Sprint 1. 인공지능의 이해

Part 3. 개발 협업 프로세스

개발자와 소통하기

개발적 관점과 사고방식

강사: 권구현 | rngus4656@gmail.com

Sprint 1 part3 목차

- DIKW 피라미드 구조의 이해
- 개발자 사고방식 살펴보기
- 전부 공개해도 좋은 이유

실습 자료: https://colab.research.google.com/drive/1kqJ9LikNGnTv8bw_9oVFQcD_aWQiGMyF?usp=sharing

S1 Part 3

DIKW피라미드 구조의 이해

AI는 이미 우리보다 많은 일을 더 잘하고 있다

그럼 우리는 AI에게만 의존해야 할까?

AI가 데이터 수집, 정제 모든 것을 잘하는데 왜 사람을?

AI는 이미 사람의 업무를 대부분 앞서고 있다.

✓ **AI가 앞서고 있는 업무들은 무엇일까?**

- **아이데이션** 확장, 향후 **액션 플랜** 제안
- 데스크 리서치, 연관 자료 탐색, **광범위한 데이터 수집**
- **데이터 정리 및 요약**, 패턴 기반 **데이터 분석**
- 반복 업무 자동화, **개발 보조**
- 결과물 검수 및 수정 제안, 흐름과 내용 검토
- UX/UI 시안 제안, **디자인 초안 설계**

👉 반복 업무, 창작, 디자인 등 **대부분의 작업을 빠르고 다양하게 수행**할 수 있다.

✓ **역 질문. 그럼 왜 아직도 AI가 아니라 사람의 수요가 있을까?**

- AI에게 배우고 싶은 내용을 찾고, 배우면 되는데 **어째서 사람의 강의를** 들을까?
- 문제 정의부터 솔루션까지 **기획 전체를 AI 에이전트로 해결하면** 되지 않을까?
- 향후 액션 플랜과 업무 우선순위까지 전부 알아서 해주는데 **AI에게 모든 걸 맡기는 것이** 낫지 않을까?

👉 **AI를 의존하지 않아야 할 이유**가 있을까?

S1 Part 3

AI vs 사람

AI가 잘하는 것과 사람이 해야 할 것을 구분하는 것이 200% 활용의 시작

✓ AI가 잘하는 것

- 빠르고 다양한 **작업물 생성 및 제안**
- 문제 상황에 따른 **분석 및 솔루션**
- **작업물 검토**와 보완점 **개선사항 정리**
- 대량의 단순 반복 작업을 **패턴을 학습**하여 빠르게 처리
- 결과물 완성도 **검수 및 피드백**

✓ 사람이 해야 할 것

- **지적재산권 보호** 및 방향성을 기반한 작업물 채택
- 솔루션 점검 및 **상위 결정권자와의 협의**
- 전체적인 맥락과 흐름의 **적합성 판단**
- 작업 체계 구체화 및 **정책 수립 결정**
- 리스크에 관한 **책임과 권한 부여**

✓ AI에게 전적으로 맡겨서는 안되는 영역이 여전히 존재한다.

- **윤리 및 재산권 보호** (폭력·선정적·정치적 콘텐츠 제한, 창작물·지적재산권 보호)
- **책임 및 권한 관리** (전문지식이 필수적인 의료·법률·정책 등)

👉 **단 한번의 할루시네이션**이 **서비스 존폐 위기**를 불러올 수 있다.

- **AI는** 판단의 기준을 사람보다 월등히 더 빠르고 **효율적으로 준비**해 줄 수 있다.
- 하지만 **결정과 책임은 사람의 몫**이다.

AI 의존성 vs AI 활용성

어디까지 AI에게 맡기고, 어디서부터 사람이 개입해야 할까?

✓ **항상 불변의 정답을 찾을 수는 없다.**
상황과 맥락에 따라 기준을 정하는 사람이 필요하다.

- **같은 문제에도 상황마다** 누가, 어떻게, 무엇을 위해 써야 되는지에 따라 **답은 달라질 수 있다.**
- 종종 **그럴듯한 정답보다** 오히려 직접 부딪혀 보고 **실패하는 경험이 필요**한 순간이 있다.
- **협업을 하며** 각양각색의 사람, 다양한 경험, **넓은 시야를 가진 PM의 인사이트**는 **아직 AI가 대신할 수 없다.**
- **AI가 필수 덕목**으로 여겨 지기도 하지만, **분야에 따라서**는 전통을 고집하는 것이 더 희소가치가 있을 수 있다.

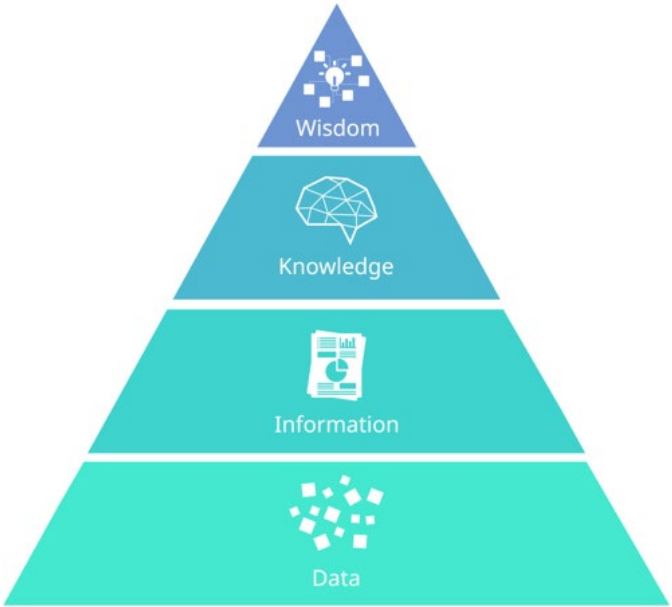
✓ **판단과 결정은 사람에게, 속도와 퍼포먼스는 AI 에게**

- 어떤 사람은 **AI 없이** 답을 찾는 것을 선호하기도, 어떤 사람은 **매 순간 AI를** 활용하기도 한다.
- AI를 온 종일 붙들고 있어도 **답을 찾지 못하는 문제**가 있는 반면, 하나부터 열까지 AI 활용으로 **해결할 수 있는 문제**도 있다.
- 관점에 따라, 사람에 따라, 분야와 그 외 다양한 **컨텍스트를 읽어내고 조화를 이루어 내는 것이 필요**하다.

👉 **AI에 의존**하는 사람이 아니라, **AI를 활용하는 사람**으로 보이기 위해서는 **판단을 잘하기 위한 연습**이 중요하다.

DIKW 피라미드

DIKW에서 AI와 사람의 역할을 나눠서 생각해보기



Wisdom(지혜)

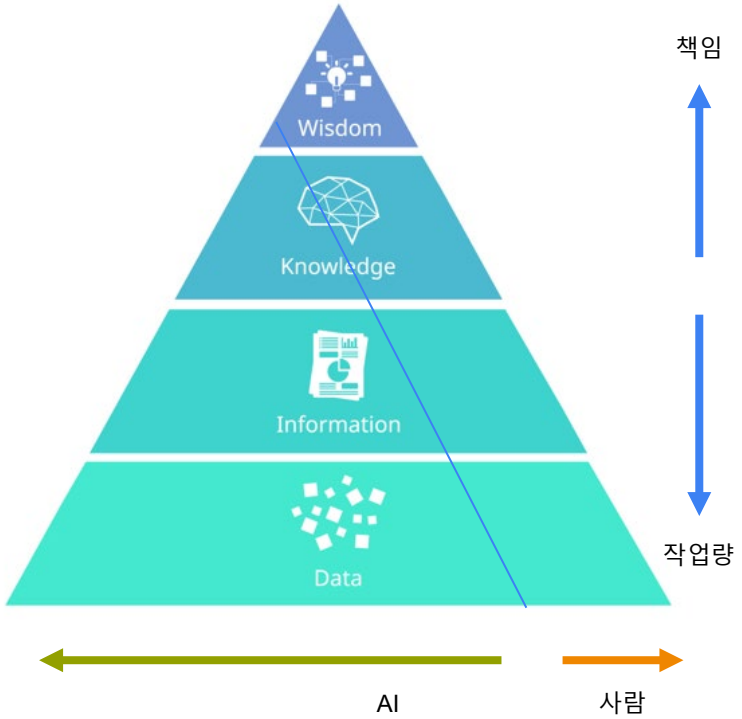
Knowledge(지식)

Info(정보)

Data(자료)

DIKW 피라미드

DIKW에서 AI와 사람의 역할을 나눠서 생각해보기



Wisdom(지혜)

- 지식을 바탕으로 얻은 인사이트를 통한 **의사결정 및 전략 수립**

  e.g.) 전체 타겟에 중 30대 여성을 핵심 타겟으로 설정
  해당 타겟에 맞춘 광고 모델과 CTA 구조 설계

Knowledge(지식)

- 정보 관계를 통해 얻어낸 **학습 정보를 일반화**

  e.g.) 장바구니가 많을수록 구매 가능성도 높아진다.
  30대 여성은 특정 상품군에서 반복 구매 성향이 있다.

Info(정보)

- 데이터를 기반으로 **연관관계와 패턴 정리**

  e.g.) 장바구니가 많을수록 구매 횟수도 많다, 30대 여성 구매 비중이 가장 높다.

Data(자료)

- 객관적, 단순 **사실이나 수치**

  e.g.) 구매 횟수, 장바구니 수, 회원 정보, 클릭 로그 등

AI는 자료와 정보를 빠르게 준비해 준다.

S1 Part 3

개발자 사고방식 살펴보기

프로그래밍하는 사람들은 무슨 생각을 가지고 있을까

복잡하고 어려운 문제를 마주하는 자세

고차원적 연산으로 답을 도출하는 척척박사 vs 가장 작은 단위까지 쪼개서 나누는 조직적 부서

✓ 파이썬답게 코딩한다는 것은 무엇일까?

파이썬에서는 **import this**를 입력하면 숨겨진 이스터에그인 The Zen of Python - 파이썬의 선(철학)이 등장한다.

Beautiful is better than ugly.	아름다운 것이 못난 것보다 낫다 (가독성 및 간결성)
Explicit is better than implicit.	명시적인 것이 암묵적인 것보다 낫다 (요구사항 정의서)
Simple is better than complex.	단순한 것이 복잡한 것보다 낫다
Complex is better than complicated.	복잡한 것이 난해한 것보다 낫다
Flat is better than nested.	평평한 구조가 중첩된 구조보다 낫다
Sparse is better than dense.	여백이 있는 것이 밀집된 것보다 낫다
Readability counts.	가독성은 중요하다
Special cases aren't special enough to break the rules.	규칙을 어겨도 될 만큼의 특별한 상황이란 것은 없다.
Although practicality beats purity.	비록, 실용성이 이상을 넘어서더라도.
Errors should never pass silently.	오류는 절대 조용히 넘어가서는 안된다.
Unless explicitly silenced.	단, 의도적으로 무시하는 경우는 제외.
In the face of ambiguity, refuse the temptation to guess.	애매함을 마주했을 때, 추측하려는 유혹을 거부해라.
There should be one-- and preferably only one --obvious way to do it.	분명 하나의 – 바람직하고 유일한 – 명백한 방법이 있을 것이다.
Although that way may not be obvious at first unless you're Dutch.	비록, 그 방법이 처음부터는 명백하지 않을 수 있을지라도.
Now is better than never.	지금 하는 것이 전혀 안 하는 것보다 낫다.
Although never is often better than *right* now.	하지만, 종종 '지금 당장'하려는 것 보단, 차라리 안 하는 것이 나을 때도 있다.
If the implementation is hard to explain, it's a bad idea.	구현 방식이 설명하기 어려운 것이라면, 좋지 않은 아이디어다.
If the implementation is easy to explain, it may be a good idea.	구현 방식이 설명하기 쉬운 것이라면, 좋은 아이디어일 수 있다.
Namespaces are one honking great idea -- let's do more of those!	Namespace는 정말 좋은 아이디어다. 많이들 사용하자!

