실전 소스/배포 관리 프로세스 개선안

현재 상황 분석 및 핵심 문제점

너희 상황을 보니까 정말 많이 겪는 문제들이야[1] [2] [3]. 특히 **테스트 서버에 모든 기능이 섞여서 올라가고, 운영에는 일부만 반영하려다 실수하는 부분**이 가장 위험해[4] [5].

주요 위험 요소들

- **소스 혼입으로 인한 서비스 오작동**[6] [7]
- **수동 배포 과정에서의 인적 오류**[8] [9]
- **브랜치 관리 복잡성과 롤백 어려움**[1] [10]
- **협업 충돌 관리의 한계**[11] [12]

1단계: Git 브랜치 전략 재정의

브랜치 구조 (Git-Flow 개선 버전)

```
main (운영 배포용 - 절대 직접 수정 금지)
├── develop (테스트 서버용 - 모든 기능 통합)
├── feature/기능명 (개발용)
└── hotfix/긴급수정명 (운영 긴급 수정용)
```

실전 브랜치 운영 규칙

main 브랜치 (운영용)

- **운영에 반영할 기능만 선별적으로 병합**
- 클라이언트가 "운영 반영" 승인한 기능만 들어감
- 항상 배포 가능한 안정된 상태 유지[1] [10]

develop 브랜치 (테스트 서버용)

- 개발된 모든 기능이 들어감
- 클라이언트 테스트용으로 사용
- 운영 반영과는 별개로 관리

feature 브랜치 (개발용)

- 각 기능별로 독립적인 브랜치 생성

- `feature/로그인기능`, `feature/결제시스템` 이런 식으로 명명
- develop에서 분기해서 develop으로 병합[13] [14]

