# GitHub Desktop을 활용한 실전 소스/배포 관리 프로세스 개선안

## 1. 현재 상황 분석 및 문제점

### 주요 문제점

- **소스 혼입 문제**: 테스트 서버에 모든 개발 기능이 섞여 있어 운영 반영 시 오작동 발생[1] [2]
- **수동 관리의 위험**: 소스 삭제/추리기 과정에서 실수 발생 가능성
- **브랜치 관리 미숙**: 병합, 충돌 해결, 롤백 등 Git 기본 기능 활용 부족[3] [4]
- **배포 프로세스 비효율**: 수동 war 파일 빌드 및 서버별 개별 배포[5] [6]

## 2. GitHub Desktop 기반 브랜치 전략 설계

### 2.1 브랜치 구조 재설계

**기본 브랜치 구조**[7] [8]:

- `main`: 운영 서버 배포용 (안정적인 코드만)
- `develop`: 개발 통합 브랜치 (모든 완성된 기능)
- `test`: 테스트 서버 배포용 (클라이언트 테스트용)
- `feature/*`: 개별 기능 개발용

### 2.2 GitHub Desktop 브랜치 생성 매뉴얼

**1단계: 새 브랜치 생성**[9] [10]

1. GitHub Desktop에서 Current Branch 드롭다운 클릭
2. "New branch" 버튼 클릭
3. 브랜치명 입력 (예: `feature/login`, `feature/board`)
4. "Create branch" 버튼 클릭
5. "Publish branch"로 원격 저장소에 공유

**2단계: 브랜치 전환**[11] [12]

1. Current Branch 드롭다운에서 작업할 브랜치 선택
2. 파일 변경 후 Changes 탭에서 변경사항 확인

3. 커밋 메시지 작성 후 "Commit to [브랜치명]" 클릭
4. "Push origin" 버튼으로 원격 저장소에 업로드

### 2.3 브랜치별 역할 분담

| 브랜치 | 용도 | 배포 대상 | 관리 방법 |
|---|---|---|---|
| `main` | 운영 배포 | REAL 톰캣 | 승인된 기능만 병합 |
| `test` | 클라이언트 테스트 | DEV 톰캣 | 선택된 기능만 병합 |
| `develop` | 개발 통합 | - | 모든 완성 기능 통합 |
| `feature/*` | 개별 기능 | 로컬 | 개발자별 독립 작업 |

## 3. 실무 적용 워크플로우

### 3.1 일반적인 개발 프로세스

**개발자 A의 작업 흐름**:

```
1. develop 브랜치에서 feature/A-기능 생성
2. 기능 개발 완료 후 커밋/푸시
3. develop에 병합 후 feature/A-기능 삭제
```

**개발자 B의 동시 작업**:

```
1. develop 브랜치에서 feature/B-기능 생성
2. 독립적으로 개발 진행
3. A와 별개로 develop에 병합
```

### 3.2 테스트 서버 배포 프로세스

**선택적 기능 배포 방법**[13] [14]:

1. `test` 브랜치로 전환
2. GitHub Desktop에서 "Branch" → "Merge into current branch" 선택
3. 테스트할 feature 브랜치들만 선택하여 병합
4. war 파일 빌드 후 DEV 톰캣에 배포

### 3.3 운영 서버 배포 프로세스

**안전한 운영 배포**[15] [16]:

1. 테스트 서버에서 검증 완료된 기능들만 선별
2. `main` 브랜치로 해당 기능들 병합
3. war 파일 빌드

4. REAL 톰캣에 수동 업로드

## 4. 충돌 해결 및 협업 관리

### 4.1 GitHub Desktop 충돌 해결법[3] [4]

**충돌 발생 시 처리 절차**:

1. GitHub Desktop에서 충돌 알림 확인
2. "Open in Visual Studio Code" 클릭
3. `<<<<<<< HEAD`와 `>>>>>>> 브랜치명` 사이 내용 정리
4. 충돌 마커(`<<<<<<<`, `=======`, `>>>>>>>`) 모두 삭제
5. 저장 후 GitHub Desktop에서 "Continue merge" 클릭

### 4.2 협업 모범 사례[17] [18]

**2명 협업 권장사항**:

- 매일 아침 `develop` 브랜치 최신 상태로 pull
- 기능별로 작은 단위로 커밋
- 명확한 커밋 메시지 작성 (예: "[기능] 로그인 화면 개발")
- 코드 리뷰 없이도 안전한 병합을 위해 철저한 로컬 테스트

## 5. 수동 배포 프로세스 최적화

### 5.1 war 파일 관리 체계화[19] [6]

**빌드 및 배포 절차**:

```
1. STS4에서 Maven clean & package 실행
2. target 폴더에서 war 파일 확인
3. 파일명을 배포 환경에 맞게 변경
   - 개발: project-dev.war
   - 운영: project-prod.war
4. 서버 VDI 접속 후 해당 tomcat/webapps에 업로드
5. 톰캣 재시작 후 배포 확인
```

### 5.2 배포 이력 관리

**수동 배포 기록 관리**:

- GitHub Desktop History 탭에서 배포된 커밋 태그 추가
- 배포 일시, 담당자, 변경사항을 커밋 메시지에 상세 기록
- 롤백 시 이전 커밋으로 체크아웃 후 재배포