다이나믹 프로그래밍

작은 문제부터 확실하게, 효율적으로

✓ 다이나믹 프로그래밍이란?

대규모의 복잡한 연산이 필요할 때, 메모리 공간의 효율성을 비약적으로 향상 시키는 방법으로,
큰 문제를 작은 단위로 쪼개어 나누고, **부분 문제들의 답을 재사용**해 **최종 문제를 해결**하는 방식으로 실행 된다.

✓ 다이나믹 프로그래밍(이하 DP)의 작동 조건

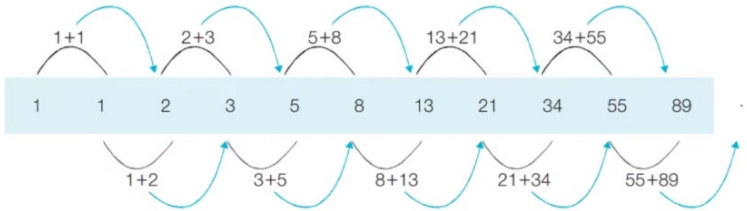
- 최적 부분 구조(Optimal Substructure)
  큰 문제를 작은 문제로 나눌 수 있으며, 작은 문제의 답을 모아서 큰 문제를 해결할 수 있다.
- 중복되는 부분 문제(Overlapping Subproblem)
  반복되어 나오는 동일한 작은 문제의 답은 항상 같다.

대표적인 DP의 적용 예시

피보나치 수열: 이전 두개의 항의 합이 현재의 항이다.
이를 점화식으로 표현 하면 다음과 같다.

$$a_n = a_{n-1} + a_{n-2}, \quad a_1 = 1, \quad a_2 = 1$$

40번째 항을 구하기 위한 연산은 얼마나 걸릴까?



내용 참고: https://youtu.be/5Lu34WIx2Us?si=lOxarTgRmjZkA_RF 동빈나_다이나믹 프로그래밍

직관적 해결 방식

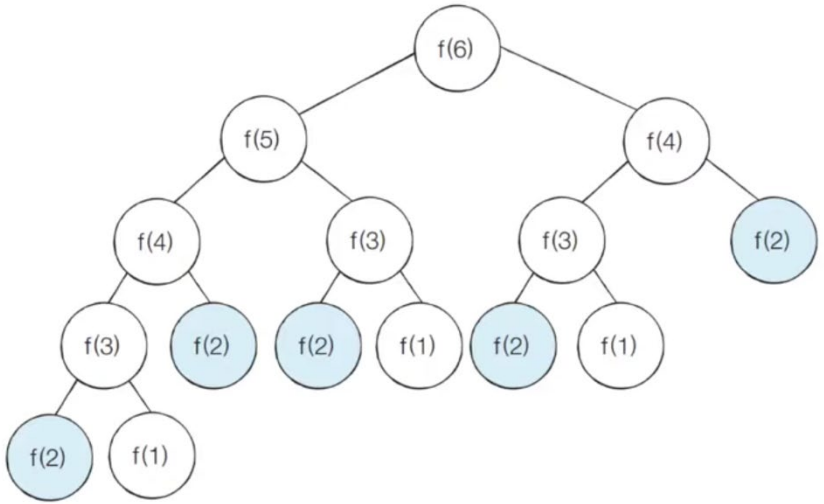
계산식대로 연산을 수행 한다면?

✓ 직관적으로 접근 했을때,

문제가 작고 단순하다면, 직관적으로 해결해도 큰 문제가 되지 않는다.
하지만 **일정 수준 이상의 규모**를 넘어가게 된다면,
입력값이 조금만 커져도 필요한 **연산량은 폭발적**으로 늘어나게 되고,
이는 컴퓨터의 **연산속도와 비용에 직접적 영향**을 미치게 된다.

f(40)을 구하기 위해 필요한 연산 횟수

f(6)을 구하기 위해서는 **f(2)가 총 5번 호출** 된다.
만약, f(7)을 구하기 위해서는 **f(2)는 8번 호출** 되며,
이러한 규칙으로 f(40)에서의 **f(2) 호출은 기하급수적**으로 늘어나게 된다.



피보나치 수열의 시간 복잡도는 $O(2^n)$ 이다.
즉, 40번째 항의 해를 구할 때의 시간 복잡도는
$O(2^{40}) = 1,099,511,627,776$ 이다.

DP의 Top-down

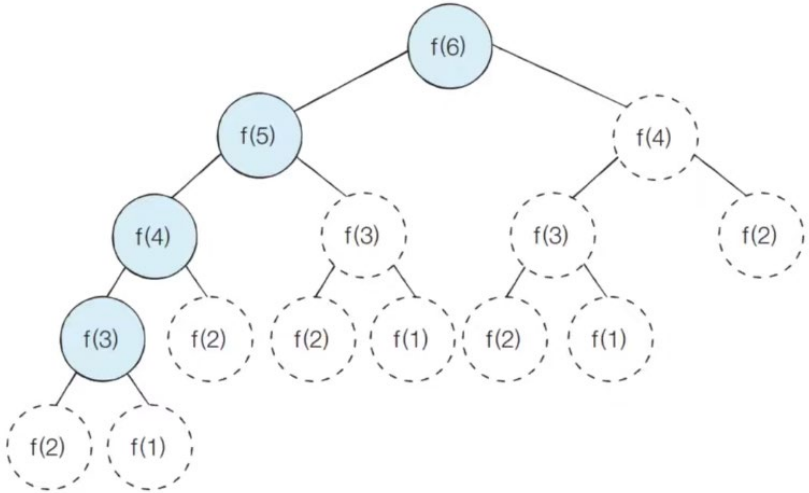
하향식 접근 방법

✓ 한 번 계산한 값은 '기억'한다.

하향식, Top-down, **Memoization** 이라고도 한다.
탑 f(6)을 구하기 위해 f(5)를 구하고, **f(1)까지 하향식 접근**을 수행한다.
이때, **한번 구한 값은 저장해 두어 중복 연산을 하지 않는다.**
따라서, 6번째 항까지의 **연산은 각 항당 한번씩만 수행** 된다.

✓ 중복 연산 제거의 효과

중복 연산만 제거해도, 각 항의 값이 최대 한 번만 계산된다.
F(40)의 값을 구하기 위해서도 **중복 연산을 반복하지 않으며,**
문제 규모에 비례한 연산만 수행하면 된다.



👉**반복적인 문제를 해결**하는 것 만으로도 **효율성이 크게 향상** 됨

Memoization은 한번 도출 된 값을 저장해 두는 기법으로, DP에서만 쓰이는 방법은 아니다.

DP의 Bottom-up

상향식 접근 방법

✓ DP에서 주로 쓰이는 방식

상향식, Bottom-up, **Tabluation** 이라고도 한다.
해결 가능한 가장 작은 문제부터 차례대로 해결하고,
그 결과를 쌓아 큰 문제의 답을 만들어 간다.

✓ 기초 공사를 먼저 하고 건물을 올린다.

- f(1), f(2)는 1로 값이 정해져 있는 상태
- **f(3)을 계산**하고, **그 값을 통해 f(4)의 값을 도출**한다.
- 이를 통해 **최종적으로 f(40)까지 도달**한다.

상향식 또한, **한번 계산 된 값을 통해 다음 항을 계산**하므로
각 항의 값은 단 한 번만 계산된다.

👉최소문제부터 해결하며, 중복 연산이 구조적으로 차단된다.

Top-down	Bottom-up
최상위 문제부터 시작	최하위 문제 해결부터 시작
재귀 함수 사용	반복문 사용
체계적, 구조가 복잡할 수 있음	간결함, 코드가 길어질 수 있음

Top-down과 Bottom-up 중 어떤 것을 사용해야 좋은지는
문제의 유형에 따라 다르다.

때로는 **Top-down의 체계적 구조가 안정적인 결과**를 줄 수 있고,
Bottom-up으로 순차적인 해결이 명확할 때가 있을 것이다.

PM으로서 직면하는 문제도 이와 유사하며,
개발자 역시 같은 고민을 하고있다.

개발자와의 커뮤니케이션

개발자의 주 업무 '거절'?

✓ **개발자는 왜 그렇게도 의심이 많은 걸까?**

- **요청이 모호하고 추상적**이라면, **규모가 커질수록 수정 리소스는 급격히 증가**한다.
- 작고 사소하더라도 **명확한 출발선이 있어야 구조를 설계**하기 용이하다.
- 꼭 필요한 것인지, 예외 처리가 필요한 것인지 **확인 요청을 반복**하게 된다.

👉 **리스크 제거 과정**이 없다면 위험하다고 판단할 수 있다.

PM	개발자
• 전체 그림 구상, 방향성 확보	• 단위별 구현 가능성, 실행 조건
• 가능성 탐색, 실험적 사고	• 에러 발생 가능 요인 체크
• 성공했을 때의 **지표와 성과측정**	• **실행 유지보수**와 오류 발생시 트러블슈팅

👉 **서로 같은 목표**를 가졌지만, **역할의 차이**에서 생길 수 있는 오해

S1 Part 3

전부 공개해도 좋은 이유

오픈 라이선스의 기본 이해

힘들게 작업했는데... 전부 공개한다고?

수료생을 위한 산출물 공유 노션

그럼 정말, 다 가져가도 되는 걸까?

활용 가능하다면 대답은 YES, 하지만 출처 표시는 기본

✓ 내 작업물을 그대로 가져다 쓰면 어떡하지?

- 면접관과 투자자는 **결과만 보지 않는다.**
- 면접과 발표에서는 **작업물의 맥락, 선택 이유, 사고 과정** 등을 함께 보게 된다.
- 그대로 **가져오기만 한 작업물을 검증**하는 방법은 **생각보다 크게 어렵지 않다**.

✓ 모든 창작은 모방에서 시작한다.

마음껏 염탐하고, 잘하는 사람이 있다면 **그대로 따라해보려 노력**해도 된다. **그것이 공유 페이지의 목적**이기도 하며,
역설적으로, **다른 사람의 작업물을 보며 내용을 습득하고 체화** 하다 보면, 어느새 **카피가 아닌 오리지널리티**로 거듭나게 될 것이다.

발전을 위한 기록과 경험 공유는 **제 3자**가 **다양하게 해석하고 피드백** 하는 것을 토대로 **제 2의 발견을 할 수 있는 가능성**을 가지게 된다.
이러하듯 Github, 허깅 페이스, Colab의 오픈소스 등 **'작업물을 독점'하는 것이 아닌 '안전하게 나누는'문화**를 왜 채택했는지 알 수 있다.

하지만, 정말 보안에 취약한 내용이나 개인정보가 있다면 공유 페이지에서는 노출이 되지 않도록 주의하는 것이 좋다.
그런 이유에서 Github처럼 '오픈 라이선스'를 명시해두거나, 저작물 보호 규정을 간단하게 숙지하는 것도 필요하다.
일반적으로 상업적 사용 가능여부, 수정 가능여부, 재배포 가능여부, 출처 표시 여부 등을 기준으로 한다.