2단계: 서버별 소스 분리 전략

테스트/운영 서버 분리 원칙

테스트 서버 (develop 브랜치)

```
- 모든 개발 기능 포함
- 클라이언트 테스트 전용
- 불안정해도 상관없음
```

운영 서버 (main 브랜치)

```
- 승인된 기능만 선별 포함
- 안정성 최우선
- 클라이언트 서비스 제공용
```

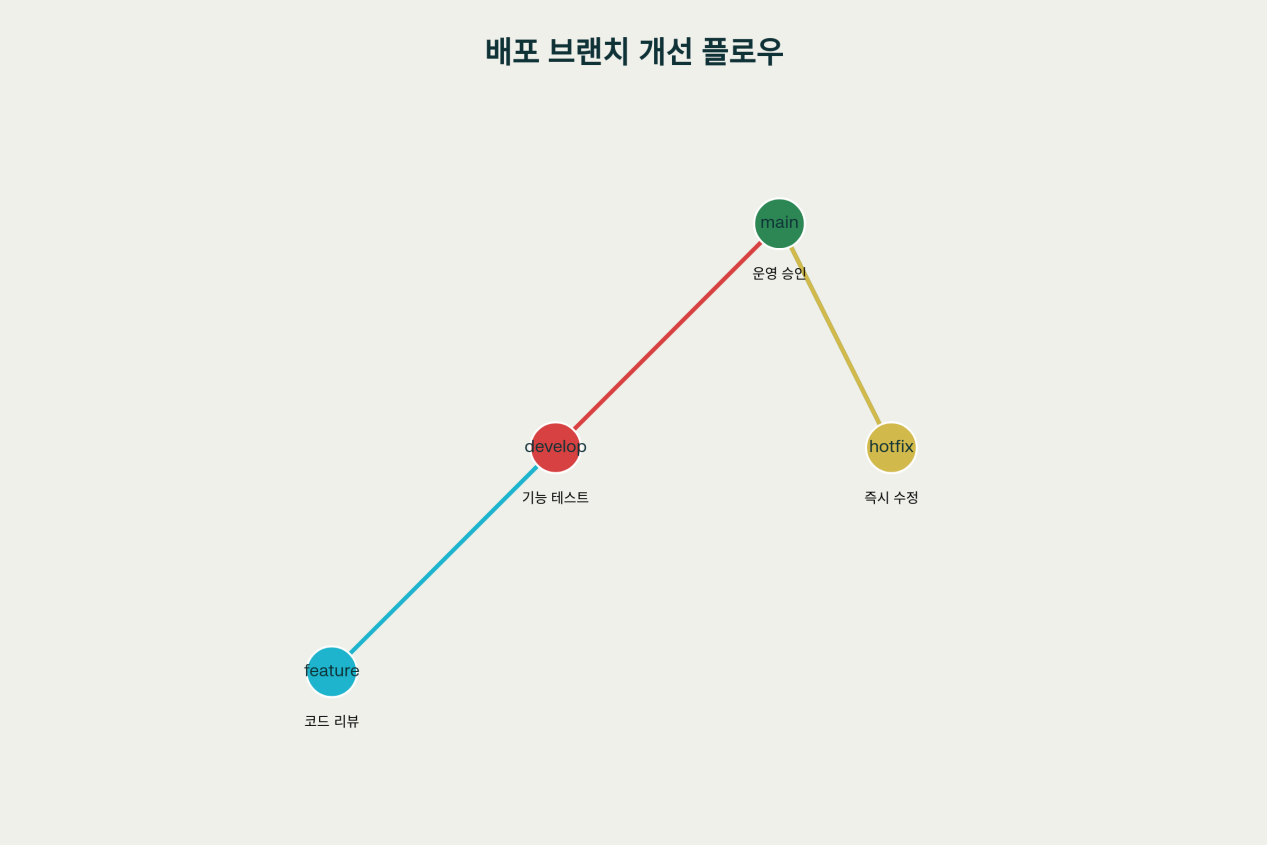
소스 혼입 방지 실전 팁

방법 1: Release 브랜치 활용

```
# 운영 배포할 때
1. main에서 release/v1.2.0 브랜치 생성
2. 필요한 feature만 선별적으로 병합
3. 테스트 완료 후 main으로 병합
4. war 파일 빌드 후 운영 배포
```

방법 2: Cherry-pick 사용

```
# 특정 커밋만 운영에 반영
git checkout main
git cherry-pick [승인받은 기능의 커밋ID]
```



개선된 소스 관리 및 배포 프로세스 플로우차트

3단계: 수동 배포 안전화 프로세스

배포 전 체크리스트

1. 소스 검증 단계

```
□ 클라이언트 승인받은 기능만 포함했는가?
□ 테스트 서버에서 해당 기능들 정상 동작 확인했는가?
□ 다른 기능에 영향 주지 않는가?
□ 데이터베이스 변경사항 확인했는가?
```

2. 백업 및 준비

```
□ 현재 운영 war 파일 백업 완료
□ 데이터베이스 백업 완료
□ 롤백 계획 수립 완료
□ 배포 시간대 확인 (사용자 적은 시간)
```

3. 배포 실행

```
□ 서버 점검 모드 전환
□ 기존 war 파일 백업 후 교체
□ 서버 재시작 및 동작 확인
```

```
□ 주요 기능 테스트 실행
□ 서비스 정상화 확인
```

War 파일 관리 전략

파일 명명 규칙

```
프로젝트명_버전_날짜.war
예: myproject_v1.2.0_20250806.war
```

백업 관리

```
운영서버/backup/
├── myproject_v1.1.0_20250801.war (이전 버전)
├── myproject_v1.2.0_20250806.war (현재 운영)
└── rollback_guide.txt (롤백 매뉴얼)
```

4단계: 협업 충돌 해결 전략

충돌 예방법

작업 분배 원칙

- 파일 단위로 담당자 명확히 분리
- 같은 파일 작업 시 사전 협의 필수
- 매일 오전 develop 브랜치 최신화[11] [15]

실전 협업 플로우

```
# 매일 작업 시작 전
git checkout develop
git pull origin develop

# 새 기능 작업 시작
git checkout -b feature/내기능명

# 작업 완료 후 충돌 방지
git checkout develop
git pull origin develop
git checkout feature/내기능명
git merge develop  # 충돌 미리 해결

# 최종 병합
git checkout develop
git merge feature/내기능명
```

충돌 해결 단계별 가이드

충돌 발생 시

```
1. 당황하지 말고 git status 확인
2. 충돌 파일 열어서 다음 표시 찾기:
   <<<<<<< HEAD
   (내 코드)
   =======
   (상대방 코드)
   >>>>>>> 브랜치명

3. 둘 다 필요하면 합치고, 하나만 필요하면 선택
4. 표시 제거 후 저장
5. git add [파일명]
6. git commit -m "충돌 해결"
```

5단계: 롤백 및 복구 전략

단계별 롤백 방법

Level 1: 단순 재시작

```
- 서버 재시작으로 해결되는 경우
- 임시 데이터 문제 등
```

Level 2: War 파일 교체

```
1. 서버 중지
2. 백업된 이전 war 파일로 교체
3. 서버 재시작
4. 정상 동작 확인
```

Level 3: 소스 레벨 롤백

```
# Git에서 이전 커밋으로 되돌리기
git checkout main
git reset --hard [이전_커밋_ID]
# 또는
git revert [문제_커밋_ID]
```

Level 4: 데이터베이스 복구

```
1. 애플리케이션 중지
2. DB 백업 파일로 복구
3. 애플리케이션 재시작
```

복구 시간 단축 팁

빠른 롤백을 위한 준비

- 배포 전 반드시 현재 상태 백업
- 롤백 스크립트 미리 작성해 두기
- 주요 기능별 테스트 시나리오 준비
- 긴급 연락망 구축

6단계: 실전 적용 시나리오

시나리오 1: 새 기능 개발 및 배포

개발자 A가 로그인 기능 개발

```
# 1. feature 브랜치 생성
git checkout develop
git pull origin develop
git checkout -b feature/login

# 2. 개발 작업 수행
# 코딩...

# 3. 커밋 및 푸시
git add .
git commit -m "로그인 기능 구현 완료"
git push origin feature/login

# 4. develop에 병합 (테스트 서버 배포)
git checkout develop
git merge feature/login
git push origin develop

# 5. 클라이언트 테스트 후 승인 받으면
# main 브랜치에 선별적 병합
git checkout main
git cherry-pick [로그인_기능_커밋들]
git push origin main

# 6. main에서 war 빌드 후 운영 배포
```

