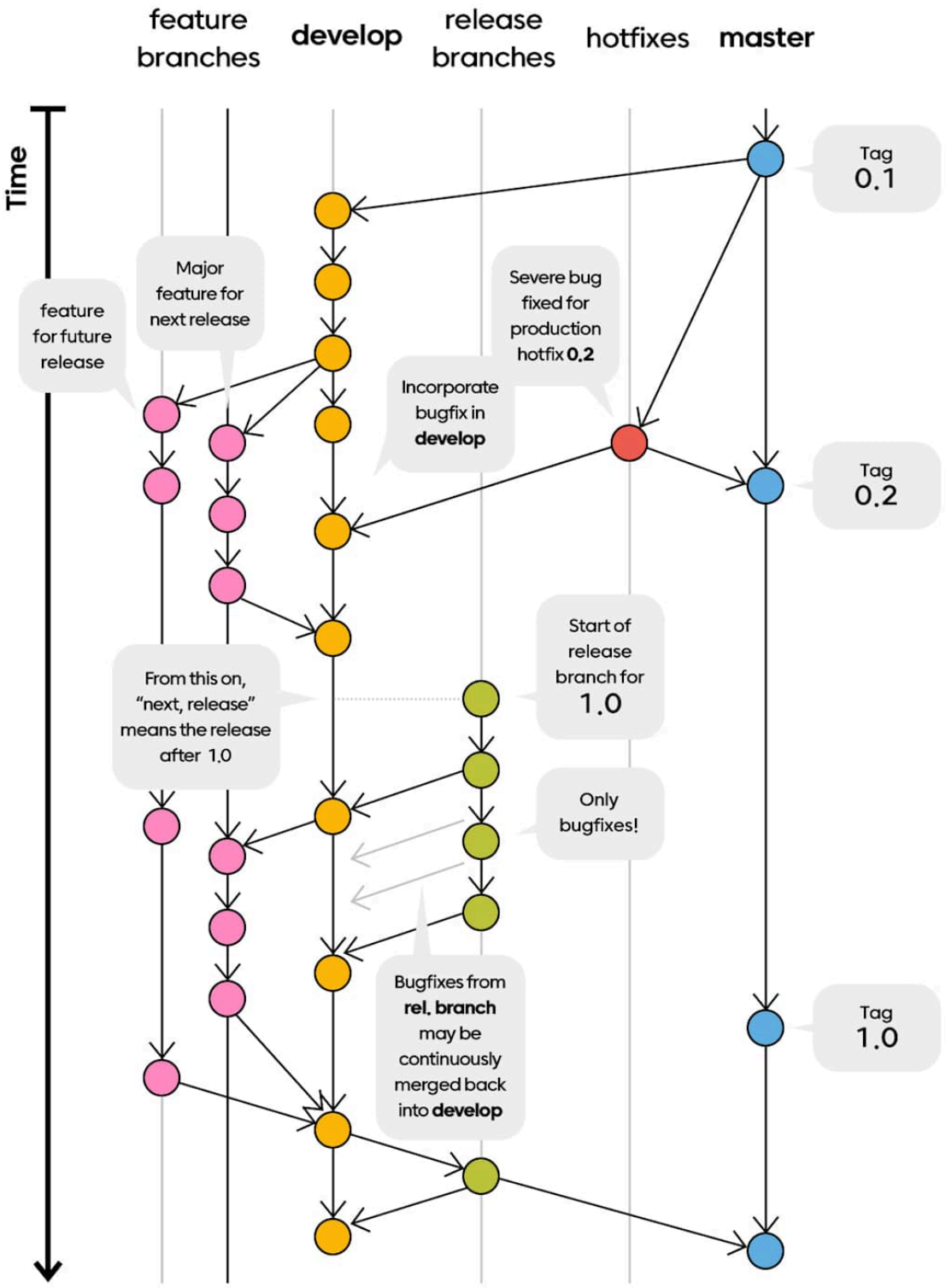
GitHub Desktop을 활용한 안전한 개발/배포 프로세스 개선안

안녕하세요! 여러분이 겪고 있는 소스 관리와 배포 문제들을 GitHub Desktop만으로도 충분히 해결할 수 있는 실무적인 방법을 단계별로 설명드릴게요.