우리 수료생 커뮤니티의 공유 속에서도 경험이 누적되어 엄청난 파급효과가 될 수 있다!

Git과 GitHub

Git

Git은 2005년 리누스 토르발스가 처음 개발한 DVCS(Distributed Version Control System)이다.
소프트웨어 개발 및 소스 코드 관리에 주로 사용되었지만,
현재는 GitHub를 통해 개발자들의 작업물 공유와 협업 툴 및 저장소로도 활용되고 있다.

GitHub

Git으로 관리하는 프로젝트들을 온라인 웹에 공유할 수 있게 해주는 인터넷 호스팅 서비스다.
기본적으로 CLI형식인 Git을 GUI로 사용할 수 있으며, 단순히 온라인에 백업하는 것을 넘어
전 세계 개발자들과 협업(Pull Request)하고, 코드 리뷰, 오픈소스 공유를 하는 커뮤니티/플랫폼 역할을 한다.

내용 출처: https://ko.wikipedia.org/wiki/%EA%B9%83_(%EC%86%8C%ED%94%84%ED%8A%B8%EC%9B%A8%EC%96%B4)

Git 실습

Git, 어디까지 알아야 하나요?

✓ 개발자와의 협업 프로세스 경험

Git에서 내 컴퓨터(Local)의 버전 기록을 기록하고, GitHub 웹(Remote)에 업로드 할 수 있는 것을 목표로 따라 해보기

✓ Git 실습 목표

- Git 설치 및 **GitHub Repository 생성** 및 **MIT 오픈 라이선스** 설정
- **Git Clone - Readme.md 수정 - Git add - Git commit - Git push** 실습
- Git을 완벽히 이해하는 것이 아닌 **'용도 파악'**
- 능숙하게 사용하는 것이 아니라 한 번 **설치, 실행해보는 '경험'**
- '기술 마스터'가 목표가 아니라 **기본적인 '환경 세팅'까지 성공하면 OK!**

✓ 실습 전 유의사항

- 운영체제 / 환경마다 설치 방법이 다를 수 있음
- 중간에 막히는 것이 있거나, 놓쳐서 **따라가지 못하는 것도 정상**
- 지금 안 되는 것을 해결하려는 것 보다, **강의를 최대한 따라오며 교안으로 복습해보는 것을 목표**로 실습

👉 안되는 것이 있거나, **문제 해결이 필요하다면 쉬는 시간, TIL시간을 활용하여 지원요청!**

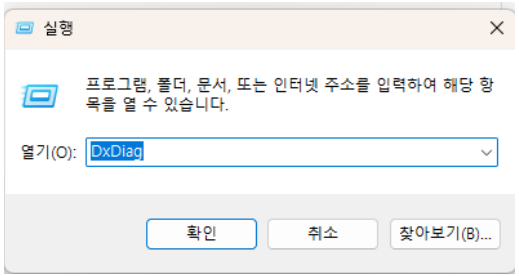
지금 당장은 따라오는 것만 목표로 해도 OK, 능숙해지는 건 천천히!

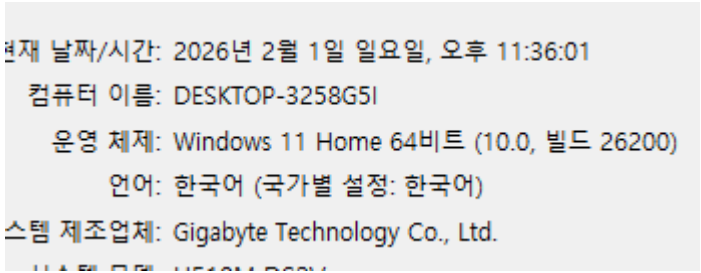
Git Install

운영체제에 맞는 Git 설치

Windows - Git for Windows/x64 Setup 설치
https://git-scm.com/install/windows

32비트, 64비트 여부 모를경우 (32비트일 경우 도움요청)
윈도우(시작)키 + R 실행 후 DxDiag 입력 - 64비트 확인



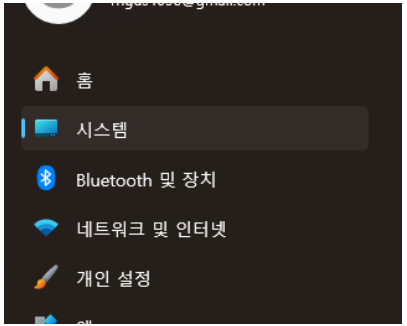


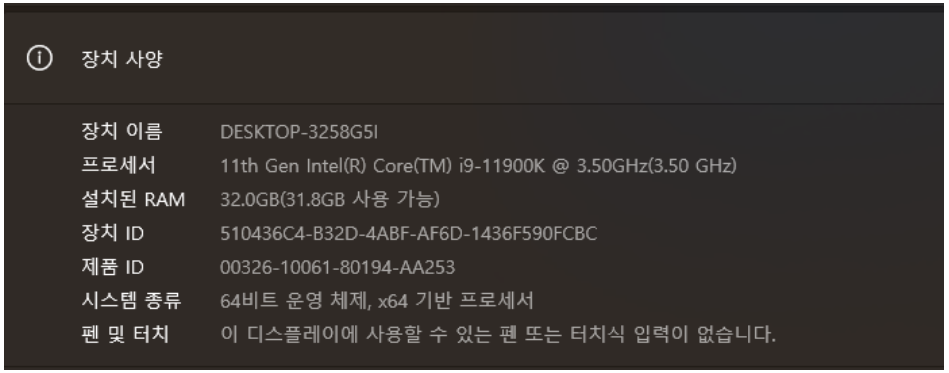
macOS
https://brew.sh/ Homebrew 설치 (이미 설치된 경우 생략)
https://git-scm.com/install/mac git 설치

Git Install

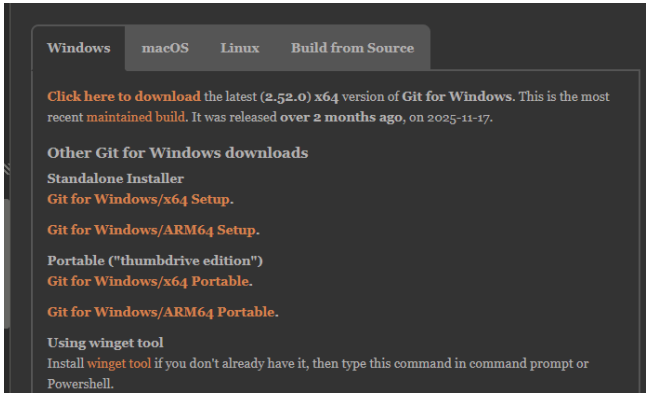
Windows x64, ARM64 확인 방법

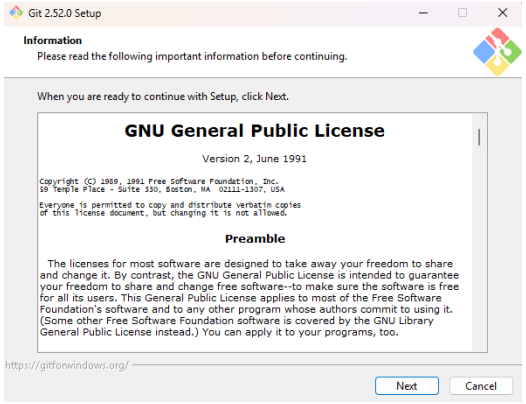
시작버튼 - 설정 - 시스템 - 정보 - 장치사양의 시스템 종류에서 확인

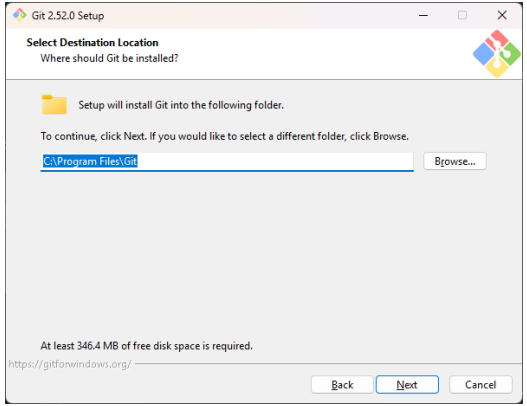


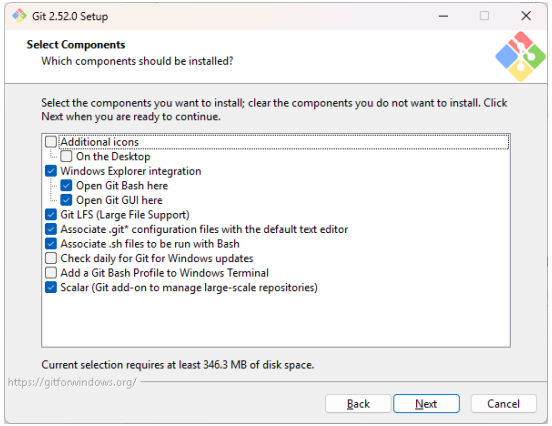


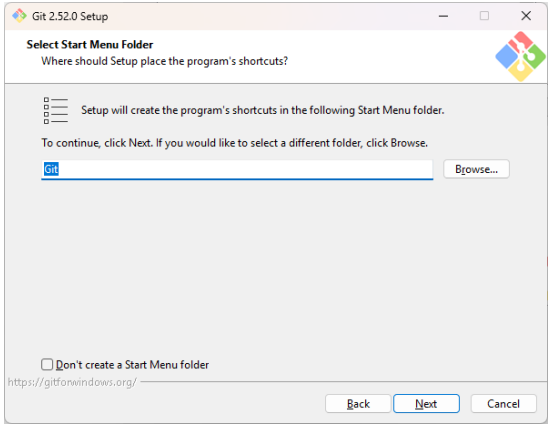
S1 Part 3 Git for Windows | 개인적인 개발 환경이 있지 않는 이상 전부 디폴트 옵션으로 Next

