GIT을 기반으로 한 프로젝트 개발프로세스

김지헌, <ihoneymon@gmail.com> – Version 0.0.1, 08-12-2015

깃을 사용합시다. 깃을 쓰자. 깃을 쓰란 말야!!

- SVN은 변경이력이 많아질수록 속도가 느리지.
  - 커밋 및 처리속도가 빠르다. 변경이력이 많이 축적되어 있어도 속도저하가 거의 없다.
- 커밋찍기가 어렵다.

  변경사항 개발이 아직 완료되지 않았는데 이 변경사항을 중간에 커밋할 수가 없어. 커밋을 찍으면 SVN 중앙저장소에 반영되잖아.
  - 그런거 신경쓰지마. 맘껏 커밋을 찍어. 기능개발이 완료되면 그 때 푸시해.
  - 필요하다면 브랜치를 땋아서 거기서 개발하고 커밋찍고 개발내용을 반영하고 싶으면 병합merge하고 푸쉬해.
- 변경이력을 어떻게 봐야하지?
  - 커밋을 찍은 로그를 별도로 쉽게볼 수 있고 다양한 도구를 이용해서 확인할 수 있어.

깃git 이란?

> *Git is a **free and open source** (http://git-scm.com/about/free-and-open-source) distributed version control system designed to handle everything from small to very large projects with speed and efficiency.*

깃은 크고작은 프로젝트의 모든 것들을 빠르고 효과적으로 제어할 수 있도록 설계된 **무료 그리고 오픈소스 분산형 변경이력관리시스템** 이다.

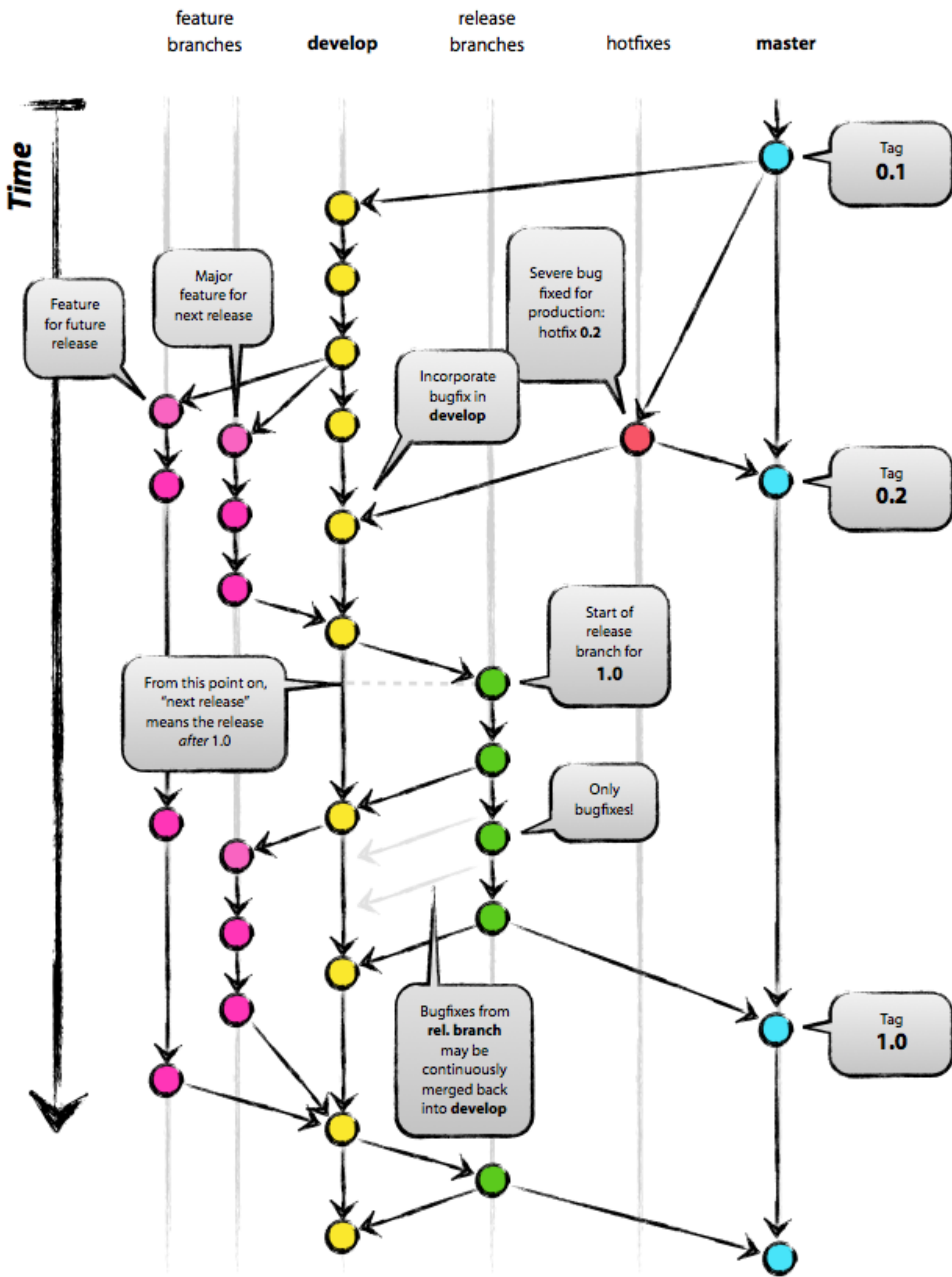
> *Git is **easy to learn** (http://git-scm.com/documentation) and has a tiny footprint with lightning fast performance (http://git-scm.com/about/small-and-fast). It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like cheap local branching (http://git-scm.com/about/branching-and-merging), convenient staging areas (http://git-scm.com/about/staging-area), and multiple workflows (http://git-scm.com/about/distributed).*

깃은 쉽게 배울 수 있으며 작은 크기에 비해 빛과 같은 빠른 속도를 자랑한다. 서브버전Subversion, CVS,

Perforce 그리고 ClearCase 등의 기존 SCM 도구들을 능가하는 손쉬운 로컬 브랜칭, 편리한 스태징영역, 그리고 다중 업무흐름 기능을 제공한다.

git: --fast-version-control (http://git-scm.com/) 페이지에서 소개하는 '깃git'에 관한 설명이다. 이전의 변경이력관리도구들에 비해 훨씬 쉽고 저렴한 '브랜칭' 관리기능을 제공하여 변경이력관리에 관한 막강함을 자랑하고 있는 이 도구를 사용해보지 않겠는가?

`git flow`을 기준으로 한 브랜치 전략



깃을 이용하면 '자연스레' '가지뻗기branch'하고 '커밋'하고 '병합merge'한다.

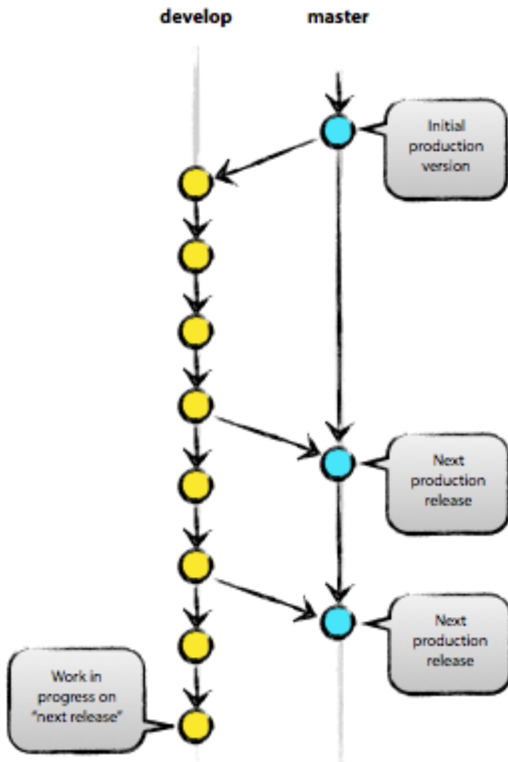
> *그 자유로운 브랜칭 사용방법에 '전략strategy'을 팀원들과 공유하여 만들고 있는 소프트웨어를 관리할 수 있다면 얼마나 좋을까?*

깃플로우git-flow 전략은 소프트웨어의 소스코드를 관리하고 출시하기 위한 '브랜칭 관리 전략branch management strategy'이다. 꽤 오래전에 나온 전략이며 이 전략의 단점을 극복하기 위해 github flow

(https://guides.github.com/introduction/flow/)와 gitlab flow (https://about.gitlab.com/2014/09/29/gitlab-flow/) 전략이 나왔다. 이 전략들을 그대로 적용할 수도 있지만 팀과 프로젝트 내에서 충분한 협의와 합의를 거친 후 사용하길 권한다.

우선은 기본전략이라할 수 있는 깃플로우git flow를 설명하고 깃플로우를 중심으로 하여 개발과정을 설명하여 숙달하도록 한다. 그 다음에 응용을 들어갈 수 있도록 해보자.

주요 브랜치



배포`master` 브랜치

깃 사용자라면 누구나 익숙한 기본 브랜치다. 먼저 배포했거나 배포준비(production-ready)된 코드는 `origin/master`에 두고 관리한다.

`master` 브랜치에 '병합' 한다는 것은 새버전을 배포한다는 것을 의미한다. `master` 브랜치에서 커밋될 때 git hook 스크립트를 걸어서 자동으로 빌드하여 운영서버로 배포하는 형식을 취한다.

개발`develop` 브랜치

다음에 배포하기 위해 개발하는 코드는 `origin/develop`에서 관리한다. 프로젝트를 진행하는 개발자들이 함께 보며 업무를 진행하는 브랜치이며 가장 역동적인 브랜치라고 할 수 있다. `develop` 브랜치의 코드가 안정화되고 배포할 준비가 되면 `master`로 '병합'하고 배포버전으로 태그를 단다.

