

Git/GitHub



학습 목표

버전관리를 설명할 수 있다.

마크다운 기반 문서 작성을 할 수 있고, Git으로 관리할 수 있다.

Git으로 로컬저장소를 생성하고 프로젝트 버전관리를 한다.

GitHub 원격 저장소를 생성하고 관리한다.

GitHub Pull Request 기반 협업을 경험하고 활용할 수 있다.

Git? GitHub?







Overview Repositories 182 Projects Packages Stars 262

jojoldu / README.md

Hi there 🎉

DongUk Lee's GitHub Stats

- ★ Total Stars Earned: 10.3k
- ⌚ Total Commits (2021): 992
- ↑ Total PRs: 21
- ⓘ Total Issues: 235
- ✉️ Contributed to: 6

A++

Pinned

- blog-code (Public)
http://jojoldu.tistory.com/에서 제공하는 예제 code
Java ⭐ 466 ⚡ 261
- markdown-tistory (Public)
작성된 마크다운의 내용과 이미지를 본인 티스토리에 업로드하는 프로젝트
JavaScript ⭐ 198 ⚡ 40
- junior-recruit-scheduler (Public)
주니어 개발자 채용 정보
JavaScript ⭐ 7k ⚡ 1.3k
- translator (Public)
IntelliJ Translate Plugin
Java ⭐ 176 ⚡ 24
- springboot-webservice (Public)
스프링부트로 웹서비스 구축하기 시리즈
Java ⭐ 394 ⚡ 132
- spring-boot-aws-mock (Public)
Queue, Redis, Etc (Aws Service Mock Library for Spring Boot)
Java ⭐ 45 ⚡ 16

1,089 contributions in the last year

2021

2020

2019

2018

2017

Less More

Learn how we count contributions

DongUk Lee
jojoldu

Follow

Backend Application Developer

3.5k followers · 56 following

@inlearn
Seoul
jojoldu@gmail.com
http://jojoldu.tistory.com/
@jojoldu

Achievements

Block or Report

Mon

Wed

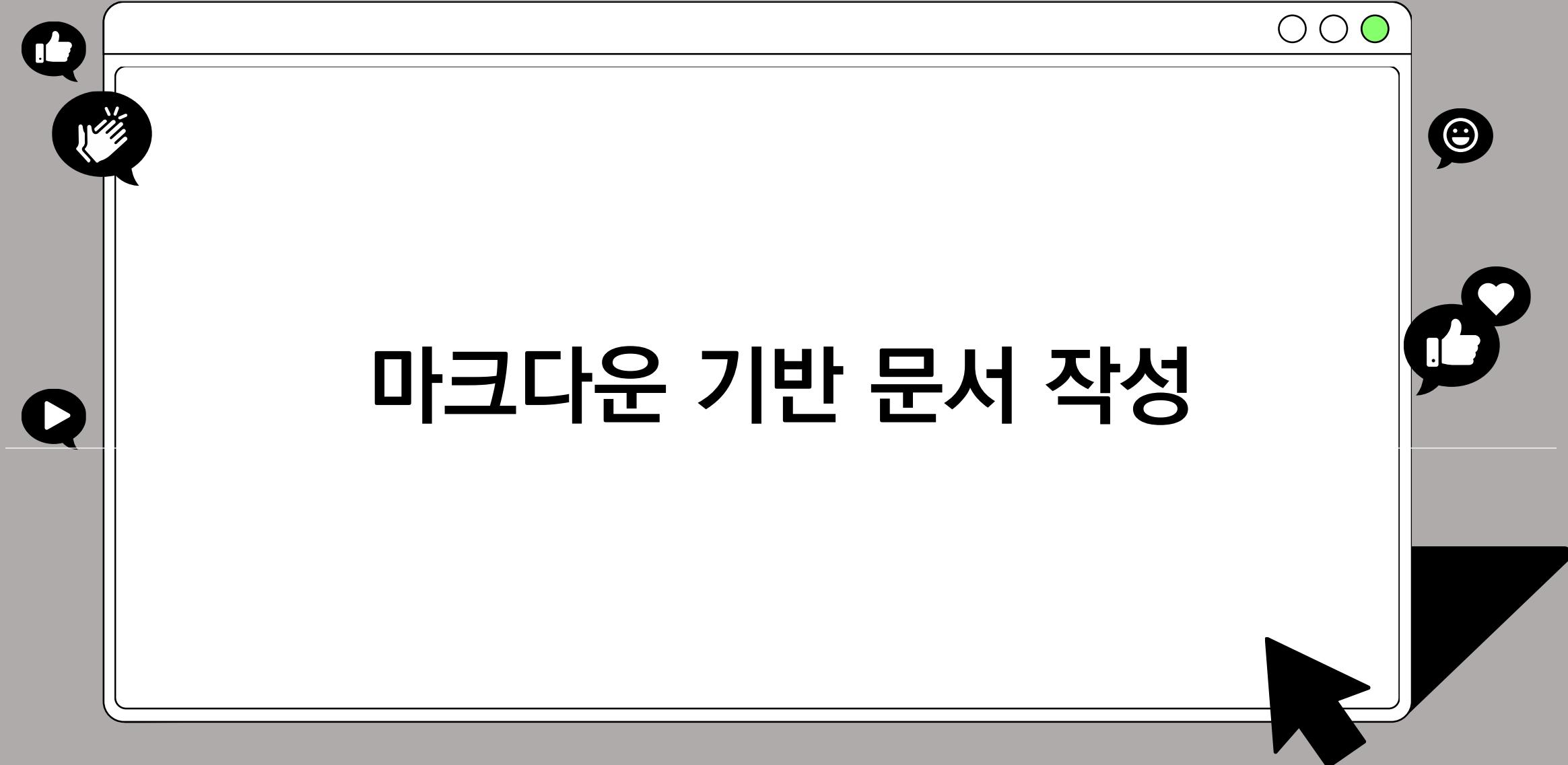
Fri

Dec Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec

목차

- Markdown을 활용한 문서 작성
- Git을 활용한 버전 관리
 - 버전 관리 기본
 - Git branch
- GitHub을 활용한 포트폴리오 관리 및 개발 프로젝트 시나리오
 - 개인 포트폴리오 관리
 - TIL (Today I Learned)
 - 개인 개발 프로젝트
 - 프로젝트(협업)
 - GitHub Flow를 활용한 개발 프로젝트 가이드라인

마크다운 기반 문서 작성



학습 목표

워드프로세서와 마크다운의 차이점을 알고 설명할 수 있다.

마크다운 기반 문법을 활용하여 문서 작성을 할 수 있다.

마크다운 개요

- 2004년 존 그루버가 만든 텍스트 기반의 가벼운 마크업 언어
- 최초 마크다운에 비해 확장된 문법(표, 주석 등)이 있지만, 본 수업에서는 Github에서 사용 가능한 문법(Github Flavored Markdown)을 기준으로 설명

Markdown is a **text-to-HTML conversion tool** for web writers.

Markdown allows you to write using an **easy-to-read, easy-to-write plain text format**, then convert it to structurally valid XHTML (or HTML).

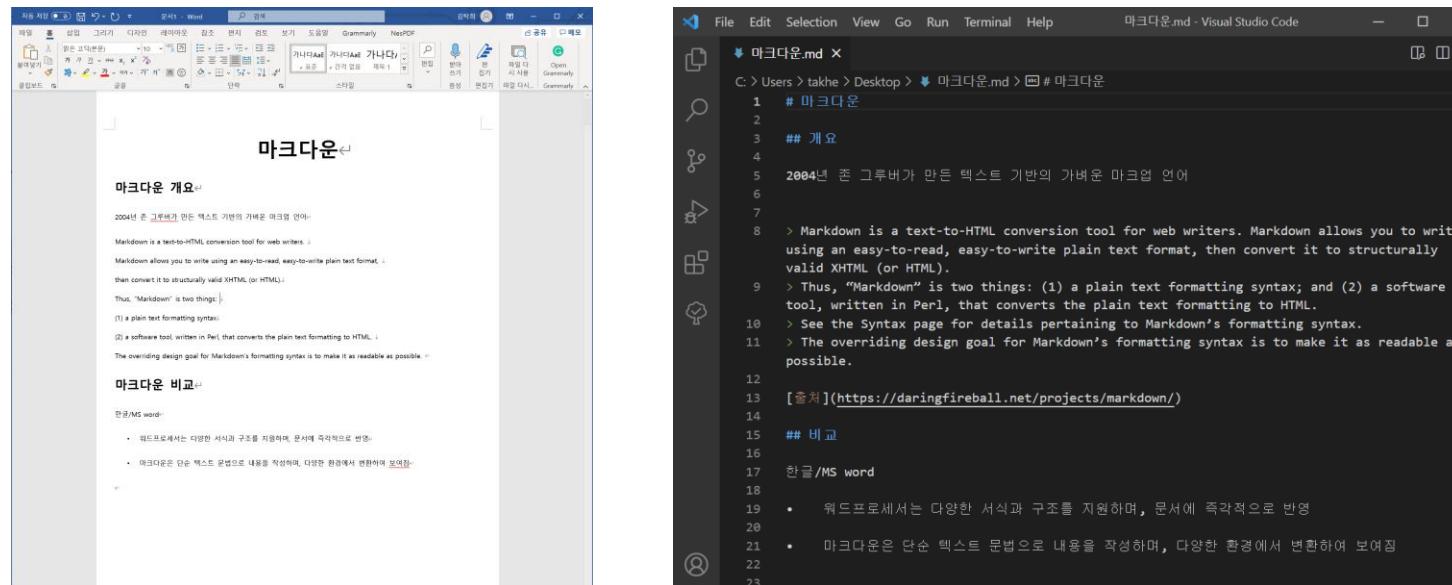
Thus, “Markdown” is two things:

- (1) **a plain text formatting syntax**
- (2) a software tool, written in Perl, **that converts the plain text formatting to HTML**.

The overriding **design goal** for Markdown’s formatting syntax is to make it as readable as possible.

마크다운 특징

- 워드프로세서(한글/MS word)는 다양한 서식과 구조를 지원하지만, 다른 프로그램으로 호환에 제약이 되며, 워드프로세스 상의 기능을 활용해야 함
- 마크다운은 최소한의 문법으로 구성되어 있으며 순수 텍스트로 작성 가능함



마크다운 특징

- 마크다운은 단순 텍스트 문법으로 내용을 작성하며, 다양한 환경에서 변환하여 보여짐
 - text editor, 웹 환경 등에서 모두 지원(HTML 변환)

C:\> Users > takeh> Desktop > 마크다운.md > # 마크다운

```
1 # 마크다운
2
3 ## 개요
4
5 2004년 존 그루버가 만든 텍스트 기반의 가벼운 마크업 언어
6
7
8 > Markdown is a text-to-HTML conversion tool for web writers. Markdown allows you to write using an easy-to-read, easy-to-write plain text format, then convert it to structurally valid XHTML (or HTML).
9 > Thus, "Markdown" is two things: (1) a plain text formatting syntax; and (2) a software tool, written in Perl, that converts the plain text formatting to HTML.
10 > See the Syntax page for details pertaining to Markdown's formatting syntax.
11 > The overriding design goal for Markdown's formatting syntax is to make it as readable as possible.
12
13 [출처](https://daringfireball.net/projects/markdown/)
14
15 ## 비교
16
17 한글/MS word
18
19 • 워드프로세서는 다양한 서식과 구조를 지원하며, 문서에 즉각적으로 반영
20
21 • 마크다운은 단순 텍스트 문법으로 내용을 작성하며, 다양한 환경에서 변환하여 보여짐
22
23
```

Ln 1, Col 1 Spaces: 4 UTF-8 CRLF Markdown ⚡ 🔍

마크다운

개요

2004년 존 그루버가 만든 텍스트 기반의 가벼운 마크업 언어

Markdown is a text-to-HTML conversion tool for web writers. Markdown allows you to write using an easy-to-read, easy-to-write plain text format, then convert it to structurally valid XHTML (or HTML). Thus, "Markdown" is two things: (1) a plain text formatting syntax; and (2) a software tool, written in Perl, that converts the plain text formatting to HTML. See the Syntax page for details pertaining to Markdown's formatting syntax. The overriding design goal for Markdown's formatting syntax is to make it as readable as possible.

비교

한글/MS word

- 워드프로세서는 다양한 서식과 구조를 지원하며, 문서에 즉각적으로 반영
- 마크다운은 단순 텍스트 문법으로 내용을 작성하며, 다양한 환경에서 변환하여 보여짐

Search or jump to... Pull requests Issues Marketplace Explore

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

main · t branch · 0 tags Go to file Add file + About No description, website, or topics provided.

README.md Create README.md 21 seconds ago Readme

README.md Releases No releases published Create a new release

마크다운

개요

2004년 존 그루버가 만든 텍스트 기반의 가벼운 마크업 언어

Markdown is a text-to-HTML conversion tool for web writers. Markdown allows you to write using an easy-to-read, easy-to-write plain text format, then convert it to structurally valid XHTML (or HTML). Thus, "Markdown" is two things: (1) a plain text formatting syntax and (2) a software tool, written in Perl, that converts the plain text formatting to HTML. See the Syntax page for details pertaining to Markdown's formatting syntax. The overriding design goal for Markdown's formatting syntax is to make it as readable as possible.

비교

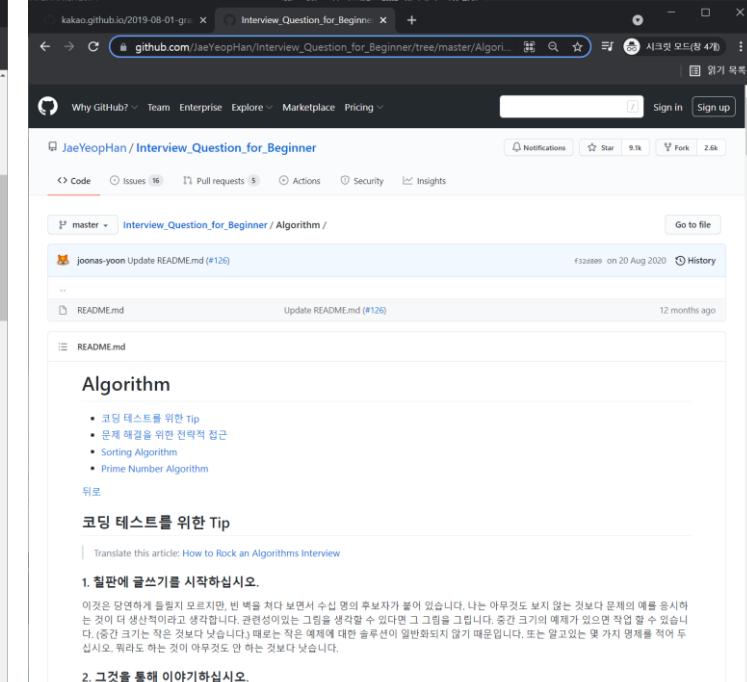
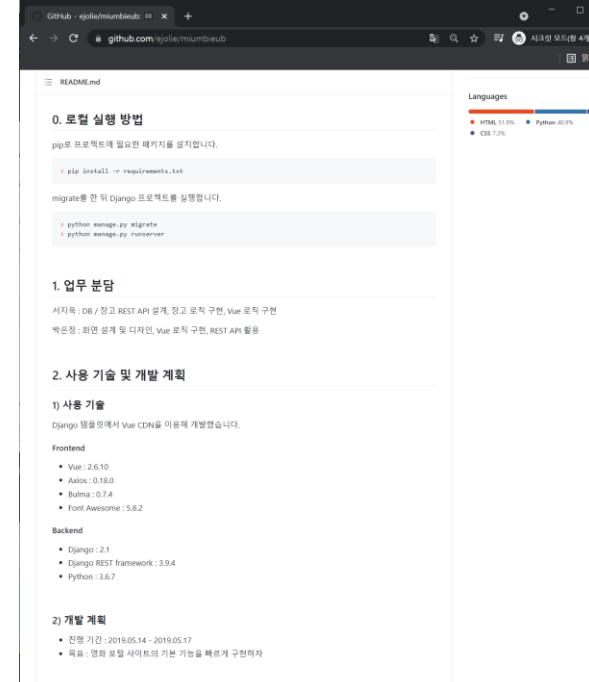
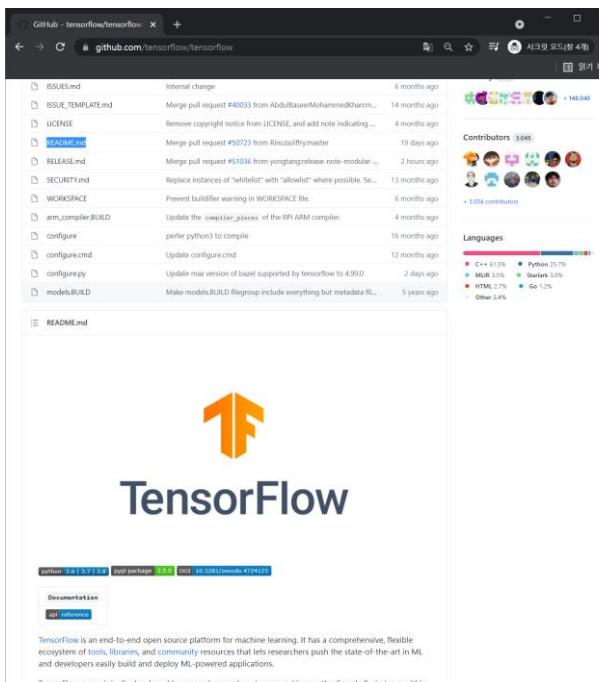
한글/MS word

- 워드프로세서는 다양한 서식과 구조를 지원하며, 문서에 즉각적으로 반영
- 마크다운은 단순 텍스트 문법으로 내용을 작성하며, 다양한 환경에서 변환하여 보여짐

© 2021 GitHub, Inc. Terms Privacy Security Status Docs Contact GitHub Pricing API Training Blog About

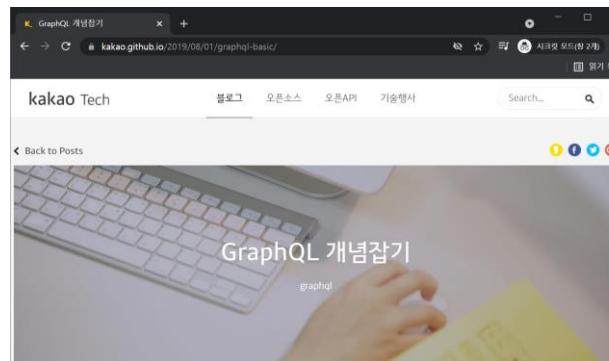
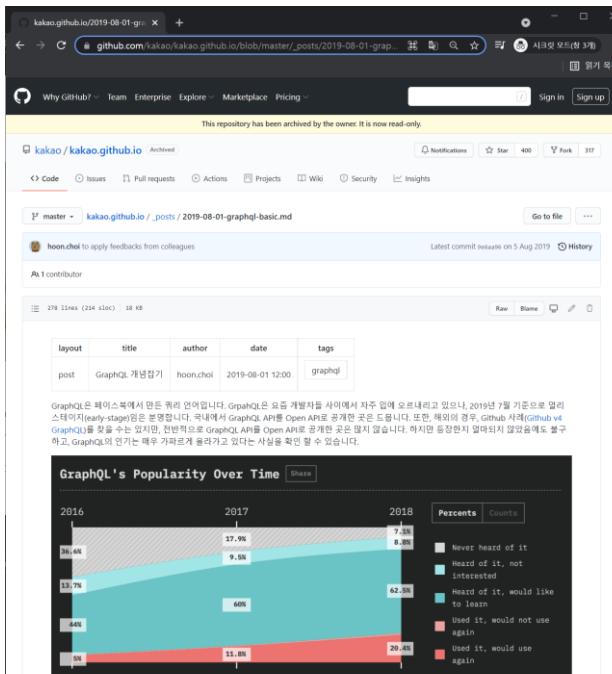
마크다운 활용 예시

- README.md
 - 오픈소스의 공식 문서를 작성하거나 개인 프로젝트의 프로젝트 소개서로 활용
 - 혹은 모든 페이지에 README.md를 넣어 문서를 바로 볼 수 있도록 활용



마크다운 활용 예시

- 정적사이트생성기(Static site generator) 기반 블로그
 - Jekyll, Gatsby, Hugo, Hexo 등으로 작성된 마크다운을 HTML, CSS, JS 파일 등으로 변환하고, Github pages 기능을 통해 무료 호스팅이 가능함

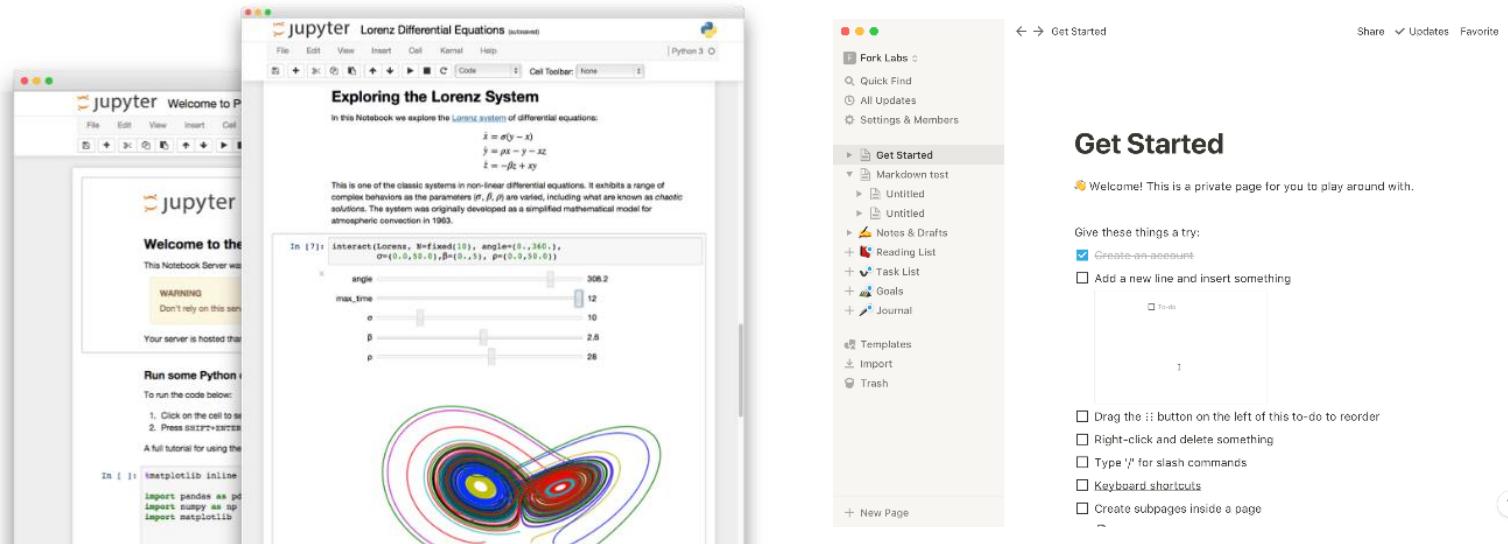


GraphQL은 페이스북에서 만든 뭐라 언어입니다. GraphQL은 요즘 개발자들 사이에서 자주 입에 오르내리고 있으나, 2019년 7월 기준으로 알려지지(early-stage) 않은 분위입니다. 국내에서 GraphQL API를 Open API로 공개한 곳은 드뭅니다. 또한, 해외의 경우, Github 사례(Github v4 GraphQL)을 찾을 수는 있지만, 전방적으로 GraphQL API를 Open API로 공개한 곳은 많지 않습니다. 하지만 등장한지 얼마되지 않았음에도 불구하고, GraphQL의 인기는 매우 기파르게 올라가고 있다는 사실을 확인 할 수 있습니다.



마크다운 활용 예시

- 마크다운 기반 SW
 - Jupyter notebook에는 별도의 마크다운 셀으로 프로젝트 내용과 분석 결과를 정리 가능함
 - Notion과 같은 메모/노트 필기 SW 역시 마크다운 기반 문서 작성을 기본으로 함



문서 작성은 시작하기 전에…

마크다운 기반 문서 작성

hypergrowth

위키백과
우리 모두의 백과사전

위키백과와 검색

위키미디어 송년회가 12월 31일 오후 3시에 신도림역 대회의실에서 열립니다.

마크다운

문서 토론

읽기 편집 역사 보기

마크다운(Markdown)은 일반 텍스트 기반의 경량 마크업 언어다. 일반 텍스트로 서식이 있는 문서를 작성하는 데 사용되며, 일반 마크업 언어에 비해 문법이 쉽고 간단한 것이 특징이다. HTML과 리치 텍스트(RTF) 등 서식 문서로 쉽게 변환되기 때문에 [훌륭한 소프트웨어](#)와 함께 배포되는 README 파일이나 온라인 게시판 등에 많이 사용된다.

역사

존 그루버는 2004년에 문법 면에서 예쁜 스위즈와 중대한 협업을 통해 마크다운 언어를 만들었으나,[2][4] 사람들이 읽기 쉽고 쓰기 좋은 플레인 텍스트 포맷을 사용하여 알 수 있으면서 구조적으로 유연한 XHTML(또는 HTML)로 선형적 변환이 가능하게 하는 것이 목표이다.[5]

문법

마크다운에서는 엔터기 한 번을 먹지 않는다. 문장이 같은 줄에 연이어 표시된다.

마크다운	작용결과
첫째 문장.	첫째 문장.
둘째 문장.	둘째 문장.
셋째 문장.	셋째 문장.

마크다운은 빈 칸 두 개로 구분한다.

마크다운	작용결과
첫째 문장.	첫째 문장.
둘째 문장.	둘째 문장.
셋째 문장.	셋째 문장.

