Open Source Software

버전 관리와 파일 상태

About..

컴퓨터소프트웨어공학과

김 원 일

기존 파일 버전 관리 – 1

- **파일 차이를 이용한 버전 관리**
  - 기존 코드 관리 프로그램들은 patch 형식으로 정보를 관리
  - 텍스트 패치 형식은 변화된 부분만을 별도 보관하는 형태
  - 개방형 OS에서는 diff 명령으로 간단히 patch 파일 생성이 가능
    - 2개의 파일을 비교하여 다른 점을 확인하는 프로그램

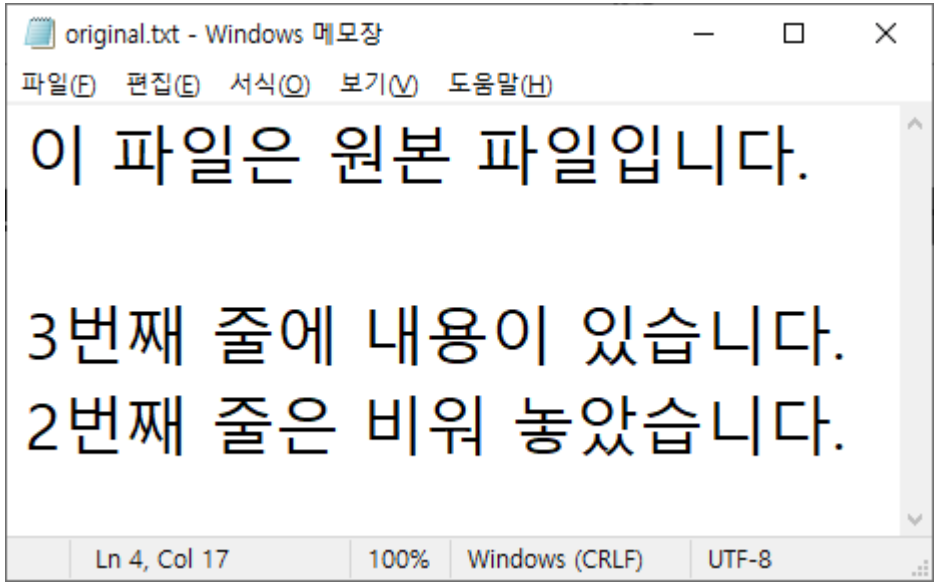
```
unangel@unangel: ~/work$ more test_a.txt
#include <stdio.h>
#include <stdlib.h>

int main( int argc, char **argv )
{
        printf( "Hello World₩n" );

        return 0;
}
```

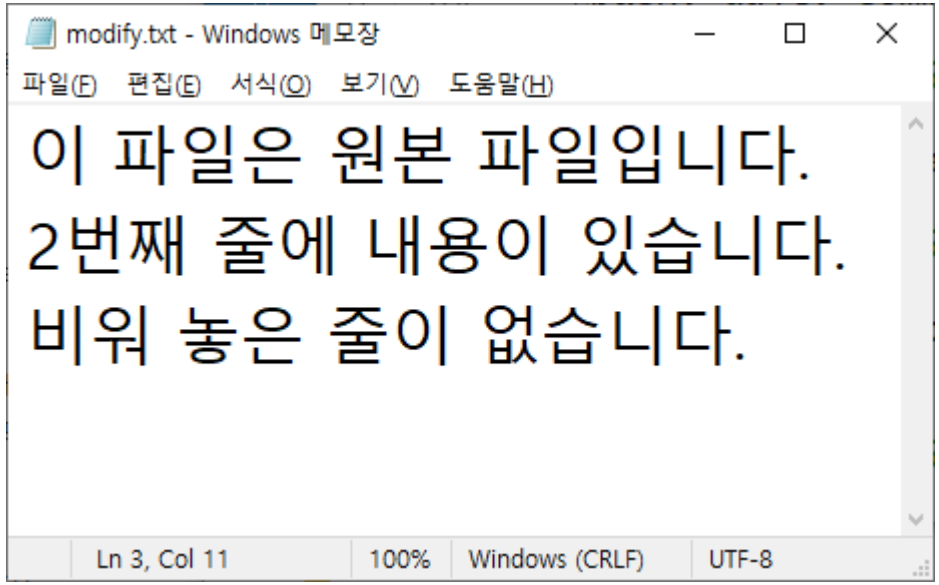
```
unangel@unangel: ~/work$ more test_b.txt
#include <stdio.h>

void main( void )
{
        printf( "hello world₩n" );

        return 0;
}
```

```
unangel@unangel: ~/work$ diff test_a.txt test_b.txt
2d1
< #include <stdlib.h>
4c3
< int main( int argc, char **argv )
---
> void main( void )
6c5
<       printf( "Hello World₩n" );
---
>       printf( "hello world₩n" );
unangel@unangel: ~/work$
```

기존 파일 버전 관리 – 2

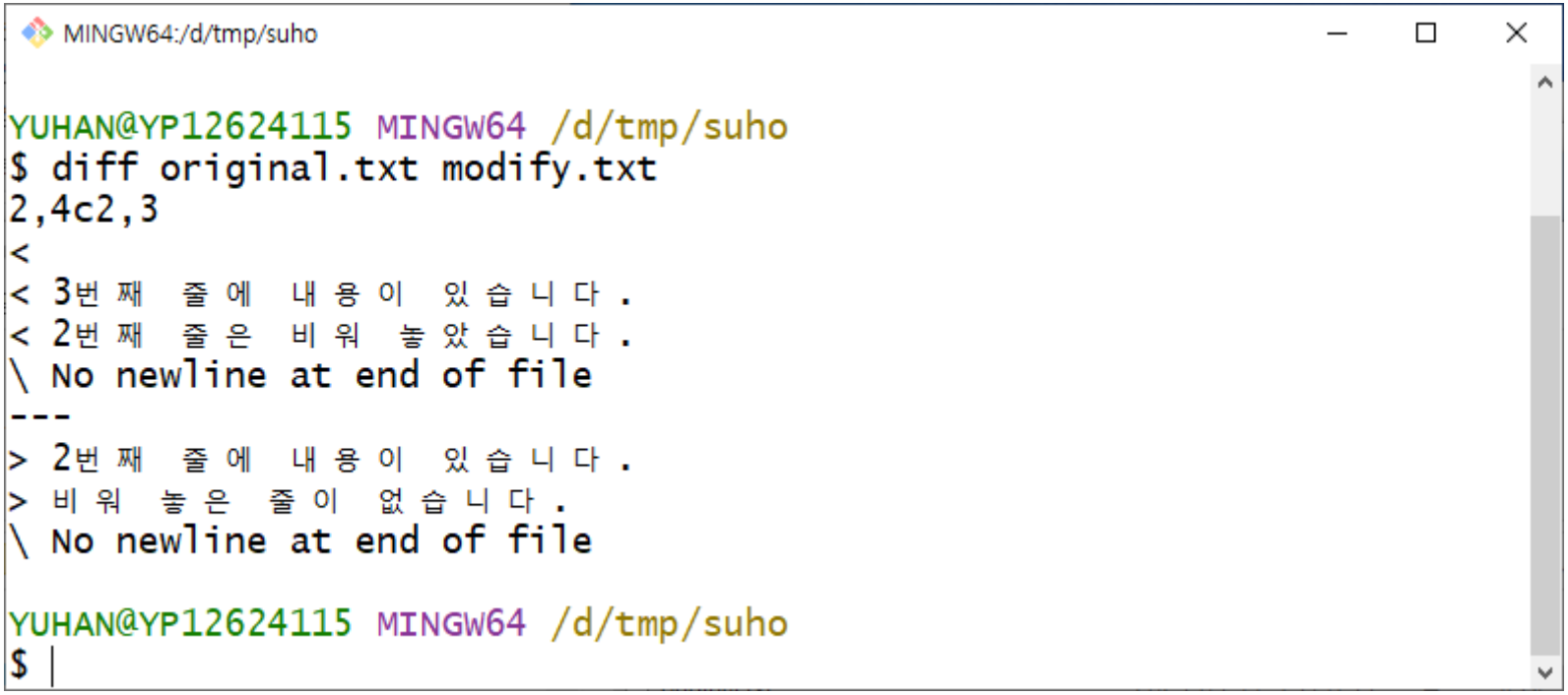
- **파일 차이점 확인**
  - **적당한 디렉터리를 선택하고 텍스트 파일 2개를 생성**
  - original.txt**와** modify.txt**에 같은 내용과 다른 내용을 입력**
  - **원본 파일과 수정한 파일을 서로 비교하여 확인**
  - bash**에서 확인 가능**





