CI/CD 기본!

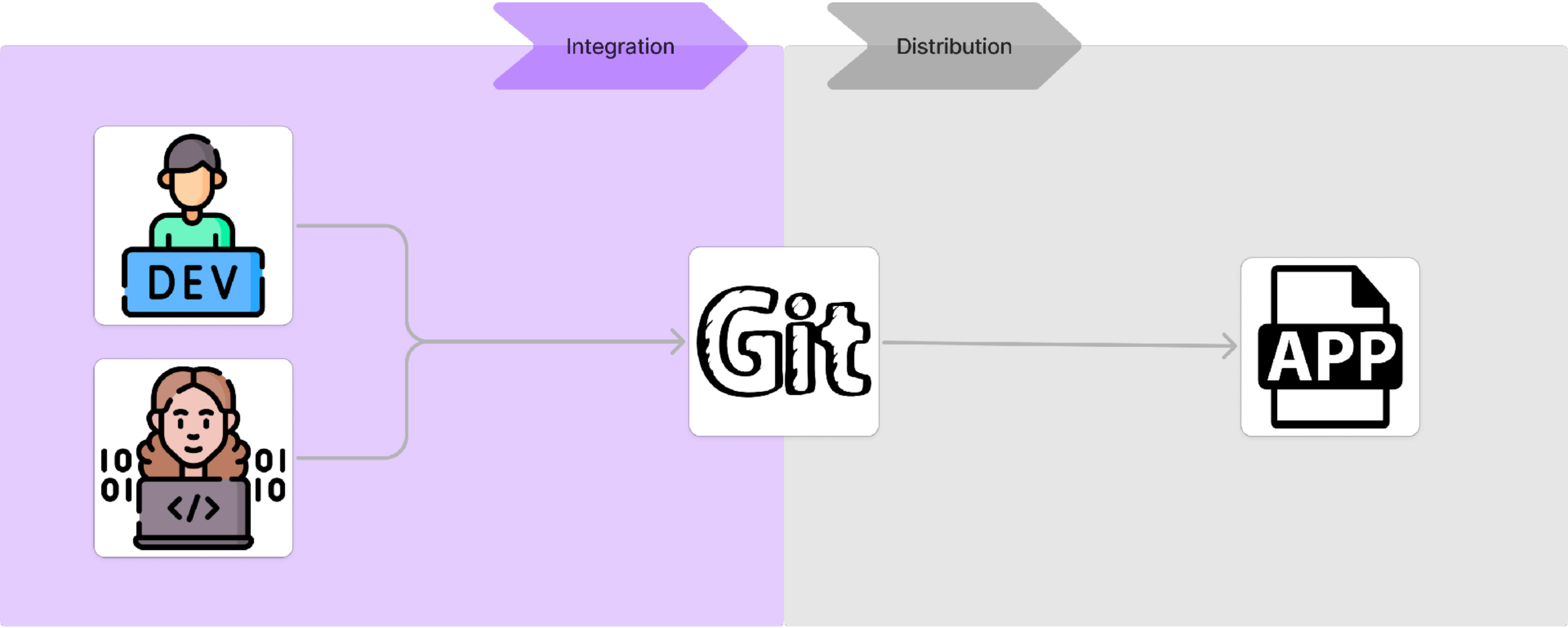
정말 기본 개념만...