보조브랜치

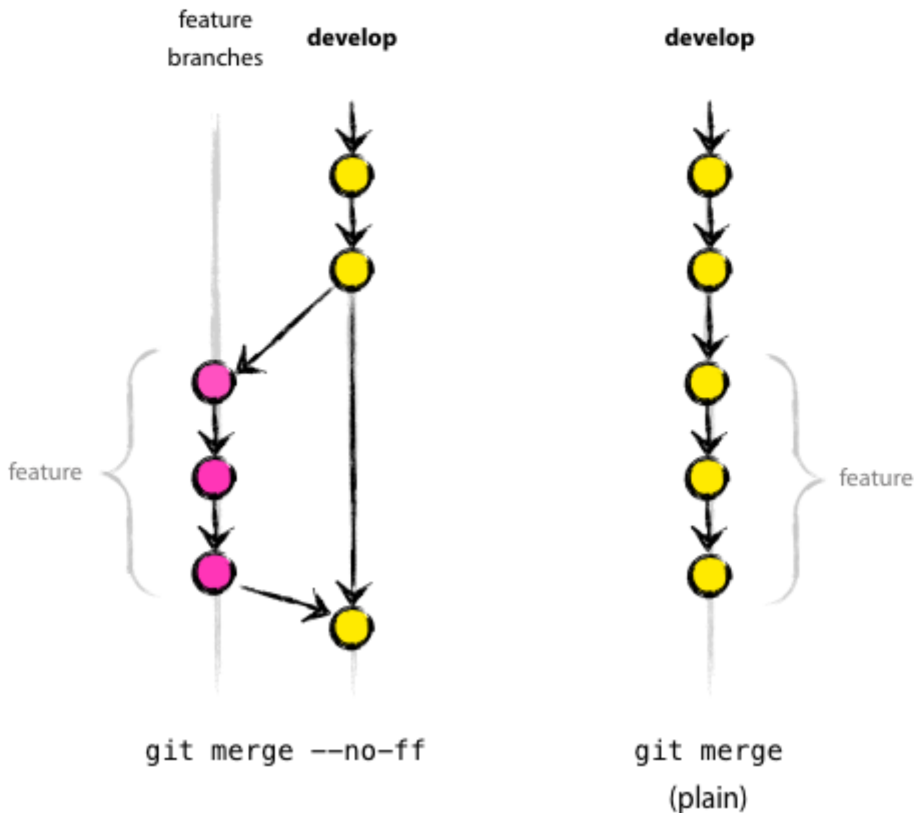
배포를 준비하고, 이미 배포한 제품이나 서비스의 버그를 빠르게 해결(hotfix) 해야한다. 이 모든 것을 동시에 진행하기 위해서 다양한 브랜치가 필요하다.

기능feature 브랜치

NOTE

기능 브랜치

- 시작브랜치: `develop`
- 병합대상 브랜치: `develop`
- 브랜치이름 규칙: `master`, `develop`, `release-`, `hotfix-` 를 제외한 것



기능`feature` 브랜치는 배포하려고 하는 **기능을 개발하는 브랜치**다. 기능을 개발하기 시작할 때는 언제 배포할 수 있을지 알 수 없다. 프로젝트를 진행하면서 애자일 방법론을 도입했다면, 스프린트 기간 동안에 개발해야할 기능이라면 스프린트 기간동안 개발해야할 브랜치를 말한다. 기능`feature` 브랜치는 그 기능을 다 완성할 때까지 유지하고 있다가 다 완성되면 개발`develop` 브랜치로 병합한다. 개발된 결과가 실망스럽거나 필요없을 때는 삭제하면 된다. 삭제하는 것에 미련을 가질 필요는 없다.

- git-flow 이용시 `feature/{branch-name}` 형식
- 이슈추적을 사용한다면 `feature/{issue-number}-{feature-name}` 형식
  - Ex) feature/1-init-project, feature/2-build-gradle-script-write

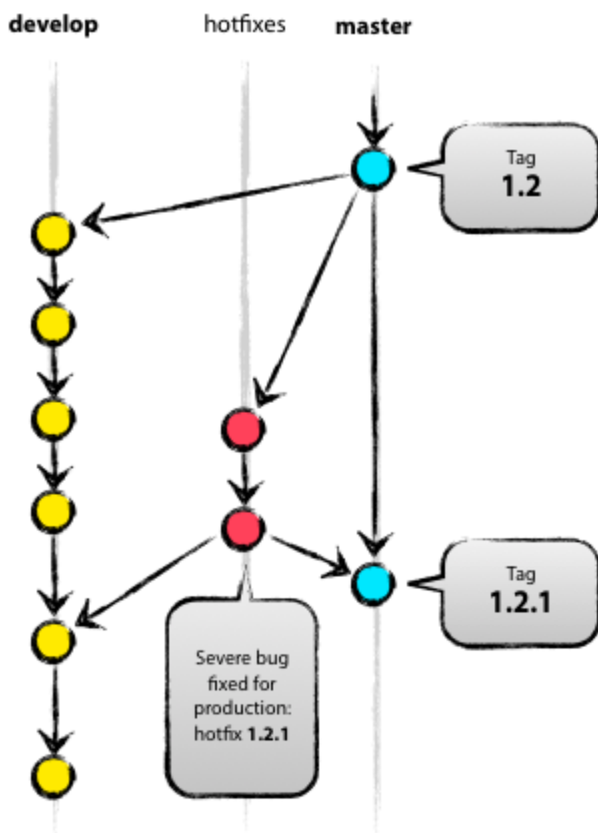
출시release 브랜치

NOTE	*출시 브랜치* • 시작브랜치: `develop` • 병합대상 브랜치: `develop`, `master` • 브랜치이름 규칙: `release-*`

출시`release` 브랜치는 실제 배포할 상태가 된 경우에 생성하는 브랜치다. 마스터`master` 브랜치를 통해 배포하기로 했으므로 먼저 출시`release`를 마스터`master` 브랜치로 병합한다. 나중에 이 배포버전을 찾기 쉽도록 태그를 만들어 현재 병합되는 커밋을 가리키도록 한다. 이 때 배포된 기능에 반영될 수 있도록 개발`develop` 브랜치에도 함께 병합한다.

긴급수정hotfix 브랜치

NOTE	*긴급수정 브랜치* • 시작브랜치: `master` • 병합대상 브랜치: `develop`, `master` • 브랜치이름 규칙: `hotfix-*`



미리 계획되지 않은 배포를 위한 브랜치다. 기본적인 동작방식은 출시`release`와 비슷하다. 이미 배포한 운영버전에서 발생한 문제를 해결하기 위해 만든다. 운영 버전에 생긴 치명적인 버그는 즉시 해결해야하기 때문에 문제가 생기면 마스터`master` 브랜치에 만들어둔 태그`tag`로부터 긴급수정`hotfix` 브랜치를

생성한다.

개발과정을 기준으로 한 깃 사용방법 설명

윈도우와 맥에서 사용할 수 있는 깃클라이언트로 많이 사용되는 소스트리sourceTree (https://www.sourcetreeapp.com/)를 기준으로 진행한다.

사전 확인사항

1. 깃설정에서 '**사용자명**'과 '**이메일**' 정보가 입력되어 있는지 확인한다.
2. 깃 저장소에 `master` 브랜치가 존재해야 한다.