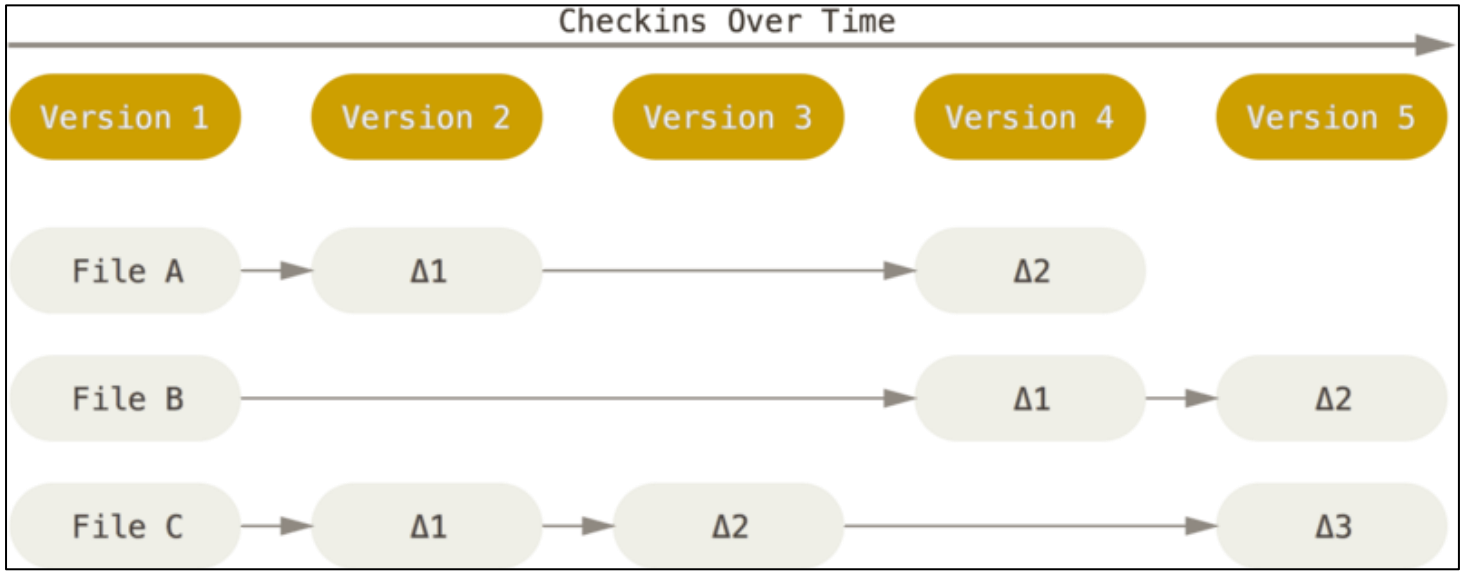
기존 파일 버전 관리 – 3

- **파일 차이점 확인**
  - "diff" 명령어로 파일의 차이점을 확인할 수 있음
  - "diff A B" 형태인 경우 각 파일에 존재하는 내용을 별도 출력
  - 동일한 내용에 대해서는 출력하지 않음
  - <는 왼쪽 파일, >는 오른쪽 파일에만 존재하는 내용을 출력



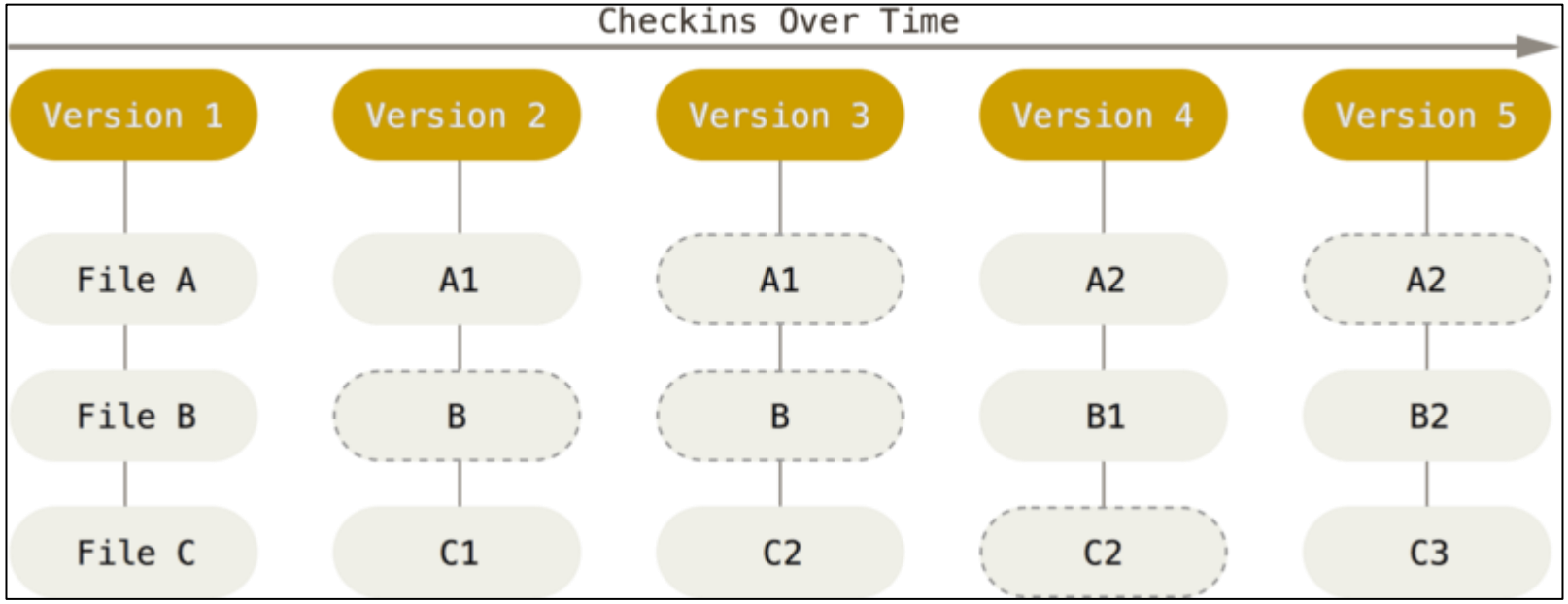
기존 파일 버전 관리 – 4

- **파일 목록의 관리**
  - **파일 변경을 patch 형태로 저장하여 사용하는 형태**
  - **파일 변화를 시간 순으로 관리하면서 파일 집합을 관리**
  - **델타 기반 버전 관리 시스템의 사용**
    - version 4를 가져올(Checkout) 경우
    - "File A"에 델타1 △1을 적용하고, △2를 적용하여 파일을 가져오는 형식
    - **원본 파일을 항상 보관한 상태에서 시작해야 함**



git을 이용한 파일 버전 관리 – 1

- **스냅샷을 이용한 버전 관리**
  - **스냅샷을 이용한 파일 관리는 변경 전 파일에 대한 정보만 링크로 사용**
  - **스냅샷 기반 버전 관리 시스템의 사용**
    - Version 4**를 가져올**(Checkout) **경우**
    - A2, B1, C2 **파일을 그대로 가져오기만 하면 문제 해결**
    - C2 **파일은 링크로 구성되고, 변경이 발생하면** C3**로 저장하면 완료**



git을 이용한 파일 버전 관리 – 2

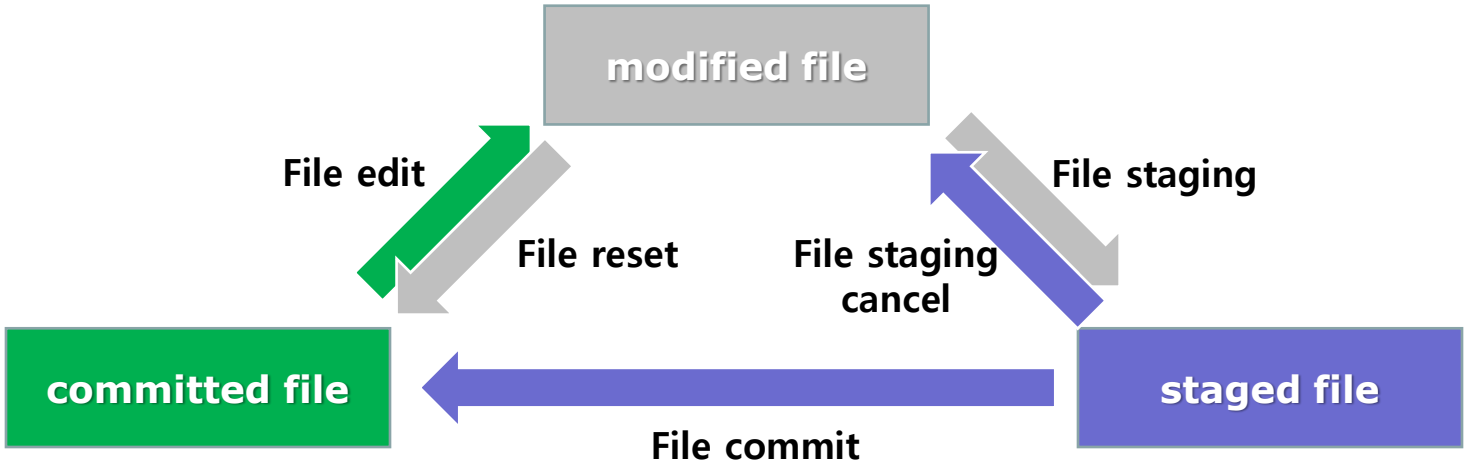
- **저장소**(Repository)
  - 말 뜻 그대로 **파일과 프로젝트를 보관**하는 곳
  - 일반적으로 프로젝트나 관리할 파일이 포함된 디렉터리
  - **디렉터리저장소 관리 단위**는 기본적으로 디렉터리 단위
  - 저장소에서 발생하는 파일 변화를 git에서 **추적**하도록 구성
  - **git이 파일 이력을 추적하도록 구성된 디렉터리**를 의미
  - 디렉터리 내에는 관리를 위한 **데이터베이스가 자동 생성**
  - 보통 내 컴퓨터에 있는 저장소를 "**로컬 저장소**"라 지칭함
  - 저장소 내용은 데이터베이스 형태로 구성되어 관리
  - 사용자는 데이터베이스 관련 처리를 할 필요는 없음
  - git 소프트웨어가 데이터베이스 관련 처리를 수행

git을 이용한 파일 버전 관리 – 3

- 저장소의 모든 정보를 모두가 가짐
  - git을 통한 복제는 원본을 그대로 복제하여 로컬에 저장
    - 서버에 문제가 발생해도 모든 복제 본에 파일 이력과 변경 정보가 보관되는 형태
    - 하나의 복제 본만 존재해도 서버를 다시 구축할 수 있음
    - 파일 이력에 대한 데이터베이스 정보를 모두가 동일하게 보관
    - 서버에 업로드가 된 경우에 한함
  - git은 로컬 기반으로 데이터를 관리하는 것이 기본 설정
    - 로컬에 모두 복제되므로, 이전 파일 정보도 로컬에서 즉시 확인 가능
    - CVS와 같은 시스템은 이전 파일 정보 확인을 위해 서버에 반드시 접속해야 함
  - 원격지 접근은 필요한 경우에만 접근
    - 서버 업데이트를 제외하고, 모든 작업을 로컬에서 수행할 수 있음
    - 협업 정보의 교환 전에는 굳이 서버에 업로드하지 않아도 문제 없음
  - 무결성 지원
    - 스냅샷을 작성할 때, 모든 파일에 대한 체크섬을 구하여 데이터를 관리
    - 체크섬은 SHA-1 해시를 이용하여 생성 (40자 길이의 16진수 문자열)

