DK

Git & GitHub

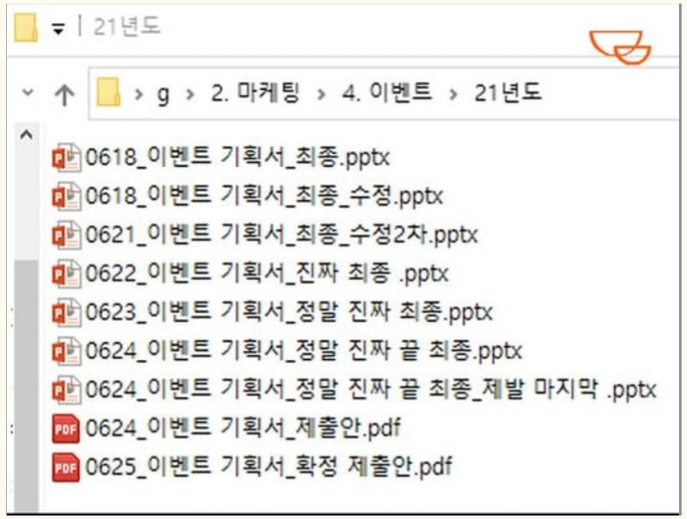
Contents

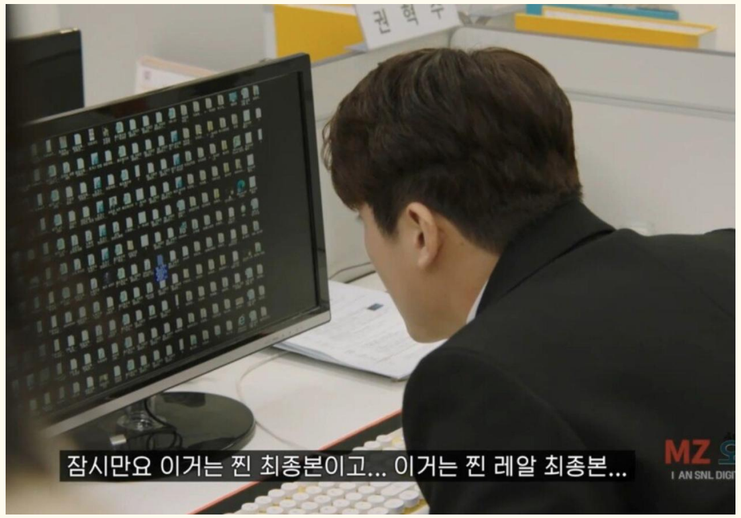
01. GIT 이란?

- **형상 관리 시스템 / 버전 관리 시스템**
  - 코드나 문서 작업의 변경 이력을 저장

01. GIT 이란?

- **형상 관리 시스템 / 버전 관리 시스템**
  - 코드나 문서 작업의 변경 이력을 저장





01. GIT 이란?

- **형상 관리 시스템 / 버전 관리 시스템**
  - 코드나 문서 작업의 변경 이력을 저장

COMMENT	DATE
덧셈 기능 추가	14 HOURS AGO
뺄셈 기능 추가	12 HOURS AGO
곱셈 기능 추가	10 HOURS AGO
나눗셈 죄종 버전	9 HOURS AGO
계산기 최종 버전	6 HOURS AGO
계산기 레알 최종 버전	3 HOURS AGO
계산기 레알 최종 찐 버전	2 HOURS AGO
계산기 레알 최종최종 찐 버전	2 HOUR AGO
계산기 레알 최종최종 찐찐 버전	1 HOUR AGO

01-1. 내 작업을 기록하는 도구

- **작업 기록 도구**
  - 수정한 코드, 내용의 기록과 이력 추적 가능
  - 실수했을 때 되돌리기 가능
  - 작업 흐름을 시간순으로 보관

01-2. 협업을 가능하게 하는 도구

- **협업 도구**
  - 여러 사람이 같은 파일을 고칠 수 있음
  - 누가, 언제, 무엇을 수정했는지 확인 가능
  - 충돌이 나도 해결 및 정리 하는 방식 제공

COMMENT	DATE	WHO
덧셈 기능 추가	14 HOURS AGO	PERSON A
빨셈 기능 추가	12 HOURS AGO	PERSON B
곱셈 최종 버전	9 HOURS AGO	PERSON C
나눗셈 최종 버전	6 HOURS AGO	PERSON C
계산기 최종 버전	6 HOURS AGO	PERSON B
빨셈 버그 개선	4 HOURS AGO	PERSON C
계산기 레알 최종 버전	3 HOURS AGO	PERSON C
계산기 레알 최종최종 찐	3 HOUR AGO	PERSON C
계산기 레알 최종최종 찐버전	3 HOUR AGO	PERSON C
계산기 레알 최종최종 찐찐 버전	1 HOUR AGO	PERSON C

01-3. 프로그램 버전을 관리하는 도구

- **버전 관리 도구**
  - 기능 추가, 버그 수정 등 변경 단계를 버전으로 관리
  - 릴리즈 버전 이력도 관리 가능
  - 이전 버전으로 돌아갈 수 있음

01-4. Git 외에 어떤 도구가 있었을까?

- **Git 외 사용할 수 있는 도구**
  - **CVS, SVN(Subversion)**
    - 중앙 서버 기반
  - **Mercurial**
    - Git과 유사한 분산 방식

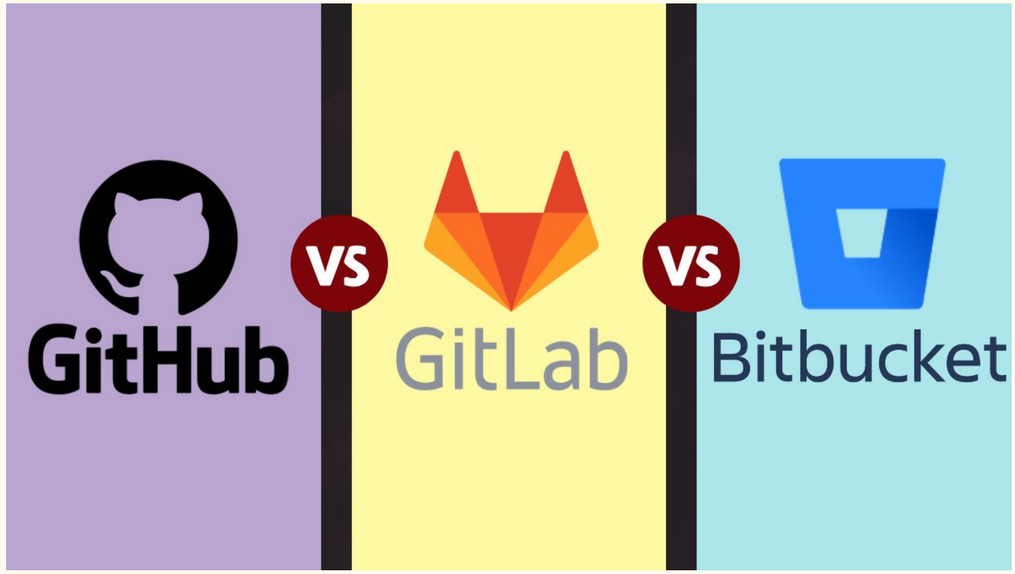
01-5. Git vs GitHub

- **Git**
  - 버전 관리 도구
  - 기록하고 복구 기능 지원

- **GitHub**
  - Git 내용 및 코드를 저장 및 공유 (클라우드 서비스)
  - 팀원과 공유하는 Git 기반 플랫폼

01-6. Git의 장점

- **Git 의 장점**
  - GitHub, GitLab, Bitbucket 등 플랫폼 지원
  - 오픈소스와 기업들이 널리 사용
  - 분산형 구조 + 빠른 속도 + 신뢰성