시나리오 2: 긴급 버그 수정

운영에서 결제 버그 발생

```
# 1. hotfix 브랜치 생성
git checkout main
git checkout -b hotfix/payment-bug

# 2. 버그 수정
# 수정 작업...
```

```
# 3. 테스트 및 커밋
git add .
git commit -m "결제 버그 긴급 수정"

# 4. main에 즉시 병합
git checkout main
git merge hotfix/payment-bug
git push origin main

# 5. develop에도 동기화
git checkout develop
git merge hotfix/payment-bug
git push origin develop

# 6. 즉시 운영 배포
```

시나리오 3: 복합 기능 관리

A기능은 승인, B기능은 보류인 경우

```
# 1. release 브랜치로 선별 관리
git checkout main
git checkout -b release/v1.3.0

# 2. A기능만 선택적으로 병합
git cherry-pick [A기능_관련_커밋들]

# 3. 테스트 후 main으로 병합
git checkout main
git merge release/v1.3.0

# 4. 운영 배포 (B기능은 자연스럽게 제외됨)
```

7단계: 도구 및 환경 최적화

STS4.5 설정 최적화

Git 통합 설정

```
1. Window > Show View > Git Repository
2. Project > Properties > Project Facets
3. Maven 프로젝트 자동 빌드 설정
4. War 파일 자동 생성 설정
```

빌드 자동화 (Maven)

```
<!-- pom.xml에서 프로파일 설정 -->
<profiles>
    <profile>
        <id>dev</id>
```

```
        <properties>
            <spring.profiles.active>dev</spring.profiles.active>
        </properties>
    </profile>
    <profile>
        <id>prod</id>
        <properties>
            <spring.profiles.active>prod</spring.profiles.active>
        </properties>
    </profile>
</profiles>
```

수동 배포 스크립트 예시

Windows 배치 파일 (deploy.bat)

```
@echo off
echo 배포 시작...

rem 백업 생성
copy "C:\tomcat\webapps\myproject.war" "C:\backup\myproject_backup_%date%.war"

rem 서버 중지
net stop Tomcat9

rem 새 war 파일 배포
copy "%1" "C:\tomcat\webapps\myproject.war"

rem 서버 시작
net start Tomcat9

echo 배포 완료!
pause
```

결론 및 점진적 적용 방안

단계적 도입 계획

1주차: 브랜치 정리

- 현재 소스를 main 브랜치로 정리
- develop 브랜치 생성 및 동기화
- 팀원들에게 새 규칙 안내

2주차: 프로세스 적용

- feature 브랜치 방식으로 개발 시작
- 소규모 기능으로 프로세스 검증
- 문제점 파악 및 수정

3주차: 도구 정착

- 백업 및 롤백 절차 정립
- 배포 체크리스트 활용
- 팀 내 노하우 공유

4주차: 완전 정착

- 모든 배포에 새 프로세스 적용
- 문제 발생 시 빠른 대응 체계 구축
- 지속적인 개선 방안 논의

핵심 포인트 요약

1. **테스트와 운영을 브랜치로 완전 분리**해서 혼입 방지
2. **선택적 병합**(cherry-pick)으로 원하는 기능만 운영 반영
3. **단계별 백업과 롤백** 절차로 안전성 확보
4. **팀원 간 소통**으로 충돌 최소화
5. **점진적 적용**으로 무리 없이 정착

이렇게 하면 **"소스 삭제/추리기 과정에서 실수로 원치 않는 소스까지 반영"**되는 문제를 원천적으로 해결할 수 있어. 특히 Git의 브랜치 기능을 제대로 활용하면 수동 배포의 위험성을 크게 줄일 수 있고, 협업도 훨씬 수월해질 거야[16] [17].

⁂

1. https://inpa.tistory.com/entry/GIT-⚡-github-flow-git-flow--브랜치-전략
2. https://proengineer.tistory.com/112
3. https://change-words.tistory.com/entry/개발서버-운영서버
4. https://devocean.sk.com/blog/techBoardDetail.do?ID=165571&boardType=techBlog
5. https://weaklion1.tistory.com/35
6. https://megak.tistory.com/293
7. https://web-dev.tistory.com/925
8. https://learn.microsoft.com/ko-kr/biztalk/technical-guides/best-practices-for-deploying-an-application
9. https://appmaster.io/ko/glossary/ci-cd-rolbaeg-jeonryag
10. https://yeoonjae.tistory.com/entry/프로젝트-Git-flow-master-및-develop-branch-설정
11. https://moon99610.tistory.com/62
12. https://printf100.tistory.com/34
13. https://kimforest.tistory.com/34
14. https://adjh54.tistory.com/364
15. https://www.jiniai.biz/2025/05/08/git-협업-실전-가이드-브랜치-충돌-해결-pull-request-완전-정복/
16. https://blog.hwahae.co.kr/all/tech/14184
17. https://meetup.nhncloud.com/posts/377