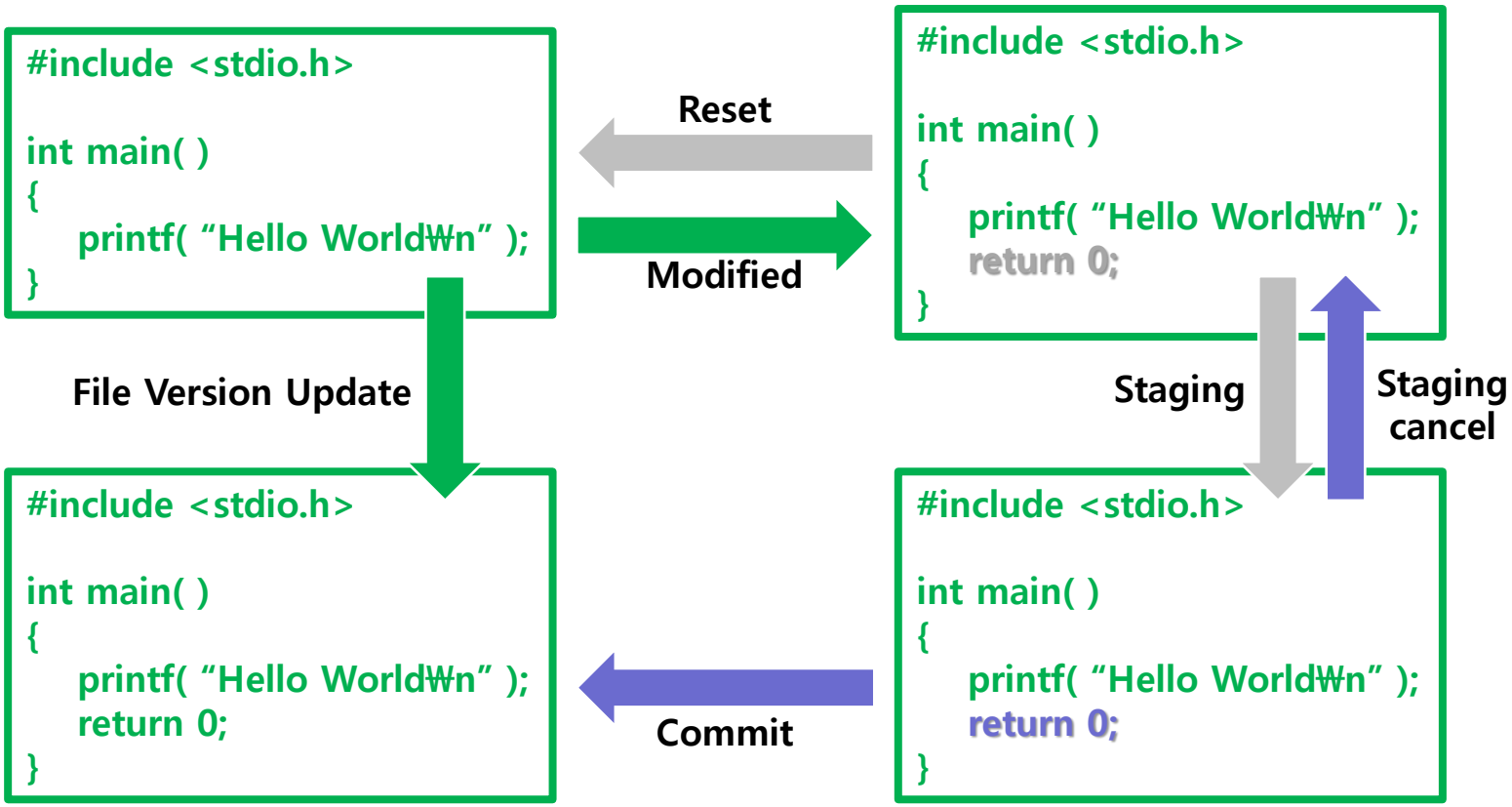
git을 이용한 파일 버전 관리 – 4

- **파일 상태 – 1**
  - **committed** : git에서 파일을 안전하게 관리 및 추적하고 있는 상태
    - 수정 발생 전으로 언제든지 복원이 가능
  - **modified** : 수정이 발생했으나, 수정 내용이 아직 관리되지 않는 상태
    - 수정 내용이 데이터베이스에 적용되지 않았음
  - **staged** : 수정 파일이 곧 committe될 예정인 상태
    - 데이터베이스에 바로 적용이 가능한 상태로, 추가적인 파일 수정이 불가능한 상태
    - 취소를 통해 파일을 다시 수정할 수 있음



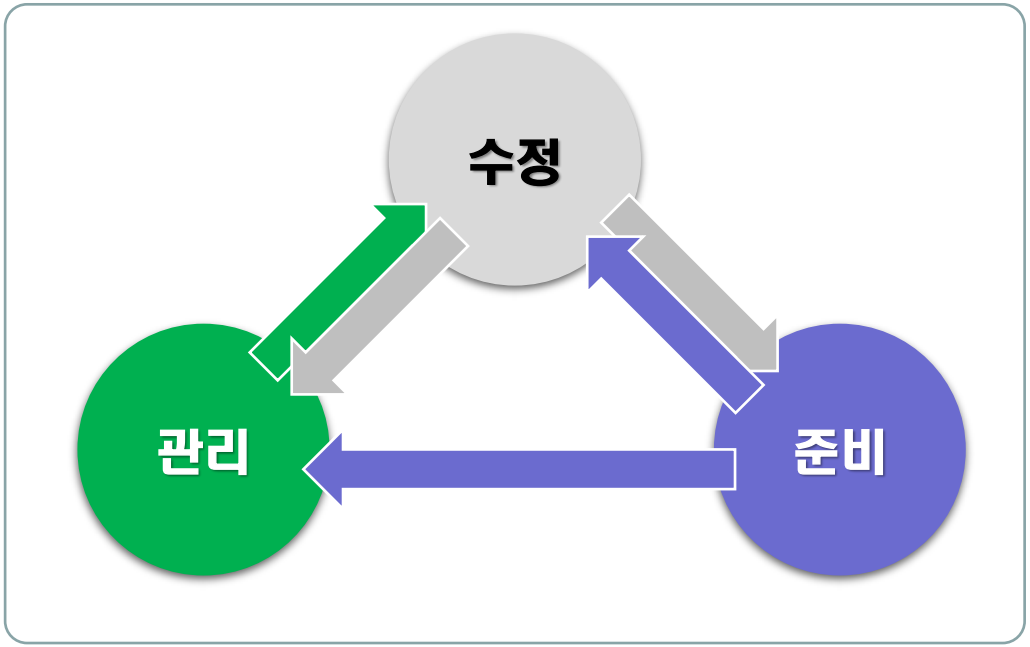
git을 이용한 파일 버전 관리 – 5

- **파일 상태** – 2



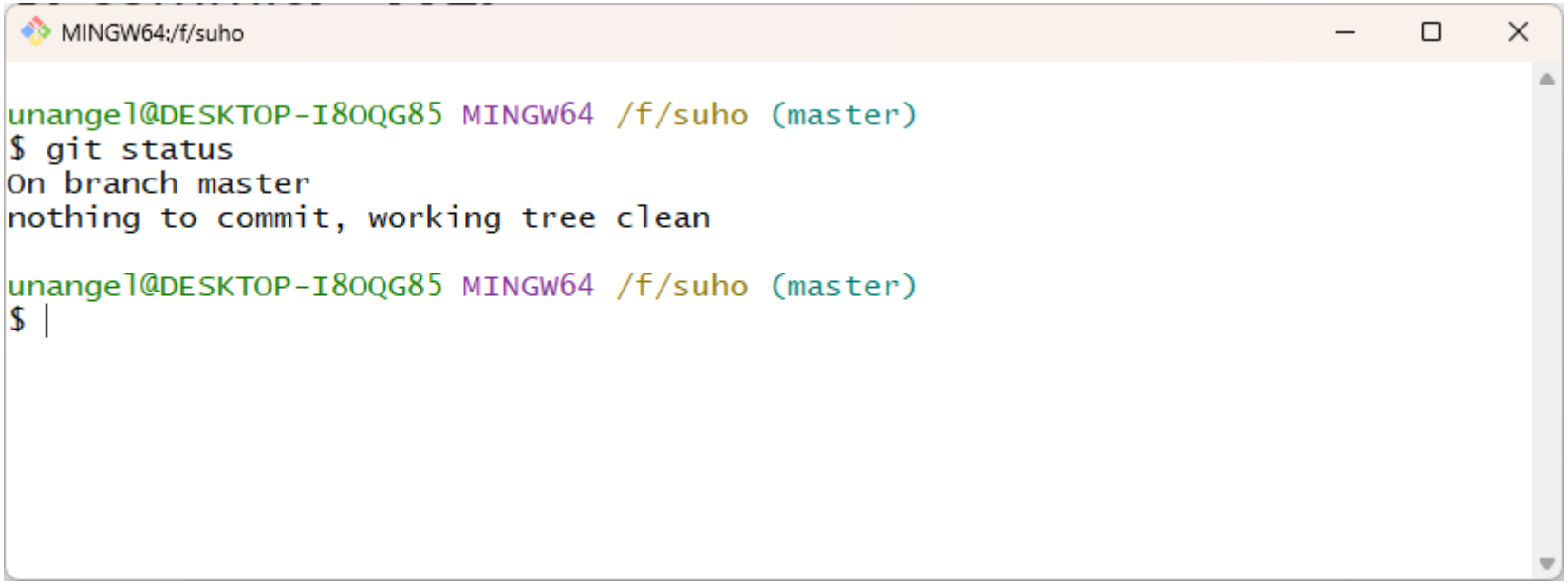
저장소 상태 – 1

- git **저장소의 기본 상태**
  - 기본적인 저장소의 상태는 3가지 상태로 구분
  - 관리 상태 : 파일이 모두 관리되고 있는 상태
  - 수정 상태 : 파일이 수정되어 관리 상태와 현재 상태가 다른 상태
  - 준비 상태 : 파일 수정이 완료된 상태로 관리 상태로 변경 가능한 상태



