

오픈소스 응용 프로그래밍

- 0-2. **Git** 소개 및 버전 관리 -

김정은

jekim@inha.ac.kr

본 사이트에서 수업 자료로 이용되는 저작물은 저작권법 제25조
수업목적저작물 이용 보상금제도에 의거, 한국복제전송저작권협회와
약정을 체결하고 적법하게 이용하고 있습니다. 약정범위를 초과하는
사용은 저작권법에 저촉될 수 있으므로 수업자료의 대중 공개·공유 및
수업 목적 외의 사용을 금지합니다.

2020. 3. 인하대학교·한국복제전송저작권협회

수업 내용

- Git이란?
- Git으로 할 수 있는 작업들
- Git 활용을 위한 기초 리눅스 명령어
- Git에서의 버전 관리
- stage 공간의 필요성

Git 소개

- **Git은 오픈소스 분산 버전 관리 시스템의 일종**
 - 2005년에 리눅스의 창시자인 리누스 토발즈에 의해 제작됨
 - 소규모 프로젝트부터 대규모 프로젝트까지도 효율적으로 관리 가능
 - Git은 처음에는 리눅스 소스코드를 효율적으로 관리하고 협업하기 위해 제작되었으나 현재는 수많은 소프트웨어들의 소스코드 및 협업 프로젝트가 Git에 의해 관리되고 있음



Git logo
(Source: Wiki)



Linus Torvalds
(Source: Wiki)

Git과 Github

■ Git

- 소프트웨어
- 로컬 환경에 설치하여 사용
- 저장소를 바탕으로 소스 코드를 관리하기 위한 소프트웨어
- CLI 환경에서 동작



VS

■ GitHub

- 웹 서비스
- 웹 상에서 활용
- Git 저장소에 대한 호스팅 서비스
- GUI 환경에서 동작



Git으로 할 수 있는 작업들

- **Git**을 이용하여 아래와 같은 작업들을 수행할 수 있음
 - 버전 관리
 - 백업
 - 협업
- 협업 시에 특히 버전 관리 시스템이 유용하며, 개인적으로 개발할 때에도 백업 및 예전 버전으로의 복구가 필요할 수 있기 때문에 **Git** 등 버전 관리 시스템 활용을 추천함

Git으로 할 수 있는 작업들

- 버전 관리

- 소프트웨어 개발 진행 시에는 중간중간 버전 저장이 꼭 필요함

- 새로 추가한 기능이 최종적으로는 쓰이지 않을 수도 있고, 예기치 못한 버그가 발생하여 새롭게 시도한 파트를 다시 날리게 될 수도 있음
- 버전 관리의 가장 기본적인 방법으로 여러 개의 파일을 생성하여 저장하는 방법이 있음
 - Ex. Main.cpp, main_1.cpp, main_2.cpp, ...
- 그러나 작업의 규모가 커지게 되면 단순히 여러 개의 파일로 버전을 남겨놓으면 관리가 힘든 경우가 많음



Ver 0.1



Ver 0.2



Ver 0.3

...



Final

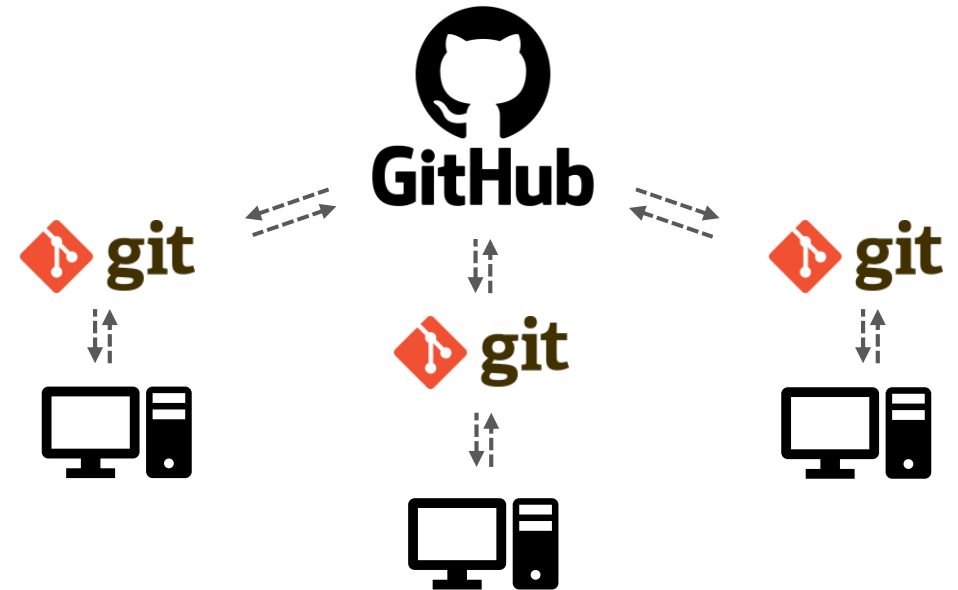
- Git은 저장소를 바탕으로 버전 관리를 손쉽게 할 수 있도록 함

- 저장소의 버전에 따라 누가, 언제, 무엇을 고쳤는지가 다 기록에 남으며 이를 언제든지 손쉽게 확인할 수 있고, 필요하다면 메인 작업 저장소를 예전 버전으로 손쉽게 되돌릴 수 있음

Git으로 할 수 있는 작업들

- 백업

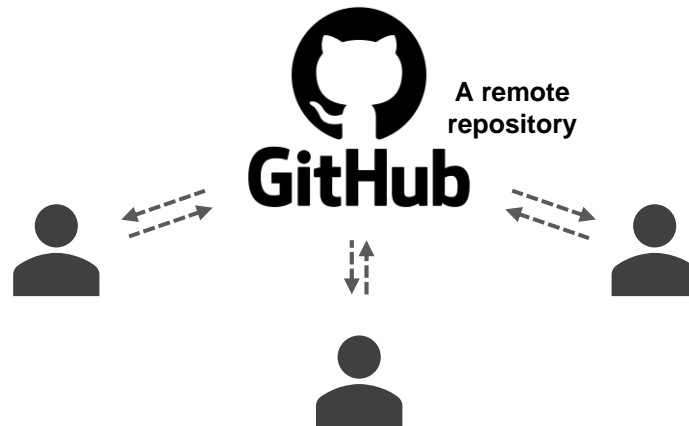
- 문서 작업 시에 예기치 못한 에러나 디스크 장애에 대비해 백업을 해야 하듯이 소프트웨어 개발에서도 백업이 필수적임
- 기본적으로는 아래와 같은 백업 방식들이 있음
 - 추가 저장장치: USB메모리, 외장하드 등 활용
 - 클라우드: Dropbox, 구글 드라이브 등 활용
- **Git의 경우에는 GitHub를 주로 활용함**
 - GitHub는 Git 저장소 호스팅 서비스로써, 클라우드 상에 Git 저장소 관련 파일들을 백업할 수 있음



Git으로 할 수 있는 작업들

- 협업

- **GitHub**를 활용하면 여러 명의 개발자가 손쉽게 협업하여 소프트웨어 개발을 진행할 수 있음
 - 네트워크로 접근하는 원격 저장소 하나에서 전체 소스코드 및 관련 파일들을 관리하므로 여러 개발자들이 하나의 저장소에 접근하여 최신 코드를 바탕으로 작업 가능
 - 통합된 원격 저장소 없이 개발자마다 로컬 환경의 개별 저장소만을 두고 작업하면 취합 시 어려움이 따를 수 있음
 - 버전마다 기록이 남으므로 소스코드 상의 문제 발생 시 파악 및 해결이 용이함

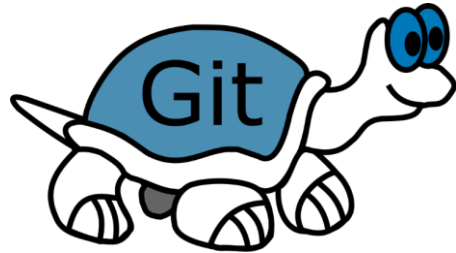


Git 프로그램

- **Git**은 여러 가지 **OS**에 대한 여러 프로그램들로 개발되어 있으며
본 강의에서는 **CLI Git** 활용에 대해 다루고자 함
 - GitHub Desktop, TortoiseGit, Sourcetree는 GUI 환경을 제공함



GitHub Desktop
(Source: Wiki)



TortoiseGit
(Source: Wiki)