현재 상황 분석과 핵심 문제점

여러분의 환경을 정리하면:

- 개발환경: STS4.5, Zulu-8, Tomcat 9, Maven, MSSQL, MyBatis
- 협업: 2명 개발자, GitHub 경험 부족
- 서버: 같은 서버에 DEV/REAL 톰캣 분리 운영
- 배포: 수동 WAR 빌드 → VDI 접속 → webapps 폴더 복사



Git-Flow branch strategy diagram showing feature, develop, release, hotfix, and master branches with key workflow events and tagging.

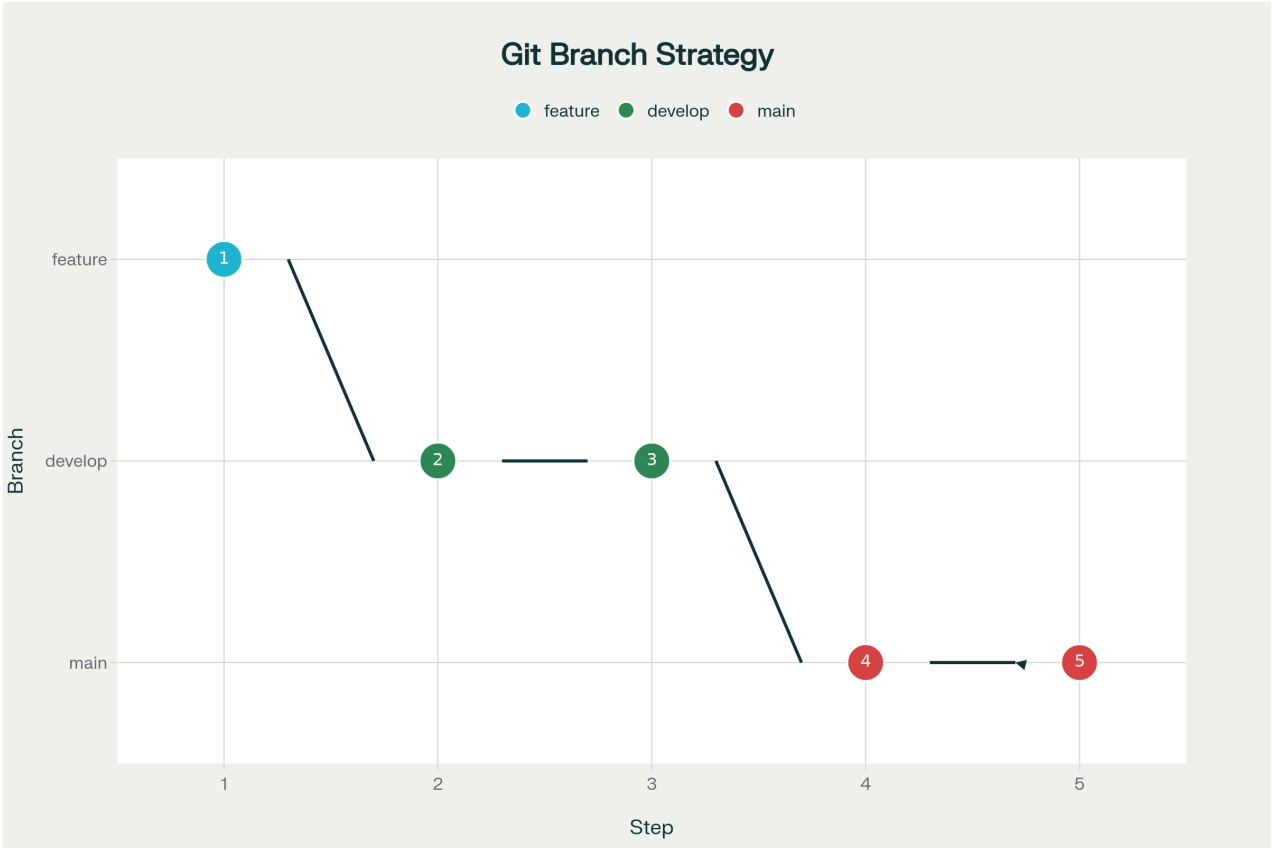
가장 큰 문제는 **모든 기능이 develop에 혼재되어 운영 배포시 불필요한 기능까지 섞여 들어가는 것**입니다.