문단과 문단 사이는 빈 줄(엔터기 두 번)로 구분한다.

하나의 문단.
다른 문단.

문단 제목은 다음과 같이 표현한다.

```
# 큰 제목  
## 중간 제목  
### 작은 제목
```



파일 확장자	nd
인터넷 미디어 타입	text/markdown[1]
개발	존 그루버, 예쁜 스위즈
발표일	2004년 3월 19일(18년 전)[2]
최신 버전	1.0.1 (2004년 12월 17일(18년 전)[3])
포맷 종류	마크업 언어
웹사이트	daringfireball.net/projects/markdown/ [4]

파이썬[2](영어: Python)은 1991년[3] 네덜란드 소프트웨어 엔지니어인 귀도 반 로섬[4]이 발표한 고급 프로그래밍 언어로, 플랫폼에 독립적이며 인터프리터식, 객체지향적, 동적 타이핑(dynamically typed) 대화형 언어이다. 파이썬이라는 이름은 귀도가 좋아하는 코미디인 <Monty Python's Flying Circus>에서 따온 것이다. 이름에서 고대신화에 나오는 커다란 뱀을 연상하는 경우도 있겠지만, 이와는 무관하다. 다만 로고에는 뱀 두 마리가 형상화되어 있다.

파이썬은 비영리의 파이썬 소프트웨어 재단이 관리하는 개방형, 공동체 기반 개발 모델을 가지고 있다.

개요

파이썬은 조보자부터 전문가까지 사용자층을 보유하고 있다. 동적 타이핑(dynamic typing) 범용 프로그래밍 언어로, 편 및 루비와 자주 비교된다. 다양한 플랫폼에서 쓸 수 있고, 라이브러리(모듈)가 풍부하여, 대학을 비롯한 여러 교육 기관, 연구 기관 및 산업체에서 이용이 증가하고 있다. 또 파이썬은 순수한 프로그램 언어로서의 기능 외에도 다른 언어로 쓰인 모듈들을 연결하는 접착제 언어로써 자주 이용된다. 실제 파이썬은 많은 상용 웹용 프로그램에서 스크립트 언어로 채용되고 있다. 도움말 문서도 정기적 갱신 되어 있으며, 유니코드 문자열을 지원해서 다양한 언어의 문자 처리에도 가능하다.

파이썬은 기본적으로 해석기(인터프리터) 위에서 실행될 것을 염두에 두고 설계되었다.

주요 특징

- 동적 타이핑(dynamic typing). (실행 시간에 자료형을 검사한다)
- 객체의 웹버에 무제한으로 접근할 수 있다. (속성이나 전용의 메서드 혹은 만들어 제한할 수는 있음)
- 모듈, 클래스, 객체와 같은 언어의 요소가 내부에서 접근할 수 있고, 리플렉션을 이용한 기술을 쓸 수 있다.
- 해석 프로그램의 종류
 - Cython: C로 작성된 인터프리터.
 - 스택리스 파이썬: C 스택을 사용하지 않는 인터프리터.
 - 자바썬: 자바에서 머신용 인터프리터. 과거에는 제이파이썬(JPython)이라고 불렸다.
 - IronPython: .NET 플랫폼용 인터프리터.
 - PyPy: 파이썬으로 작성된 파이썬 인터프리터.

현대의 파이썬은 여전히 인터프리터 언어처럼 통작이나 사용자가 모르는 사이에 스스로 파이썬 소스 코드를 컴파일하여 바이트 코드(byte code)를 만들어 널로써 다음에 수행할 때에는 빠른 속도를 보여 준다.

파이썬에서는 둘째쓰기를 사용해서 복록을 구분하는 독특한 문법을 채용하고 있다. 이 문법은 파이썬에 의해 사용자나 기존 프로그래밍 언어에서 들여쓰기의 중요성을 높이 평가하는 사용자에게는 잘 받아들여지고 있지만, 다른 언어의 사용자에게는 프로그래머의 코딩 스타일을 제한한다는 비판도 많다. 이 밖에도 실행 시간에서奔 아니네 네이티브 이진 파일을 만들어 주는 C/C++ 등의 언어에 비해 수행 속도가 느린다는 단점이 있다. 그러나 사업 분야 등 일반적인 컴퓨터 용융 환경에서는 속도가 그리 중요하지 않고, 빠른 속도를 요하는 프로그램의 경우에도 프로토 타이핑한 빠른 속도가 필요한 부분만 끌라서 C 언어 등으로 모듈화할 수 있다([ctypes](#), [SWIG](#), [SIP](#) 등의 래퍼 생성 프로그램들이 많이 있다). 또한 [Pyrex](#), [Psyco](#), [NumPy](#) 등을 이용하면 수치를 빠르게 연산할 수 있기 때문에 과학·공학 분야에서도 많이 이용되고 있다. 절차적인 중요성이 강조 대한민국에서도 점차 그 활용도가 커지고 있다.[5]

역사

이 부분의 본문은 파이썬의 역사입니다.

파이썬은 1980년대 말 고안되어 네덜란드 CWI의 귀도 반 로섬이 1989년 12월 구현하기 시작하였다. 이는 역시 SETL에서 영감을 받은 ABC 언어의 후계로서, 예외 처리가 가능하고, 아메바 OS와 연동이 가능하였다. 반 로섬은 파이썬의 주 저자로 계속 중심적 역할을 맡아 파이썬의 방향을 결정하여, 파이썬 공동체로부터 '자선 중심 이사'의 칭호를 부여받았다. 이 같은 예로는 리눅스의 리눅스 토발즈 등이 있다.

파이썬
Python

파이썬은 비영리의 파이썬 소프트웨어 재단이 관리하는 개방형, 공동체 기반 개발 모델을 가지고 있다.

파이썬은 조보자부터 전문가까지 사용자층을 보유하고 있다. 동적 타이핑(dynamic typing) 범용 프로그래밍 언어로, 편 및 루비와 자주 비교된다. 다양한 플랫폼에서 쓸 수 있고, 라이브러리(모듈)가 풍부하여, 대학을 비롯한 여러 교육 기관, 연구 기관 및 산업체에서 이용이 증가하고 있다. 또 파이썬은 순수한 프로그램 언어로서의 기능 외에도 다른 언어로 쓰인 모듈들을 연결하는 접착제 언어로써 자주 이용된다. 실제 파이썬은 많은 상용 웹용 프로그램에서 스크립트 언어로 채용되고 있다. 도움말 문서도 정기적 갱신 되어 있으며, 유니코드 문자열을 지원해서 다양한 언어의 문자 처리에도 가능하다.

파이썬은 기본적으로 해석기(인터프리터) 위에서 실행될 것을 염두에 두고 설계되었다.

주요 특징

- 동적 타이핑(dynamic typing). (실행 시간에 자료형을 검사한다)
- 객체의 웹버에 무제한으로 접근할 수 있다. (속성이나 전용의 메서드 혹은 혹은 만들어 제한할 수는 있음)
- 모듈, 클래스, 객체와 같은 언어의 요소가 내부에서 접근할 수 있고, 리플렉션을 이용한 기술을 쓸 수 있다.
- 해석 프로그램의 종류
 - Cython: C로 작성된 인터프리터.
 - 스택리스 파이썬: C 스택을 사용하지 않는 인터프리터.
 - 자바썬: 자바에서 머신용 인터프리터. 과거에는 제이파이썬(JPython)이라고 불렸다.
 - IronPython: .NET 플랫폼용 인터프리터.
 - PyPy: 파이썬으로 작성된 파이썬 인터프리터.

주요 구현체

C파이썬, IronPython, 자바썬, 마이크로파이썬, 누비, PyPy, 스택리스 파이썬, 사이언, R파이썬

영향을 받은 언어

ABC, 알골 68, C, C++, 딜란, 하스켈, 아이콘, 누비, 리스프, 모듈리-3, 편

영향을 준 언어

부, 코브라, 커피스크립트, D, F#, 웜컨, 제니, Go, 그루비, 자바스크립트, 줄리아, 님, 루비, 스위프트



마크다운 문법

- Heading : 문서의 제목이나 소제목
 - #의 개수에 따라 대응되는 수준(Heading level)이 있으며, h1 ~ h6까지 표현 가능
 - 문서의 구조를 위해 작성되며 글자 크기를 조절하기 위해 사용되어서는 안됨

Markdown	HTML	Rendered Output
# Heading level 1	<h1>Heading level 1</h1>	Heading level 1
## Heading level 2	<h2>Heading level 2</h2>	Heading level 2
### Heading level 3	<h3>Heading level 3</h3>	Heading level 3
#### Heading level 4	<h4>Heading level 4</h4>	Heading level 4
##### Heading level 5	<h5>Heading level 5</h5>	Heading level 5
###### Heading level 6	<h6>Heading level 6</h6>	Heading level 6

<input checked="" type="checkbox"/> Do this	<input type="checkbox"/> Don't do this
# Here's a Heading	#Here's a Heading
<input checked="" type="checkbox"/> Do this	<input type="checkbox"/> Don't do this
Try to put a blank line before... # Heading ...and after a heading.	Without blank lines, this might not look right. # Heading Don't do this!

마크다운 문법

- List : 목록
 - 순서가 있는 리스트(ol)와 순서가 없는 리스트.ul)로 구성
 - Tab으로 하위 항목으로 구성할 수 있음

Markdown	HTML	Rendered Output
1. First item		
2. Second item		
3. Third item		
4. Fourth item		
	 First item Second item Third item Fourth item 	1. First item 2. Second item 3. Third item 4. Fourth item

Markdown	HTML	Rendered Output
- First item - Second item - Third item - Fourth item	 First item Second item Third item Fourth item 	• First item • Second item • Third item • Fourth item

마크다운 문법

- Fenced Code block : 코드 블록
 - 코드 블록은 backtick 기호 3개를 활용하여 작성(" " ")
 - 코드 블록에 특정 언어를 명시하면 Syntax Highlighting 적용 가능

```
```  
{
 "firstName": "John",
 "lastName": "Smith",
 "age": 25
}
```
```

The rendered output looks like this:

```
{  
  "firstName": "John",  
  "lastName": "Smith",  
  "age": 25  
}
```

```
```json  
{
 "firstName": "John",
 "lastName": "Smith",
 "age": 25
}
```
```

The rendered output looks like this:

```
{  
  "firstName": "John",  
  "lastName": "Smith",  
  "age": 25  
}
```

마크다운 문법

- **Inline Code block** : 코드 블록
 - 코드 블록은 backtick 기호 1개를 인라인에 활용하여 작성('')

| Markdown | HTML | Rendered Output |
|--|---|--------------------------------------|
| At the command prompt, type
`nano`. | At the command prompt, type
<code>nano</code>. | At the command prompt,
type nano. |

마크다운 문법

- link : 링크
 - [문자열](url)을 통해 링크 작성 가능

To create a link, enclose the link text in brackets (e.g., [Duck Duck Go]) and then follow it immediately with the URL in parentheses (e.g., (<https://duckduckgo.com>)).

```
My favorite search engine is [Duck Duck Go](<a href="https://duckduckgo.com">https://duckduckgo.com</a>).
```

The rendered output looks like this:

My favorite search engine is [Duck Duck Go](https://duckduckgo.com).

마크다운 문법

- image : 이미지
 - `![문자열](url)`을 통해 이미지를 사용 가능
 - 특정 파일들 포함하여 연결 시킬 수도 있음



마크다운 문법

- Blockquotes : 인용문
 - >으로 인용문을 작성

To create a blockquote, add a > in front of a paragraph.

```
> Dorothy followed her through many of the beautiful rooms in her castle.
```

The rendered output looks like this:

Dorothy followed her through many of the beautiful rooms in her castle.

마크다운 문법

- 테이블

| Syntax | Description |
|-----------|-------------|
| ----- | ----- |
| Header | Title |
| Paragraph | Text |

The rendered output looks like this:

| Syntax | Description |
|-----------|-------------|
| Header | Title |
| Paragraph | Text |

마크다운 문법

- 텍스트 강조
 - 굵게(bold), 기울임(Italic)을 통해 특정 글자들을 강조

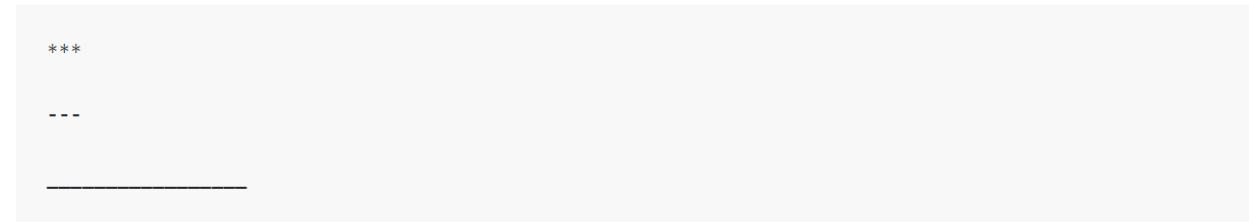
| Markdown | HTML | Rendered Output |
|----------------------------|---|--------------------------------|
| I just love **bold text**. | I just love bold text. | I just love bold text . |
| I just love __bold text__. | I just love bold text. | I just love bold text . |
| Love**is**bold | Loveisbold | Love is bold |

| Markdown | HTML | Rendered Output |
|--------------------------------------|---|--|
| Italicized text is the *cat's meow*. | Italicized text is the cat's meow. | Italicized text is the <i>cat's meow</i> . |
| Italicized text is the _cat's meow_. | Italicized text is the cat's meow. | Italicized text is the <i>cat's meow</i> . |
| A*cat*meow | Acatmeow | A <i>cat</i> meow |

마크다운 문법

- 수평선
 - 3개 이상의 asterisks (***) , dashes (---) , or underscores (____)

To create a horizontal rule, use three or more asterisks (***) , dashes (---) , or underscores (____) on a line by themselves.



The rendered output of all three looks identical:

마크다운 추가 자료

- GitHub Flavored Markdown (<https://github.github.com/gfm/>)
- Mastering Markdown (<https://guides.github.com/features/mastering-markdown/>)
- Markdown Guide (<https://www.markdownguide.org/>)

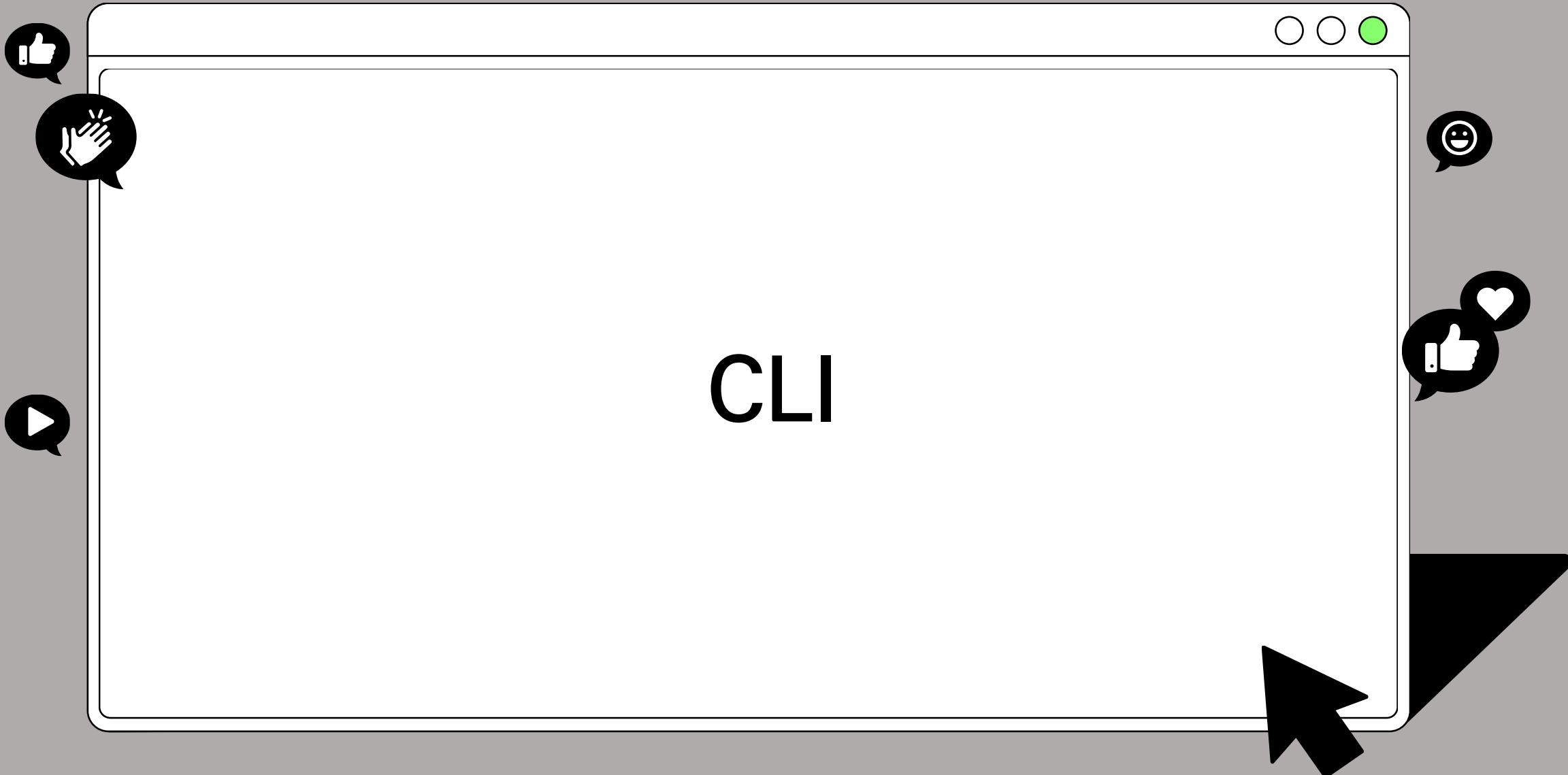
개발자에게 문서 작성이란..?

- 백엔드 개발자를 꿈꾸는 학생 개발자들에게 (<https://d2.naver.com/news/3435170>)
 - 레벨 2 개발자 : ‘자신이 경험한 사용법을 문서화해서 팀 내에 전파할 수 있음’
- Google Technical Writing (<https://developers.google.com/tech-writing>)
 - Every engineer is also a writer
- Technical writing conference (<https://engineering.linecorp.com/ko/blog/write-the-docs-prague-2018-recap/>)
 - Clova 기술 문서 작성 및 관리 업무

마크다운 실습

- 지금까지 배운 마크다운 문법으로 마크다운 정리 문서를 만들고 제출하세요.
 - 참고) <https://www.markdownguide.org/cheat-sheet/>

CLI



학습 목표

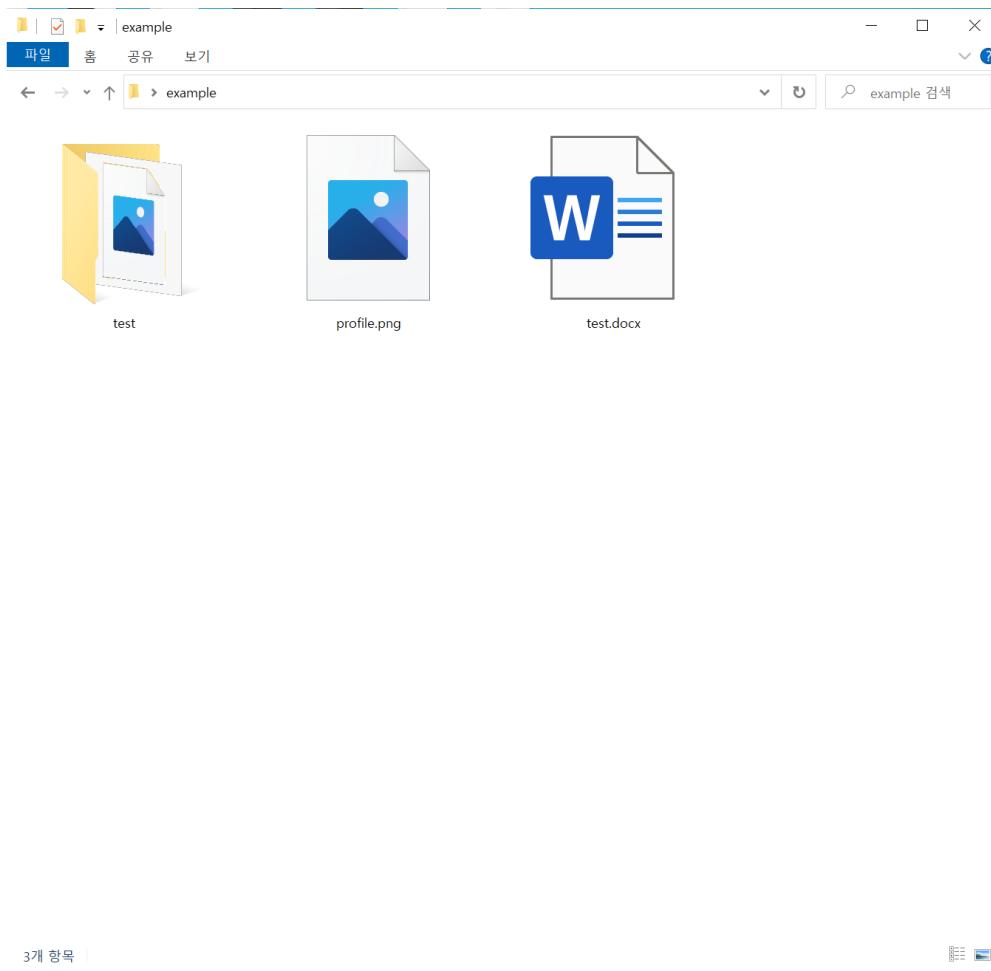
CLI 환경과 GUI 환경의 차이점을 비교할 수 있다.

CLI 환경에서 기초 파일시스템 조작을 할 수 있다.

CLI (Command Line Interface)

- CLI (커맨드 라인 인터페이스) 또는 명령어 인터페이스는 가상 터미널 또는 텍스트 터미널을 통해 사용자와 컴퓨터가 상호 작용하는 방식을 뜻한다.
- 작업 명령은 사용자가 툴바 키보드 등을 통해 문자열의 형태로 입력하며, 컴퓨터로부터의 출력 역시 문자열의 형태로 주어진다.
- 이 같은 인터페이스를 제공하는 프로그램을 명령 줄 해석기 또는 셸이라고 부른다. 이를테면, 유닉스 셸(sh, ksh, csh, tcsh, bash 등)과 CP/M, 도스의 command.com("명령 프롬프트") 등이 있다.

GUI vs CLI



```
MINGW64:/c/Users/lec/Desktop/example
$ ls
profile.png  test/  test.docx

$
```

**터치 기반의 스마트폰
키보드 마우스 기반의 컴퓨터
‘전혀 다르게’ 생각하고 조작**

GUI - 그래픽 기반의 인터페이스

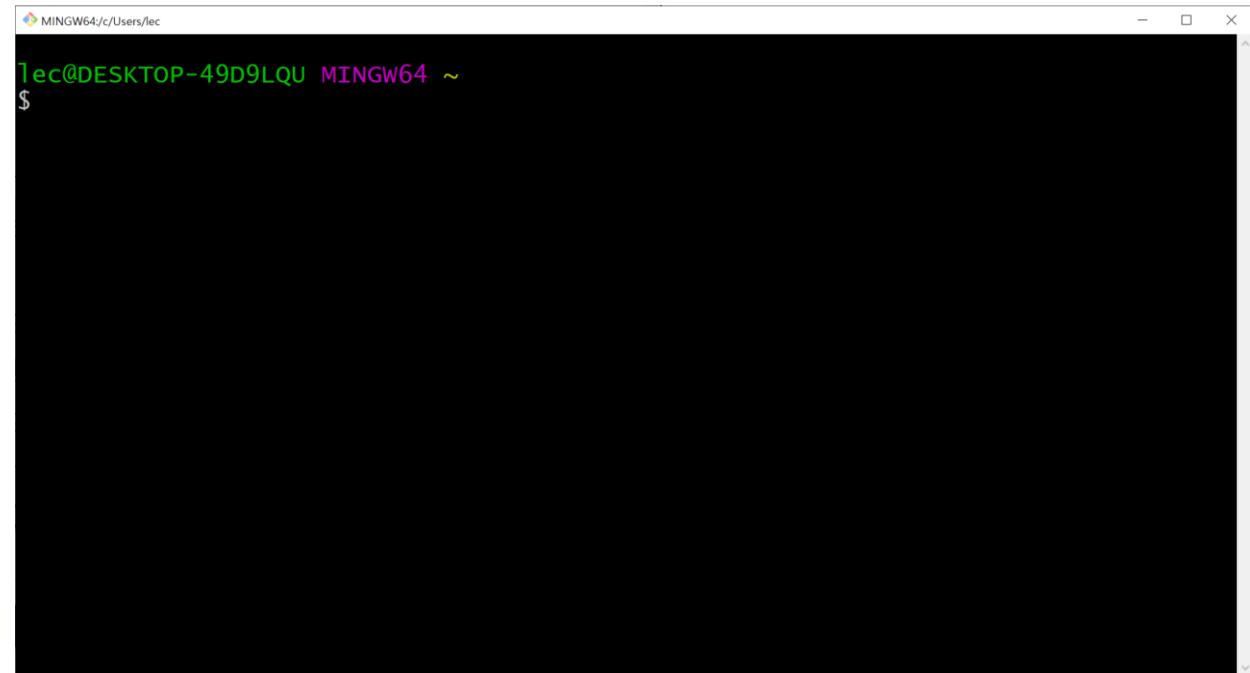
CLI - 명령 기반의 인터페이스

**내가 무엇인가를 알고 싶으면,
명령을 하고 그 결과를 읽어야 한다.**

불편한 것이 아니라
‘전혀 다르게’ 생각하고 조작하자.