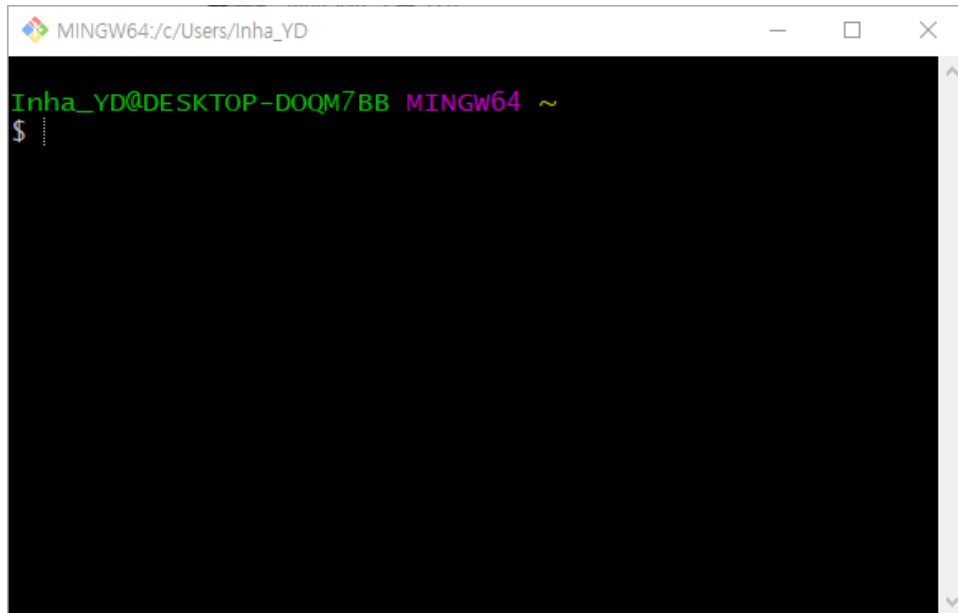
SourceTree
(Source: Wiki)



CLI
(Source: Wiki)

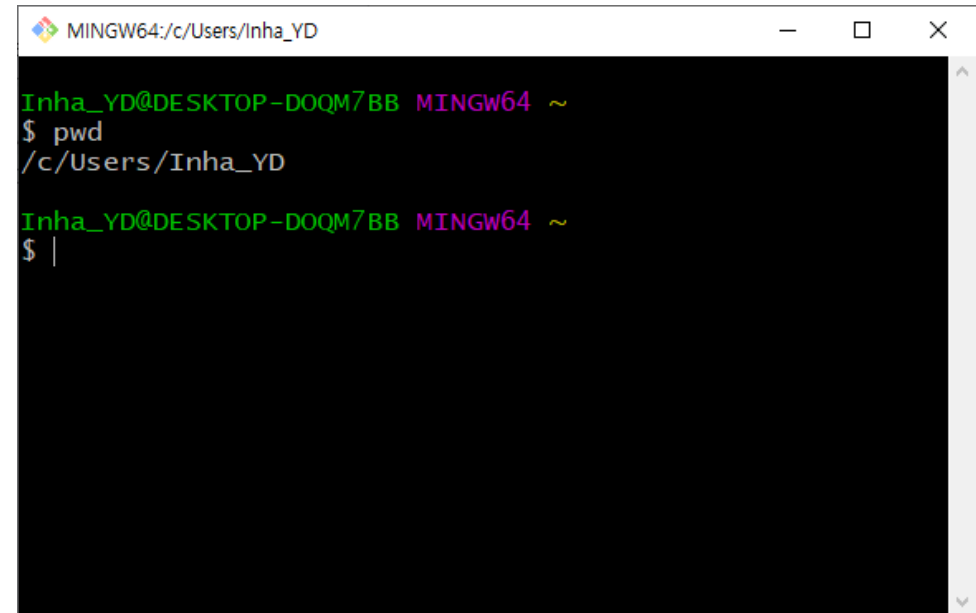
Git 활용을 위한 기초 리눅스 명령어

- ~는 홈 디렉터리를 나타냄
- 현재 작업 중인 디렉터리명 출력 명령어: `$ pwd`



```
MINGW64:/c/Users/Inha_YD

Inha_YD@DESKTOP-DOQM7BB MINGW64 ~
$ |
```



```
MINGW64:/c/Users/Inha_YD

Inha_YD@DESKTOP-DOQM7BB MINGW64 ~
$ pwd
/c/Users/Inha_YD

Inha_YD@DESKTOP-DOQM7BB MINGW64 ~
$ |
```

Git 활용을 위한 기초 리눅스 명령어

- 현재 디렉터리 내 파일 리스트 출력: `$ ls`
 - `-l` 옵션은 상세 정보 출력, `-a` 옵션은 숨겨진 파일 및 디렉터리까지 출력
 - 두 옵션을 모두 엮어서 `-la`로 옵션 부여 가능

```
MINGW64:/c/Users/Inha_YD

Inha_YD@DESKTOP-DOQM7BB MINGW64 ~
$ pwd
/c/Users/Inha_YD

Inha_YD@DESKTOP-DOQM7BB MINGW64 ~
$ ls
'3. Python_Conditional_Statements_Exercise.ipynb'
'3D Objects'/
'4. Python_Loop_Statements_Exercise.ipynb'
'5. Python_Functions_Exercise.ipynb'
'7. Python_File_Handling_Exercise.ipynb'
AppData/
'Application Data'@
Contacts/
Cookies@
'Creative Cloud Files'/
Desktop/
```

```
MINGW64:/c/Users/Inha_YD

Inha_YD@DESKTOP-DOQM7BB MINGW64 ~
$ ls -la
total 13625
drwxr-xr-x 1 Inha_YD 197121      0 Oct  6 17:53 ./
drwxr-xr-x 1 Inha_YD 197121      0 Dec 24  2020 ../
-rw-r--r-- 1 Inha_YD 197121    44 Oct  6 17:53 .gitconfig
drwxr-xr-x 1 Inha_YD 197121      0 Sep 27 16:44 .ipynb_checkpoints/
drwxr-xr-x 1 Inha_YD 197121      0 Sep  8 15:18 .ipython/
drwxr-xr-x 1 Inha_YD 197121      0 Sep  8 15:38 .jupyter/
-rw-r--r-- 1 Inha_YD 197121 2913 Sep  8 15:38 '3. Python_Conditional_Stat
ements_Exercise.ipynb'
drwxr-xr-x 1 Inha_YD 197121      0 Dec 23  2020 '3D Objects'/
-rw-r--r-- 1 Inha_YD 197121 7137 Sep  8 16:31 '4. Python_Loop_Statements_
Exercise.ipynb'
-rw-r--r-- 1 Inha_YD 197121 6560 Sep 15 16:28 '5. Python_Functions_Exerci
se.ipynb'
-rw-r--r-- 1 Inha_YD 197121 2703 Sep 15 19:35 '7. Python_File_Handling_Ex
```

Git 활용을 위한 기초 리눅스 명령어

- 현재 디렉터리 이동: `$ cd <옮겨갈 위치>`
 - `..` 은 상위 디렉터리를 나타냄
 - 우측 예시와 같이 `Users`라는 디렉터리명을 붙여 해당 디렉터리로 이동 가능

```
MINGW64:/c
Inha_YD@DESKTOP-DOQM7BB MINGW64 ~
$ cd ..

Inha_YD@DESKTOP-DOQM7BB MINGW64 /c/Users
$ cd ..

Inha_YD@DESKTOP-DOQM7BB MINGW64 /c
$ |
```

```
MINGW64:/c/Users
Inha_YD@DESKTOP-DOQM7BB MINGW64 ~
$ cd ..

Inha_YD@DESKTOP-DOQM7BB MINGW64 /c/Users
$ cd ..

Inha_YD@DESKTOP-DOQM7BB MINGW64 /c
$ cd Users

Inha_YD@DESKTOP-DOQM7BB MINGW64 /c/Users
$ |
```

Git 활용을 위한 기초 리눅스 명령어

- 디렉터리 생성: `$ mkdir <디렉터리명>`
- 파일/디렉터리 삭제: `$ rm <파일/디렉터리명>`
 - 디렉터리의 경우 `-r` 옵션으로 삭제 가능

```
MINGW64:/c/Users/Inha_YD/Documents
Inha_YD@DESKTOP-DOQM7BB MINGW64 ~
$ cd Documents

Inha_YD@DESKTOP-DOQM7BB MINGW64 ~/Documents
$ mkdir test

Inha_YD@DESKTOP-DOQM7BB MINGW64 ~/Documents
$ ls
Adobe/           'Welcome to Word.docx'
Koino/           Zoom/
MVOpenVC/        desktop.ini
MediaLog/        test/
'My Music'@      '네이버 MYBOX 보관 파일 '/
'My Pictures'@   '사용자 지정 Office 서식 파일 '/
'My Videos'@    '카카오톡 받은 파일 '/
'Welcome to Hwp.hwp'
```

```
MINGW64:/c/Users/Inha_YD/Documents
'My Pictures'@   '사용자 지정 Office 서식 파일 '/
'My Videos'@    '카카오톡 받은 파일 '/
'Welcome to Hwp.hwp'

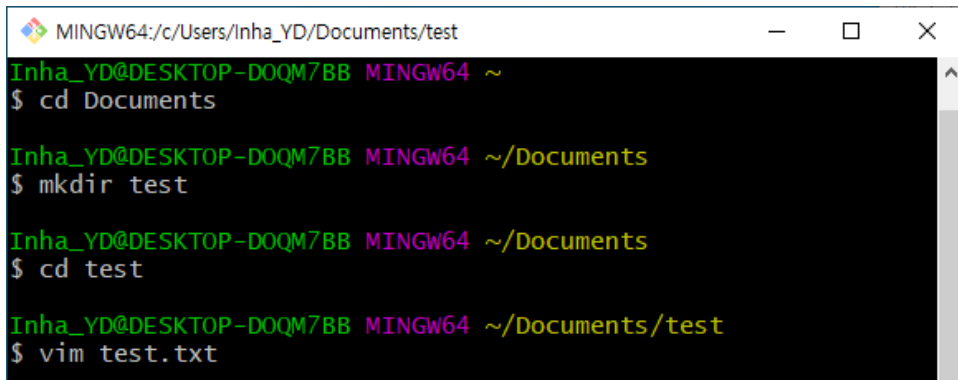
Inha_YD@DESKTOP-DOQM7BB MINGW64 ~/Documents
$ rm -r test

Inha_YD@DESKTOP-DOQM7BB MINGW64 ~/Documents
$ ls
Adobe/           'Welcome to Hwp.hwp'
Koino/           'Welcome to Word.docx'
MVOpenVC/        Zoom/
MediaLog/        desktop.ini
'My Music'@      '네이버 MYBOX 보관 파일 '/
'My Pictures'@   '사용자 지정 Office 서식 파일 '/
'My Videos'@    '카카오톡 받은 파일 '/

Inha_YD@DESKTOP-DOQM7BB MINGW64 ~/Documents
$ |
```