git flow 시작하기

1. 저장소 선택 후 [깃 플로우] 버튼 클릭

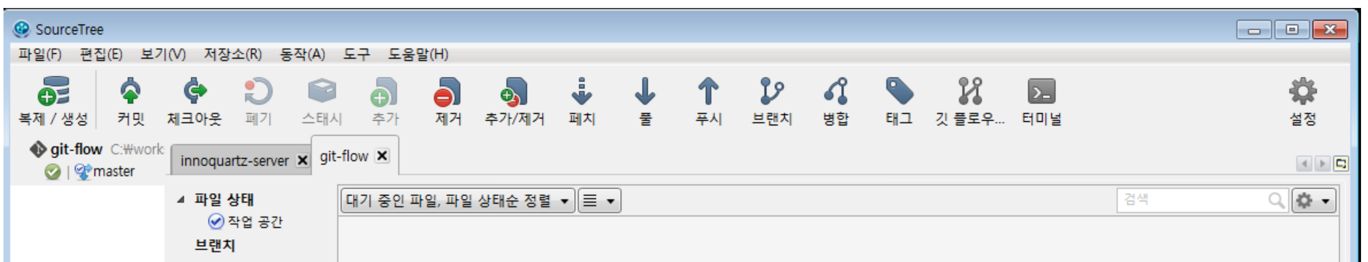
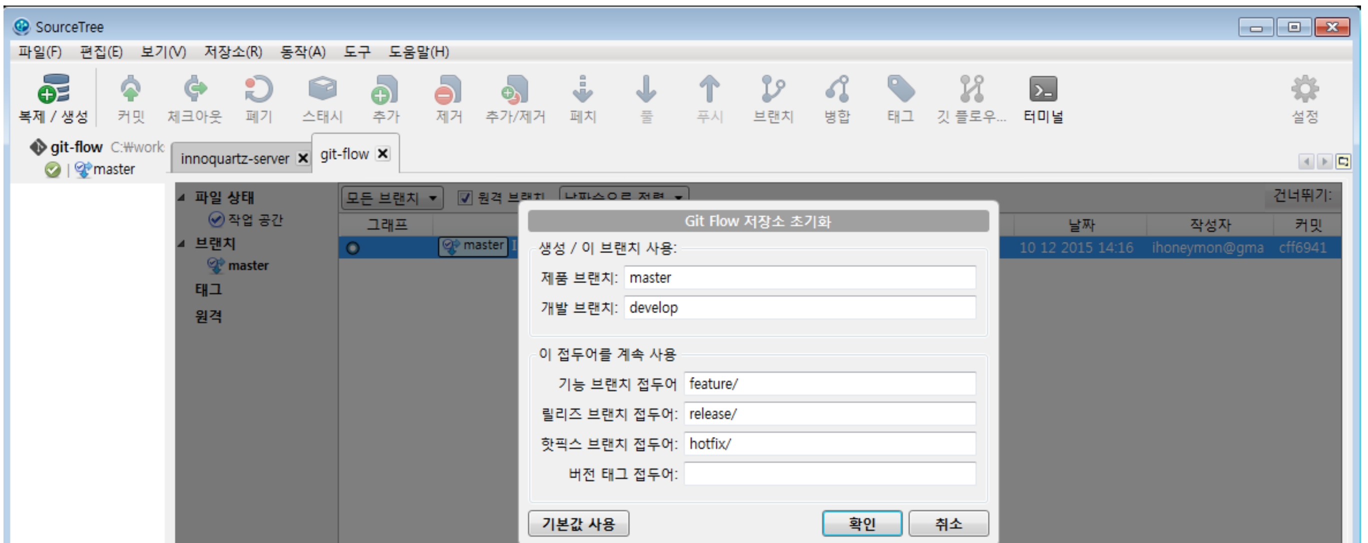
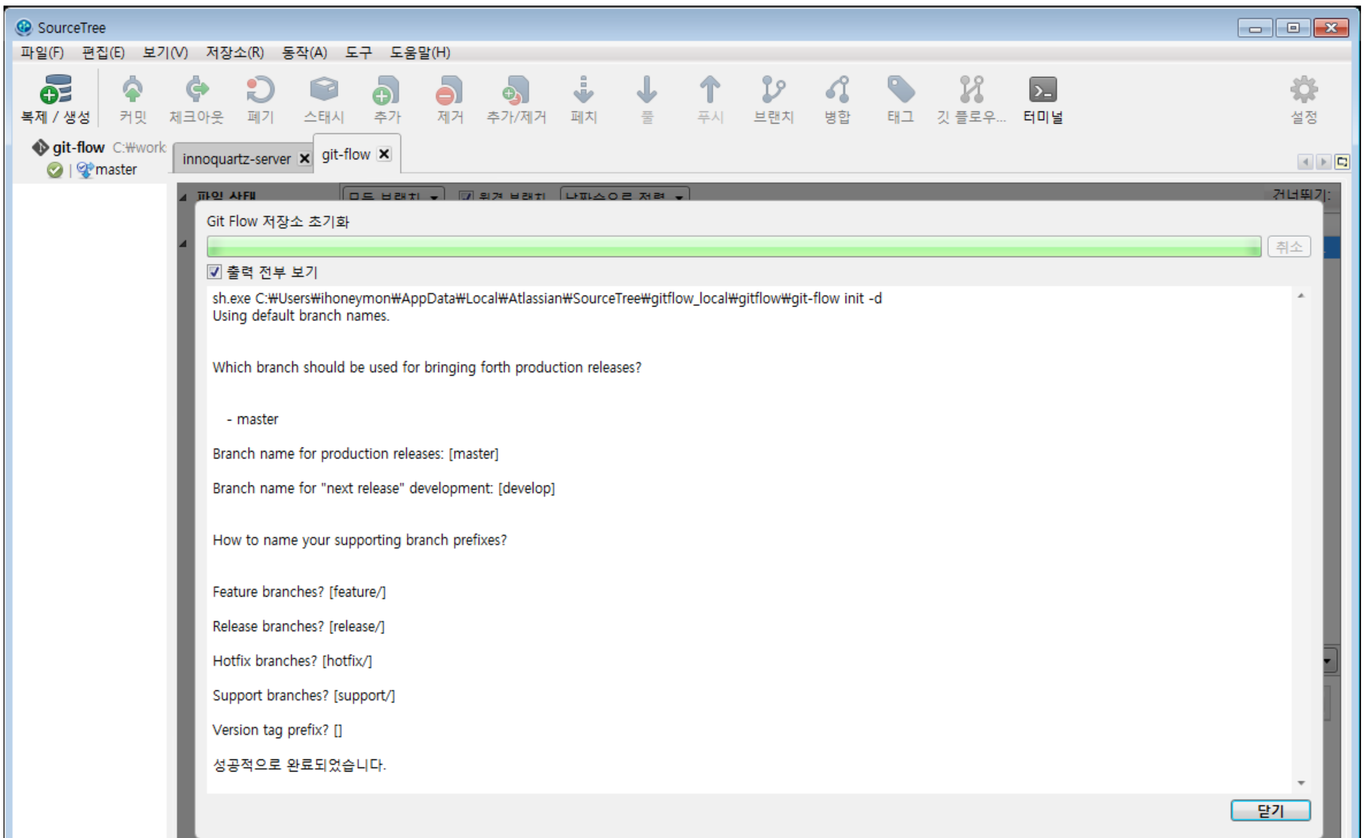


Figure 1. [깃 플로우] 버튼 클릭

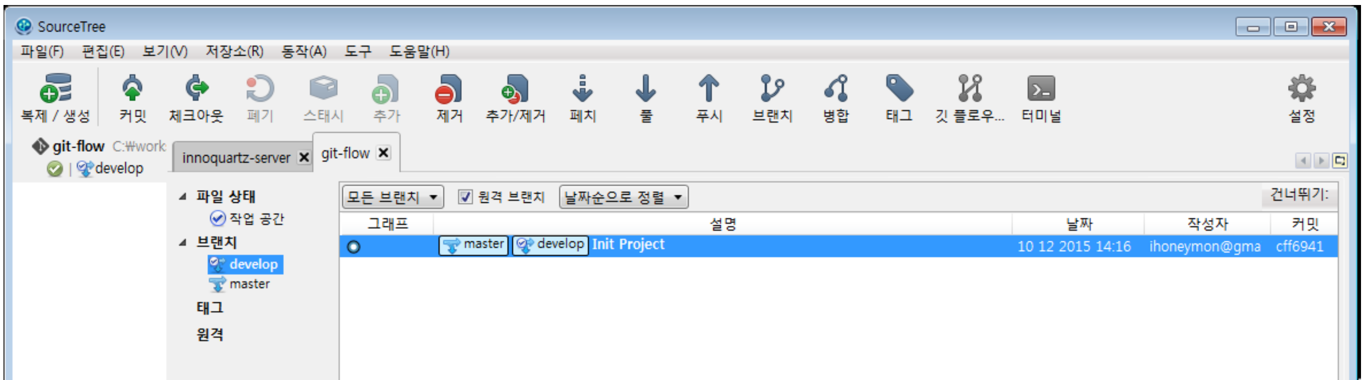
2. 깃플로우 목적별 브랜치 초기설정



3. 깃플로우 생성



4. 깃플로우 초기화 완료



깃플로우 초기화가 완료되면 `develop` 브랜치가 생성되어 체크아웃 상태가 되어 있다.

기능개발

다음과 같은 '74. 빌드'와 관련된 기능을 개발해야 한다고 가정해보자.

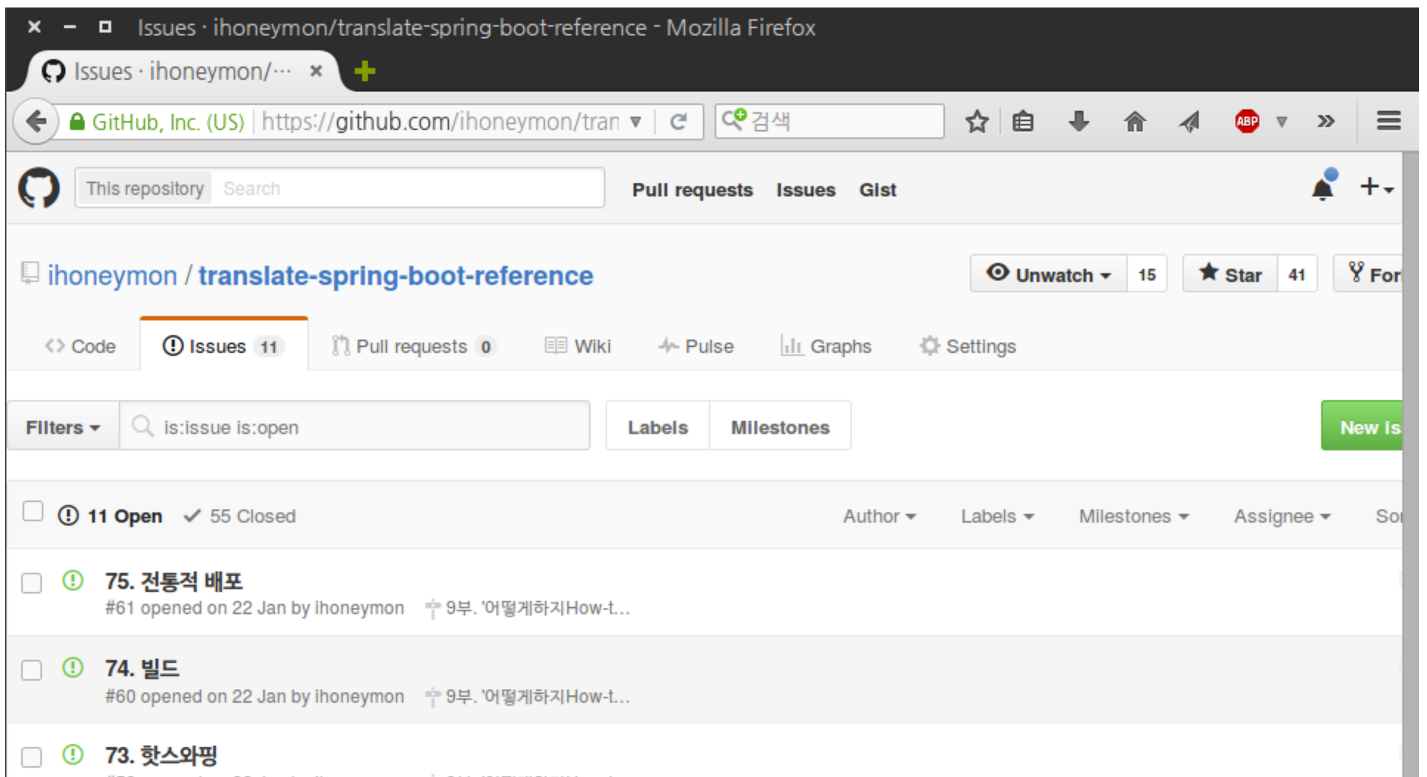


Figure 2. '74. 빌드'

기능feature 브랜치 생성

1. 깃플로우버튼 클릭
2. [새 기능 시작] 버튼을 클릭한다.