EJ

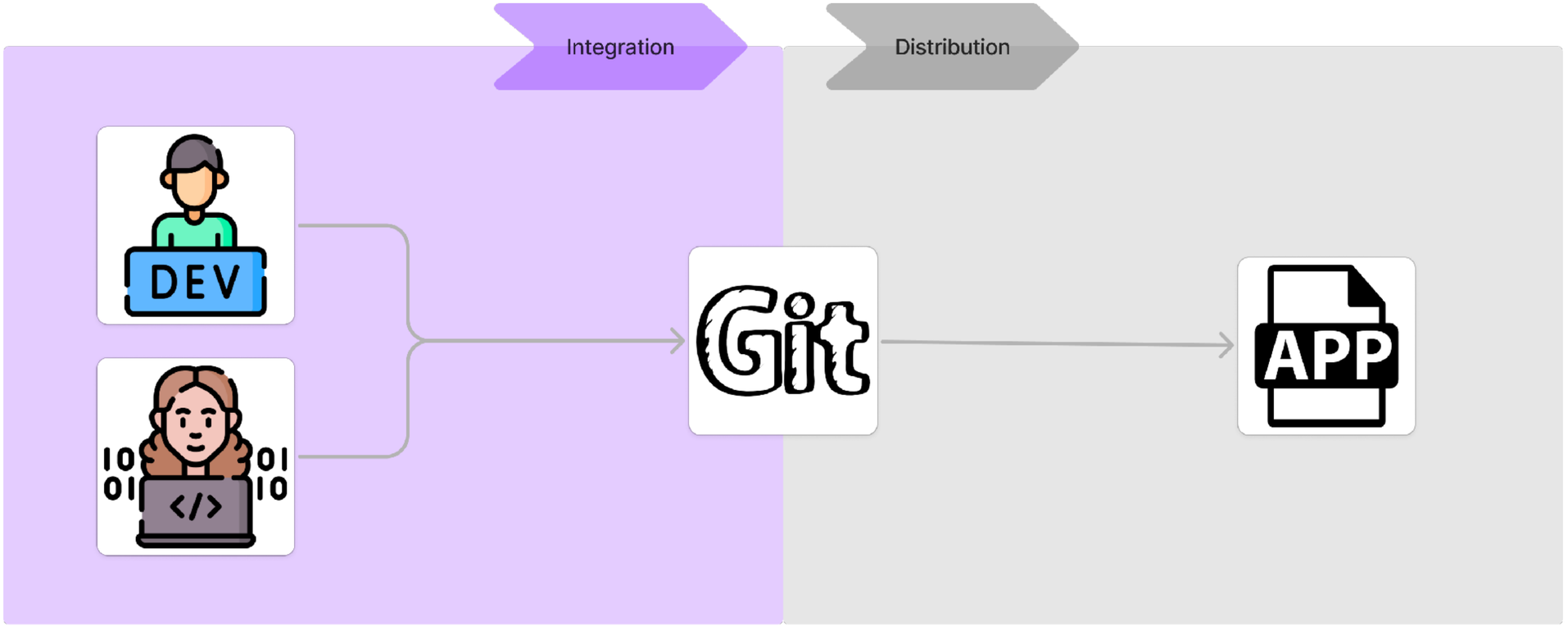
1. 목차

2. Software Lifecycle

3. Integration, Distribution

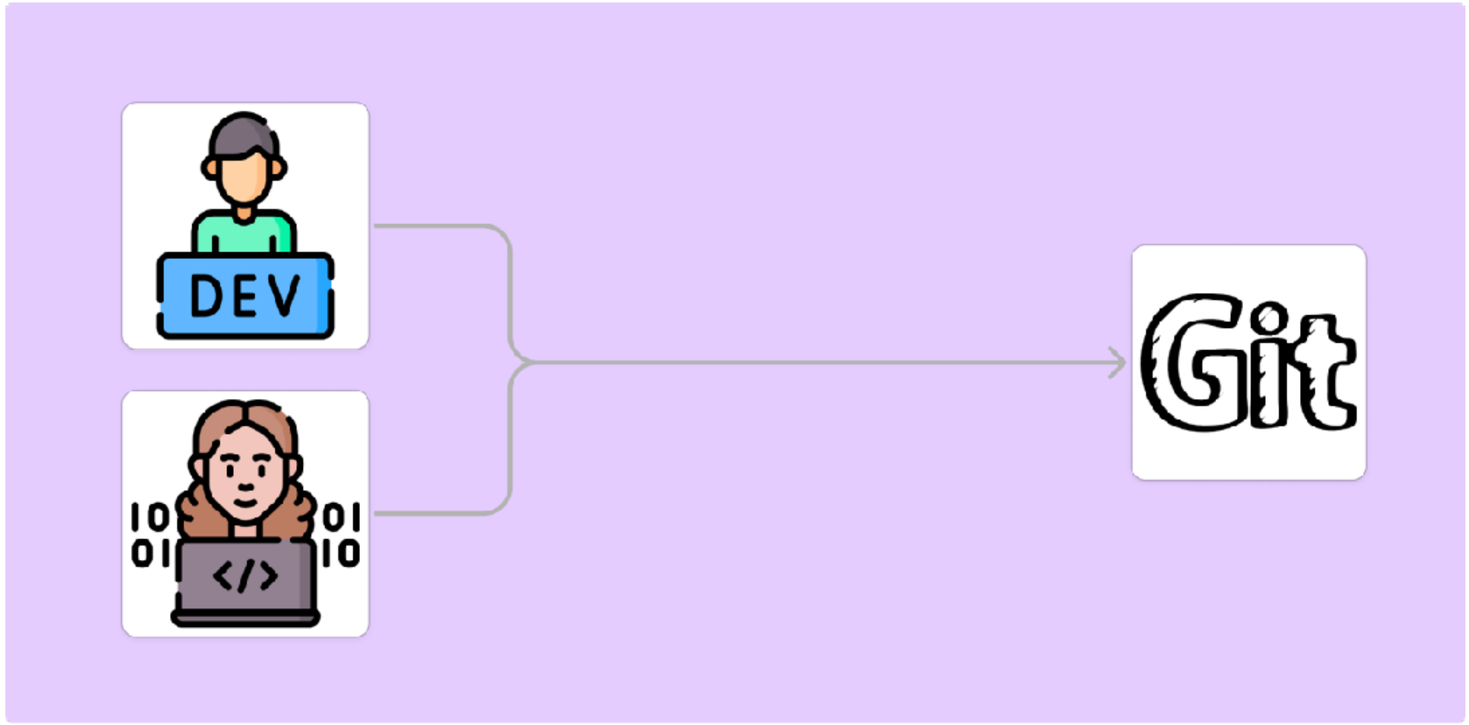


4. CI/CD



자동화.. 어디까지 알아보셨어요?

5. CI



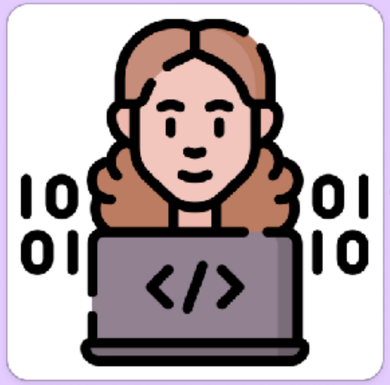
Integration : 각 개발자의 작업사항을 병합하는 전체 프로세스

이걸 🤯 왜 굳이 이걸 자동화 해야 될까?