Git 활용을 위한 기초 리눅스 명령어

- **VIM** 에디터로 새로운 파일 생성: `$ vim <파일명>`
 - **VIM**은 리눅스에서 널리 쓰이는 텍스트 에디터의 일종
 - 익숙해지면 리눅스 환경에서 작업할 때 큰 도움이 되나,
Window/Mac 사용 시에는 **GUI** 텍스트 에디터로 파일을 생성하고 수정해도 무방함

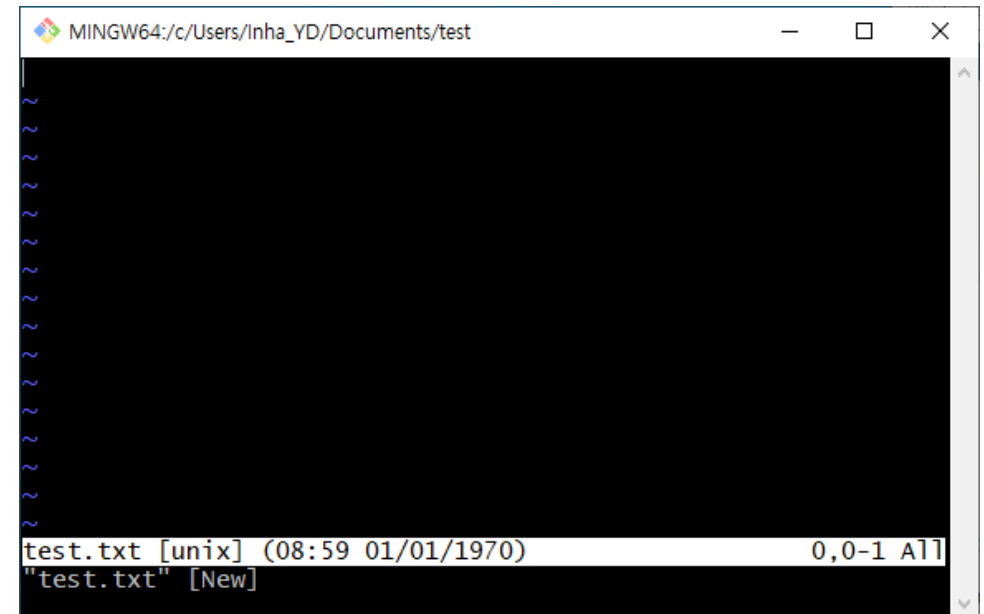


```
MINGW64:/c:/Users/Inha_YD/Documents/test
Inha_YD@DESKTOP-DOQM7BB MINGW64 ~
$ cd Documents

Inha_YD@DESKTOP-DOQM7BB MINGW64 ~/Documents
$ mkdir test

Inha_YD@DESKTOP-DOQM7BB MINGW64 ~/Documents
$ cd test

Inha_YD@DESKTOP-DOQM7BB MINGW64 ~/Documents/test
$ vim test.txt
```

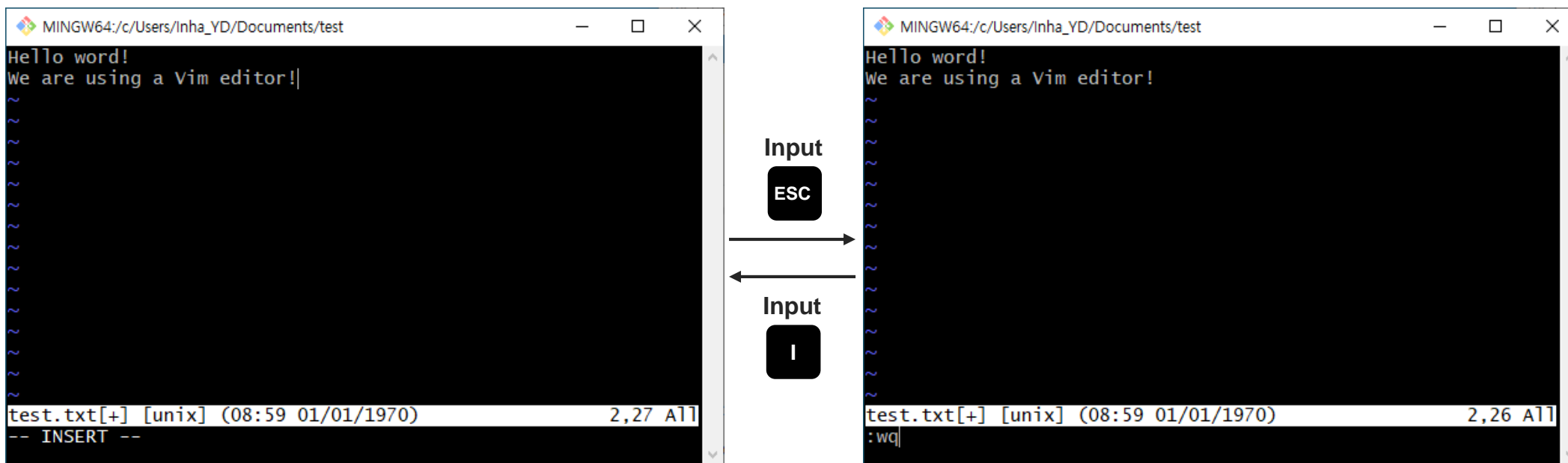


```
test.txt [unix] (08:59 01/01/1970) 0,0-1 All
"test.txt" [New]
```

Git 활용을 위한 기초 리눅스 명령어

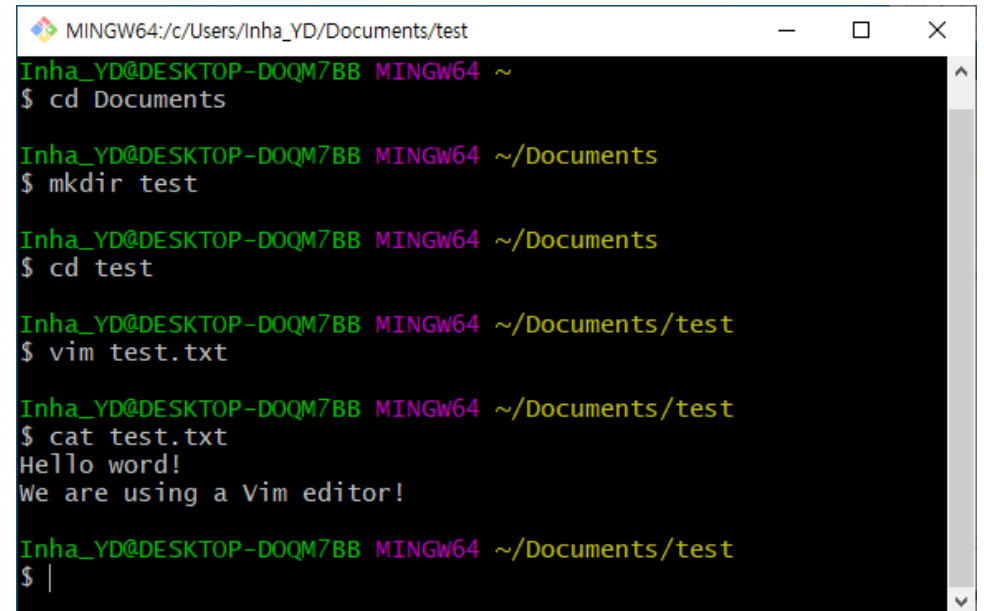
• VIM 에디터 기본 사용법

- **i**를 누르면 입력 모드로 바뀌고 텍스트 수정 가능 (입력 모드는 아래에 - **INSERT** - 라고 표시됨)
- 입력 모드에서 **Esc**를 누르면 **Ex** 모드로 전환되고 , 텍스트 수정은 안되지만 커서 이동, 검색, 저장과 종료 등 가능



