Git으로 협업하기

Git?

버전 관리 시스템

- 소스코드를 주고 받을 필요 없이, 같은 파일을 **여러 명이 동시에 작업하는 병렬 개발이 가능**하다.
  즉, 브랜치를 통해 개발한 뒤, 본 프로그램에 합치는 방식(Merge)으로 개발을 진행할 수 있다.

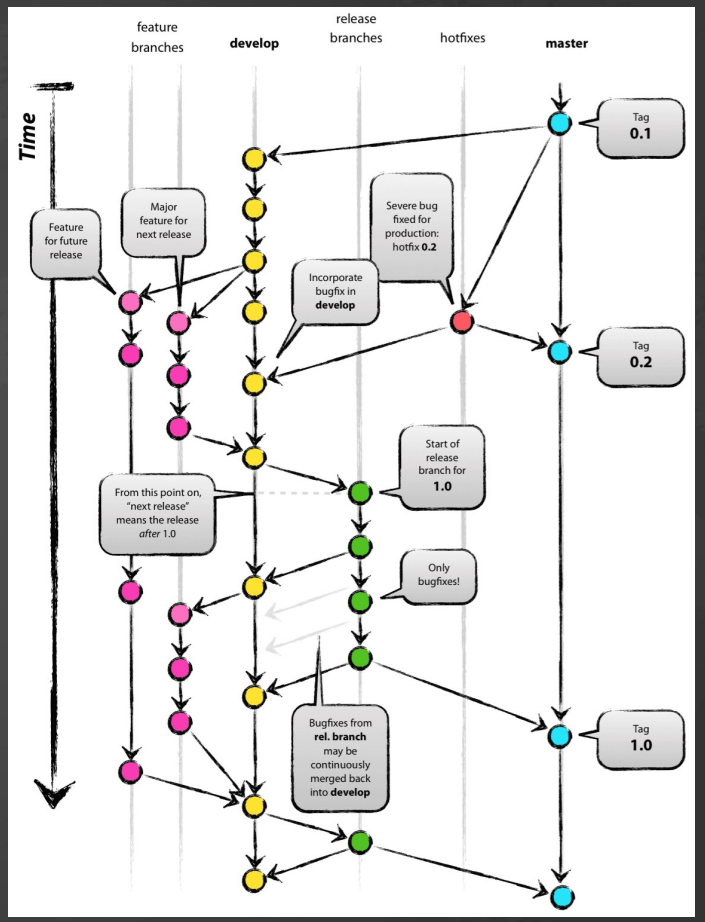
- 분산 버전관리이기 때문에 인터넷이 연결되지 않은 곳에서도 개발을 진행할 수 있으며, 중앙 저장소가 날라가버려도 다시 원상복구할 수 있습니다.

- 팀 프로젝트가 아닌, 개인 프로젝트일지라도 GIT을 통해 버전 관리를 하면 체계적인 개발이 가능해지고, 프로그램이나 패치를 배포하는 과정도 간단해진다. (pull을 통한 업데이트, patch 파일 배포)

Git-flow 전략이란?

Git-flow에는 5가지 종류의 브랜치가 존재합니다.
항상 유지되는 메인 브랜치들(master, develop)과 일정 기간 동안만 유지되는 보조 브랜치들(feature, release, hotfix)

- master: 제품으로 출시되는 브랜치
- develop: 다음 출시 버전을 개발하는 브랜치
- feature: 기능을 개발하는 브랜치
- release: 이번 출시 버전을 준비하는 브랜치
- hotfix: 출시 버전(master)에서 발생한 버그를 수정하는 브랜치