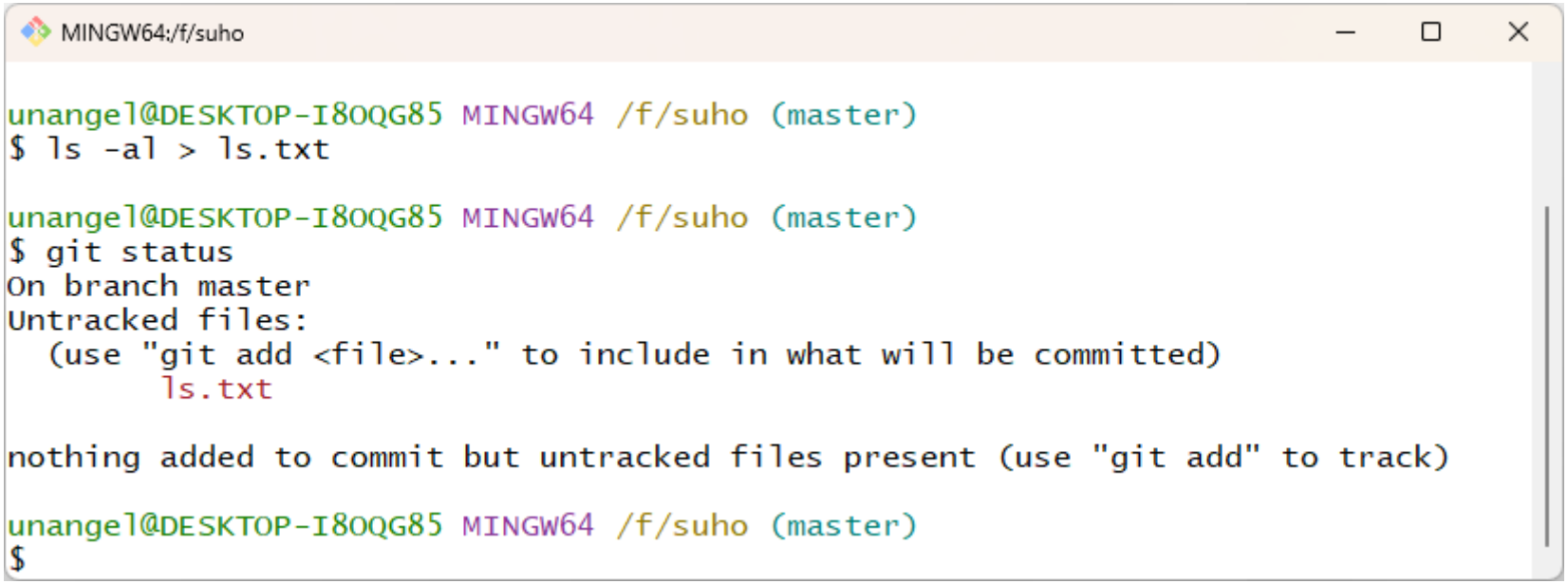
저장소 상태 – 2

- **관리**(Committed) **상태**
  - **서버로 부터 파일들을 복제한 상태 또는 저장소를 최초 생성한 상태**
  - **모든 파일의 수정 내용이** commit**된 상태**
  - **파일들을 모두 로컬 데이터베이스에 기록한 상태를 나타냄**
  - **관리 상태의 파일들은** "git status" **명령어에서 출력되지 않음**
  - **작업 공간에** commit**할 내용이 없고 수정도 없는 상태**



저장소 상태 – 3

- **수정**(Modified) **상태**
  - 관리 상태에서 하나의 파일이라도 수정이 발생한 상태
  - 데이터베이스 원본 상태와 다른 상태를 갖는 파일이 발생
  - 수정 내용은 원본 파일과 관계 없이 별도 수정 파일로 관리됨
  - 데이터베이스에 기록된 원본은 수정 파일과 별도의 파일로 인식
    - 사용자에게는 별도 파일이지만 하나의 파일처럼 보임
  - "git status" **명령어 실행에서 붉은 색으로 나타나는 파일**



저장소 상태 – 4

- **준비**(Staged) **상태**
  - **수정된 파일을 로컬 데이터베이스에 기록하기 위한 상태**
  - **파일이 읽기 전용의 속성을 가지게 되어 수정되지 않음**
  - **읽기 전용 파일이라도 수정은 가능(?)하므로 주의**
  - git add **명령어로 준비 상태로 상태를 변경할 수 있음**
  - git status **명령어 실행에서 녹색으로 나타남**

```
MINGW64:/f/suho
unangel@DESKTOP-I8OQG85 MINGW64 /f/suho (master)
$ git add ls.txt
warning: in the working copy of 'ls.txt', LF will be replaced by CRLF the next t
ime Git touches it

unangel@DESKTOP-I8OQG85 MINGW64 /f/suho (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   ls.txt


unangel@DESKTOP-I8OQG85 MINGW64 /f/suho (master)
$
```

Open Source Software

로컬 저장소 기초 실습

About..

컴퓨터소프트웨어공학과

김 원 일

로컬 실습 환경 - 1

- 실습할 디렉터리 선택
  - 컴퓨터의 적절한 디렉터리를 선택 및 생성하여 실습 준비
  - 디렉터리 내에는 별도의 파일이 존재하지 않도록 준비
  - 최상위 디렉터리는 모든 파일이 포함되므로 사용하지 않음
  - 터미널 명령어로 새로운 디렉터리를 생성하는 것도 방법
  - 생성한 디렉터리로 이동하여 내용 확인

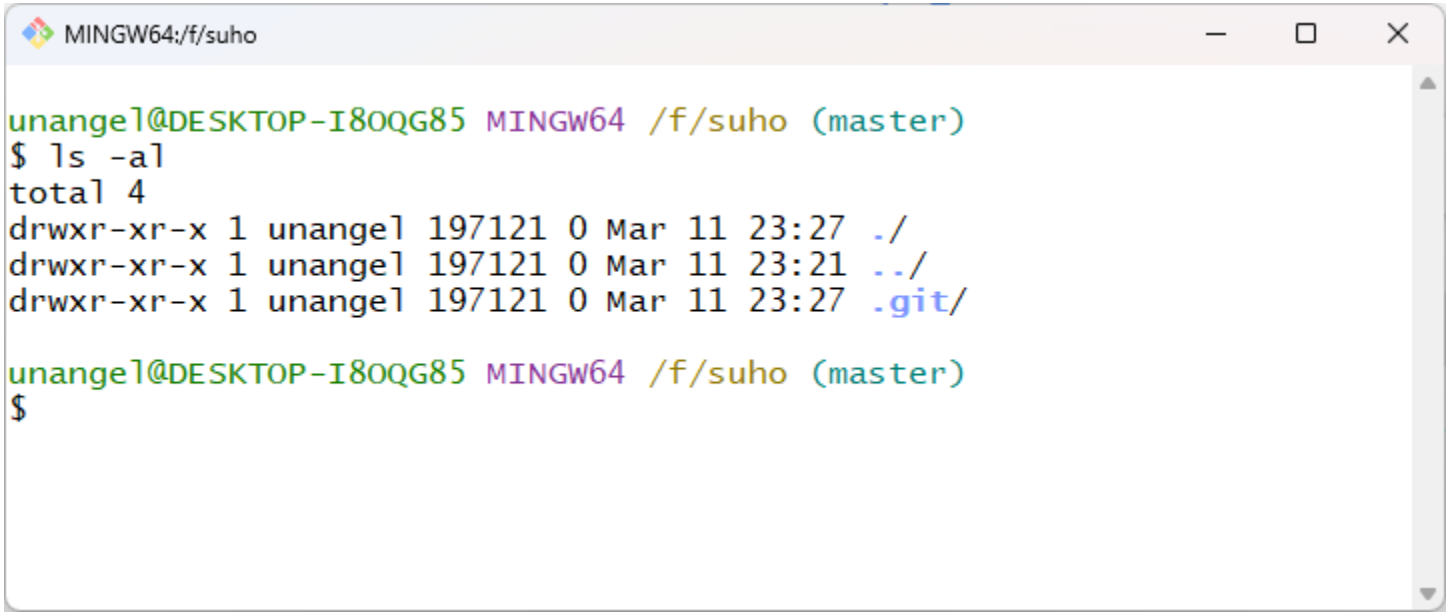
```
MINGW64:/f/suho

unangel@DESKTOP-I8OQG85 MINGW64 /f
$ mkdir suho

unangel@DESKTOP-I8OQG85 MINGW64 /f
$ cd suho/

unangel@DESKTOP-I8OQG85 MINGW64 /f/suho
$ ls -al
total 4
drwxr-xr-x 1 unangel 197121 0 Mar 11 23:19 ./
drwxr-xr-x 1 unangel 197121 0 Mar 11 23:19 ../

unangel@DESKTOP-I8OQG85 MINGW64 /f/suho
$
```

로컬 실습 환경 – 2

- **저장소 만들기**
  - 생성한 디렉터리 내부에서 저장소 생성 명령어를 실행
  - "git init" 명령어로 현재(./) 디렉터리를 저장소로 설정
  - 저장소는 파일/디렉터리를 git이 관리하는 디렉터리를 지칭
  - 관리되는 저장소가 되면 이력 관리가 시작됨

