Desktop/
Documents/
Downloads/
```

- 현재 디렉터리의 파일과 하위 디렉터리 확인

- `$ ls -l`

→ -l 옵션을 사용하면 현재 위치의 하위 디렉터리와 파일의 상세 정보까지 모두 출력



```
MINGW64; c:/Users/jin
jin@JIN-Desktop MINGW64 ~
$ ls -l
total 20143
drwxr-xr-x 1 jin 197121      0 Sep 25  2022 '3D Objects'/
drwxr-xr-x 1 jin 197121      0 Jun 17 13:20  AppData/
drwxr-xr-x 1 jin 197121      0 Dec 10  2021  Apple/
lrwxrwxrwx 1 jin 197121    28 Jun 17 13:19 'Application Data' -> /c/Users/ji
n/AppData/Roaming/
drwxr-xr-x 1 jin 197121      0 Jun 17 13:28  Contacts/
lrwxrwxrwx 1 jin 197121    56 Jun 17 13:19  Cookies -> /c/Users/jin/AppData/
Local/Microsoft/Windows/INetCookies/
-rw-r--r-- 1 jin 197121  84973 Dec 14  2022  DEPT_jpa.zip
```

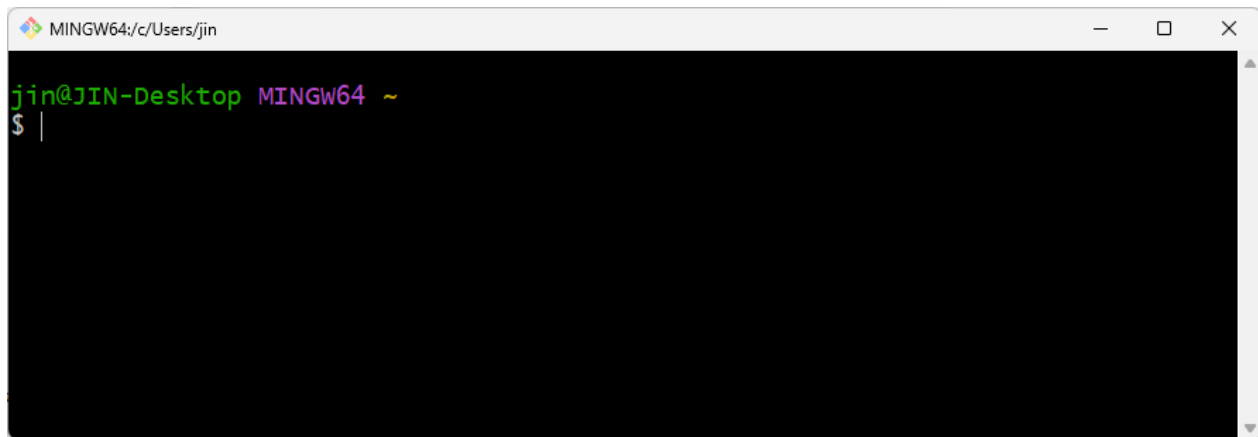

- 현재 디렉터리의 파일과 하위 디렉터리 확인
 - ls 명령 옵션
 - ls 명령 다음에 붙임표(-)을 붙이고 옵션을 나타내는 문자를 작성
 - 이때 -al처럼 옵션을 2개 이상 사용할 수 있음

옵션	설명
-a	숨긴 파일이나 디렉터리도 함께 표시합니다.
-l	파일이나 디렉터리의 상세 정보를 함께 표시합니다.
-r	파일의 정렬 순서를 거꾸로 표시합니다.
-t	파일 작성 시간순으로 (내림차순) 표시합니다.

- 터미널 창 내용 지우기

- `$ clear`

화면에 명령어와 결과 출력이 가득 차서 결과를 쉽게 확인하기 어려울 때
clear 명령을 사용하면 터미널 화면의 출력된 내용을 비울 수 있음



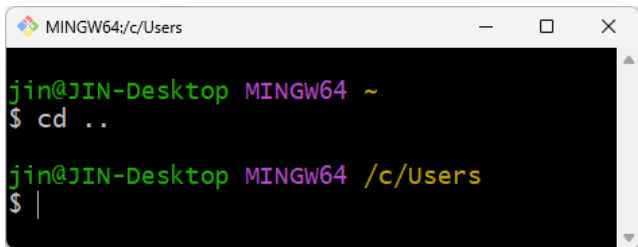
A screenshot of a terminal window titled "MINGW64; c/Users/jin". The terminal shows the prompt "jin@JIN-Desktop MINGW64 ~" followed by a dollar sign "\$" and a vertical cursor bar. The rest of the terminal area is empty, indicating that the output has been cleared.

- 디렉터리 이동

- 디렉터리 사이를 이동할 때는 'cd'라는 명령을 사용

- `$ cd ..`

상위 디렉터리로 이동



```
MINGW64/c/Users
jin@JIN-Desktop MINGW64 ~
$ cd ..
jin@JIN-Desktop MINGW64 /c/Users
$ |
```

c/Users라고 나타나는데 이는 c/Users/사용자 아이디에서 한 단계 위로 올라간 경로임을 나타냄



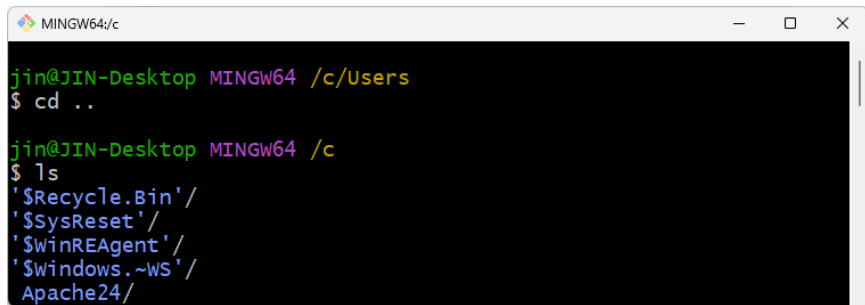
```
MINGW64/c/Users
jin@JIN-Desktop MINGW64 /c/Users
$ ls
'All Users'@   Public/
Default/      desktop.ini
'Default User'@  jin/
```

- 디렉터리 이동

- 디렉터리 사이를 이동할 때는 'cd'라는 명령을 사용

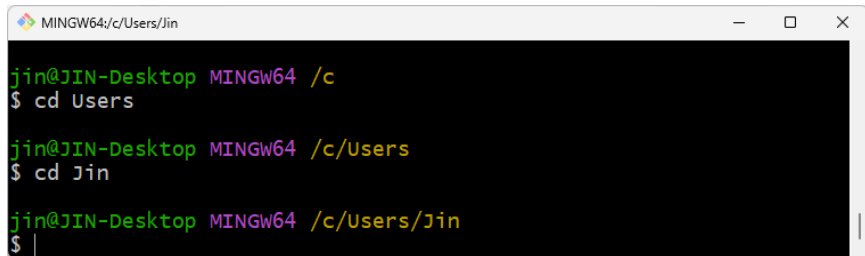
- `$ cd Users`

하위 디렉터리로 이동



```
MINGW64:/c
jin@JIN-Desktop MINGW64 /c/Users
$ cd ..

jin@JIN-Desktop MINGW64 /c
$ ls
'$Recycle.Bin'/
'$SysReset'/
'$WinREAgent'/
'$Windows.~WS'/
Apache24/
```



```
MINGW64:/c/Users/Jin
jin@JIN-Desktop MINGW64 /c
$ cd Users

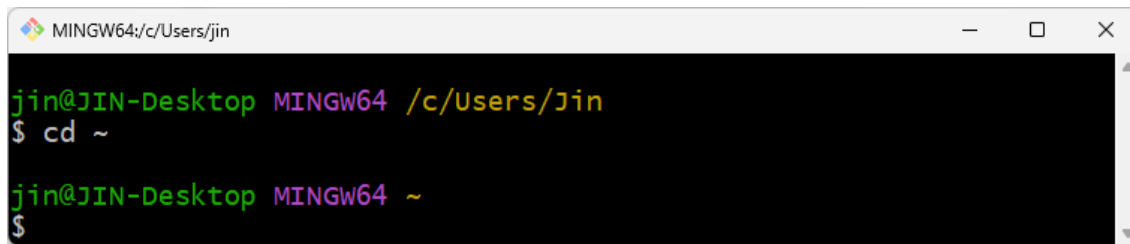
jin@JIN-Desktop MINGW64 /c/Users
$ cd Jin

jin@JIN-Desktop MINGW64 /c/Users/Jin
$ |
```

- 디렉터리 이동
 - 홈 디렉터리로 이동

- `$ cd ~`

하위 디렉터리로 이동



```
MINGW64:/c/Users/jin
jin@JIN-Desktop MINGW64 /c/Users/Jin
$ cd ~

jin@JIN-Desktop MINGW64 ~
$
```

- 디렉터리 이동

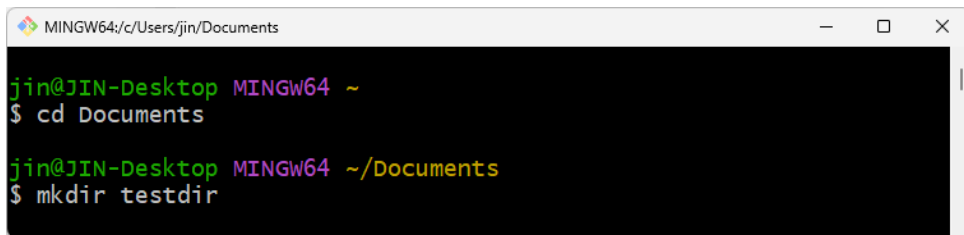
- 리눅스에서 디렉터리를 나타내는 기호

- 리눅스에서는 현재 위치나 파일 경로를 나타낼 때 몇 가지 약속된 기호를 사용

기호	설명
~	현재 접속 중인 사용자 디렉터를 가리킵니다. 'c/Users/사용자 아이디'가 사용자 디렉터리입니다. 터미널 창에서 \$ 기호 윗줄에 있는 ~가 바로 사용자 디렉터를 가리킵니다. 사용자 아이디는 5글자까지만 나타납니다.
.	현재 사용자가 작업 중인 디렉터리입니다.
..	현재 디렉터리의 상위 디렉터리입니다.

- 디렉터리 생성 및 삭제

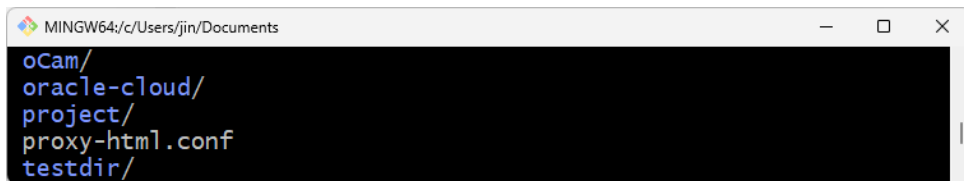
- 현재 디렉터리 안에 하위 디렉터를 생성할 때는 'mkdir' 명령을 사용
홈 디렉터리 안에 있는 Documents 디렉터리에 testdir 하위 디렉터를 생성



```
MINGW64/c/Users/jin/Documents

jin@JIN-Desktop MINGW64 ~
$ cd Documents

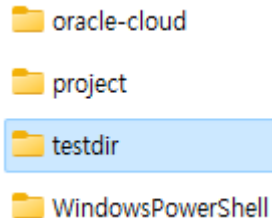
jin@JIN-Desktop MINGW64 ~/Documents
$ mkdir testdir
```



```
MINGW64/c/Users/jin/Documents

oCam/
oracle-cloud/
project/
proxy-htm1.conf
testdir/
```

탐색기에서 확인
C:\Users\W**사용자계정이름**\Documents



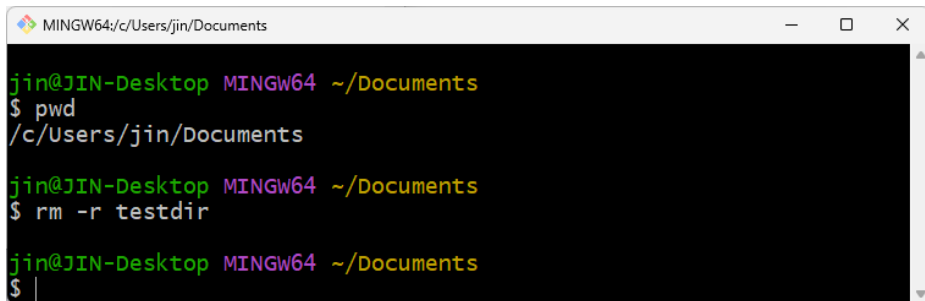
- 디렉터리 생성 및 삭제

- 디렉터리를 삭제할 때는 'rm'이라는 명령을 사용

Documents 디렉터리 위치에서 testdir 하위 디렉터리를 삭제

-r 옵션을 붙이면 디렉터리 안에 있는 하위 디렉터리와 파일을 모두 삭제

\$ rm -r testdir



```
MINGW64:/c/Users/jin/Documents
jin@JIN-Desktop MINGW64 ~/Documents
$ pwd
/c/Users/jin/Documents
jin@JIN-Desktop MINGW64 ~/Documents
$ rm -r testdir
jin@JIN-Desktop MINGW64 ~/Documents
$
```


- 터미널 종료
 - 'exit' 명령을 입력해서 종료

```
$ exit
```

A screenshot of a terminal window titled 'MINGW64: c/Users/jin'. The prompt is 'jin@JIN-Desktop MINGW64 ~'. The command '\$ exit' has been entered and is being processed, as indicated by a vertical cursor line at the end of the text.