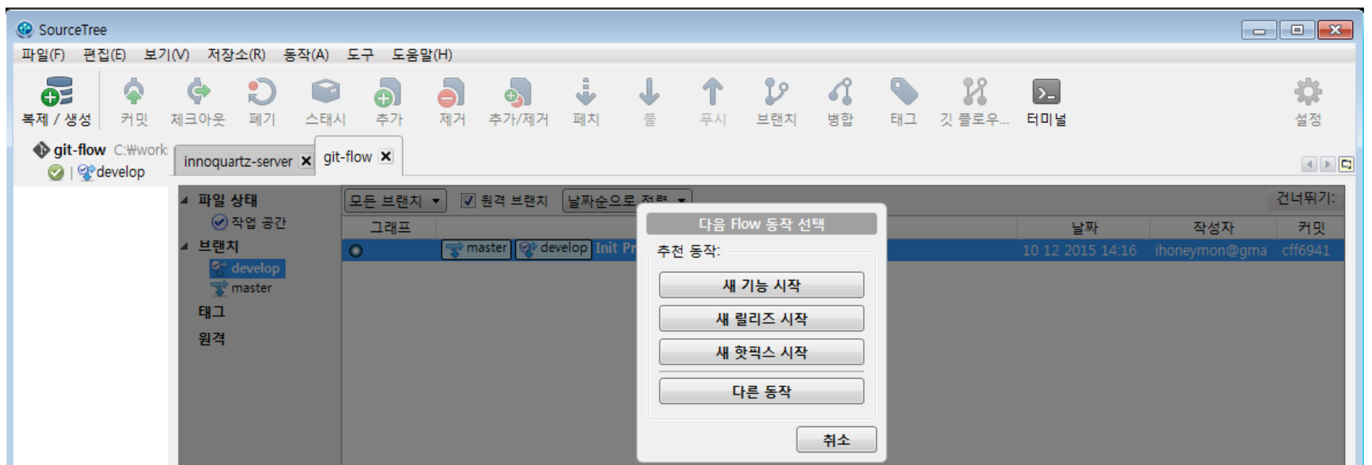


Figure 3. [새 기능 시작] 버튼 클릭

3. 기능명 입력후 [확인] 클릭

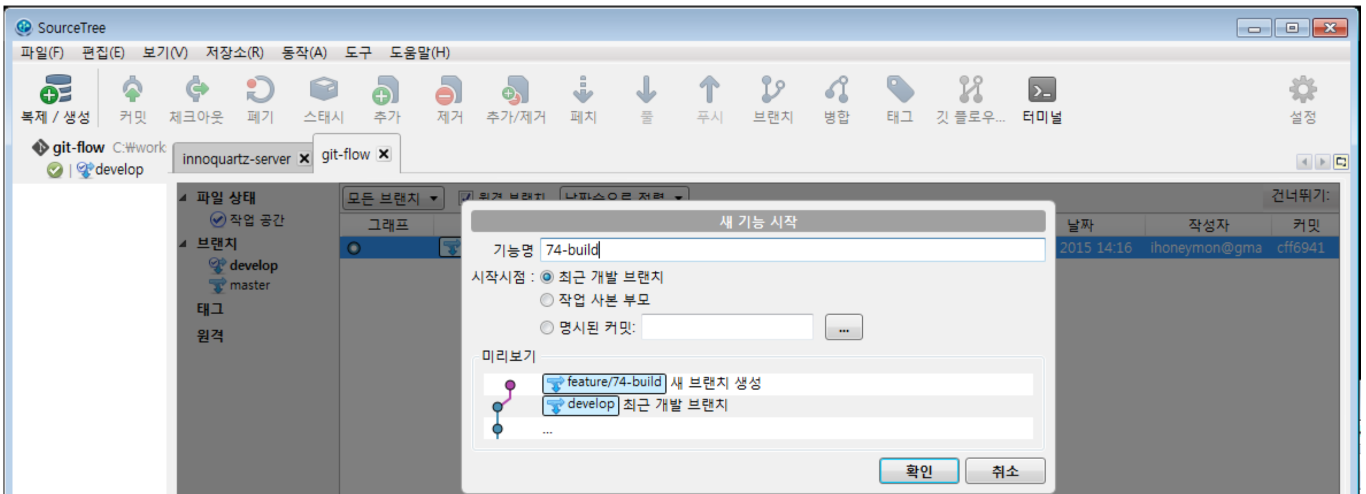


Figure 4. feature branch name: 74-build

a. 기능브랜치의 이름은 `[issue number]-[기능명 혹은 기능 서술]` 형태면 좋다.

4. 기능브랜치가 생성된 것 확인

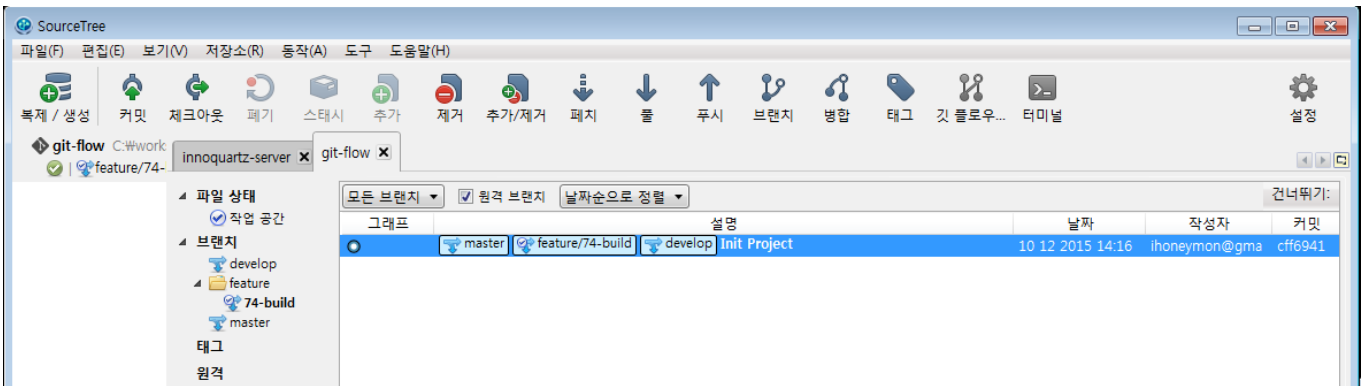


Figure 5. feature branch complete

기능feature 브랜치 내에서 커밋

커밋메시지에는 현재 개발하고 있는 기능과 관련된 이슈번호를 함께 넣어주는 것이 좋다. 커밋메시지에서 `# 메시지`의 경우에는 주석으로 인식하지만 `#이슈번호` 인 경우에는 깃 저장소 개발플랫폼에 따라 이슈와 연결시켜준다.

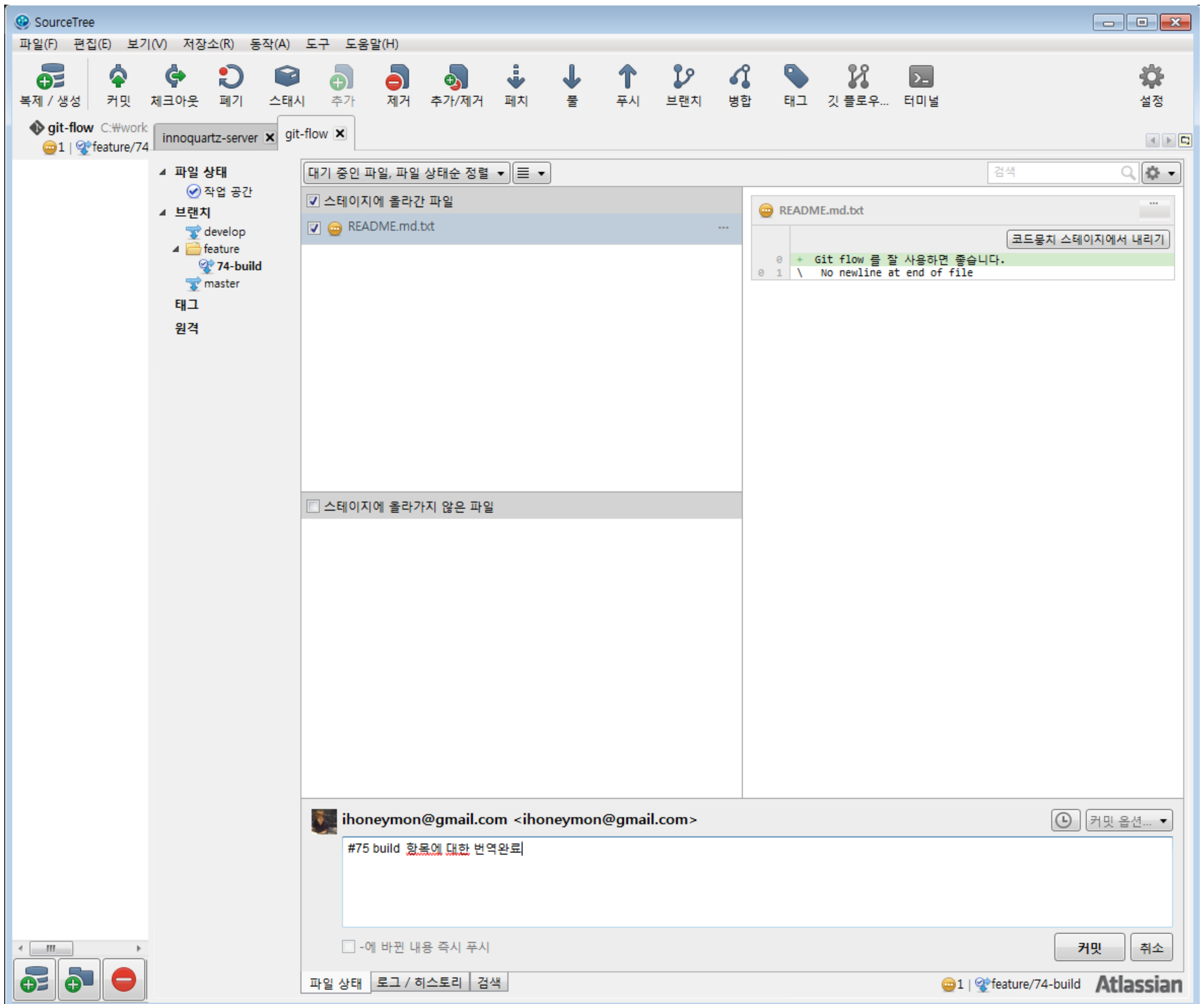


Figure 6. commit message on feature branch

> *깃메시지는 메시지만으로 해당 커밋에서 수행한 항목들을 이해할 수 있도록 설명하는 것이 좋고, 기능개발과정 중간중간 테스트를 통과시킨 후 커밋을 찍도록 한다.*