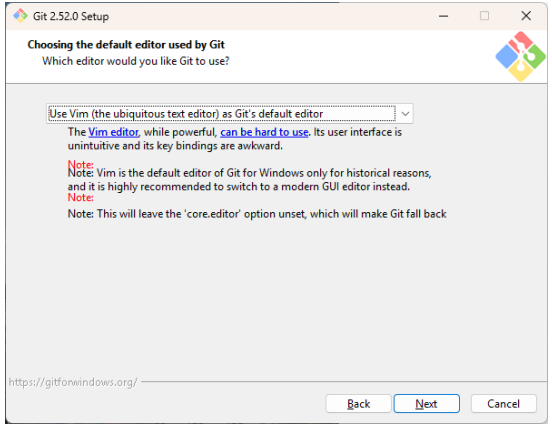




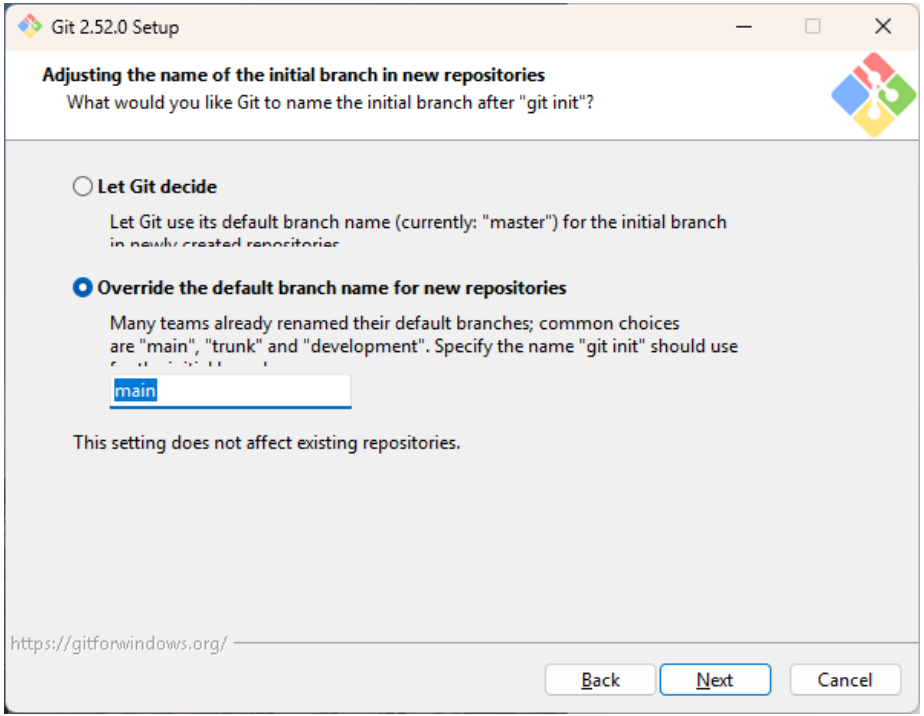






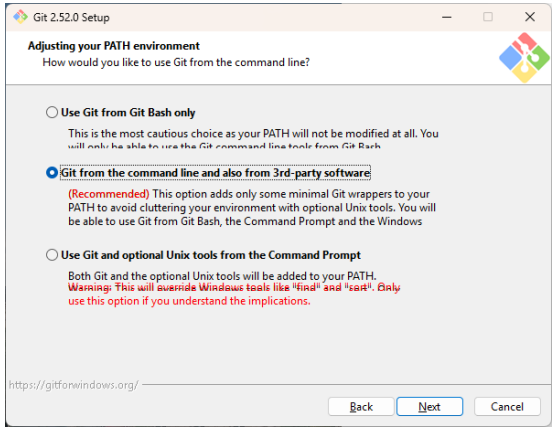


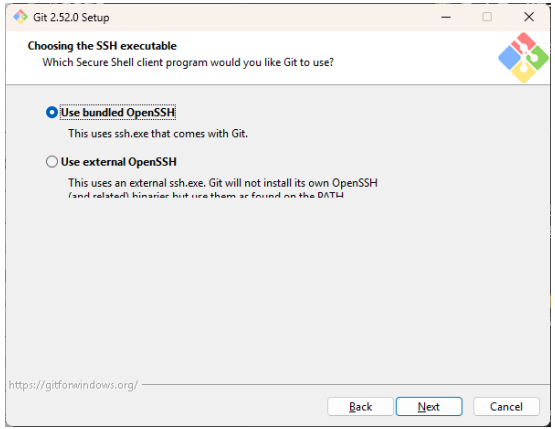
Git for Windows | branch name은 ‘main’으로 한다.

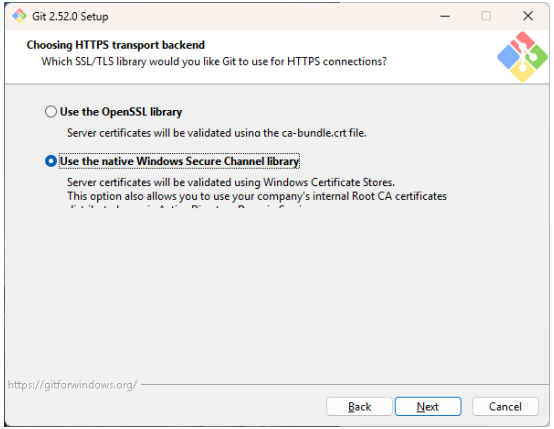


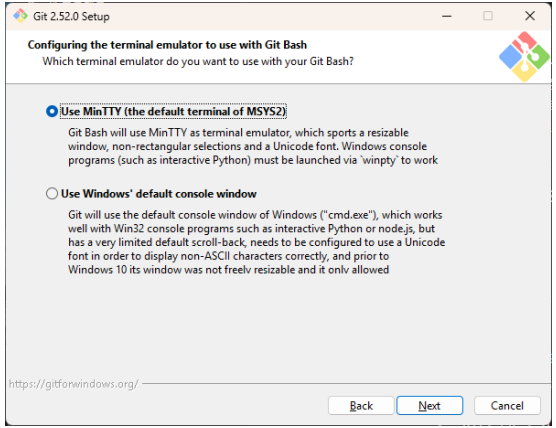
기존 사용자일경우 ‘master’로 되어 있을 수 있음
신규 설치시 반드시 ‘main’으로 선택
개인적인 branch name은 협업시 권장하지 않음

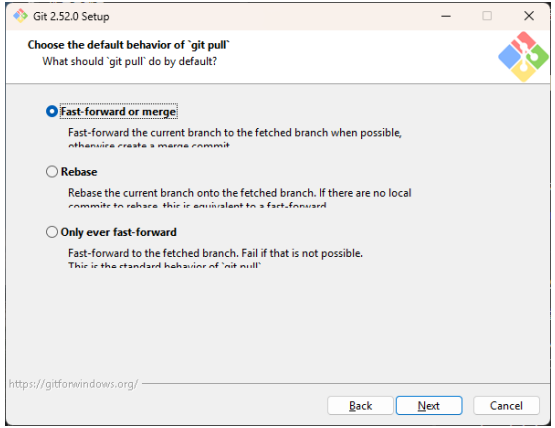
Git for Windows | 개인적인 개발 환경이 있지 않는 이상 전부 디폴트 옵션으로 Next

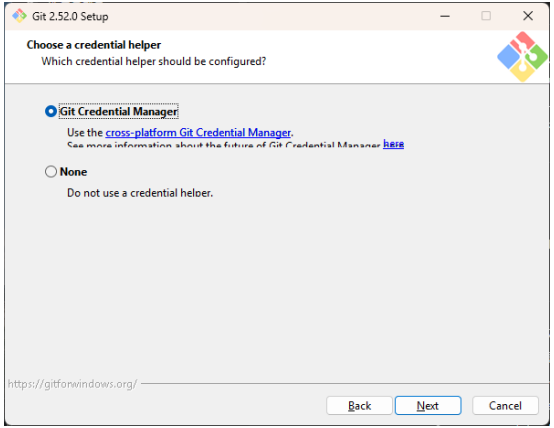




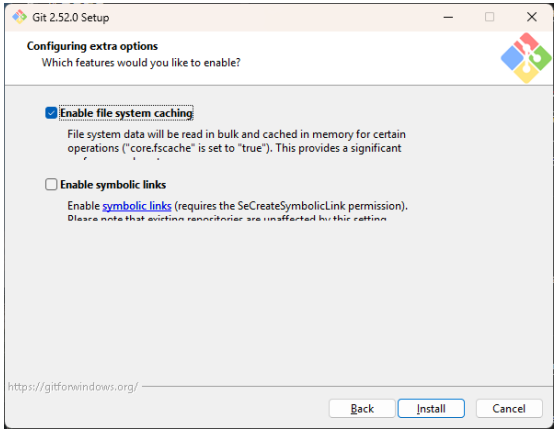


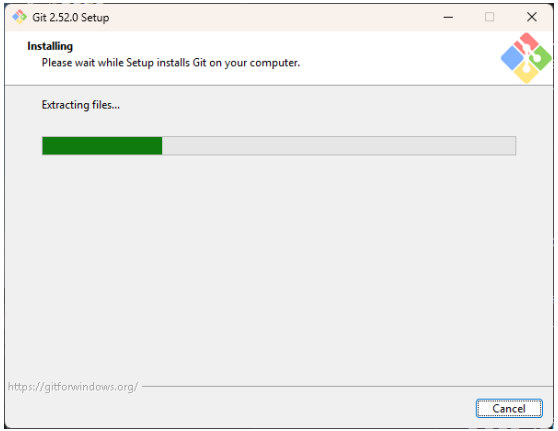


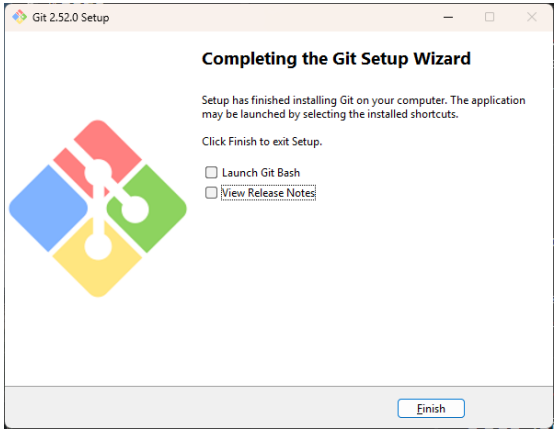


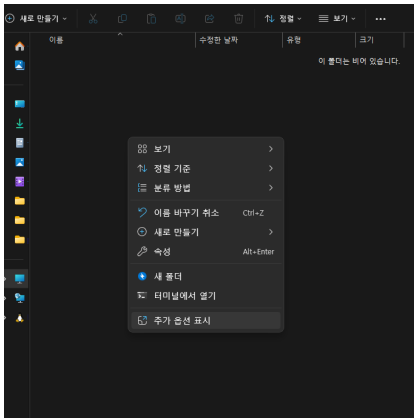


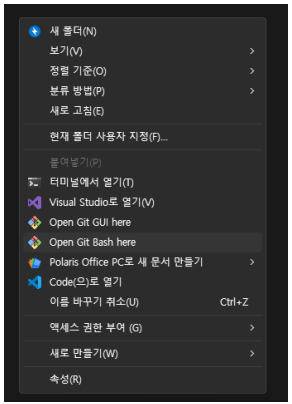
S1 Part 3 Git for Windows | 개인적인 개발 환경이 있지 않는 이상 전부 디폴트 옵션으로 Next