```
MINGW64: c/Users/jin
jin@JIN-Desktop MINGW64 ~
$ exit
```

- 텍스트 문서 생성

- 텍스트 편집기 vim(Vim)을 이용하여 텍스트 문서 생성

터미널 환경에서 설정파일이나 기타 텍스트 문서를 생성하거나 편집할 때 사용

터미널 환경에서 사용하는 메모장이라고 생각하면 됨.

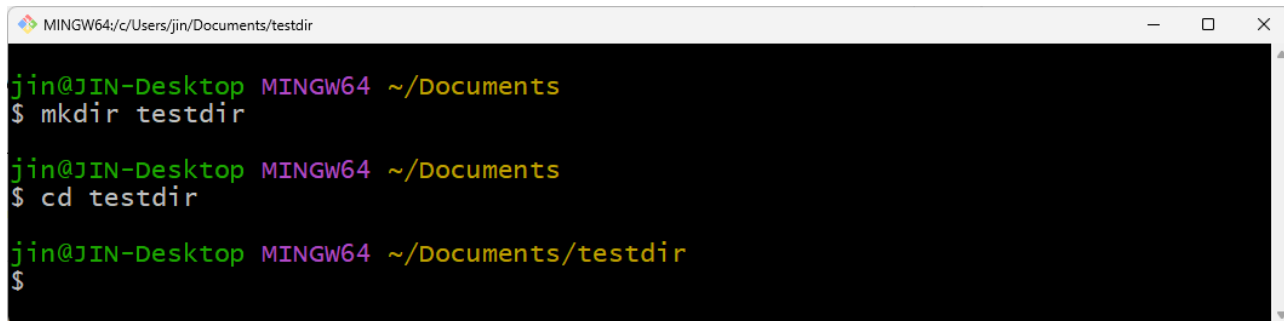
- 윈도우의 메모장과 다른 점은 터미널 화면에서 키보드만 이용해서 문서 생성 및 편집

- 텍스트 문서 생성

1. Documents 디렉터리로 이동 후 testdir 디렉터를 생성하고 testdir 디렉터리로 이동

```
$ mkdir testdir
```

```
$ cd testdir
```



```
MINGW64/c/Users/jin/Documents/testdir
jin@JIN-Desktop MINGW64 ~/Documents
$ mkdir testdir

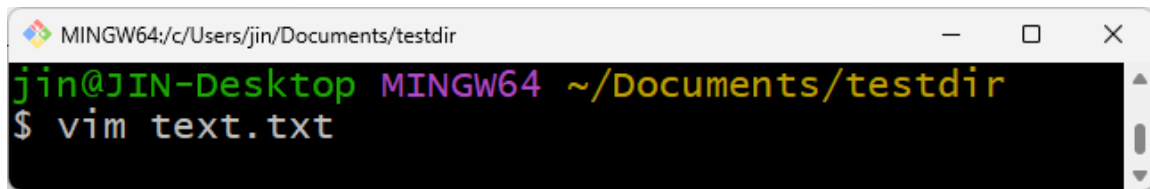
jin@JIN-Desktop MINGW64 ~/Documents
$ cd testdir

jin@JIN-Desktop MINGW64 ~/Documents/testdir
$
```

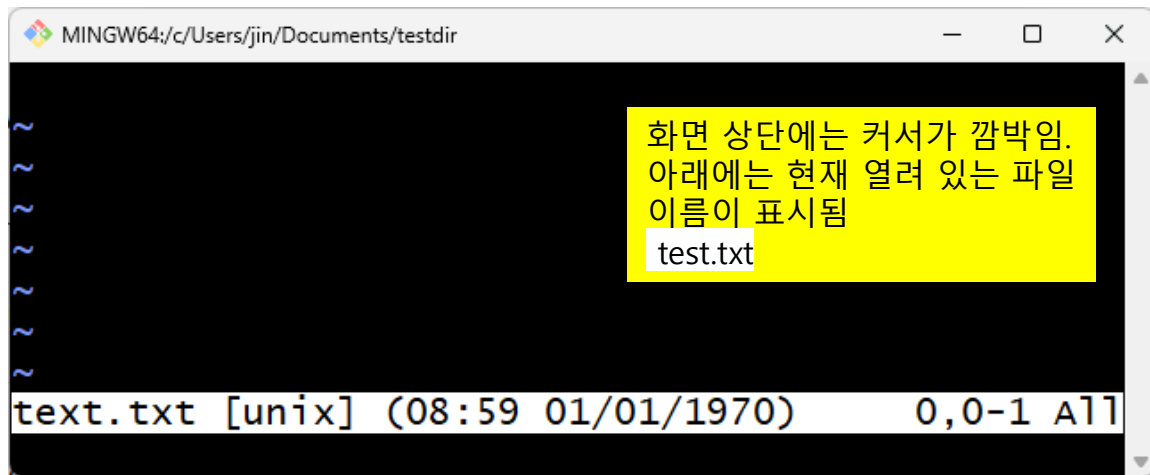
- 텍스트 문서 생성

2. testdir 디렉터리에 test.txt 파일을 생성하기 위해 **vim 명령**을 사용

\$ vim test.txt



```
MINGW64:/c/Users/jin/Documents/testdir
jin@JIN-Desktop MINGW64 ~/Documents/testdir
$ vim test.txt
```



```
MINGW64:/c/Users/jin/Documents/testdir

~
~
~
~
~
~
~
~
~
~

test.txt [unix] (08:59 01/01/1970) 0,0-1 A11
```

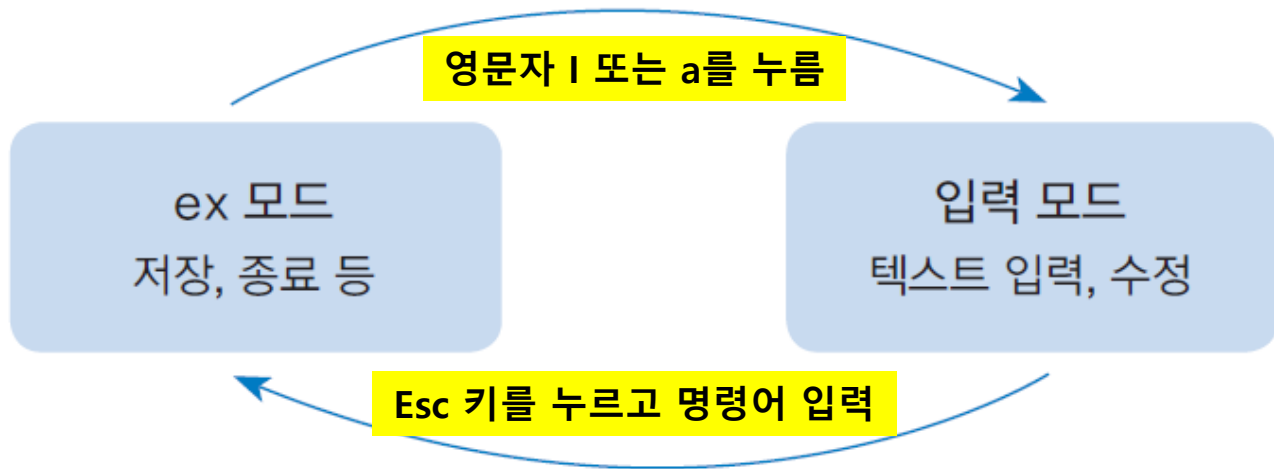
화면 상단에는 커서가 깜박임.
아래에는 현재 열려 있는 파일
이름이 표시됨
test.txt

- 텍스트 문서 생성

3. 새로 만든 파일에 아무 내용이나 입력해 보면 제대로 입력되지 않음

Vim에는 문서를 작성하는 '입력 모드'와 문서를 저장하는 'ex 모드'가 있음

Vim 명령으로 문서를 생성하거나 편집할 때 'ex모드'가 기본 모드



- 텍스트 문서 생성

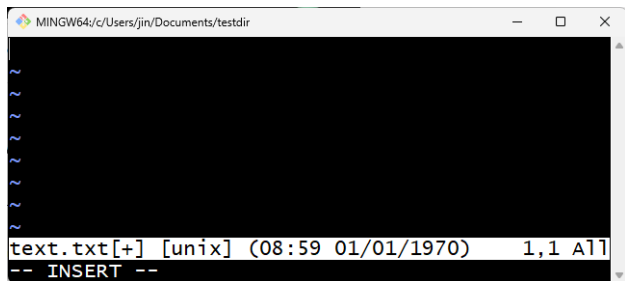
Vim의 ex 모드 명령

Vim의 ex 모드에서 사용하는 명령은 콜론(:)으로 시작

명령	설명
:w 또는 :write	편집하던 문서를 저장합니다.
:q 또는 :quit	편집기를 종료합니다.
:wq	편집하던 문서를 저장하고 종료합니다.
:q!	편집하던 문서를 저장하지 않고 편집기를 종료합니다. 확장자가 .swp인 임시 파일이 생깁니다.
:wq 파일명	편집하던 문서를 지정한 파일 이름으로 저장합니다.

- 텍스트 문서 생성

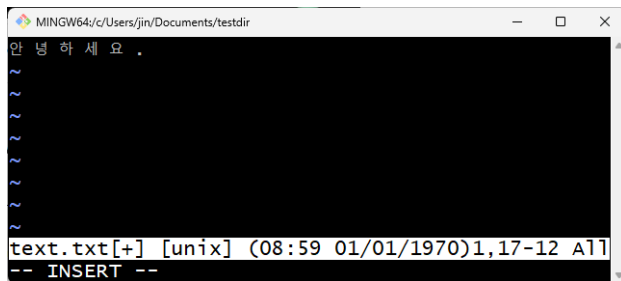
4. 텍스트 입력을 위해 ex 모드 상태에서 **i 또는 a를 눌러서 입력 모드 상태로 변경** → "안녕하세요." 입력



```

text.txt[+] [unix] (08:59 01/01/1970) 1,1 A|
-- INSERT --

```



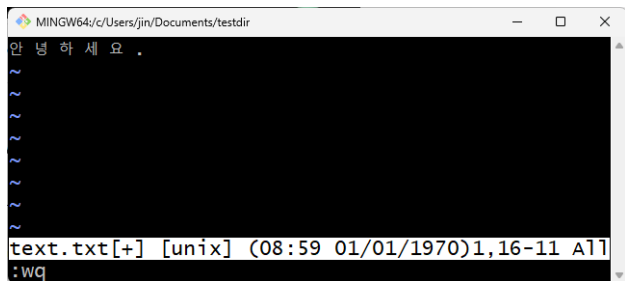
```

안녕하세요.
text.txt[+] [unix] (08:59 01/01/1970) 1,17-12 A|
-- INSERT --

```

파일을 저장하기 위해 입력 모드에서 **Esc 키를 누르고 ex 모드로 변경**

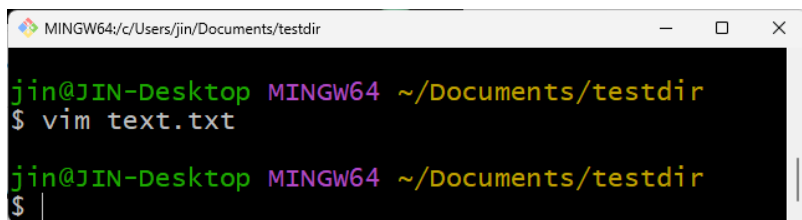
':wq'를 입력하고 저장하고 Vim 편집기 종료하고 터미널 환경으로 돌아옴