기능feature 브랜치 완료

기능 개발이 완료되었다면,

1. [깃 플로우 버튼] 클릭
2. [기능 마무리] 클릭

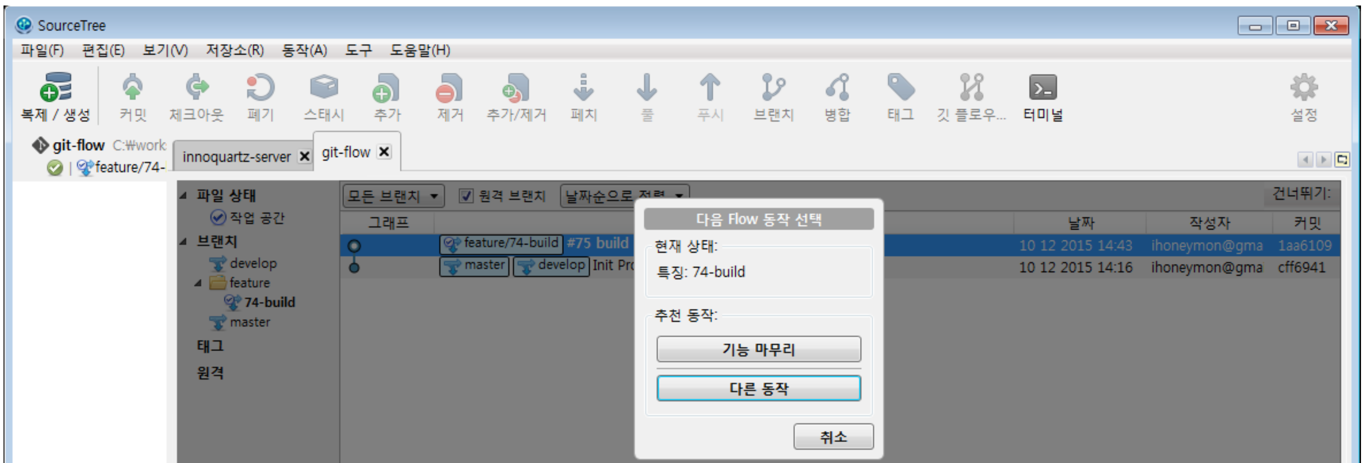


Figure 7. [기능 마무리] 버튼 클릭

3. '기능 마무리' 항목에서 [확인] 버튼 클릭

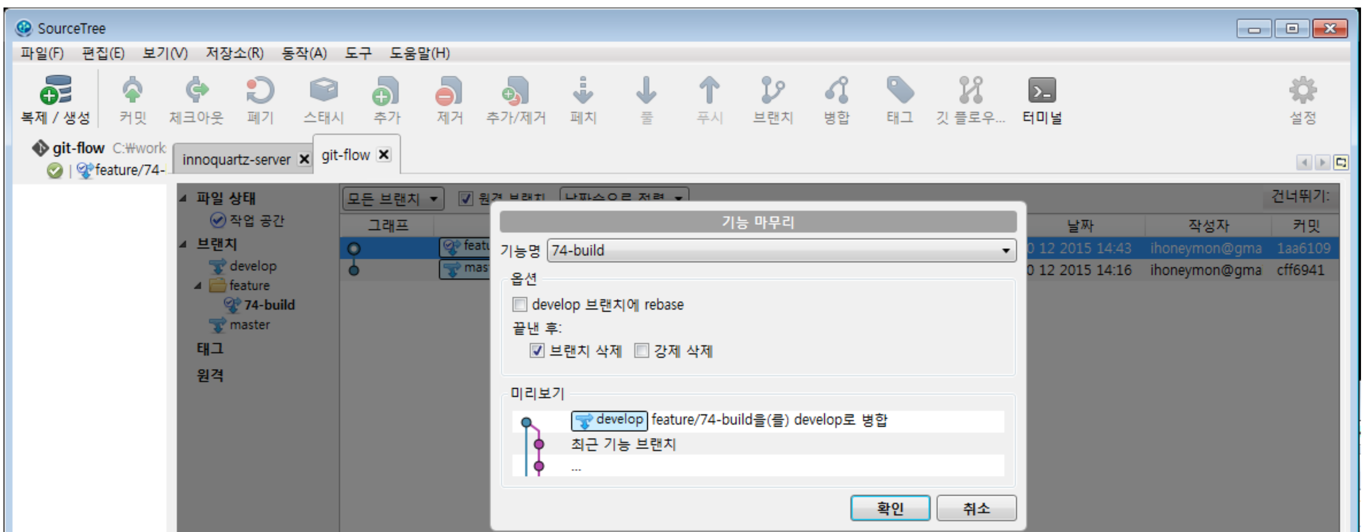


Figure 8. [확인] 클릭

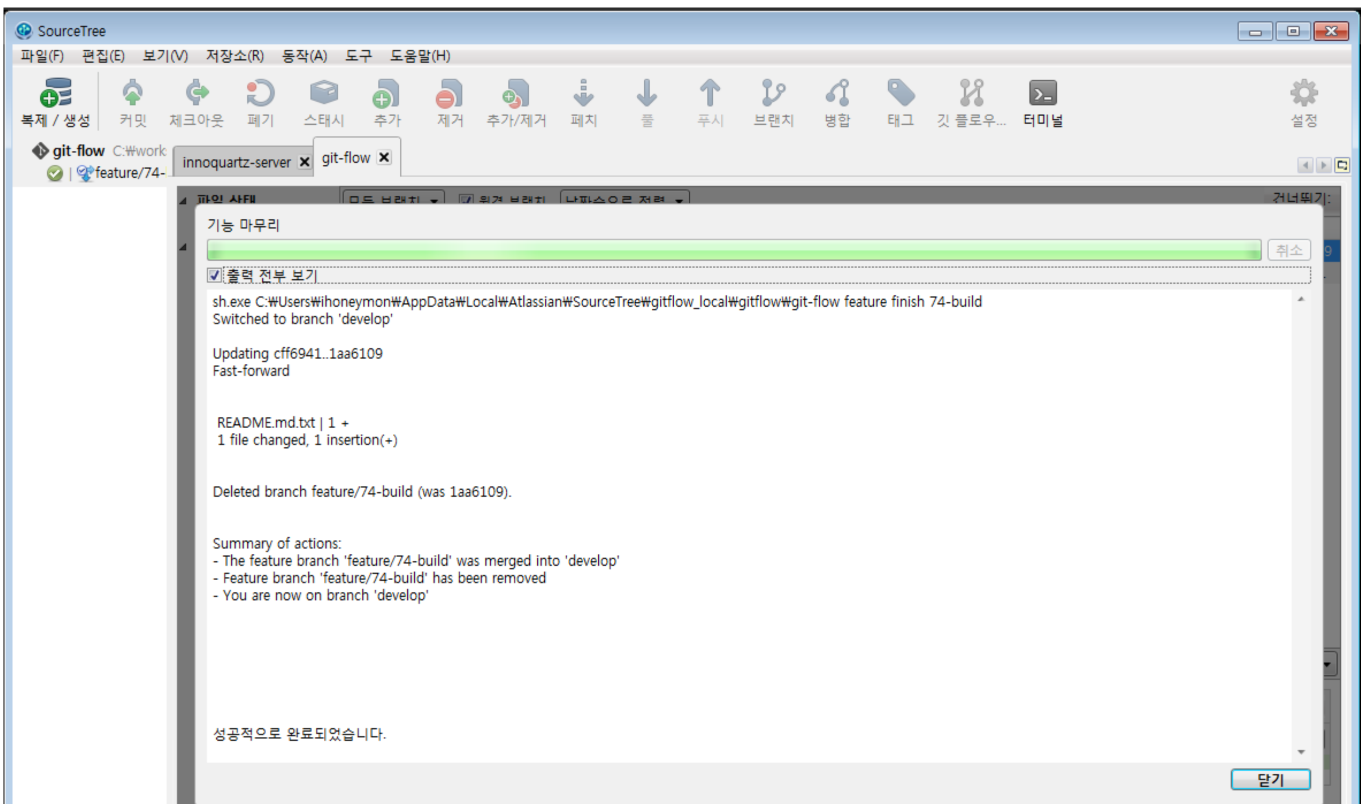


Figure 9. feature 브랜치 완료화면

a. 'develop 브랜치에 rebase'는 클릭하지 않는다.

i. 개발자가 기능브랜치를 생성하여 작업한 이력을 남길 수 있어 추후 변경이력을 추적하는데 용이하다.