Git 활용을 위한 기초 리눅스 명령어

- VIM 에디터 Ex 모드 기본 사용법
 - :w 입력 시 저장
 - :q 입력 시 에디터 종료 (:q!와 같이 !를 뒤에 붙이면 강제로 종료함)
 - :wq 와 같이 저장 및 종료 명령을 합칠 수 있음
- 파일내용 출력: \$cat <파일명>



```
MINGW64: c:/Users/Inha_YD/Documents/test
Inha_YD@DESKTOP-DOQM7BB MINGW64 ~
$ cd Documents

Inha_YD@DESKTOP-DOQM7BB MINGW64 ~/Documents
$ mkdir test

Inha_YD@DESKTOP-DOQM7BB MINGW64 ~/Documents
$ cd test

Inha_YD@DESKTOP-DOQM7BB MINGW64 ~/Documents/test
$ vim test.txt

Inha_YD@DESKTOP-DOQM7BB MINGW64 ~/Documents/test
$ cat test.txt
Hello word!
We are using a Vim editor!

Inha_YD@DESKTOP-DOQM7BB MINGW64 ~/Documents/test
$ |
```

Git 설정하기

- 본인 PC에 사용자 이름 및 이메일 등록하기 (**Git 처음 사용 시**)
 - \$ git config --global user.name "사용자 이름" // 이름은 가급적 영어 사용 권장
 - \$ git config --global user.email [사용자 이메일 주소]

```
MINGW64:/c/Users/yhkim/Do x + v - □ ×

yhkim@TCLAB185 MINGW64 ~/Documents/Git
$ git config --global user.name "Kim"

yhkim@TCLAB185 MINGW64 ~/Documents/Git
$ git config --global user.email yhkim85@inha.ac.kr

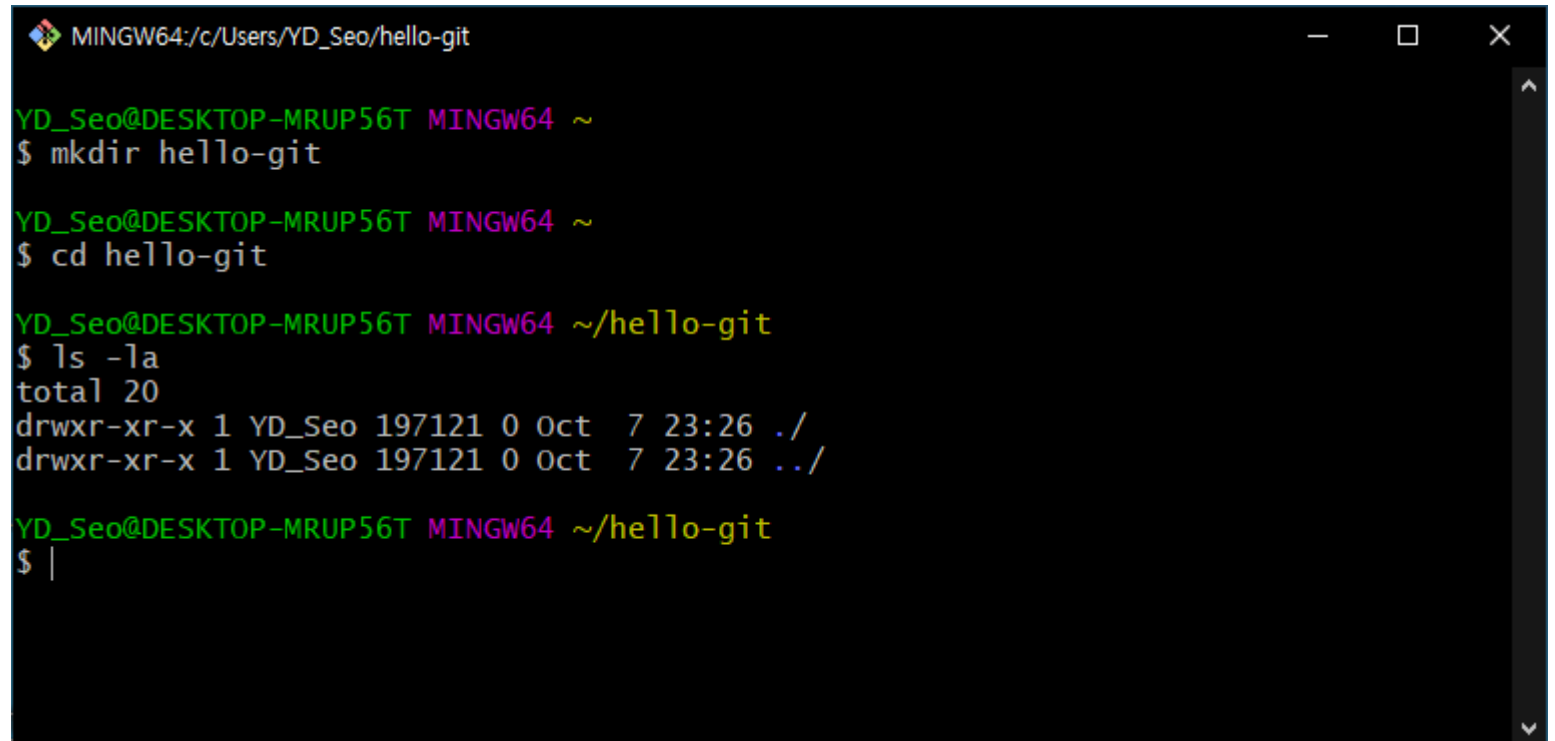
yhkim@TCLAB185 MINGW64 ~/Documents/Git
$ git config user.name
Kim

yhkim@TCLAB185 MINGW64 ~/Documents/Git
$ git config user.email
yhkim85@inha.ac.kr

yhkim@TCLAB185 MINGW64 ~/Documents/Git
$ |
```

Git 저장소 생성

- **Git** 저장소 생성 예시를 들기 위해 홈 디렉터리 아래에 **hello-git** 디렉터리 생성
 - \$ mkdir hello-git
 - \$ cd hello-git
 - \$ ls -la



```
MINGW64:/c/Users/YD_Seo/hello-git

YD_Seo@DESKTOP-MRUP56T MINGW64 ~
$ mkdir hello-git

YD_Seo@DESKTOP-MRUP56T MINGW64 ~
$ cd hello-git

YD_Seo@DESKTOP-MRUP56T MINGW64 ~/hello-git
$ ls -la
total 20
drwxr-xr-x 1 YD_Seo 197121 0 Oct  7 23:26 ./
drwxr-xr-x 1 YD_Seo 197121 0 Oct  7 23:26 ../

YD_Seo@DESKTOP-MRUP56T MINGW64 ~/hello-git
$ |
```

Git 저장소 생성

- **git init** 명령을 통해서 현재 디렉토리를 기준으로 **git** 저장소 생성 가능
 - `$ git init`
 - `.git`라는 이름의 숨겨진 디렉터리가 생성된 것을 확인 가능

```
MINGW64:/c/Users/YD_Seo/hello-git
YD_Seo@DESKTOP-MRUP56T MINGW64 ~/hello-git
$ git init
Initialized empty Git repository in C:/Users/YD_Seo/hello-git/.git/

YD_Seo@DESKTOP-MRUP56T MINGW64 ~/hello-git (master)
$ ls -la
total 24
drwxr-xr-x 1 YD_Seo 197121 0 Oct  7 23:27 ./
drwxr-xr-x 1 YD_Seo 197121 0 Oct  7 23:26 ../
drwxr-xr-x 1 YD_Seo 197121 0 Oct  7 23:27 .git/

YD_Seo@DESKTOP-MRUP56T MINGW64 ~/hello-git (master)
$ |
```

Git에서의 버전 관리

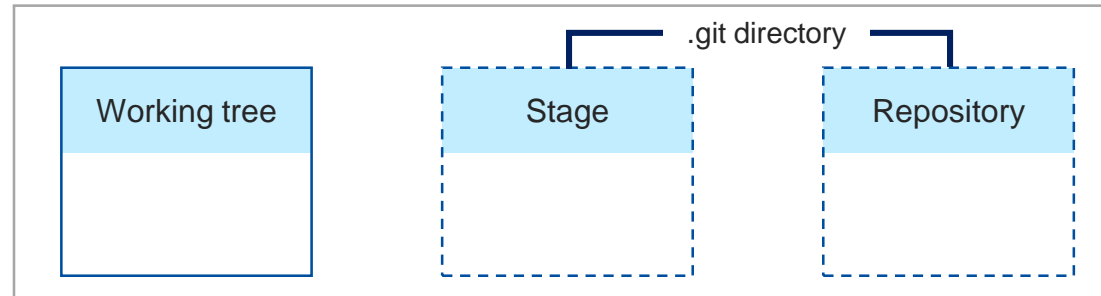
- **git을 이용하면 아래와 같이 소스코드 버전 관리가 가능함**
 - 같은 파일명을 가진 소스코드를 여러 시점에서의 버전으로 저장 가능
 - 그렇지 않다면 각각 다른 파일명으로 저장해야 할 것