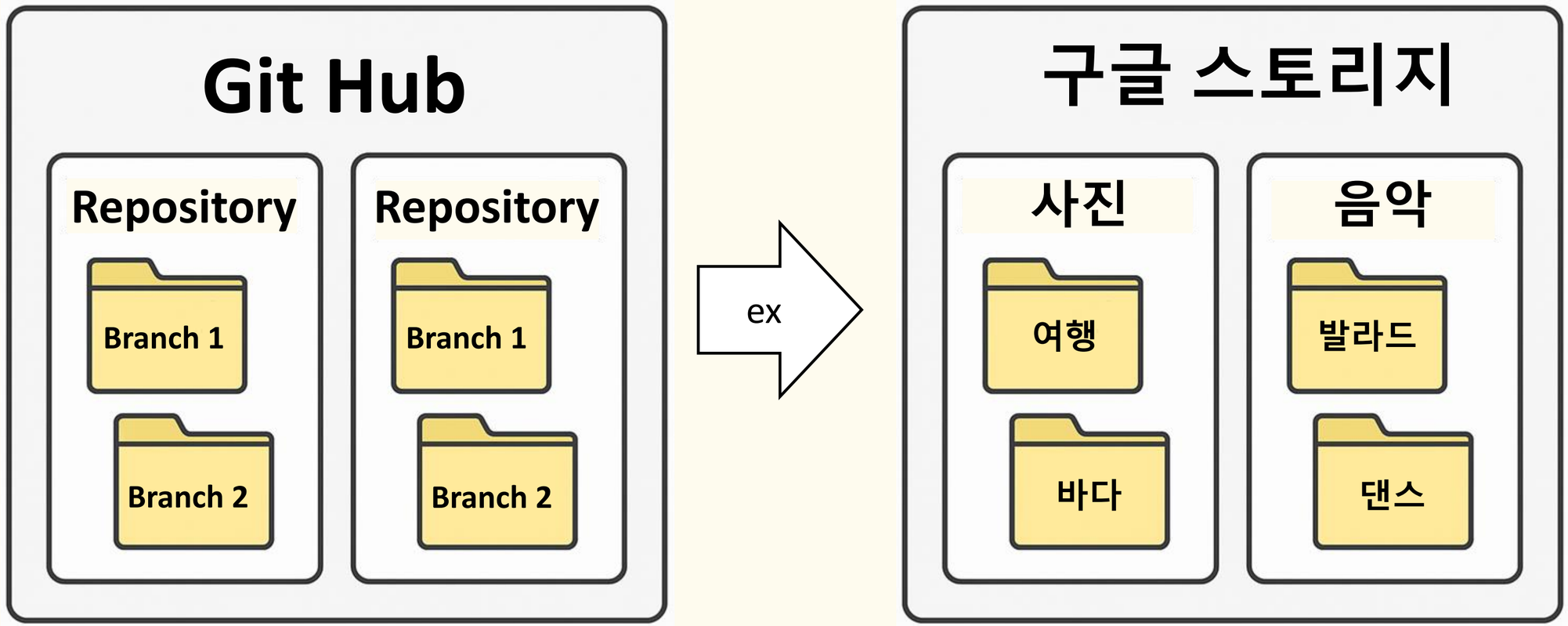


01-7. Git 정리

- **Git**
  - 개발자를 위한 기록 도구, 협업 도구, 버전 관리 시스템
  - 많은 개발자, 회사들이 사용하는 도구

02. Git Hub 구조

- Git Hub > Repository > Branch



02. Git 구조

- **Git 작업 흐름**

  1. Working Dirctory

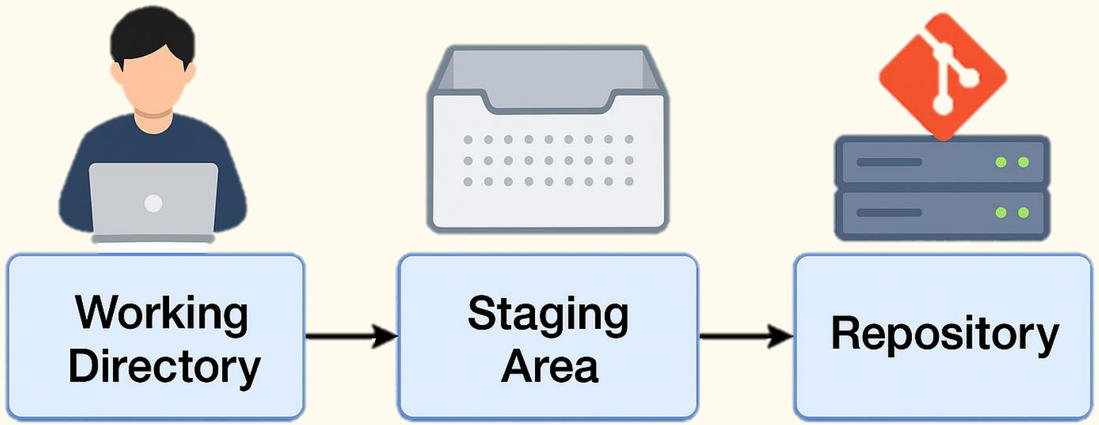
  → 내 컴퓨터 (작업 공간)

  2. Staging Area

  → 내 컴퓨터 (임시 공간)

  3. Repository

  → Git hub



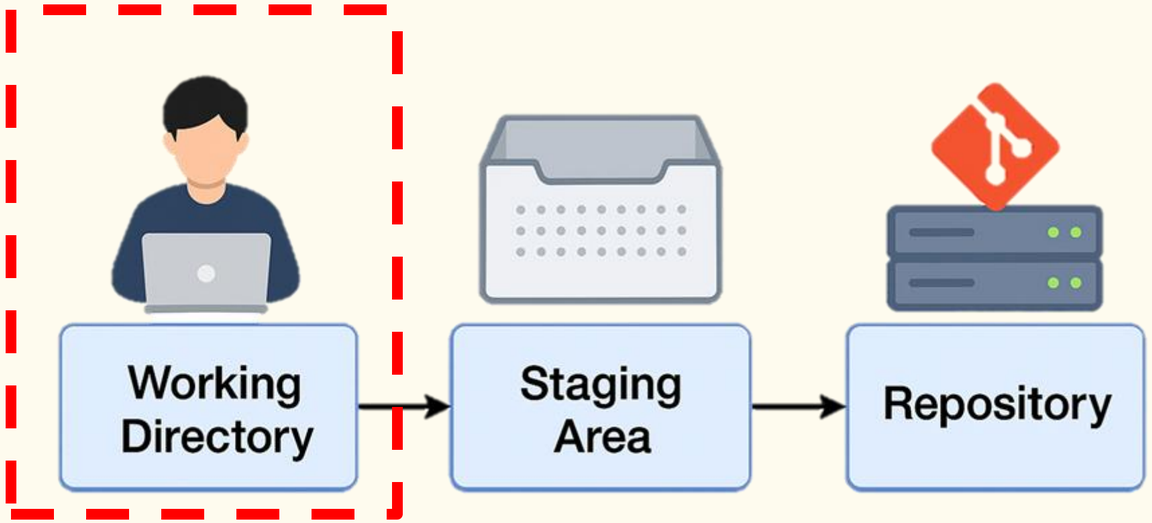
02-1. Git 기본 명령어

- init
- add
- status
- diff
- commit
- push
- log
- reset

02-3. 기본 명령어(1)

- **init**
  - 내가 만든 코드를 git에 등록

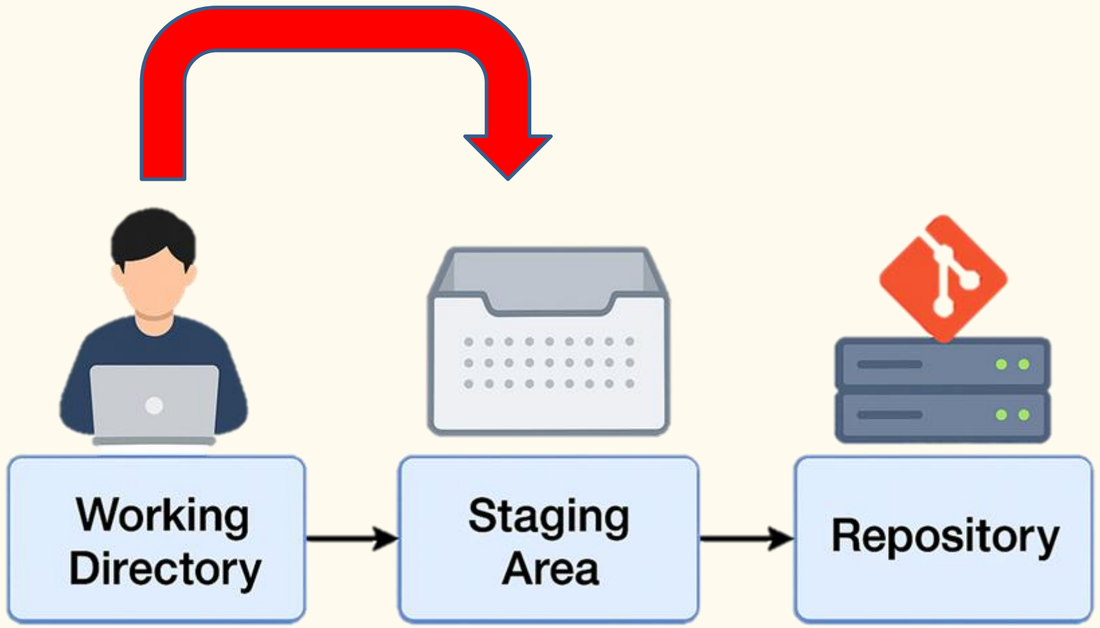
- 명령어
  - git init



02-3. 기본 명령어(2)