CLI 구성 요소

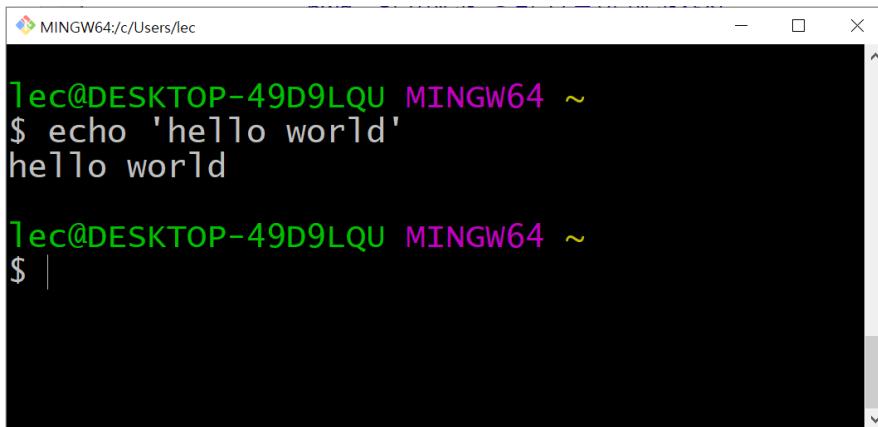
- 프롬프트 기본 인터페이스
 - 컴퓨터 정보
 - 디렉토리
 - ~의 경우 홈 디렉토리를 의미
 - \$



A screenshot of a Windows terminal window titled "MINGW64/c/Users/lec". The window shows a command prompt with the text "Tec@DESKTOP-49D9LQU MINGW64 ~" followed by a dollar sign (\$) indicating where input can be entered.

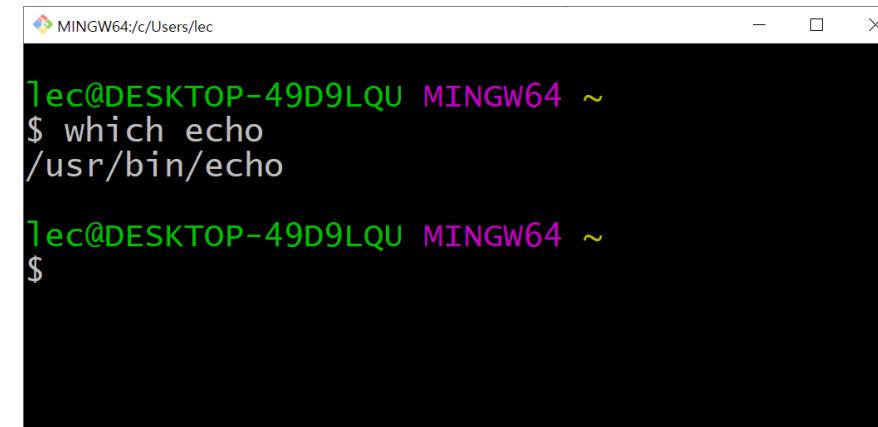
CLI 명령어 구조

- 명령어 기본 구조
 - 특정 프로그램을 어떠한 인자와 함께 호출하도록 명령
 - 예) echo 프로그램을 ‘hello world’로 호출



```
MINGW64:/c/Users/lec
$ echo 'hello world'
hello world

$ |
```



```
MINGW64:/c/Users/lec
$ which echo
/usr/bin/echo

$ |
```

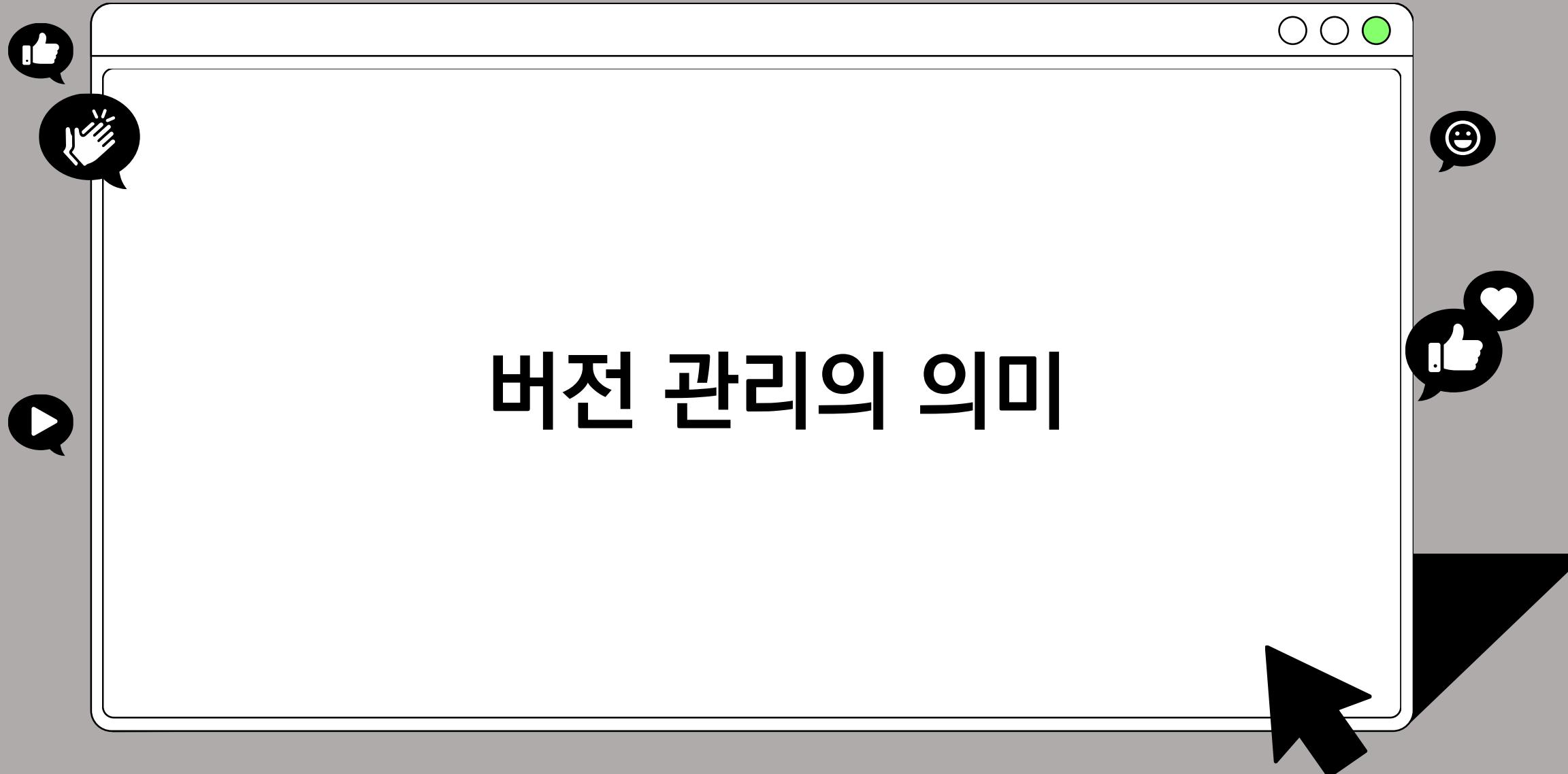
기초 파일시스템 명령어

- `pwd` (print working directory) : 현재 디렉토리 출력
- `cd` 디렉토리이름(change directory) : 디렉토리 이동
 - .. : 현재 디렉토리, ... : 상위 디렉토리
- `ls` (list) : 목록
- `mkdir` (make directory) : 디렉토리 생성
- `touch` : 파일 생성
- `rm` 파일명: 파일 삭제하기
 - `rm -r` 폴더명 : 폴더 삭제하기

CLI 조작 실습

- <https://bit.ly/kdt-cli>
- CLI 명령어를 활용하여 아래에 해당하는 이름을 확인한다.
 - 1반 13번
 - 5반 11번
 - 3반 15번
- CLI 명령어를 활용하여 새로운 폴더와 파일을 만든다.
 - 6반 1번 본인 이름.txt
 - 6반 2번 친구 이름.txt

버전 관리의 의미



학습 목표

버전관리의 의미를 이해하고 버전관리도구의 종류를 설명할 수 있다.





분산 버전 관리 시스템

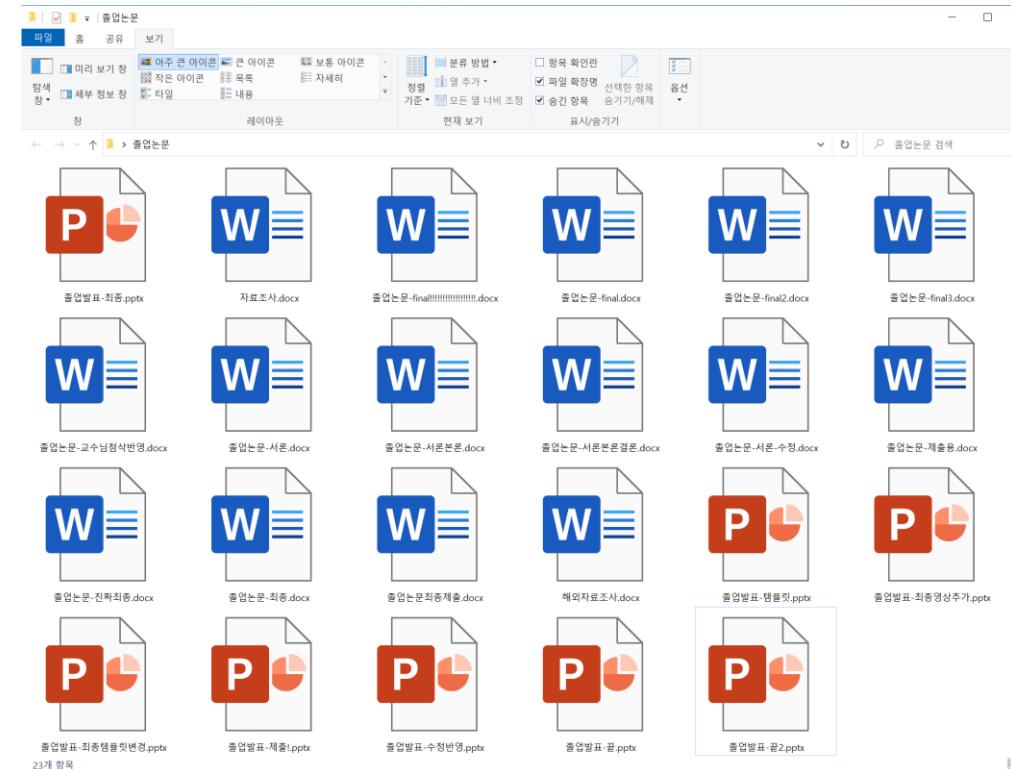
버전 관리?

버전 관리!

컴퓨터 소프트웨어의 특정 상태

우리가 알고 있는 버전관리

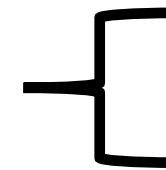
일반적인 우리의 버전관리 방식



일반적인 우리의 버전관리 방식



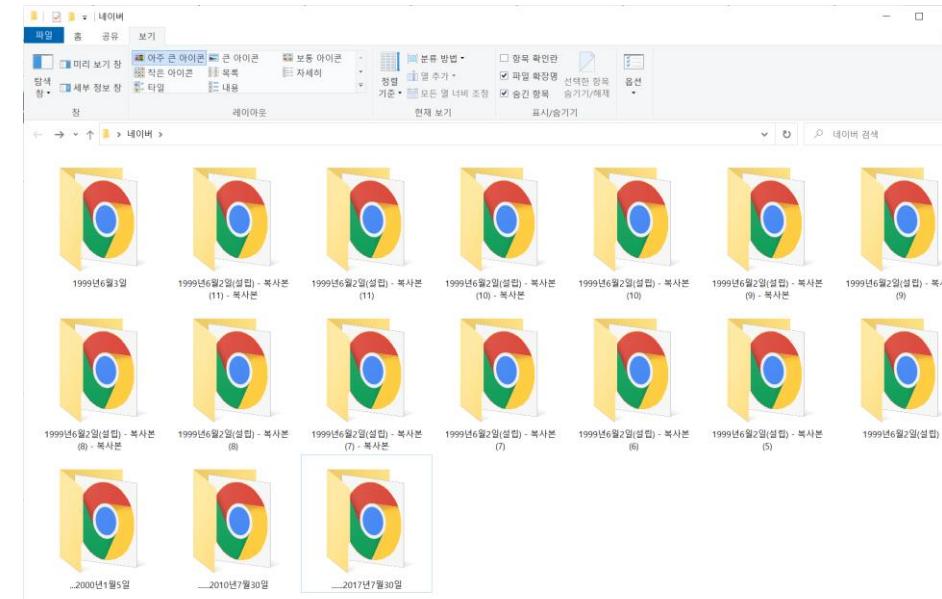
이 자료 간에 뭐가 바뀌었는지
차이(diff)를 알 수 없다.



졸업논문_1차.hwp
졸업논문_수정.hwp
졸업논문_수정_2.hwp
졸업논문_최종본.hwp
졸업논문_진짜최종.hwp
졸업논문_진짜최종이다.hwp
졸업논문_레알진짜킹갓최종.hwp

개발 환경에서의 버전 관리

- 네이버의 홈페이지의 소스코드 버전 관리를 상상해보자.
 - 1999년에 설립된 네이버
 - 매일 업데이트된다고 할 때, 모든 소스코드를 복제해서 관리하고 있을까?



실제 오픈소스는?

오픈소스(크로미움)

- 크로미움(크롬 브라우저의 오픈소스)
 - 최신 버전의 용량 1.58GB
 - 현재까지 1,000,000여개의 커밋(버전) 20,000여개의 릴리즈

 **Chromium**

Chromium is an open-source browser project that aims to build a safer, faster, and more stable way for all users to experience the web.

The project's web site is <https://www.chromium.org>.

To check out the source code locally, don't use `git clone`! Instead, follow [the instructions on how to get the code](#).

Documentation in the source is rooted in [docs/README.md](#).

Learn how to [Get Around the Chromium Source Code Directory Structure](#).

For historical reasons, there are some small top level directories. Now the guidance is that new top level directories are for product (e.g. Chrome, Android WebView, Ash). Even if these products have multiple executables, the code should be in subdirectories of the product.

If you found a bug, please file it at <https://crbug.com/new>.

master 8 branches 21,117 tags Go to file Code

chromium-autoroll and Chromium LUCI CQ Roll Chromite from 923a131e2a... 0743f39 40 minutes ago 1,029,316 commits

| Folder | Commit Message | Time Ago |
|-----------------|---|----------------|
| android_webview | Add cookie_partition_key argument to CanonicalCookie::Create. | 1 hour ago |
| apps | Merge //base/util/values into //base/json | 7 days ago |
| ash | Refactor how AppListView gets drag state | 2 hours ago |
| base | Check getClassificationStatus() before getConfidenceScore() | 1 hour ago |
| build | Android: Switch build scripts to use python3 (reland) | 1 hour ago |
| build_overrides | Reland "Replace 'blacklist' with 'ignorelist' in ./tools/msan." | 2 months ago |
| buildtools | update re-client/chromium-win-nacl-cross to follow nacl roll | 18 days ago |
| cc | Add GLES2 usage for creating shared image with Passthrough | 1 hour ago |
| chrome | Chromium Updater system level install should have different permissions | 40 minutes ago |
| chromecast | Glue Together Pieces of CastRuntimeService | 23 hours ago |
| chromeos | Personalization: button background polish | 1 hour ago |
| cloud_print | Update OWNERS for translation artifacts | 11 days ago |
| codelabs | [owners] Ensure OWNERS file ends with newline in // | 2 months ago |
| components | Add null check for embedding app's package name | 43 minutes ago |
| content | Add cookie_partition_key argument to CanonicalCookie::Create. | 1 hour ago |

About

The official GitHub mirror of the Chromium source

chromium.googlesource.com/chromium...

Readme

BSD-3-Clause License

Releases

21,117 tags

Packages

No packages published

Contributors 2,178

+ 2,167 contributors



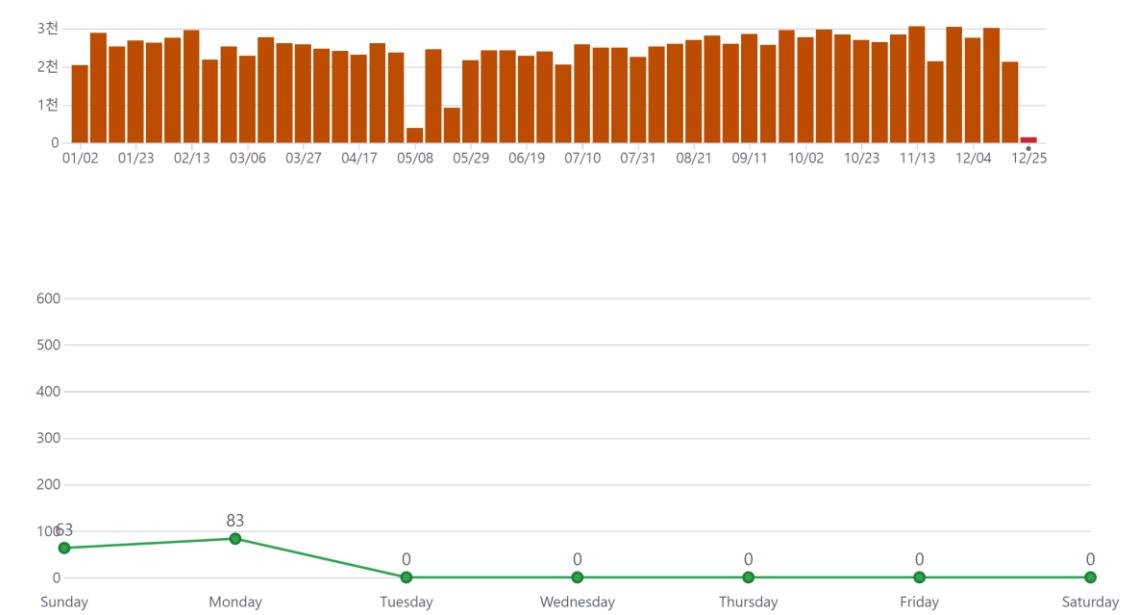
모든 버전의 용량은?

모든 버전의 용량은?
약 25GB

오픈소스(크로미움)

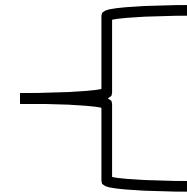
- 크로미움(크롬 브라우저의 오픈소스)
 - 커밋 단위로 버전 업데이트 현황을 볼 수 있다.

| Roll devtools-internal from 95cad5dfe634 to 14f6e298d8ce (1 revision) ... | 5c566df | <> |
|--|---------|----|
| chromium-internal-autoroll authored and Chromium LUCI CQ committed 7 hours ago | | |
| Roll Skia from 7442335dce20 to eeec7a127312 (1 revision) ... | 086bc91 | <> |
| chromium-autoroll authored and Chromium LUCI CQ committed 7 hours ago | | |
| Roll Media App from Xauc-4n48tia5Qvjt... to fPI9wT2OA39hd4Svw... ... | 1733b3b | <> |
| chromium-internal-autoroll authored and Chromium LUCI CQ committed 7 hours ago | | |
| Roll Help App from l5ciOW8zKntcTiXlo... to vCix3VGzFLpVKVyk... ... | d2f7f46 | <> |
| chromium-internal-autoroll authored and Chromium LUCI CQ committed 7 hours ago | | |
| Roll RTS model from sToGPH4uBVWaKk745... to iE7BlhwQe2SFGQyH7... ... | 7c045b9 | <> |
| chromium-autoroll authored and Chromium LUCI CQ committed 7 hours ago | | |
| Roll RTS model from bxc-LuKcrPzfO2L5i... to mD9ULgclbz87ARa5X... ... | 26f87e6 | <> |
| chromium-autoroll authored and Chromium LUCI CQ committed 7 hours ago | | |
| Roll RTS model from 2jw2JrFOGSPBtRKkC... to 9jLKrKNkfM5elR34Q... ... | 7860cf4 | <> |
| chromium-autoroll authored and Chromium LUCI CQ committed 7 hours ago | | |
| third_party/mako: Update Mako to version 1.2.4 ... | 048cc41 | <> |
| yuki3 authored and Chromium LUCI CQ committed 8 hours ago | | |
| Roll Chrome Win64 PGO Profile ... | 5046656 | <> |
| chromium-autoroll authored and Chromium LUCI CQ committed 8 hours ago | | |
| Roll Dawn from 7b674ab973e5 to 2f0123d1f98e (1 revision) ... | e7acadb | <> |
| chromium-autoroll authored and Chromium LUCI CQ committed 8 hours ago | | |
| Roll Chrome Mac Arm PGO Profile ... | f7a9998 | <> |
| chromium-autoroll authored and Chromium LUCI CQ committed 8 hours ago | | |



버전 관리를 해주는 Git 덕분!

차이(diff)와 수정 이유를
메시지로 남길 수 있다.



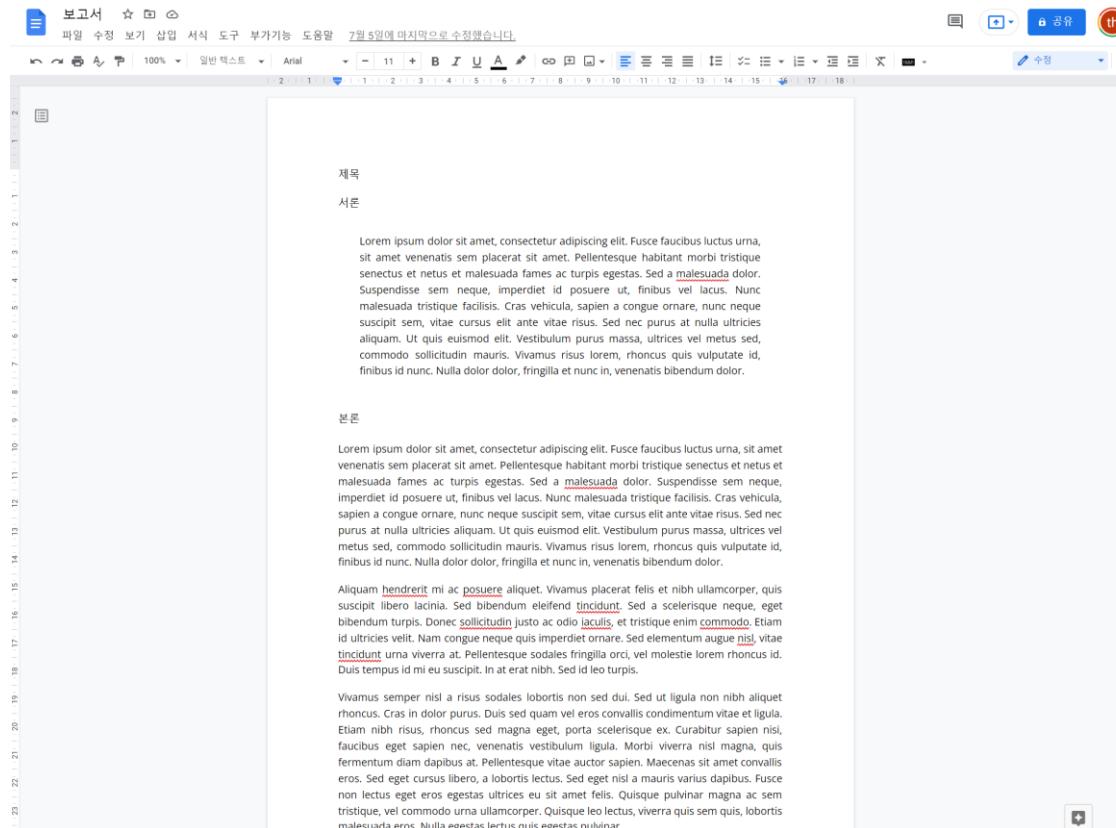
현재 파일들을 안전한 상태로
과거 모습 그대로 복원 가능 (반대도 마찬가지)

뼈대 코드구성
메인 기능 구현
로그인 기능 구현
채팅 기능 구현
디자인 적용



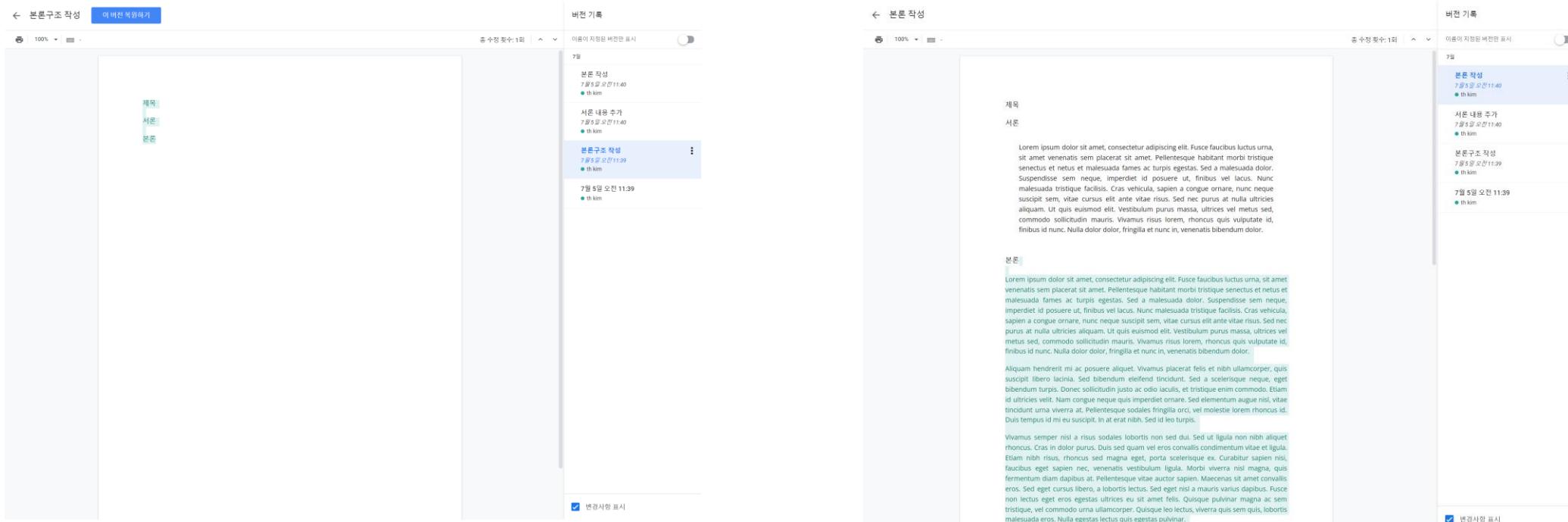
...

