GitHub Flow 가이드 문서

목차

GitHub Flow 소개

GitHub Flow는 간단하고 효과적인 브랜치 관리 전략으로, 지속적인 통합과 배포(CI/CD)를 지원합니다. 복잡한 브랜치 구조 대신 Main(또는 Master) 브랜치를 중심으로 기능별 브랜치를 생성하여 개발하는 방식입니다.

GitHub Flow의 핵심 원칙

1. **Main** 브랜치는 항상 배포 가능한 상태로 유지
2. 새로운 작업은 항상 기능 브랜치에서 진행
3. 정기적으로 원격 저장소에 푸시하여 작업 내용 공유
4. **Merge Request(MR)**를 통한 코드 리뷰 후 병합
5. **Main**에 병합되면 즉시 배포 가능

브랜치 관리 전략

GitHub Flow에서는 브랜치를 다음과 같이 관리합니다:

1. **Main** 브랜치: 항상 안정적이고 배포 가능한 코드만 포함
2. 기능 브랜치: Main에서 분기하여 새로운 기능, 버그 수정 등 작업 수행

브랜치 명명 규칙

가독성을 높이기 위해 다음과 같은 명명 규칙을 사용하는 것을 권장합니다 :

- `feature/login-page`: 새로운 기능 개발
- `bugfix/login-error`: 버그 수정
- `hotfix/security-issue`: 긴급 수정
- `docs/api-documentation`: 문서 작업
- `refactor/user-service`: 코드 리팩토링

실무 워크플로우

GitHub Flow의 실무 워크플로우는 다음과 같습니다 :

1. **Main** 브랜치에서 최신 코드 가져오기
2. 기능 브랜치 생성
3. 기능 개발 및 커밋
4. 원격 저장소에 푸시
5. **Merge Request(MR)** 생성
6. 코드 리뷰 및 논의
7. 테스트 통과 확인
8. **Main** 브랜치로 병합
9. 배포
10. 기능 브랜치 삭제

주요 Git 명령어

1. 저장소 복제

```
# 저장소 초기 복제
git clone https://gitlab.company.com/project/repository.git
cd repository
```

```
# 저장소 특정 브랜치만 복제 복제
git clone --single-branch --branch [branch-name] https://gitlab.company.com/project/repository.git [my-directory]
```

```
cd my-directory
```

2. 브랜치 관리

```
# 현재 브랜치 확인
git branch

# 원격 브랜치 포함한 모든 브랜치 확인
git branch -a

# Main 브랜치에서 최신 코드 가져오기
git checkout main
git pull

# 새 기능 브랜치 생성 및 전환
git checkout -b feature/user-authentication

# 기존 브랜치로 전환
git checkout feature/user-authentication
```

3. 변경사항 관리

```
# 변경된 파일 확인
git status

# 특정 파일 스테이징
git add src/components/LoginForm.js

# 모든 변경 사항 스테이징
git add .

# 변경 사항 커밋
git commit -m "로그인 폼 UI 구현 "

# 커밋 히스토리 확인
git log
git log --oneline --graph
```

4. 원격 저장소 연동

```
# 원격 저장소에 푸시
git push origin feature/user-authentication

# 원격 저장소에서 변경사항 가져오기
git pull origin feature/user-authentication

# 원격 저장소 확인
git remote -v
```

5. 변경사항 병합

```
# Main 브랜치 최신화
git checkout main
git pull

# 기능 브랜치 병합 (일반적으로 MR을 통해 수행)
git merge feature/user-authentication

# 브랜치 삭제 (로컬)
git branch -d feature/user-authentication

# 브랜치 삭제 (원격)
git push origin --delete feature/user-authentication
```

실제 개발 예시

다음은 로그인 기능을 개발하는 과정을 GitHub Flow로 진행하는 예시입니다 :

1. 개발 시작 준비

```
# 최신 코드 가져오기
git checkout main
git pull
```

```
# 기능 브랜치 생성
git checkout -b feature/login-implementation
```

2. 개발 및 커밋

```
# 로그인 컴포넌트 파일 생성 및 코드 작성
# ... 코드 작성 ...

# 변경 사항 확인
git status

# 변경 사항 스테이징 및 커밋
git add src/components/Login.js
git commit -m "로그인 컴포넌트 기본 구조 구현"

# 추가 수정 작업
# ... 코드 수정 ...

# 변경 사항 스테이징 및 커밋
git add src/api/authService.js
git commit -m "로그인 API 연동 기능 구현"

# 로그인 스타일 작업
# ... 스타일 적용 ...

git add src/styles/login.css
git commit -m "로그인 페이지 스타일링 완료"
```

3. 정기적인 원격 저장소 푸시

```
# 작업 내용을 원격 저장소에 푸시
git push origin feature/login-implementation
```

4. 테스트 및 리뷰 준비

```
# 최신 Main 코드를 가져와 충돌 확인 및 해결
git checkout main
git pull
git checkout feature/login-implementation
git merge main

# 충돌이 있다면 해결
# ... 충돌 해결 ...

git add .
git commit -m "Main 브랜치와 충돌 해결 "

# 최종 테스트 후 푸시
git push origin feature/login-implementation
```

5. MR 생성 및 병합 후 정리

```
# MR이 승인되고 병합된 후 , 로컬 브랜치 정리
git checkout main
git pull
git branch -d feature/login-implementation
```

GitLab에서의 MR 활용

GitLab에서 Merge Request(MR)는 코드 리뷰와 병합을 위한 핵심 기능입니다 .