- 저장된 각 버전의 소스코드 내용을 확인할 수 있으며, 서로 다른 버전에서의 차이점 또한 변경된 부분에만 집중하여 확인 가능
- 파일들을 특정 시점의 버전으로 되돌릴 수 있음

Git에서의 버전 관리

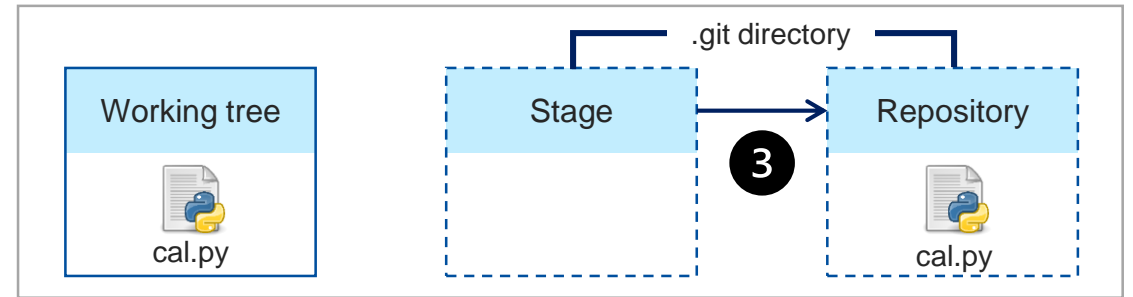
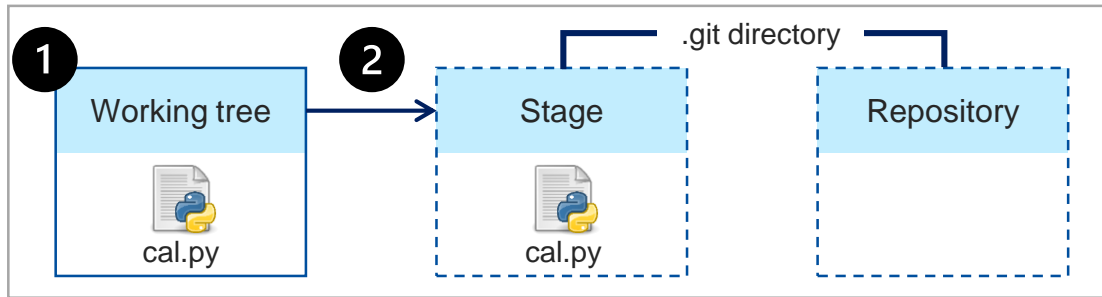
- git에서는 아래와 같은 세 가지의 공간을 활용하여 저장소를 관리함



- Working tree
 - 소스코드를 직접 수정 및 저장하는 디렉터리 (예시에서는 hello-git 디렉터리에 해당)
- Stage (또는 staging area)
 - 버전관리를 수행할 대상이 되는 파일들이 모인 공간
- Repository
 - 저장소라고 하며 각 버전이 저장되어 있는 공간

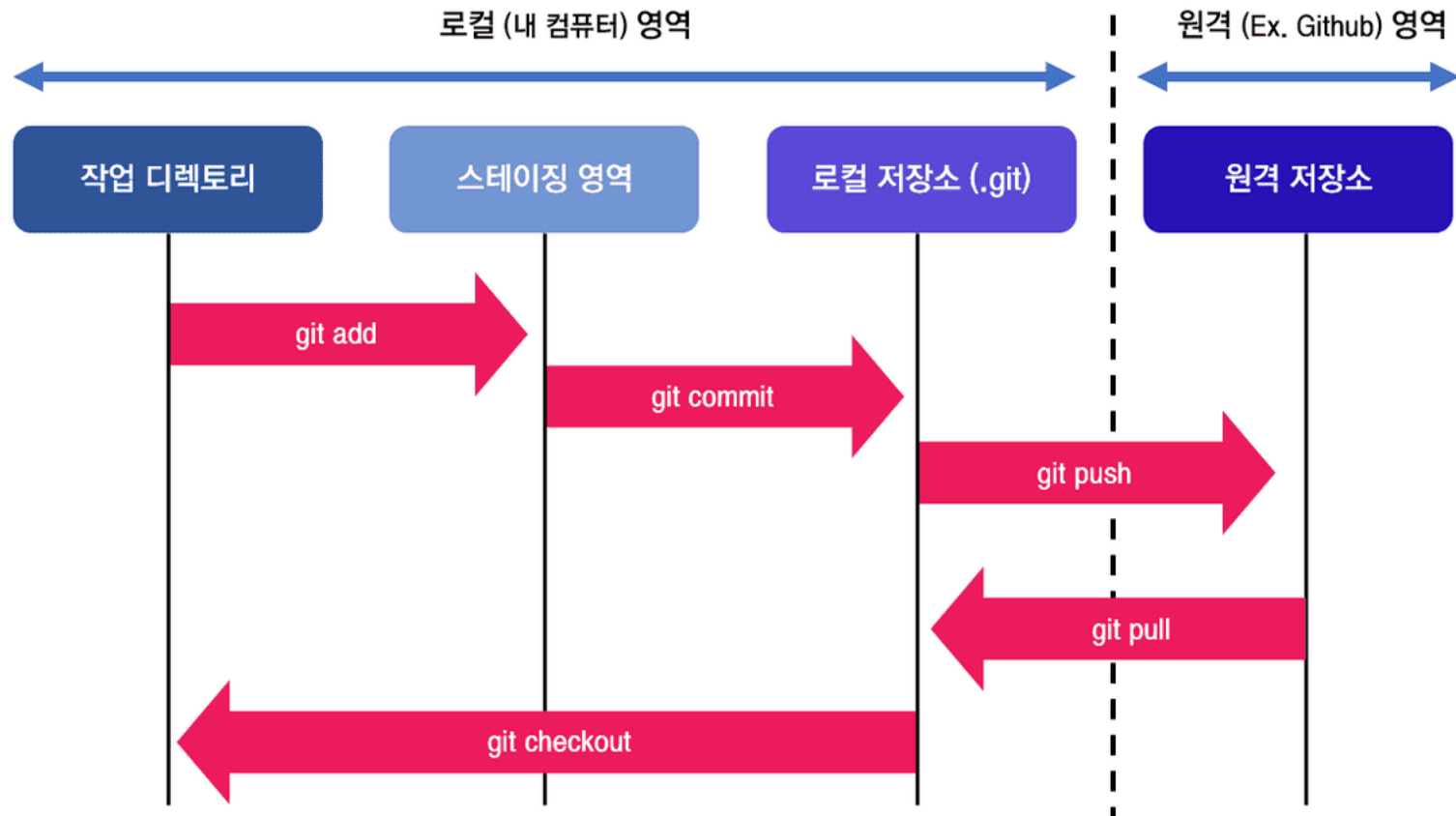
Git에서의 버전 관리

- git에서의 버전 관리 과정 예시



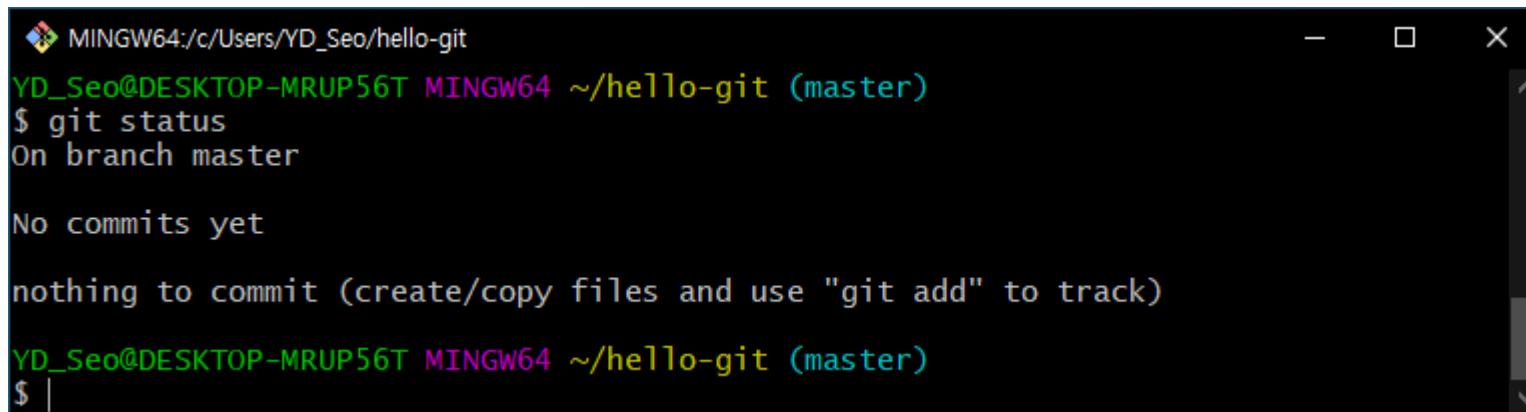
- ① Working tree에서 cal.py라는 소스코드를 새로 생성하고 저장하면 아직은 stage는 비어 있음
- ② **add** 명령을 통해 stage에 추가할 수 있고 이후 저장소에 commit할 수 있음
- ③ **commit**을 할 경우 stage에 있는 파일들이 저장소로 이동하며 새로운 버전이 생성됨