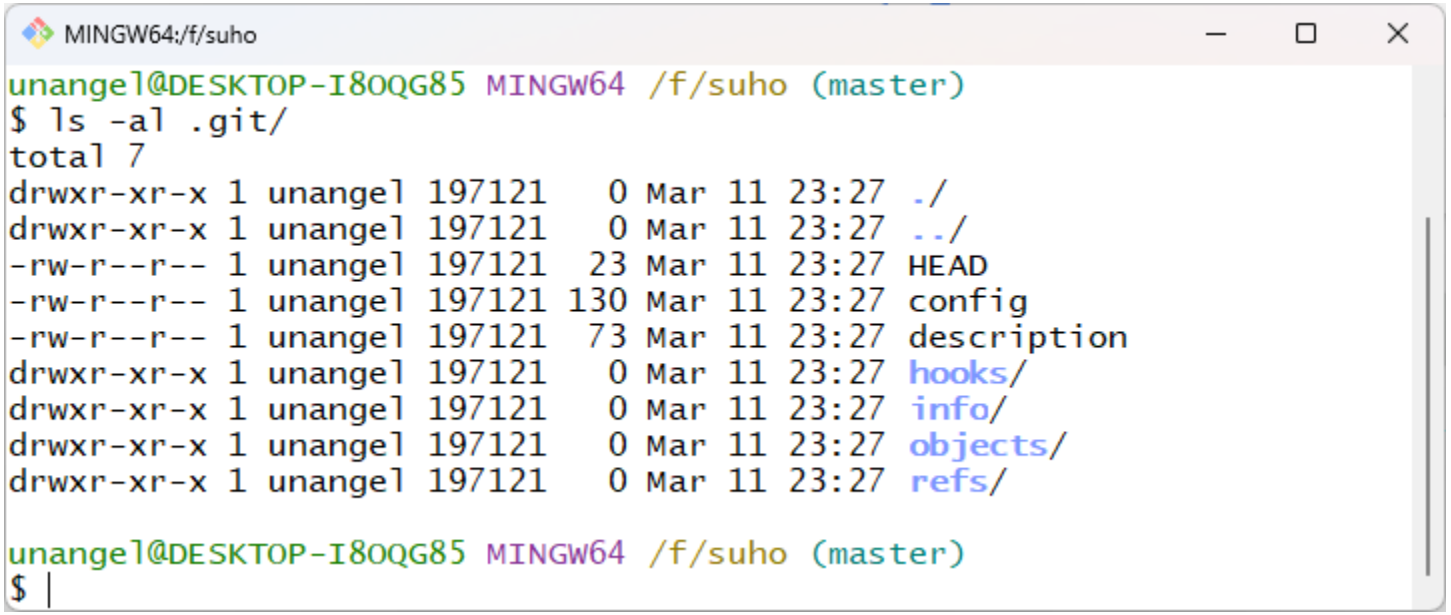
로컬 실습 환경 – 3

- **저장소 확인**
  - **저장소가 되면 디렉터리에 ".git/" 디렉터리가 생성됨**
  - **디렉터리 설정이 숨김이기 때문에 보이지 않을 수 있음**
  - **터미널의 경로 다음에 "(master)"와 같은 추가 정보가 출력됨**
  - bash **프로그램에서 항상 저장소 여부를 확인할 수 있음**



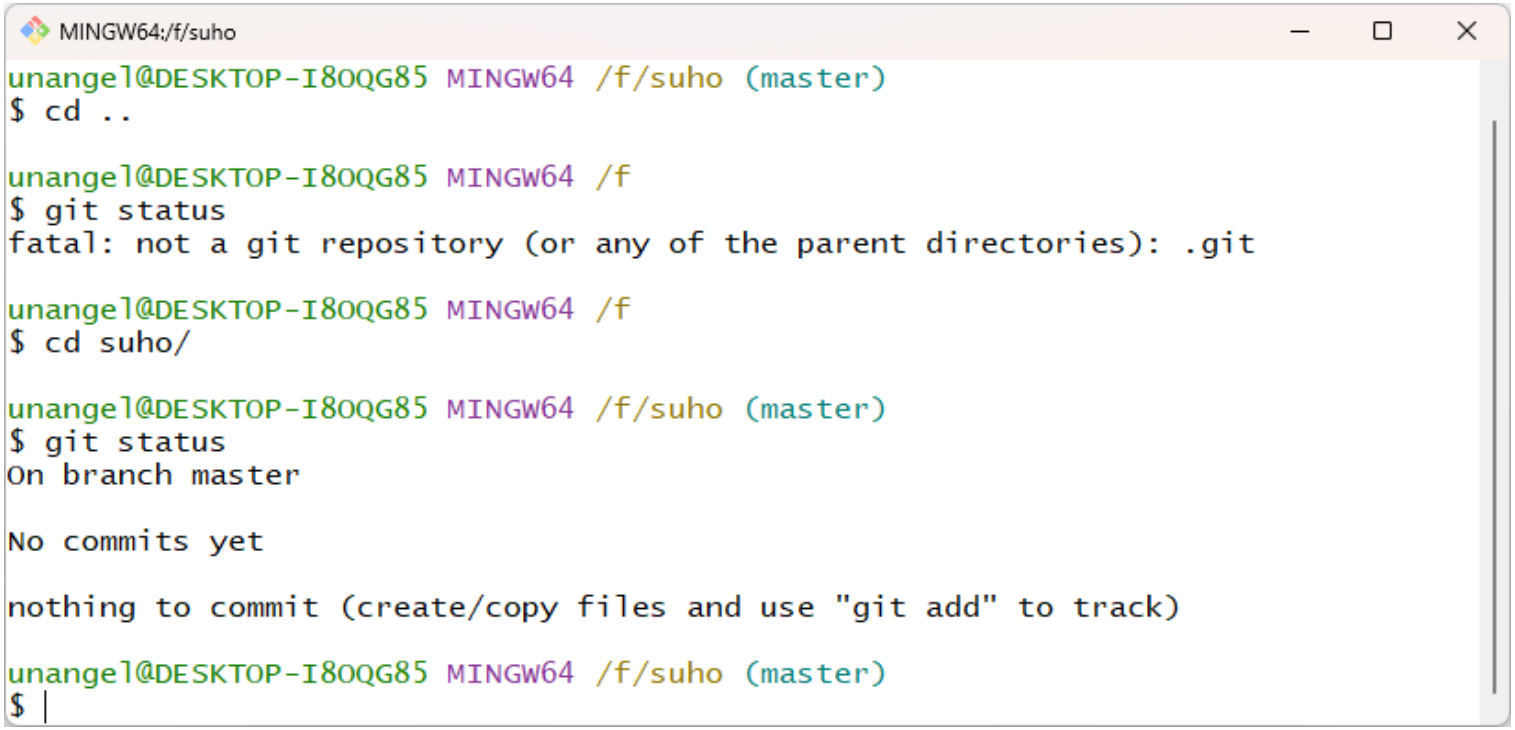
로컬 실습 환경 – 4

- "**.git/" 디렉터리의 내용**
  - **디렉터리에 파일 관리를 위한 정보들이 포함되어 있음**
  - **현재 상태에서는 정보가 의미하는 내용을 알기 어려움**
  - **추가적인 수업을 진행한 다음 내부 내용에 대한 확인 진행 예정**
  - **디렉터리가 "."으로 시작하는 이유는 리눅스 숨김 파일 지정 방법**



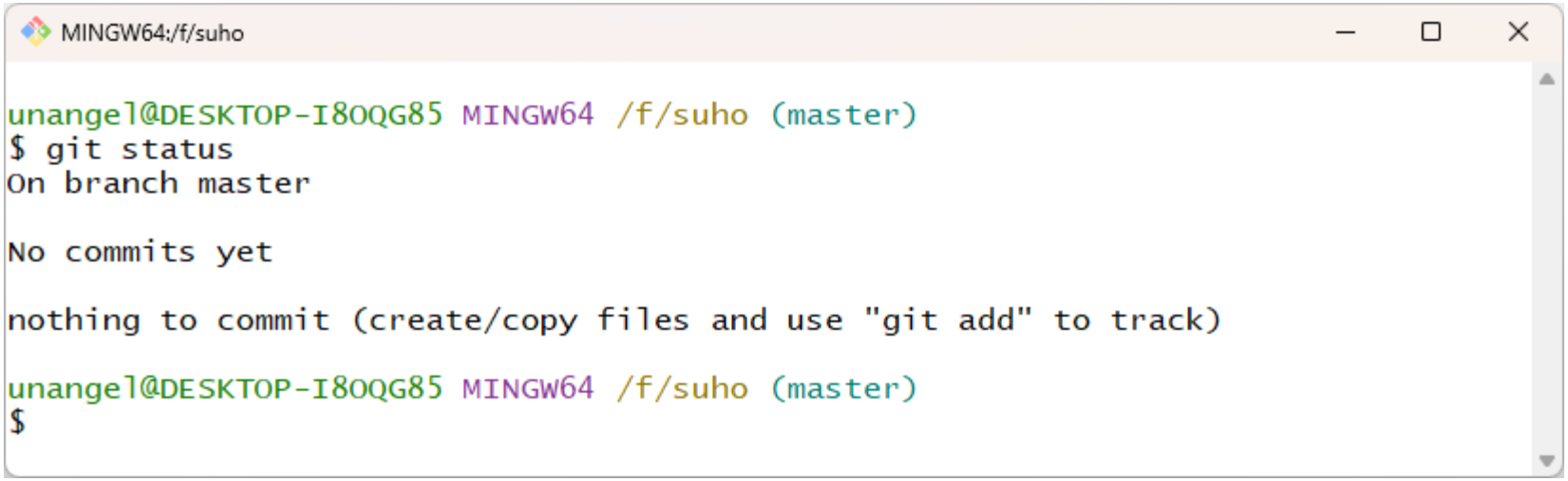
git 로컬 저장소 실습 – 1

- **저장소 확인 명령어**
  - 저장소가 아닌 위치에서 "git status" 명령어 실행
  - 저장소에서 발생한 변화를 확인할 때 사용
  - 저장소가 아닌 디렉터리 : 저장소가 아니라는 오류 출력이 발생
  - 저장소 내부 : 저장소의 현재 상태 정보를 출력



git 로컬 저장소 실습 – 2

- "git stauts" **명령어**
  - **저장소의 현재 상태를 출력하는 간단한 명령어**
  - **주의 : 저장소 이외의 디렉터리에서는 정상적으로 동작하지 않음**
  - **모든 git 명령어는 저장소 이외에서 정상적으로 동작하지 않음**
  - **저장소의 브렌치 정보와 수정, 준비 상태의 파일 정보를 출력**
    - **브렌치는 추후 관련 강의에서 진행할 예정**
  - **추가적으로 현재 상태에서 사용 가능한 명령어 제시**



git 로컬 저장소 실습 – 3

- **파일 생성**
  - 저장소에 새로운 파일을 파이프를 이용하여 생성
  - 새롭게 생성된 파일과 파일의 내용을 확인

MINGW64:/f/suho

```
unangel@DESKTOP-I8OQG85 MINGW64 /f/suho (master)
$ ls -al
total 8
drwxr-xr-x 1 unangel 197121 0 Mar 19 19:31 ./
drwxr-xr-x 1 unangel 197121 0 Mar 14 18:12 ../
drwxr-xr-x 1 unangel 197121 0 Mar 18 23:28 .git/

unangel@DESKTOP-I8OQG85 MINGW64 /f/suho (master)
$ ls -al > list.txt

unangel@DESKTOP-I8OQG85 MINGW64 /f/suho (master)
$ cat list.txt
total 8
drwxr-xr-x 1 unangel 197121 0 Mar 19 19:32 ./
drwxr-xr-x 1 unangel 197121 0 Mar 14 18:12 ../
drwxr-xr-x 1 unangel 197121 0 Mar 18 23:28 .git/
-rw-r--r-- 1 unangel 197121 0 Mar 19 19:32 list.txt

unangel@DESKTOP-I8OQG85 MINGW64 /f/suho (master)
$
```