```

안녕하세요.
text.txt[+] [unix] (08:59 01/01/1970) 1,16-11 A|
:wq

```



```

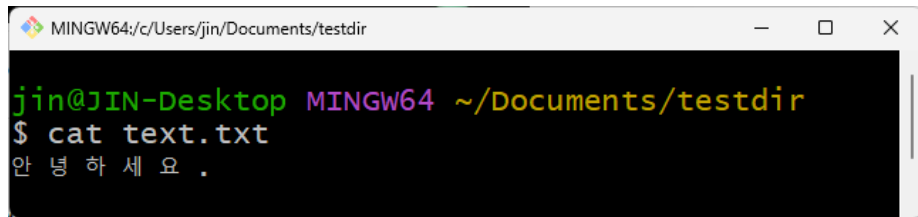
jin@JIN-Desktop MINGW64 ~/Documents/testdir
$ vim text.txt
jin@JIN-Desktop MINGW64 ~/Documents/testdir
$

```

- 텍스트 문서 확인

- 터미널 환경에서 텍스트 문서의 내용을 간단히 확인할 때는 리눅스의 `cat {파일이름} 명령`을 사용
- test.txt 파일의 내용 확인

`$ cat text.txt`



```
MINGW64/c/Users/jin/Documents/testdir
jin@JIN-Desktop MINGW64 ~/Documents/testdir
$ cat text.txt
안녕하세요.
```


- 현재 디렉터리 확인 : `$ pwd`
- 현재 디렉터리의 파일과 하위 디렉터리 확인 : `$ ls`, `$ ls --l`
- 터미널 창 내용 지우기 : `$ clear`
- 디렉터리 이동 : `$ cd ..`, `$ cd 폴더명`, `$ cd ~`
- 디렉터리 생성 및 삭제 : `$ mkdir`, `$ rm`, `$ rm -r`
- 터미널 종료 : `$ exit`
- 텍스트 문서 생성 : `$ vim 파일이름`
- 텍스트 문서 확인 : `$ cat 파일이름`

Git & GitHub

4. Git을 이용한 버전 관리

- Git의 3가지 영역
- Git의 저장소 구조
- Git 버전 생성 과정
- Commit 기록 확인
- 변경사항 확인
- 단계별 파일 상태 확인
- 작업 되돌리기

- Git의 3가지 영역

- 작업 폴더 (Working Directory), 작업 트리

- 사용자가 변경하는 실제 파일이 들어가는 폴더

- 스테이지 (Stage, Index)

- 변경사항을 관리할 파일들의 리스트, 버전으로 만들 파일이 대기하는 곳

- 변경이력 (History), 저장소 (repository)

- 커밋 (Commit)이라 불리는 변경사항 묶음과 커밋 들의 연결관계를 저장하는 곳

- 리포지토리 (repository)는 스테이지에서 대기하고 있던 파일들을 버전으로 만들어 저장

※ 커밋 (Commit)

하나의 트랜잭션에 포함되는 조작이 모두 실행되고 그에 따른 갱신 내용이 작업 영역(기억 장치)에 기록되어 트랜잭션의 적용이 완료되었다고 판단되는 시점에서 그 종료를 요구하는 동작

※ 트랜잭션

특정 목적을 수행하는 일련의 작업들의 집합

- Git의 3가지 영역

- 스테이지와 저장소는 눈에 보이지 않음
- 깃을 초기화했을 때 만들어지는 .git 디렉터리 안에 숨은 파일 형태로 존재하는 영역

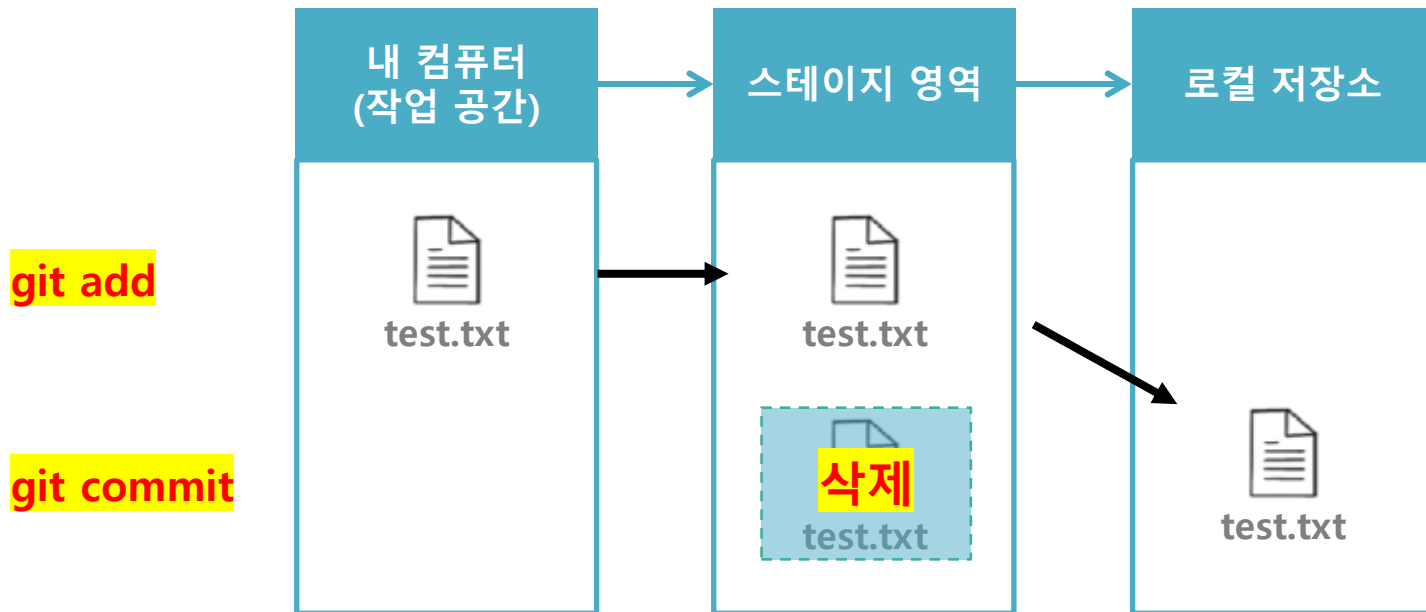


- Git에는 로컬 저장소에 커밋을 하기 전, 스테이지 영역(인덱스) 단계가 있음

스테이지 영역
(Staging Area)

스테이지 영역에 변경된 파일들 중 커밋 할 파일들을 저장하는 공간
으로 이후 로컬 저장소로 커밋 하는 영역

- Git의 저장소 구조



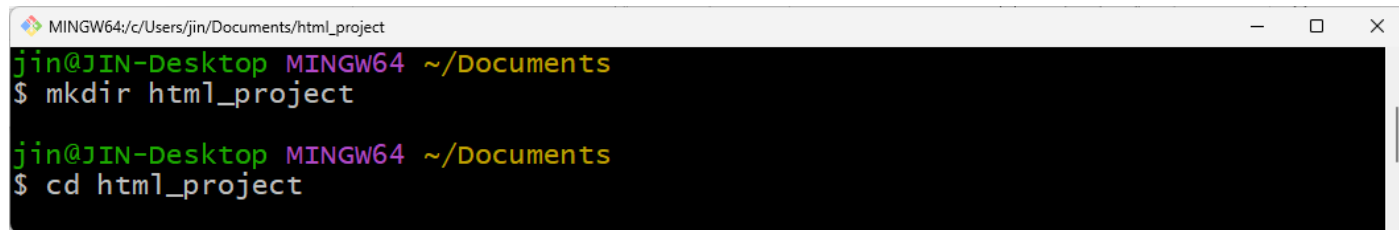
- Git 버전 생성 과정
 1. 저장소로 사용할 디렉터리 생성
 2. 생성한 디렉터리에서 Git을 사용할 수 있도록 초기화
 3. 작업 디렉토리에서 문서를 생성 또는 수정
 4. 생성 또는 수정한 파일 가운데 버전으로 만들고 싶은 파일을 스테이징 영역(스테이지)에 저장
 5. 스테이지에 있던 파일을 저장소로 커밋하면 버전이 생성됨

- Git 버전 생성 과정

- 저장소로 사용할 디렉터리 생성

```
$ mkdir html_project
```

```
$ cd html_project
```

A screenshot of a terminal window with a title bar that reads 'MINGW64; c:/Users/jin/Documents/html_project'. The terminal has a black background with green and yellow text. The first prompt shows the user 'jin@JIN-Desktop' in a 'MINGW64' environment at the '~ /Documents' location, where they enter '\$ mkdir html_project'. The second prompt shows the same user and environment, where they enter '\$ cd html_project'.

```
MINGW64; c:/Users/jin/Documents/html_project
jin@JIN-Desktop MINGW64 ~/Documents
$ mkdir html_project

jin@JIN-Desktop MINGW64 ~/Documents
$ cd html_project
```

- Git 버전 생성 과정

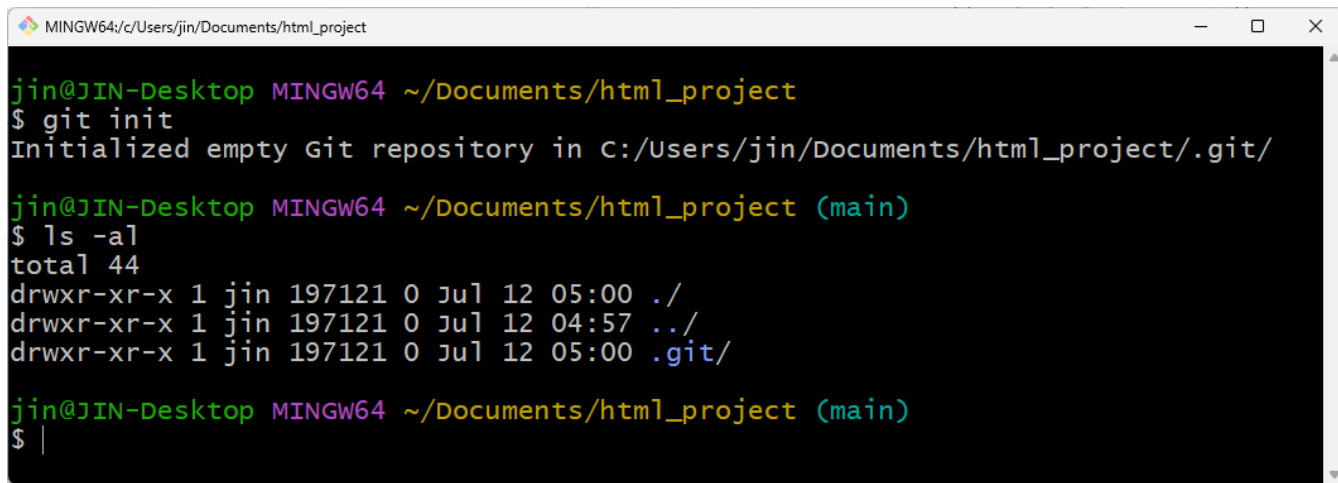
2. 생성한 디렉터리에서 Git을 사용할 수 있도록 초기화

jin@JIN-Desktop MINGW64 ~/Documents/html_project

\$ git init

Initialized empty Git repository in C:/Users/jin/Documents/html_project/.git/

jin@JIN-Desktop MINGW64 ~/Documents/html_project **(main) ← 저장소 생성 됨**



```
MINGW64/c:/Users/jin/Documents/html_project
jin@JIN-Desktop MINGW64 ~/Documents/html_project
$ git init
Initialized empty Git repository in c:/Users/jin/Documents/html_project/.git/

jin@JIN-Desktop MINGW64 ~/Documents/html_project (main)
$ ls -al
total 44
drwxr-xr-x 1 jin 197121 0 Jul 12 05:00 ./
drwxr-xr-x 1 jin 197121 0 Jul 12 04:57 ../
drwxr-xr-x 1 jin 197121 0 Jul 12 05:00 .git/

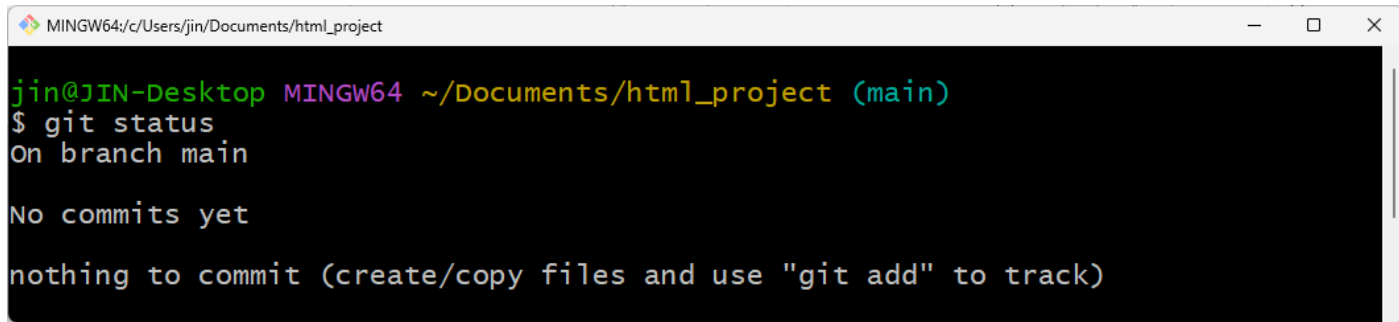
jin@JIN-Desktop MINGW64 ~/Documents/html_project (main)
$ |
```


- Git 버전 생성 과정

- 3. 작업 디렉토리에서 문서를 생성 또는 수정

Git 상태를 확인

`$ git status`

A terminal window titled 'MINGW64:/c:/Users/jin/Documents/html_project' showing the output of the 'git status' command. The output indicates the user is on the 'main' branch, there are no commits yet, and nothing is staged for commit.