Git에서의 버전 관리



Git에서의 버전 관리

- `$git status` 명령을 통해 저장소 상태를 확인 가능
- 아직 새로 생성한 파일이 없는 상태에서는 아래와 같이 출력됨

A screenshot of a Windows command prompt window titled 'MINGW64: c:/Users/YD_Seo/hello-git'. The prompt shows the user 'YD_Seo@DESKTOP-MRUP56T' in the 'MINGW64' environment at the directory '~/hello-git' on the 'master' branch. The command '\$ git status' has been entered, and the output is: 'On branch master', 'No commits yet', and 'nothing to commit (create/copy files and use "git add" to track)'. The prompt is ready for the next command.

```
MINGW64: c:/Users/YD_Seo/hello-git
YD_Seo@DESKTOP-MRUP56T MINGW64 ~/hello-git (master)
$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
YD_Seo@DESKTOP-MRUP56T MINGW64 ~/hello-git (master)
$
```

- master branch(main branch)에 대한 저장소라는 의미. branch 개념은 다음 시간에 다룰 예정
- 아직 commit된 버전이 없음
- commit을 할 수 있는 대상 파일들(stage의 파일들)이 없는 상태

Git에서의 버전 관리

- **VIM 에디터를 이용하여 cal.py라는 소스코드를 새로 생성한 상태**
 - cal.py는 버전 관리 설명을 위한 예시 코드로써 간단한 사칙연산을 위한 소스코드
 - 새로 생성한 소스코드가 있다는 점이 중요함. GUI상에서 코드 생성 후 저장해도 무방함

```
MINGW64:/c/Users/YD_Seo/hello-git
class FourCal:
    def __init__(self, first, second):
        self.first = first
        self.second = second
    def setdata(self, first, second):
        self.first = first
        self.second = second
    def add(self):
        result = self.first + self.second
        return result
~
~
~
~
~
cal.py[+] [unix] (08:59 01/01/1970) 10,21 All
:wq|
```

```
MINGW64:/c/Users/YD_Seo/hello-git
total 28
drwxr-xr-x 1 YD_Seo 197121 0 Oct 8 00:42 ./
drwxr-xr-x 1 YD_Seo 197121 0 Oct 8 00:39 ../
drwxr-xr-x 1 YD_Seo 197121 0 Oct 8 00:33 .git/

YD_Seo@DESKTOP-MRUP56T MINGW64 ~/hello-git (master)
$ vim cal.py

YD_Seo@DESKTOP-MRUP56T MINGW64 ~/hello-git (master)
$ ls -la
total 29
drwxr-xr-x 1 YD_Seo 197121 0 Oct 8 00:42 ./
drwxr-xr-x 1 YD_Seo 197121 0 Oct 8 00:42 ../
drwxr-xr-x 1 YD_Seo 197121 0 Oct 8 00:33 .git/
-rw-r--r-- 1 YD_Seo 197121 287 Oct 8 00:42 cal.py

YD_Seo@DESKTOP-MRUP56T MINGW64 ~/hello-git (master)
$
```

Git에서의 버전 관리

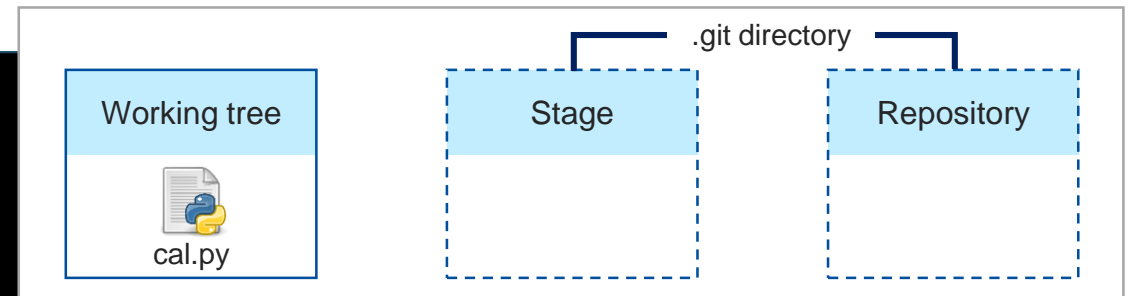
- **Working tree**에 새로운 파일 추가 후 **git status** 확인
 - 앞서 확인했던 상태에 추가로 Untracked files라는 항목이 더 생겼다는 것을 알 수 있음
 - 현재 cal.py는 working tree에는 있지만 stage로 추가되지 않았으며, 이러한 파일들은 **untracked** 상태라고 함. 즉, 아직 git에 의해 버전이 관리되고 있지 않은 파일로 볼 수 있음.

```
MINGW64:/c/Users/YD_Seo/hello-git
YD_Seo@DESKTOP-MRUP56T MINGW64 ~/hello-git (master)
$ git status
On branch master

No commits yet

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

nothing added to commit but untracked files present (use "git add" to track)
YD_Seo@DESKTOP-MRUP56T MINGW64 ~/hello-git (master)
$
```



Git에서의 버전 관리

- `$git add <파일명>` 명령을 통해서 특정 파일을 **stage**에 추가할 수 있음

```
MINGW64:/c/Users/YD_Seo/hello-git
YD_Seo@DESKTOP-MRUP56T MINGW64 ~/hello-git (master)
$ git add cal.py
warning: LF will be replaced by CRLF in cal.py.
The file will have its original line endings in your working directory
YD_Seo@DESKTOP-MRUP56T MINGW64 ~/hello-git (master)
$
```

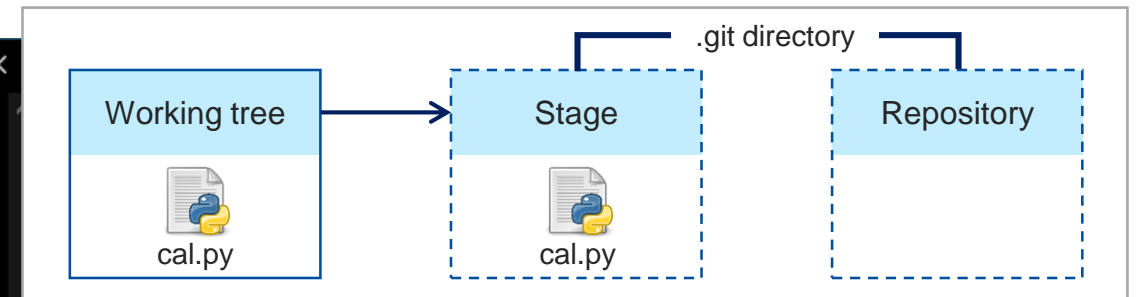
- 다시 `git status`로 확인해보면 `cal.py`는 **commit**할 수 있는 파일로 변경되어 있음

```
MINGW64:/c/Users/YD_Seo/hello-git
YD_Seo@DESKTOP-MRUP56T MINGW64 ~/hello-git (master)
$ git status
On branch master

No commits yet

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

YD_Seo@DESKTOP-MRUP56T MINGW64 ~/hello-git (master)
$
```



Git에서의 버전 관리

- **\$git commit** 명령을 통해서 현재 **stage**의 파일들의 새로운 버전 생성 가능
 - -m 옵션은 버전 기록에 남을 메시지를 추가하는 옵션
 - **\$git commit -m "create cal.py"**
 - cal.py를 생성하였다는 메시지를 남기고, 현재 **stage**에 있는 파일들을 이용하여 버전을 새롭게 생성하는 명령

```
MINGW64:/c/Users/YD_Seo/hello-git
YD_Seo@DESKTOP-MRUP56T MINGW64 ~/hello-git (master)
$ git commit -m "create cal.py"
[master (root-commit) 11bf220] create cal.py
1 file changed, 10 insertions(+)
create mode 100644 cal.py

YD_Seo@DESKTOP-MRUP56T MINGW64 ~/hello-git (master)
$ |
```

Git에서의 버전 관리

- **commit 후 git status 확인**

- commit된 버전이 있기 때문에 앞서 나왔던 “No commits yet” 메시지는 사라진 것을 확인 가능
- commit과 함께 stage는 비워졌으며 working tree 상태도 현재 버전의 파일 구성과 일치
 - 따라서 commit할만한 내용이 없으며 working tree 또한 clean 상태라고 나옴

```
MINGW64:/c/Users/YD_Seo/hello-git
YD_Seo@DESKTOP-MRUP56T MINGW64 ~/hello-git (master)
$ git status
On branch master
nothing to commit, working tree clean

YD_Seo@DESKTOP-MRUP56T MINGW64 ~/hello-git (master)
$ |
```