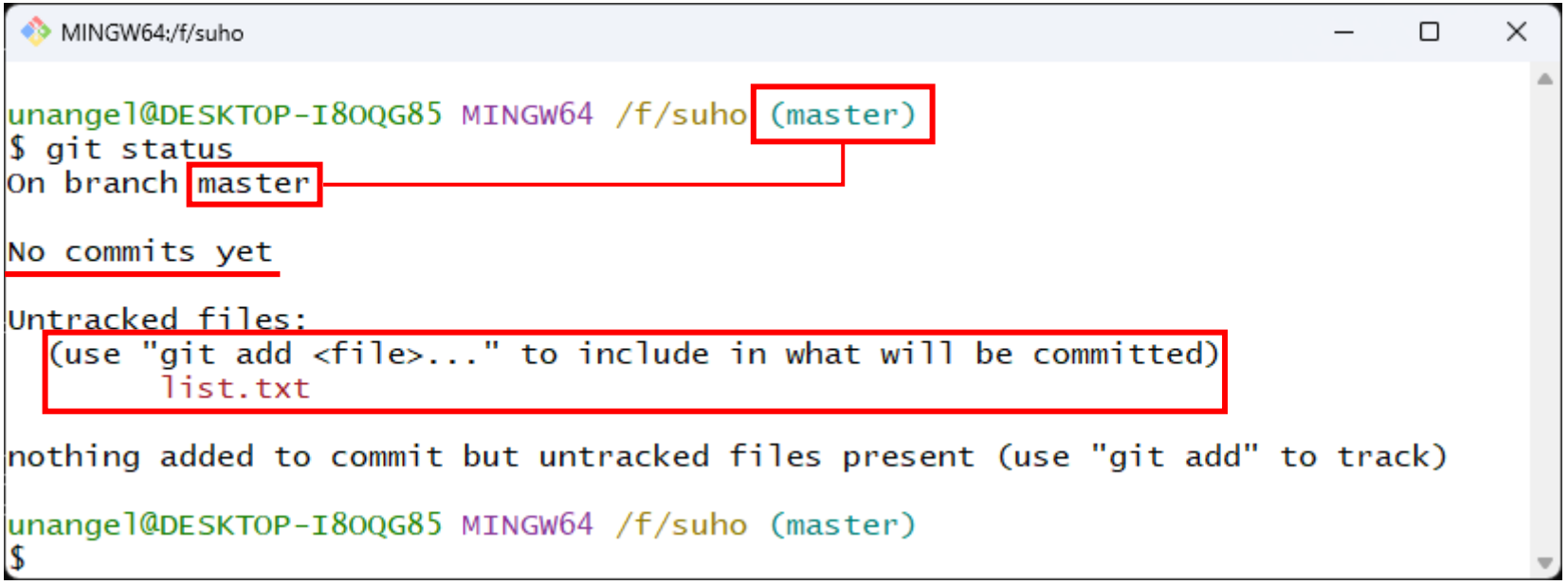
git 로컬 저장소 실습 – 4

- **저장소 정보 확인** – 1
  - **파일이 새롭게 생성되었기 때문에 저장소 상태 확인이 필요**
  - **브렌치에** commit**은 없으나 새로운 파일이 생겼음을 확인**
  - **새로운 파일들은** "Untracked files" **항목으로 나타남**
  - commit**할 내용은 없으나 추적되지 않는 파일이 존재함을 출력**
  - **추적되지 않는 파일을 추적할 수 있는 상태로 변경하는 명령어 출력**

```
MINGW64:/f/suho

unangel@DESKTOP-I8OQG85 MINGW64 /f/suho (master)
$ git status
On branch master

No commits yet

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

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

unangel@DESKTOP-I8OQG85 MINGW64 /f/suho (master)
$
```

git 로컬 저장소 실습 – 3

- **저장소 정보 확인** – 2
  - **브렌치가 동일하게 출력, 저장소 생성 후 아직 커밋이 없었음**
  - "Untracked files" : **새롭게 생성된 파일이나 관리하지 않음**
  - **현재 관리되지 않거나 수정이 발생한 파일은 붉은색으로 나타남**
  - git**은 항상 현재 상황과 필요한 정보를 메시지로 전달**



git 로컬 저장소 실습 – 4

- **생성된 파일을 준비 상태로 변경**
  - "git add" **명령어로 파일을 준비 상태로 변경**
  - **준비 상태로 변경된 파일은 읽기 전용으로 변경**
  - **변경된 파일은 녹색으로 나타나며 새로운 파일임을 출력**
  - **추가 메시지로 준비 상태를 취소할 수 있는 명령어도 출력**

```
MINGW64:/f/suho
unangel@DESKTOP-I8OQG85 MINGW64 /f/suho (master)
$ git add list.txt
warning: in the working copy of 'list.txt', LF will be replaced by CRLF the next
 time Git touches it

unangel@DESKTOP-I8OQG85 MINGW64 /f/suho (master)
$ git status
On branch master

No commits yet

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


unangel@DESKTOP-I8OQG85 MINGW64 /f/suho (master)
$
```

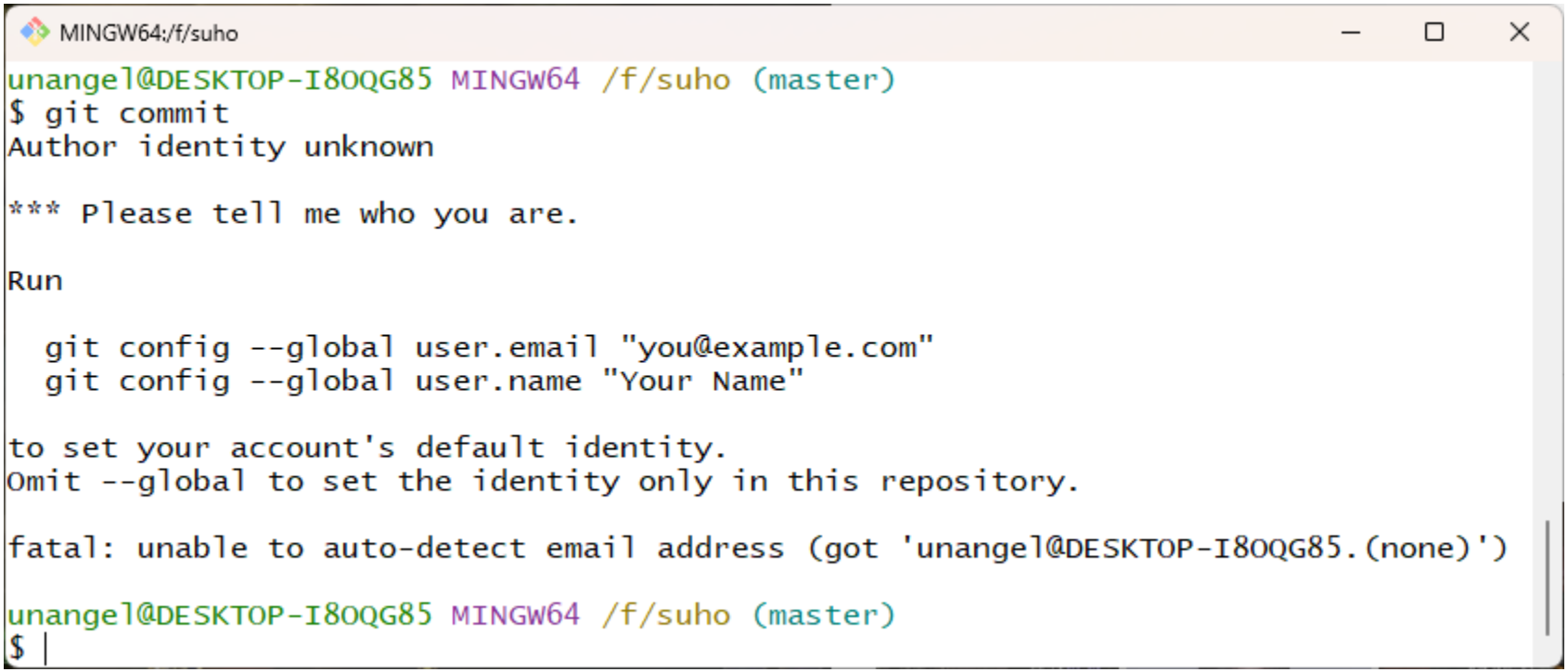
git 로컬 저장소 실습 – 5

- **준비 상태의 취소**
  - "git rm --cached <파일명>" 으로 준비 상태 취소
  - 지정한 파일이 다시 붉은색으로 변경된 것을 확인할 수 있음
  - "git add" 명령으로 준비 상태로 보내기 전과 동일 상태
  - commit으로 원본을 교체하기 전에는 자유롭게 상태 변경이 가능

```
MINGW64:/f/suho
unangel@DESKTOP-I8OQG85 MINGW64 /f/suho (master)
$ git rm --cached list.txt
rm 'list.txt'

unangel@DESKTOP-I8OQG85 MINGW64 /f/suho (master)
$ git status
On branch master

No commits yet

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

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

unangel@DESKTOP-I8OQG85 MINGW64 /f/suho (master)
$
```