- **add**
  - 내가 추가 및 수정한 코드를 스테이징 영역에 반영

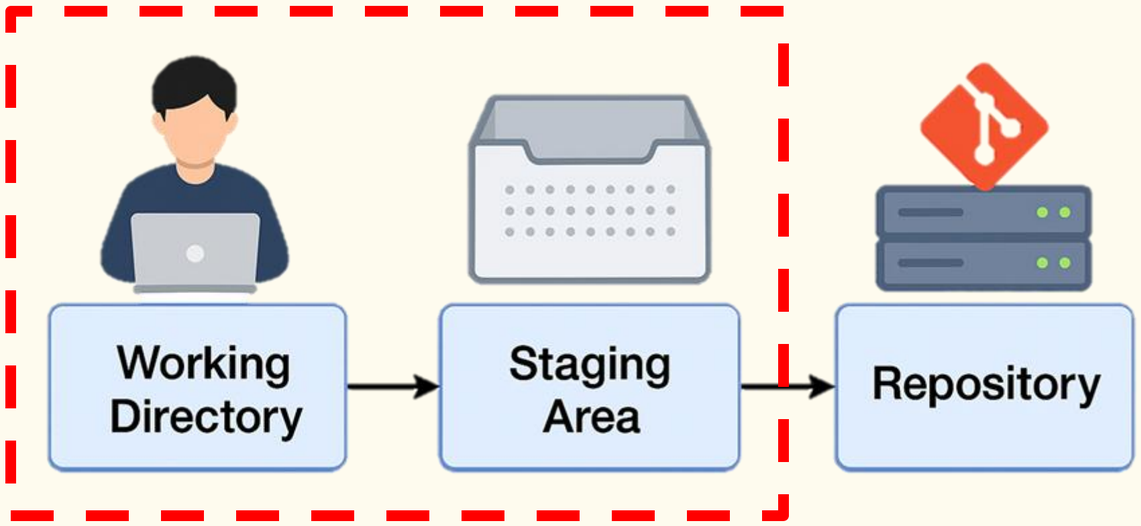
- 명령어
  - git add [파일명]
  - git add .



02-3. 기본 명령어(3)

- **status**
  - 내가 추가 및 수정한 코드 파일 확인
  - 내가 어떤걸 add 하고 안했는지 확인

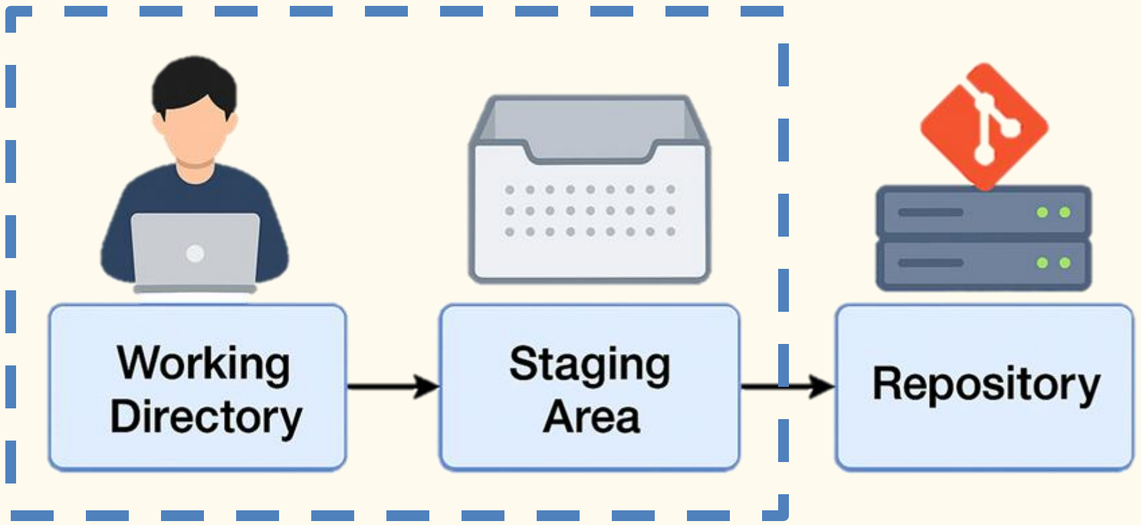
- 명령어
  - git status



02-3. 기본 명령어(4)

- **diff**
  - 내가 변경한 코드의 변경분을 확인

- 명령어
  - git diff [파일명]

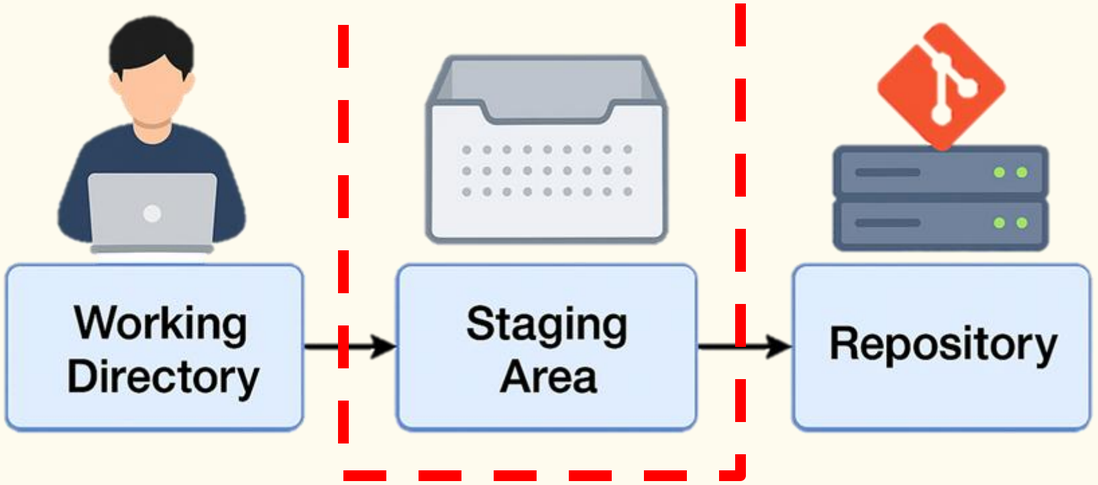


02-3. 기본 명령어(5)

- **commit**
  - 내가 변경한 코드를 설명하는 메시지 추가

- 명령어
  - git commit -m “메세지”

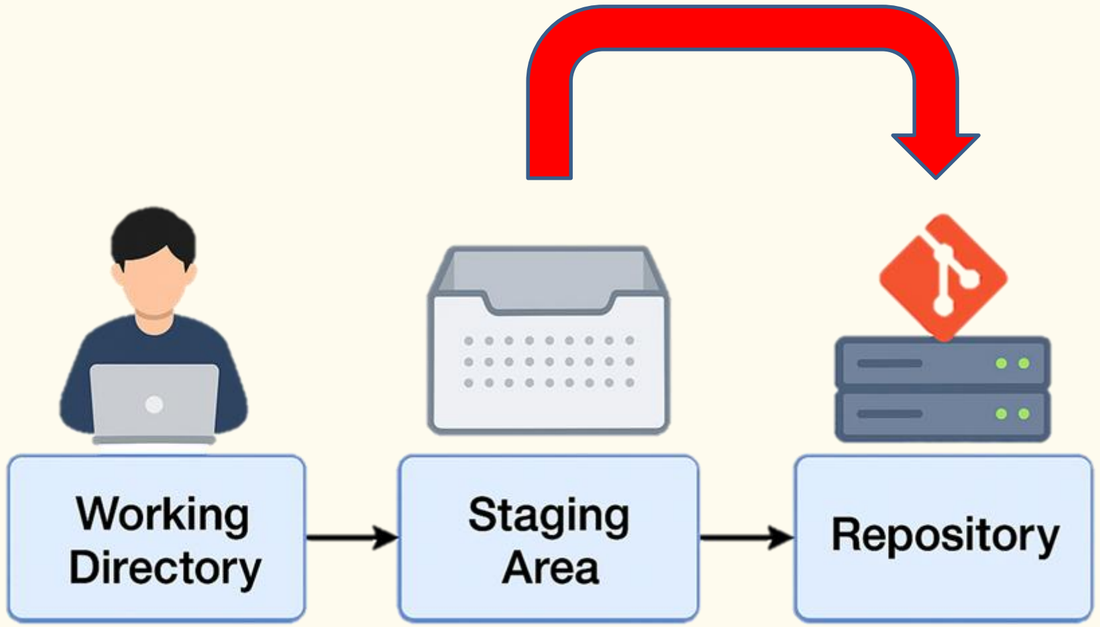
COMMENT	DATE
덧셈 기능 추가	14 HOURS AGO
뺄셈 기능 추가	12 HOURS AGO
곱셈 기능 추가	10 HOURS AGO
나눗셈 죄종 버전	9 HOURS AGO
계산기 최종 버전	6 HOURS AGO
계산기 레알 최종 버전	3 HOURS AGO
계산기 레알 최종 찐 버전	2 HOURS AGO
계산기 레알 최종최종 찐 버전	2 HOUR AGO
계산기 레알 최종최종 찐찐 버전	1 HOUR AGO