5. CI

내 코드가 병합 되었을 때 문제가 없을까? 🤓

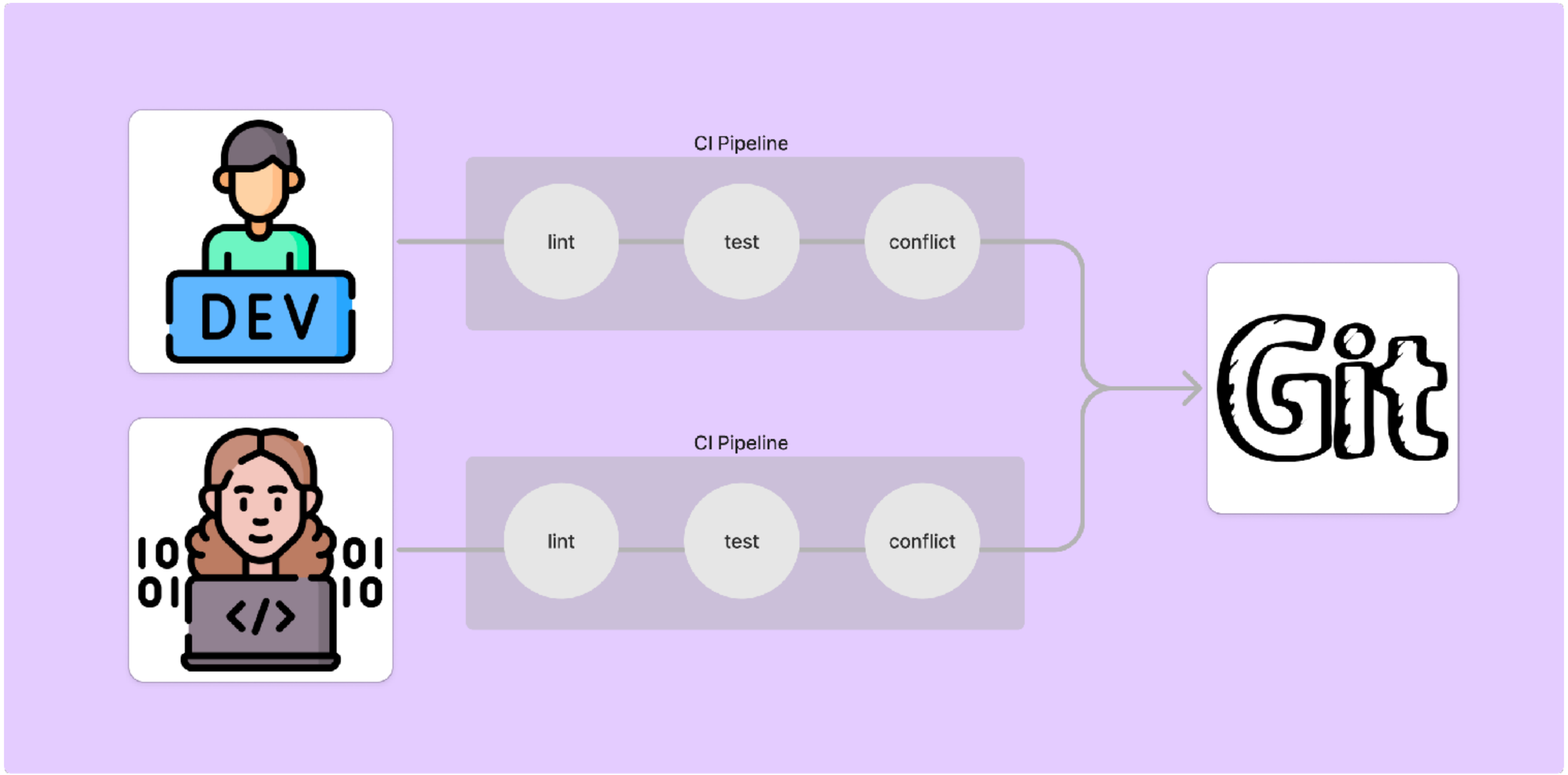
lint → test → conflict



lint → test → conflict