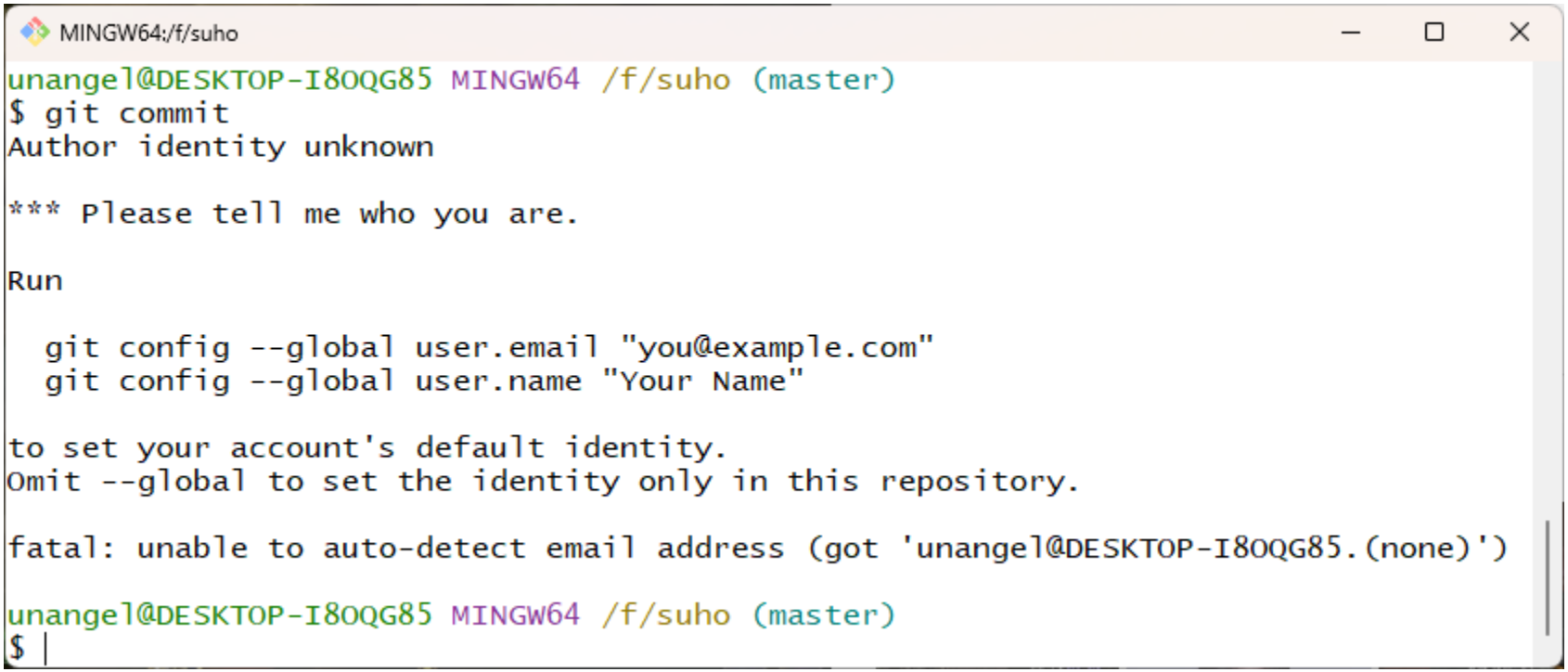
git 로컬 저장소 실습 – 6

- **최초** commit **실행** – 1
  - "git add"**로 파일을 다시 준비 상태로 변경**
  - **변경된 상태에서** "git commit" **명령을 실행**
  - git **소프트웨어 설치 후, 최초** commit **명령어 사용시 실행되지 않음**
  - **저장소에 저작자를 등록하는 작업을 먼저 진행해야 함**



git 로컬 저장소 실습 – 7

- **최초 commit 실행 – 2**
  - **저작자 등록은 전자메일과 이름을 등록하는 것으로 완료**
  - **출력된 메시지와 동일하게 입력하는 것으로 등록 완료**
  - **"git config" 명령은 저장소 설정을 변경하거나 확인하는 명령어**
  - **본인의 이름/닉네임/이니셜과 본인 메일 주소로 대체하여 입력**

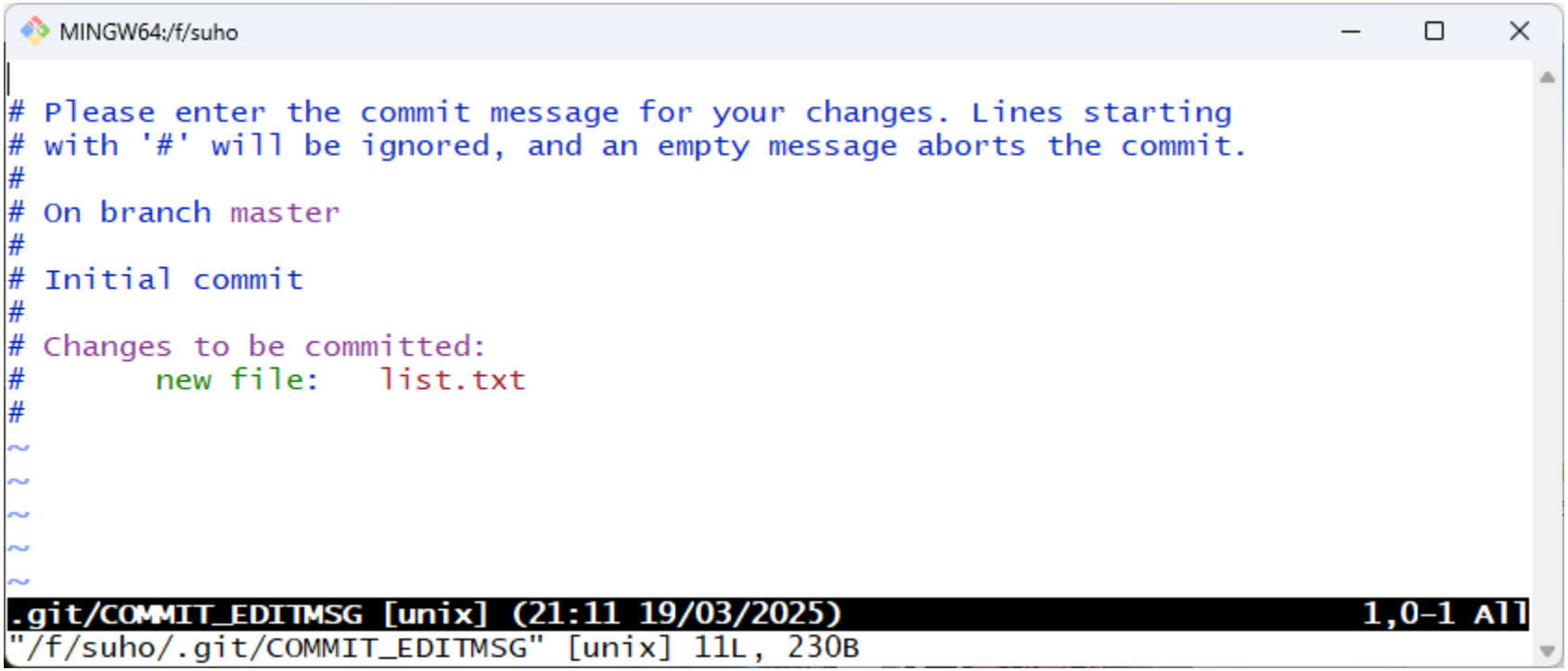


git 로컬 저장소 실습 – 8

- **최초** commit **실행** – 3
  - **설정이 기본적으로 전역으로 설정되므로 주의 필요**
  - user.email/name **뒤에 내용을 입력하지 않으면 현재 설정을 출력**
  - **이미 입력된 설정이 맞지 않는 경우에는 맞는 정보로 수정**
  - **다양한 정보 확인이 필요한 경우라면** "git config --help"**로 확인**

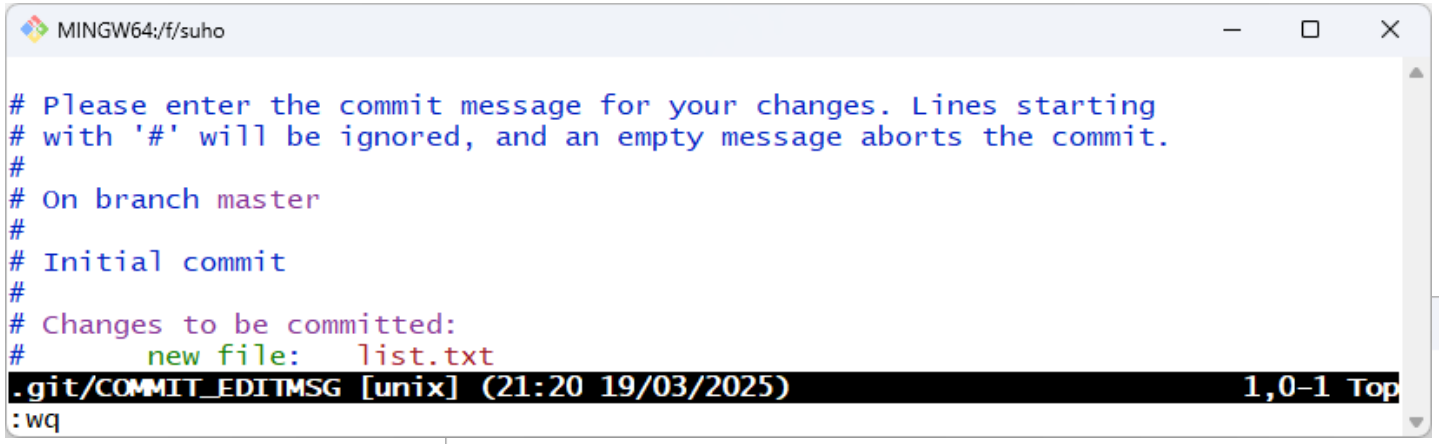
git 로컬 저장소 실습 – 9

- commit **메시지 입력** – 1
  - **로컬 데이터베이스에 파일을 등록하려면 메시지 입력을 요청**
  - **메시지는 수정 내용이나 주요 변경/추가 사항에 대한 내용을 기록**
  - **설치에서 Vim을 기본 에디터로 선택했기 때문에 Vim이 실행**
  - **하단부에 파일 정보가 출력. #으로 시작된 줄은 주석으로 처리**