Google Document 버전 관리



버전 관리 예시

- Google Document 버전 관리
 - 문서는 하나지만 버전이 기록되어 있으면, 이전 시점을 조회하거나 복원시킬 수도 있음



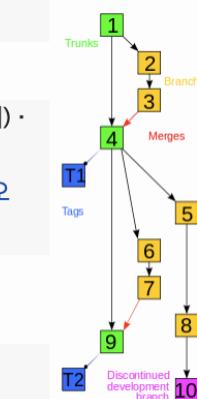
버전관리시스템

- In software engineering, version control (also known as revision control, source control, or source code management) is a class of systems responsible for **managing changes to** computer programs, documents, large web sites, or other **collections of information**.
- 버전관리, 소스코드 관리란 동일한 정보에 대한 여러 버전을 관리하는 것을 말한다.

버전 관리 소프트웨어

-

| V · T · E | | 버전 관리 소프트웨어 | [접기] |
|--|--|---|------|
| 연도로 표시된 부분은 최초 안정판의 날짜를 가리킨다. 별표(*)로 표시된 시스템은 더 이상 유지보수가 되지 않거나 EOL 날짜가 예정되어 있음을 나타낸다. | | | |
| 로컬 전용 | 자유/오픈 소스 | RCS (1982) · SCCS (1972) | |
| | 사유 | PVCS (1985) · QVCS* (1991) | |
| 클라이언트-서버 | 자유/오픈 소스 | CVS (1986, C의 경우 1990) · CVSNT (1998) · QVCS 엔터프라이즈 (1998) · 서브버전 (2000) | |
| | 사유 | AccuRev SCM (2002) · ClearCase (1992) · CMVC* (1994) · Dimensions CM (1980년대) · DSEE* (1984) · Endevor (1980년대) · Integrity (2001) · Panvalet (1970년대) · 퍼포스 헬릭스 (1995) · Software Change Manager (1970년대) · 스타팀 (1995) · 서라운드 SCM (2002) · Synergy (1990) · Team Concert (2008) · 팀 파운데이션 서버 (2005) · 마이크로소프트 비주얼 스튜디오 (2014) · Vault (2003) · 비주얼 소스세이프* (1994) | |
| 분산 | 자유/오픈-소스 | ArX* (2003) · 비트키퍼 (1998) · Codeville* (2005) · Darcs (2002) · DCVS* (2002) · Fossil (2007) · 깃 (2005) · GNU arch* (2001) · Bazaar (2005) · 머큐리얼 (2005) · 모노톤 (2003) · SVK* (2003) · Veracity (2010) | |
| | 사유 | TeamWare* (1990년대) · Code Co-op (1997) · 플라스틱 SCM (2006) · 팀 파운데이션 서버 (2013) · 마이크로소프트 비주얼 스튜디오 (2014) | |
| 개념 | 브랜치 · 포크 · 체인지셋 · 커밋 (게이티드 커밋) · Interleaved deltas · 델타 압축 · 데이터 비교 · 머지 · 저장소 · 태그 · Trunk | | |



Git 기초 흐름



Git

- Git은 분산버전관리시스템으로 코드의 버전을 관리하는 도구
- 2005년 리눅스 커널을 위한 도구로 리누스 토르발스가 개발
- 컴퓨터 파일의 변경사항을 추적하고 여러 명의 사용자들 간에 파일들의 작업을 조율

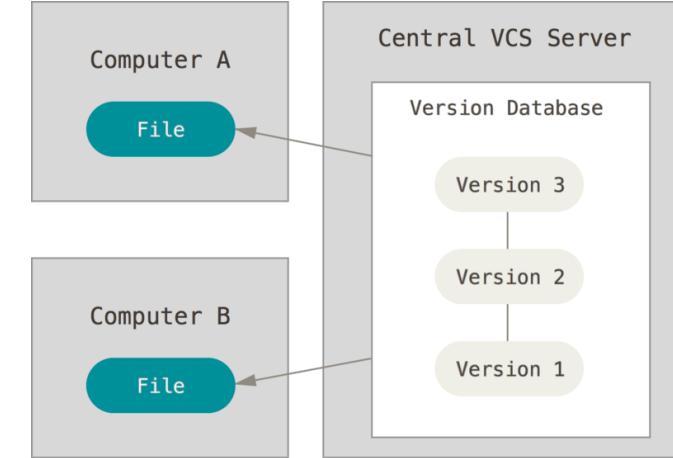


분산 버전 관리?

분산버전관리시스템(DVCS)

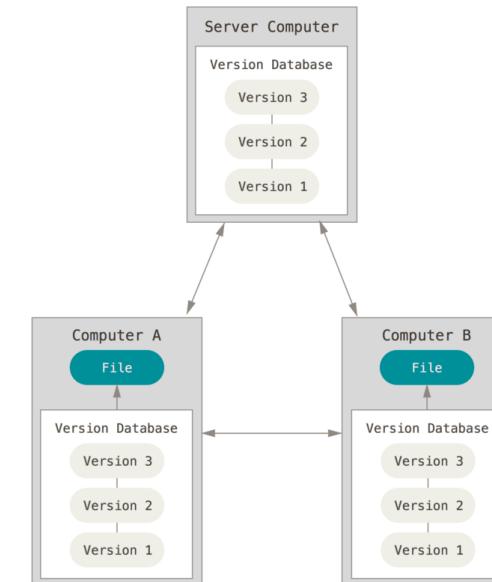
- 중앙집중식버전관리시스템

- 로컬에서는 파일을 편집하고 서버에 반영
- 중앙 서버에서만 버전을 관리



- 분산버전관리시스템

- 로컬에서도 버전을 기록하고 관리
- 원격 저장소(remote repository)를 활용하여 협업



저장소를 만들어보자!

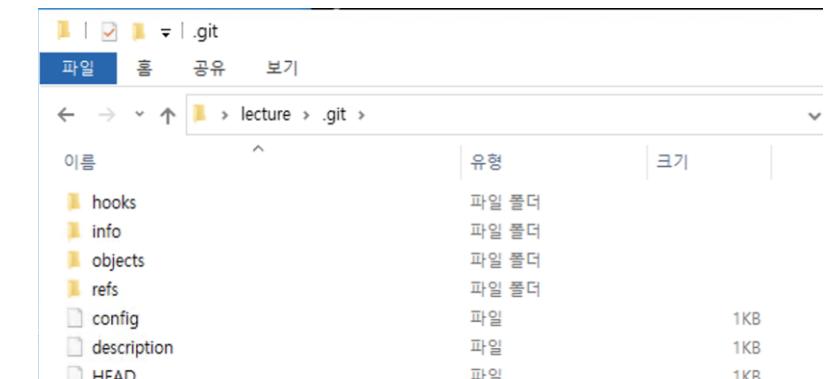
저장소 생성(초기화)

\$ git init

- 특정 폴더에 git 저장소(repository)를 만들고 버전 관리
 - .git 폴더가 생성되며
 - git bash에서는 (master)라는 표기를 확인할 수 있음

```
lecture@DESKTOP-49D9LQU MINGW64 ~/Desktop/lecture
$ git init
Initialized empty Git repository in C:/Users/takhe/Desktop/lecture/.git/

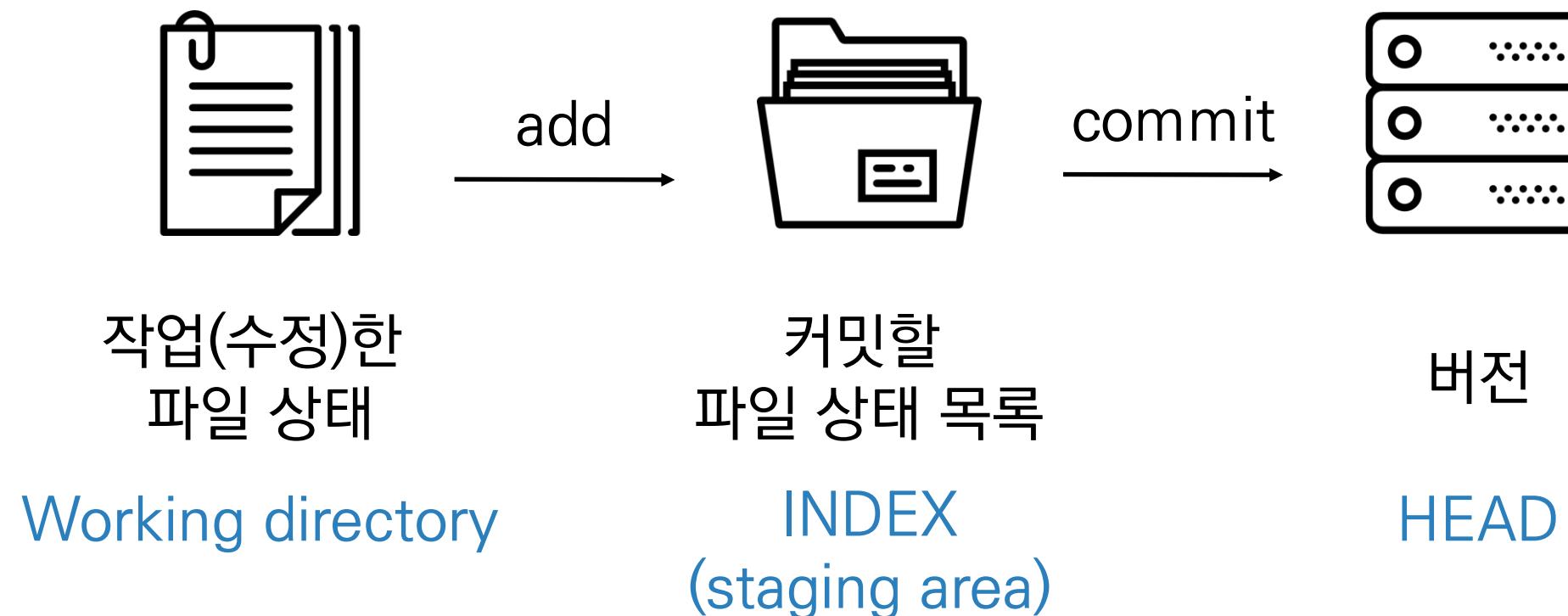
lecture@DESKTOP-49D9LQU MINGW64 ~/Desktop/lecture (master)
$ ls -al
total 44
drwxr-xr-x 1 lecture 197121 0 Mar 29 04:19 .
drwxr-xr-x 1 lecture 197121 0 Mar 29 04:16 ..
drwxr-xr-x 1 lecture 197121 0 Mar 29 04:19 .git/
```



버전은 어떻게 기록할까?

Git 버전 관리 기초 흐름

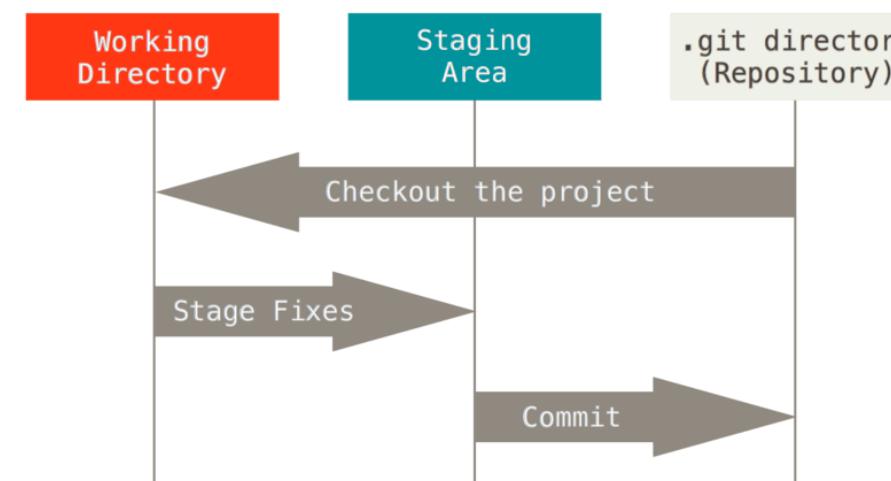
- 1) 작업하면 2) add하여 Staging area에 모아 3) commit으로 버전 기록



-
1. 작업을 하고
 2. 변경된 파일을 모아 (add)
 3. 버전으로 남긴다. (commit)

Git 버전 관리 기초 흐름

- Git은 파일을 modified, staged, committed로 관리
 - modified : 파일이 수정된 상태 (add 명령어를 통하여 staging area로)
 - staged : 수정한 파일을 곧 커밋할 것이라고 표시한 상태 (commit 명령어로 저장소)
 - committed : 커밋이 된 상태



Working Directory

파일의 변경사항



Staging Area

버전으로 기록하기 위한
파일 변경사항의 목록

Repository

커밋(버전)들이 기록되는 곳







Working Directory

파일의 변경사항

Staging Area

버전으로 기록하기 위한
파일 변경사항의 목록

Repository

커밋(버전)들이 기록되는 곳



staged



committed

\$ git commit

기본 명령어 – add(커밋 대상 기록)

\$ git add <file>

- working directory상의 변경 내용을 staging area에 추가하기 위해 사용
 - untracked 상태의 파일을 staged로 변경
 - modified 상태의 파일을 staged로 변경

```
MINGW64:/c/Users/takhe/Desktop/git-1
lecture@DESKTOP-49D9LQU MINGW64 ~/Desktop/git-1 (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    b.txt

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

lecture@DESKTOP-49D9LQU MINGW64 ~/Desktop/git-1 (master)
$ |
```

```
MINGW64:/c/Users/takhe/Desktop/git-1
lecture@DESKTOP-49D9LQU MINGW64 ~/Desktop/git-1 (master)
$ git add b.txt

lecture@DESKTOP-49D9LQU MINGW64 ~/Desktop/git-1 (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   b.txt

lecture@DESKTOP-49D9LQU MINGW64 ~/Desktop/git-1 (master)
$ |
```

기본 명령어 – 커밋(버전 기록)

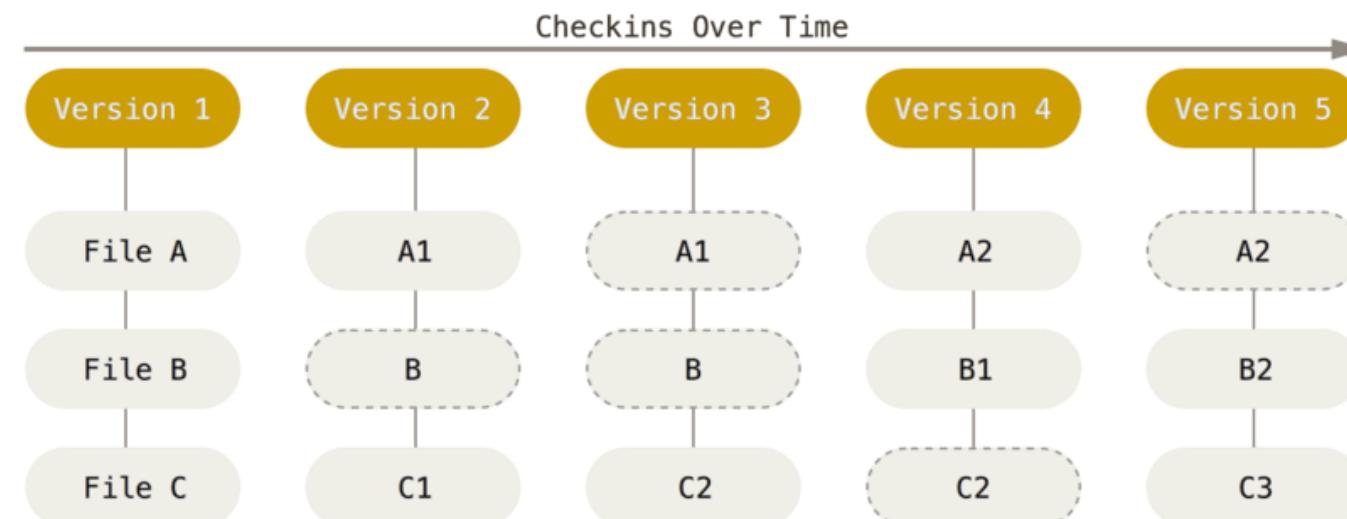
```
$ git commit -m '〈커밋메시지〉'
```

- staged 상태의 파일들을 커밋을 통해 버전으로 기록
- SHA-1 해시를 사용하여 40자 길이의 체크섬을 생성하여 고유한 커밋을 표기
- 커밋 메시지는 변경 사항을 나타낼 수 있도록 명확하게 작성해야 함

```
lecture@DESKTOP-49D9LQU MINGW64 ~/Desktop/lecture (master)
$ git commit -m 'First commit'
[master (root-commit) 14a0074] First commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 README.md
```

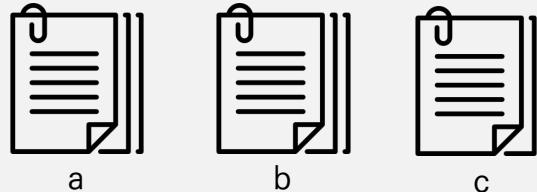
Git의 버전 관리

- Git은 데이터를 파일 시스템의 스냅샷으로 관리하고 매우 크기가 작음
- 파일이 달라지지 않으면 성능을 위해 파일을 새로 저장하지 않음
- 기존의 델타 기반 버전 관리시스템과 가장 큰 차이를 가짐



Working Directory

파일의 변경사항

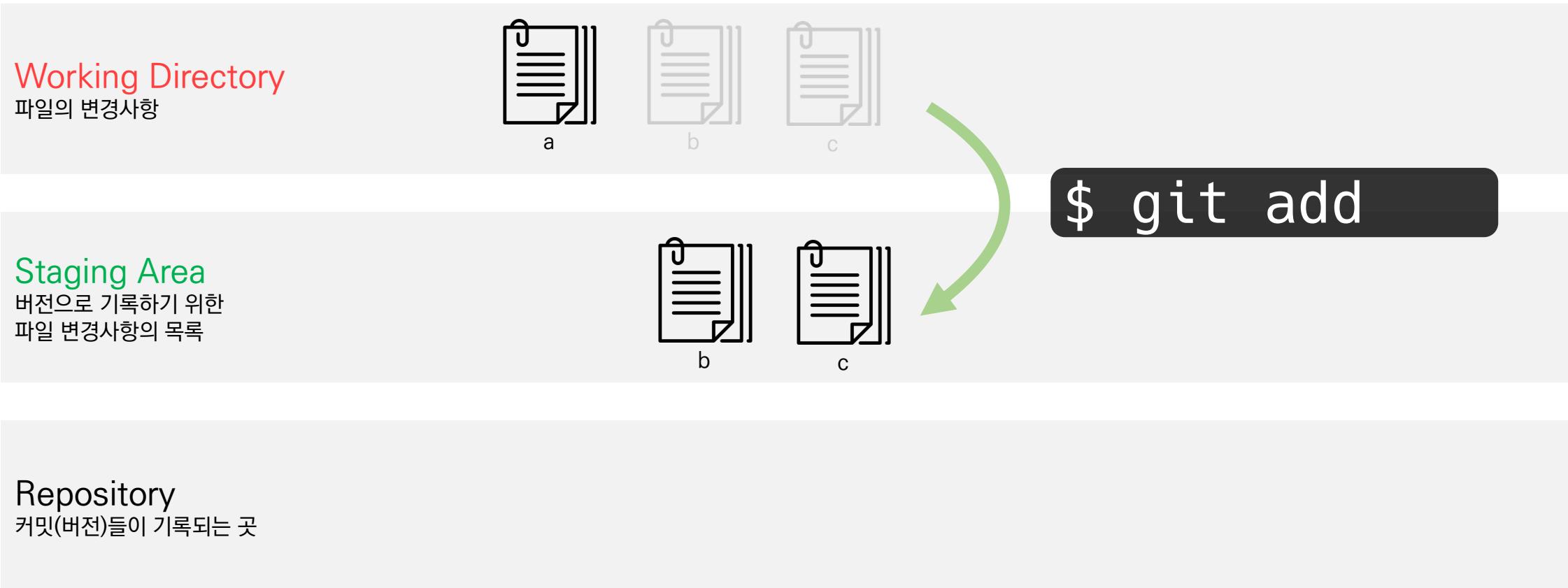


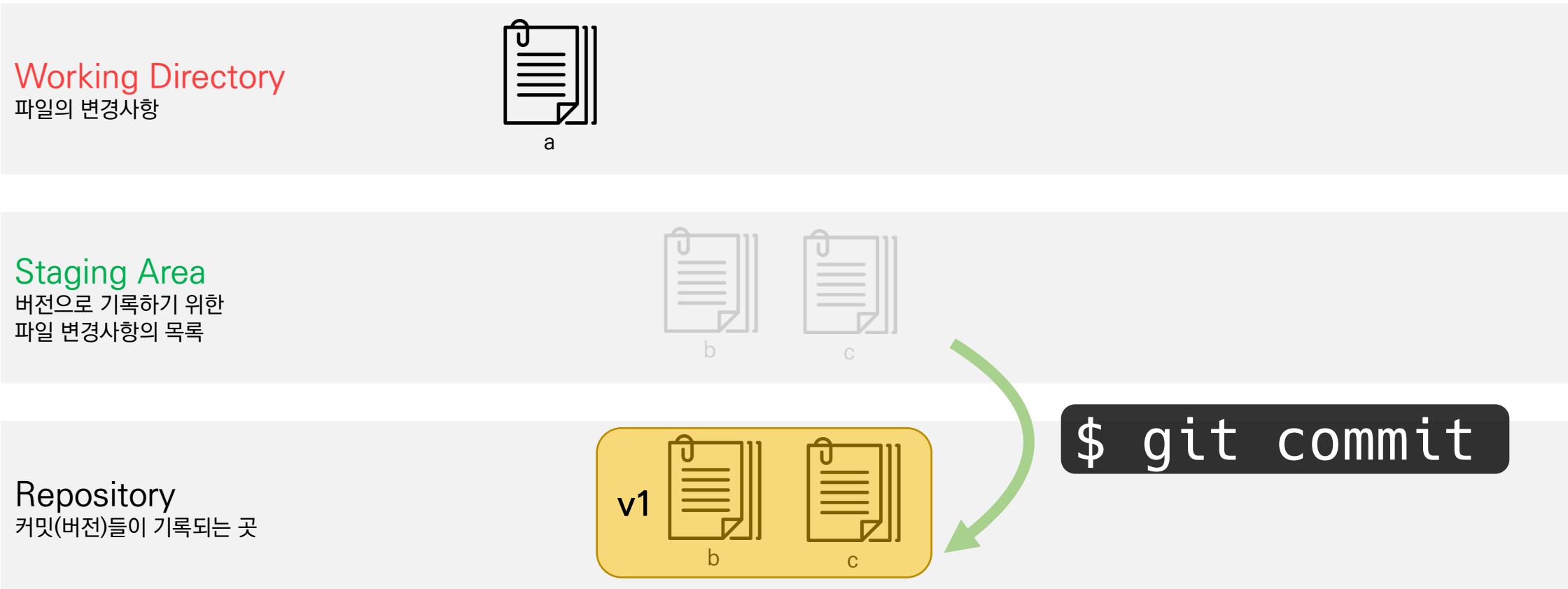
Staging Area

버전으로 기록하기 위한
파일 변경사항의 목록

Repository

커밋(버전)들이 기록되는 곳







Working Directory

파일의 변경사항

Staging Area

버전으로 기록하기 위한
파일 변경사항의 목록

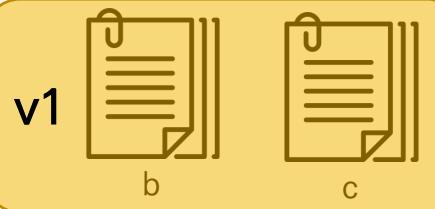
\$ git commit

Repository

커밋(버전)들이 기록되는 곳



a



현재 상태를 알고 싶어요.