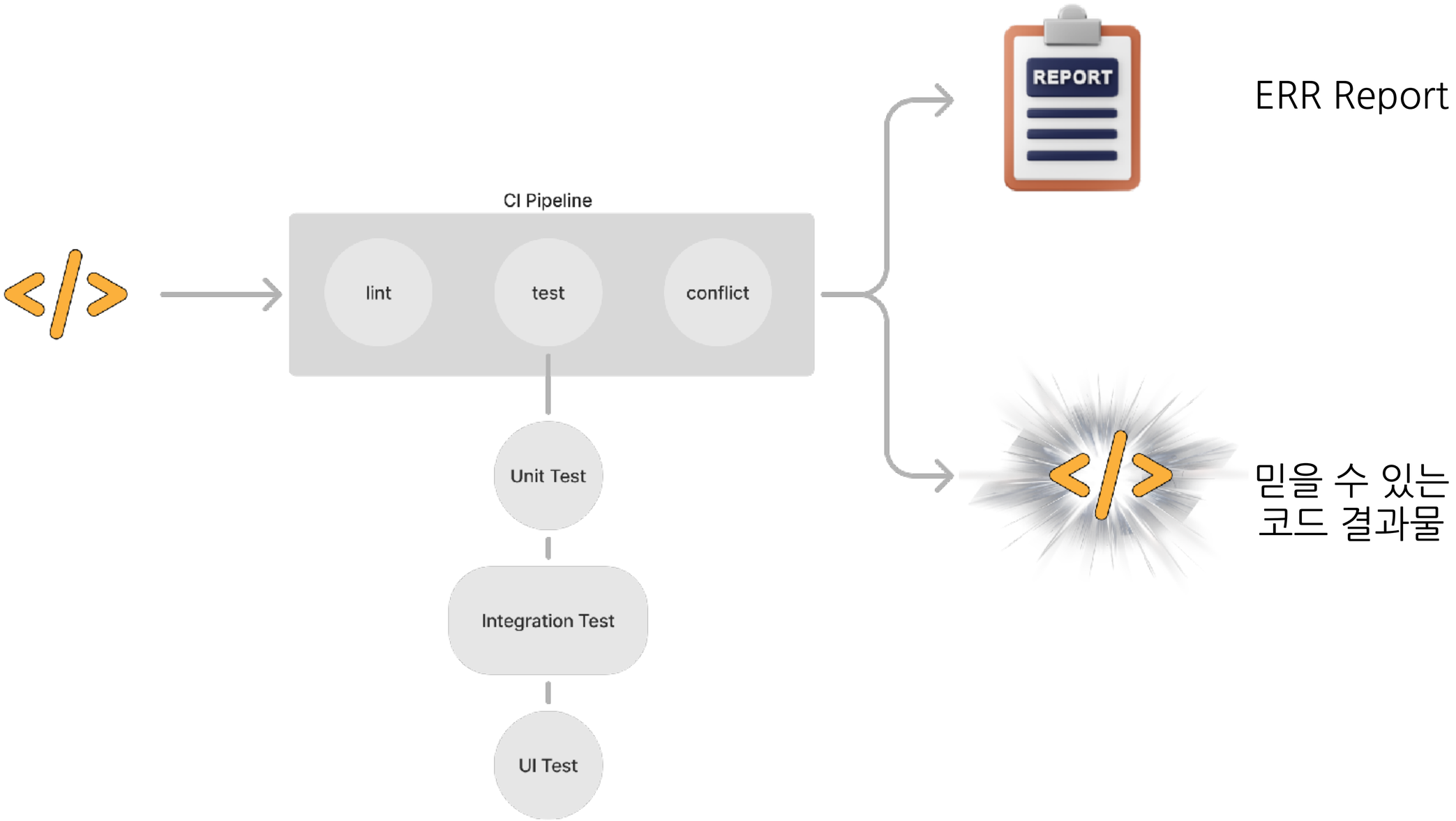
근데 이 과정이 너무 귀찮아!

5. CI



🚀 코드가 병합 되었을 때 문제가 없게끔 해주는 과정을 자동화하자 !