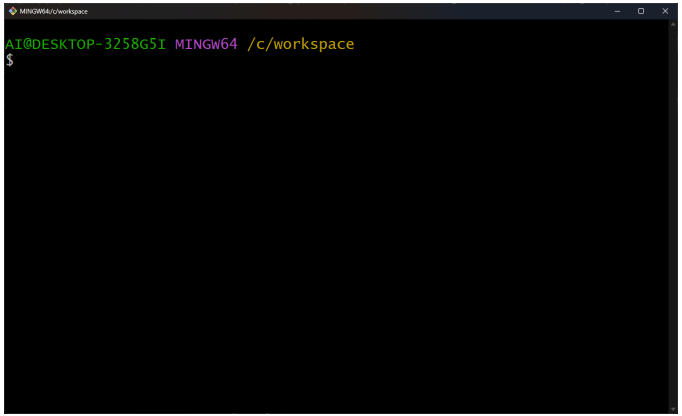






C:\workspace
폴더 생성하여 이동

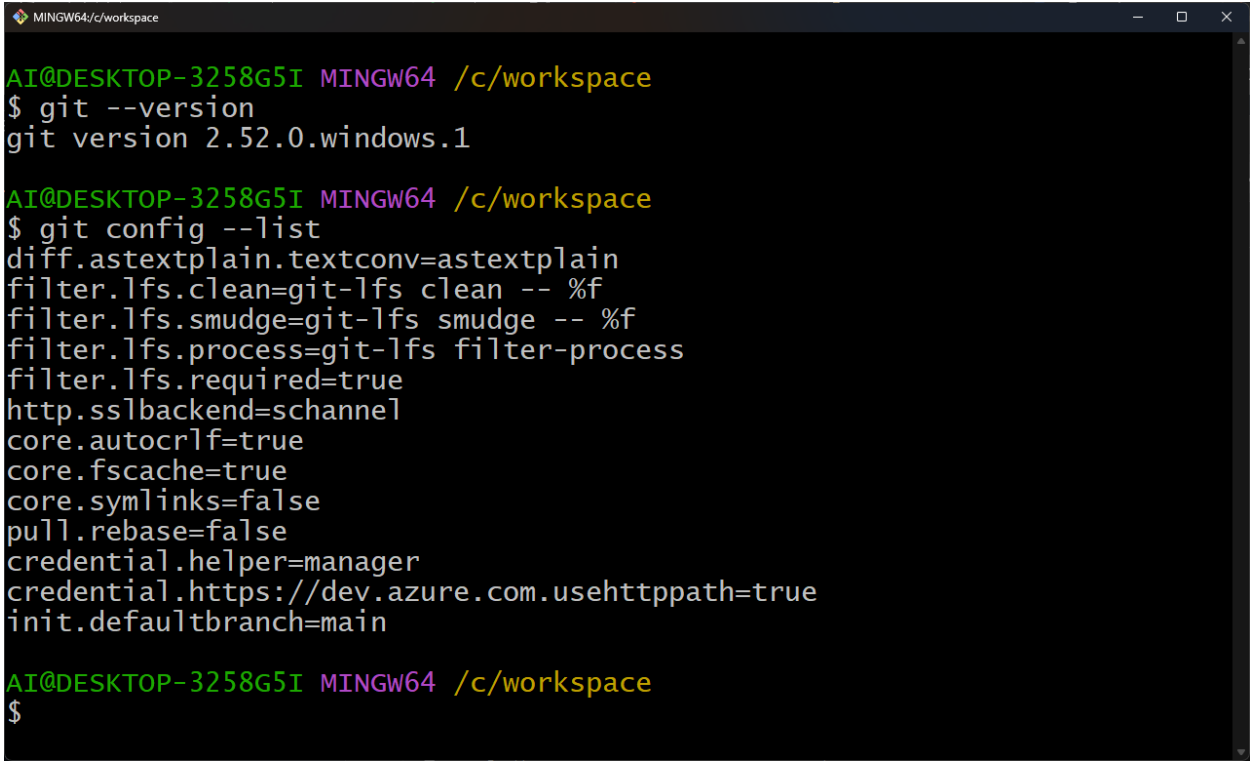
폴더 빈 공간 우 클릭
추가 옵션 표시 클릭
Open Git Bash here
실행



Git Bash 실행 확인

✓ **Git Bash 실행 후**
$ 표시 되어있는 줄 선택한 뒤
git --version
입력 후 엔터, **버전 확인**

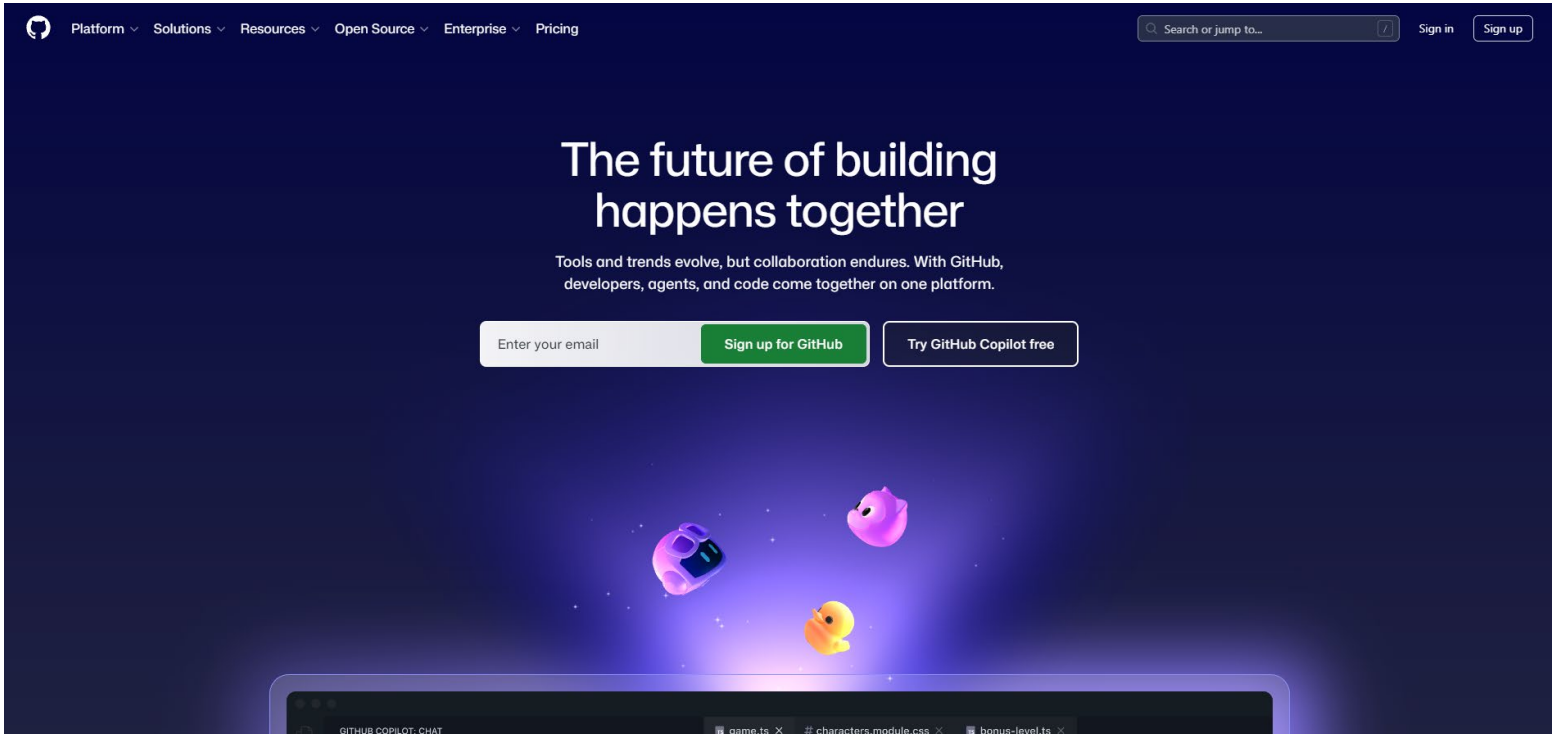
그 뒤에 동일하게 입력줄 선택 후
git config --list
입력 후 엔터, **정상 작동 확인**



GitHub Sign in

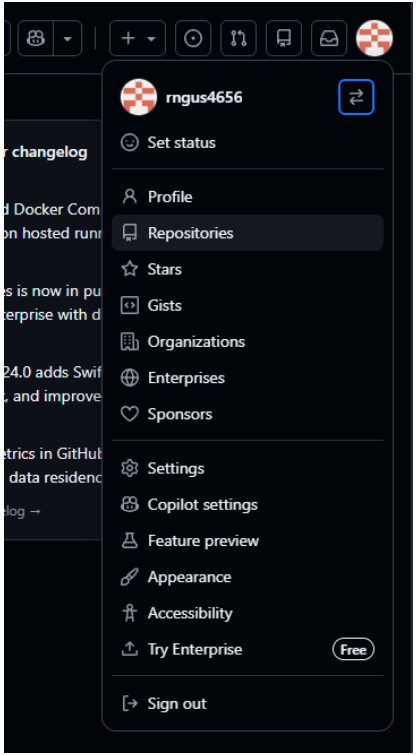
계정이 없을경우 Sign up으로 계정 생성 후 Sign in

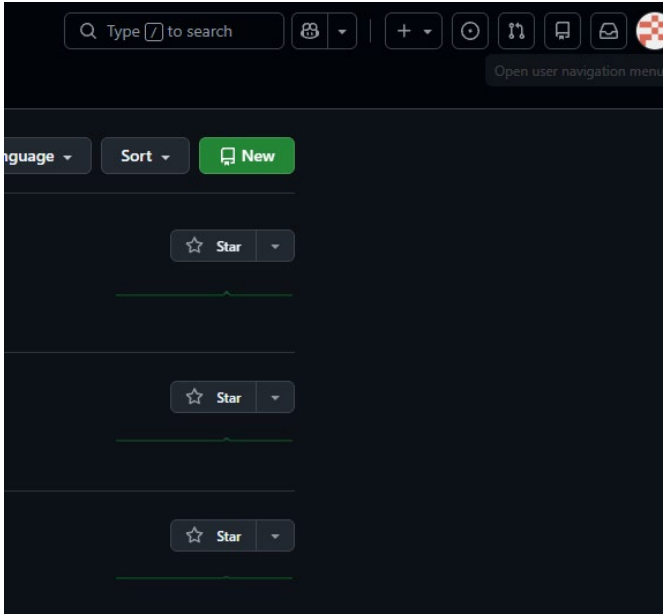
https://github.com/ 이동 후 Sign in



GitHub - Repository 생성

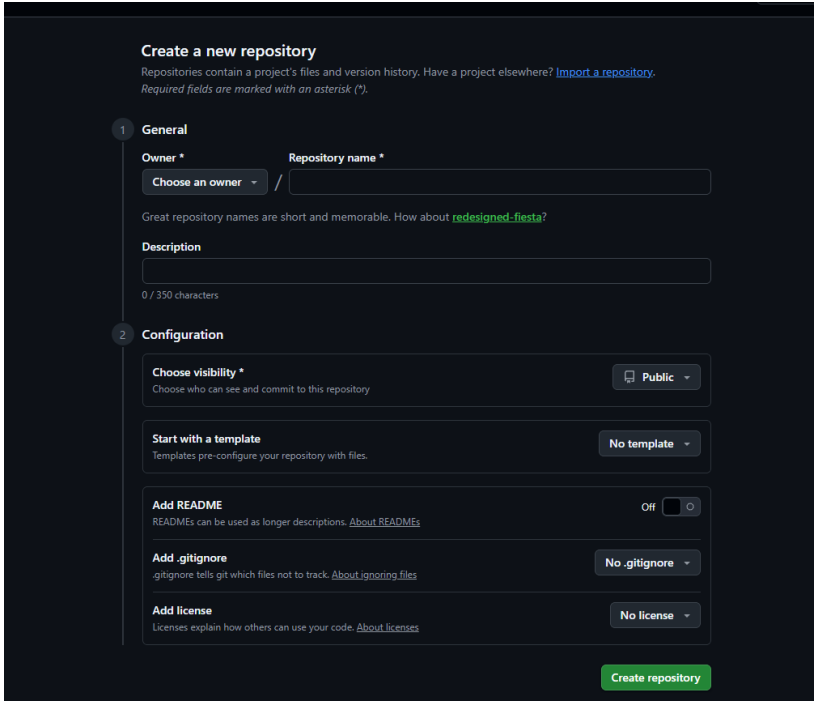
Sign in 후, 우측 상단의 프로필 클릭 - Repositories 클릭 - New버튼 클릭