02-3. 기본 명령어(6)

- **push**
  - 내가 변경한 코드를 git-hub 에 실제 반영

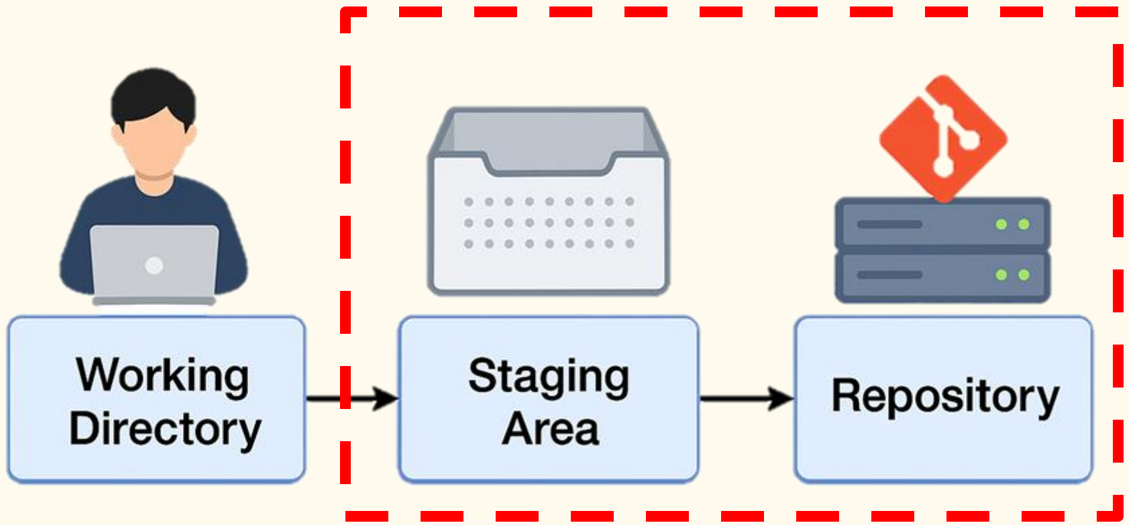
- 명령어
  - git push
  - git push origin [레퍼지토리명]



02-3. 기본 명령어(7)

- log
  - 내가 commit한 모든 기록을 확인

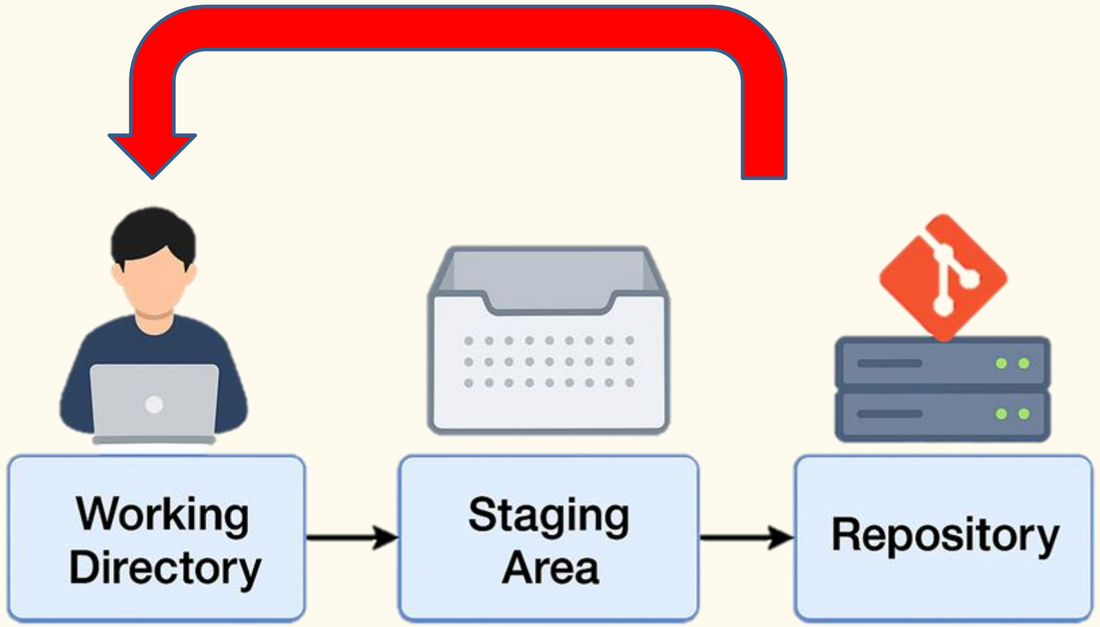
- 명령어
  - git log



02-3. 기본 명령어(8)

- reset
  - 내가 commit 한 기록 복원하기

- 명령어
  - git reset [commit_id]



03. Git 설치 및 Github 계정 생성

- Git : https://git-scm.com
- Git Hub : https://github.com
- 가이드 문서 참고 (채팅방 공유)