```
MINGW64:/c:/Users/jin/Documents/html_project

jin@JIN-Desktop MINGW64 ~/Documents/html_project (main)
$ git status
On branch main

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

Git 상태 메시지

`On branch main`: 현재 main 브랜치에 있음

`No commits yet`: 아직 커밋한 파일이 없음

`nothing to commit`: 현재 커밋할 파일이 없음

- Git 버전 생성 과정

3. 작업 디렉토리에서 문서를 생성 또는 수정

작업 디렉터리에 index.html 파일 생성

\$ vim index.html

```
MINGW64:/c/Users/jin/Documents/html_project
jin@JIN-Desktop MINGW64 ~/Documents/html_project (main)
$ vim index.html
```

```
MINGW64:/c/Users/jin/Documents/html_project
<html>
<head>
  <title>test page</title>
</head>
<body>
  <h1>안 념 하 세 요 .</h1>
</body>
</html>
~
<html[+] [unix] (08:59 01/01/1970)8,8 All
-- INSERT --
```

```
MINGW64:/c/Users/jin/Documents/html_project
<html>
<head>
  <title>test page</title>
</head>
<body>
  <h1>안 념 하 세 요 .</h1>
</body>
</html>
~
<html[+] [unix] (08:59 01/01/1970)8,7 All
:wq
```

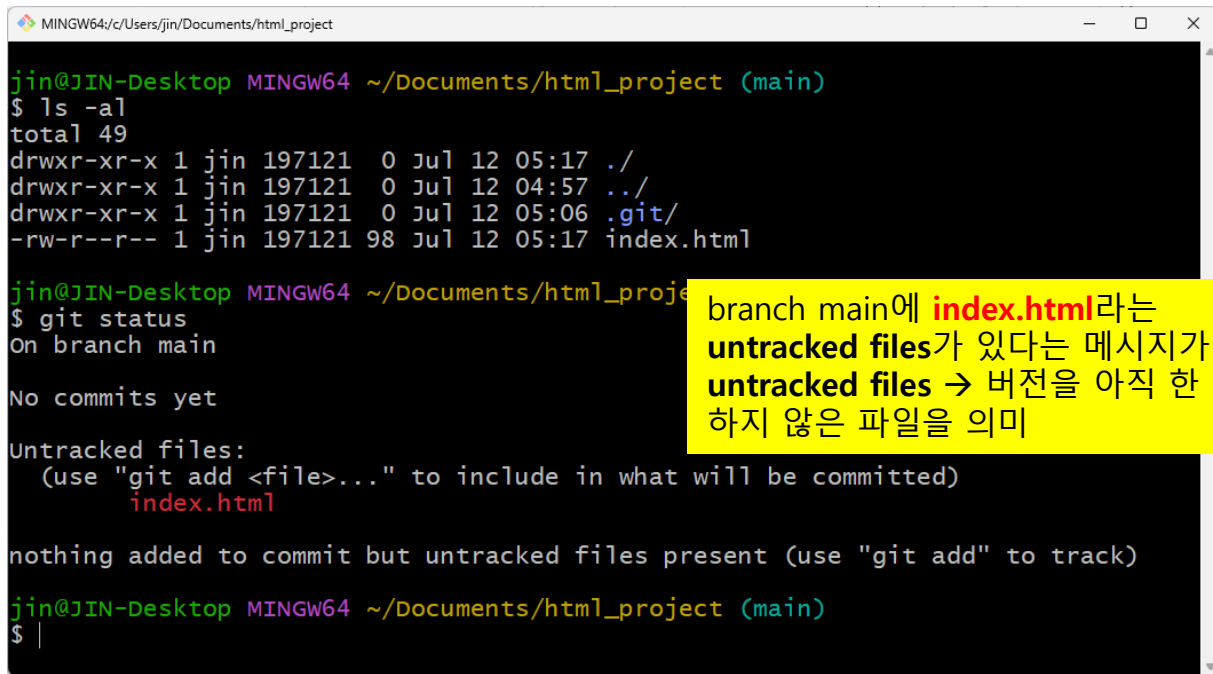
- Git 버전 생성 과정

- 3. 작업 디렉토리에서 문서를 생성 또는 수정

작업 디렉토리에 index.html 파일 생성 확인 및 깃 상태 확인

`$ ls -al`

`$ git status`

A terminal window titled 'MINGW64: c:/Users/jin/Documents/html_project' showing the execution of 'ls -al' and 'git status' commands. The 'ls -al' command lists the directory contents, including a new 'index.html' file. The 'git status' command shows that the file is untracked and not yet committed.

```
jinn@JIN-Desktop MINGW64 ~/Documents/html_project (main)
$ ls -al
total 49
drwxr-xr-x 1 jin 197121  0 Jul 12 05:17 ./
drwxr-xr-x 1 jin 197121  0 Jul 12 04:57 ../
drwxr-xr-x 1 jin 197121  0 Jul 12 05:06 .git/
-rw-r--r-- 1 jin 197121 98 Jul 12 05:17 index.html

jinn@JIN-Desktop MINGW64 ~/Documents/html_project (main)
$ git status
On branch main

No commits yet

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

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

jinn@JIN-Desktop MINGW64 ~/Documents/html_project (main)
$
```

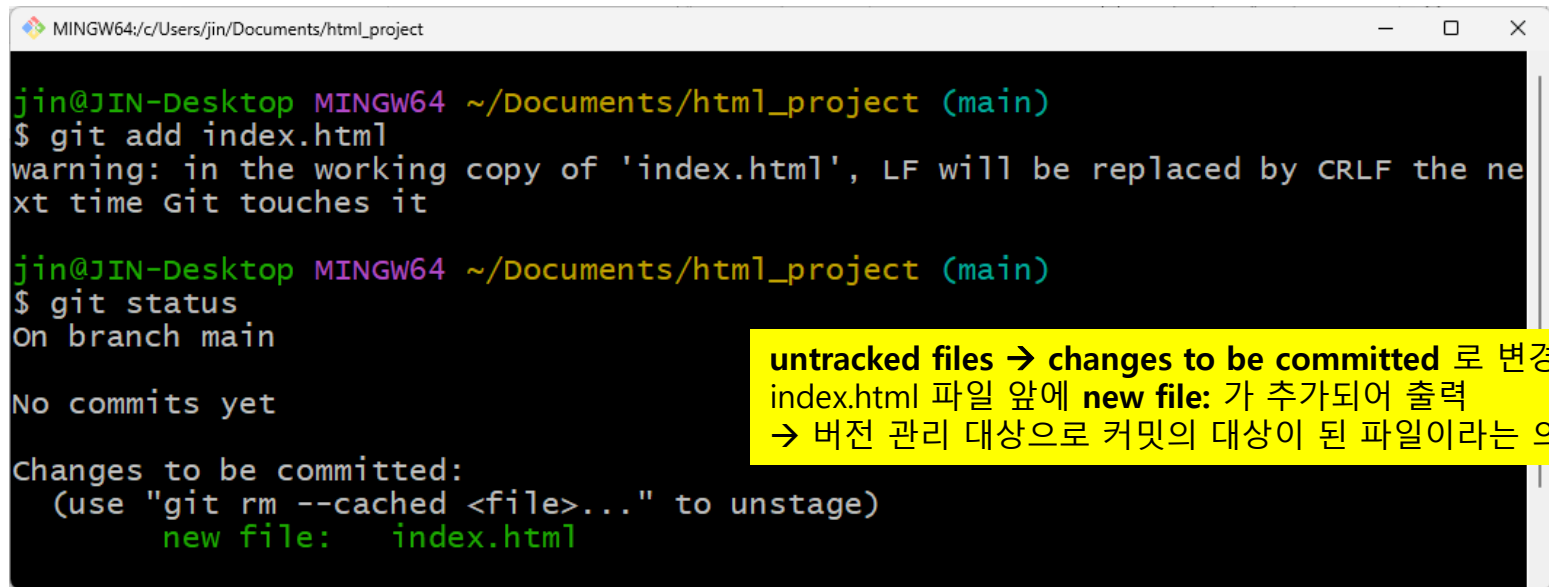
branch main에 **index.html**라는 **untracked files**가 있다는 메시지가 출력
untracked files → 버전을 아직 한 번도 관리하지 않은 파일을 의미

- Git 버전 생성 과정

- 생성 또는 수정한 파일 가운데 버전으로 만들고 싶은 파일을 스테이징 영역(스테이지)에 저장

```
$ git add index.html
```

```
$ git status
```



```
MINGW64/c:/Users/jin/Documents/html_project
jin@JIN-Desktop MINGW64 ~/Documents/html_project (main)
$ git add index.html
warning: in the working copy of 'index.html', LF will be replaced by CRLF the next time Git touches it

jin@JIN-Desktop MINGW64 ~/Documents/html_project (main)
$ git status
On branch main

No commits yet

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

untracked files → changes to be committed 로 변경
index.html 파일 앞에 **new file:** 가 추가되어 출력
→ 버전 관리 대상으로 커밋의 대상이 된 파일이라는 의미

- Git 버전 생성 과정

5. 스테이지에 있던 파일을 저장소로 커밋하면 버전이 생성됨

- 커밋하는 명령은 `git commit`을 사용,

한 칸 띄운 후에 `-m` 옵션을 붙이고 커밋과 함께 저장할 메시지를 적고 실행

`$ git commit -m "index.html 파일 생성"`

`$ git status`

MINGW64:/c:/Users/jin/Documents/html_project

jin@JIN-Desktop MINGW64 ~/Documents/html_project (main)

`$ git commit -m "index.html 파일 생성"`

`[main (root-commit) 716814b] index.html 파일 생성`

`1 file changed, 8 insertions(+)`

`create mode 100644 index.html`

jin@JIN-Desktop MINGW64 ~/Documents/html_project

`$ git status`

`On branch main`

`nothing to commit, working tree clean`

파일 1개가 변경되었고(1 file changed)

파일에 8개의 내용이 추가되었다(1 insertion(+))고 출력

Git에서 변경사항은 파일의 라인을 단위로 변경 체크

버전으로 만들 파일이 없고(nothing to commit)

작업 트리도 수정 사항 없이 깨끗하다(working tree

clean)고 출력

- Git 버전 생성 과정

- 5. 스테이지에 있던 파일을 저장소로 커밋하면 버전이 생성됨

- 스테이징과 커밋 한꺼번에 처리하기 → `git commit -am` 명령 사용

`$ vim index.html`

`$ git status`

```
MINGW64/c/Users/jin/Documents/html_project
<html>
<head>
  <title>test page</title>
</head>
<body>
  <h1>안녕하세요.</h1>
  <p>Hello!!!</p>
</body>
</html>
index.html[+] [unix] (05:17 12/07/2023)
-- INSERT --
```

Changes not staged for commit:

→ commit 을 위한 stage에 추가되지 않은 상태를 의미
index.html 파일 앞에 **modified file:** 가 추가되어 출력
→ 파일이 수정되었음을 표시

```
jin@JIN-Desktop MINGW64 ~/Documents/html_project (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:   index.html