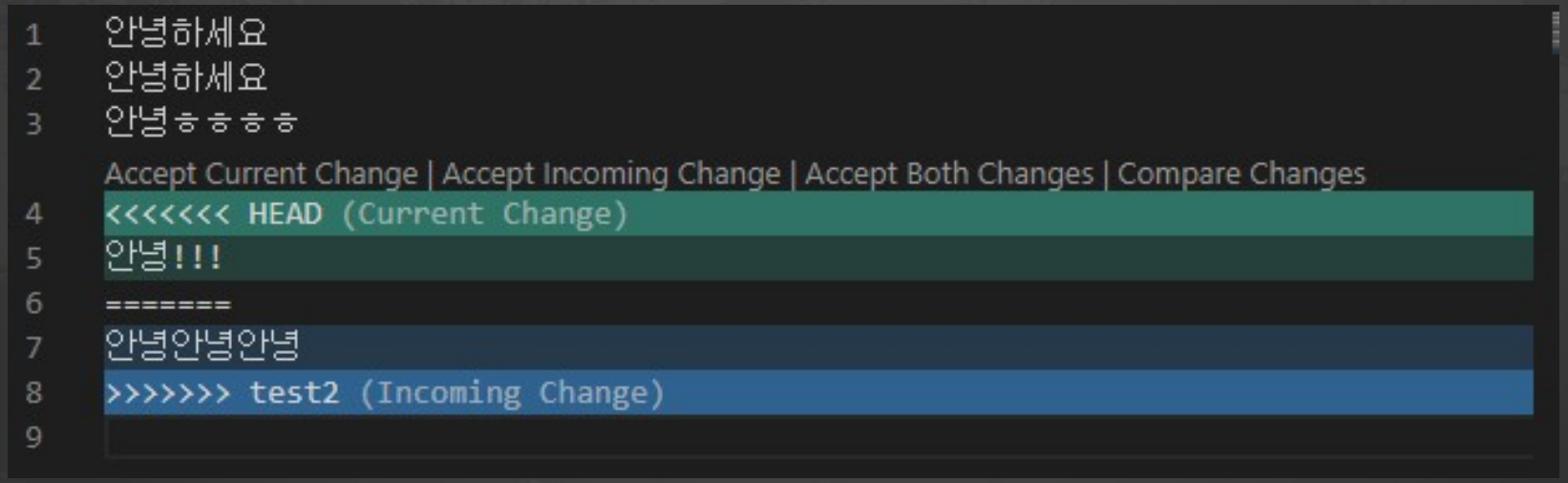


CLI(Command-line Interface)

1. git pull origin master (master 브랜치에서 코드 불러오기 * 항상 새로운 작업 실행하실때마다 master 브랜치에서 pull을 해야합니다!)
2. git checkout -b 새 브랜치명
(이 작업은 새 브랜치 생성과 브랜치 이동을 한번에 하는 명령어에요!)
ex) git checkout -b test
예시 처럼 명령어 작성하시면 master 브랜치에서
test 브랜치로 이동합니다!
3. 코딩!!!
4. git add .
5. git commit -m "커밋메세지"
7. git checkout master // master 브랜치 이동
8. git merge test
9. git push origin master
이렇게 하시면 test 브랜치에서 작업하던 내용들이 master 브랜치에 병합됩니다!

병합시 충돌

- 두 브랜치에서 같은 곳을 수정하면 충돌이 일어난다.
- 수정하고 커밋해주면 됩나다!!



DevOps로 개발환경 세팅하기

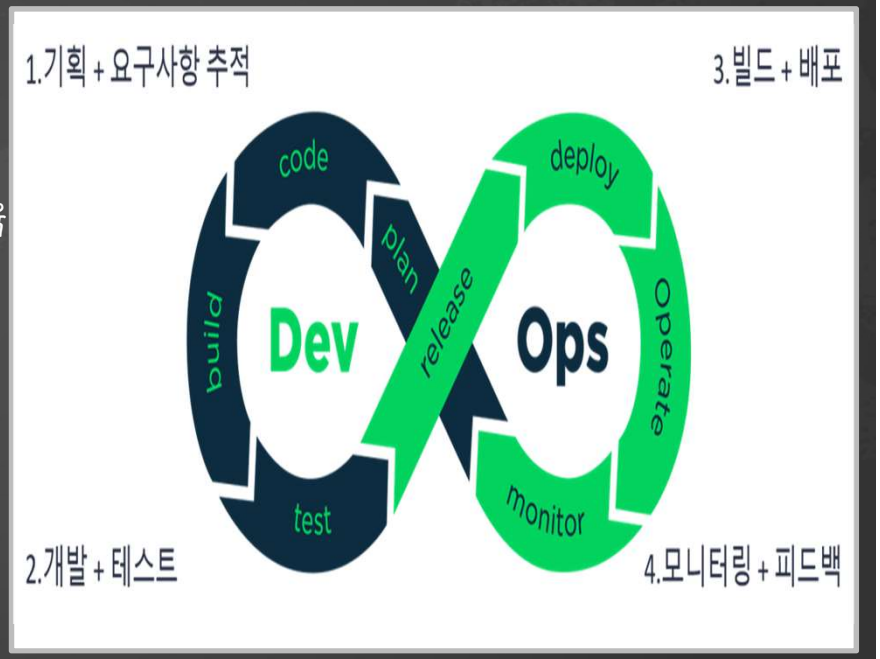
DevOps? Development(개발) + Operations(운영)

1. 신속한 제공
릴리스의 빈도와 속도를 개선하여 제품을 더 빠르게 혁신하고 향상할 수 있다. 새로운 기능의 릴리스와 버그 수정 속도가 빨라질수록 고객의 요구에 더 빠르게 대응할 수 있다.

2. 안정성
최종 사용자에게 지속적으로 긍정적인 경험을 제공하는 한편 더욱 빠르게 안정적으로 제공할 수 있도록, 애플리케이션 업데이트와 인프라 변경의 품질을 보장할 수 있다.