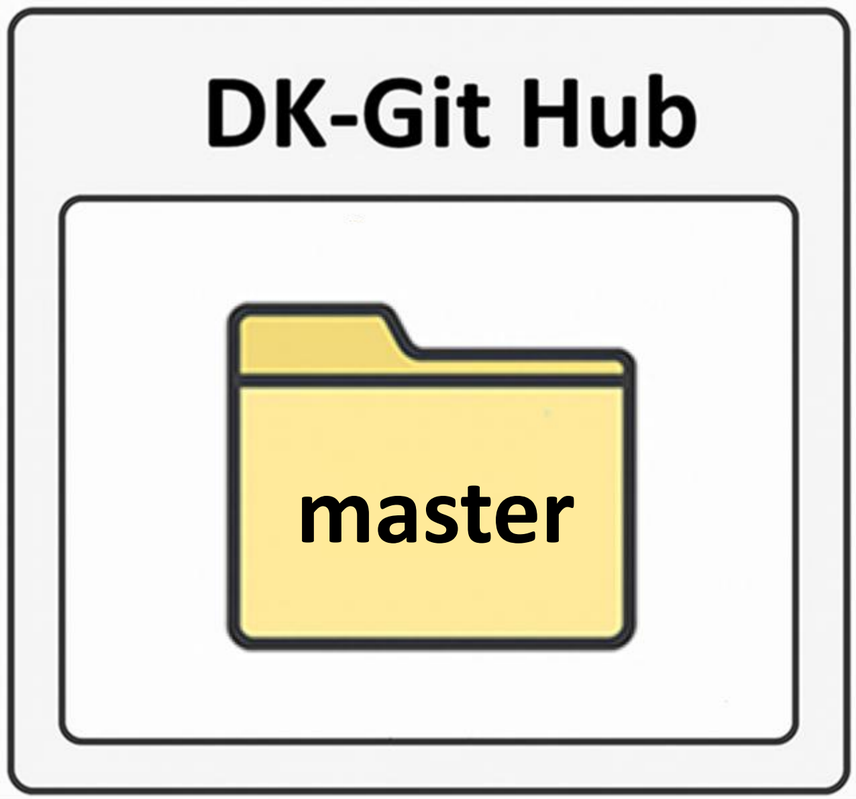
04. Git 실습

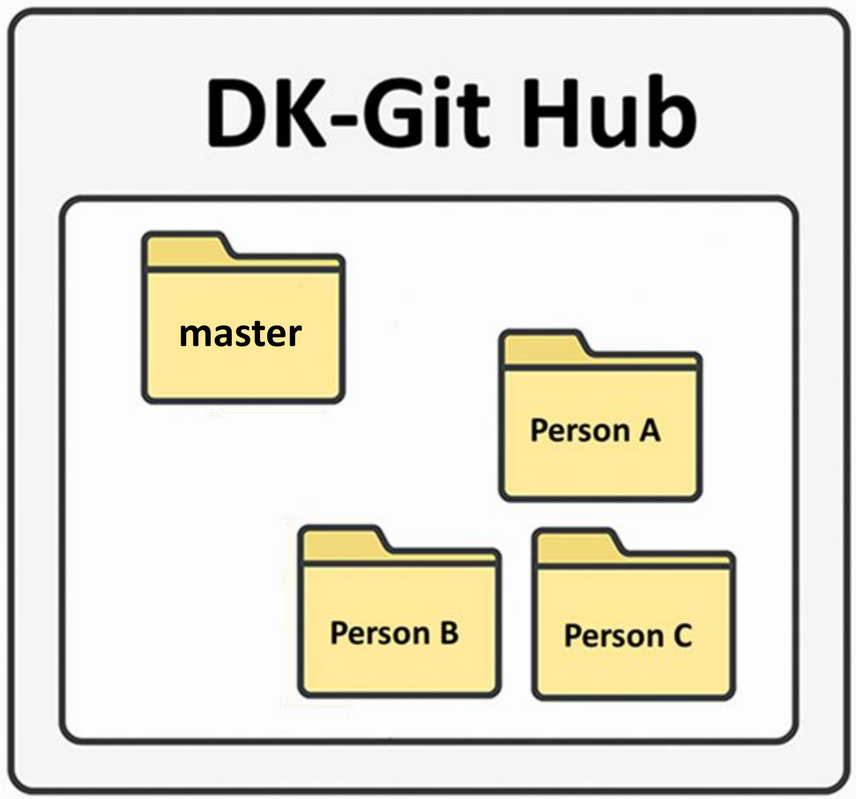
- 가이드 문서 참고 (채팅방 공유)

팀 프로젝트를 위한 Git

05. 팀 프로젝트 - 브랜치

- 개인 프로젝트 vs 팀 프로젝트





05. 팀 프로젝트 - 브랜치

- **메인 브랜치 (main/master)**

  프로젝트의 대표 코드 (안정 버전)

- **개별 브랜치**

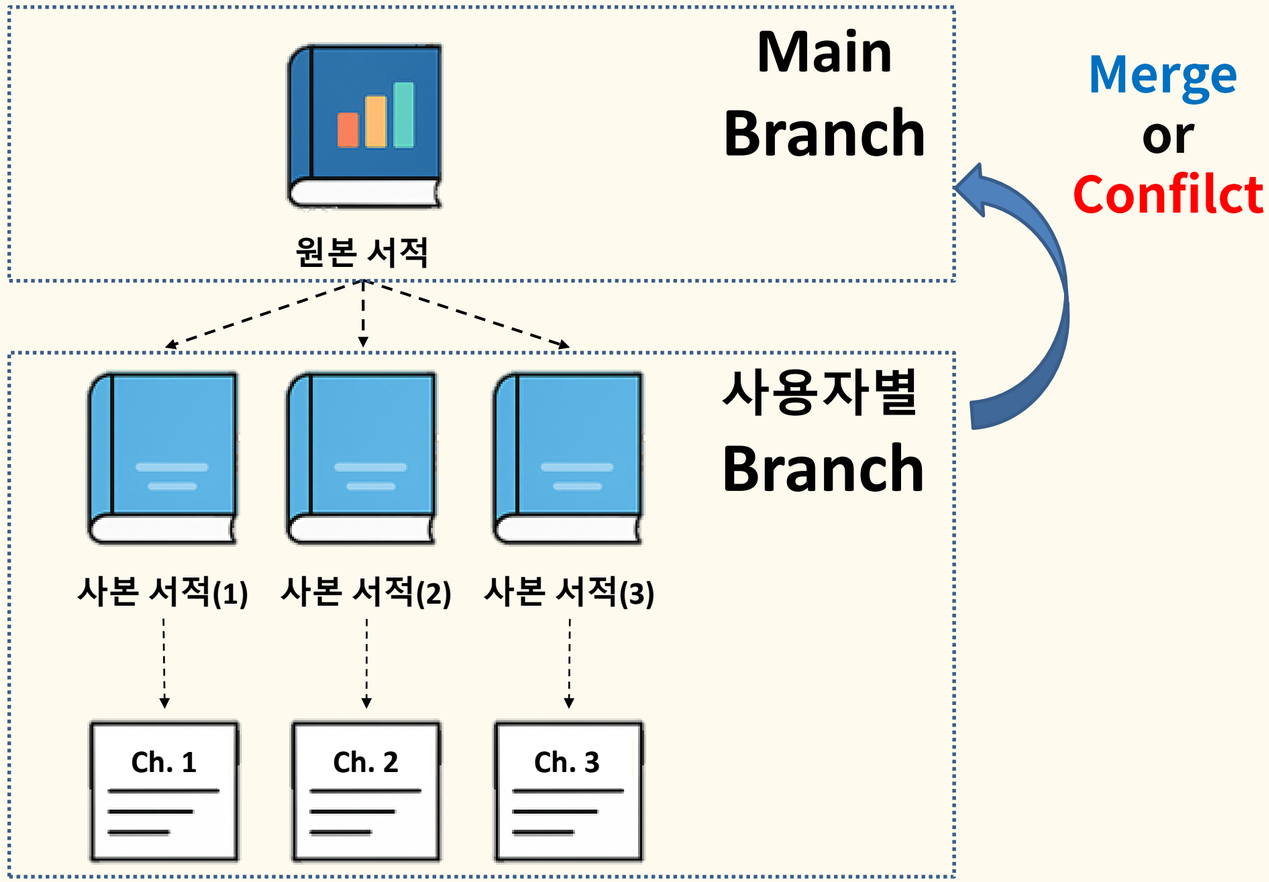
  메인 브랜치를 복사하여, 독립적 작업

- **병합 (Merge)**

  개별 브랜치 작업 → 메인 브랜치 반영

- **충돌 (Conflict)**

  병합 과정에서 충돌이 난 경우



05. 팀 프로젝트를 위한 GIT 전략

- **효율적인 협업을 위한 전략 필요**
  1. 프로젝트에 필요 기능 확인
  2. 우선순위 선정
  3. 필요 기능 파일 구성
  4. 파일별로 역할 분배
  5. 개별 브랜치 생성 및 프로젝트 진행

05. 팀 프로젝트를 위한 GIT 전략

순위	필요 기능	파일명	팀원
1	덧셈	add.py	A
1	뺄셈	subtract.py	B
2	나눗셈	divide.py	C
2	곱셈	multiply.py	D

3	main	main.py	E

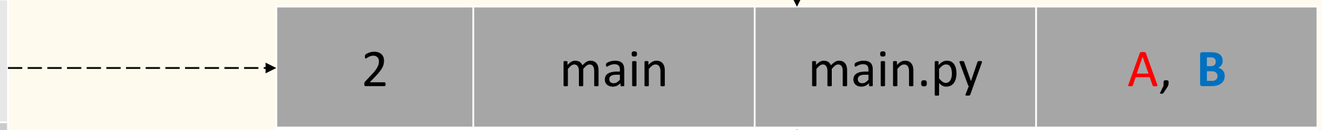
Merge

GitHub

05. 팀 프로젝트를 위한 GIT 전략

- **효율적인 협업을 위한 전략 필요**

순위	필요 기능	파일명	팀원
1	덧셈	add.py	A
1	뺄셈	subtract.py	B
1	나눗셈	divide.py	A
1	곱셈	multiply.py	B



05. 팀 프로젝트를 위한 GIT 전략 (Conflict)

```python
from add import add

def calculator():  1 usage
    a = float(input("1st number: "))
    b = float(input("2nd number: "))
    op = input("연산자 (+): ")
    operations = {
        '+': add,
    }
    if op in operations:
        try:
            result = operations[op](a, b)
            print("result:", result)
        except Exception as e:
            print("오류:", e)
    else:
        print("Invalid operator.")

if __name__ == "__main__":
    calculator()
```

A가 작성한

main.py

+

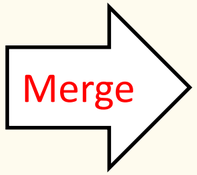
```python
from subtract import subtract

def calculator():  1 usage
    a = float(input("1st number: "))
    b = float(input("2nd number: "))
    op = input("연산자 (-): ")
    operations = {
        '-': subtract,
    }
    if op in operations:
        try:
            result = operations[op](a, b)
            print("result:", result)
        except Exception as e:
            print("오류:", e)
    else:
        print("Invalid operator.")

if __name__ == "__main__":
    calculator()
```

B가 작성한

main.py




```python
from add import add
from subtract import subtract

def calculator():  1 usage
    a = float(input("1st number: "))
    b = float(input("2nd number: "))
    op = input("연산자 (+, -): ")
    operations = {
        '+': add,
        '-': subtract,
    }
    if op in operations:
        try:
            result = operations[op](a, b)
            print("result:", result)
        except Exception as e:
            print("오류:", e)
    else:
        print("Invalid operator.")

if __name__ == "__main__":
    calculator()
```

기대하는

main.py

05. 팀 프로젝트를 위한 GIT 전략 (Conflict)

- **Conflict (충돌) 이 나는 대표 경우**
  1. 같은 파일, 같은 줄 또는 인접 줄 수정
  2. 한 명은 파일 수정, 다른 한 명은 삭제
  3. 같은 이름으로 파일/폴더 생성
  4. Cherry-pick 중 충돌

Git 충돌은 대부분 동일 파일을 수정하고 병합할 때, 발생

05. 팀 프로젝트를 위한 GIT 전략 (Conflict)

- **Conflict (충돌) 을 해결 하는 방법**

```python
def main():
    print("안녕 나는 DK야")
```

A가 만든 main.py 의 main 함수

```python
def main():
    print("안녕 나는 돼지야")
```

B가 만든 main.py 의 main 함수

```
<<<<<<< HEAD
def main():
    print("안녕 나는 돼지야")
=======
def main():
    print("안녕 나는 돼지야")
>>>>>>> 6a23gasd43jop3213jsdf121gave
```

충돌 후, 내용을 확인 후 수정하여 add → commit → push 진행

05. 팀 프로젝트를 위한 GIT 전략

- **Conflict (충돌) 을 방지하는 방법**
  1. 하나의 파일은 한명이 수정한다.