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

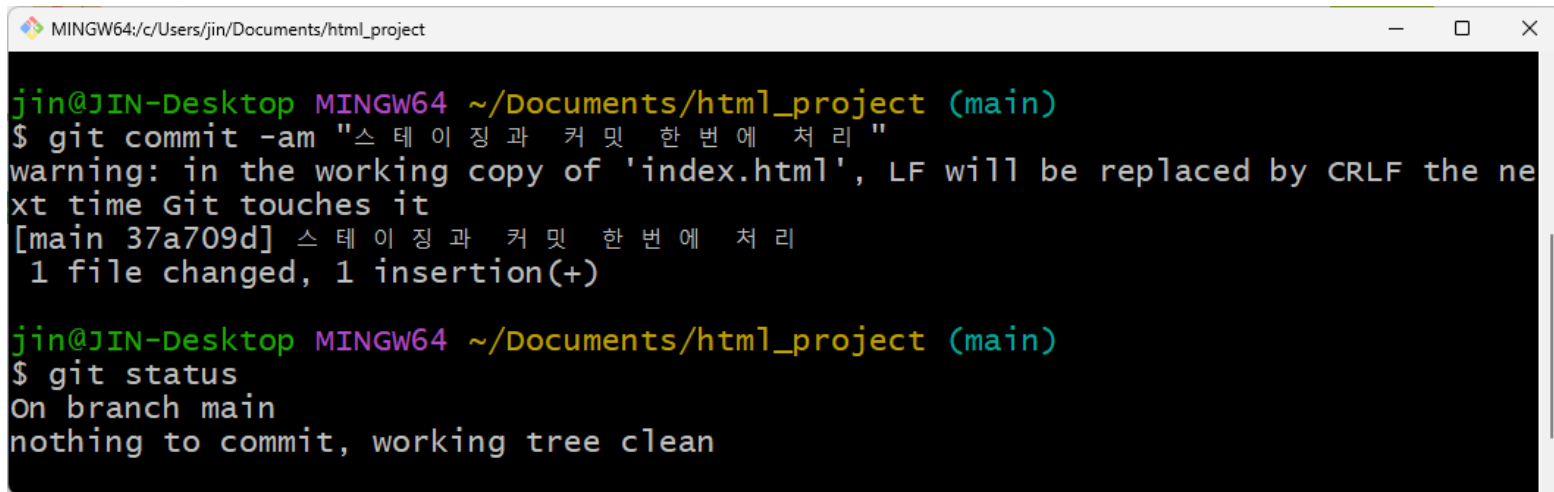
- Git 버전 생성 과정

5. 스테이지에 있던 파일을 저장소로 커밋하면 버전이 생성됨

- 스테이징과 커밋 한꺼번에 처리하기 → `git commit -am` 명령 사용

`$ git commit -am "<p> 태그 추가 스테이징과 커밋을 한번에 처리"`

`$ git status`



```
MINGW64/c/Users/jin/Documents/html_project

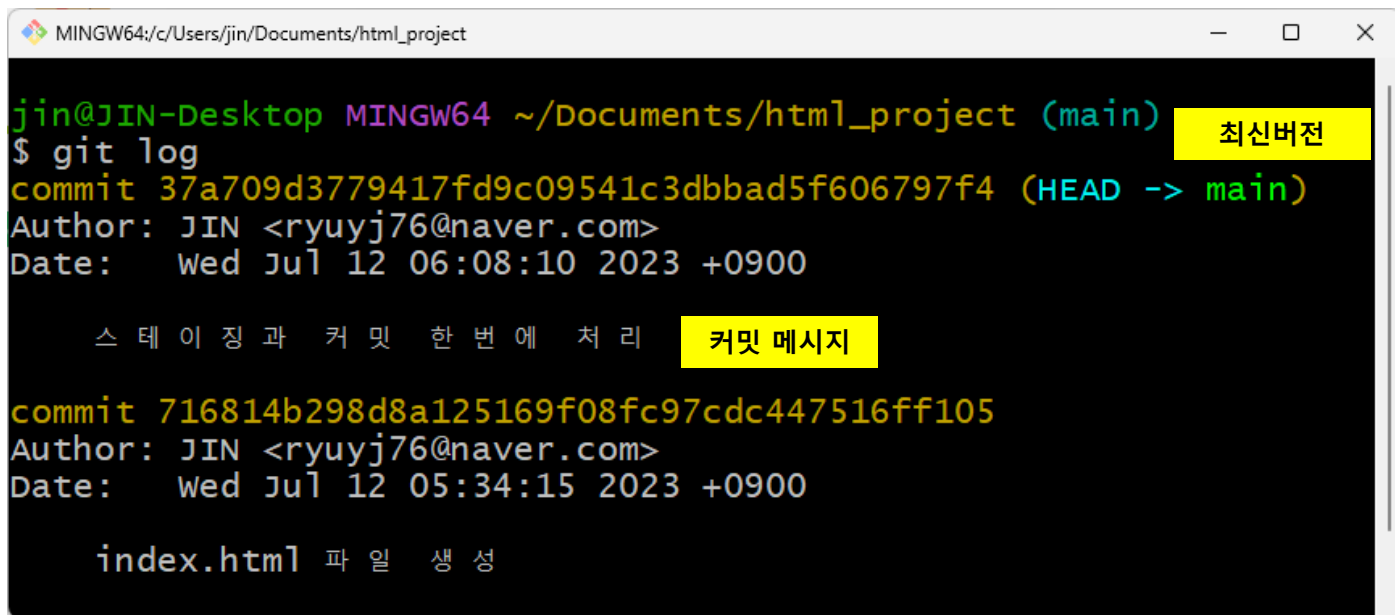
jin@JIN-Desktop MINGW64 ~/Documents/html_project (main)
$ git commit -am "스 테 이 징 과 커 밋 한 번 에 처 리 "
warning: in the working copy of 'index.html', LF will be replaced by CRLF the ne
xt time Git touches it
[main 37a709d] 스 테 이 징 과 커 밋 한 번 에 처 리
 1 file changed, 1 insertion(+)

jin@JIN-Desktop MINGW64 ~/Documents/html_project (main)
$ git status
On branch main
nothing to commit, working tree clean
```

- Commit 기록 확인
 - 커밋 로그 확인 - 최근 commit 정보가 가장 먼저 출력

\$ git log

커밋 해시
작성자
커밋 날짜



```
MINGW64:/c/Users/jin/Documents/html_project

jin@JIN-Desktop MINGW64 ~/Documents/html_project (main)
$ git log
commit 37a709d3779417fd9c09541c3dbbad5f606797f4 (HEAD -> main)
Author: JIN <ryuyj76@naver.com>
Date:   Wed Jul 12 06:08:10 2023 +0900

    스테이징과 커밋 한번에 처리

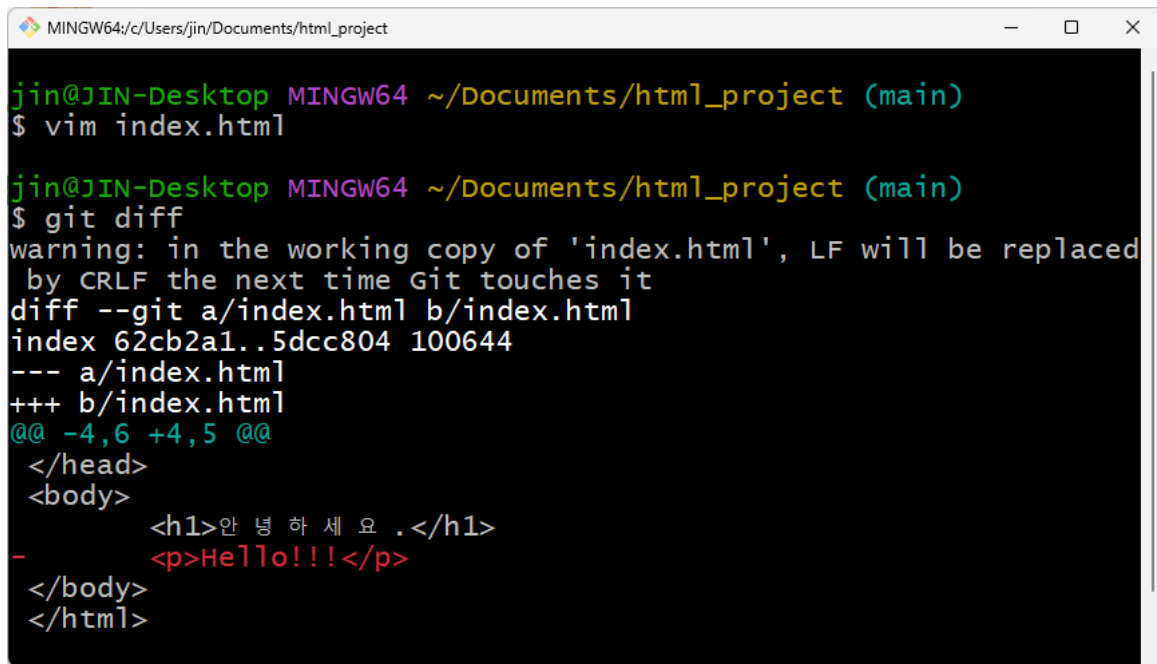
commit 716814b298d8a125169f08fc97cdc447516ff105
Author: JIN <ryuyj76@naver.com>
Date:   Wed Jul 12 05:34:15 2023 +0900

    index.html 파일 생성
```


- 변경사항 확인
 - 마지막 커밋 상태의 파일과 현재 수정한 파일이 어떻게 다른 지 확인
 - `$ git diff` 명령 사용

`$ vim index.html`

`$ git diff`

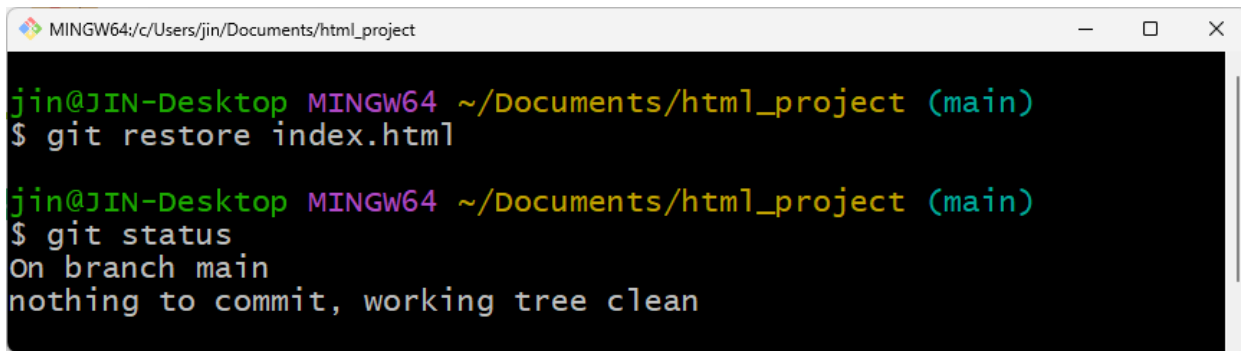


```
MINGW64/c:/Users/jin/Documents/html_project

jin@JIN-Desktop MINGW64 ~/Documents/html_project (main)
$ vim index.html

jin@JIN-Desktop MINGW64 ~/Documents/html_project (main)
$ git diff
warning: in the working copy of 'index.html', LF will be replaced
by CRLF the next time Git touches it
diff --git a/index.html b/index.html
index 62cb2a1..5dcc804 100644
--- a/index.html
+++ b/index.html
@@ -4,6 +4,5 @@
 </head>
 <body>
     <h1>안 념 하 세 요 .</h1>
-    <p>Hello!!!</p>
 </body>
 </html>
```

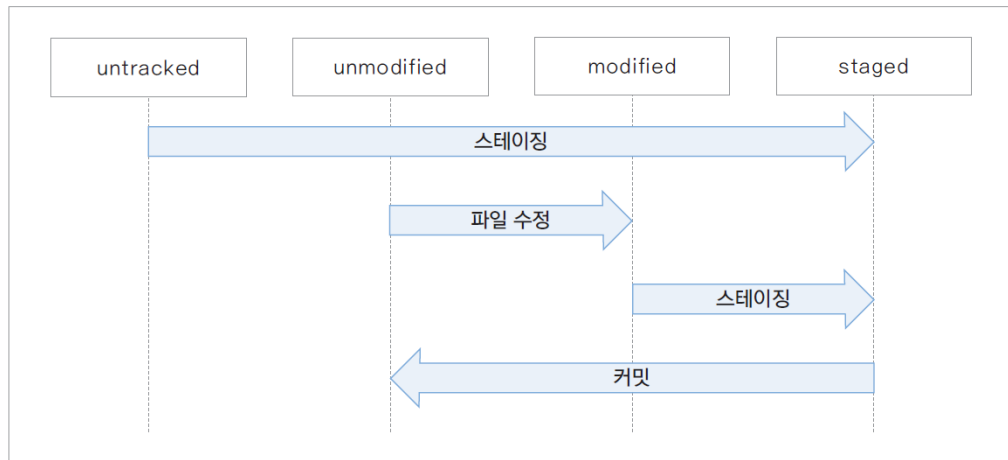
- 변경사항 확인
 - 현재 수정한 파일의 변경 내용을 삭제
 - `$ git restore {파일명}` 명령 사용
- `$ git status`