MR 생성 방법

1. GitLab 웹 인터페이스에서 프로젝트로 이동
2. 왼쪽 메뉴에서 "Merge Requests" 선택
3. "New merge request" 버튼 클릭
4. 소스 브랜치 (기능 브랜치)와 목표 브랜치 (main) 선택
5. "Compare branches and continue" 클릭
6. MR 제목 , 설명 , 담당자 , 리뷰어 등 정보 작성
7. "Submit merge request" 클릭

효과적인 MR 작성 가이드

- 명확한 제목 : 무엇을 구현했는지 한눈에 알 수 있게 작성
- 상세한 설명 :
  - 구현 내용 요약
  - 테스트 방법
  - 관련 이슈 링크
  - 스크린샷 (UI 변경 시)
- 작은 크기로 유지 : 리뷰하기 쉽도록 MR은 작게 유지
- 라벨 활용 : 버그 수정 , 기능 추가 등 MR의 성격 표시

예시 MR 설명

```
## 로그인 기능 구현

### 구현 내용
- 로그인 페이지 UI 구현
- 인증 API 연동
- 로그인 상태 관리 기능 추가
- 로그인 실패 시 에러 처리

### 테스트 방법
1. `/login` 페이지 접속
2. 올바른 /잘못된 계정 정보로 로그인 시도
3. 로그인 후 메인페이지 리다이렉트 확인

### 관련 이슈
- Closes #42

### 스크린샷
[로그인 화면 스크린샷 ]
```

자주 발생하는 문제와 해결 방법

1. 병합 충돌 (Merge Conflict)

문제 : 같은 파일의 같은 부분을 여러 개발자가 수정하여 발생

해결 방법 :

```
# 충돌 파일 확인
git status

# 충돌 해결 (파일 편집 )
# <<<<<< HEAD (현재 브랜치 코드 )
# ======= (구분선 )
# >>>>>>> feature/branch (병합할 브랜치 코드 )

# 충돌 해결 후 커밋
git add .
git commit -m "충돌 해결 "
```

2. 실수로 Main에 직접 커밋한 경우

문제 : GitHub Flow를 위반하여 Main에 직접 변경사항을 커밋

해결 방법 :

```
# 커밋을 취소하지 않고 새 브랜치로 이동
git checkout -b feature/intended-changes
git push origin feature/intended-changes

# Main 브랜치 복구
git checkout main
git reset --hard origin/main
git pull
```

3. 커밋 메시지 수정

문제 : 마지막 커밋의 메시지를 수정해야 함

해결 방법 :

```
# 마지막 커밋 메시지 수정
git commit --amend -m "새로운 커밋 메시지 "
```

```
# 이미 원격에 푸시한 경우 (주의 : 협업 시 문제가 될 수 있음 )
git push origin feature/branch --force
```

효율적인 코드 리뷰 방법

리뷰어를 위한 팁

1. 작은 변경사항 단위로 리뷰 : 한 번에 모든 것을 보려고 하지 말고 파일별 , 기능별로 나누어 리뷰
2. 코드뿐만 아니라 설계도 검토 : 코드의 품질뿐만 아니라 설계의 적절성도 확인
3. 긍정적인 피드백도 제공 : 문제점뿐만 아니라 잘 작성된 부분도 언급
4. 명확한 이유와 개선 방향 제시 : "이렇게 바꾸세요 "가 아닌 "이런 이유로 이렇게 개선하면 좋을 것 같습니다 "

MR 작성자를 위한 팁

1. 자체 리뷰 먼저 : MR을 제출하기 전에 자신의 코드를 먼저 리뷰
2. 테스트 케이스 포함 : 코드가 예상대로 작동하는지 확인할 수 있는 테스트 추가
3. 적절한 크기로 분할 : 너무 큰 MR은 여러 개의 작은 MR로 나누기
4. 컨텍스트 제공 : 리뷰어가 코드를 이해하는 데 필요한 배경 정보 제공

이 가이드가 GitHub Flow를 이해하고 효과적으로 적용하는 데 도움이 되길 바랍니다 . 추가 질문이나 도움이 필요하면 팀 리드에게 문의하세요 .

Revision #2
Created 26 April 2025 04:49:41 by Admin
Updated 26 April 2025 04:54:13 by Admin