     → 역할 분배로, 충돌 나는 케이스를 만들지 않는다.
  2. 동일한 파일을 수정하는 경우, 대화를 자주한다.

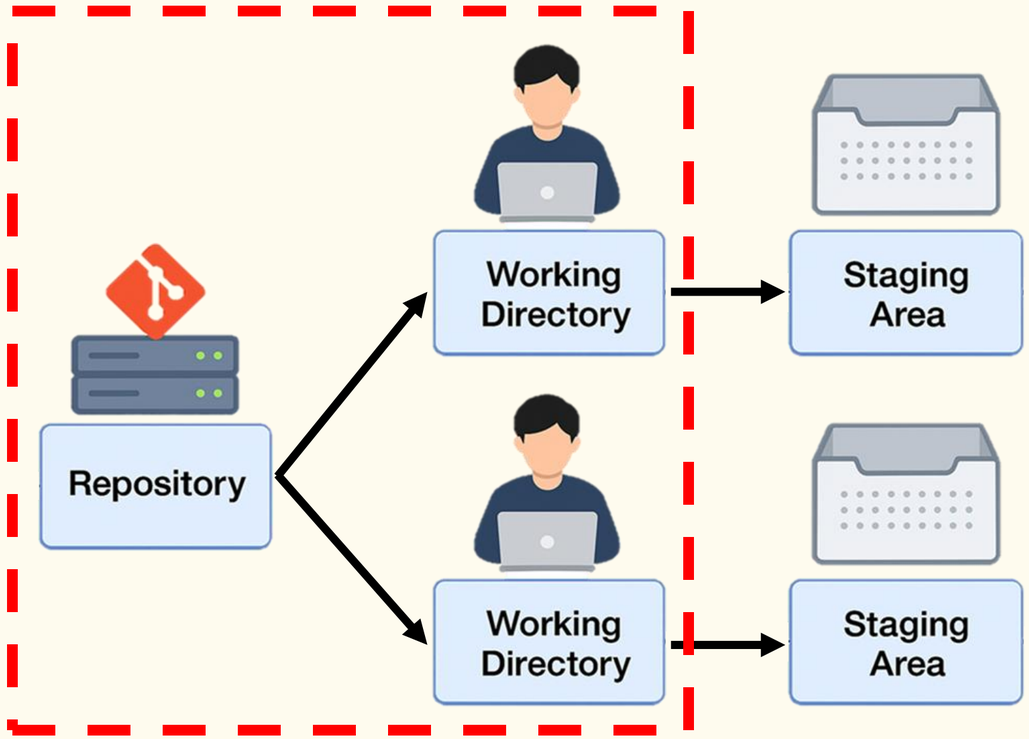
     → “나 이거 바꾼다~”, “바꿨다~”, “바꿈 수고~”

05. 협업을 위한 명령어 모음

- clone
- pull
- branch
- checkout
- stash
- cherry-pick
- merge
- log

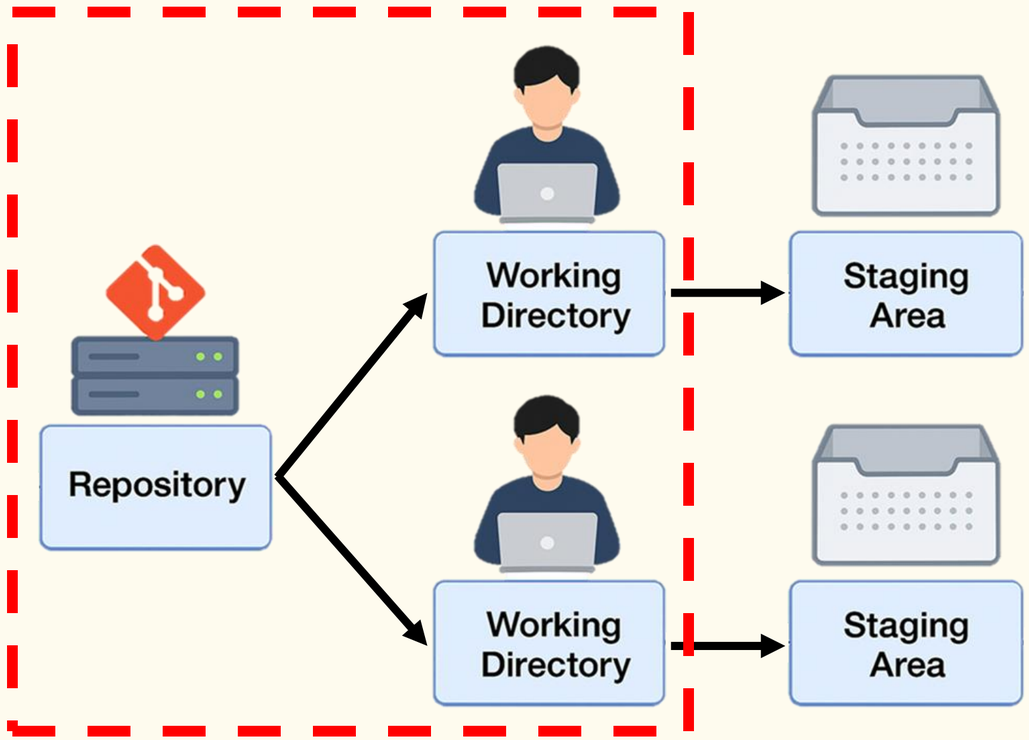
05. 협업을 위한 명령어(1)

- clone
  - Git Hub에서 내 컴퓨터로 복사
- 명령어
  - git clone [url]



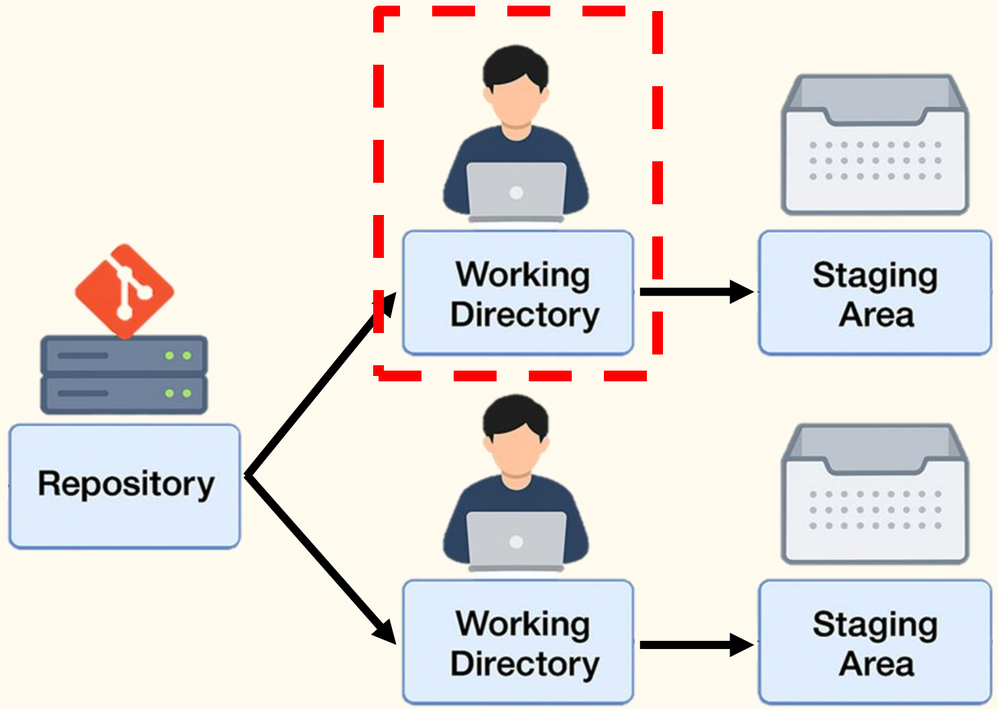
05. 협업을 위한 명령어(2)