A screenshot of a terminal window titled 'MINGW64; c:/Users/jin/Documents/html_project'. The window shows a user named 'jin@JIN-Desktop' in a 'MINGW64' environment at the directory '~/Documents/html_project' on the 'main' branch. The user enters the command '\$ git restore index.html'. After a second prompt, the user enters '\$ git status'. The output of the status command is 'On branch main' and 'nothing to commit, working tree clean'.

```
MINGW64; c:/Users/jin/Documents/html_project
jin@JIN-Desktop MINGW64 ~/Documents/html_project (main)
$ git restore index.html

jin@JIN-Desktop MINGW64 ~/Documents/html_project (main)
$ git status
On branch main
nothing to commit, working tree clean
```

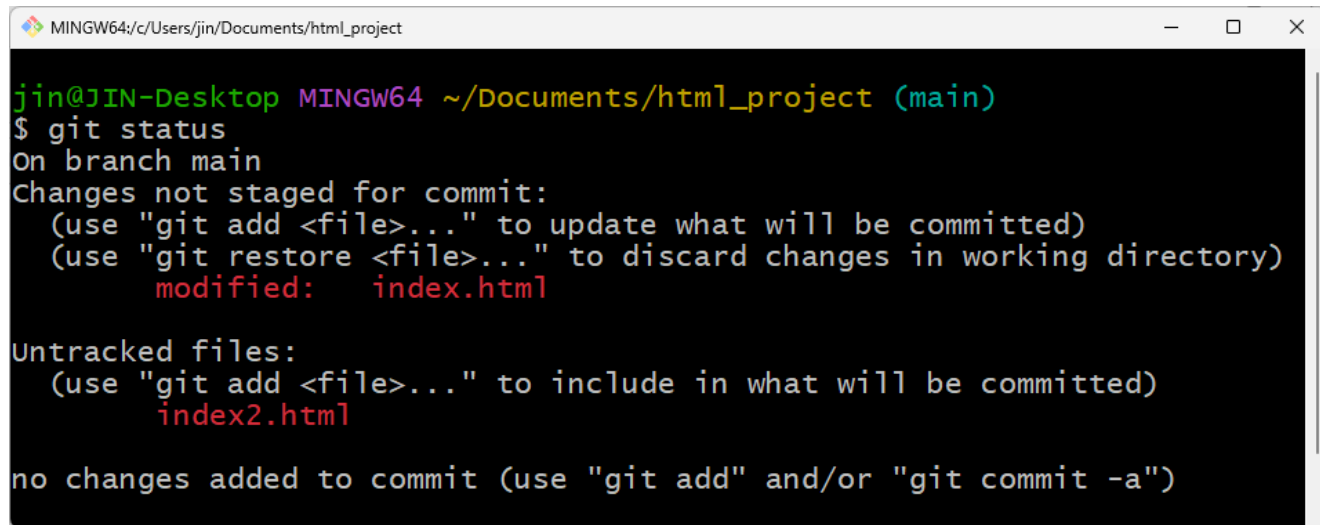
- 단계별 파일 상태 확인



- **tracked** : 버전을 만든 후에 깃에서 변경 내역을 추적 중인 상태
- **untracked** : 한번도 버전을 만들지 않은 상태
- **unmodified** : 수정되지 않은 상태 (working tree is clean)
- **modified** : 작업 트리에서 수정한 후 아직 스테이지에 저장하지 않은 상태
- **staged** : 스테이지에 있고 아직 커밋하지 않은 상태

- 단계별 파일 상태 확인
 - tracked 파일과 untracked 파일
 - ① index.html 파일 수정
 - ② index2.html 파일 생성
 - ③ 상태 확인

\$ git status

A terminal window titled 'MINGW64/c/Users/jin/Documents/html_project' showing the output of the 'git status' command. The output indicates that the 'main' branch is checked out, and there are changes to 'index.html' that are not staged for commit. It also shows an untracked file 'index2.html'.

```
jin@JIN-Desktop MINGW64 ~/Documents/html_project (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:   index.html

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        index2.html

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

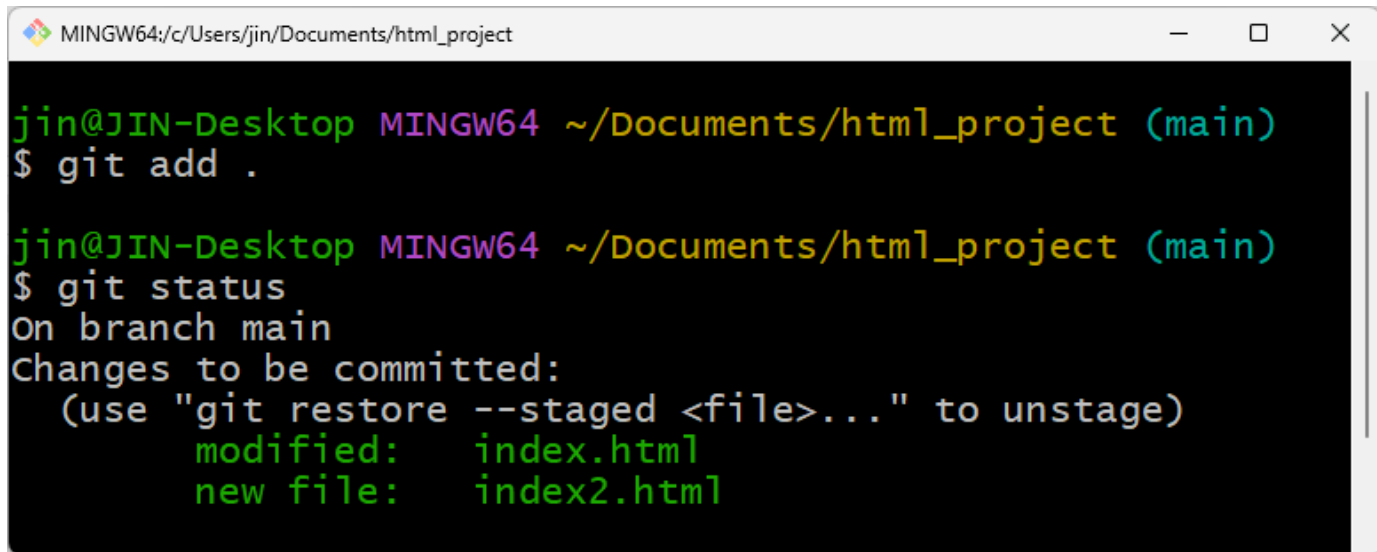
- 단계별 파일 상태 확인

- tracked 파일과 untracked 파일

수정한 index.html 파일과 새롭게 생성한 index2.html 파일 모두 git add 명령을 사용해서 스테이징

`$ git add .`

`$ git status`



A screenshot of a terminal window titled "MINGW64:/c:/Users/jin/Documents/html_project". The terminal shows the following commands and output:

```
jin@JIN-Desktop MINGW64 ~/Documents/html_project (main)
$ git add .

jin@JIN-Desktop MINGW64 ~/Documents/html_project (main)
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   index.html
        new file:   index2.html
```

- 단계별 파일 상태 확인
 - tracked 파일과 untracked 파일
- 커밋 로그에서 커밋된 파일까지 확인

\$ git log --stat

```
commit b796a308fb1b5cb9605c2a54c1c0b581355a7c26 (HEAD -> main)
Author: JIN <ryuyj76@naver.com>
Date:   Wed Jul 12 06:52:40 2023 +0900

    3번 째 커밋

index.html | 2 +-
index2.html | 8 ++++++++
2 files changed, 9 insertions(+), 1 deletion(-)

commit 37a709d3779417fd9c09541c3dbbad5f606797f4
Author: JIN <ryuyj76@naver.com>
Date:   Wed Jul 12 06:08:10 2023 +0900

    스테이징과 커밋 한번에 처리

index.html | 1 +
1 file changed, 1 insertion(+)
```

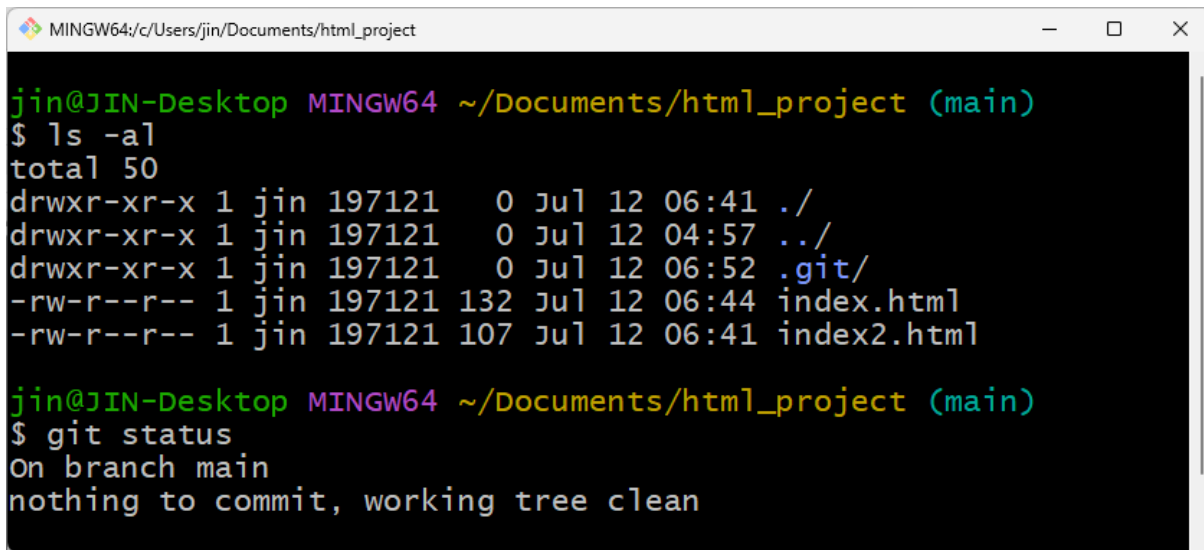
- 단계별 파일 상태 확인
 - unmodified, modified, staged 상태

① 워크 트리에 파일 리스트 확인

`$ ls -al`

② Git 상태 확인

`$ git status`



The screenshot shows a terminal window titled 'MINGW64/c/Users/jin/Documents/html_project'. The user 'jin@JIN-Desktop' is in the 'main' branch of a repository. They run the command `$ ls -al`, which lists the files in the working tree: `total 50`, `drwxr-xr-x 1 jin 197121 0 Jul 12 06:41 ./`, `drwxr-xr-x 1 jin 197121 0 Jul 12 04:57 ../`, `drwxr-xr-x 1 jin 197121 0 Jul 12 06:52 .git/`, `-rw-r--r-- 1 jin 197121 132 Jul 12 06:44 index.html`, and `-rw-r--r-- 1 jin 197121 107 Jul 12 06:41 index2.html`. Then, they run `$ git status`, which returns `On branch main` and `nothing to commit, working tree clean`.