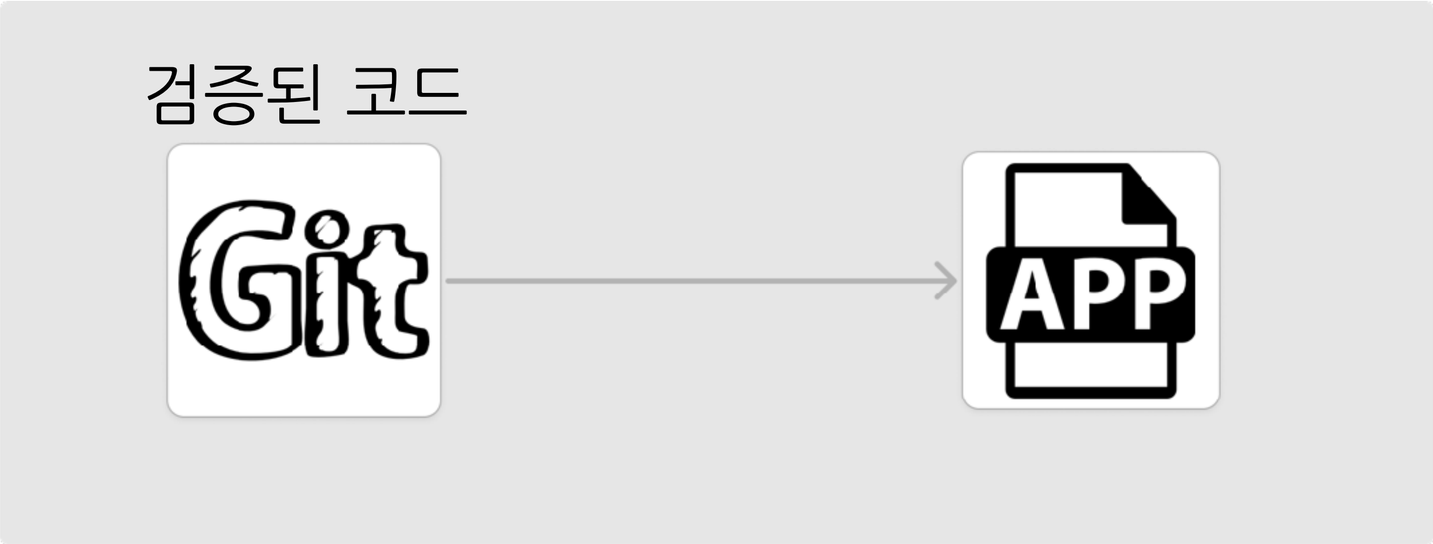
5. CI


REPORT
ERR Report
CI Pipeline
lint
test
conflict
Unit Test
Integration Test
UI Test
믿을 수 있는
코드 결과물

5. CI - 장점

1. 개발자가 덜 귀찮아진다. << 매우 강력!
2. 실수를 줄일 수 있다.
    1. 사람이 아니라, 기계가 하는 일임. 누락은 없음.
3. 문제 발견이 빨라진다.
    1. 병합 후 문제 발견이 아니라, 병합 전 발견이 가능해짐.
    2. 대응도 자연히 빨라지게 됨.

6. CD



배포 :: 우리의 작업물을 유저가 사용하는

Production Level로 넘기는 것

얼마나 ✨자동✨으로 배포할 것이냐

6. CD

Delivery	Deployment
출시 가능한 레벨로 유지 최종 프로덕션 레벨로의 배포는 수동으로 진행.	개발자의 변경 사항이 파이프라인 상으로 문제가 없다면? → 프로덕션까지 바로 배포
완전 자동화인 지속적 배포에 비해서는 느릴 수 있음.	믿을만한 CI 단계를 거치지 않는다면, CD의 결과물이 불완전한 소프트웨어일 수도 있음.

6. CD - for Android Dev

1. 자동 패키징
    1. CI 결과물로 AAB, APK 빌드.
2. 팀 내 테스트 용 어플리케이션 빌드
3. 프로덕션 레벨 앱 빌드
4. ~~fastlane 사용해서 play console에 앱 업로드~~

6. CD - 장점

1. 개발자가 덜 귀찮아진다. << 매우 강력!
    1. 내부 테스트 용 앱을 파이프라인 한 번 태우면 나오게 할 수 있기 때문.
2. 문제 발견이 빨라진다. (distribution을 했을 때)
    1. 한 번에 많은 걸 배포하면? 문제가 발생했을 때 추적이 어려움.
    2. 그러나 작게 여러번 배포 시 문제 발생 추적이 비교적 쉬워짐.
    3. cicd가 user한테 빨라지는게 아니라 기디한테도 빠름.

7. CI/CD Tool

CI/CD Pipeline도 프로그램임.

이걸 돌릴 컴퓨터가 필요함.

➡️ 컴퓨터를 어디서 제공하느냐에 따라 설치형/클라우드형 CI/CD Tool로 나눠짐.

설치형	
Jenkins	Travis CI, Github Actions

8. Software Development Lifecycle