- **pull**
  - Git Hub에서 최신 변경사항을 받아와 내 컴퓨터에 반영
- 명령어
  - git pull origin master



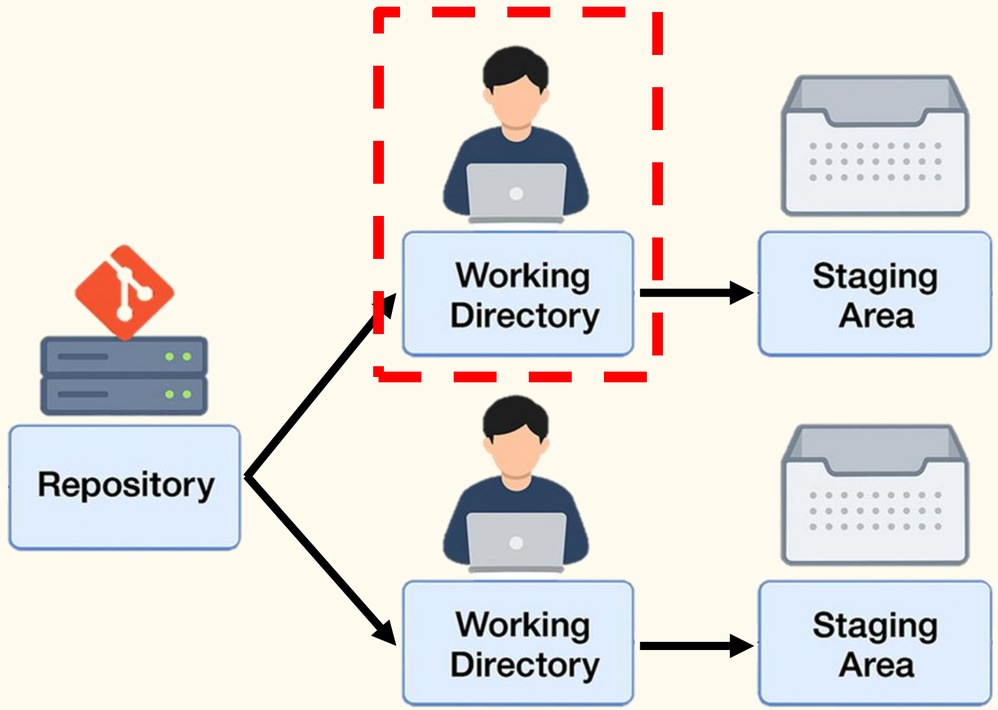
05. 협업을 위한 명령어(3)

- branch
  - 현재 브랜치 목록을 확인 하거나 새 Branch 생성
- 명령어
  - 생성
    - git branch [생성할 이름]
  - 목록 확인
    - git branch
    - git branch –r



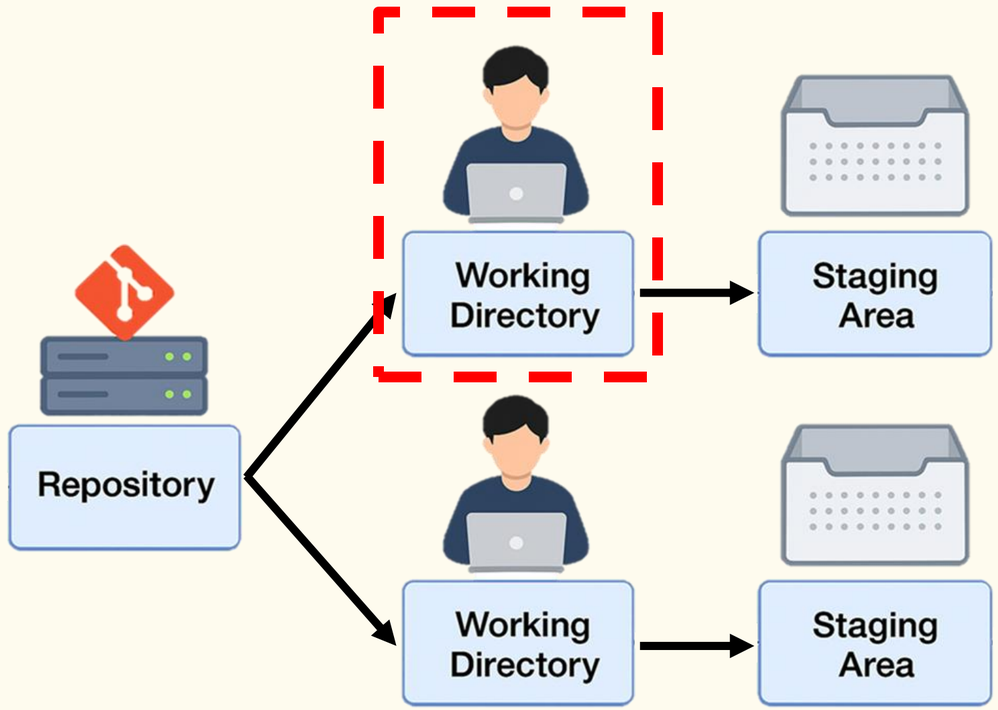
05. 협업을 위한 명령어(4)

- checkout
  - 브랜치를 이동
- 명령어
  - git checkout [branch명]
    - 브랜치를 생성하면서 이동 (-b 옵션)
      - git checkout –b [branch명]



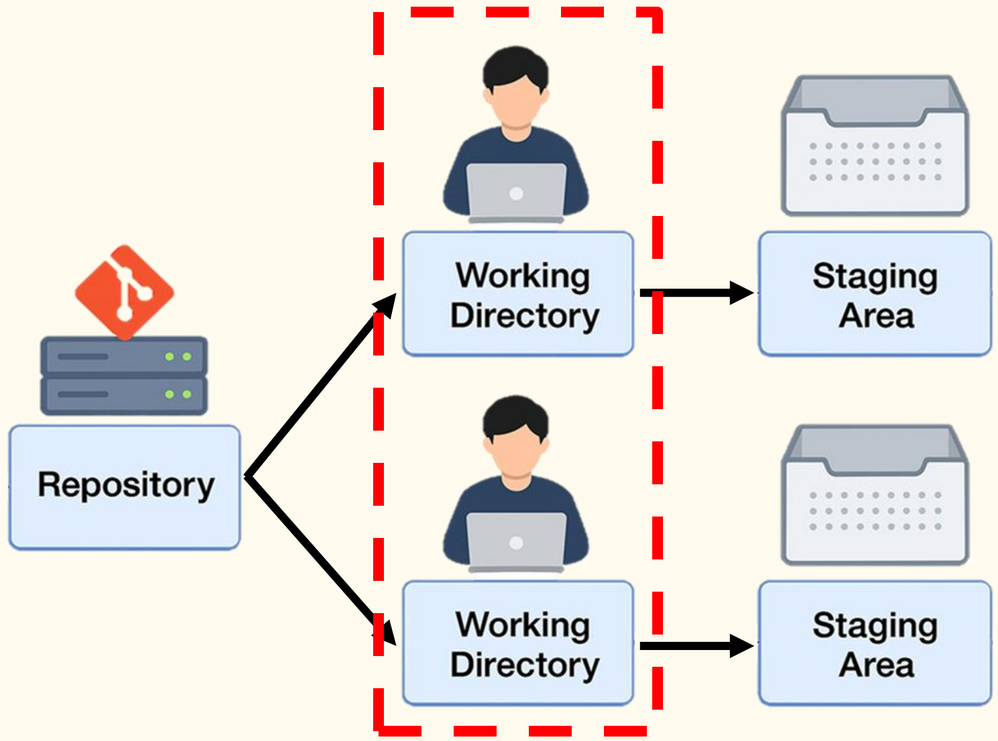
05. 협업을 위한 명령어(5)

- stash
  - 작업 중인 변경사항을 임시로 저장
- 명령어
  - git stash
  - git stash apply (복원)