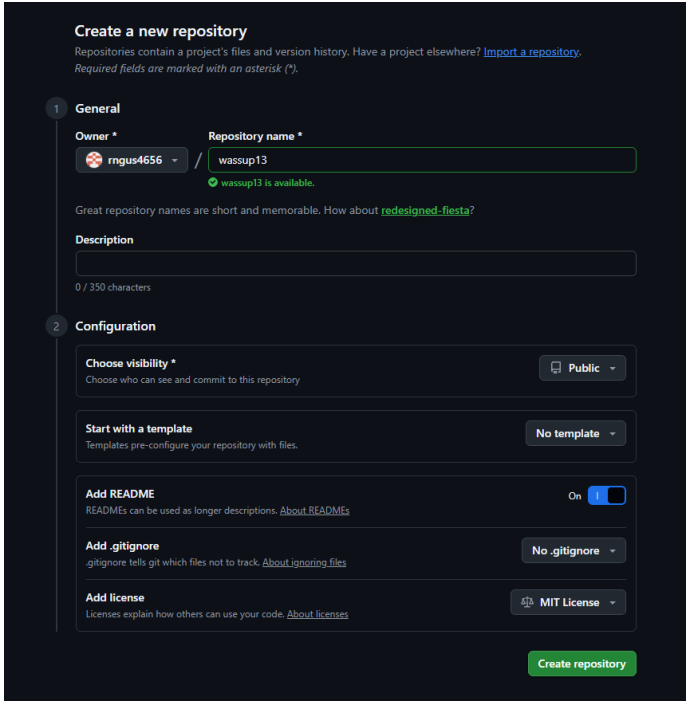




GitHub - Repository 생성

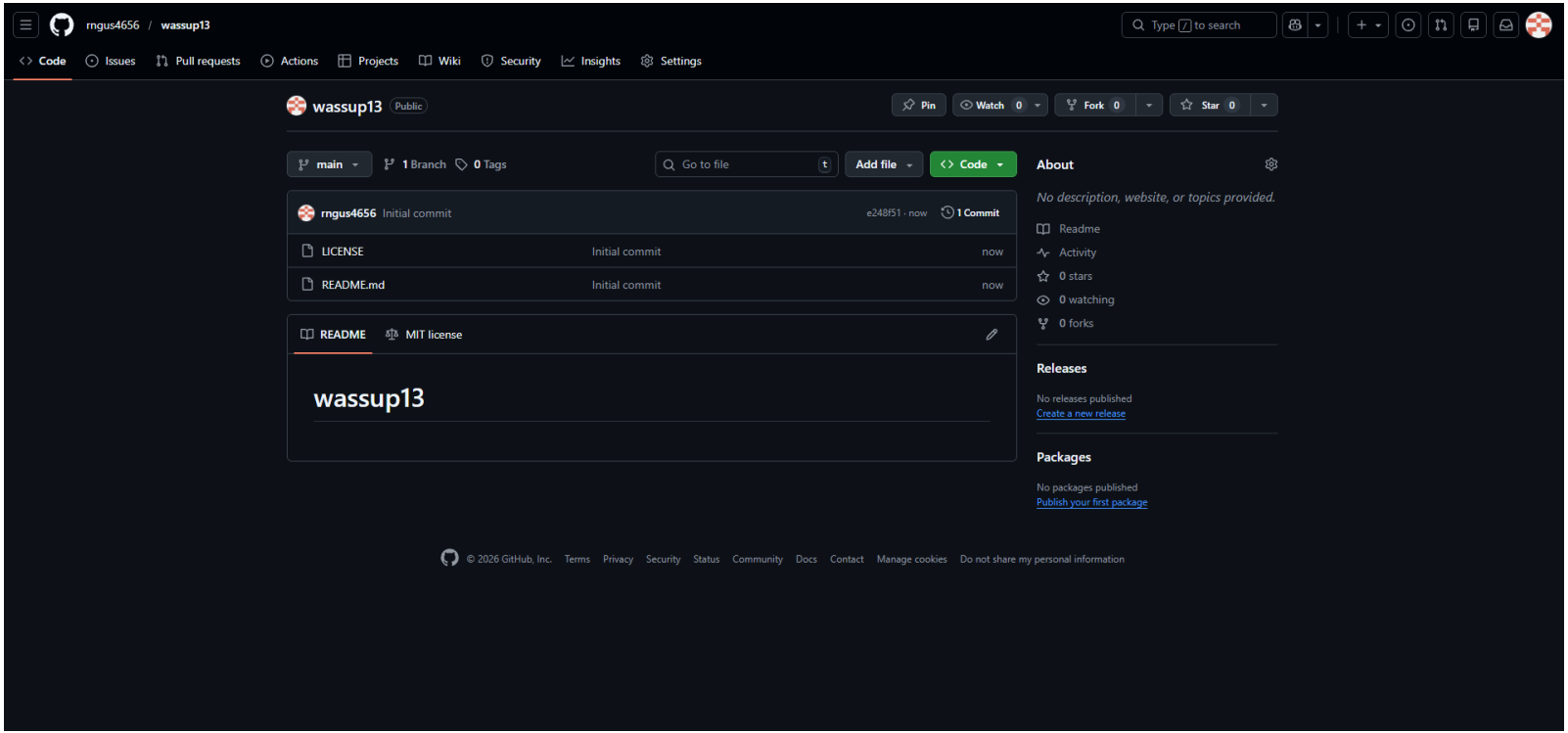
Owner 클릭 - 본인 계정 선택 - Repository name: wassup13 - Add README On 설정 - Add license: MIT License 설정
Create repository 클릭





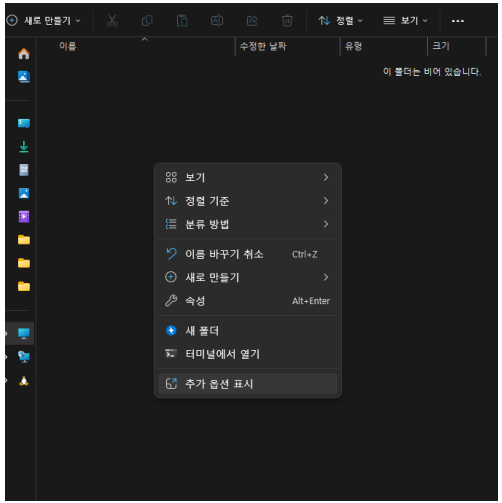
GitHub - Repository 확인

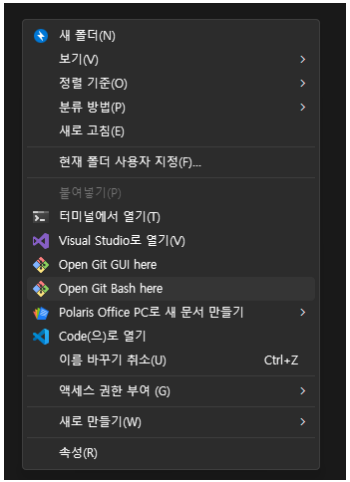
Repository 정상 생성 확인

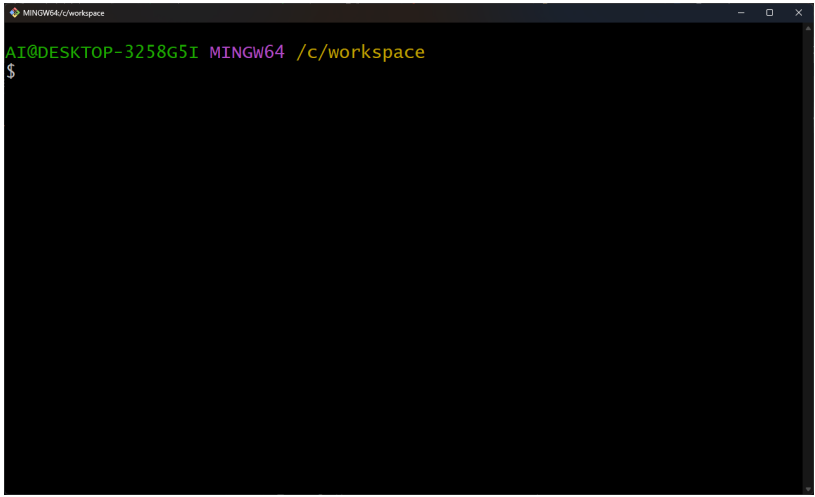


Git - Git bash 실행

C:\workspace 폴더 이동 - 폴더 빈 공간 우 클릭 - 추가 옵션 표시 클릭 - Open Git Bash here 실행







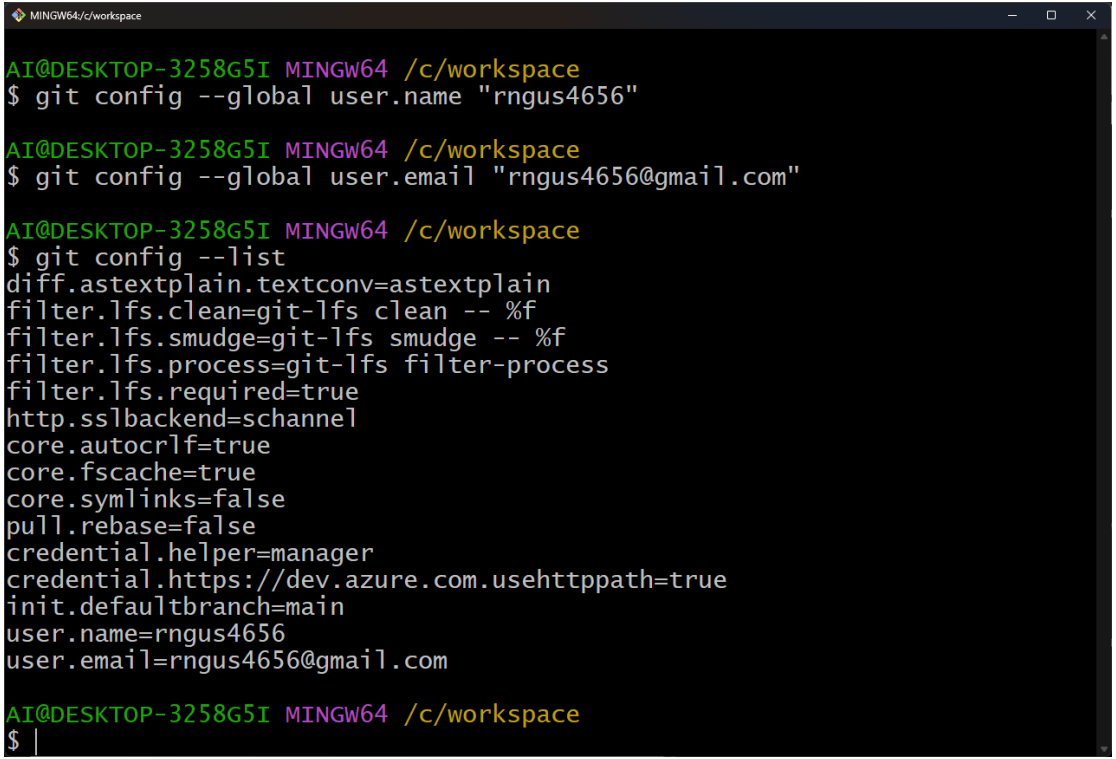
Git - config 설정

1. 터미널 입력창에 본인 닉네임 입력
git config --global user.name “User name”

2. 입력 후 다시 본인 이메일 입력
git config --global user.email “example@email.com”