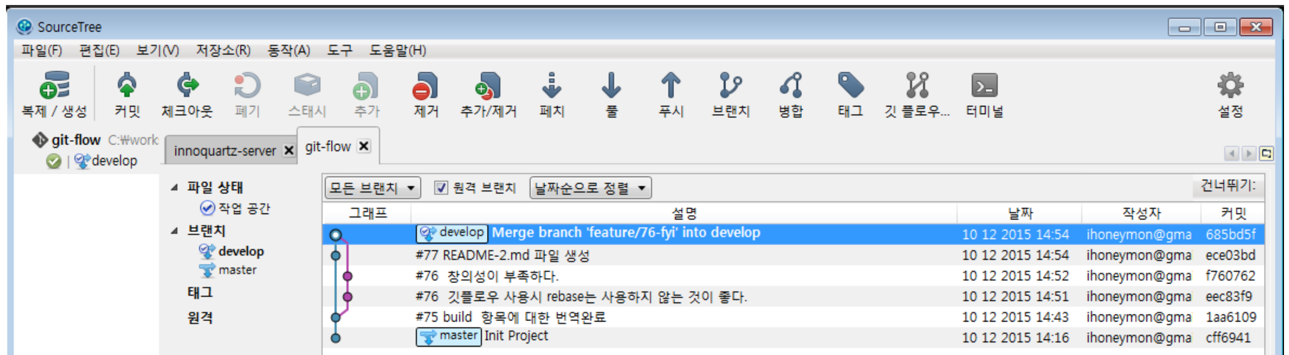


Figure 10. rebase를 사용하지 않은 브랜치모습

출시

애플리케이션을 배포할 때가 되었다.