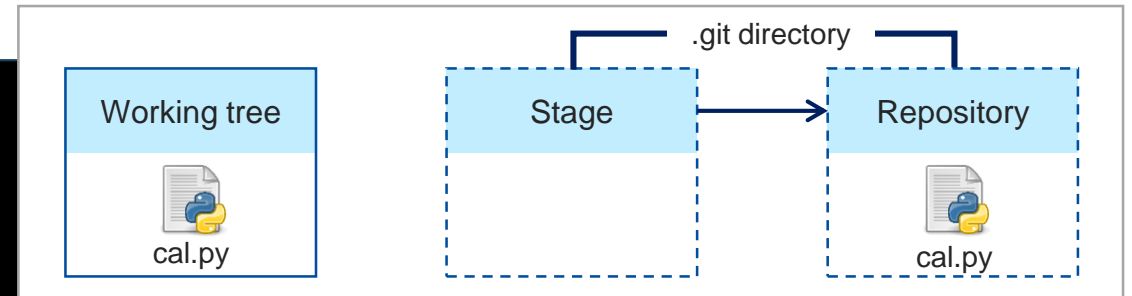
Git에서의 버전 관리

- `$git log` 명령을 통해 버전 이력 확인 가능

```
MINGW64:/c/Users/YD_Seo/hello-git
YD_Seo@DESKTOP-MRUP56T MINGW64 ~/hello-git (master)
$ git log
commit 11bf220b10c72867c3b4877ce38895c4b71ebf80 (HEAD -> master)
Author: Inha <Inha@gmail.com>
Date:   Fri Oct 8 01:08:31 2021 +0900

    create cal.py

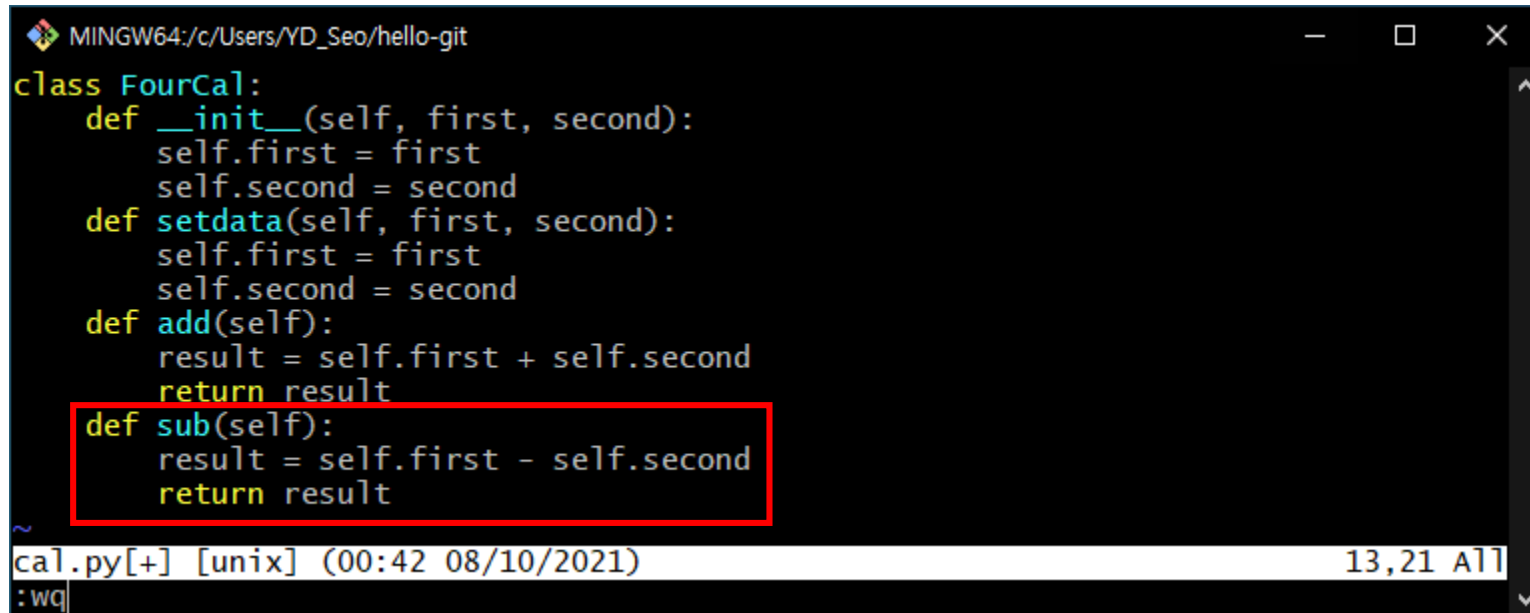
YD_Seo@DESKTOP-MRUP56T MINGW64 ~/hello-git (master)
$ |
```



- 가장 위쪽에 나오는 내용은 commit hash로써 각 commit에 대한 고유 식별자임
 - **HEAD -> master**라는 표시는 최신 버전임을 의미함
- commit을 누가, 언제 하였는지와 **commit** 메시지가 출력됨
- 우측 그림과 같이 현재 상태에서는 stage는 비워진 상태임에 유의

Git에서의 버전 관리

- cal.py 코드에 sub라는 함수를 추가하고 다시 commit하는 상황 가정



```
MINGW64:/c/Users/YD_Seo/hello-git
class FourCal:
    def __init__(self, first, second):
        self.first = first
        self.second = second
    def setdata(self, first, second):
        self.first = first
        self.second = second
    def add(self):
        result = self.first + self.second
        return result
    def sub(self):
        result = self.first - self.second
        return result

~
cal.py[+] [unix] (00:42 08/10/2021) 13,21 All
:wq|
```


Git에서의 버전 관리

- “add sub method”라는 메시지와 함께 **commit** 수행하는 상황 가정
 - \$git add cal.py
 - \$git commit -m “add sub method”
- **commit 이후 git log 실행**
 - 총 2번의 commit이 있었으며, “add sub method”라는 메시지가 있는 버전이 최신임을 알 수 있음

```
MINGW64:/c/Users/YD_Seo/hello-git
YD_Seo@DESKTOP-MRUP56T MINGW64 ~/hello-git (master)
$ git log
commit 0029ec5f3a7321cf04829c1f3faa62308f9c2c68 (HEAD -> master)
Author: Inha <Inha@gmail.com>
Date:   Fri Oct 8 01:23:38 2021 +0900

    add sub method

commit 11bf220b10c72867c3b4877ce38895c4b71ebf80
Author: Inha <Inha@gmail.com>
Date:   Fri Oct 8 01:08:31 2021 +0900

    create cal.py

YD_Seo@DESKTOP-MRUP56T MINGW64 ~/hello-git (master)
$ |
```

Git에서의 버전 관리

- “sub”라는 함수명이 마음에 들지 않아 “subtraction”으로 함수명을 바꾼 상황 가정

```
MINGW64:/c/Users/YD_Seo/hello-git
class FourCal:
    def __init__(self, first, second):
        self.first = first
        self.second = second
    def setdata(self, first, second):
        self.first = first
        self.second = second
    def add(self):
        result = self.first + self.second
        return result
    def sub(self):
        result = self.first - self.second
        return result

cal.py[+] [unix] (00:42 08/10/2021)
:wq
```



```
MINGW64:/c/Users/YD_Seo/hello-git
class FourCal:
    def __init__(self, first, second):
        self.first = first
        self.second = second
    def setdata(self, first, second):
        self.first = first
        self.second = second
    def add(self):
        result = self.first + self.second
        return result
    def subtraction(self):
        result = self.first - self.second
        return result

cal.py[+] [unix] (01:22 08/10/2021)
:wq
```

Git에서의 버전 관리

- 파일 수정 후 아직 **add**하지 않은 상태에서 **git status**를 출력하면 아래와 같음
 - git에서는 cal.py가 수정된 것을 파악하고 있음
 - 그리고 commit하기 위해서는 add를 먼저 하라고 하며, 만약 변경사항을 취소를 원한다면 restore를 하라고 안내해주고 있음

```
MINGW64:/c/Users/YD_Seo/hello-git
YD_Seo@DESKTOP-MRUP56T MINGW64 ~/hello-git (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   cal.py

no changes added to commit (use "git add" and/or "git commit -a")
YD_Seo@DESKTOP-MRUP56T MINGW64 ~/hello-git (master)
$ |
```