Working Directory

파일의 변경사항

```
$ git status
```

Staging Area

버전으로 기록하기 위한
파일 변경사항의 목록

Repository

커밋(버전)들이 기록되는 곳

```
$ git log
```

기본 명령어 – 로그(커밋 기록 조회)

\$ git log

- 현재 저장소에 기록된 커밋을 조회
- 다양한 옵션을 통해 로그를 조회할 수 있음

\$ git log -1

\$ git log --oneline

\$ git log -2 --oneline

```
MINGW64:/c/Users/takhe/Desktop/chromium
lecture@DESKTOP-49D9LQU MINGW64 ~/Desktop/chromium (master)
$ git log -5 --oneline
0743f396477c (HEAD -> master, origin/master, origin/main) Merge from 923a131e2a89 to 349dde1f1c62 (3 revisions)
80dae1472404 Chromium Updater system level install shou...
ions
8f27a5b8199d [iOS] Notify SyncedSessionsObserver of ch...
7dc9ecca09d3 Add null check for embedding app's package...
df0d55429722 ChromeVox reads the opening of tab strip ...
lecture@DESKTOP-49D9LQU MINGW64 ~/Desktop/chromium (ma...
$
```

기본 명령어 – 상태(파일 변경 상태)

\$ git status

- Git 저장소에 있는 파일의 상태를 확인하기 위하여 활용
 - 파일의 상태를 알 수 있음
 - Untracked files
 - Changes not staged for commit
 - Changes to be committed
 - Noting to commit, working tree clean

```
MINGW64:/c/Users/takhe/Desktop/git-1
lecture@DESKTOP-49D9LQU MINGW64 ~/Desktop/git-1 (master)
$ git add b.txt

lecture@DESKTOP-49D9LQU MINGW64 ~/Desktop/git-1 (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   b.txt

lecture@DESKTOP-49D9LQU MINGW64 ~/Desktop/git-1 (master)
$ git commit -m 'Add b.txt'
[master 9f47127] Add b.txt
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 b.txt

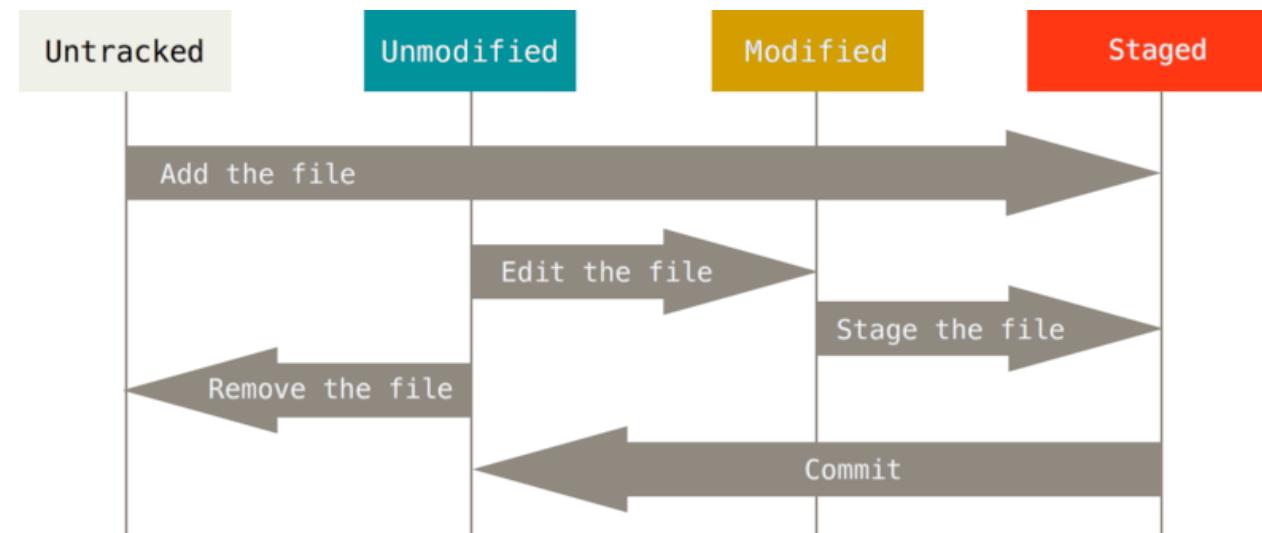
lecture@DESKTOP-49D9LQU MINGW64 ~/Desktop/git-1 (master)
$ git status
On branch master
nothing to commit, working tree clean

lecture@DESKTOP-49D9LQU MINGW64 ~/Desktop/git-1 (master)
$ |
```

파일 관리 상태 라이프사이클

- Status로 확인할 수 있는 파일의 상태
 - Tracked : 이전부터 버전으로 관리되고 있는 파일 상태
 - Unmodified : git status에 나타나지 않음 (X)
 - Modified : Changes not staged for commit
 - Staged : Changes to be committed
 - Untracked : 버전으로 관리된 적 없는 파일 상태 (파일을 새로 만든 경우)

파일 관리 상태 라이프사이클

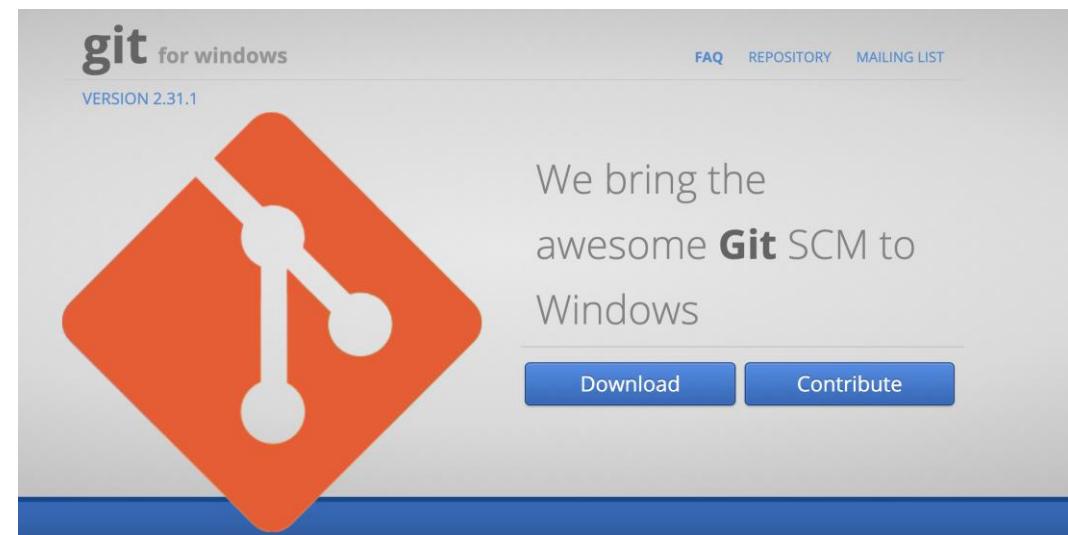


<https://git-scm.com/book/ko/v2/Git의-기초-수정하고-저장소에-저장하기>

| 명령어 | 내용 |
|-------------------------|-------------------|
| git init | 로컬 저장소 생성 |
| git add <파일명> | 특정 파일/폴더의 변경사항 추가 |
| git commit -m '<커밋메시지>' | 커밋 (버전 기록) |
| git status | 상태 확인 |
| git log | 버전 확인 |

(1) 설치하기

- Git bash 설치



Git 필수 설정 정보

- 사용자 정보 (commit author) : 커밋을 하기 위해 반드시 필요
 - git config —global user.name “*username*”
 - *Github*에서 설정한 *username*으로 설정
 - git config —global user.email “*my@email.com*”
 - *Github*에서 설정한 *email*로 설정
- 설정 확인
 - git config -l
 - git config —global -l
 - git config user.name

Git config의 이해

- —system
 - /etc/gitconfig
 - 시스템의 모든 사용자와 모든 저장소에 적용(관리자 권한)
- —global
 - ~/.gitconfig
 - 현재 사용자에게 적용되는 설정
- —local
 - .git/config
 - 특정 저장소에만 적용되는 설정

저장소 만들고 첫번째 버전 기록하기

- 1) 바탕화면에 test 폴더를 만들고 git 저장소를 만들기
- 2) a.txt 파일 넣고 커밋하기
- 3) 임의의 파일을 만들고 커밋하기
- 4) a.txt 파일 수정하고 커밋하기

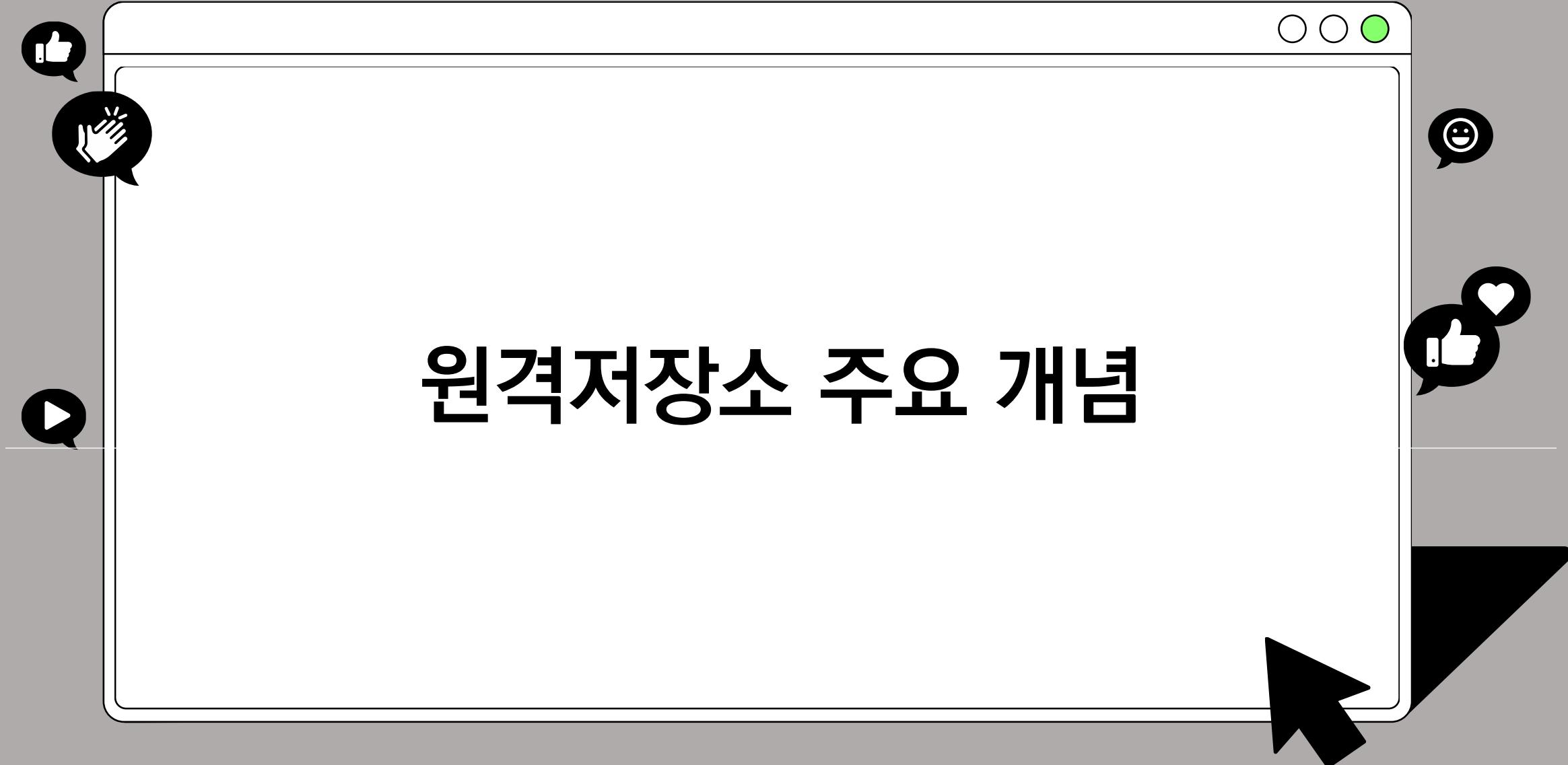
각 단계별로 status와 log를 보세요.

- 주의
 - 어떠한 파일도 생성하지 않으면 당연하게도 버전이 만들어지지 않습니다.

저장소 실습

- 1) TIL 폴더를 만들고 git 저장소를 만들기
- 2) README.md 파일을 만들고 커밋하기
- 3) 오늘 작성한 마크다운 파일을 옮기고 커밋하기
 - 마크다운 파일별로 각각 커밋을 진행해보세요!
- 4) 이제까지 배운 내용들을 정리하고 커밋하기

원격저장소 주요 개념



학습 목표

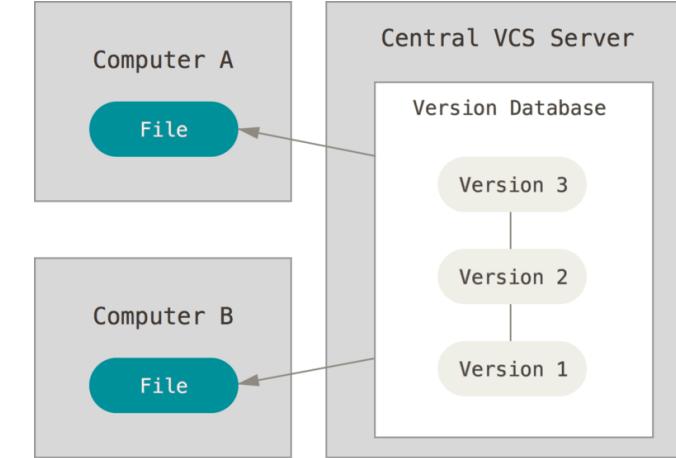
원격저장소 활용 명령어를 이해하고 설명할 수 있다.

분산버전관리시스템을 이해하고 설명할 수 있다.

분산버전관리시스템(DVCS) 복습

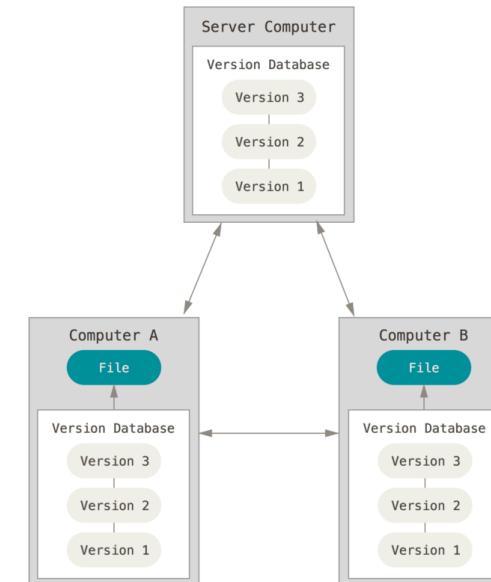
- 중앙집중식버전관리시스템

- 로컬에서는 파일을 편집하고 서버에 반영
- 중앙 서버에서만 버전을 관리



- 분산버전관리시스템

- 로컬에서도 버전을 기록하고 관리
- 원격 저장소(remote repository)를 활용하여 협업



원격저장소 (Remote Repository)

- 원격저장소를 제공하는 서비스는 다양하며, 본 수업에서는 GitHub을 활용한다.





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

Git은 █을 관리한다.

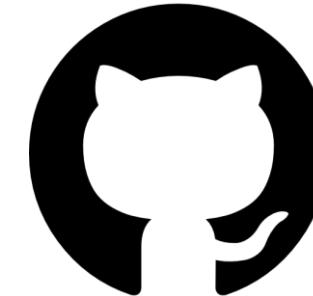
Git은 버전을 관리한다.

GitHub도 버전을 관리한다.

원격저장소 활용하기 (1)

- Push
 - 로컬 저장소의 버전을 원격저장소로 보낸다.

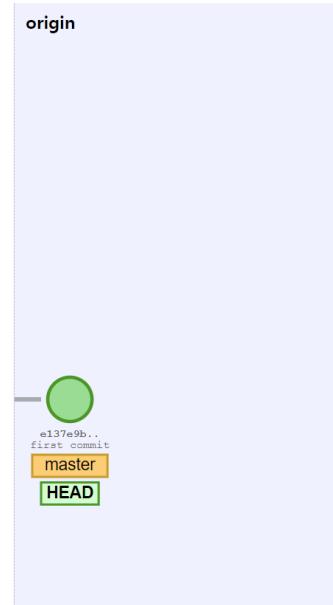
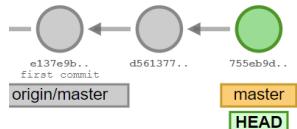
```
$ git push
```



원격저장소 활용하기 (1)

- Push
 - 로컬 저장소의 버전을 원격저장소로 보낸다.

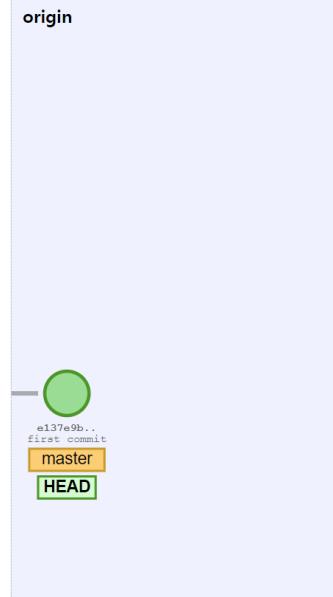
Local Repository
HEAD: master



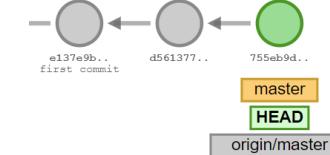
원격저장소 활용하기 (1)

- Push
 - 로컬 저장소의 버전을 원격저장소로 보낸다.

Local Repository
HEAD: master

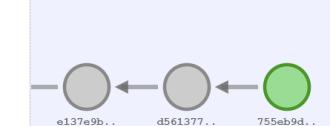


Local Repository
HEAD: master



\$ git push

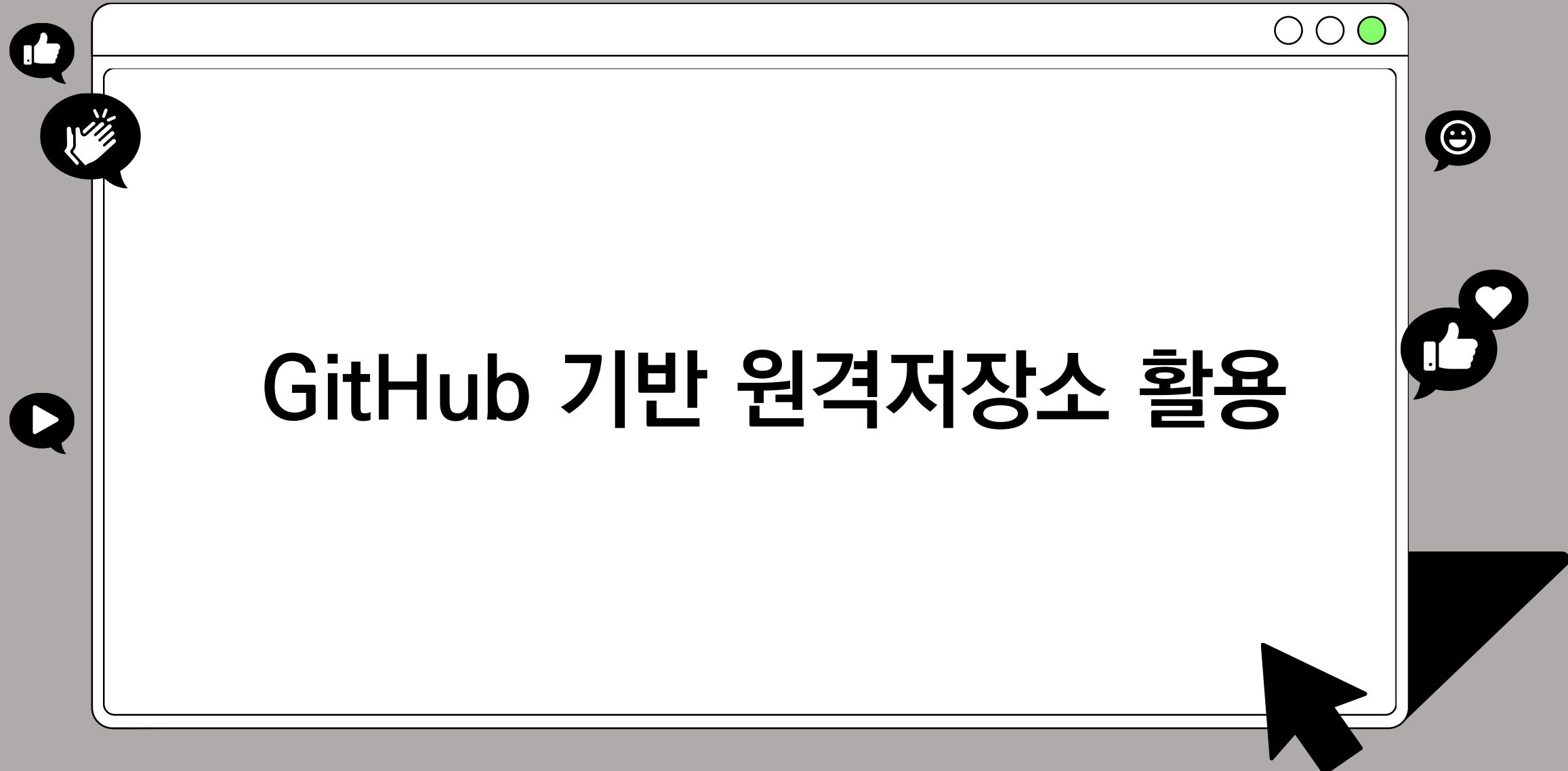
origin



원격저장소 활용하기 (2)

- Pull
 - 원격저장소의 버전을 로컬 저장소로 가져온다.





학습 목표

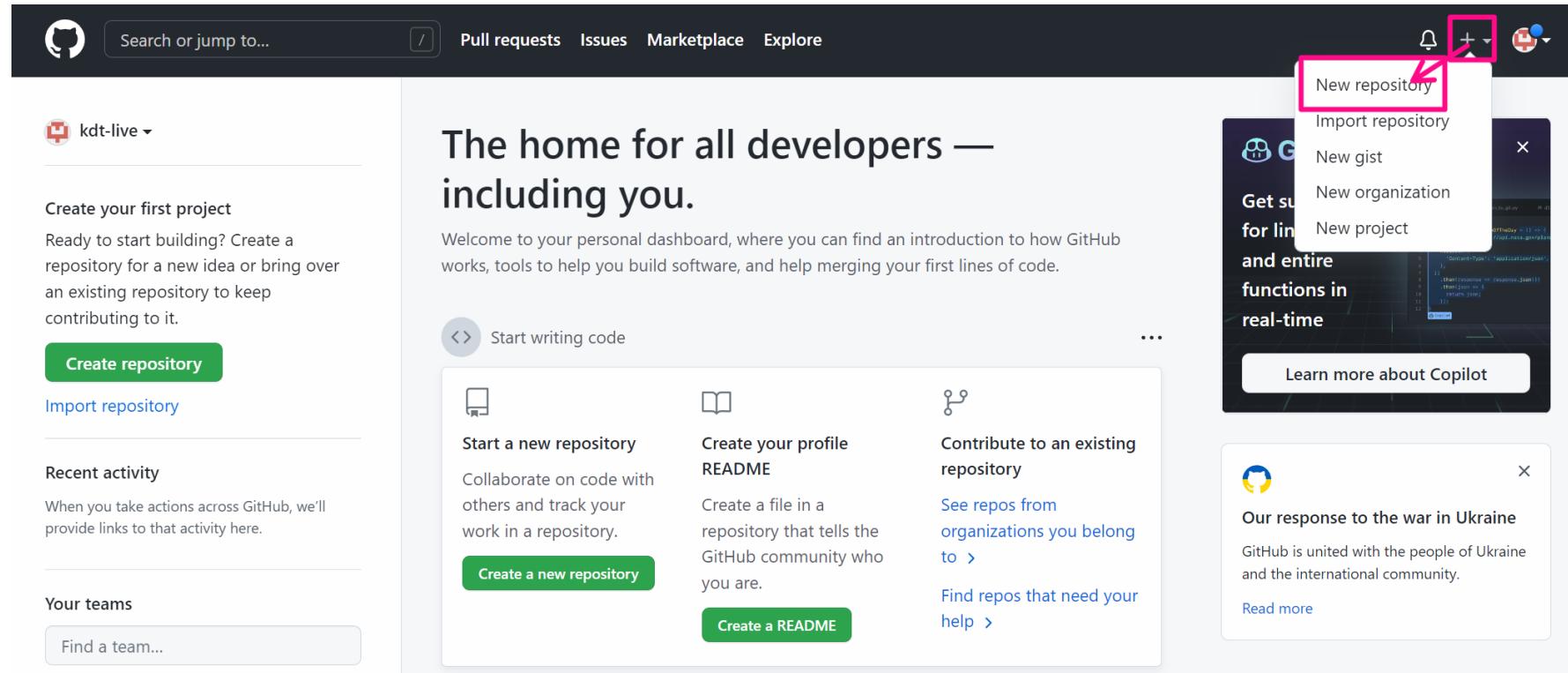
GitHub 원격저장소에 로컬 저장소를 올려 관리할 수 있다.

원격저장소 활용 명령어를 이해하고 설명할 수 있다.