3. 입력 후 설정 확인
git config --list

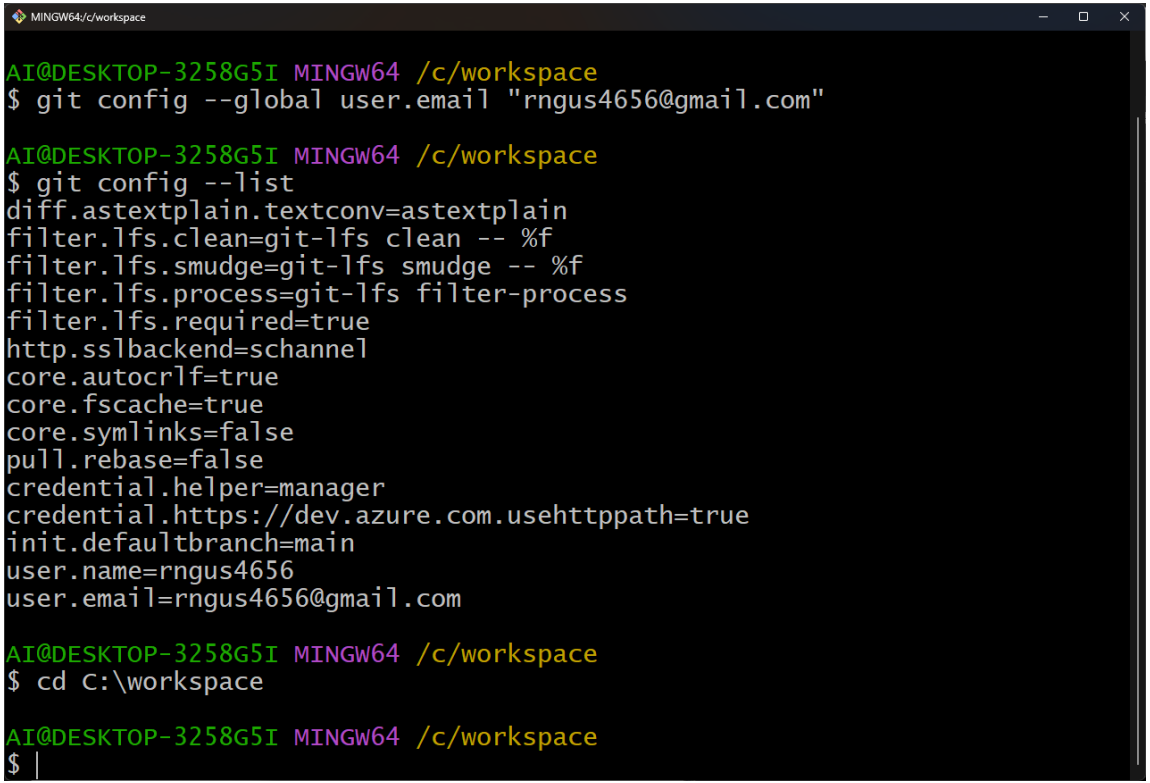
닉네임, 이메일을 잘못 입력했다면?
git config --global --unset user.name
혹은
git config --global --unset user.email
을 입력하여 삭제 후 다시 입력할 수 있다.
(꼭 삭제하지 않고 추가로 입력해도 된다.)



Git - 작업경로 재확인(필수)

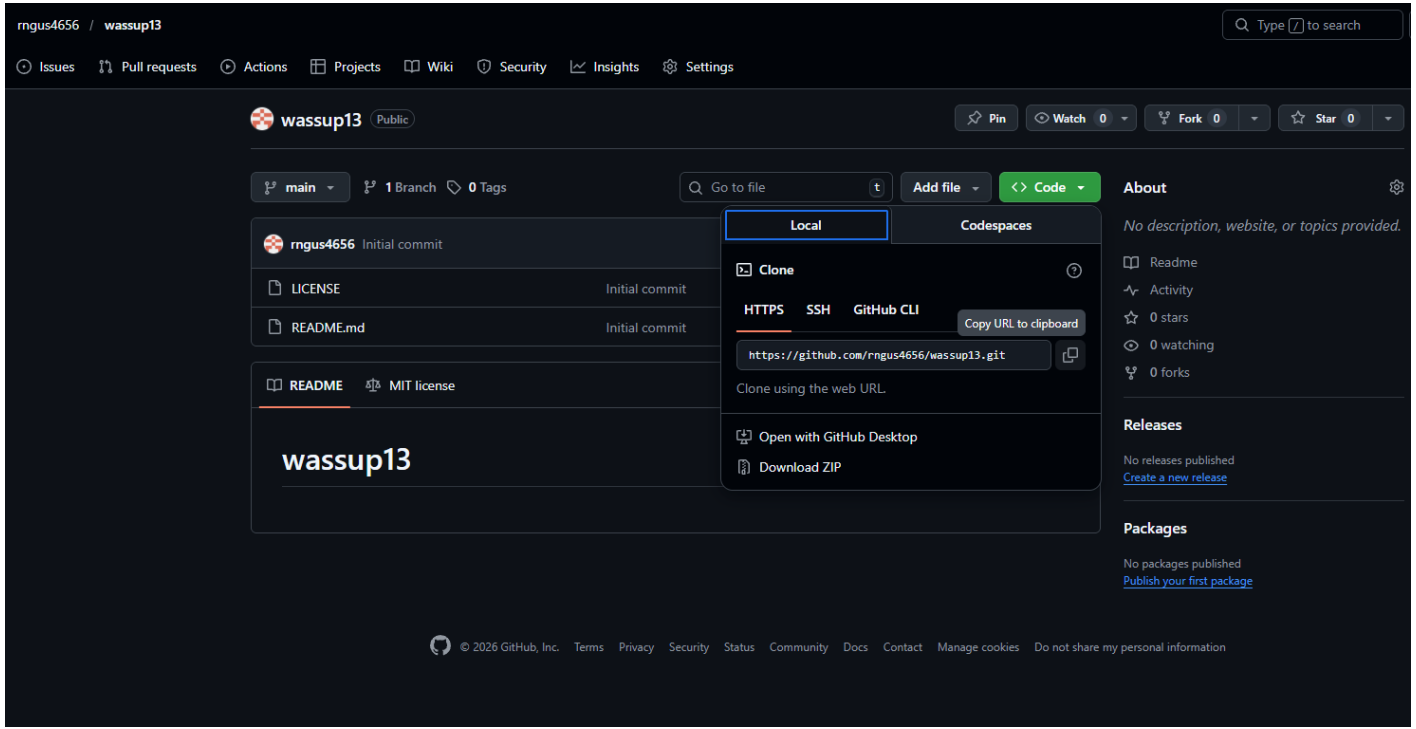
폴더 경로가 C:\workspace 로 되어있는지 재확인
만약, 경로가 다르게 되어 있을 경우
cd C:\workspace
입력창에 입력 후 엔터

이동이 안 될 경우
C드라이브 안에 workspace 폴더 이름이 제대로 생성되어 있는지 확인



GitHub - Repository 주소 복사

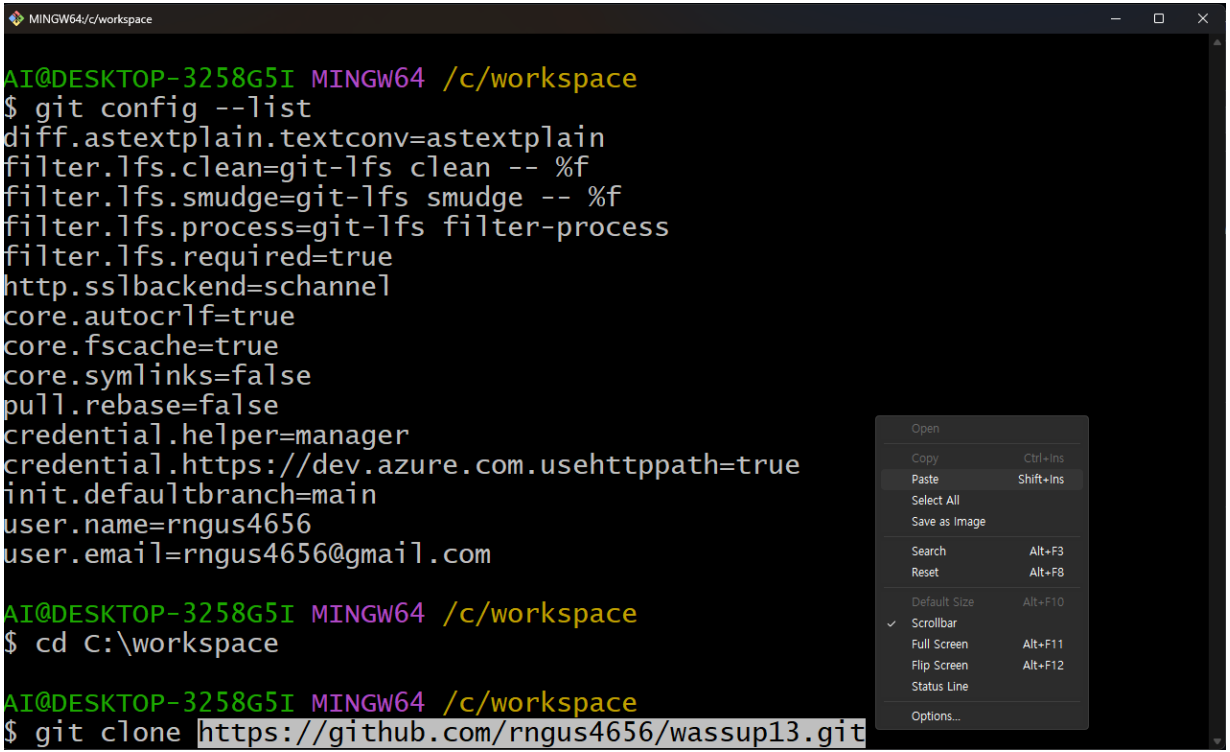
GitHub Repository의 녹색 버튼 Code 클릭 - 주소 복사 클릭



Git - Git clone

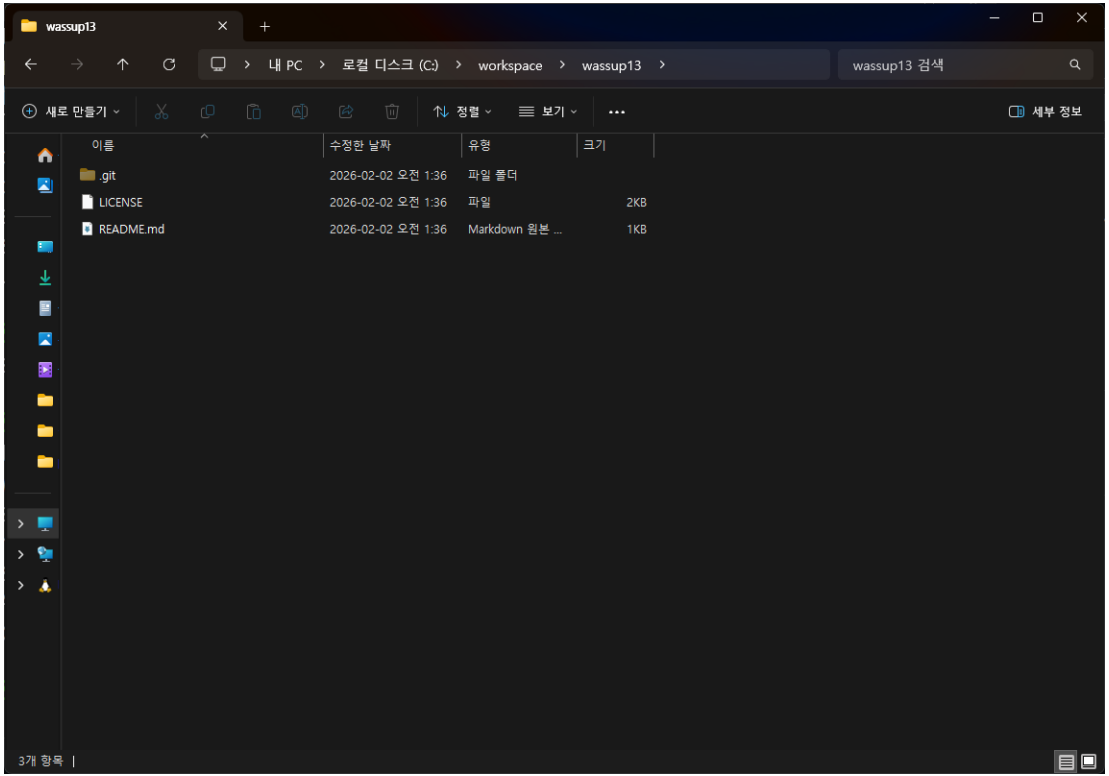
주의!! git bash는 ctrl+v 입력이 안 된다.
안전하게 우클릭 후 Paste를 누르자.