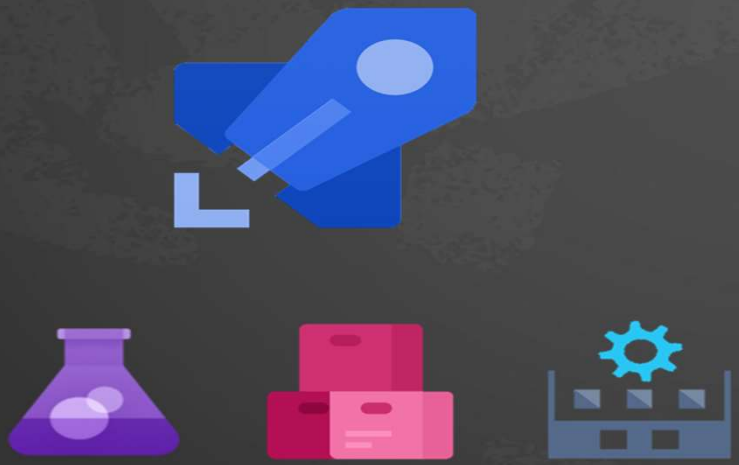
3. 협업 강화
개발자와 운영 팀은 긴밀하게 협력하고, 많은 책임을 공유할 수 있도록, 워크플로를 결합한다. 이를 통해 비효율성을 줄이고 시간을 절약할 할 수 있다.

4. 보안
제어를 유지하고 규정을 준수하면서 신속하게 진행할 수 있다. 자동화된 규정 준수 정책, 세분화된 제어 및 구성 관리 기술을 사용할 수 있다.



CI(Continuous Integration) – 지속적 통합

Development에 속하는 작업으로 지속적으로 프로젝트의 요구사항을 추적하며, 개발된 코드를 테스트 및 빌드를 수행한다.



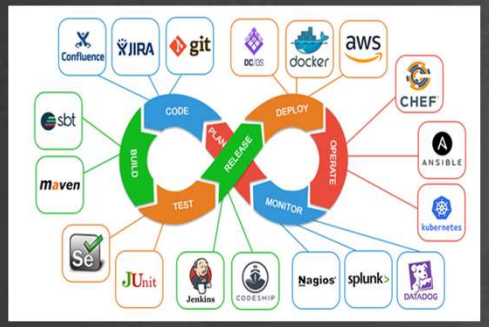
1. 프로젝트 기획 + 요구사항 추적
   - 프로젝트 시작
   - 기획(프로젝트 방법론 채택)
   - 작업관리(Backlog 관리)
   - 진행상황 추적
2. 개발 + 테스트
   - 코드작성
   - 단위 테스트
   - 소스제어
   - 빌드
   - 빌드 확인

CD(Continuous Deployment(Delivery)) – 지속적 제공

Operations에 속하는 작업으로 CI가 완료되어 빌드된 소스를 통합 테스트(개발, QA, Staging)를 거쳐 배포를 하며, 배포된 사항들을 지속적으로 모니터링하고 프로젝트 요구사항에 피드백하는 작업을 수행한다.

3. 빌드 + 배포
   - 자동화된 기능 테스트
   - 통합 테스트 환경(Dev)
   - 사전 제작 환경 (QA, Load testing)
   - 스테이징 환경(Staging)
4. 모니터링 + 피드백
   - 모니터링
   - 피드백

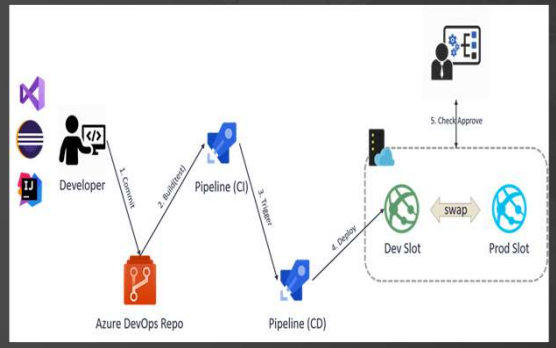
DevOps의 각 단계에 맞추어서 원하는 (특화된)제품을 선택하여 사용하면 된다.



프로젝트에 참여하는 모든인원이 이 제품들에 대해서 이해하고 사용하기 어려우며, 추가적인 관리가 어렵다!

Auzre DevOps

Azure DevOps에서는 파이프라인이라는 형태로 CI/CD를 구성하도록 되어 있으며, 각 단계 구성은 아래 그림과 같다.



1. Git push 할 사람의 정보를 입력

2. push 할 branch를 가져오는 명령어(loop)입력

3. Push 하기 전에 DevOps에 커밋된 최신 버전을 Pull

4. Push 명령어 입력