현재 프로세스 문제점과 개선방안 비교표

1단계: 개선된 브랜치 전략 (3브랜치 구조)

브랜치 역할 정의

- **feature 브랜치**: 개별 기능 개발 (예: `feature/로그인개선`, `feature/결제모듈`)
- **develop 브랜치**: 개발 완료된 모든 기능 통합 → DEV 톰캣 배포용
- **main 브랜치**: 운영 승인받은 기능만 → REAL 톰캣 배포용



개선된 GitHub Desktop 기반 브랜치 관리 전략 프로세스

GitHub Desktop에서 브랜치 생성하기

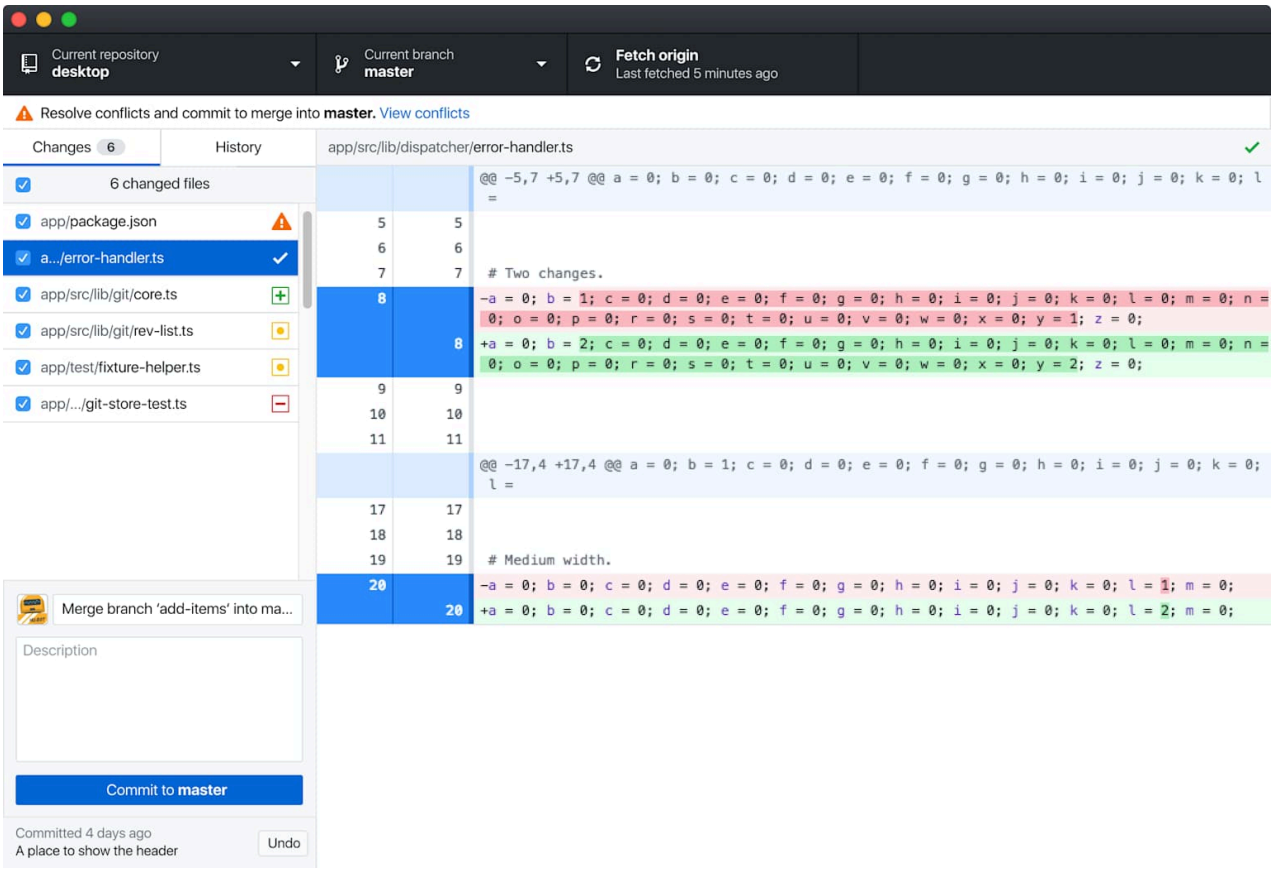
1. **Current Branch** 클릭 → **New Branch** 선택
2. 브랜치명 입력: `feature/회원가입수정` (기능명 명확히)
3. **Create Branch** 클릭하여 생성
4. **Publish Branch**로 원격에 업로드