Git bash 터미널 입력창에
git clone 우클릭 후 Paste
엔터



Git - Repo 생성 확인

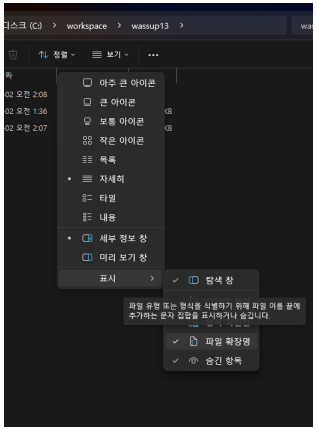
이제 C드라이브에 wassup13 Repo가 성공적으로 Clone 되었다.



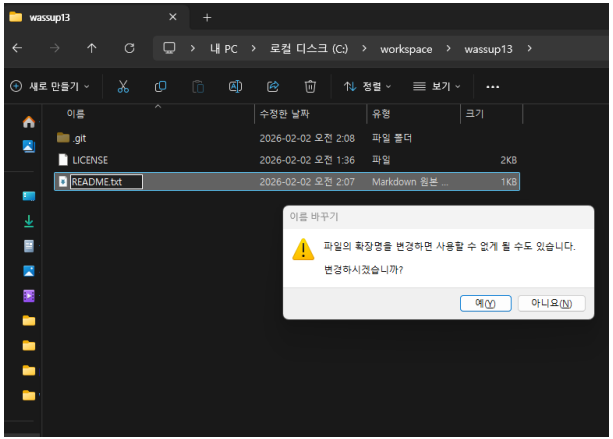
Git - Readme.md 실습

Readme.md 파일 편집 실습

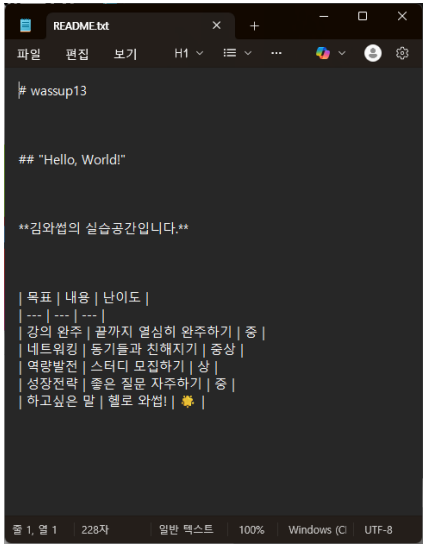
파일 확장명 보이기 체크

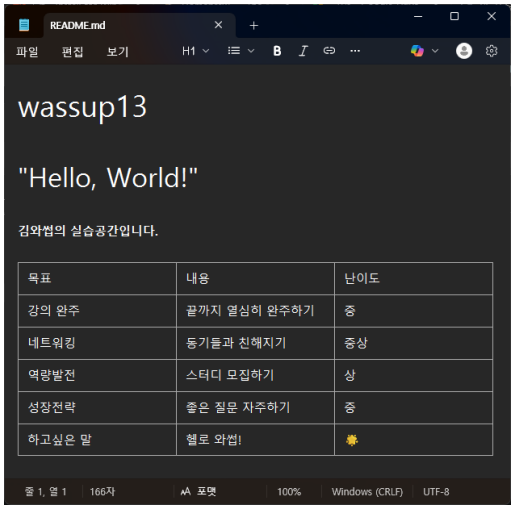


README.md의 확장자를 txt로 변경



마크다운을 활용하여 자유롭게 편집 후 다시 확장자를 .md로 되돌려 놓으면 된다.





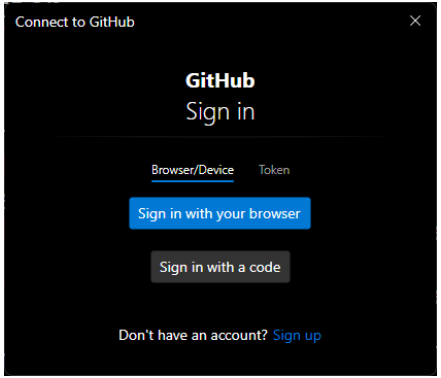
S1 Part 3 Git - Git push

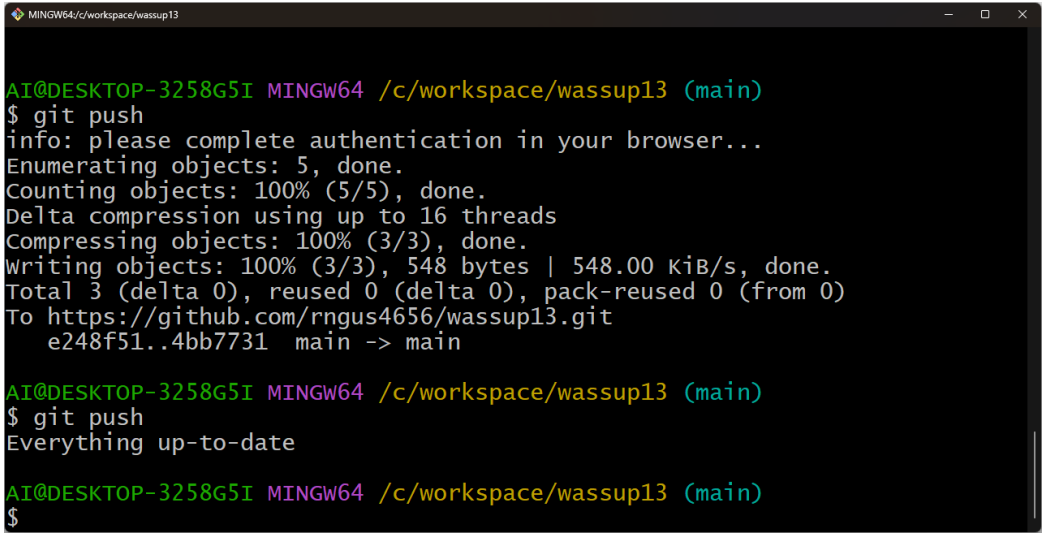
1. C:\workspace\wassup13 폴더 이동 후
우클릭 - Open git bash here
혹은, git bash 터미널 입력창에
cd c/workspace/wassup13
입력 후 엔터
청록색의 (main)이 표시된다면 연동이 잘 된 것.

2. 명령어 순서대로 입력
git add . ➔ add 뒤에 공백 넣고 .입력
git status ➔(상태 확인, 선택사항)
git commit -m '0213_test1' ➔(메시지 내용은 자유)
git push
순서대로 입력

2.5 첫 git push시 Connect to GitHub 요청
Sign in with your browser로 쉽게 연결 가능

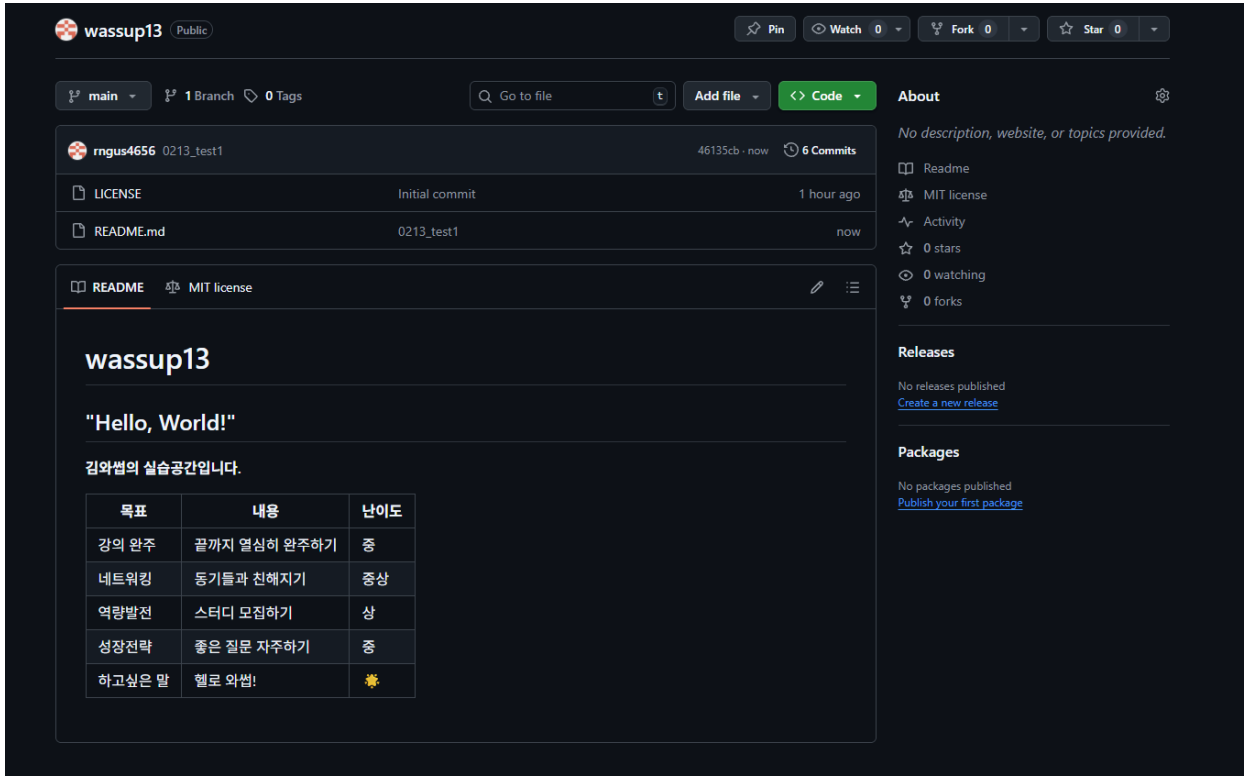
3. git bash로 돌아와서
git push
입력 후 엔터





GitHub - 업로드 확인

READMD.md 업로드 완료



회고 및 Q&A

감사합니다.

Appendix

개발지식에 관심이 있다면?

파이썬 및 개발지식

심화 학습

점프 투 파이썬 (파이썬 기초)
https://wikidocs.net/book/1

프로그래머스 파이썬 코딩테스트 lv1~2 목표
https://school.programmers.co.kr/learn/challenges?order=recent&levels=1%2C0&page=1&languages=python3

동빈나 (파이썬 기초, 머신러닝, 딥러닝 논문 등 지식 공유 유튜브)
https://www.youtube.com/@dongbinna

3Blue1Brown (쉽게 설명해주는 어려운 지식들)
https://www.youtube.com/@3blue1brown
https://www.youtube.com/@3Blue1BrownKR 한국어 채널

‘Attention is All you need’ 트랜스포머의 구조를 처음 발표한 논문
https://arxiv.org/abs/1706.03762 논문 자체를 보는 것 보다 관련 학습 자료를 참고하는 것이 이해하기 편하다.