## 6. 롤백 및 복구 전략

### 6.1 GitHub Desktop 롤백 방법[20]

**커밋 되돌리기**:

1. History 탭에서 되돌릴 커밋 우클릭
2. "Revert changes in this commit" 선택
3. 되돌리기 커밋이 자동 생성됨
4. Push 후 해당 상태로 war 파일 재배포

**브랜치 초기화**:

1. 문제가 생긴 브랜치에서 "Branch" → "Reset to this commit"
2. 안전한 커밋 지점 선택
3. 초기화 후 재개발 진행

## 7. 보안 및 안전 관리

### 7.1 접근 권한 관리

**GitHub 저장소 설정**:

- Private 저장소로 설정
- 개발자 2명만 Collaborator 권한 부여
- main 브랜치에 Branch protection rules 적용

### 7.2 백업 및 복구

**정기 백업 전략**:

- 매주 main 브랜치 Release 태그 생성
- 중요 변경사항마다 로컬 저장소 백업
- 서버 war 파일도 버전별로 보관

## 8. 현재 프로세스 대비 개선점

### 8.1 비효율성 해결

**기존 문제 → 개선 효과**:

- 소스 혼입 → 브랜치별 격리로 깔끔한 관리
- 수동 추리기 → 선택적 브랜치 병합으로 정확성 향상
- 충돌 해결 어려움 → GUI 도구로 직관적 처리
- 이력 추적 곤란 → 모든 변경사항 자동 기록

### 8.2 협업 효율성 증대

**팀 협업 개선**:

- 동시 개발 시 충돌 최소화
- 브랜치별 독립 작업으로 영향도 차단
- 변경사항 투명성 확보
- 롤백 및 복구 용이성

## 9. 점진적 도입 계획

### 9.1 1단계: 기본 브랜치 구조 적용 (1-2주)

1. 현재 develop 브랜치를 main으로 변경
2. 새로운 develop, test 브랜치 생성
3. feature 브랜치 개발 패턴 적용 시작
4. GitHub Desktop 기본 기능 숙달

### 9.2 2단계: 고급 기능 활용 (2-3주)

1. 충돌 해결 프로세스 정착
2. 선택적 배포 프로세스 확립
3. 롤백 및 복구 절차 마스터
4. 배포 이력 관리 체계화

### 9.3 3단계: 프로세스 최적화 (지속)

1. 팀만의 커밋 메시지 규칙 정립
2. 배포 자동화 가능성 검토
3. 성능 모니터링 및 개선
4. 새로운 Git 기능 학습 및 적용

## 10. 주의사항 및 팁

### 10.1 초보자 주의사항

**반드시 지켜야 할 원칙**:

- main 브랜치에 직접 개발 금지
- 충돌 발생 시 혼자 해결 시도 금지 (팀원과 상의)
- 중요한 변경사항은 반드시 백업 후 진행
- 모르는 기능은 테스트 저장소에서 먼저 실험

### 10.2 유용한 팁 [21] [11]

**GitHub Desktop 활용 팁**:

- History 탭에서 파일별 변경사항 세부 확인 가능
- Changes 탭에서 파일별 체크박스로 선택적 커밋 가능
- 브랜치 전환 시 작업 중인 변경사항 자동 보존
- Stash 기능으로 임시 작업 저장 후 브랜치 전환

이러한 체계적인 프로세스 개선을 통해 현재의 소스 관리 문제를 해결하고, 안전하고 효율적인 개발/배포 환경을 구축할 수 있습니다. GitHub Desktop의 직관적인 GUI를 활용하여 Git 초보자도 쉽게 협업할 수 있으며, 수동 배포의 위험성을 최소화하면서 체계적인 버전 관리가 가능합니다.

⁂


1. https://www.inflearn.com/community/questions/1193887/하나의-가상서버에-테스트-서버와-운영-서버를-관리하는-방법
2. https://www.sharedit.co.kr/freeboards/22519
3. https://hasumang.tistory.com/59
4. https://rootdev.tistory.com/22
5. https://jungmin5600.tistory.com/40
6. https://yoon-ssi.tistory.com/11
7. https://inpa.tistory.com/entry/GIT-⚡-github-flow-git-flow--브랜치-전략
8. https://devocean.sk.com/blog/techBoardDetail.do?ID=165571&boardType=techBlog
9. https://lee-ding.tistory.com/117
10. https://dev-hyonie.tistory.com/31
11. https://www.lainyzine.com/ko/article/github-desktop-installation-and-usage-guide/
12. https://doshisha.tistory.com/71
13. https://kakaobusiness.gitbook.io/main/tool/chatbot/main_notions/deploy
14. https://platypusss.tistory.com/entry/깃허브-브랜치-깃허브-깃배쉬-깃허브-데스크탑을-통한-협업-가이드
15. https://splendidlolli.tistory.com/748
16. https://velog.io/@gourd_erased/스테이징-서버와-운영-서버를-나눠-CICD-구축하기
17. https://flrjtwjrjt.tistory.com/entry/Github-협업-전략-Github-Desktop
18. https://ddongyeop.tistory.com/14
19. https://www.lifencoding.com/web/28
20. https://docs.github.com/ko/desktop/managing-commits/options-for-managing-commits-in-github-desktop
21. https://ossam5.tistory.com/611