```
jin@JIN-Desktop MINGW64 ~/Documents/html_project (main)
$ ls -al
total 50
drwxr-xr-x 1 jin 197121 0 Jul 12 06:41 ./
drwxr-xr-x 1 jin 197121 0 Jul 12 04:57 ../
drwxr-xr-x 1 jin 197121 0 Jul 12 06:52 .git/
-rw-r--r-- 1 jin 197121 132 Jul 12 06:44 index.html
-rw-r--r-- 1 jin 197121 107 Jul 12 06:41 index2.html

jin@JIN-Desktop MINGW64 ~/Documents/html_project (main)
$ git status
On branch main
nothing to commit, working tree clean
```

- 단계별 파일 상태 확인
 - unmodified, modified, staged 상태

③ index2.html 파일 수정

④ Git 상태 확인

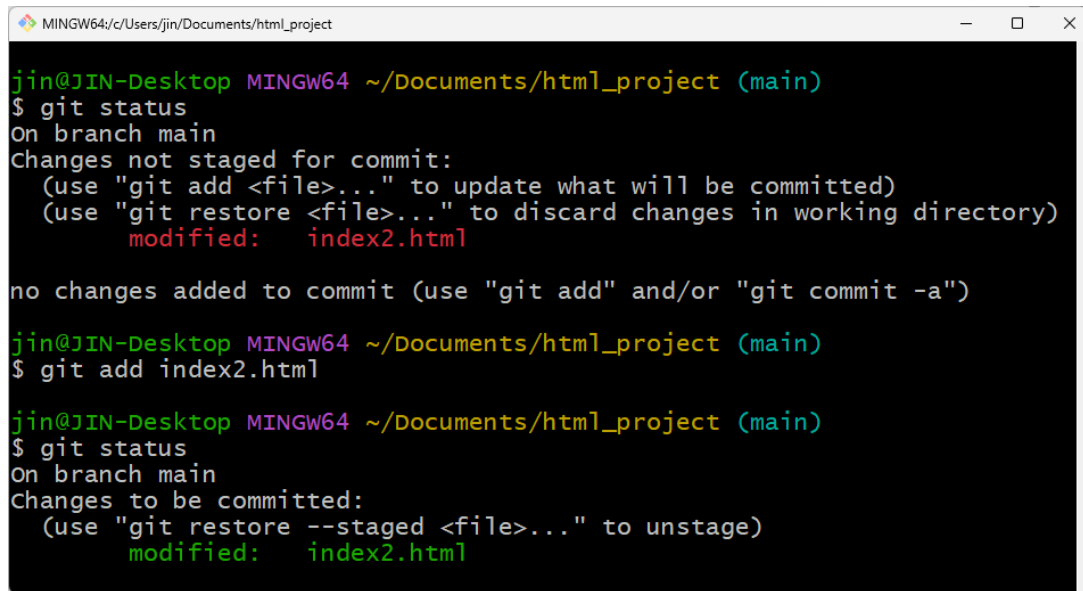
`$ git status`

⑤ 변경된 파일을 스테이지에 올림

`$ git add index2.html`

⑥ Git 상태 확인

`$ git status`



```
MINGW64/c:/Users/jin/Documents/html_project
jin@JIN-Desktop MINGW64 ~/Documents/html_project (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:   index2.html

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

jin@JIN-Desktop MINGW64 ~/Documents/html_project (main)
$ git add index2.html

jin@JIN-Desktop MINGW64 ~/Documents/html_project (main)
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   index2.html
```


- 단계별 파일 상태 확인

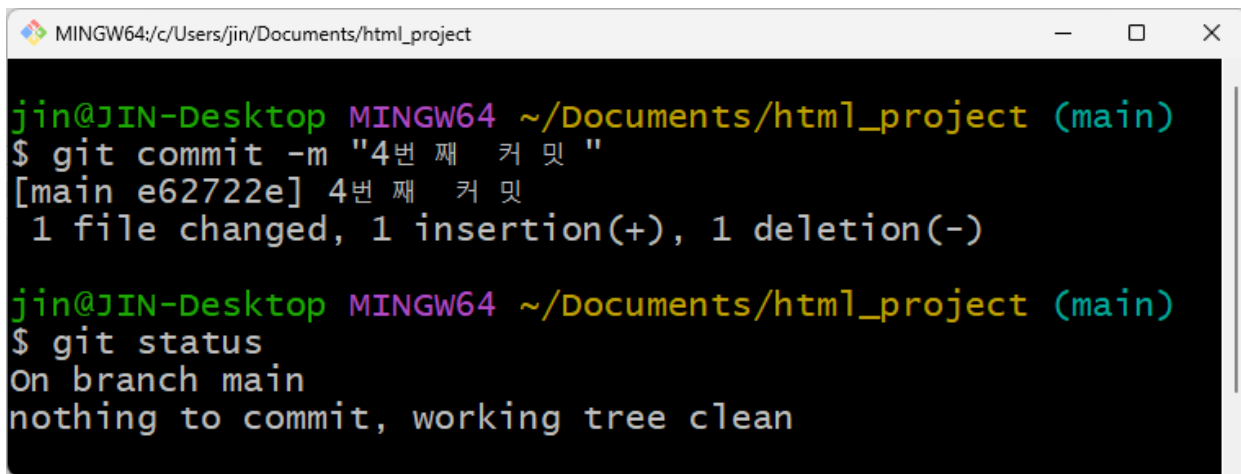
- unmodified, modified, staged 상태

- ⑦ 스테이지에 있는 파일 커밋

- \$ git commit -m "4번째 커밋"

- ⑧ Git 상태 확인

- \$ git status



```
MINGW64:/c/Users/jin/Documents/html_project

jin@JIN-Desktop MINGW64 ~/Documents/html_project (main)
$ git commit -m "4번째 커밋"
[main e62722e] 4번째 커밋
1 file changed, 1 insertion(+), 1 deletion(-)

jin@JIN-Desktop MINGW64 ~/Documents/html_project (main)
$ git status
On branch main
nothing to commit, working tree clean
```

- 작업 되돌리기
 - 작업 트리에서 수정한 파일 되돌리기
`$ git restore`
 - 스테이징 되돌리기
`$ git restore -staged`
 - 최신 커밋 되돌리기
`$ git reset HEAD^`
 - 특정 커밋으로 되돌리기
`$ git reset 해시`

- 작업 되돌리기

- 작업 트리에서 수정한 파일 되돌리기

`$ git restore`

① index.html 파일 수정

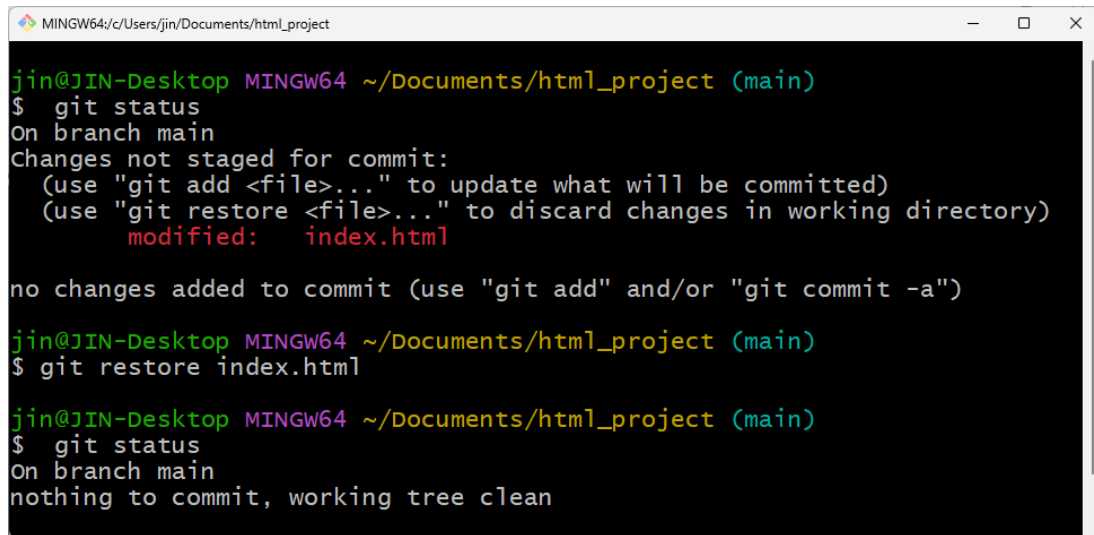
② Git 상태 확인

`$ git status`

③ index.html 파일 수정 취소

`$ git restore index.html`

④ index.html 파일 확인

A screenshot of a Windows terminal window titled 'MINGW64/c/Users/jin/Documents/html_project'. The terminal shows a series of commands and their outputs. First, 'git status' is run, showing that 'index.html' is modified but not staged. Then, 'git restore index.html' is run, which discards the changes. Finally, 'git status' is run again, showing that the working tree is clean.

```
jin@JIN-Desktop MINGW64 ~/Documents/html_project (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:   index.html

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

jin@JIN-Desktop MINGW64 ~/Documents/html_project (main)
$ git restore index.html

jin@JIN-Desktop MINGW64 ~/Documents/html_project (main)
$ git status
On branch main
nothing to commit, working tree clean
```

- 작업 되돌리기

- 스태이징 되돌리기

`$ git restore --staged`

① index.html 파일 수정

② index.html 파일 스테이지에 올림

`$ git add index.html`

③ Git 상태 확인

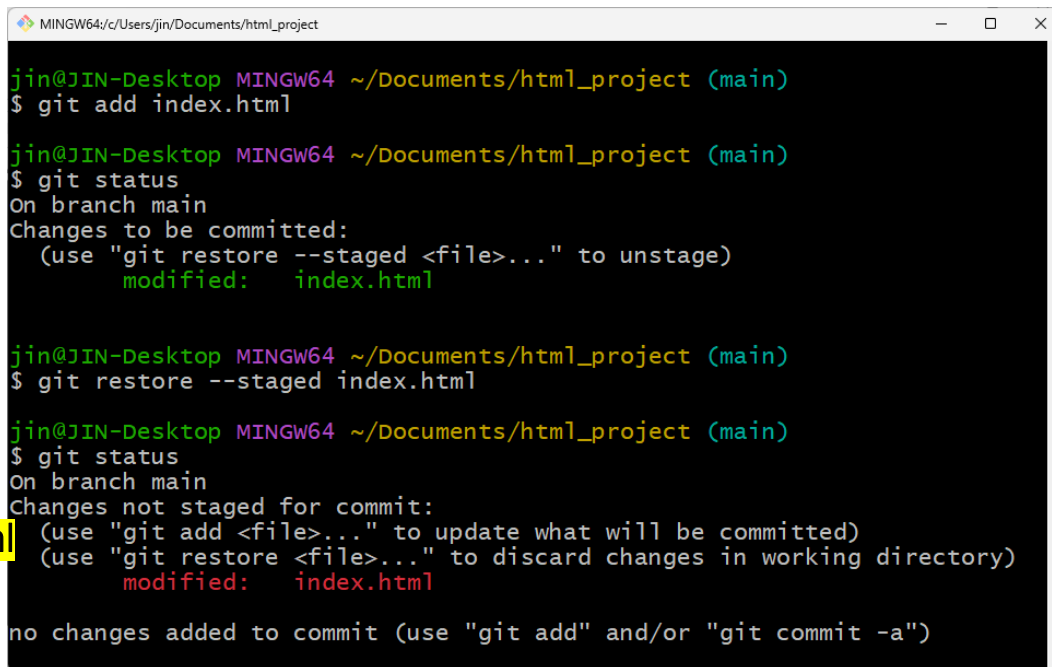
`$ git status`

④ index.html 파일 수정 취소

`$ git restore --staged index.html`

⑤ Git 상태 확인

`$ git status`



```
MINGW64/c/Users/jin/Documents/html_project

jin@JIN-Desktop MINGW64 ~/Documents/html_project (main)
$ git add index.html

jin@JIN-Desktop MINGW64 ~/Documents/html_project (main)
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   index.html

jin@JIN-Desktop MINGW64 ~/Documents/html_project (main)
$ git restore --staged index.html

jin@JIN-Desktop MINGW64 ~/Documents/html_project (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:   index.html

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

- 작업 되돌리기
 - 최신 커밋 되돌리기

```
$ git reset HEAD^
```

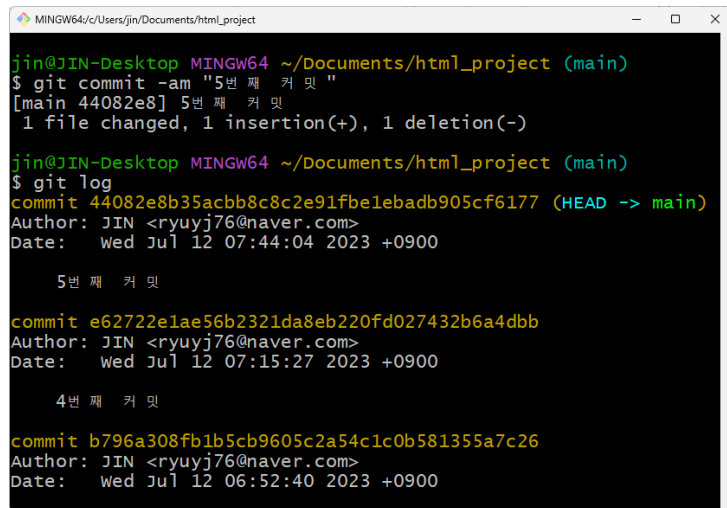
HEAD^는 현재 HEAD가 가리키는 브랜치의 최신 커밋을 가리킴 → 가장 최신 커밋에 (HEAD -> main) 기억
main 브랜치의 최신 커밋도 취소되고 스테이지에서도 삭제되어 작업 트리에만 파일이 남음