Git에서의 버전 관리

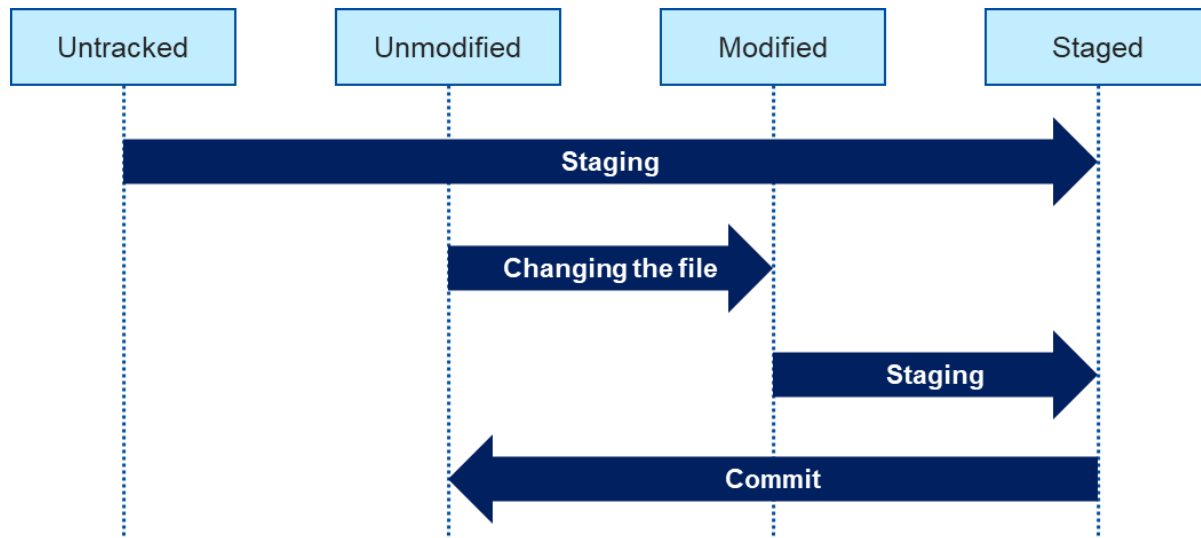
- **git status** 출력 후 구체적인 변경사항을 확인하고 싶은 경우
 - \$git diff를 통해 최신 버전과 현재 working tree 내의 차이점들을 요약하여 확인 가능
 - - 표시 뒤의 최신 버전에 있던 라인이 사라졌고, + 표시 뒤의 내용으로 대체되었음을 의미함

```
MINGW64:/c/Users/YD_Seo/hello-git
YD_Seo@DESKTOP-MRUP56T MINGW64 ~/hello-git (master)
$ git diff
warning: LF will be replaced by CRLF in cal.py.
The file will have its original line endings in your working directory
diff --git a/cal.py b/cal.py
index df7f9f3..42488c5 100644
--- a/cal.py
+++ b/cal.py
@@ -8,6 +8,6 @@ class FourCal:
     def add(self):
         result = self.first + self.second
         return result
-    def sub(self):
+    def subtraction(self):
         result = self.first - self.second
         return result

YD_Seo@DESKTOP-MRUP56T MINGW64 ~/hello-git (master)
$ |
```

Git에서의 버전 관리

- 요약하면 **git**에서의 **commit** 프로세스는 아래와 같음
 - 새로 추가된 파일(untracked) 및 기존에 있었으나 수정된 파일(modified)들을 stage에 추가
 - 이후 commit 수행 가능하며 기본적으로 **stage**에 있는 파일들만 반영됨
 - -a 옵션을 통해 변경된 파일들을 자동으로 stage에 반영 후 commit 가능하나, 해당 옵션으로 한번에 commit하기 보다는 stage를 꼼꼼히 확인 후 commit하길 추천
 - unmodified 파일들은 최신 버전 그대로의 파일들을 말함



Git에서의 버전 관리

- **stage**에 추가된 파일을 다시 **stage**에서 제거하고 싶은 경우
 - `$git reset HEAD <파일명>`
 - 특정 파일을 stage로부터 제거하는 명령
 - `$ git restore --staged <파일명>`도 동일한 연산
- 변경했던 파일을 다시 최신 버전의 내용으로 되돌리고 싶은 경우
 - `$git checkout -- <파일명>`
 - **working tree**에 수정한 파일을 저장소에 있던 버전으로 다시 덮어쓰는 명령
 - 수정했던 사항이 사라짐에 유의

Git에서의 버전 관리

- **commit**된 파일의 내용을 이전 버전으로 되돌리고 싶은 경우
 - 최신 버전
 - `$git reset HEAD^`
 - 특정 버전
 - `$git reset --hard <원하는 버전 commit hash>`
- 최신 버전이 아닌 그 이전 버전으로 강제로 되돌릴 경우 되돌린 버전 이후의 **commit**들이 기본적으로는 보이지 않음에 유의
 - `git log`를 통해 살펴보면 이후 버전들이 사라져 있지만 **git reflog**로 확인 및 복구를 할 수는 있음
 - `git reset --hard` 명령은 굉장히 위험할 수 있다는 것을 꼭 유념해야 함

Git에서의 버전 관리

- 여러 개의 **commit**이 있는 상태에서 이전 버전으로 되돌린 예시

```
MINGW64:/c/Users/YD_Seo/hello-git
YD_Seo@DESKTOP-MRUP56T MINGW64 ~/hello-git (master)
$ git log
commit 7a81b3f55986dde4d64c5ab1fa6fd11b05264a48 (HEAD -> master)
Author: Inha <Inha@gmail.com>
Date: Sat Oct 9 01:34:55 2021 +0900

    print 5

commit 874afd75e3af13ae4e072698fe81cd252cfb7
Author: Inha <Inha@gmail.com>
Date: Sat Oct 9 01:32:26 2021 +0900

    print 4

commit ddae2bc724a2448bd36dce107592452bd3bfa81b
Author: Inha <Inha@gmail.com>
Date: Sat Oct 9 01:31:56 2021 +0900

    print 3

commit 3d38e8febebb19430f81afa30ebb5a87c8e8d73f
Author: Inha <Inha@gmail.com>
Date: Sat Oct 9 01:29:32 2021 +0900

    print2

commit cd0d6249a6156f01c09bf5a44519e014d264f55a
Author: Inha <Inha@gmail.com>
Date: Sat Oct 9 01:24:07 2021 +0900

    print 1
```



```
MINGW64:/c/Users/YD_Seo/hello-git
YD_Seo@DESKTOP-MRUP56T MINGW64 ~/hello-git (master)
$ git reset --hard 3d38e8febebb19430f81afa30ebb5a87c8e8d73f
HEAD is now at 3d38e8f print2

YD_Seo@DESKTOP-MRUP56T MINGW64 ~/hello-git (master)
$

MINGW64:/c/Users/YD_Seo/hello-git
YD_Seo@DESKTOP-MRUP56T MINGW64 ~/hello-git (master)
$ git log
commit 3d38e8febebb19430f81afa30ebb5a87c8e8d73f (HEAD -> master)
Author: Inha <Inha@gmail.com>
Date: Sat Oct 9 01:29:32 2021 +0900

    print2

commit cd0d6249a6156f01c09bf5a44519e014d264f55a
Author: Inha <Inha@gmail.com>
Date: Sat Oct 9 01:24:07 2021 +0900

    print 1

commit 89930eda359fceec2f3ba907e4f256027a55923a
Author: Inha <Inha@gmail.com>
Date: Sat Oct 9 00:56:51 2021 +0900
```


Git에서의 버전 관리 요약

- 새 저장소 생성: `$git init`
- 현재 버전 및 각종 파일들의 상태 확인: `$git status`
- 버전 이력 확인: `$git log`

- **Stage**에 파일 추가: `$git add <파일명>`
- **Stage**의 파일들 **commit**: `$git commit -m "<커밋 메시지>"`
- 최신 버전과 현재 **working tree** 차이점 요약: `$git diff`

Git에서의 버전 관리 요약

- **Stage**에서의 파일 제거: `$git reset HEAD <파일명>`
- **Working tree**에서의 변경사항 취소: `$git checkout -- <파일명>`
- 최신 버전의 바로 이전 버전으로 되돌리기: `$git reset HEAD^`
 - commit된 파일과 stage의 파일 내용을 되돌림 (default: --mixed)
- 특정 버전으로 **working tree**를 되돌리기: `$git reset --hard <commit hash>`

stage 공간의 필요성

- **Working tree** 및 **repository**만으로도 버전 관리가 가능한 하지만, **stage** 공간이 있음으로써 아래와 같은 장점들을 누릴 수 있음
 - 수정된 부분 중 일부분만 **commit** 가능
 - 코드를 열심히 고치다가 일부 파일은 수정이 완료되고 일부는 아직 수정 중일 때, 특정 파일만 골라서 **commit**할 수 있음
 - 대규모 프로젝트에서는 각자 맡은 부분이 있기 마련인데, 자신이 작업한 부분만 골라서 **commit**하기에 굉장히 용이함
 - **commit** 전 코드 리뷰 및 테스트 용이
 - 위의 장점과 유사하나 **commit** 전에 **stage** 공간의 코드들을 이용하여 테스트를 진행하고 문제 없을 시 **commit**하는 방식으로 실수가 포함된 **commit**을 줄일 수 있음

질문?