2단계: 일상적인 개발 워크플로우

기능 개발 프로세스

1. **feature 브랜치에서 작업**
   - STS4에서 코딩 완료
   - GitHub Desktop에서 변경사항 확인
   - 커밋 메시지 구체적으로 작성: "회원가입 시 이메일 중복체크 추가"
2. **develop으로 통합**
   - develop 브랜치로 전환
   - **Branch** 메뉴 → **Merge into current branch**
   - feature 브랜치 선택하여 병합
3. **충돌 해결 (중요!)**

- 충돌 발생시 **Open in Visual Studio Code** 클릭
- `<<<<<<<`, `=======`, `>>>>>>>` 표시 부분 수정
- 저장 후 GitHub Desktop에서 **Continue merge**



GitHub Desktop interface showing merge conflicts in a source file with options to resolve and commit changes to the master branch.

3단계: 테스트 서버 배포 (DEV 톰캣)

develop 브랜치 기준 배포

1. develop 브랜치로 체크아웃 확인
2. STS4에서 Maven 빌드:

```
프로젝트 우클릭 → Run As → Maven clean
성공 후 → Run As → Maven install
```

3. `target` 폴더의 WAR 파일을 DEV 톰캣 `webapps`에 복사
4. 클라이언트에게 테스트 요청

4단계: 운영 배포를 위한 선택적 반영

Cherry-pick으로 필요한 기능만 main에 반영

GitHub Desktop에서 Cherry-pick 방법:

1. **History** 탭에서 운영 반영할 커밋 선택
2. 우클릭 → **Cherry-pick commit in...**
3. main 브랜치 선택
4. 충돌 발생시 동일하게 해결
[1]

중요한 점: 전체 develop을 main에 병합하지 말고, **승인받은 기능의 커밋만** cherry-pick으로 가져오세요!

5단계: 운영 서버 배포 (REAL 톰캣)

1. main 브랜치로 체크아웃
2. Maven clean → install로 WAR 빌드
3. REAL 톰캣 `webapps`에 복사
4. 배포 완료 후 서비스 정상 동작 확인

협업 및 실수 방지 전략

충돌 예방법

- 작은 단위로 자주 커밋
- **Fetch origin** 자주 클릭하여 최신 상태 유지
- 병합 전 동료와 코드 리뷰

롤백 방법

- **History** 탭에서 문제된 커밋 우클릭
- **Revert this commit** 선택
- 이전 상태로 안전하게 되돌리기

브랜치 관리 팁

```
현재 브랜치 확인: GitHub Desktop 상단의 Current Branch
브랜치 전환: Current Branch 클릭 후 원하는 브랜치 선택
브랜치 삭제: Branch 메뉴 → Delete 선택 (병합 완료 후)
```

실무 체크리스트

비효율 요소와 개선 효과

현재 방식의 문제

- 수동 소스 복사/삭제로 인한 실수
- 테스트되지 않은 기능의 운영 반영
- 변경 이력 추적 어려움

개선 후 효과

- Git 이력으로 모든 변경사항 추적 가능
- 브랜치별 배포로 안정성 확보
- 롤백/복구 작업 간편화

점진적 도입 방안

1주차: 브랜치 구조 익히기

- feature → develop 병합 연습
- 충돌 해결 연습

2주차: Cherry-pick 활용

- main 브랜치에 선택적 반영 연습
- 운영 배포 프로세스 정착

3주차: 완전 정착

- 모든 개발/배포를 새로운 방식으로 진행
- 문제점 보완 및 최적화

마무리: 성공을 위한 핵심 포인트

1. **절대 develop을 main에 직접 병합하지 마세요** - Cherry-pick으로 필요한 것만!
2. **커밋 메시지를 구체적으로** - 나중에 추적하기 쉽게
3. **작은 단위로 자주 커밋** - 실수해도 부분만 롤백
4. **배포 전 반드시 빌드 테스트** - BUILD SUCCESS 확인
5. **동료와 크로스 체크** - 운영 배포는 2명이 확인

이렇게 하면 자동화 도구 없이도 GitHub Desktop만으로 안전하고 체계적인 소스 관리가 가능합니다. 처음에는 복잡해 보여도 2-3번 반복하면 자연스러워질 거예요!

혹시 특정 단계에서 막히거나 더 자세한 설명이 필요한 부분이 있으면 언제든 질문해 주세요. 실제 작업하면서 생기는 문제들도 하나씩 해결해 나갈 수 있어요! 😊

⁂

1. https://docs.github.com/ko/desktop/managing-commits/cherry-picking-a-commit-in-github-desktop