초기 원격저장소 설정하기

초기 원격저장소 설정하는 법

- (1) New Repository



초기 원격저장소 설정하는 법

- (2) 저장소 설정하기

The screenshot shows the GitHub 'Create a new repository' interface. Key elements highlighted with pink boxes and numbered steps are:

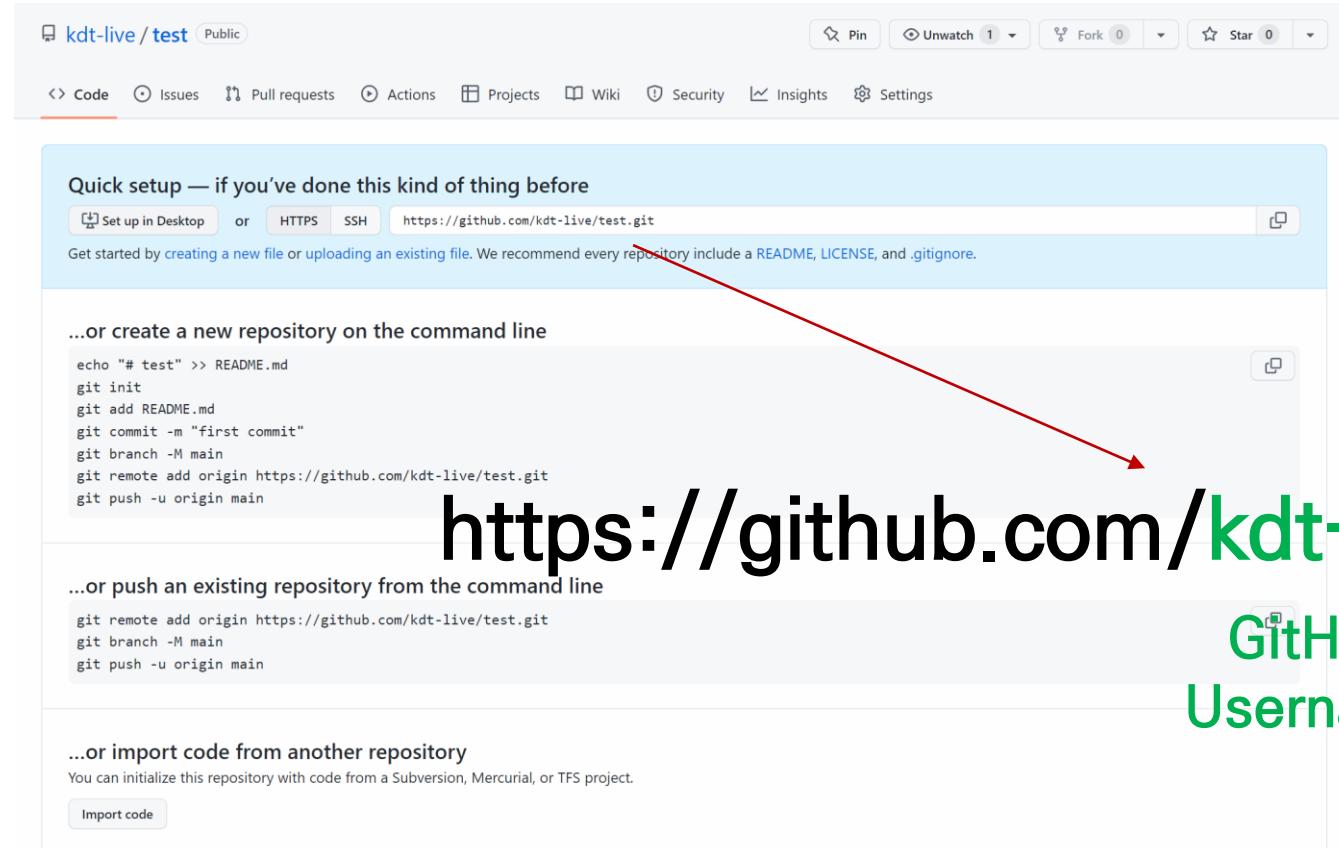
- repository 이름 설정 (Repository name): 'test'
- 저장소 설명(옵션) (Description): An empty text input field.
- 공개 여부 설정 (Visibility): 'Public' (radio button selected).
- 저장소 생성 (Create repository): A green button at the bottom.

Annotations in red text on the right side of the form area:

- Git/GitHub을 이해하기 전에는 설정을 하지 않습니다.
- 체크 없이 모두 None!

초기 원격저장소 설정하는 법

- (3) URL 확인



https://github.com/kdt-live/test.git

**GitHub
Username**

**저장소
이름**

초기 원격저장소 설정하는 법

- (4) 로컬 저장소에 원격 저장소 정보 설정하기

The screenshot shows a GitHub repository setup page for a repository named 'kdt-live / test'. The 'Code' tab is selected. A 'Quick setup' section provides options to 'Set up in Desktop' or 'HTTPS / SSH' (with the URL <https://github.com/kdt-live/test.git>). It also suggests creating a new file or uploading an existing file, and recommends including a README, LICENSE, and .gitignore. Below this, a section for command-line setup shows a sequence of Git commands:

```
echo "# test" >> README.md  
git init  
git add README.md  
git commit -m "first commit"  
git branch -m main  
git remote add origin https://github.com/kdt-live/test.git  
git push -u origin main
```

A red box highlights the line `git remote add origin https://github.com/kdt-live/test.git`, and a red arrow points from this line to a terminal window at the bottom of the page.

...or push an existing repository from the command line

```
$ git remote add origin https://github.com/kdt-live/test.git
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

[Import code](#)

초기 원격저장소 설정하는 법

- (4) 로컬 저장소에 원격 저장소 정보 설정하기
 - 로컬 저장소에는 한번만 설정하면 된다.

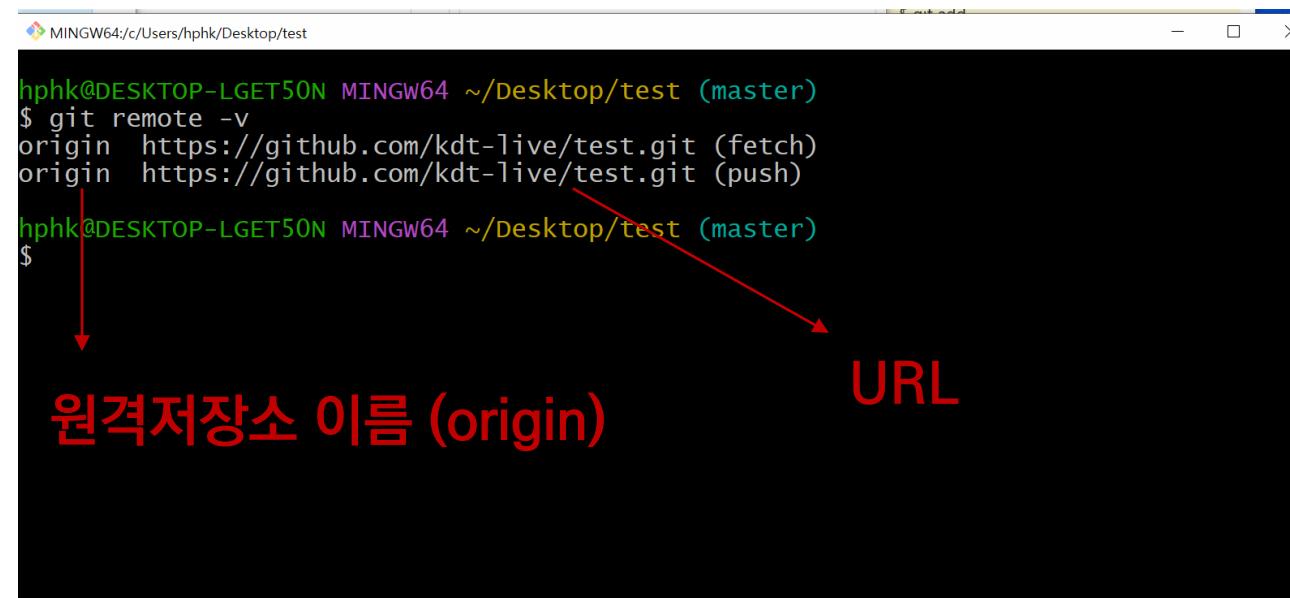
```
$ git remote add origin https://github.com/kdt-live/test.git
```

원격저장소 추가해 Origin
으로

GitHub Username 저장소 이름

초기 원격저장소 설정하는 법

- (5) 원격 저장소의 정보를 확인함



```
MINGW64:/c/Users/hphk/Desktop/test
hphk@DESKTOP-LGET50N MINGW64 ~/Desktop/test (master)
$ git remote -v
origin  https://github.com/kdt-live/test.git (fetch)
origin  https://github.com/kdt-live/test.git (push)

hphk@DESKTOP-LGET50N MINGW64 ~/Desktop/test (master)
$
```

원격저장소 이름 (origin)

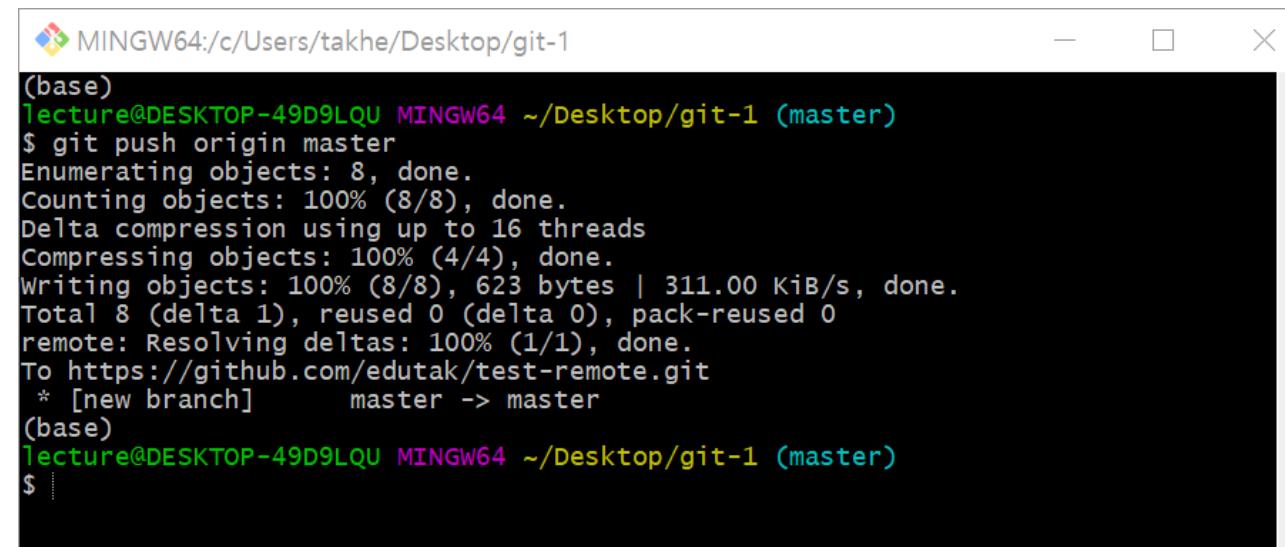
URL

로컬 저장소의 버전을 원격 저장소로 Push하기

push

\$ git push <원격저장소이름> <브랜치이름>

- 원격 저장소로 로컬 저장소 변경 사항(커밋)을 올림(push)
 - 원격 저장소는 로컬 폴더의 파일/폴더가 아닌 저장소의 버전(커밋)을 관리하는 것

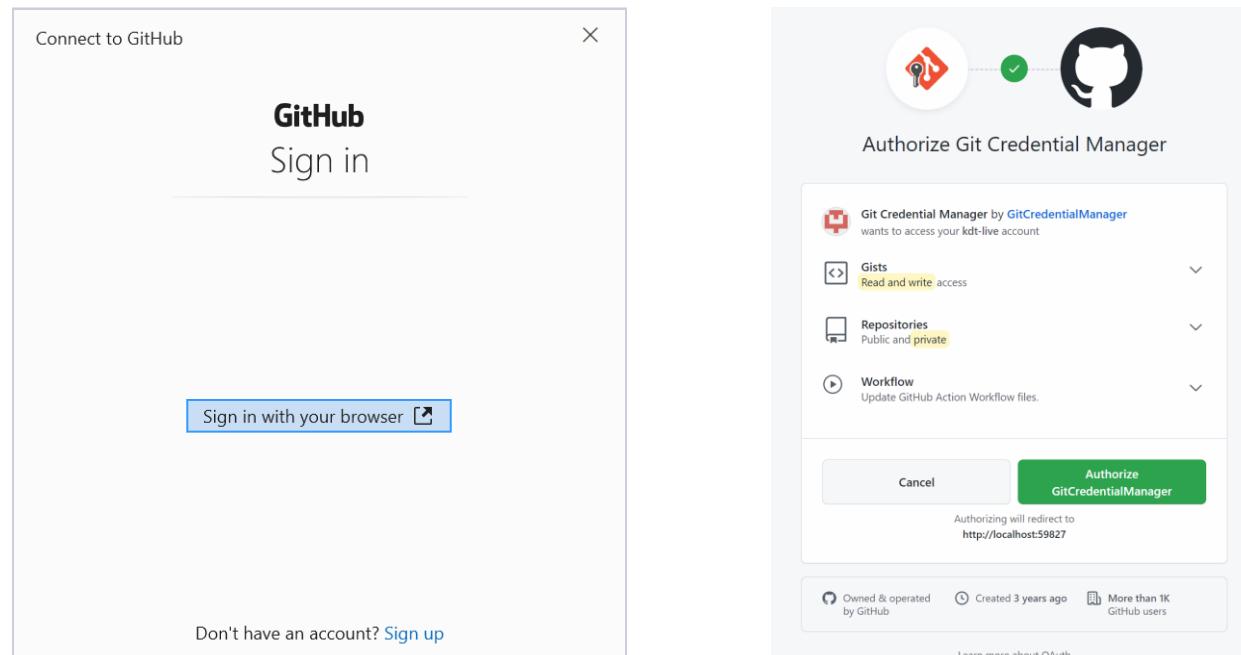


The screenshot shows a terminal window titled 'MINGW64:/c/Users/takhe/Desktop/git-1'. The command \$ git push origin master is being run. The output shows the progress of the push operation, including object enumeration, counting, compressing, writing, and resolving deltas. It also indicates that a new branch 'master' was created on the remote repository.

```
(base)
lecture@DESKTOP-49D9LQU MINGW64 ~/Desktop/git-1 (master)
$ git push origin master
Enumerating objects: 8, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 16 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (8/8), 623 bytes | 311.00 KiB/s, done.
Total 8 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/edutak/test-remote.git
 * [new branch]      master -> master
(base)
lecture@DESKTOP-49D9LQU MINGW64 ~/Desktop/git-1 (master)
$
```

Push 주의사항

- Push할 때는 인증 정보가 필수적입니다.
- 윈도우는 아래의 화면에서 인증을 하여야 합니다. (크롬 브라우저 활용)



Push 주의사항

- Push할 때는 인증 정보가 필수적입니다.
- 맥은 토큰을 발급 받아 비밀번호로 활용합니다.

The screenshot shows the GitHub developer settings page under 'Personal access tokens'. It displays a table with one token listed:

| Token | Action |
|--|--------|
| ghp_Ijlonyy0eTGkxE1MBWaIJVmmlqhXe04Id3dL | Delete |

A note at the bottom states: "Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication."

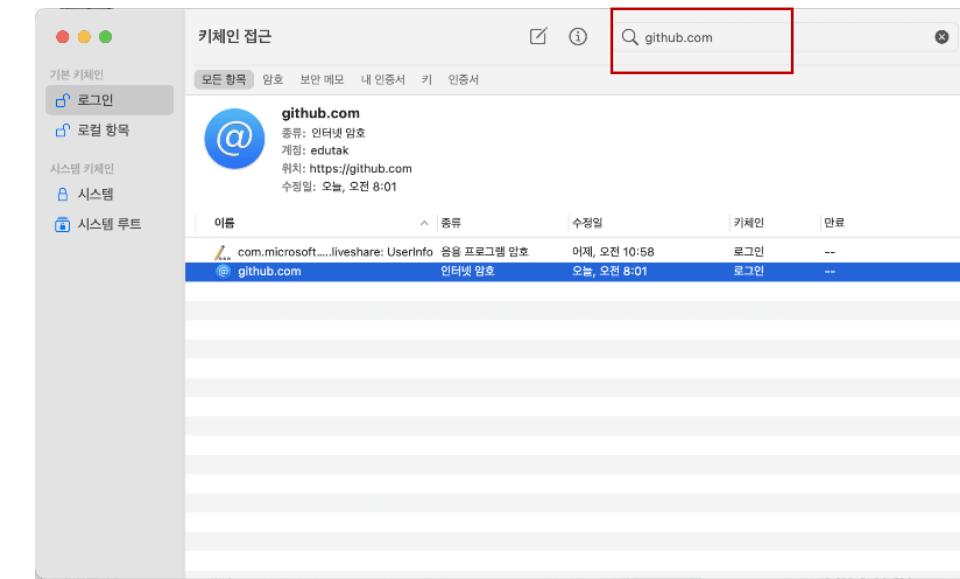
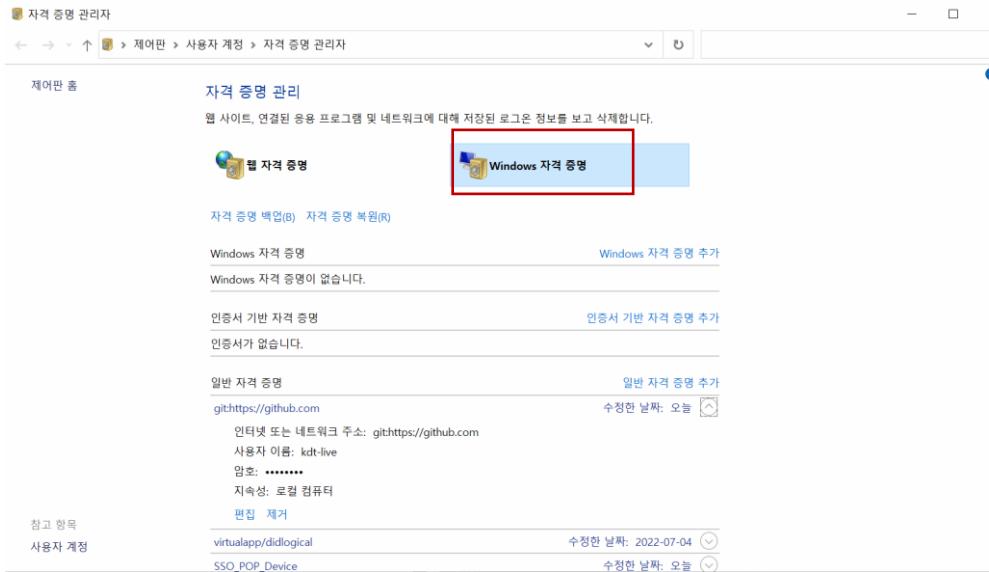
The screenshot shows a terminal window titled 'github-example -- zsh --'. It displays the following command:

```
tak@takui-MacBookPro github-example % git push origin master
Username for 'https://github.com': edutak
Password for 'https://edutak@github.com':
```

GitHub 로그인 비밀번호가 아닌 Token값!

Push 주의사항

- Push가 Authentication failed되는 경우 인증 정보를 확인 부탁드립니다.
 - 윈도우 : 자격증명관리자
 - 맥 : 키체인 접근

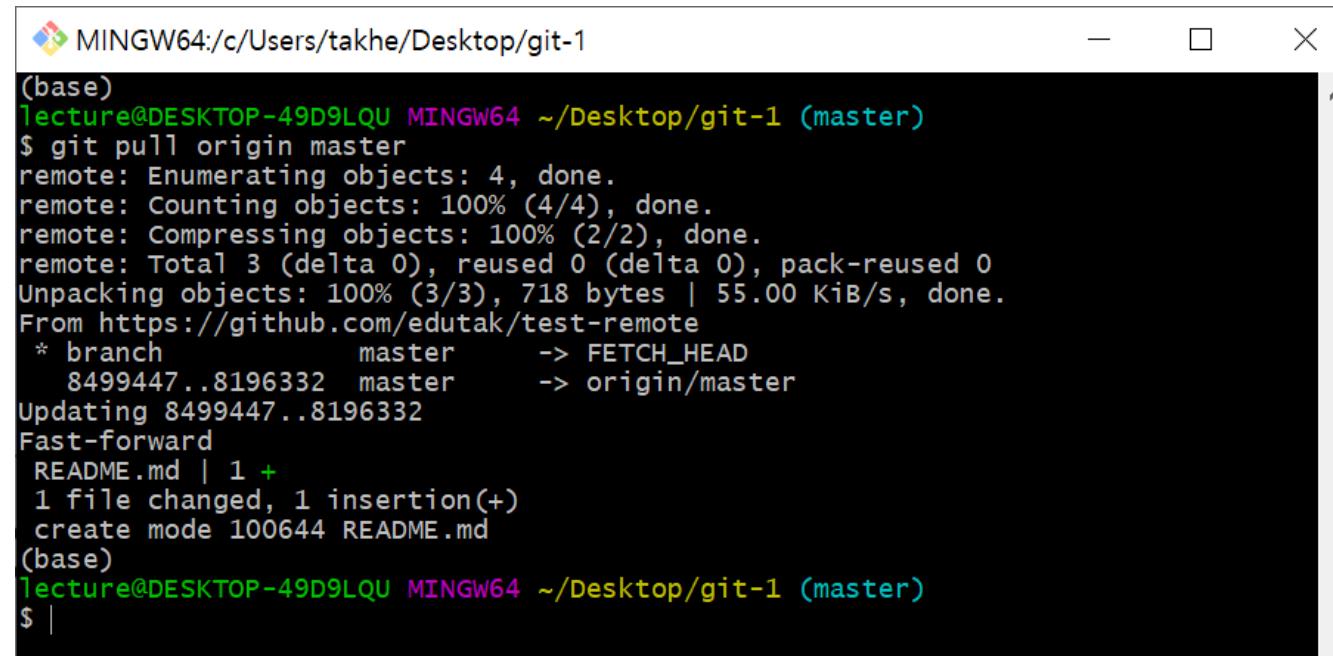


원격 저장소의 버전을 로컬 저장소로 Pull하기

pull

\$ git pull <원격저장소이름> <브랜치이름>

- 원격 저장소로부터 변경된 내역을 받아와서 이력을 병합함



The screenshot shows a terminal window titled 'MINGW64:/c/Users/takhe/Desktop/git-1'. The command \$ git pull origin master is run, followed by its output:

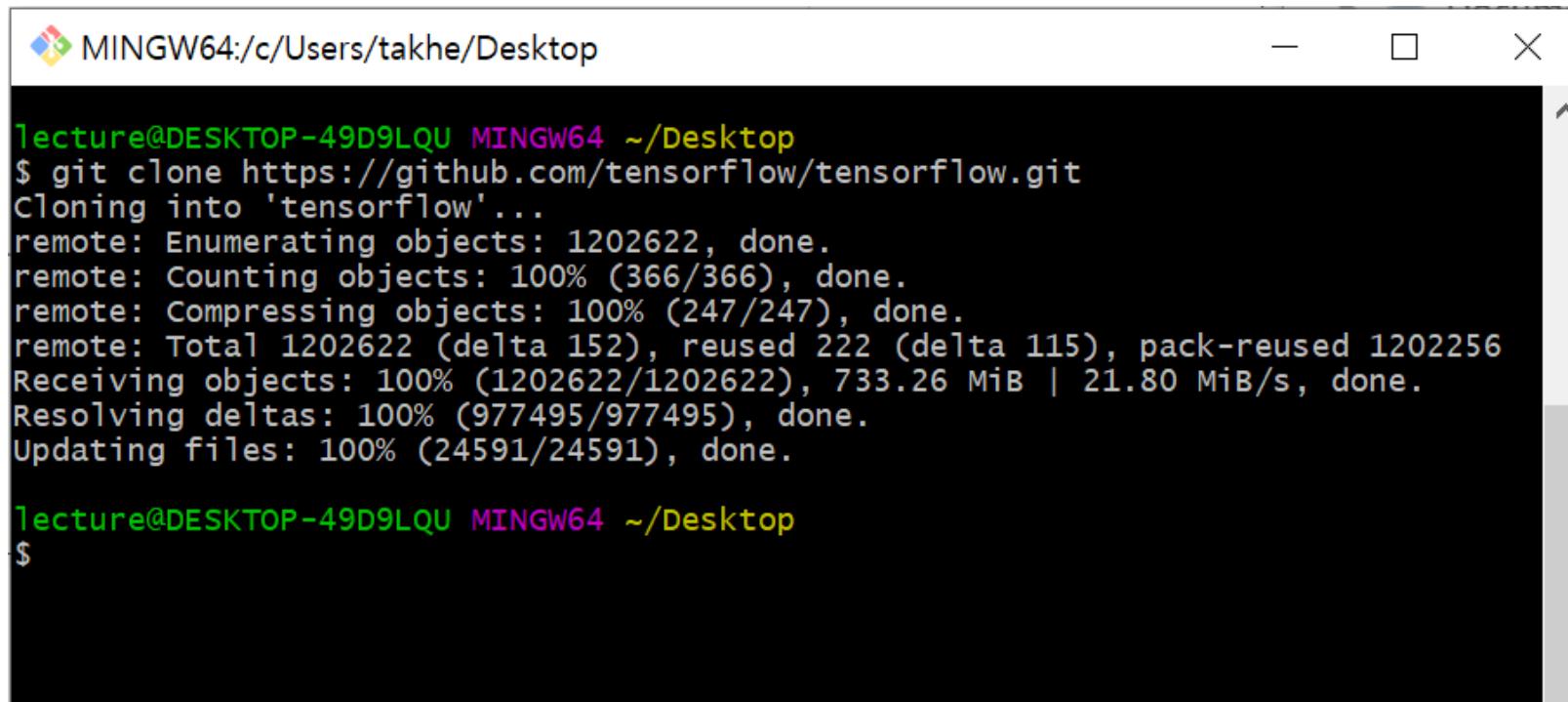
```
(base)
Tecture@DESKTOP-49D9LQU MINGW64 ~/Desktop/git-1 (master)
$ git pull origin master
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 718 bytes | 55.00 KiB/s, done.
From https://github.com/edutak/test-remote
 * branch            master      -> FETCH_HEAD
   8499447..8196332  master      -> origin/master
Updating 8499447..8196332
Fast-forward
 README.md | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 README.md
(base)
Tecture@DESKTOP-49D9LQU MINGW64 ~/Desktop/git-1 (master)
$ |
```

**원격저장소 프로젝트를
시작하고 싶어요**

clone

\$ git clone <원격저장소주소>

- 원격 저장소를 복제하여 가져옴