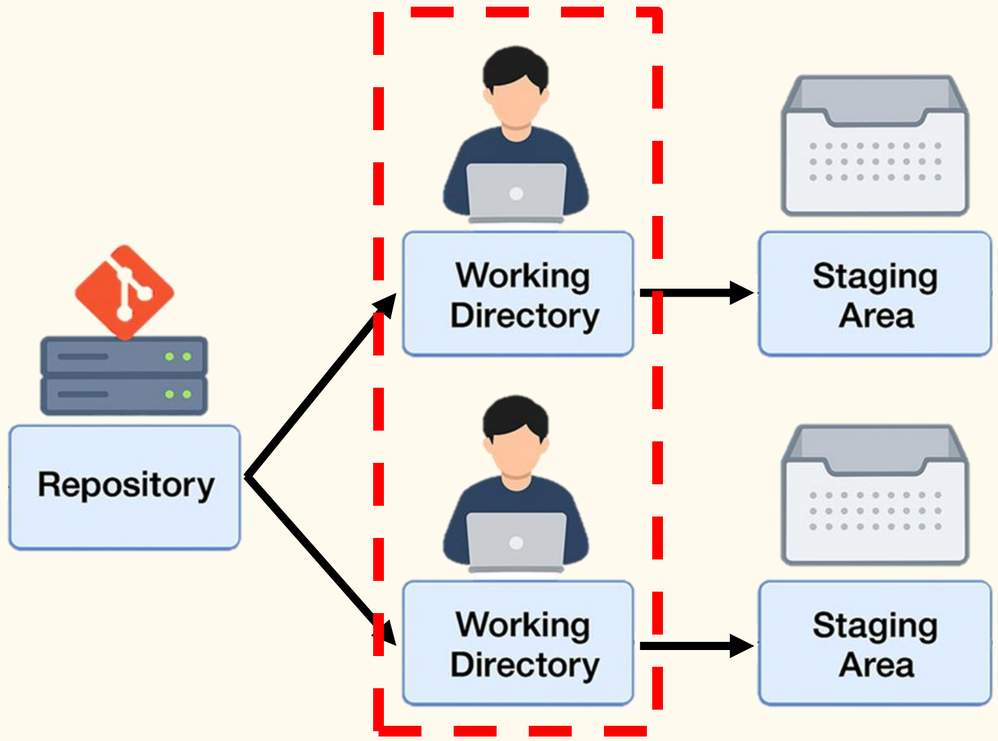
05. 협업을 위한 명령어(6)

- cherry-pick
  - 다른 사람의 commit을 내 작업에 반영
- 명령어
  - git fetch
  - git cherry-pick [commit id]



05. 협업을 위한 명령어(7)

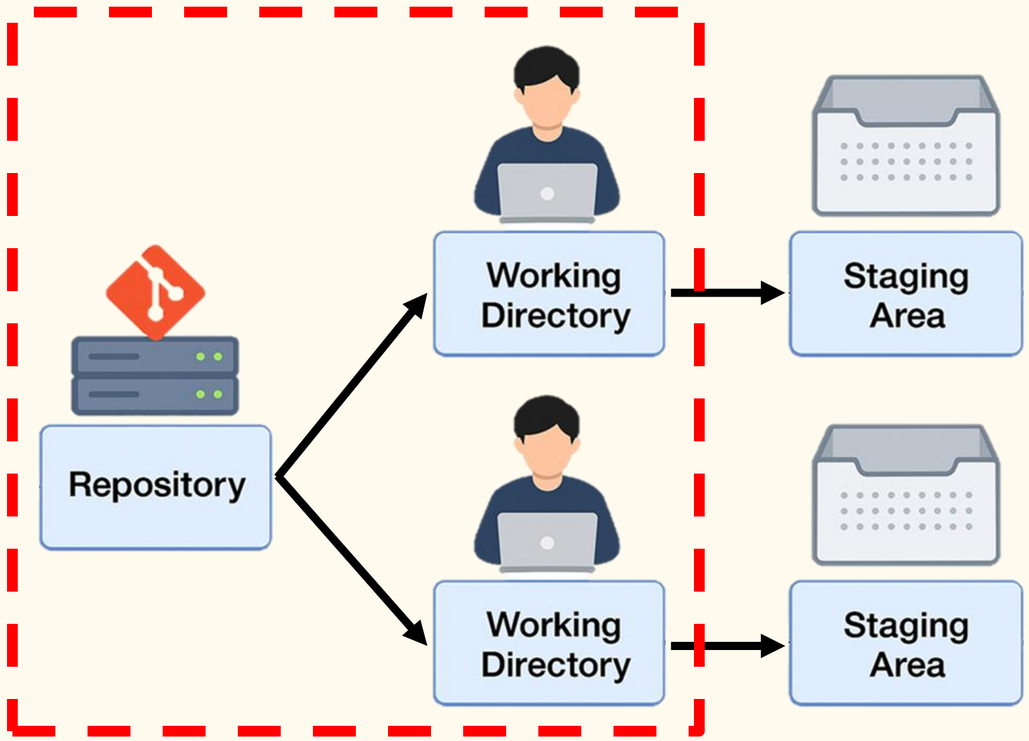
- **merge**
  - Branch를 병합
- 명령어
  - Merge 할 브랜치로 이동 후
    - git merge [branch 명]



05. 협업을 위한 명령어(8)

- log
  - 브랜치 내 commit한 모든 기록을 확인

- 명령어
  - git log



06. README.md

- README.md 파일이란?
  - Git 첫페이지를 장식할 인사말
  - 담으면 좋을 내용
    - 프로젝트 팀원들에게 알릴 사항
    - 프로젝트 개요: 프로젝트 목표, 기술 스택 등
    - 설치 방법: 라이브러리, 프로그램 설치 과정
    - 사용법: 기본적인 코드 예시 및 사용법
    - 라이선스 정보: 오픈소스 프로젝트일 경우 라이선스 규정

06. requirements.txt

- requirements.txt 파일이란?
  - Python 프로젝트에서 필요한 패키지 목록을 기록하는 파일
  - 특정 버전의 패키지를 명시
  - **주요 명령어:**
    - pip freeze > requirements.txt:
      - 설치된 패키지 목록을 requirements.txt 파일로 저장
    - pip install -r requirements.txt
      - 해당 파일을 기준으로 패키지 설치

※ 프로젝트 팀원 모두가 동일한 모듈(버전) 을 사용할 수 있음

06. Git Tag – 버전 관리

- Tag란?
  - 특정 커밋에 이름을 붙여 기억하거나 공유하기 쉽게 만드는 기능
  - 주요 활용 사례
    - 버전 관리 (v0.0.1, v1.0.1, v1.0, v1.2 …)
    - 차수 관리 (1주차 완성, 2주차 완성 …)
    - 특정 내용 추가 (로그인 에러 발생 코드, 로그인 에러 해결 …)
  - 명령어
    - git tag v1.0.0 (목록 확인: git tag, 삭제 : git tag –d v1.0.0)
    - git push origin v1.0.0

06. 코드 작업할 때, 유의사항

- **절대 올리면 안되는 것**
  - 개인 / 민감 정보
    - 로그인 ID / PW
    - 클라우드 Credential 정보
    - 데이터 베이스 정보
    - API Key

▶ 원격 저장소 공개되면 복구 어렵기 때문에, 커밋 전 반드시 코드 리뷰 및 파일 체크

▶ 실제 사례: AWS 키 유출 → 요금 폭탄 발생

06. .gitignore

- .gitignore 이란?
  - Git이 추적하지 않을 파일/폴더 지정
  - 주로 제외하는 항목
    - 개인 / 민감 정보 파일
    - 산출물(결과 파일)
    - 개인 환경 설정 파일
    - 테스트를 위한 임시 파일

.gitignore

```
dk_test.py
myinfo/
dist/
*.exe
*.zip
*.test
```

06. Commit 메세지 전략

- Commit 메시지 규칙
  - 팀원들간 규칙을 만들면 좋음
  - 타입:간결한 설명
    - feat:덧셈기능추가
    - fix:덧셈기능버그개선

타입	설명
feature	새로운 기능 개발
fix	버그 수정
docs	문서 수정
refactor	코드 구조 변경
style	코드 스타일 수정
chore	설정 파일 변경
test	테스트 코드

06. 브랜치 이름 전략

- 브랜치 네이밍 규칙
  - 이름/문서번호, 이름/기능내용
    - daniel/note_1, daniel/add_function
  - 실제 협업에서 사용하는 네이밍 규칙

타입	설명	예시
feature	새로운 기능 개발	feature/login-page
bugfix	버그 수정	bugfix/fix-header-error
hotfix	긴급 버그 수정	hotfix/critical-crash
release	배포 준비 및 버전 관리	release/v1.2.0
experiment	실험적 기능 개발	experiment/new-algorithm

06. 브랜치 전략

- 브랜치 전략
  - Feture/* : 개별 개발자 브랜치
  - Dev : 개발 브랜치
  - Stg : 테스트 브랜치
  - Master : 실제 서비스 코드