① index.html 파일 커밋

```
$ git commit -am "5번째 커밋"
```

② Commit 상태 확인

```
$ git log
```



```
jin@JIN-Desktop MINGW64 ~/Documents/html_project (main)
$ git commit -am "5번째 커밋"
[main 44082e8] 5번째 커밋
1 file changed, 1 insertion(+), 1 deletion(-)

jin@JIN-Desktop MINGW64 ~/Documents/html_project (main)
$ git log
commit 44082e8b35acbb8c8c2e91fbb1ebadb905cf6177 (HEAD -> main)
Author: JIN <ryuyj76@naver.com>
Date:   Wed Jul 12 07:44:04 2023 +0900

    5번째 커밋

commit e62722e1ae56b2321da8eb220fd027432b6a4dbb
Author: JIN <ryuyj76@naver.com>
Date:   Wed Jul 12 07:15:27 2023 +0900

    4번째 커밋

commit b796a308fb1b5cb9605c2a54c1c0b581355a7c26
Author: JIN <ryuyj76@naver.com>
Date:   Wed Jul 12 06:52:40 2023 +0900
```

- 작업 되돌리기

- 최신 커밋 되돌리기

- ③ 최신 커밋을 이전 상태로 되돌리기

- \$ git reset HEAD^

- ④ Commit 상태 확인

- \$ git log

```
MINGW64/c/Users/jin/Documents/html_project

jin@JIN-Desktop MINGW64 ~/Documents/html_project (main)
$ git reset HEAD^
Unstaged changes after reset:
M   index.html

jin@JIN-Desktop MINGW64 ~/Documents/html_project (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:   index.html

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

```
MINGW64/c/Users/jin/Documents/html_project

jin@JIN-Desktop MINGW64 ~/Documents/html_project (main)
$ git log
commit e62722e1ae56b2321da8eb220fd027432b6a4dbb (HEAD -> main)
Author: JIN <ryuyj76@naver.com>
Date:   Wed Jul 12 07:15:27 2023 +0900

    4번 째 커밋

commit b796a308fb1b5cb9605c2a54c1c0b581355a7c26
Author: JIN <ryuyj76@naver.com>
Date:   Wed Jul 12 06:52:40 2023 +0900

    3번 째 커밋
```

- 작업 되돌리기

- 특정 커밋으로 되돌리기

`$ git reset 해시`

- ① index3.html 파일 생성, 파일 커밋

`$ git commint -am "6번째 커밋"`

- ② index3.html 파일 수정, 파일 커밋

`$ git commint -am "7번째 커밋"`

- ③ index3.html 파일 수정, 파일 커밋

`$ git commint -am "8번째 커밋"`

```
MINGW64/c:/Users/jin/Documents/html_project

jin@JIN-Desktop MINGW64 ~/Documents/html_project (main)
$ git add index3.html

jin@JIN-Desktop MINGW64 ~/Documents/html_project (main)
$ git commit -m "6번째 커밋"
[main 2bf4816] 6번째 커밋
1 file changed, 8 insertions(+)
create mode 100644 index3.html

jin@JIN-Desktop MINGW64 ~/Documents/html_project (main)
$ git add index3.html

jin@JIN-Desktop MINGW64 ~/Documents/html_project (main)
$ git commit -m "7번째 커밋"
[main fe50ae5] 7번째 커밋
1 file changed, 1 insertion(+)

jin@JIN-Desktop MINGW64 ~/Documents/html_project (main)
$ git add index3.html

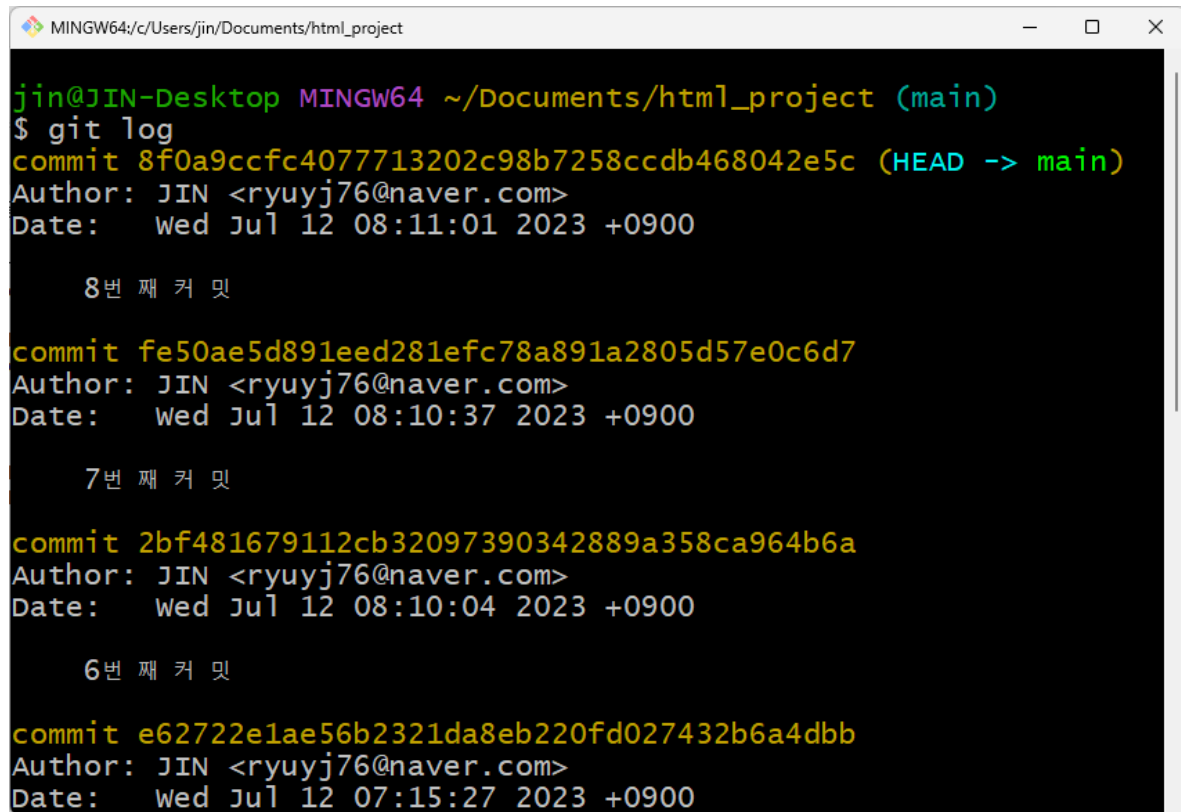
jin@JIN-Desktop MINGW64 ~/Documents/html_project (main)
$ git commit -m "8번째 커밋"
[main 8f0a9cc] 8번째 커밋
1 file changed, 1 insertion(+)
```

- 작업 되돌리기
 - 특정 커밋으로 되돌리기

\$ git reset 해시

④ Commit 로그 확인

\$ git log

A screenshot of a terminal window titled 'MINGW64/c/Users/jin/Documents/html_project'. The terminal shows the output of the 'git log' command. The output lists four commits in reverse chronological order. The first commit is '8f0a9ccfc4077713202c98b7258ccdb468042e5c (HEAD -> main)' by JIN <ryuyj76@naver.com> on Wed Jul 12 08:11:01 2023 +0900. The second commit is 'fe50ae5d891eed281efc78a891a2805d57e0c6d7' by JIN <ryuyj76@naver.com> on Wed Jul 12 08:10:37 2023 +0900. The third commit is '2bf481679112cb32097390342889a358ca964b6a' by JIN <ryuyj76@naver.com> on Wed Jul 12 08:10:04 2023 +0900. The fourth commit is 'e62722e1ae56b2321da8eb220fd027432b6a4dbb' by JIN <ryuyj76@naver.com> on Wed Jul 12 07:15:27 2023 +0900. Between the commit entries, there are Korean labels: '8번째 커밋' (8th commit), '7번째 커밋' (7th commit), and '6번째 커밋' (6th commit).

```
jinn@JIN-Desktop MINGW64 ~/Documents/html_project (main)
$ git log
commit 8f0a9ccfc4077713202c98b7258ccdb468042e5c (HEAD -> main)
Author: JIN <ryuyj76@naver.com>
Date:   wed Jul 12 08:11:01 2023 +0900

    8번째 커밋

commit fe50ae5d891eed281efc78a891a2805d57e0c6d7
Author: JIN <ryuyj76@naver.com>
Date:   wed Jul 12 08:10:37 2023 +0900

    7번째 커밋

commit 2bf481679112cb32097390342889a358ca964b6a
Author: JIN <ryuyj76@naver.com>
Date:   wed Jul 12 08:10:04 2023 +0900

    6번째 커밋

commit e62722e1ae56b2321da8eb220fd027432b6a4dbb
Author: JIN <ryuyj76@naver.com>
Date:   wed Jul 12 07:15:27 2023 +0900
```


- 작업 되돌리기
 - 특정 커밋으로 되돌리기

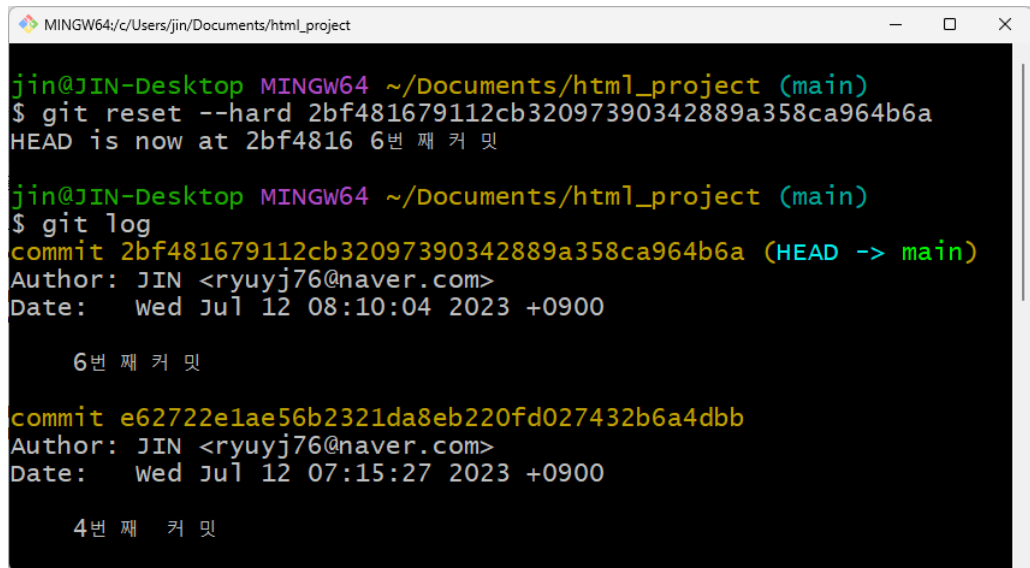
`$ git reset 해시`

- ④ “6번째커밋” 로그 메시지가 있는
커밋으로 리셋

`$ git reset --hard 복사한 커밋 해시`

- ⑤ Commit 로그 확인

`$ git log`



```
MINGW64~/c:/Users/jin/Documents/html_project
jin@JIN-Desktop MINGW64 ~/Documents/html_project (main)
$ git reset --hard 2bf481679112cb32097390342889a358ca964b6a
HEAD is now at 2bf4816 6번 째 커밋

jin@JIN-Desktop MINGW64 ~/Documents/html_project (main)
$ git log
commit 2bf481679112cb32097390342889a358ca964b6a (HEAD -> main)
Author: JIN <ryuyj76@naver.com>
Date:   Wed Jul 12 08:10:04 2023 +0900

    6번 째 커밋

commit e62722e1ae56b2321da8eb220fd027432b6a4dbb
Author: JIN <ryuyj76@naver.com>
Date:   Wed Jul 12 07:15:27 2023 +0900

    4번 째 커밋
```

- Git의 저장소 구조
 - 작업 폴더 (Working Directory), 스테이지 (Stage, Index), 저장소 (repository)
- Git 버전 생성 과정
 - 작업 폴더 생성 -> Git 초기화 -> 문서 생성/수정 -> 스테이지 영역에 저장 -> 커밋
- Commit 기록 확인 : `$ git log`
- 변경사항 확인 : `$ git diff`
- 단계별 파일 상태 확인 : tracked, untracked, unmodified, modified, staged
- 작업 되돌리기
 - 작업 트리에서 수정한 파일 되돌리기 : `$ git restore`
 - 스테이징 되돌리기 : `$ git restore -staged`
 - 최신 커밋 되돌리기 : `$ git reset HEAD^`
 - 특정 커밋으로 되돌리기 : `$ git reset 해시`