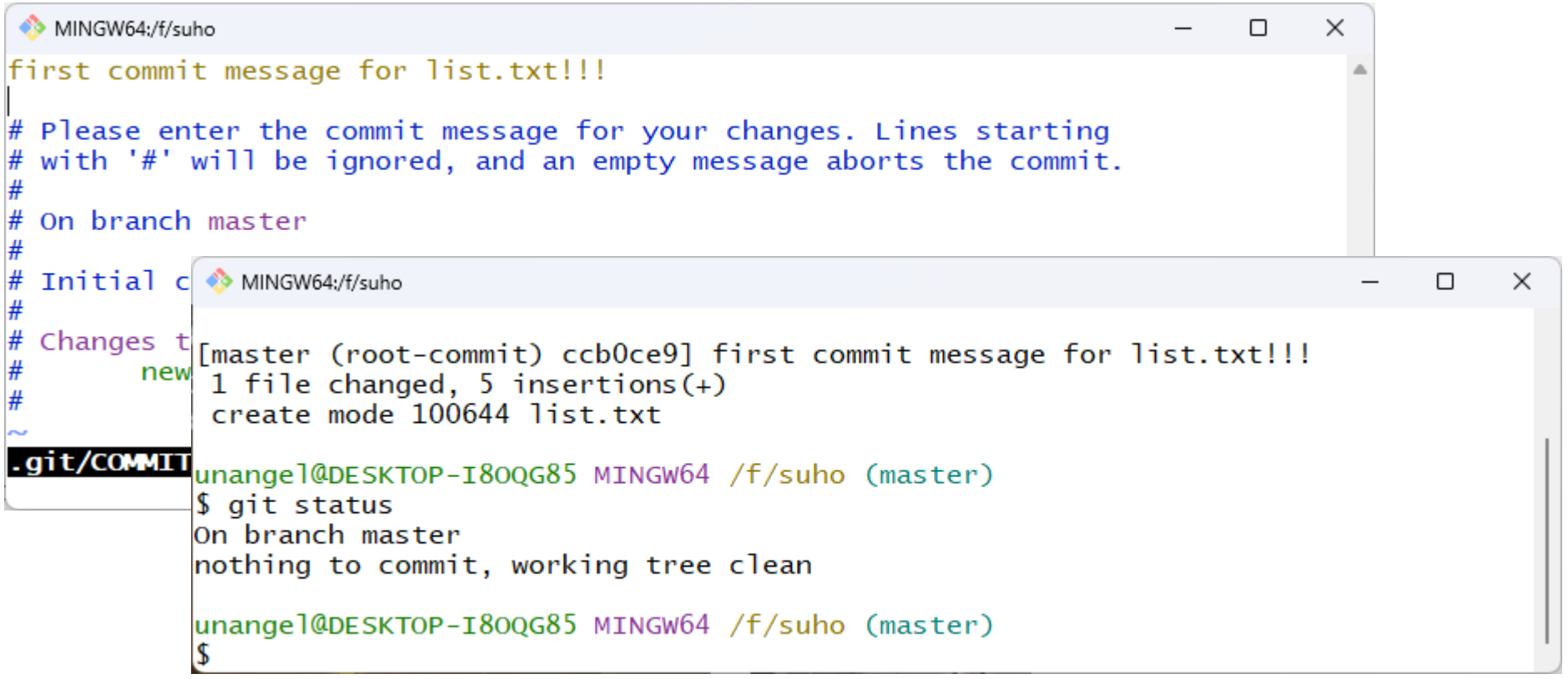
git 로컬 저장소 실습 – 10

- commit **메시지 입력** – 2
  - ESC**를 두 번 정도 입력하면 화면의 깜빡임이 발생**
  - **이때 "**:**"(**Shift + ;**)을 입력하면 하단에 명령 입력이 가능**
  - **"**wq**"를 입력하고 엔터 키를 입력하면** Vim**에서 빠져 나옴**
  - **메시지를 입력하지 않아** commit **이 처리되지 않음**



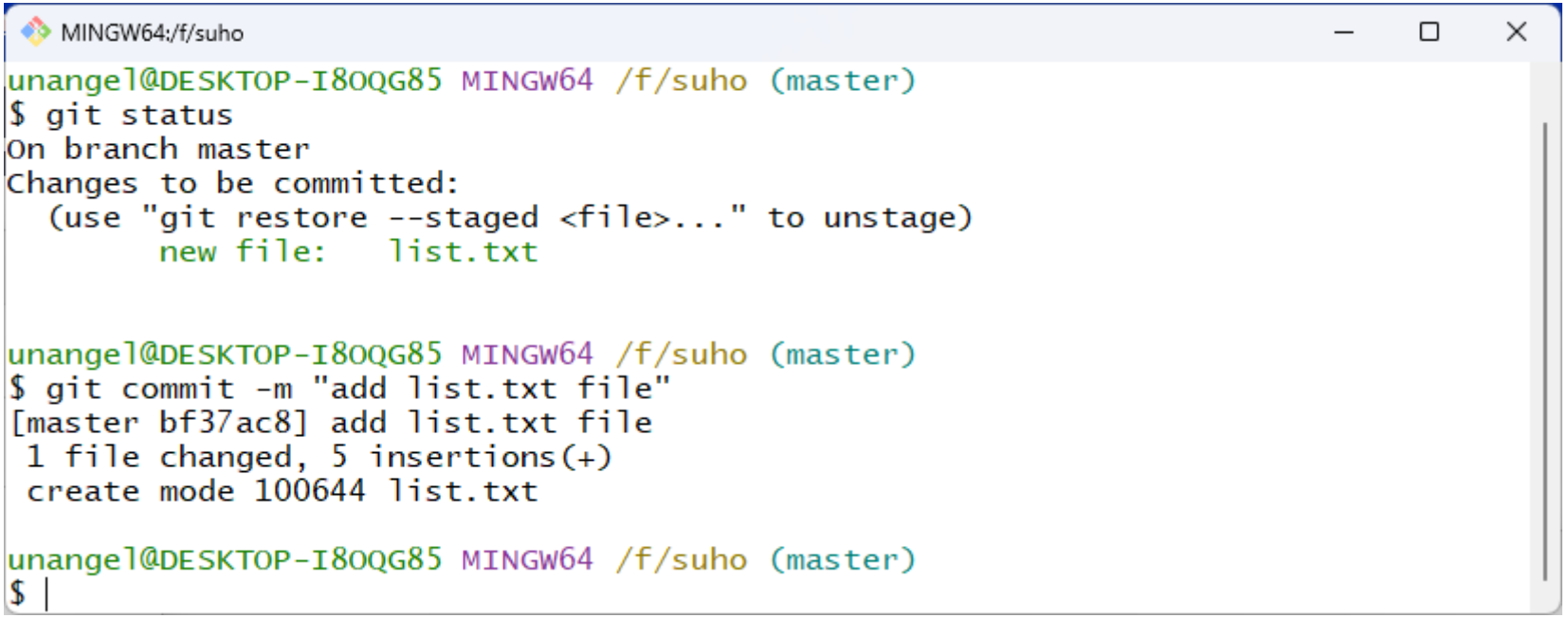
git 로컬 저장소 실습 – 11

- commit **메시지 입력** – 3
  - **메시지 입력이 필수이므로 반드시 입력 필요**
  - **Vim을 사용할 수 있어야만 정상적인 처리가 가능함**
  - **메시지 입력이 완료되면 변경된 파일 이력에 대한 정보가 출력**



git 로컬 저장소 실습 – 12

- commit **메시지 입력** – 4
  - **터미널에서 메시지를 입력할 수 있는 방법을 제공**
  - **사용법** : git commit –m "**입력할 메시지**"
  - **메시지 입력은 간단하지만, 여러 줄이나 다양한 기록을 남기기는 어려움**
  - **간단한 정보만을 남기기 때문에 이력 관리에 도움이 되지는 않음**
  - commit**은 정상적으로 수행**



git 로컬 저장소 실습 – 13

- commit **후 상태** – 1
  - **모든 파일이** commit **된 상태라면 아래와 같은 상태 정보 출력**
  - commit **완료된 상태에 대한 정보는 상태 정보에서 확인이 불가능**
  - "git log" **명령어로 이전에** commit**한 내용을 확인할 수 있음**
  - commit **해시 정보와 저작자 및 날짜와 시간 정보도 출력**
  - commit **메시지도 같이 출력되어 전체적인 정보 확인이 가능**

```
MINGW64:/f/suho
unangel@DESKTOP-I8OQG85 MINGW64 /f/suho (master)
$ git status
On branch master
nothing to commit, working tree clean

unangel@DESKTOP-I8OQG85 MINGW64 /f/suho (master)
$ git log
commit bf37ac880ca5876bda5c7eb63e48f245120523c6 (HEAD -> master)
Author: wikim <unangel@yuhan.ac.kr>
Date:   Wed Mar 19 22:11:07 2025 +0900

    add list.txt file
```

git 로컬 저장소 실습 – 14

- commit **후 상태** – 2
  - git GUI**를 이용하여** commit**된 정보를 확인할 수 있음**
  - **저장소에서** "Open git GUI here" **프로그램 실행**
  - commit**된 모든 내용을** GUI**에서 확인할 수 있음**