출시release 브랜치 생성

1. [깃플로우] 버튼 클릭

2. [새 릴리즈 시작] 버튼 클릭

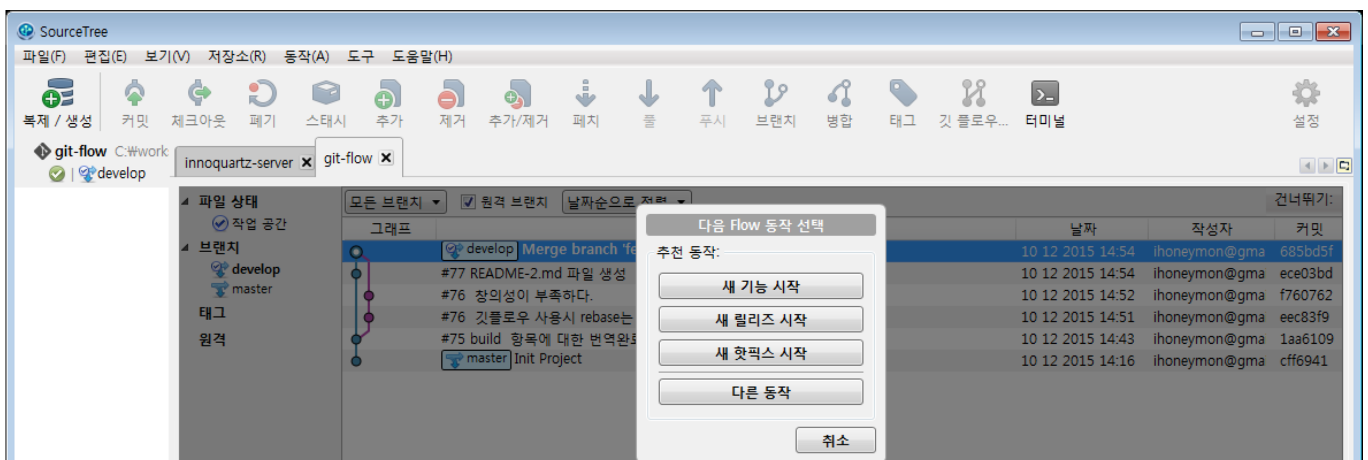


Figure 11. [새 릴리즈 시작] 버튼 클릭

3. 출시버전 입력 후 [확인] 버튼 클릭

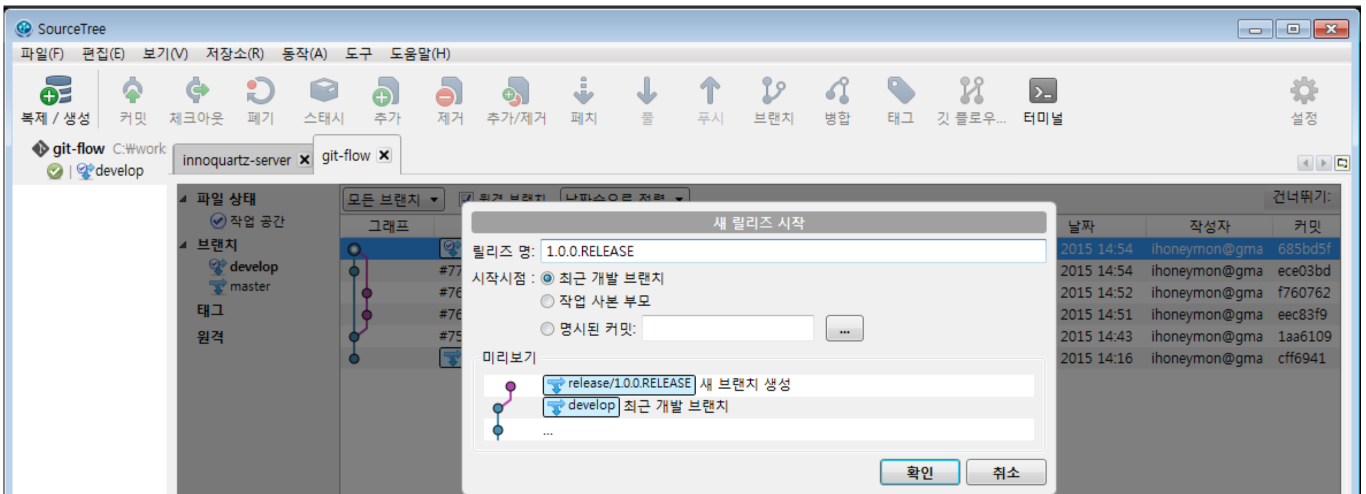


Figure 12. 출시버전 입력

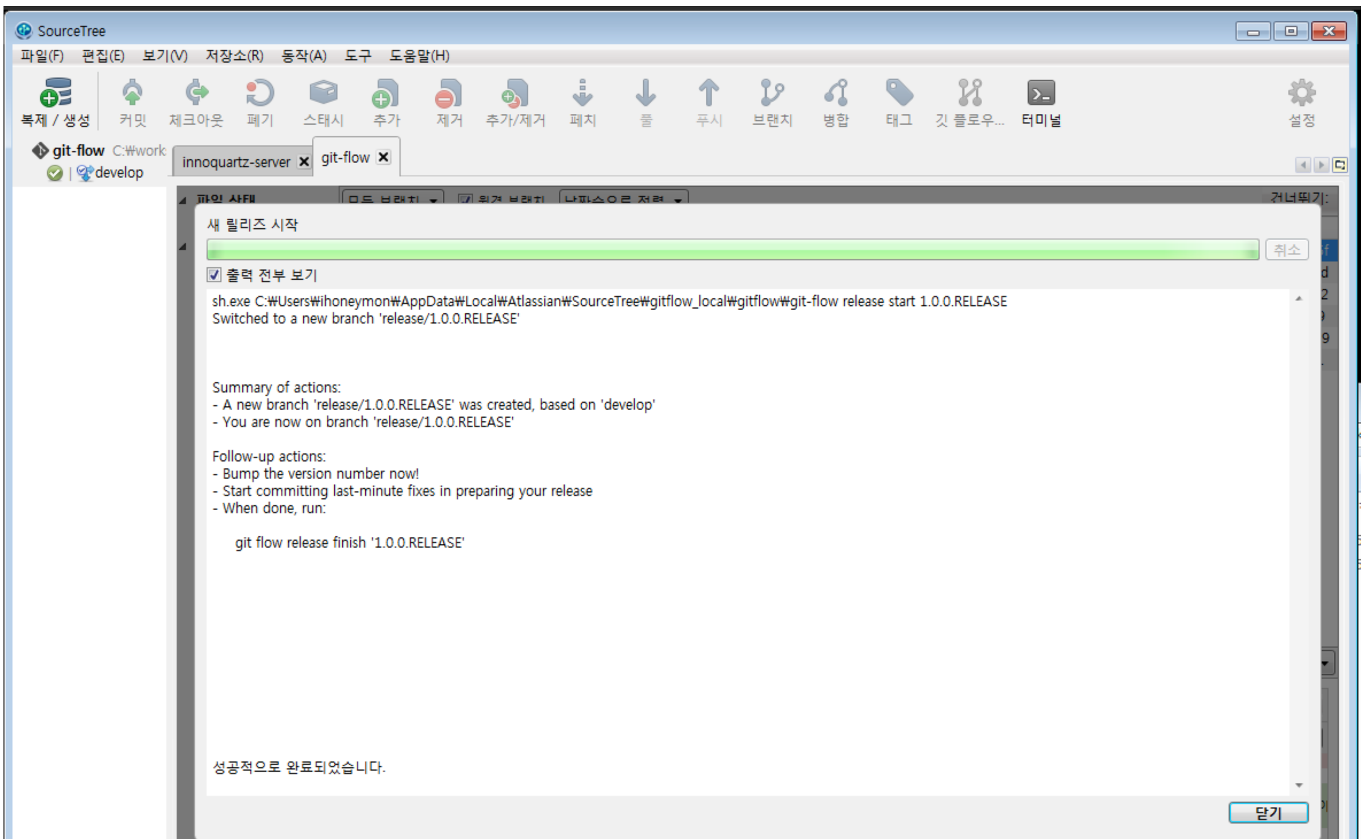


Figure 13. 출시브랜치 생성명령 실행화면

4. 출시버전 브랜치 생성확인

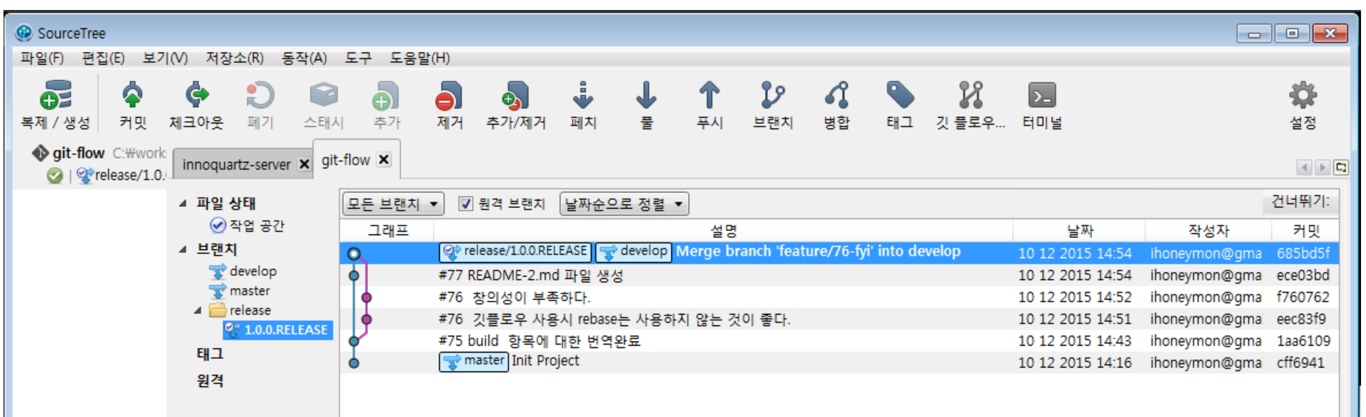


Figure 14. 출시버전 브랜치 생성확인

출시에 필요한 기능들 합치기

출시 브랜치를 생성한 후에 기능을 추가적으로 개발한 경우들이 발생할 수 있다. 그럴 경우에는 기능개발을 완료하고 `develop` 브랜치에 병합한 후에 `develop` 브랜치를 출시 브랜치로 병합하면 되겠다.

출시release 브랜치 완료

1. [깃 플로우] 버튼 클릭

2. [릴리즈 마무리] 버튼 클릭

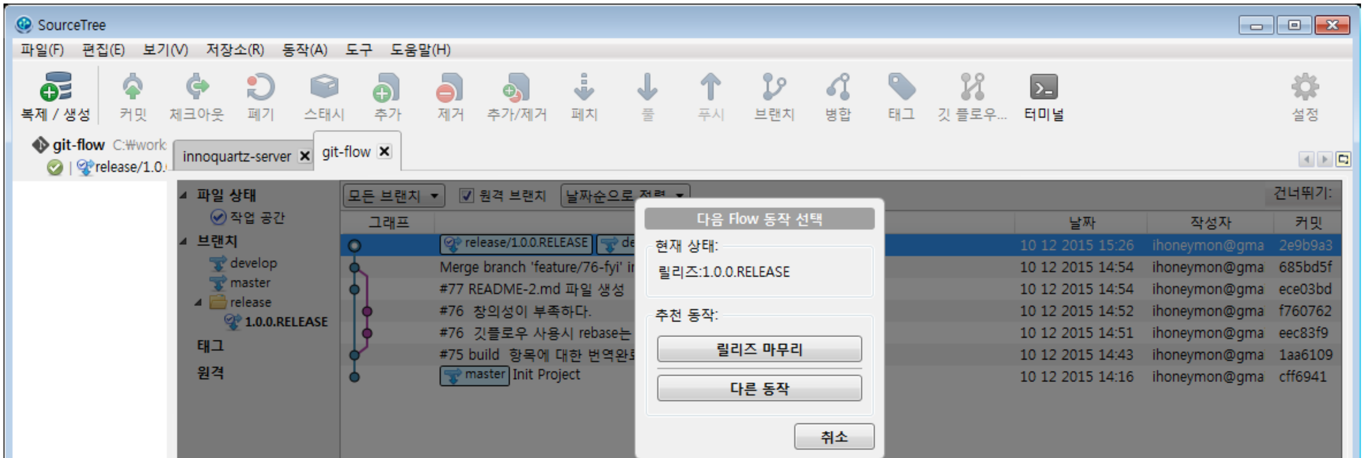


Figure 15. [릴리즈 마무리] 버튼 클릭

3. '릴리즈 마무리' [확인] 클릭

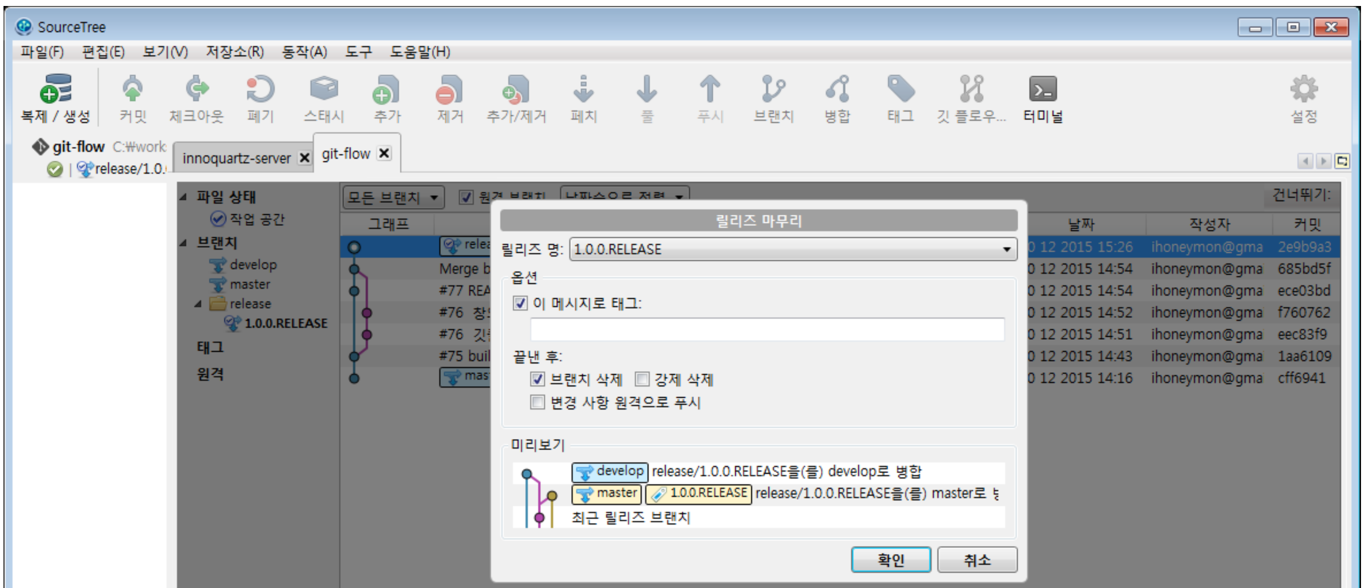


Figure 16. '릴리즈 마무리' [확인] 클릭

긴급수정hotfix 브랜치

출시이후 긴급수정해야할 내용이 발생하는 경우 출시버전에 맞춰서 핫픽스를 생성한다.

긴급수정hotfix 브랜치 생성

1. [깃 플로우] 버튼 클릭

2. [새 핫픽스 시작] 버튼 클릭

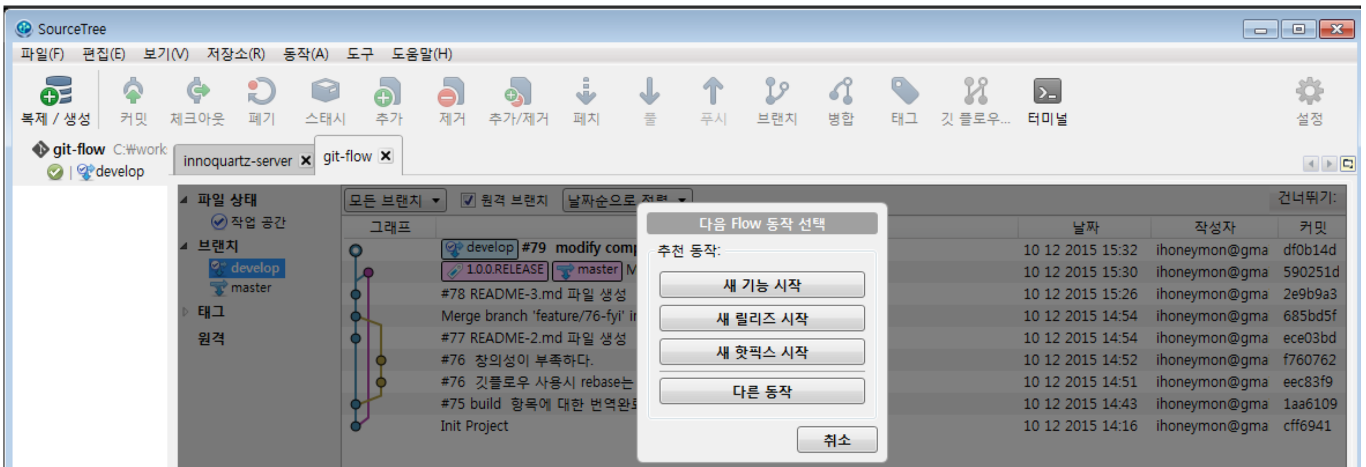


Figure 17. [새 핫픽스 시작] 버튼 클릭

3. 핫픽스이름 입력 후 [확인]