```
MINGW64:/c/Users/takhe/Desktop
lecture@DESKTOP-49D9LQU MINGW64 ~/Desktop
$ git clone https://github.com/tensorflow/tensorflow.git
Cloning into 'tensorflow'...
remote: Enumerating objects: 1202622, done.
remote: Counting objects: 100% (366/366), done.
remote: Compressing objects: 100% (247/247), done.
remote: Total 1202622 (delta 152), reused 222 (delta 115), pack-reused 1202256
Receiving objects: 100% (1202622/1202622), 733.26 MiB | 21.80 MiB/s, done.
Resolving deltas: 100% (977495/977495), done.
Updating files: 100% (24591/24591), done.

lecture@DESKTOP-49D9LQU MINGW64 ~/Desktop
$
```

Clone과 Pull의 차이점은?

Clone : 원격저장소 복제

Pull : 원격저장소 커밋 가져오기

로컬에서 새로운 프로젝트의 시작 ?

로컬에서 새로운 프로젝트의 시작 ?

git init

원격에 있는 프로젝트 시작?

원격에 있는 프로젝트 시작?

git clone

협업 프로젝트, 외부 오픈소스 참여
Git 저장소를 GitHub에서 생성 후 시작 등..

프로젝트 개발 중 다른 사람 커밋 받아오기?

프로젝트 개발 중 다른 사람 커밋
받아오기?

git pull

내가 한 로컬 프로젝트 개발 공유?

내가 한 로컬 프로젝트 개발 공유?

git push

명령어 정리

| 명령어 | 내용 |
|------------------------------|----------------------------|
| git clone <url> | 원격 저장소 복제 |
| git remote -v | 원격저장소 정보 확인 |
| git remote add <원격저장소> <url> | 원격저장소 추가
(일반적으로 origin) |
| git remote rm <원격저장소> | 원격저장소 삭제 |
| git push <원격저장소> <브랜치> | 원격저장소에 push |
| git pull <원격저장소> <브랜치> | 원격저장소로부터 pull |

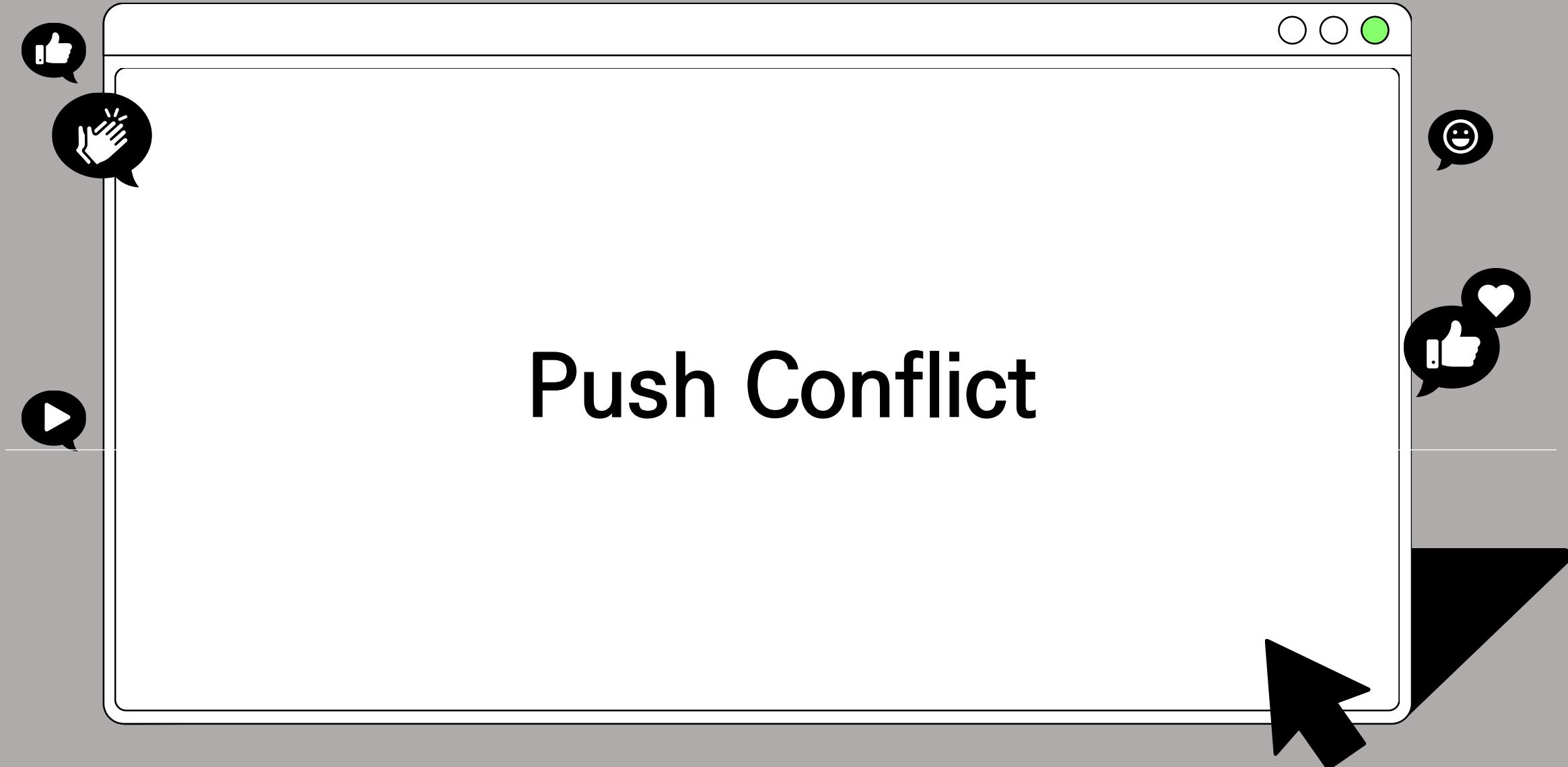
자기소개 프로젝트

- 바탕화면에 프로필 폴더를 만들고 로컬저장소로 지정합니다.

원격저장소 활용 실습

- 로컬저장소로 만들어진 TIL폴더를 원격저장소에 올리고 링크를 제출합니다.
 - 원격저장소 이름은 TIL로 해주세요.

Push Conflict



- 협업을 하다보면 아래의 메시지를 확인하게 된다.

```
student@m503INS MINGW64 ~/Desktop/edutak (master)
$ git push origin master
To https://github.com/edutak/edutak.git
  ! [rejected]          master -> master (fetch first)
error: failed to push some refs to 'https://github.com/edutak/edutak.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

- 로컬과 원격 저장소의 커밋 이력이 다른 경우 발생한 것이다.

A screenshot of a GitHub repository page for 'edutak / edutak'. The 'Code' tab is selected. The 'master' branch is chosen. Below it, a commit history is shown for 'Commits on Jun 4, 2021'. Two commits are listed:

- 'Update README.md' by edutak committed 7 minutes ago. It has a green 'Verified' badge, a copy icon, a link to e884994, and a diff icon.
- 'Add README' by edutak committed 37 minutes ago. It has a copy icon, a link to dcf3558, and a diff icon.

At the bottom, there are 'Newer' and 'Older' buttons.

```
student@M503INS MINGW64 ~/Desktop/edutak (master)
$ git log --oneline
3946b62 (HEAD -> master) Update 경 력
dcf3558 (origin/master) Add README
```

- 1. 원격저장소의 커밋을 원격저장소로 가져와서(pull)
- 2. 로컬에서 두 커밋을 병합 (추가 커밋 발생)
 - 동시에 같은 파일이 수정된 경우 merge conflict가 발생하나 이 부분은 브랜치 학습
- 3. 다시 GitHub으로 push

o- Commits on Jun 4, 2021

| | |
|---|----------|
| Merge branch 'master' of https://github.com/edutak/edutak | b7b72e1 |
| edutak committed 2 minutes ago | <> |
| Update 경력 | 3946b62 |
| edutak committed 10 minutes ago | <> |
| Update README.md | e884994 |
| edutak committed 12 minutes ago | Verified |
| Add README | dcf3558 |
| edutak committed 42 minutes ago | <> |

gitignore



버전관리랑 상관 없는 파일?



secret.csv

어떻게 관리해야 할까?

gitignore

- 일반적인 개발 프로젝트에서 버전 관리를 별도로 하지 않는 파일/디렉토리가 발생한다.
- Git 저장소에 .gitignore 파일을 생성하고 해당 내용을 관리한다.
- 작성 예시
 - 특정 파일 : a.txt (모든 a.txt), test/a.txt (테스트 폴더의 a.txt)
 - 특정 디렉토리 : /my_secret
 - 특정 확장자 : *.exe
 - 예외 처리 : !b.exe
- 주의! 이미 커밋된 파일은 반드시 삭제를 하여야 .gitignore로 적용됩니다.
 - 따라서, 프로젝트 시작전에 미리 설정하시기 바랍니다 😊

일반적으로
어떤 파일들이 있을까?

.gitignore

- 개발 언어 (<https://github.com/github/gitignore>)
 - 예시) 파이썬 : venv/ , 자바스크립트 : node_modules/
- 개발 환경
 - 운영체제 (windows, mac, linux)
 - 텍스트 에디터 / IDE (visual studio code 등)

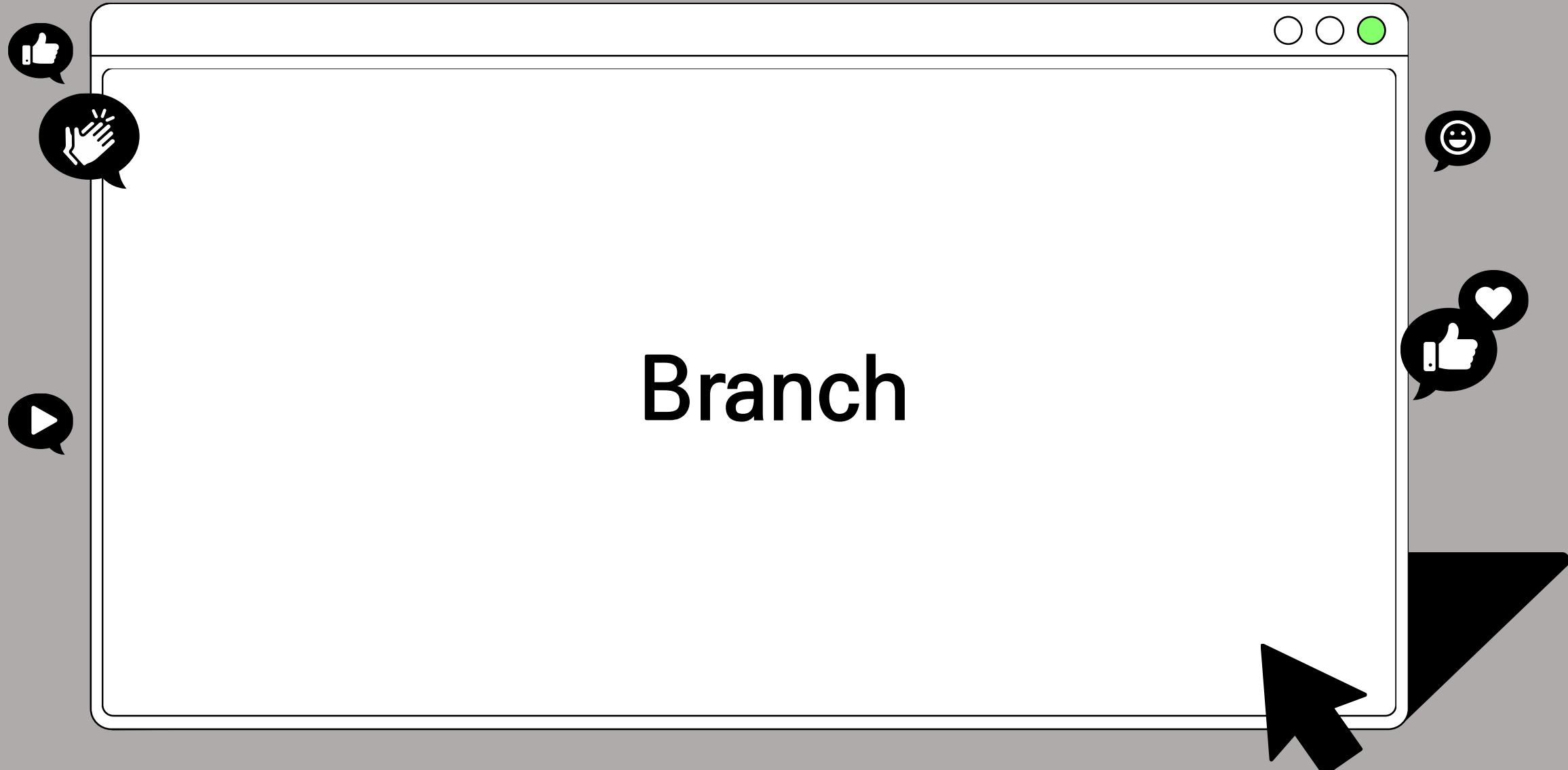
↳ gitignore.io

자신의 프로젝트에 꼭 맞는 .gitignore 파일을 만드세요



<https://gitignore.io>

Branch

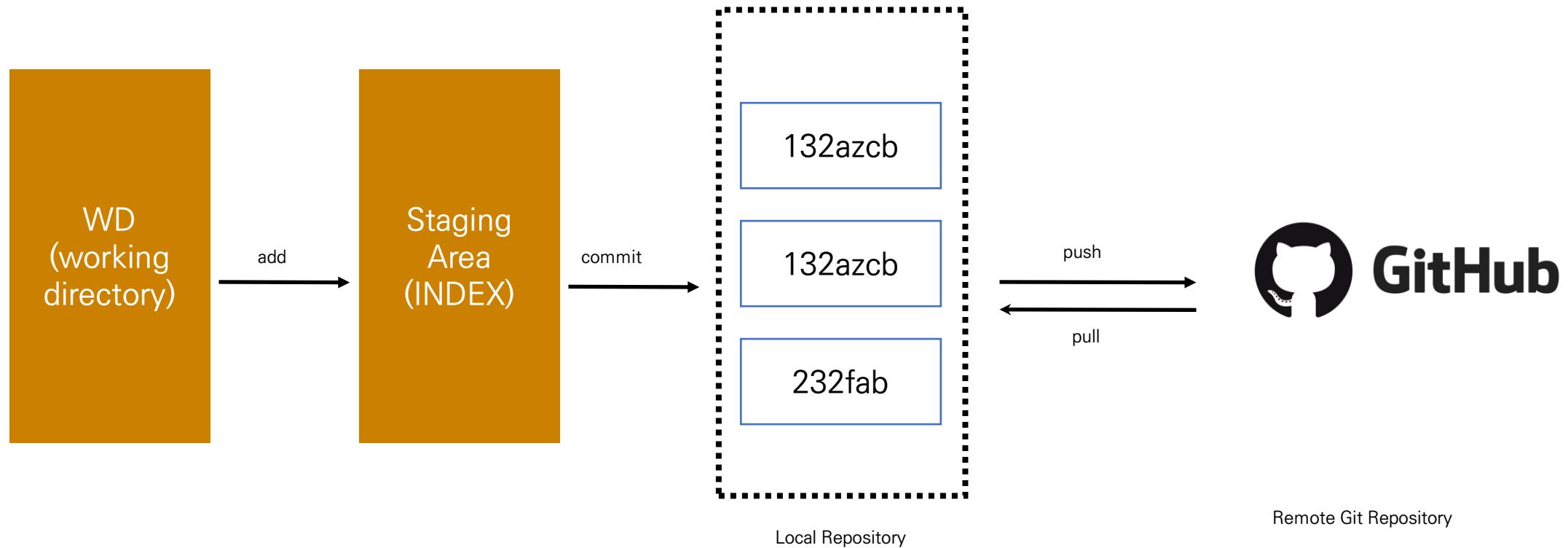


학습 목표

Branch의 목적을 알고 활용할 수 있다.

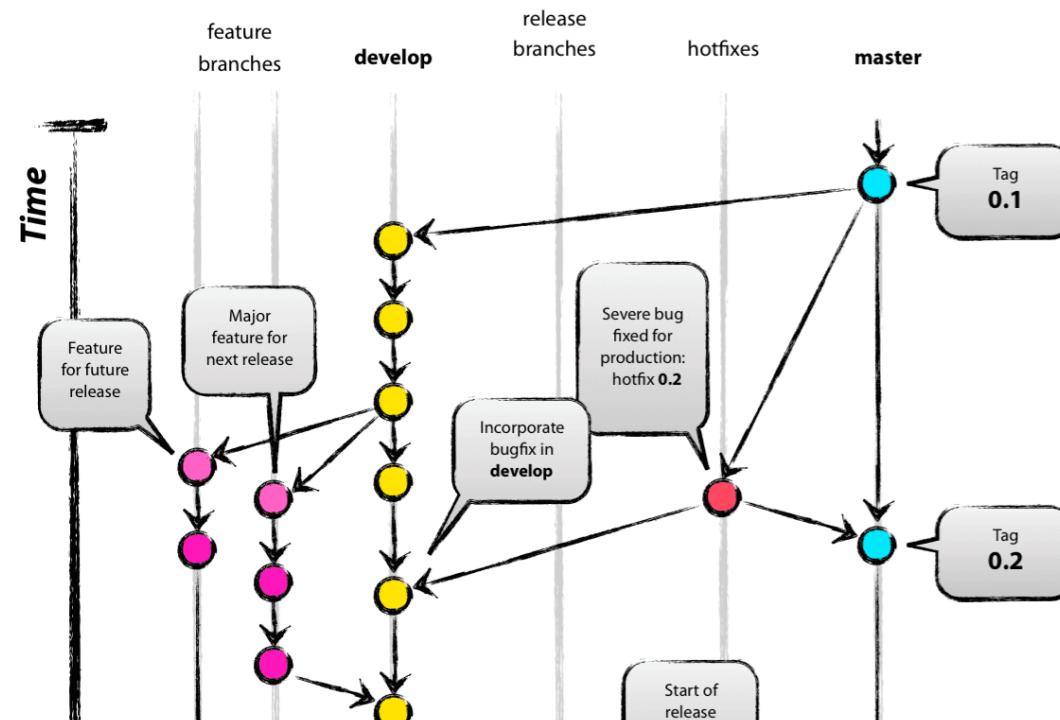
Branch 병합 과정의 상황을 이해하고 충돌을 해결할 수 있다.

Git (이미 알고 있어야하는 개념 정리)



branch

- 독립적인 작업흐름을 만들고 관리



브랜치 주요 명령어

1. 브랜치 생성

```
(master) $ git branch {branch name}
```

2. 브랜치 이동

```
(master) $ git checkout {branch name}
```

3. 브랜치 생성 및 이동

```
(master) $ git checkout -b {branch name}
```

4. 브랜치 목록

```
(master) $ git branch
```

5. 브랜치 삭제

```
(master) $ git branch -d {branch name}
```

merge

- 각 branch에서 작업을 한 이후 이력을 합치기 위해 merge 명령어를 사용
- 병합을 진행할 때, 서로 다른 이력(commit)에서
 - 동일한 파일을 수정한 경우 충돌이 발생
 - 이 경우에는 반드시 직접 해당 파일을 확인하고 적절하게 수정
 - 수정한 이후에 직접 커밋 실행
 - 다른 파일을 수정한 경우
 - 충돌 없이 자동으로 Merge Commit이 생성됨

(1) merge – fast forward

- 기존 master 브랜치에 변경사항이 없어 단순히 앞으로 이동
 1. feature-a branch로 이동 후 commit
 2. master 별도 변경 없음
 3. master branch로 병합

```
(master) $ git merge feature-a  
Updating 54b9314..5429f25  
Fast-forward
```

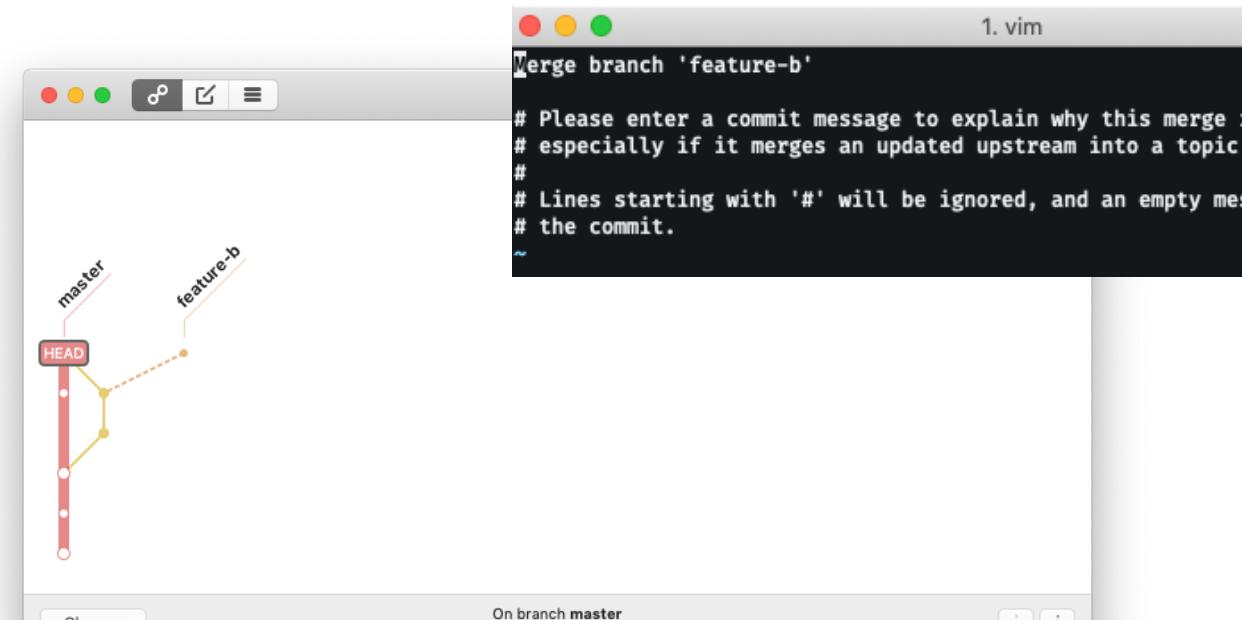


(2) merge – merge commit

- 기존 master 브랜치에 변경사항이 있어 병합 커밋 발생
 1. feature-a branch로 이동 후 commit
 2. master branch commit
 3. master branch로 병합



```
(master) $ git merge feature-a  
Already up to date!  
Merge made by the 'ort' strategy.
```

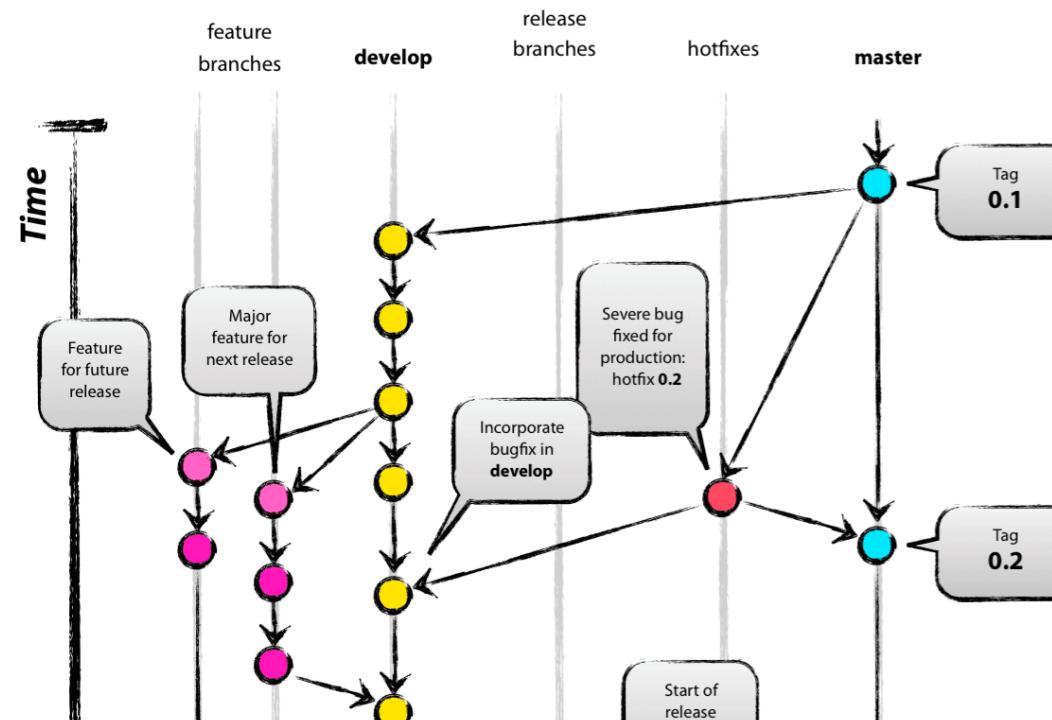


Git Flow



Git Flow

- Git을 활용하여 협업하는 흐름으로 branch를 활용하는 전략을 의미한다.
- 가장 대표적으로 활용되는 전략은 다음과 같다.



Git Flow

| branch | 주요 특징 | 예시 |
|----------------------------------|---|---------------------------------|
| master
(main) | * 배포 가능한 상태의 코드 | LOL 클라이언트 라이브 버전(9.23.298.3143) |
| develop
(main) | * feature branch로 나뉘어지거나, 발생된 버그 수정 등 개발 진행
* 개발 이후 release branch로 분기 | 다음 패치를 위한 개발 (9.24) |
| feature branches
(supporting) | * 기능별 개발 브랜치(topic branch)
* 기능이 반영되거나 드랍되는 경우 브랜치 삭제 | 개발시 기능별
예) 챔피언, 몬스터 등 |
| release branches
(supporting) | * 개발 완료 이후 QA/Test 등을 통해 얻어진 다음 배포 전
minor bug fix 등 반영 | 9.24a, 9.24b, ... |
| hotfixes
(supporting) | * 긴급하게 반영 해야하는 bug fix
* release branch는 다음 버전을 위한 것이라면,
hotfix branch는 현재 버전을 위한 것 | 긴급 패치를 위한 작업 |

Git Flow는 정해진 답이 있는 것은 아니다.

Github Flow, Gitlab Flow 등의 각 서비스별 제안되는 흐름이 있으며,
변형되어 각자의 프로젝트/회사에서 활용 되고 있다.

간단하게 브랜치를 활용하는 명령어를 알아보고,
프로젝트에 활용할 수 있는 간단한 버전의 브랜치 전략을 배워보자.

GitHub Flow 기본 원칙

- Github Flow는 Github에서 제안하는 브랜치 전략으로 다음과 같은 기본 원칙을 가지고 있다.
- 1. **master branch는 반드시 배포 가능한 상태여야 한다.**

There's only one rule: anything in the master branch is always deployable.

- 2. **feature branch는 각 기능의 의도를 알 수 있도록 작성한다.**

Your branch name should be descriptive, so that others can see what is being worked on.

- 3. **Commit message는 매우 중요하며, 명확하게 작성한다.**

Commit messages are important. By writing clear commit messages, you can make it easier for other people to follow along and provide feedback.

- 4. **Pull Request를 통해 협업을 진행한다.**

Pull Requests are useful for contributing to open source projects and for managing changes to shared repositories.

- 5. **변경사항을 반영하고 싶다면, master branch에 병합한다.**

Now that your changes have been verified in production, it is time to merge your code into the master branch.

GitHub Flow Models

- 앞서 설명된 기본 원칙 아래 Github에서 제시하는 방법이 2가지가 있다.
 - Shared Repository Model
 - Fork & Pull Model
- 가장 큰 차이점은 내(작업자)가 원격 저장소에 직접적인 push 권한이 있는지 여부

Shared Repository Model

- Shared Repository Model은 동일한 저장소를 공유하여 활용하는 방식
- 예시는 작업 흐름을 master + feature 브랜치로 구성하여 진행합니다.
-  : repository owner (project manager)
-  : collaborator

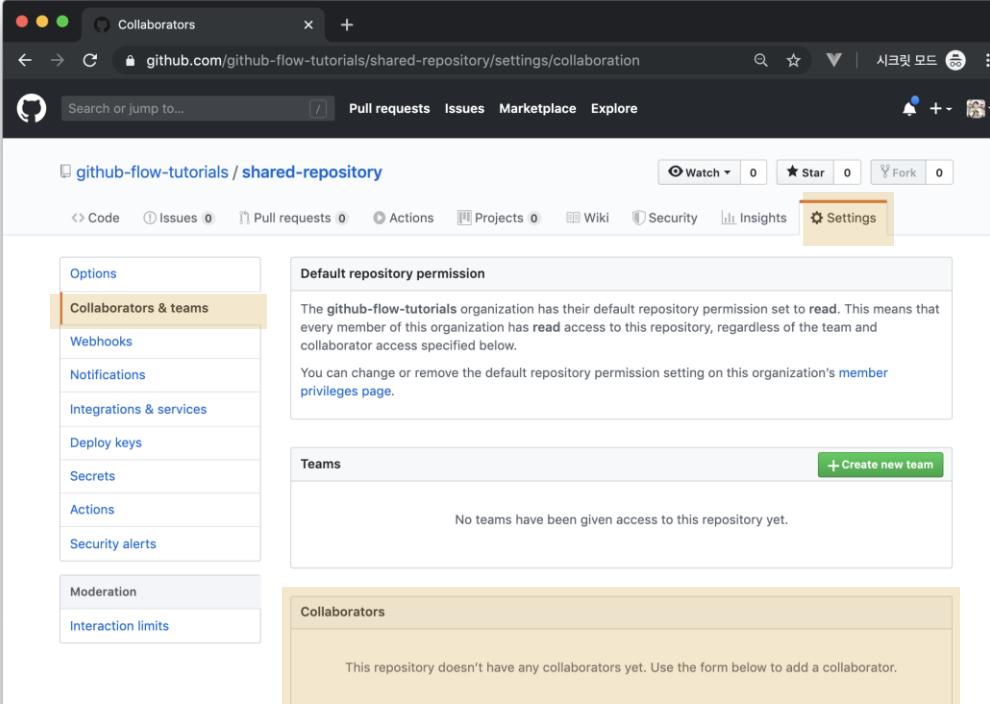
Shared Repository Model

1. 팀원 초대 및 저장소 Clone

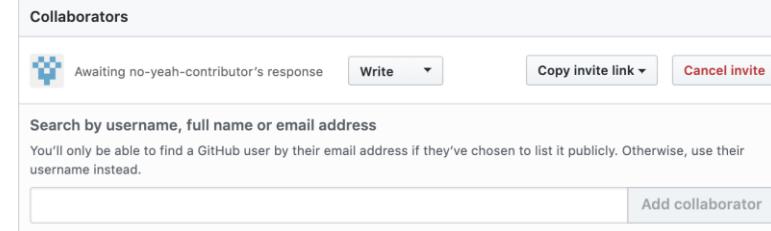
step 0-1. Invite collaborator

-  :  초대

* collaborator에 등록 되어야 해당 저장소에 대한 push 권한이 부여된다.



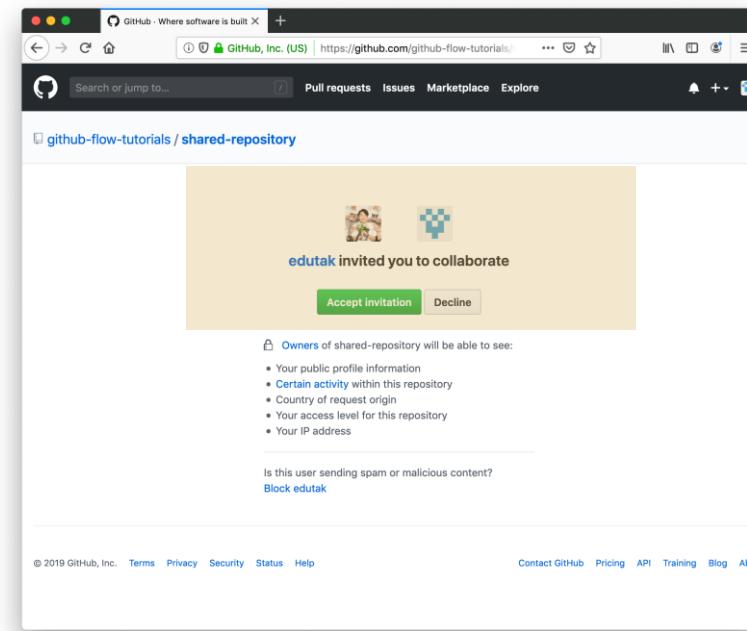
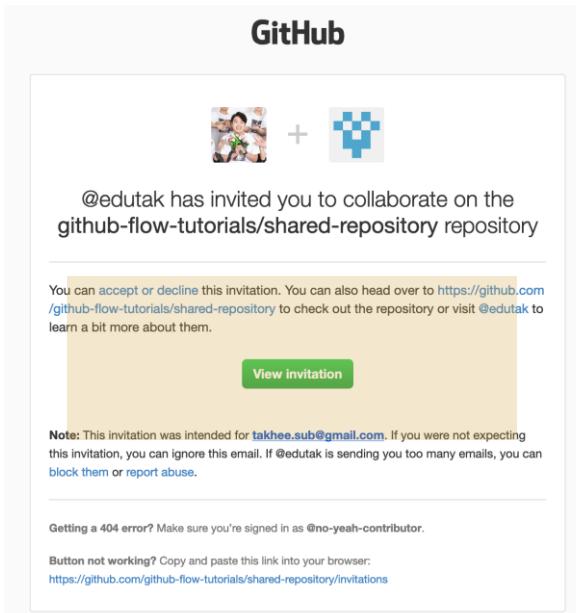
The screenshot shows the GitHub repository settings page for 'github-flow-tutorials / shared-repository'. The 'Collaborators & teams' tab is selected in the sidebar. The main content area displays the 'Default repository permission' section, which states that the organization has their default repository permission set to 'read'. It also shows the 'Teams' section, which currently has no teams assigned. At the bottom, there is a note about adding a collaborator.



The screenshot shows the 'Collaborators' form. It displays a message 'Awaiting no-yeah-contributor's response' and a status indicator 'Write'. There are buttons for 'Copy invite link' and 'Cancel invite'. Below this, there is a search bar with placeholder text 'Search by username, full name or email address' and a note about finding users by email if they've chosen to list it publicly. At the bottom right is a button labeled 'Add collaborator'.

step 0-2. Accept Invitation

-  : 이메일을 통한 초대 수락
 - * Tip! 이메일이 아닌 해당 저장소 주소 뒤에 /invitation을 붙이면 오른쪽 화면을 볼 수 있다.



step 0-3. Clone project(remote) repository

-  : Clone 이후 작업에 맞춘 작업 환경 설정을 마무리 한다.
 - 예를 들면, node의 경우 일반적으로 .gitignore에 node_moduels/가 등록되어 있으므로 npm install을 진행.

```
$ git clone {project repository url}
```

Shared Repository Model

2. 브랜치에서 작업 및 GitHub Push

Step 1. Feature branch 생성 및 작업

-  : 작업은 항상 독립적인 feature branch에서 한다.
 - master branch는 항상 배포 가능한 상태를 유지하고, 기능 개발은 독립적인 branch에서 작업
 - feature branch는 이름을 생성할 때, 기능을 명시적으로 나타냄

```
(master) $ git checkout -b feature/accounts-login  
(feature/accounts-login) $ touch develop-login.txt
```

* 작업시 항상 어떠한 branch에 있는지 확인!

step 2-1. Commit

-  : Commit으로 작업의 이력(history)을 남긴다.
 - Commit은 다른 사람들이 내가 한 작업들을 확인할 수 있는 이력이며, 코드의 변화에 맞춰 실시한다.
 - Commit 메시지는 매우 중요하며, 일관된 형식으로 해당 이력을 쉽게 파악할 수 있도록 작성한다.

```
(feature/accounts-login) $ git add develop-login.txt  
(feature/accounts-login) $ git commit -m 'Complete login feature'
```

* git status와 git log 명령어를 반드시 활용하여 상태를 파악하자.

step 2-2. Push to remote repository

- 💻 : 완성된 코드는 원격 저장소에 push를 한다.

```
(feature/accounts-login) $ git push origin feature/accounts-login
```

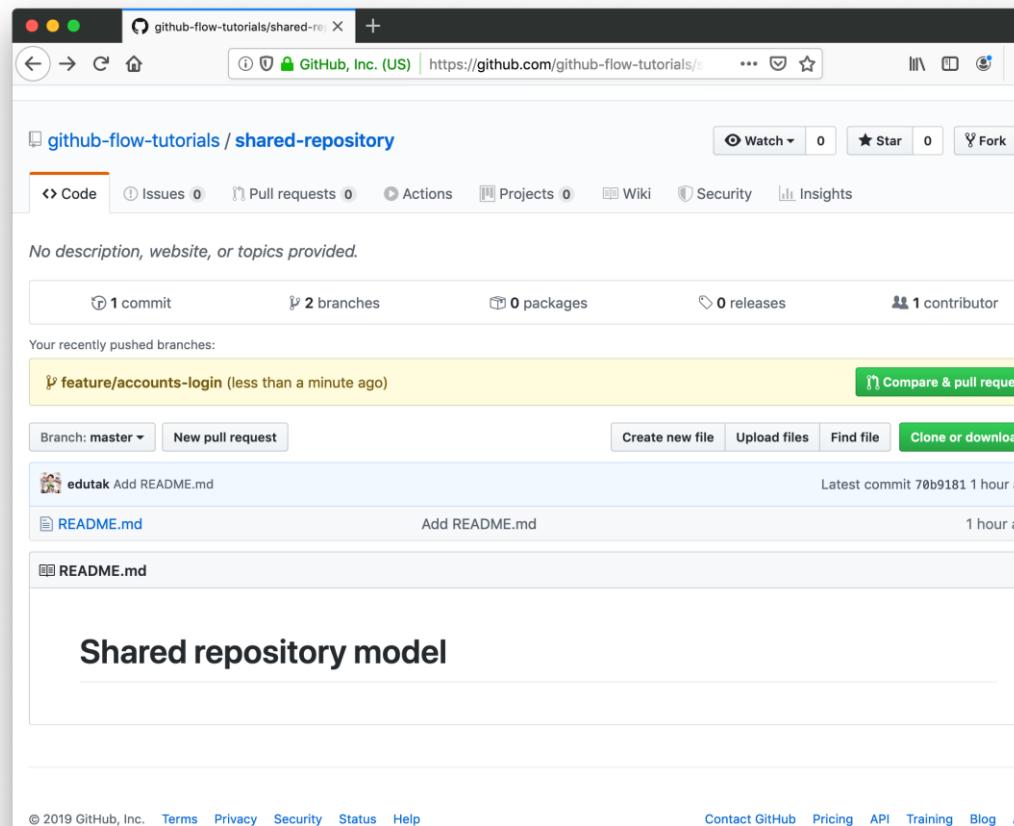
- * git push를 하기 이전에, 코드와 커밋 상태를 반드시 확인하자. (status, log)
- * 원격 저장소에 공개된 이력은 절대 변경 하여서는 안된다.

Shared Repository Model

3. Pull Request 생성

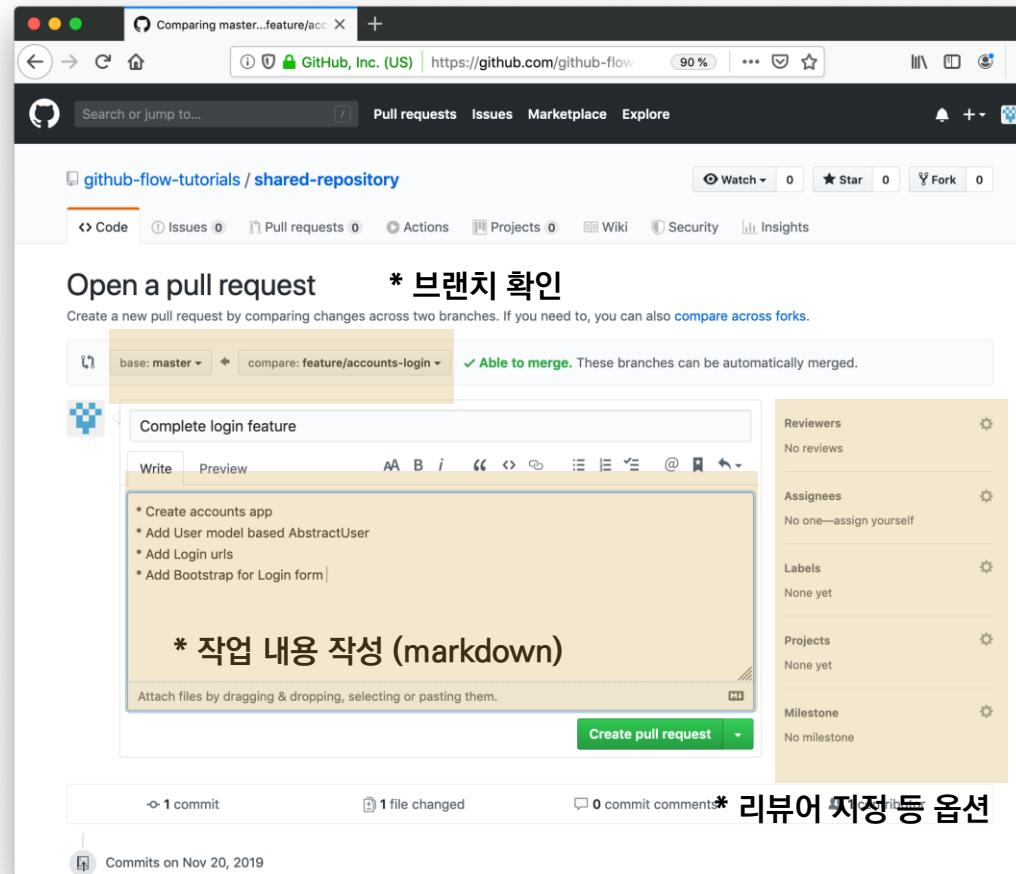
step 3-1. Open a Pull Request

-  : Github에 들어가서 Pull Request 버튼을 누른다.



step 3-2. Create Pull Request

- 💻 : PR과 관련된 설정을 진행한 후 요청을 생성한다.

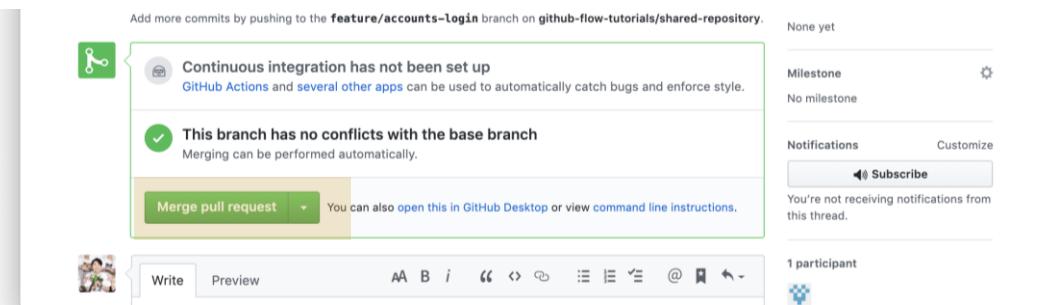


Shared Repository Model

4. Review 및 Merge

step 4. Merge pull request

-  : 작성된 코드를 확인 후 병합
 - 코드 리뷰를 진행하고, 관리자의 판단아래 병합한다.
 - 병합(merge) 과정에서 충돌이 발생할 경우 해결 후 병합을 진행한다.
 - master branch로 병합의 경우 코드가 반드시 배포 가능한 상태여야 한다.



Merged!

A screenshot of a GitHub repository page for 'github-flow-tutorials/shared-repository'. The repository has 3 commits, 2 branches, 0 packages, 0 releases, and 2 contributors. A pull request titled 'Merge pull request #1 from github-flow-tutorials/feature/accounts-login' has been merged. The latest commit is '6078103' made by 'edutak' 1 minute ago. The README.md file shows the text 'Shared repository model'.

A screenshot of the GitHub commit history for the 'master' branch of 'github-flow-tutorials/shared-repository'. It shows three commits on November 20, 2019: 1) 'Merge pull request #1 from github-flow-tutorials/feature/accounts-login' by 'edutak' (Verified, commit ID 6078103); 2) 'Complete login feature' by 'no-yeah-contributor' (commit ID 588946e); and 3) 'Add README.md' by 'edutak' (commit ID 70b9181). Navigation buttons 'Newer' and 'Older' are at the bottom.

병합 완료 후 개발을 한다면?

step 4. Merge pull request

-  : 다음 작업 준비!
 - 로컬 저장소에서는 merge된 branch는 삭제하고 master branch를 업데이트 한다.
 - 이후 1~3 과정을 반복한다.

```
(feature/accounts-login) $ git checkout master  
(master) $ git branch -d feature/accounts-login  
(master) $ git pull origin master  
(master) $ git checkout -b feature/new-feature  
(feature/new-feature) $
```

Fork & Pull Model

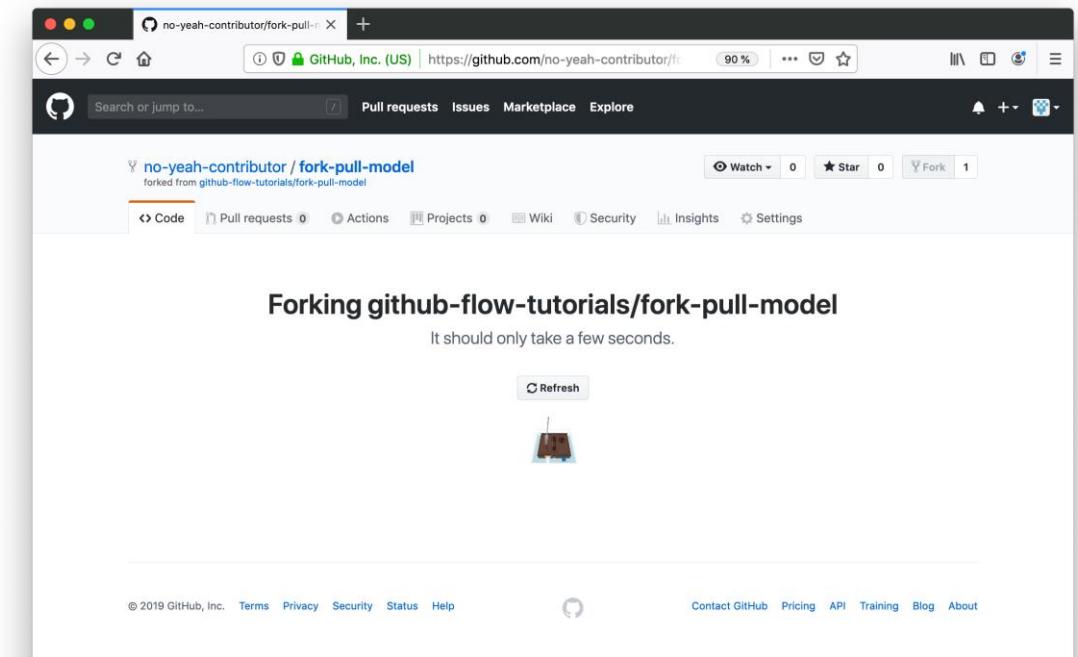
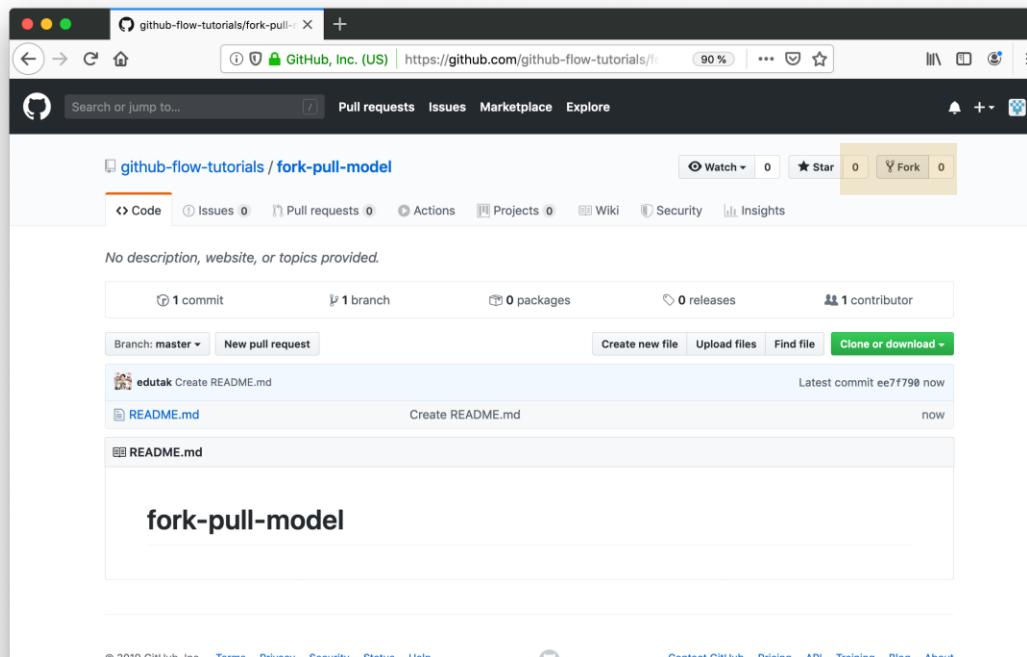
- Fork & Pull Model은 Repository에 Collaborator에 등록되지 않은 상태에서 진행
- Github 기반의 오픈소스 참여 과정에서 쓰이는 방식
-  : repository owner (project manager)
-  : contributor

Fork & Pull Request Model

1. Fork & Clone

step 0-1. Fork repository

-  : Forking project repository
 - 원격 저장소를 fork한다.
 - 내 저장소로 복제본을 가져옴으로써 로컬에서 작업 후 원격 저장소로 push할 수 있게 되는 것.



step 0-2. Clone project(remote) repository

-  : Clone을 하고 각 작업에 맞춘 작업 환경 설정을 마무리 한다.
 - 예를 들면, node의 경우 일반적으로 .gitignore에 node_moduels/가 등록되어 있으므로 npm install 등
 - **Clone시 반드시 본인 저장소인지 확인할 것**

```
$ git clone {project repository url}
```

Fork & Pull Request Model

이후 작업(커밋, push, PR)은
Shared Repository Model과 동일함

step 4. while True: but, upstream!!

-  : 다음 작업 준비!
 - 로컬 저장소에서는 merge된 branch는 삭제하고 master branch를 업데이트 한다.
 - 단, master branch는 원본 저장소를 받아와야 하며 별도의 원격 저장소를 추가하여 진행할 수 있다.
 - 혹은 GitHub에서 fetch upstream도 가능하다.

```
(feature/accounts-login) $ git checkout master  
(master) $ git branch -d feature/accounts-login  
(master) $ git remote -v  
origin https://github.com/i-am-contributor/fork-pull-model.git(fetch)  
origin https://github.com/i-am-contributor/fork-pull-model.git(push)
```

```
(master) $ git remote add upstream https://github.com/github-flow-tutorials/fork-pull-model.git  
(master) $ git pull upstream master  
(master) $ git checkout -b feature/new-feature  
(feature/new-feature) $
```