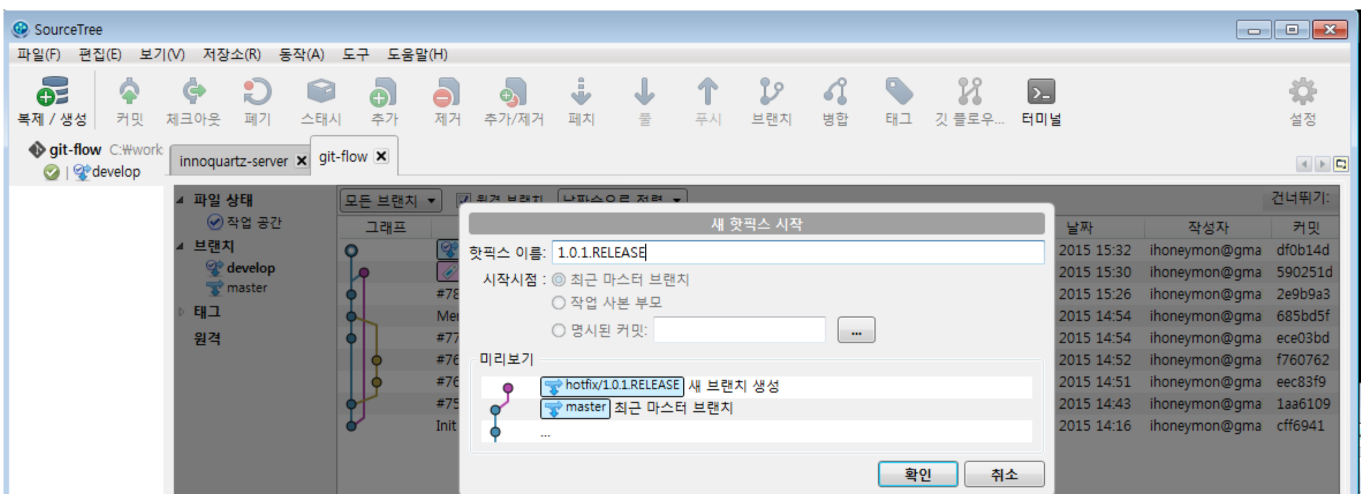


Figure 18. 핫픽스이름 입력 후 [확인]

긴급수정hotfix 작업진행

긴급수정hotfix 브랜치 완료

1. [깃 플로우] 버튼 클릭

2. [핫픽스 마무리] 버튼 클릭

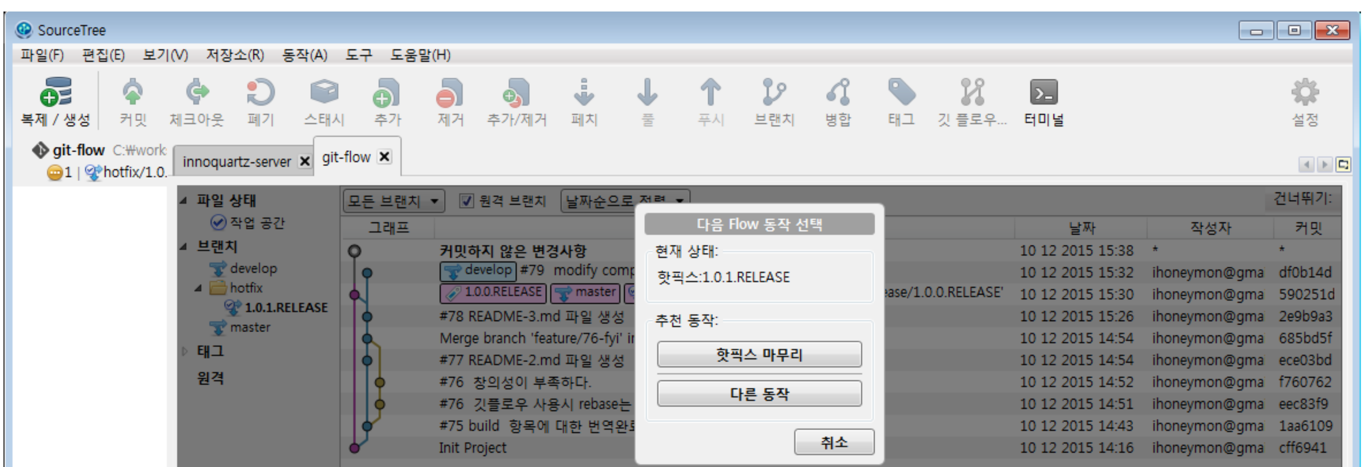


Figure 19. [핫픽스 마무리] 버튼 클릭

3. '핫픽스 마무리' [확인] 버튼 클릭

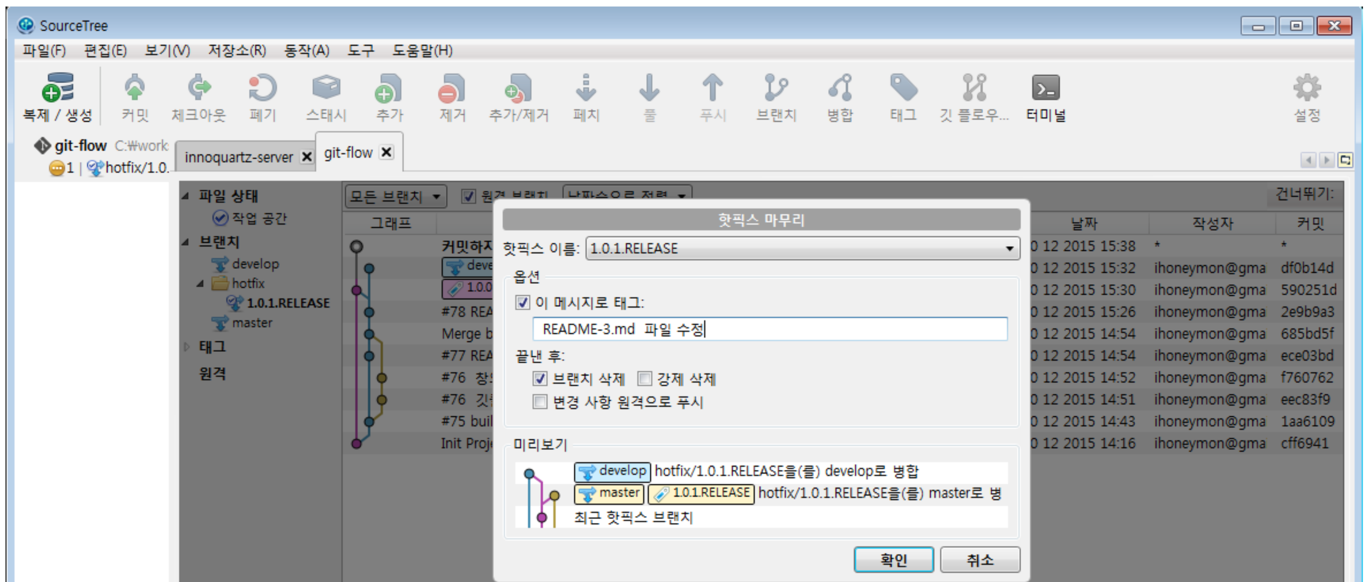


Figure 20. '핫픽스 마무리' [확인] 버튼 클릭

4. 긴급수정 브랜치 처리 완료

```
핫픽스 마무리

출력 전부 보기

sh.exe C:₩Users₩ihoneymon₩AppData₩Local₩Atlassian₩SourceTree₩gitflow_local₩gitflow₩git-flow hotfix finish -f c:₩temp₩5paesl0a.2io 1.0.1.RELEASE
Switched to branch 'master'

Merge made by the 'recursive' strategy.

 README-3.md.txt | 1 +
 1 file changed, 1 insertion(+)


Switched to branch 'develop'

Merge made by the 'recursive' strategy.

 README-3.md.txt | 1 +
 1 file changed, 1 insertion(+)


Deleted branch hotfix/1.0.1.RELEASE (was 8d98cdf).


Summary of actions:
- Latest objects have been fetched from 'origin'
- Hotfix branch has been merged into 'master'
- The hotfix was tagged '1.0.1.RELEASE'
- Hotfix branch has been back-merged into 'develop'
- Hotfix branch 'hotfix/1.0.1.RELEASE' has been deleted


성공적으로 완료되었습니다.

닫기
```

Figure 21. 긴급수정 브랜치 처리 완료 실행기록

Figure 22. 긴급수정 브랜치 처리 완료후

프로젝트 참여자들 강요사항

- 개발, 변경사항에 대한 테스트 통과 후 커밋을 찍고 밀어넣는다.

로컬-개발서버-테스트서버-운영서버 전략

- 로컬, 개발서버는 개발`develop`브랜치를 기준으로 빌드-배포한다.
- 테스트-운영서버는 마스터`master` 브랜치를 기준으로 빌드-배포한다.

브랜치 이용 관리전략

- 프로젝트에 참여중인 개발자들은 개발`develop` 브랜치를 기준으로 업무를 진행한다.
- 마스터`master` 브랜치는 아키텍트, 프로젝트리더 혹은 선임개발자가 관리한다.
- 사전에 협의되지 않은 내용은 마스터`master` 브랜치에 반영하지 않는다.

정리

깃플로우`git flow`는 깃에서 제공하는 강력한 브랜칭 기능을 활용한 변경이력 관리 전략이다. 이 전략은 상황에 따라서 다양한 변화가 가능하다. 깃플로우는 다양한 변형된 형태의 전략을 세울수 있으며 깃헙 github, 깃랩gitlab 에서 제공하는 방식이 있다.

> *중요한 것은 팀원, 프로젝트 구성원이 깃플로우 전략에 대해서 숙지하고 어떻게 사용할지를 결정하여 함께 공유하고 있어야 한다는 것이다.*

> *깃이 제공하는 손쉬운 브랜치생성 기능을 활용하자.*

참고문헌

- A successful git branching model (http://dogfeet.github.io/articles/2011/a-successful-git-branching-model.html)
- github flow

  Release가 분명하지 않은 경우엔 확실히 git-flow를 적용하기 어렵다. - 이것이 깃플로우가 가지고 있는 단점!

  - Introduction github flow (https://guides.github.com/introduction/flow/)
  - github flow (https://dogfeet.github.io/articles/2011/github-flow.html)
- gitlab flow
  - gitlab flow (https://about.gitlab.com/2014/09/29/gitlab-flow/)

Version 0.0.1
Last updated 2015-12-11 17:31:03 KST